Joint Center for Satellite Data Assimilation

CRTM: v2.4.0 User Guide

October 28, 2020; rev 1.0

Change History

Date	Author	Change
x 2012-06-11	P.van Delst	Draft release.
2012-06-14	P.van Delst	Draft #2 and #3 release.
2012-06-27	P.van Delst	Draft #4 and #5 release.
2012-07-11	P.van Delst	Initial release.
2012-11-16	P.van Delst	Updated for v2.1.1.
2013-02-13	P.van Delst	Updated for v2.1.2.
2013-06-20	P.van Delst	Updated for v2.1.3
2020-10-22	P. Stegmann	Initial Updates for v2.4.0
2020-10-27	C. Dang	Updates for v2.4.0

Contents

List of Figures

List of Tables

What's New in v2.4 - released October 16, 2020

New Science

Experimental Cloud Coefficient tables (see `fix/CloudCoeff`)

- Expanded effective radius (10 mm) table to support largest particle sizes in binary and netcdf4 formats.
- Penn State University contributed tables for closer alignment with common cloud microphysical packages: WSM-6, Thompson, GFDL. (netCDF4 format only, not included in v2.4.0-alpha)

CMAQ Aerosols Updated: CMAQ-based (v4.x) Aerosols and Radiance/AOD simulation capability. See `check_crtm.F90` for how to switch between GOCART (default) and CMAQ.

CrIS Non-LTE Finalize CrIS Non-LTE correction validation (not available in alpha release)

Updated sensor coefficient files
1. Earth Observing Nanosatellite-Microwave: eon_mw.v1
2. Sentinel-3A Sea and Land Surface Temperature Radiometer: slstr_sentinel3a
3. Meteosat-11 SEVIRI: seviri_m11
4. New coefficient for ABI_G17, and updated IDs from ABI_GR to ABI_G16
5. New coefficients for Metop-C sensors: AVHRR3_Metop-C, IASI(b1,b2,b3)_Metop-C, IASI300_Metop-C, IASI316_Metop-C, IASI616_Metop-C
6. L-Band sensors at 1.413 GHz: SMAP and SMOS (V, H, 3rd, 4th Stokes)
7. Tempest-D_cubesat: 5 microwave bands at 87, 164, 173, 178, and 181 GHz
8. Updated for a shifted WV band SRF of MI-L_COMS.v2
9. Antenna-pattern corrected AMSUA-Metop-C transmittance coefficients.
10. EON-MW, GOES 17 ABI update for 81K fix, FY4-GIIRS
11. JPSS-2 VIIRS, GEOKOMPSAT-2A AMI, Metop-SG-A1 MWS (not available in alpha release)

Software Improvements

OpenMP Parallelization OpenMP optimization support (see setup/ifort.setup, for example), optimizes profile loop only. `- export OMP_NUM_THREADS=4` is default in .setup files.

netCDF Interface Support for netCDF4 file format reading: CloudCoeff.nc4 and AerosolCoeff.nc4.

Unit Tests Added various regression and unit tests, see `README.md`.

Performance Improved loop-level performance: 4 to 5 times native improvement by optimizing loops.

Bug Fixes

Cloud Fraction Minimum Threshold Setting the `atmosphere%cloud_fraction=ZERO` now removes cloud contributions to radiance and brightness temperature calculations (but not aerosol contribution). 9. Various minor bug fixes relating to uninitialized variables, improper zeroing, etc.

ADA Stream Number Bugfix A bug with the selection of angular streams in the adding-doubling method has been fixed.

Interface Changes

`CRTM_Init()` Added more optional arguments in `CRTM_Init()` funtion to specify the format and name of aerosol and cloud scattering quantities look-up tables.

What's New in v2.3.0 - released November 21, 2017

New Science

All-Sky radiance simulation under cloud_fraction condition.

Use of all-sky transmittances in FASTEM-X reflection correction.

Improved surface reflectance in radiative transfer calculation for Microwave under scattering condition.

Added ATMS SeaIce emissivity module.

Fix the simulation near 3.9 micron by adding solar contribution in ADA_Module.

Updates of CRTM Coefficients for ABI_GOES-R, AHI_Himawari-8.

Updates of CRTM antenna correction coefficients for MHS_N19/Metop-a.

Updated AIRS coefficients for including NLTE correction.

Added new instrument coefficients for: CrIS-fsrB1/B2/B3_NPP, CrIS*_N20, CrIS-fsr431_npp/n20, AHI_Himawari-9, ABI_G16, VIIRS-JPSS1, ATMS_N20, ATMS_N20-SRF, COWVR, tropics_designed_v1.

Structural Changes

Change `CRTM_MW_Water_SfcOptics.f90` interface variable dimension.

Remove the channel number in the 'all channels' included CRTM coefficient files, e.g. 'cris1305_npp' would become simple 'cris_npp'.

Add CloudFraction test in `check_crtm.fpp`.

In this release, there is a new feature for the simulation of all-sky (cloudy) radiance, which utilizes Fortran class function, and now CRTM will support the new compiler with class function, such as ifort version (14.0+, 15.0+, 16.0+), gfortran version (gcc 4.8.5, 4.9, 5.4, 6.4, 7.2), pgi/17.3, ftn/2.3.0.

Bug Fixes

Bug fix for the function `CRTM_Compute_SfcOptics_AD` in `CRTM_SfcOptics.f90`

Removed 'ERRMSG' option in the `[DE]ALLOCATE` functions to fix an error when using gfortran compiler.

Changed 'Data Statement' to 'Array Assignment' in 6 Surface Emissivity modules to fix a compile failure when using -e08 in ifort/16.0.+.

What's New in v2.1

New Science

Updated microwave sea surface emissivity model The FASTEM4/5 microwave sea surface emissivity models have been implemented. FASTEM5 is the default (via a file loaded during initialisation) and FASTEM4 [**?**] can be selected by specifying the appropriate data file during CRTM initialisation. The previous model, a combination of FASTEM1 [English and Hewison, 1998] and a low frequency model [Kazumori et al., 2008], can still be invoked via the options input to the CRTM functions. An indication of the differences between the FASTEM5/4/1 microwave sea surface emissivity models for some AMSU-A channels (NOAA-15 through MetOp-A) are shown in figures 0.1 to 0.4.

Updated microwave land surface emissivity model The microwave land emissivity model now uses more information about the surface characteristics, specifically soil and vegetation types as well as the leaf area index (LAI), to compute the emissivity. An indication of the impact of the updated microwave land surface emissivity model for some NOAA-18 AMSU-A channels is shown in figure 0.5.

Non-LTE for hyperspectral infrared sensors A model to correct daytime radiances for the non-LTE effect in the shortwave infrared channels has been implemented [**?**]. Currently the correction is applied only to the hyperspectral infrared sensors; AIRS (Aqua), IASI (MetOp-A/B), and CrIS (Suomi NPP). An indication of the impact of including the non-LTE correction for some affected MetOp-A IASI channels is shown in figure 0.6.

Successive Order of Interaction (SOI) radiative transfer algorithm An alternative radiative transfer (RT) solution algorithm [Heidinger et al., 2006] has been implemented and can be selected for use via the options input to the CRTM functions. The default RT solver still remains the Advanced Doubling-Adding (ADA) algorithm [Liu and Weng, 2006][1].

[1]The ADA implementation in the CRTM uses the Matrix Operator Method (MOM) [Liu and Ruprecht, 1996] for calculating layer quantities

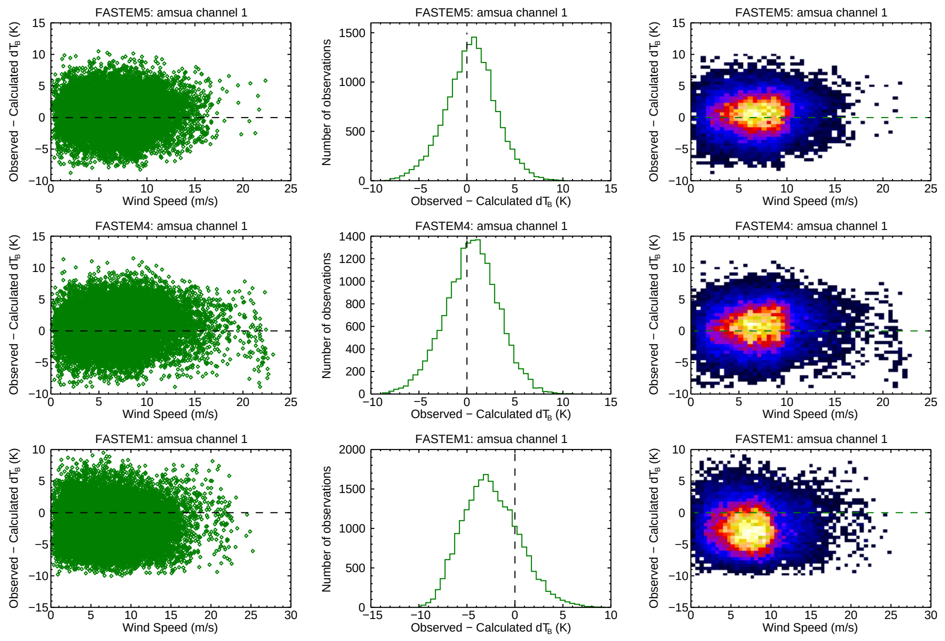

Figure 0.1: GSI single-cycle run (2012060700) results for AMSU-A channel 1 (with QC) comparing use of FASTEM5 (top panels), FASTEM4 (middle pannels), and FASTEM1 (bottom panels). The data plots are, from left to right, simple scatterplot, simple histogram, and 2-D density map (brighter colour indicates higher point density).

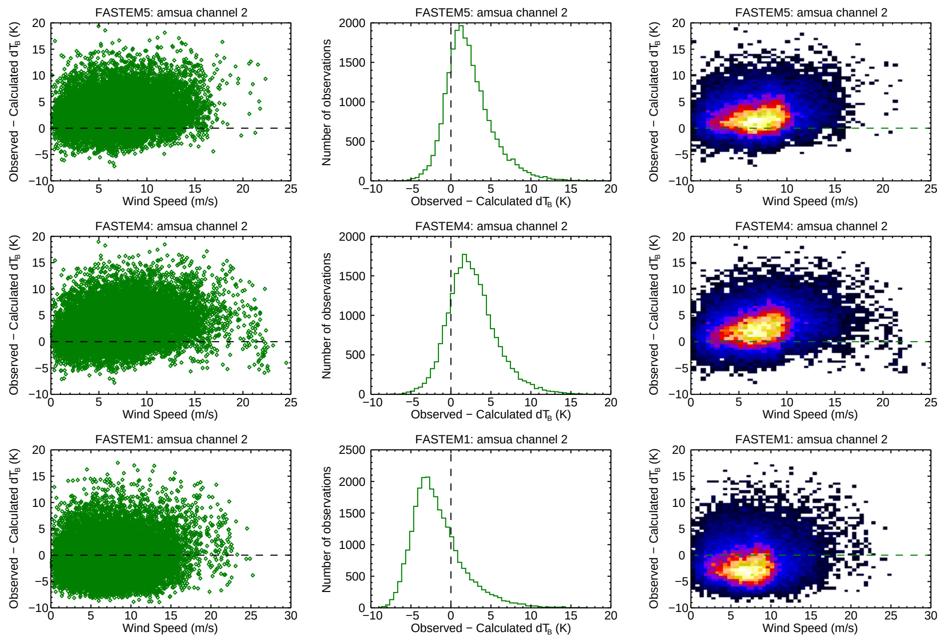

Figure 0.2: GSI single-cycle run (2012060700) results for AMSU-A channel 2 (with QC) comparing use of FASTEM5 (top panels), FASTEM4 (middle pannels), and FASTEM1 (bottom panels). The data plots are, from left to right, simple scatterplot, simple histogram, and 2-D density map (brighter colour indicates higher point density).

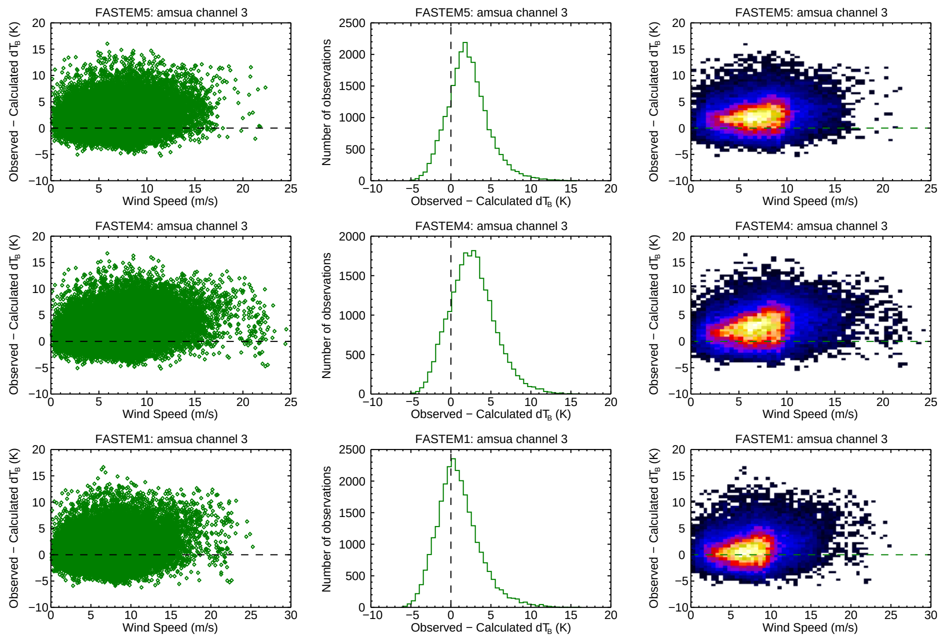

Figure 0.3: GSI single-cycle run (2012060700) results for AMSU-A channel 3 (with QC) comparing use of FASTEM5 (top panels), FASTEM4 (middle pannels), and FASTEM1 (bottom panels). The data plots are, from left to right, simple scatterplot, simple histogram, and 2-D density map (brighter colour indicates higher point density).

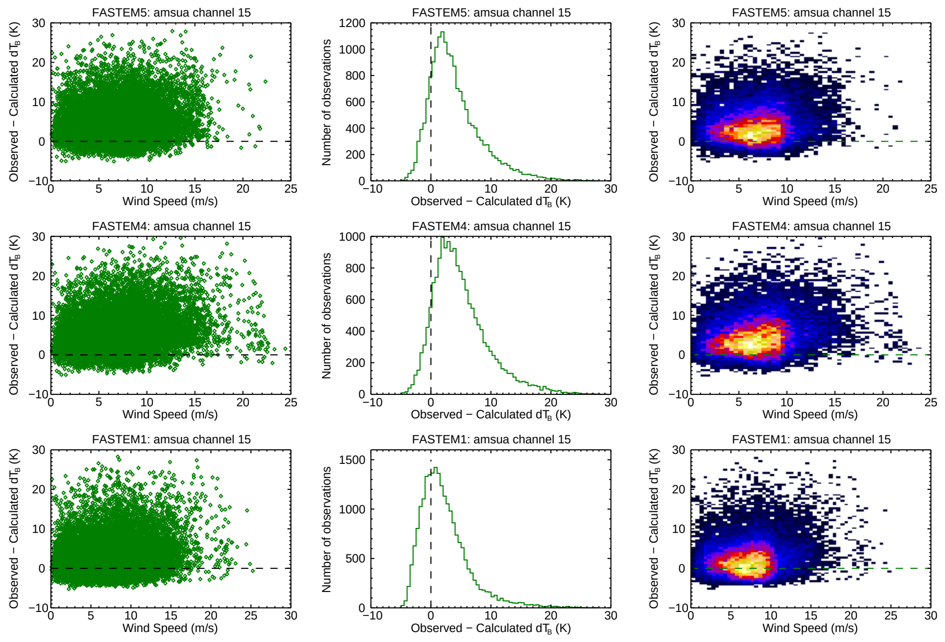

Figure 0.4: GSI single-cycle run (2012060700) results for AMSU-A channel 15 (with QC) comparing use of FASTEM5 (top panels), FASTEM4 (middle pannels), and FASTEM1 (bottom panels). The data plots are, from left to right, simple scatterplot, simple histogram, and 2-D density map (brighter colour indicates higher point density).

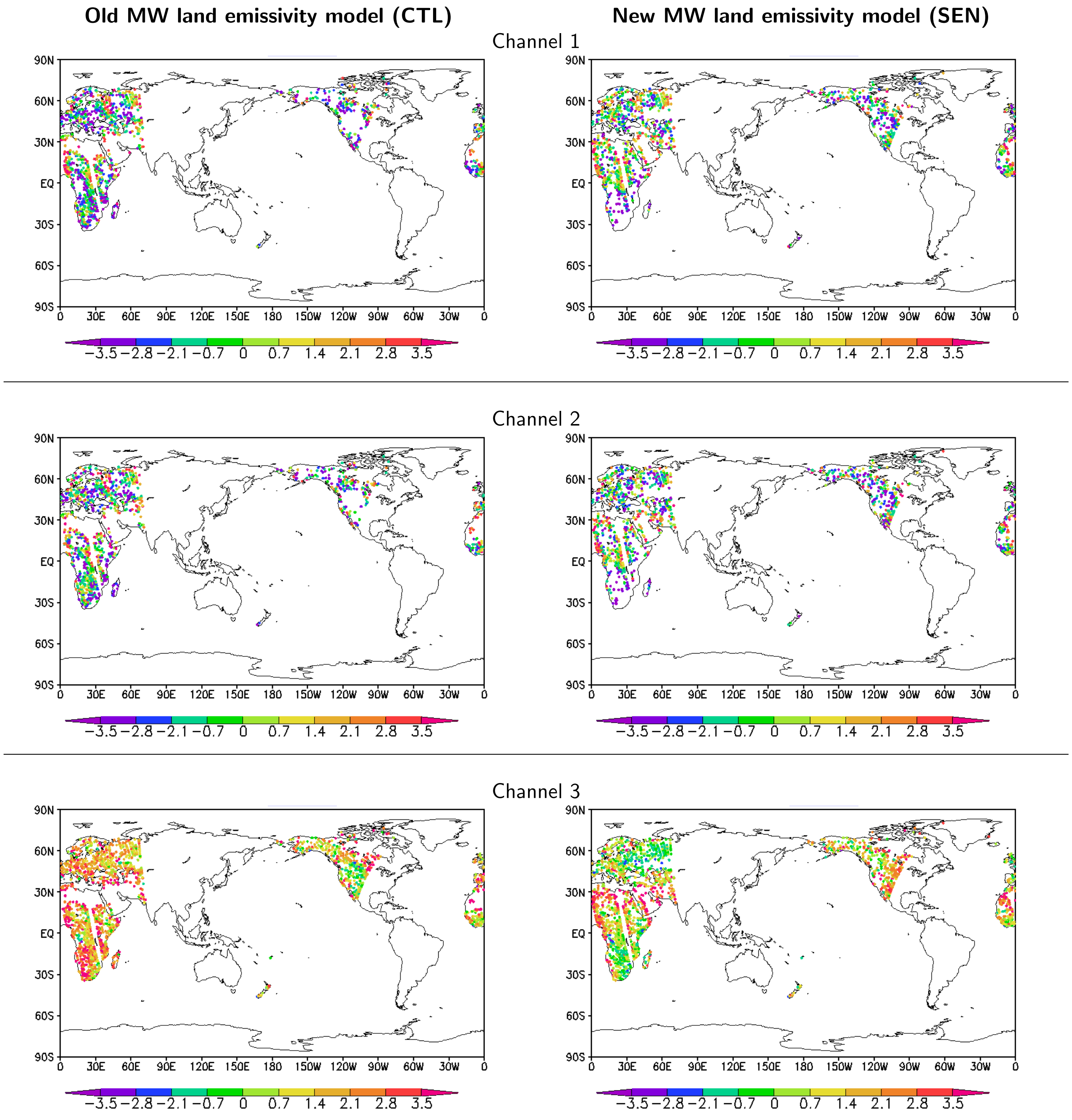

Figure 0.5: Map of NOAA-18 AMSU-A channels 1-3 brightness temperature differences (observed-calculated) for the 2010073112 period for a GSI-GFS full cycle run from 1 July 2010 to 1 August 2010. The control run (CTL) uses the currently operational microwave land emissivity model, and the sensitivity run (SEN) uses the updated microwave land emissivity model.

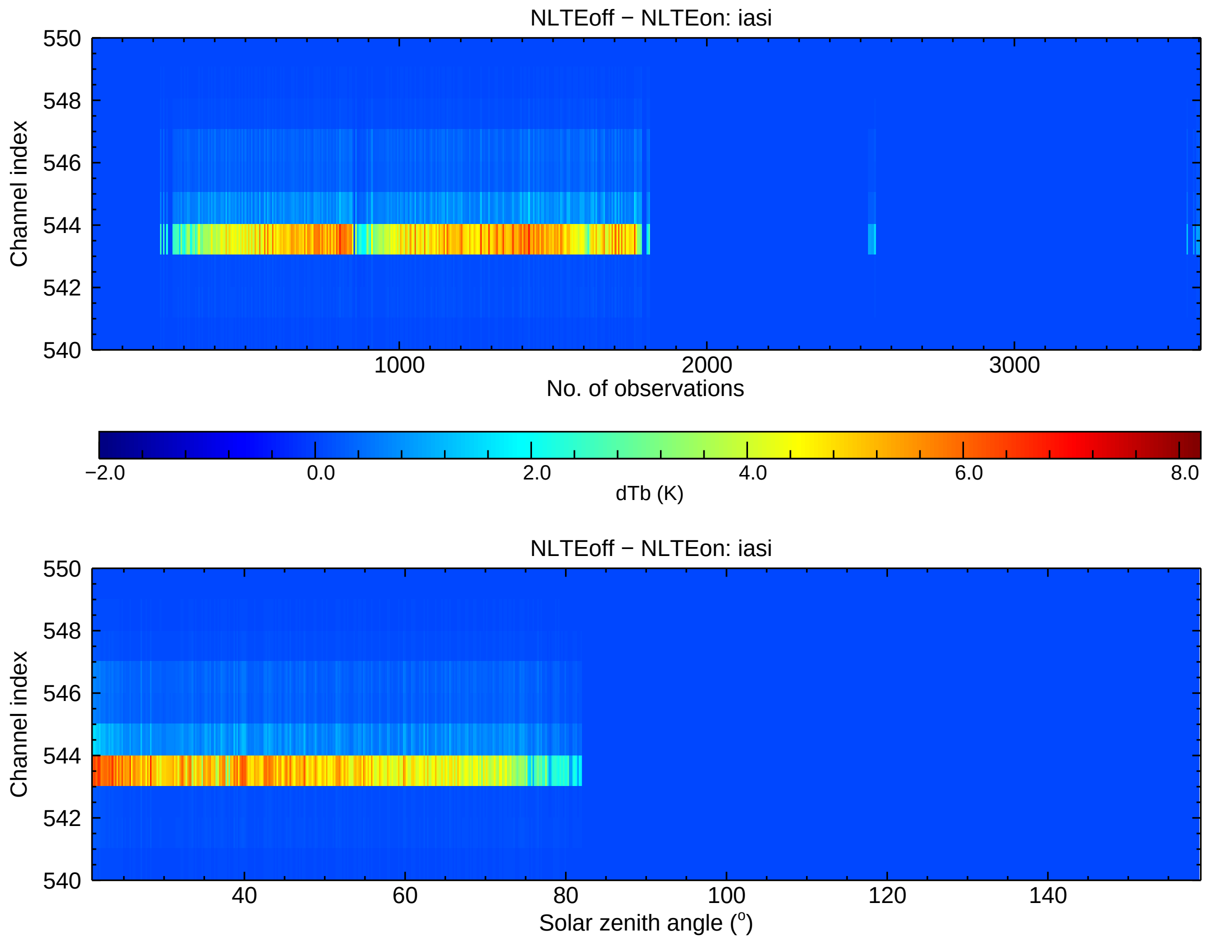

Figure 0.6: GSI single-cycle run (2012070200) results for NLTE-affected MetOp-A IASI channels. The brightness temperature differences are shown as a function of observation (top panel) as well as solar zenith angle (bottom panel). The channel indices are for the 616 channel subset used in NCEP. The channel indices shown correspond to channels in the frequency range 2236.25-2391.00cm^{-1}.

New Functionality

Aerosol optical depth functions Separate functions to compute just the aerosol optical depth have been implemented. The new main level forward, tangent-linear, adjoint, and K-matrix functions are `CRTM_AOD()`, `CRTM_AOD_TL()`, `CRTM_AOD_AD()`, and `CRTM_AOD_K()` respectively. See section 5.3 for the function interfaces.

Channel subsetting To allow users to select which channels of a sensor will be processed, a channel subsetting function has been added. This subsetting operates on the `ChannelInfo`structure and is invoked by passing the list of required channel numbers to a new `CRTM_ChannelInfo_Subset()` function. See section 4.3.6 for usage examples and section A.1.6 for the function interface.

Number of streams option For scattering atmospheres the current method to determine the number of streams to be employed in the radiative transfer calculation is based upon the Mie parameter. Generally this methodology yields a higher number of streams than is necessary. A better "stream selection" method is under development and is slated for the v2.2 CRTM release. Part of this work led to the implementation of an `n_Streams` option - that is, the user can explicitly state the number of streams they wish to use for scattering calculations and override any value determined internally. The user-define number of streams is set via the options input to the CRTM functions.

Scattering switch option for clouds and aerosols This implements a user-selectable switch to "skip" the scattering computations and only compute the cloud and aerosol absorption component when clouds and aerosols are present. The scattering switch is set via the options input to the CRTM functions.

Aircraft instrument capability The ability to simulate an aircraft instrument has been implemented in the CRTM. The user indicates that the calculation is for an aircraft instrument by specifying the flight level pressure in the options input to the CRTM functions. Note, however, that no spectral or transmittance coefficients are available for aircraft instruments. If you wish to run the CRTM for a particular aircraft sensor (microwave, infrared or visible) email the CRTM developers at ncep.list.emc.jcsda_crtm.support@noaa.gov.

Options structure I/O Previously, the CRTM `Options`structure was different from the other user accessible CRTM structures (e.g. `Atmosphere`, `Surface`, `Geometry`, etc) in that there were no means to write and read the structure to/from file. This oversight has been corrected. See section A.9 for the function interfaces.

Interface Changes

Surface type specification changes The specification of surface type in the CRTM surface structure was previously hardwired to use the NPOESS land surface classification scheme (infrared and visible spectral regions only). For users that employed a different land surface classification scheme, in particular those from USGS or IGBP, it meant there was a classification scheme remapping that was required to assign the "correct" NPOESS surface type for a particular USGS or IGBP surface type. To avoid the need to do this remapping, the land surface reflectivity data has now been provided in terms of three surface classification schemes: NPOESS (the default), USGS, and IGBP. These are loaded into the CRTM during the initialization stage.

Previously land surface type parameters such as `SCRUB` or `BROADLEAF_FOREST` were available to refer to a unique surface type index that was used to reference a look up table of spectral reflectances. Now, however, the list of allowable surface types can be different based on the classification scheme with which the CRTM was initialized, and thus the numeric index for a surface type in the list is no longer unique to that surface type. This means there can no longer be a list of pre-specified parameterized surface types like there was with v2.0.x of the CRTM.

Tables 4.13, 4.14, and 4.15 show the surface types, and their index, available for the NPOESS, USGS, and IGBP land surface classification schemes respectively.

Emissivity model initialisation file changes In the v2.0.x CRTM the only emissivity/reflectivity model data loaded during initialisation was that for the infrared sea surface emissivity model. Now datafiles are explicitly loaded for each spectral type (infrared, microwave, and visible) as well as each main surface type (land, water, snow, and ice). This was done to get set up for planned future changes and updates to the emissivity and reflectivity models for various spectral regions and surface types.

In general you can rely on the default data files loaded. See table 4.1 for a list of available data files and their associated optional argument to the CRTM initialisation function.

To migrate from the CRTM v2.0.x initialisation and surface type specification to that implemented in v2.1, see Appendix C, "Migration Path from REL-2.0 to REL-2.1."

1 Introduction

1.1 Conventions

The following are conventions that have been adhered to in the current release of the CRTM framework. They are guidelines intended to make understanding the code at a glance easier, to provide a recognisable "look and feel", and to minimise name space clashes.

1.1.1 Naming of Structure Types and Instances of Structures

The derived data type, or structure[1] type, naming convention adopted for use in the CRTM is,

`[CRTM_]`*name*`_type`

where *name* is an identifier that indicates for what a structure is to be used. All structure type names are suffixed with "`_type`" and CRTM-specific structure types are prefixed with "`CRTM_`". Some examples are,

```
CRTM_Atmosphere_type
CRTM_RTSolution_type
```

An instance of a structure is then referred to via its *name*, or some sort of derivate of its *name*. Some structure declarations examples are,

```
TYPE(CRTM_Atmosphere_type) :: atm, atm_K
TYPE(CRTM_RTSolution_type) :: rts, rts_K
```

where the K-matrix structure variables are identified with a "`_K`" suffix. Similarly, tangent-linear and adjoint variables are suffixed with "`_TL`" or "`_AD`" respectively.

1.1.2 Naming of Definition Modules

Modules containing structure type definitions are termed *definition modules*. These modules contain the actual structure definitions as well as various utility procedures that operate on the structure of the designated type. The naming convention adopted for definition modules in the CRTM is,

`[CRTM_]`*name*`_Define`

where, as with the structure type names, all definition module names are suffixed with "`_Define`" and CRTM-specific definition modules are prefixed with "`CRTM_`". Some examples are,

[1]The terms "derived type" and "structure" are used interchangably in this document.

```
CRTM_Atmosphere_Define
CRTM_RTSolution_Define
```

The actual source code files for these modules have the same name with a "`.f90`" suffix.

1.1.3 Standard Definition Module Procedures

The definition modules for the user-accessible CRTM structures (`Atmosphere`, `Cloud`, `Aerosol`, `Surface`, `Geometry`, `RTSolution`, and `Options`) contain a standard set of procedures for use with the structure being defined. The naming convention for these procedures is,

`CRTM_`*name*_*action*

where the available default actions for each procedure are listed in table 1.1. This is not an exhaustive list but procedures for the actions listed in table 1.1 are guaranteed to be present.

Note, however, that the `ChannelInfo`structure does *not* have any I/O procedures available for it. This is to ensure that the `ChannelInfo`structure can only be populated during initialization of the CRTM.

Table 1.1: Default action procedures available in structure definition modules. † I/O functions not available for the `ChannelInfo` structure.

Action	Type	Description
`OPERATOR(==)`	Elemental function	Tests the equality of two structures.
`Associated`	Elemental function	Tests if the structure components have been allocated.
`Destroy`	Elemental subroutine	Deallocates any allocated structure components.
`Create`	Elemental subroutine	Allocates any allocatable structure components.
`Inspect`	Subroutine	Displays structure contents to `stdout`.
`InquireFile`†	Function	Inquires an existing file for dimensions.
`WriteFile`†	Function	Write an instance of a structure to file.
`ReadFile`†	Function	Loads an instance of a structure with data read from file.

Some examples of these procedure names are,

```
CRTM_Atmosphere_Associated
CRTM_Surface_Inspect
CRTM_Geometry_WriteFile
CRTM_RTsolution_Destroy
```

The relational operator, `==`, is implemented via an overloaded `Equal` action procedure, as is shown below for the `Atmosphere` structure,

```
INTERFACE OPERATOR(==)
  MODULE PROCEDURE CRTM_Atmosphere_Equal
END INTERFACE OPERATOR(==)
```

For a complete list of the definition module procedures for use with the publicly available structures, see section A.

1.1.4 Naming of Application Modules

Modules containing the routines that perform the calculations for the various components of the CRTM are termed *application modules*. The naming convention adopted for application modules in the CRTM is,

`CRTM_`*name*

Some examples are,

```
CRTM_AtmAbsorption
CRTM_SfcOptics
CRTM_RTSolution
```

However, in this case, *name* does not necessarilty refer just to a structure type. Separate application modules are used as required to split up tasks in manageable (and easily maintained) chunks. For example, separate modules have been provided to contain the cloud and aerosol optical property retrieval; similarly separate modules handle different surface types for different instrument types in computing surface optics.

Again, the actual source code files for these modules have the same name with a "`.f90`" suffix. Note that not all definition modules have a corresponding application module since some structures (e.g. `SpcCoeff` structures) are simply data containers.

1.2 Components

The CRTM is designed around three broad categories: atmospheric optics, surface optics and radiative transfer.

1.2.1 Atmospheric Optics

(`AtmOptics`) This category includes computation of the absorption by atmospheric gases (`AtmAbsorption`) and scattering and absorption by both clouds (`CloudScatter`) and aerosols (`AerosolScatter`).

The gaseous absorption component computes the optical depth of the absorbing constituents in the atmosphere given the pressure, temperature, water vapour, ozone, and – for the hyperspectral infrared sensors – trace gas[2] profiles.

The scattering component simply interpolates look-up-tables (LUTs) of optical properties – such as mass extinction coefficient and single scatter albedo – for cloud and aerosol types that are then used in the radiative transfer component. See tables 4.8 and **??** for the current valid cloud and aerosol types, respectively, that are valid in the CRTM.

1.2.2 Surface Optics

(`SfcOptics`) This category includes the computation of surface emissivity and reflectivity for four main surface categories (land, water, snow, and ice). The surface optics models are implemented differently for different surface categories based upon the spectral region of a sensor. Thus, each surface category may have a number of surface types associated with it. This is fully discussed in section 4.6.2.

1.2.3 Radiative Transfer Solution

(`RTSolution`) This category takes the `AtmOptics` and `SfcOptics` data and solves the radiative transfer problem in either clear or scattering atmospheres.

[2] CO_2, CH_4, CO, and N_2O

1.3 Models

The CRTM is composed of four models: a forward model, a tangent-linear model, an adjoint model, and a K-matrix model. These can be represented as shown in equations 1.1a to 1.1d.

$$\mathbf{T_B}, \mathbf{R} = \mathbf{F}(\mathbf{T}, \mathbf{q}, T_s, ...) \tag{1.1a}$$

$$\delta\mathbf{T_B}, \delta\mathbf{R} = \mathbf{H}(\mathbf{T}, \mathbf{q}, T_s, ...\delta\mathbf{T}, \delta\mathbf{q}, \delta T_s, ...) \tag{1.1b}$$

$$\delta^*\mathbf{T}, \delta^*\mathbf{q}, \delta^*T_s, ... = \mathbf{H^T}(\mathbf{T}, \mathbf{q}, T_s, ...\delta^*\mathbf{T_B}) \tag{1.1c}$$

$$\delta^*\mathbf{T}_l, \delta^*\mathbf{q}_l, \delta^*T_{s,l}, ... = \mathbf{K}(\mathbf{T}, \mathbf{q}, T_s, ...\delta^*\mathbf{T_B}) \text{ for } l = 1, 2, ..., L \tag{1.1d}$$

Here $\mathbf{F}$ is the forward operator that, given the atmospheric temperature and absorber profiles ($\mathbf{T}$ and $\mathbf{q}$), surface temperature (T_s), etc., produces a vector of channel brightness temperatures ($\mathbf{T_B}$) and radiances ($\mathbf{R}$).

The tangent-linear operator, $\mathbf{H}$, represents a linearisation of the forward model about $\mathbf{T}$, $\mathbf{q}$, T_s, etc. and when also supplied with perturbations about the linearisation point (quantities represented by the δ's) produces the expected perturbations to the brightness temperature and channel radiances.

The adjoint operator, $\mathbf{H^T}$, is simply the transpose of the tangent-linear operator and produces gradients (the quantities represented by the δ^*'s). It is worth noting that, in the CRTM, these adjoint gradients are accumulated over channel and thus do not represent channel-specific Jacobians.

The K-matrix operator[3], $\mathbf{K}$, is effectively the same as the adjoint but with the results preserved by channel (indicated via the subscript l). In the CRTM, the adjoint and K-matrix results are related by,

$$\delta^*x = \sum_{l=1}^{L} \delta^*x_l \tag{1.2}$$

Thus, the K-matrix results are the derivatives of the diagnostic variables with respect to the prognostic variables, e.g.

$$\delta^*x_l = \frac{\partial T_{B,l}}{\partial x} \tag{1.3}$$

Typically, only the forward or K-matrix models are used in applications. However, the intermediate models are generated and retained for maintenance and testing purposes. Any changes to the CRTM forward model are translated to the tangent-linear model and the latter tested against the former. When the tangent-linear model changes have been verified, the changes then translated to the adjoint model and, as before, the latter is tested against the former. This process is repeated for the adjoint-to-K-matrix models also.

1.4 Design Framework

This document is not really the place to fully discuss the design framework of the CRTM, so it will only be briefly mentioned here. Where appropriate, different physical processes are isolated into their own modules. The CRTM interfaces presented to the user are, at their core, simply drivers for the individual parts. This is shown schematically in the forward and K-matrix model flowcharts of figure 1.1.

A fundamental tenet of the CRTM design is that each component define its own structure definition and application modules to facilitate independent development of an algorithm outside of the mainline CRTM development.

[3]The term K-matrix is used because references to this operation in the literature commonly use the symbol $\mathbf{K}$

By isolating different processes, we can more easily identify requirements for an algorithm with a view to minimise or eliminate potential software conflicts and/or redundancies. The end result sought via this approach is that components developed by different groups can more easily be added into the framework leading to faster implementation of new science and algorithms.

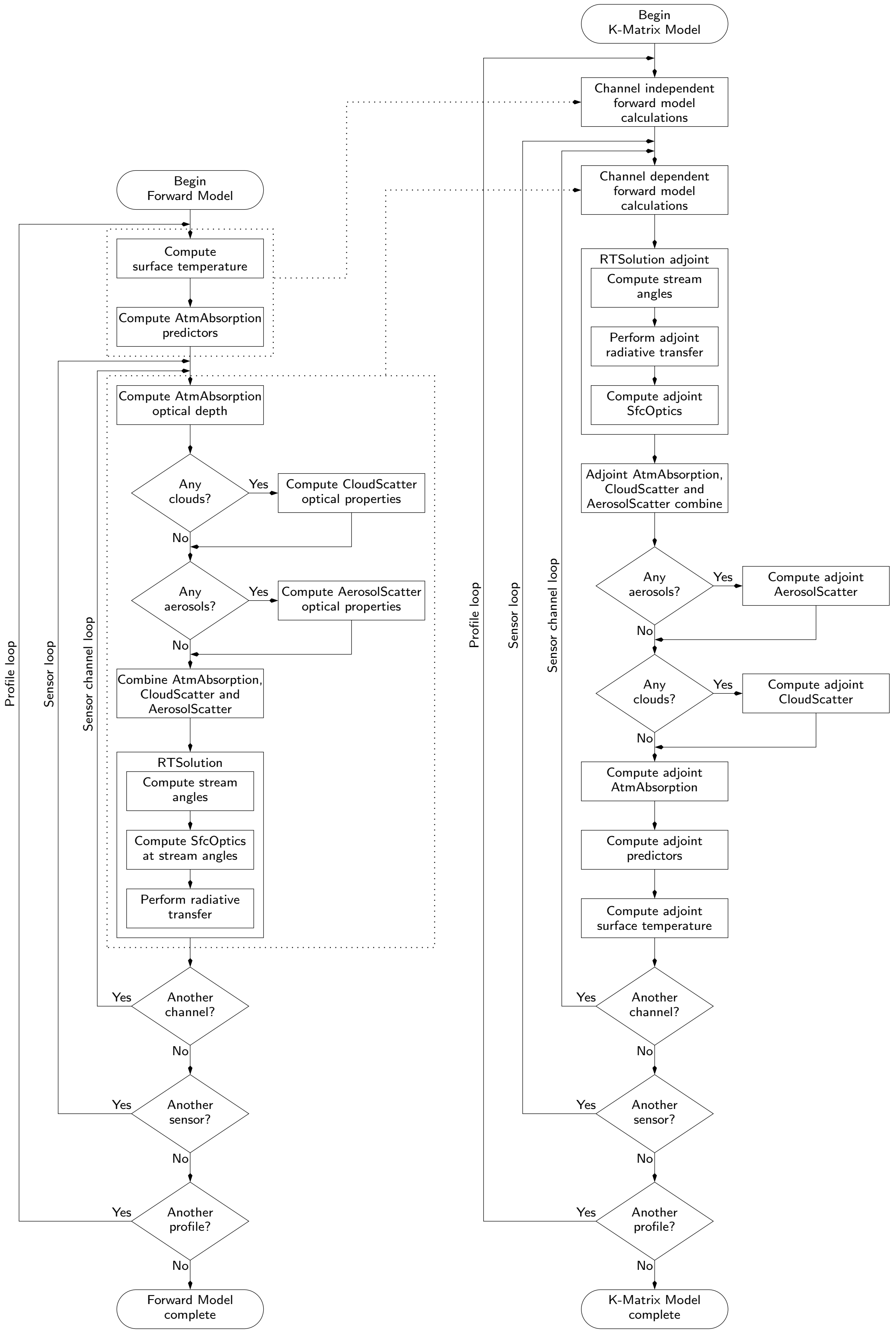
Begin Forward Model
Compute surface temperature
Compute AtmAbsorption predictors
Compute AtmAbsorption optical depth
Any clouds?
Yes
Compute CloudScatter optical properties
No
Any aerosols?
Yes
Compute AerosolScatter optical properties
No
Combine AtmAbsorption, CloudScatter and AerosolScatter
RTSolution
Compute stream angles
Compute SfcOptics at stream angles
Perform radiative transfer
Another channel?
Another sensor?
Another profile?
Forward Model complete
Profile loop
Sensor loop
Sensor channel loop
Begin K-Matrix Model
Channel independent forward model calculations
Channel dependent forward model calculations
RTSolution adjoint
Compute stream angles
Perform adjoint radiative transfer
Compute adjoint SfcOptics
Adjoint AtmAbsorption, CloudScatter and AerosolScatter combine
Any aerosols?
Compute adjoint AerosolScatter
Any clouds?
Compute adjoint CloudScatter
Compute adjoint AtmAbsorption
Compute adjoint predictors
Compute adjoint surface temperature
Another channel?
Another sensor?
Another profile?
K-Matrix Model complete
(a) Forward Model (b) K-Matrix Model

Figure 1.1: Flowchart of the CRTM Forward and K-Matrix models.

2 How to obtain the CRTM

2.1 CRTM GitHub repository

CRTM source code are stored on the following public GitHub repository,with different versions stored in corresponding branches:

https://github.com/JCSDA/crtm/

The users can directly download the code as .zip on this GitHub site.

To download the entire repository with GIT:

```
git clone https://github.com/JCSDA/crtm
```

To obtain source code of a specific branch, for example, CRTM REL-2.4.0:

```
git clone --branch=v2.4.0 https://github.com/JCSDA/crtm
```

2.2 CRTM ftp download site

If you're looking for an older version of CRTM (v2.3.0 or older) you may obtain the appropriate tarball[1] from the CRTM ftp site:

ftp://ftp.emc.ncep.noaa.gov/jcsda/CRTM/

2.3 Coefficient Data

All of the transmittance, spectral, cloud, aerosol, and emissivity coefficient data needed by the CRTM are stored on UCAR remote storage server, and can be obtained during model configuration:

```
sh Get_CRTM_Binary_Files.sh
```

The coefficient directory structure is organized by coefficient and format type as shown in figure 2.1.

[1]A compressed (e.g. gzip'd) tape archive (tar) file.

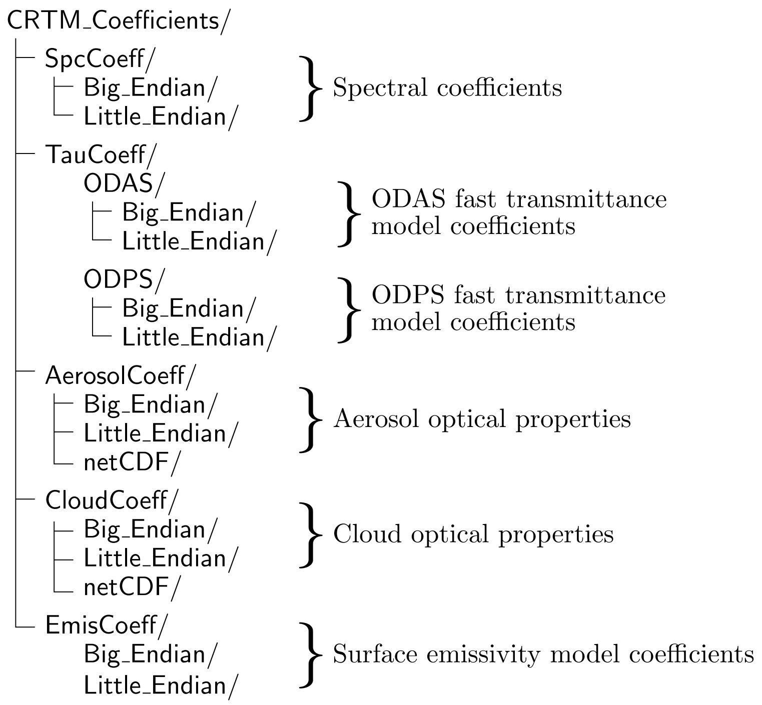

Figure 2.1: The CRTM coefficients directory structure

Both big- and little-endian format files are provided to save users the trouble of switching what they use for their system[2]. CRTM v2.4.0 also provides aerosol and cloud scattering coefficients in netCDF4 format. Note in the TauCoeff directory there are two subdirectories: ODAS and ODPS. These directories correspond to the coefficient files for the different transmittance model algorithms. The user can select which algorithm to use by using the corresponding TauCoeff file.

To run the CRTM, all the required coefficient files need to be in the right path (see the CRTM initialisation function description) so users will have to move/link the datafiles as required.

[2]All of the supplied configurations for little-endian platforms described in Section 3 use compiler switches to default to big-endian format.

3 How to build the CRTM library

3.1 Build Files

The build system for the CRTM is relatively unsophisticated and is constructed for the Unix `sh` shell (or its derivative `bsh`, `bash`, or `ksh` shells). Currently `csh` (or any of its variants) is not supported.

The build system consists of a number of make, include, and configuration files in the CRTM tarball hierarchy:

`makefile`	: The main makefile
`make.macros`	: The include file containing the defined macros.
`make.rules`	: The include file containing the suffix rules for compiling Fortran95/2003 source code.
`configure`	: The directory containing build environment definitions.

3.2 Predefined Configuration Files

The build makefiles now assumes that environment variables (envars) will be defined that describe the compilation and link environment. The envars that *must* be defined are:

`FC`	: the Fortran95/2003 compiler executable,
`FC_FLAGS`	: the flags/switches provided to the Fortran compiler,
`FL`	: the linker used to create the executable test/example programs, and
`FL_FLAGS`	: the flags/switches provided to the linker.

Several shell source files are provided for the build environment definitions for the compilers to which we have access and have tested here at the JCSDA. These shell source files are in the `configure` subdirectory of the tarball. The configuration files provided are shown in table 3.1. Both "production" and debug configurations are supplied, with the former using compiler switches to produce fast code and the latter using compiler switches to turn on all the available debugging capabilities. Note that the debug configurations will produce executables much slower than the production builds.

Table 3.1: Supplied configuration files for the CRTM library and test/example program build.

Platform	Compiler	Production	Debug
Linux	GNU gfortran	`gfortran.setup`	`gfortran_debug.setup`
	Intel ifort	`intel.setup`	`intel_debug.setup`
	PGI pgf95	`pgi.setup`	`pgi_debug.setup`
	g95	`g95.setup`	`g95_debug.setup`
IBM	AIX xlf95	`xlf.setup`	`xlf_debug.setup`

3.3 Compilation Environment Setup

To set the compilation envars for your CRTM build, you need to source the required configuration setup file. For example, to use `gfortran` to build the CRTM you would type

```
. configure/gfortran.setup
```

in the main directory. Note the "." and space preceding the filename. This should print out something like the following:

```
=========================================
 CRTM compilation environment variables:
   FC:       gfortran
   FC_FLAGS:  -c  -O3  -fimplicit-none  -fconvert=big-endian  -ffree-form
              -fno-second-underscore  -frecord-marker=4  -funroll-loops
              -ggdb  -Wall  -std=f2003
   FL:       gfortran
   FL_FLAGS:
   FL_FLAGS:
=========================================
```

indicating the values to which the envars have been set.

Change the supplied setups to suit your needs. If you use a different compiler please consider submitting your compilation setup to be included in future releases.

Note that as of CRTM v2.0, the Fortran compiler needs to be compatible with the ISO TR-15581 Allocatable Enhancements update to Fortran95. Most current Fortran95 compilers do support TR-15581.

To build CRTM v2.4.0, users need to include valid netCDF4 and HDF5 libraries with paths specified by `NC4_DIR` and `HDF_DIR`. For example, in `configure/gfortran.setup`:

```
=========================================
if [[ "$uname" == "Linux"  ]] ; then
              export NC4_DIR=/usr/local/ #singularity container only
              export HDF_DIR=/usr/local/ #singularity container only
elif [[ "$uname" == "Darwin" ]] ; then
              export NC4_DIR="/usr/local/Cellar/netcdf/4.7.4_1"  #mac OS (brew install)
              export HDF_DIR="/usr/local/Cellar/hdf5/1.12.0_1"   #mac OS (brew install)
fi
=========================================
```

Note: As of the time of writing, October 2020, only 'ifort.setup', 'ifort-debug.setup', 'gfortran.setup', 'gfortran-debug.setup' have been actively developed and tested for CRTM 2.4.0. It is strongly recommended that the user use one of these compilers for v2.4.0 until the remaining setup files are updated. Contact the support email address for specific compiler support requests. The c-shell (.csh) extension files have not been updated.

3.4 Build CRTM

3.4.1 CRTM v2.4.0

A detailed instruction for CRTM 2.4.0 configuration and build is given on `CRTM GitHub repository`. The users may also follow the steps below to build CRTM 2.4.0.

Configuration Step 1: Follow the instruction in previous section to set the compilation environment. For example

```
. configure/gfortran.setup
```

Configuration Step 2: Once the compilation environment has been set, set the environment variables to indentify various paths needed for model build

```
. ./Set_CRTM_Environment.sh
```

Configuration Step 3: Download all coefficients to `fix` folder by running:

```
sh Get_CRTM_Binary_Files.sh
```

Now moving to the `Build` directory for model src/Build:

```
cd src/
cd Build/
make clean
cd ..
make realclean
make
```

The commands `make clean` and `make realclean` ensures that the underlying links, compiled files, generated Makefiles. are removed to avoid conflicts with a clean build. The command `make` at the `src/` level performs the linking process between the upper level `src/**` directories and the `src/Build/libsrc` directory.

Note: After runnin `make`, you may see certain `.nc4` files listed as missing, these are files that will be converted to netCDF4 format, but have not yet been added.

Build Step 1: Compile the linked source codes that reside in the `libsrc/` directory:

```
cd Build/
./configure --prefix=$PWD
make clean
make -j4
```

where `-j4` sets the number of parallel make processes to 4.

Build Step 2: Verify that the .mod files have been created and the library file (which external codes link against) has also been created:

```
cd libsrc/
ls -l *.mod
ls -l *.a
```

Build and run the default CRTM test `check_crtm`, located in the `src/Build/libsrc/test` directory:

```
make check
```

Assuming no fatal error messages, you will see `'TEST SUCCESSFUL!'`, which indicates you have sucessfully built CRTM that passed all default tests.

3.4.2 CRTM v2.3.0 or older versions

Once the compilation environment has been set, the CRTM library build is performed by simply typing,

```
    make
```

after which you should see the source file compilation output. Depending on the compiler used you may see various warning messages, for example

```
warning:  'cchar[1]lb:  1 sz:  1' may be used uninitialized in this function
```

or

```
PGF90-I-0035-Predefined intrinsic scale loses intrinsic property
```

etc. The actual format of the warning message depends on the compiler. We are working on eliminating these warning messages where appropriate or necessary.

Note that the current build process is set up to generate a static library not a shared one.

3.5 Testing the library

3.5.1 Legacy Tests

Several test/example programs exercising the forward and K-matrix functions have been supplied with the CRTM. To build and run all these tests, type,

```
    make test
```

This process does generate a lot of output to screen so be prepared to scroll through it. Currently there are ten forward model test, or example, programs:

```
test/forward/Example1_Simple
test/forward/Example2_SSU
test/forward/Example3_Zeeman
test/forward/Example4_ODPS
test/forward/Example5_ClearSky
test/forward/Example6_ChannelSubset
test/forward/Example7_AOD
test/forward/Example9_Aircraft
test/forward/Example10_ScatteringSwitch
test/forward/Example11_SOI
```

And there are nine cases for the K-matrix model:

```
test/k_matrix/Example1_Simple
test/k_matrix/Example2_SSU
test/k_matrix/Example3_Zeeman
test/k_matrix/Example4_ODPS
test/k_matrix/Example5_ClearSky
test/k_matrix/Example6_ChannelSubset
test/k_matrix/Example7_AOD
test/k_matrix/Example10_ScatteringSwitch
test/k_matrix/Example11_SOI
```

Both the forward and K-matrix tests *should* end with output that looks like:

```
======================
SUMMARY OF ALL RESULTS
======================

Passed 42 of 42 tests.
Failed 0 of 42 tests.
```

Currently they both have the same number of tests. If you encounter failures you might see something like:

```
======================
SUMMARY OF ALL RESULTS
======================

Passed 32 of 42 tests.
Failed 10 of 42 tests.  <----<<<  **WARNING**
```

Some important things to note about the tests:

- The supplied results were generated using the `gfortran` DEBUG build.
- Comparisons between DEBUG and PRODUCTION builds can be different due to various compiler switches that modify floating point arithmetic (e.g. optimisation levels), or different hardware.
- For test failures, you can view the differences between the generated and supplied ASCII output files. For example, to view the K-matrix `Example1_Simple` test case differences for the `amsua_metop-a` sensor you would do something like:

  ```
  $ cd test/k_matrix/Example1_Simple
  $ diff -u amsua_metop-a.output results/amsua_metop-a.output | more
  ```

  where the `amsua_metop-a.output` file is generated during the test run, and the `results/amsua_metop-a.output` file is supplied with the CRTM tarball.
- The differences that typically result are quite small (of the order of microKelvin or less when there is a noticable difference in the computed brightness temperatures), although not always at the numerical precision limit.
- A graphical differencing tool such as tkdiff, meld, or FileMerge/opendiff (on Mac OSX) is recommended for viewing the file differences.

3.5.2 Running ctest unit tests

As of Release 2.4.0 the CRTM library provides public regression- and unit test functionality with `ctest` . This was done in anticipation of the integration of the CRTM as a *Unified Forward Operator* in the *Joint Effort for Data Assimilation Integration* framework.

Layout: The ctests are located in the following folder:

```
crtm/src/Build/
```

and said folder has the following structure:

```
  cmake/
  CMakeLists.txt
  mains/
    application/
    regression/
    unit/
  readme_crtm_tests.txt
  test_build/
```

The folder structure is largely self-explanatory. However, it should be noted that the source code for the tests is located in the `mains` folder, the build and execution directory for the tests is `test_build` and the `cmake` folder contains cmake modules necessary for the tests.

Prerequisites: In order to run the ctests, you will ned to `make` and `install` the CRTM library following the directions provided by the CRTM package. These tests use both the generated modules and the `libcrtm.a` of the CRTM library, so it needs to be able to see both of these during the compilation of the ctest executables.

`Cmake` will throw an error if the CRTM `lib/` and `include/` directories are not found. If this is the case, please edit the `CMakeLists.txt` file to point to your current installation folder of the CRTM.

Currently, it is looking for the CRTM in the following folder:

```
$CRTM_SOURCE_ROOT/Build/crtm_v2.4.0-alpha/
```

In the `CMakeLists.txt` file it is looking for this folder in the following line, as a hint to `cmake`:

```
HINTS "$ENVCRTM_SOURCE_ROOT/Build/crtm_v2.4.0-alpha/lib"
```

Running Tests: Executing the ctests is straightforward and should be done in the following way:

```
  cd ./test_build   (directory where things are built and run)
  cmake ..
  make -jn
  ctest .
```

Where `n` is the number of cores you want to dedicate to the build. A slightly more verbose mode (-VV) for the tests can be activated like so:

```
  ls -l  (find your test name)
  ctest -VV -R testname (run an individual test)
```

In order to run individual tests for more detailed information, particularly in the case that one or more tests might fail, you may also run the test by calling its executable directly in the usual way.

Cleanup: In the `test_build` directory, you may need to `"rm -rf *"` in order to remove all build files for the test, particularly if you are adding new tests to the CMakeLists.txt file.

Troubleshooting: A number of common issues are known to occur when the `CMakeLists.txt` file for the ctests is not properly defined. A brief list is given in the following:

```
    "error #7002: Error in opening the compiled module file.  Check INCLUDE paths.   [CRTM_MODULE]"
```

This means that the CRTM module files are not being seen during the compilation of the ctests.

```
    "CMake Error at CMakeLists.txt:101 (message):     CRTM library not found!"
```

This error message means that the CRTM library files (libcrtm.a or libcrtm.so) are not found.

3.5.3 Adding a ctest

With the current configuration of the CRTM, adding a ctest as a unit test is straightforward and every newly developed feature of the CRTM should also include an appropriate unit test to verify its (continued) functionality.

The Fortran side: Writing a new unit test for a new CRTM feature requires creating a Fortran executable specifically for the test. It is the responsibility of the user to find a logical test that measures the proper functionality of their new feature. Assuming that a variable called `TEST_FAILS` keeps track of whether the test passes or not, the test executable needs to be terminated in the following way:

```
PROGRAM myTest
  LOGICAL :: TEST_FAILS
  ...
  IF( TEST_FAILS .EQ. .TRUE.) THEN
    STOP 1
  ELSE
    STOP 0
  END IF
END PROGRAM myTest
```

That is, if the test succeeds, the executable needs to return the integer status 0. Likewise, if the test fails the program needs to return an integer different from zero. For the CRTM, an error code of 1 for test failure is mandatory.

The Cmake side: After creating a Fortran executable for the new unit test, the test needs to be included in the `CMakeLists.txt` file in order to add it to the roster of the testing framework. In order to add the test to the `CMakeLists.txt` file in the `crtm/src/Build/test/` folder, first the test executable needs to be added:

```
add_executable(MYTEST mains/unit/Unit_Test/myTest.f90)
set_target_properties(TEST_AD PROPERTIES
  RUNTIME_OUTPUT_DIRECTORY "$CMAKE_BINARY_DIR/mains/unit/Unit_Test")
target_link_libraries(TEST_AD
                      $CRTM
   $NETCDF_LIBRARIES
   $OpenMP_Fortran_LIBRARIES)
target_include_directories(TEST_AD PRIVATE
  $ENVCRTM_SOURCE_ROOT/Build/crtm_v2.4.0-alpha/include)
```

And subsequently the test needs to be added to the list of tests at the bottom of the file:

```
#### start add_test block ####

add_test (NAME myTest COMMAND $<TARGET_FILE:MYTEST>)
```

3.5.4 Writing TL/AD tests

Adding a new feature to the CRTM in terms of a function or subroutine often also involves coding the corresponding tangent-linear and adjoint routines. As of v2.4.0, you will be required to submit unit tests for the TL/AD procedure of a new CRTM feature that you are developing. At least three basic tests should be included in your submission:

1. A Finite-Difference threshold test for the tangent-linear procedure.
2. A Finite-Difference convergence test for the tangent-linear procedure.
3. An operator-transposition test for the Adjoint subroutine.

You are welcome to add more specialized tests of your own.
The basic algorithms to follow for these tests are very simple:

TL Finite Difference Threshold Test: The TL finite difference threshold test simply ensures that the newly coded analytical tangent linear operator **H** is sufficiently close to a finite difference approximation using the full nonlinear forward operator $\mathbf{F}(\mathbf{T}, \mathbf{q}, T_s, ...)$:

$$\mathbf{F}(\mathbf{T}+\delta\mathbf{T}, \mathbf{q}+\delta\mathbf{q}, T_s+\delta T_s, ...) - \mathbf{F}(\mathbf{T}, \mathbf{q}, T_s, ...) \approx \mathbf{H}(\mathbf{T}, \mathbf{q}, T_s, ...) + \epsilon \tag{3.1}$$

For some threshold ϵ and perturbations $\delta\mathbf{T}, \mathbf{q}, \delta\mathbf{q}, \delta T_s$... etc. . When writing the test, you should perturb **all** input variables of the forward procedure at the same time by a value of around 10 percent of the original variable, e.g.:

$$\delta\mathbf{T} = 0.1 \cdot \mathbf{T}, \tag{3.2}$$

and check if equation 3.1 is fulfilled.

TL Finite Difference Convergence Test: The TL convergence test takes the threshold test and stepwise reduces the perturbation by half upon each iteration. In order for the test to pass, the convergence needs to be monotonous.

AD Operator-Transposition Test: The AD operator-transposition test checks if the newly-coded adjoint routine $\mathbf{H^T}$ produces a Jacobian (K-) matrix **K** that is the transpose of the Jacobian resulting from the TL routine within a given tolerance.

$$(\mathbf{H}(\mathbf{T}, \mathbf{q}, T_s, ...))^T = \mathbf{H^T}(\mathbf{T}, \mathbf{q}, T_s, ...) \tag{3.3}$$

3.6 Installing the library

A very simple install target is specified in the supplied makefile to put all the necessary include files (the generated `*.mod` files containing all the procedure interface information) in an `/include` subdirectory and the library itself (the generated `libCRTM.a` file) in a `/lib` subdirectory. The make command is

```
make install
```

The `/include` and `/lib` subdirectories can then be copied/moved/linked to a more suitable location on your system, for example: `$HOME/local/CRTM`

NOTE: Currently, running the tests also invokes this install target. That will change in future tarball releases so do not rely on the behaviour.

3.7 Clean Up

Two cleanup targets are provided in the makefile:

```
make clean
```

Removes all the compilation and link products from the `libsrc/` directory.

```
make distclean
```

This does the same as the "clean" target but also deletes the library and include directories created by the "install" target.

3.8 Linking to the library

Let's assume you've built the CRTM library and placed the `/include` and `/lib` subdirectories in your own local area, `$HOME/local/CRTM`. In the makefile for your application that uses the CRTM, you will need to add

```
-I$HOME/local/CRTM/include
```

to your list of compilation switches, and the following to your list of link switches,

```
-L$HOME/local/CRTM/lib -lCRTM
```

3.9 CRTM v2.4.0 in JCSDA JEDI environment

In most cases, you'll be running CRTM inside of a JEDI bundle (e.g. fv3-bundle, ufo-bundle, etc.) so you'll have no need to follow any directions here. However, if you're interested in build CRTM stand-alone using a JEDI environment (i.e., for testing purposes, running the ctests, and etc.), please follow the instructions below.

To build CRTM 2.4.0 in a suitable JCSDA JEDI environment (either HPC enabled, a JEDI container, JEDI-stacks, or at a bare minimum ecbuild from ectools):

Configuration:

```
git clone https://github.com/JCSDA/crtm      (you've probably done this already)
cd crtm/
git fetch
git pull
sh Get_CRTM_Binary_Data.sh
```

Build:

```
mkdir build
cd build
ecbuild pathtocrtm
```

where `pathrocrtm` is where the `crtm/` diretory is located. In this example if you're in the `crtm/build` directory, typing `ecbuild ..` will work.

Linking to the library: You'll find the library file (`libcrtm.so`) in `build/lib` and the module files (e.g., `*.mod`) in `module/crtm/**`. You can link to these files using any codes that call the CRTM.

Uninstalling the library: To `uninstall` the library (assuming you haven't moved the installation directory contents somewhere else) you can type:

```
cd build/
rm -rf *  (make sure you do this in the build/ directory where you ran 'ecbuild')
```

Cleaning Up: To `uninstall` the library (assuming you haven't moved the installation directory contents somewhere else) you can type:

```
cd build/
rm -rf *  (make sure you do this in the build/ directory where you ran `ecbuild`)
```

Additional options: You can modify the various compiler flags, etc in the `crtm/cmake/` directory. There you will find several configuration files based on differen compilers.

3.10 "Build Release" Setup and Configuration (CRTM v2.4.0 optional)

3.10.1 Legacy auto configuration

Within the `src/Build` directory, the legacy build system for the CRTM uses an autoconf-generated `configure` script, which depends on the existence of a few key files: (1) the `configure.ac` file, which contains instructions for how the `configure` file will be built when the `autoconf` command is executed. (2) The `Makefile.in` file, which contains instructions on how executing the `configure` script will generate `Makefile` in libsrc and test subdirectories.

The Build `Makefile` assume that environment variables (envars) will be defined that describe the compilation environment. The envars that must be defined are: FC: the Fortran95/2003 compiler executable, FCFLAGS: the flags/switches provided to the Fortran compiler. These can be set by `./config-setup/<compiler>.setup` in the `Build` directory, as described previously.

3.10.2 Configuration options

`configure` sets an install path environment variable, among other things. This, by default, will set the `lib/` and `include/` directory paths in the `/usr/local/crtm_v2.4.0/` (or whatever string in in `src/CRTM_Version.inc`).

The `--prefix` switch sets the installation directory, make sure you have write access to that directory.

You can override this by setting a different install directory as follows:

```
./configure --prefix=<install directory>
```

For example, to create the library in the directory in which you're currently in (e.g., `crtm/src/Build/crtm_v2.4.0/`).

```
./configure --prefix=${PWD}
```

By default, the CRTM is built for big-endian I/O. The –disable-big-endian switch builds the library and test programs for little-endian I/O:

```
./configure --disable-big-endian --prefix=<install directory>
```

If you need more flexibility in the library build you can specify the necessary information directly to the configure script that generates the Makefiles. For example, for the Intel ifort compiler:

```
./configure --prefix=$PWD
            --disable-big-endian
            FC="ifort"
            FCFLAGS="-O3 -qopenmp -g -traceback"
```

This overrides the FC and FCFLAGS variables that were set by sourcing the `configuration/` file earlier, it is strongly recommended that you use the provided configuration files since they contain flags that have been added after substantial debugging and testing.

4 How to use the CRTM library

This section will hopefully get you started using the CRTM library as quickly as possible. Refer to the following sections for more information about the structures and interfaces.

There are many variations in what information is known ahead of time (and by "ahead of time" we mean at compile-time of your code), so we'll approach this via examples where pretty much all the dimensional information is unknown. It's a little more effort to set up, but makes for more flexible applications. Of course, for simplicity, one can choose to hardwire dimensions (e.g. number of profiles, number of sensors, etc) in their calling code. It is left as an exercise to the reader to tailor calls to the CRTM in their application code according to their particular needs.

With regards to sensor identification, the CRTM uses a character string – referred to as the `Sensor_Id` – to distinguish sensors and platforms. The lists of currently supported sensors, along with their associated `Sensor_Id`'s, are shown in appendix B.

4.1 Access the CRTM module

All of the CRTM user procedures, parameters, and derived data type definitions are accessible via the container module `CRTM_Module`. Thus, one needs to put the following statement in any calling program, module or procedure,

```
USE CRTM_Module
```

Once you become familiar with the components of the CRTM you require, you can also specify an `ONLY` clause with the `USE` statement,

USE CRTM_Module[, ONLY:*only-list*]

where *only-list* is a list of the symbols you want to "import" from `CRTM_Module`. This latter form is the preferred style for self-documenting your code; e.g. when you give the code to someone else, they will be able to identify from which module various symbols in your code originate.

4.2 Declare the CRTM structures

To compute satellite radiances you need to declare structures for the following information,

1. Atmospheric profile data such as pressure, temperature, absorber amounts, clouds, aerosols, etc. Handled using the `Atmosphere` structure.
2. Surface data such as type of surface, temperature, surface type specific parameters etc. Handled using the `Surface` structure.

3. Geometry information such as sensor scan angle, zenith angle, etc. Handled using the `Geometry` structure.
4. Instrument information, particularly which instrument(s), or sensor(s)[1], you want to simulate. Handled using the `ChannelInfo` structure.
5. Results of the radiative transfer calculation. Handled using the `RTSolution` structure.
6. Optional inputs. Handled using the `Options` structure.

Let's look at the general case where we want to construct CRTM calls where *all* of the relevant dimensions can be dynamically set. So, first define some variables to hold the dimension values,

```
! Dimension variable
INTEGER :: n_channels  ! l = 1, ... , L
INTEGER :: n_profiles  ! m = 1, ... , M
INTEGER :: n_sensors   ! n = 1, ... , N
```

For this general case, all of the CRTM structure array definitions will be allocatable. The forward model declarations would look something like,

```
! Processing parameters
CHARACTER(20)              , ALLOCATABLE :: sensor_id(:)  ! N
TYPE(CRTM_ChannelInfo_type) , ALLOCATABLE :: chinfo(:)     ! N
TYPE(CRTM_Geometry_type)    , ALLOCATABLE :: geo(:)        ! M
TYPE(CRTM_Options_type)     , ALLOCATABLE :: opt(:)        ! M
! Forward declarations
TYPE(CRTM_Atmosphere_type)  , ALLOCATABLE :: atm(:)        ! M
TYPE(CRTM_Surface_type)     , ALLOCATABLE :: sfc(:)        ! M
TYPE(CRTM_RTSolution_type)  , ALLOCATABLE :: rts(:,:)      ! L x M
```

If you are also interested in calling the K-matrix model, you will also need the following declarations,

```
! K-Matrix declarations
TYPE(CRTM_Atmosphere_type)  , ALLOCATABLE :: atm_K(:,:)  ! L x M
TYPE(CRTM_Surface_type)     , ALLOCATABLE :: sfc_K(:,:)  ! L x M
TYPE(CRTM_RTSolution_type)  , ALLOCATABLE :: rts_K(:,:)  ! L x M
```

4.3 Initialise the CRTM

The CRTM is initialised by calling the `CRTM_Init()` function. This loads all the various coefficient data used by CRTM components into memory for later use. The CRTM initialisation is profile independent, so we're only dealing with sensor information here. As such, we have to allocate the `sensor_id` and `chinfo` arrays to handle the number of sensors we want to process. Most users set this value to one (i.e. process a single sensor for each CRTM initialisation) but for this example we'll set it to *six* and use the various MetOp-A sensors: AMSU-A, MHS, HIRS/4, IASI, and AVHRR/3. Why not five? Keep reading...

The array allocations would look like,

```
INTEGER :: alloc_stat
....
! Allocate sensor arrays
```

[1]The terms "instrument" and "sensor" are used interchangeably in this document.

```
n_sensors = 6
ALLOCATE( sensor_id(n_sensors), &
          chinfo(n_sensors), &
          STAT = alloc_stat )
IF ( alloc_stat /= 0 ) THEN
  handle error...
END IF
```

Referring to appendix B, we can now fill the `sensor_id` array with the sensor identifiers that the CRTM understands,

```
sensor_id = (/ 'amsua_metop-a'   , &
               'mhs_metop-a'     , &
               'hirs4_metop-a'   , &
               'iasi_metop-a'    , &
               'avhrr3_metop-a'  , &
               'v.avhrr3_metop-a' /)
```

Note the last sensor identifier with the "`v.`" prefix – indicating a visible wavelength sensor. Currently the CRTM treats visible channels as a separate instrument from infrared channels in those cases where the same sensor has both.[2] This is why the five sensors required six sensor identifiers.

Now that we have our input `sensor_id` array defined, we can call the CRTM initialisation function,

```
INTEGER :: err_stat
....
err_stat = CRTM_Init( sensor_id, chinfo )
IF ( err_stat /= SUCCESS ) THEN
  handle error...
END IF
```

Here we see for the first time how the main CRTM functions let you know if they were successful. As you can see the `CRTM_Init()` function result is an error status that is checked against a parameterised integer error code, `SUCCESS`. The function result should *not* be tested against the actual value of the error code, just its parameterised name. Other available error code parameters are `FAILURE`, `WARNING`, and `INFORMATION` – although the latter is never used as a function result.

The `CRTM_Init()` function called shown above illustrates the simplest call interface assuming the default value for all the optional arguments. Some examples of the use of these optional arguments are shown below.

4.3.1 Where are the coefficient data files?

The default setup for the CRTM initialisation function is that all of the coefficient data files reside in the directory from which the calling program was invoked.

That situation is rarely the case. To get the CRTM initialisation to use a different location for the coefficient files, you use the optional `File_Path` argument. For example, let's assume that all the required datafiles reside in the subdirectory `./coeff_data`. The initialisation call would look like,

```
INTEGER :: err_stat
....
```

[2]It is a lower priority, but this will likely be changed in future CRTM releases as it exposes a wee bit too much of the internal CRTM plumbing to the user.

```
err_stat = CRTM_Init( sensor_id, chinfo, &
                      File_Path = './coeff_data' )
IF ( err_stat /= SUCCESS ) THEN
  handle error...
END IF
```

4.3.2 No clouds or aerosols?

If you know ahead of time that your CRTM usage will not require the computation of cloud and/or aerosol scattering quantities, you can use the optional `Load_CloudCoeff` and `Load_AerosolCoeff` logical arguments to the `CRTM_Init()` function to prevent the cloud and/or aerosol optical properties look-up tables (LUTs) being read in. For example, the syntax to load the cloud, but not the aerosol, LUTs would be something like,

```
INTEGER :: err_stat
....
err_stat = CRTM_Init( sensor_id, chinfo, &
                      Load_CloudCoeff   = .TRUE., &
                      Load_AerosolCoeff = .FALSE. )
IF ( err_stat /= SUCCESS ) THEN
  handle error...
END IF
```

4.3.3 Read in cloud/aerosol coefficients from netCDF look-up tables?

If you need to use the cloud and/or aerosol scattering properties, the CRTM will read in these data from the default binary LUTs (`AerosolCoeff.bin` and `CloudCoeff.bin`). If you wish to read in these data from the netCDF4 LUTs, you can set the optional arguments `AerosolCoeff_Format` and `CloudCoeff_Format` in `CRTM_Init()` function. For example, to read both aerosol and cloud coefficients from netCDF4 LUTs:

```
INTEGER :: err_stat
....
err_stat = CRTM_Init( sensor_id, chinfo, &
                      AerosolCoeff_Format = 'netCDF', &
                      AerosolCoeff_File = 'AerosolCoeff.nc4', &
                      CloudCoeff_Format = 'netCDF', &
                      CloudCoeff_File = 'CloudCoeff.nc4', &
                      NC_File_Path='./coeff_data_nc')
IF ( err_stat /= SUCCESS ) THEN
  handle error...
END IF
```

4.3.4 Which aerosol coefficient look-up table?

CRTM v2.4 supports a new aerosol coefficient LUT that generated based on the aerosol size specifications in The Community Multiscale Air Quality Modeling System (CMAQ). If you wish to use the aerosol scattering quantities in the CMAQ LUT (`AerosolCoeff.CMAQ.bin` and `AerosolCoeff.CMAQ.nc4`), you can specify the optional arguments `Aerosol_Model`, `AerosolCoeff_Format` and `AerosolCoeff_File` in `CRTM_Init()` function. For example, to read in CMAQ aerosol coefficient LUT in binary format:

```
INTEGER :: err_stat
```

```
....
err_stat = CRTM_Init( sensor_id, chinfo, &
                      Aerosol_Model = 'CMAQ', &
                      AerosolCoeff_Format = 'binary', &
                      AerosolCoeff_File = 'AerosolCoeff.CMAQ.bin', &
                      File_Path='./coeff_data')
IF ( err_stat /= SUCCE:SS ) THEN
  handle error...
END IF
```

4.3.5 What surface emissivity model?

The data required for some of the surface emissivity models are also loaded via files (in others the data are hard-coded into the source modules.) Table 4.1 shows the choices available during initialisation for setting up the surface emissivity models.

Table 4.1: Choices available for setup of the various emissivity/reflectivity models during CRTM initialisation. †Default file loaded if optional argument not specified. ‡The same classification scheme file should be loaded for both the infrared and visible land surface emissivity model.

Emissivity or Reflectivity Model	Optional argument	Available files

Infrared Land‡	`IRlandCoeff_File`	`NPOESS.IRland.EmisCoeff.bin`†
		 `USGS.IRland.EmisCoeff.bin`
		 `IGBP.IRland.EmisCoeff.bin`

Infrared Water	`IRwaterCoeff_File`	`Nalli.IRwater.EmisCoeff.bin`† `WuSmith.IRwater.EmisCoeff.bin`
Microwave Water	`MWwaterCoeff_File`	`FASTEM5.MWwater.EmisCoeff.bin`† `FASTEM4.MWwater.EmisCoeff.bin`

Visible Land‡	`VISlandCoeff_File`	`NPOESS.VISland.EmisCoeff.bin`†
		 `USGS.VISland.EmisCoeff.bin`
		 `IGBP.VISland.EmisCoeff.bin`

An example of specifying different data files for all the models listed in table 4.1 is shown below,

```
INTEGER :: err_stat
....
err_stat = CRTM_Init( sensor_id, chinfo, &
                      IRlandCoeff_File  = 'IGBP.IRland.EmisCoeff.bin', &
                      IRwaterCoeff_File = 'WuSmith.IRwater.EmisCoeff.bin', &
                      MWwaterCoeff_File = 'FASTEM4.MWwater.EmisCoeff.bin', &
                      VISlandCoeff_File = 'IGBP.VISland.EmisCoeff.bin' )
IF ( err_stat /= SUCCESS ) THEN
  handle error...
END IF
```

It must be pointed out that you should specify the same classification file for the infrared and visible land surface emissivity models. For example, do not initialise the infrared land model with the USGS file and the visible land model with the IGBP file. This is because the allowed surface types are now stored in the file and mixing the allowable surface types could cause unexpected results. See section 4.6 below regarding the specification of the surface type via the `Surface` structure.

4.3.6 *I don't want to process all of the channels!*

Prior to v2.1, once the CRTM was initialised for a sensor, the calculations were performed for *all* of the channels of that sensor. There is now a capability to dynamically select the channels to process. This is done after a CRTM initialisation has occurred but is mentioned here as the `ChannelInfo` structure is modified to achieve this.

A new series of functions that operate on the `ChannelInfo` structure have been included that allow you to select the channel to process. For example, let's say you only want to process channels 1000-1100 of hte MetOp-A IASI instrument in our example. This can be achieved via a call to the `CRTM_ChannelInfo_Subset` function,

```
INTEGER :: i
....
! Specify an IASI channel subset for processing example
err_stat = CRTM_ChannelInfo_Subset( chinfo(4), &
                                    Channel_Subset = (/(i,i=1000,1100)/) )
IF ( err_stat /= SUCCESS ) THEN
  handle error...
END IF
```

where the `chinfo(4)` references the `ChannelInfo` structure for IASI from the initialisation.

And one more example for subsetting AMSU-A (i.e. `chinfo(1)`) where we only want to process channels 5-8,

```
! Specify an AMSU-A channel subset for processing example
err_stat = CRTM_ChannelInfo_Subset( chinfo(1), &
                                    Channel_Subset = (/5,6,7,8/) )
IF ( err_stat /= SUCCESS ) THEN
  handle error...
END IF
```

You can call this function as many times as you like with different channel sets for different sensors. If you *do* want to process all the sensors channels after selecting a subset, you can easily go back to all-channel processing by using the optional `Reset` logical argument,

```
! Reset back to all-channel processing
err_stat = CRTM_ChannelInfo_Subset( chinfo(1), &
                                    Reset = .TRUE. )
IF ( err_stat /= SUCCESS ) THEN
  handle error...
END IF
```

The `Reset` argument overrides any channel subset specification.

One more thing: because the total number of channels to be processed can now vary dynamically, there is also a "channel counter" function to determine how many channels will be processed. It is an elemental[3] function so you can call it for a single `ChannelInfo` entry,

[3]An elemental procedure may be called with scalar arguments or conformable array arguments of any rank.

```
! Count the number of IASI channels to be processed
n_Channels = CRTM_ChannelInfo_n_Channels( chinfo(4) )
```

or you can call it for all the sensors defined in the `ChannelInfo` array `chinfo`,

```
! Count the number of ALL the channels to be processed
n_Total_Channels = SUM(CRTM_ChannelInfo_n_Channels( chinfo ))
```

4.4 Allocate the CRTM arrays

The first step is to allocate all of the structure arrays to the required size. For our example, let's assume we'll be processing sets of 50 atmospheric profiles, and return to some of the other structure arrays defined in section 4.2,

```
INTEGER :: alloc_stat
....
! Allocate profile-only arrays
n_profiles = 50
ALLOCATE( geo(n_profiles), &
          opt(n_profiles), &
          atm(n_profiles), &
          sfc(n_profiles), &
          STAT = alloc_stat )
IF ( alloc_stat /= 0 ) THEN
  handle error...
END IF
```

But what about the `RTSolution` structure array, `rts`, which has the dimensions `n_channels` × `n_profiles`? Or the K-matrix arrays `atm_K`, `sfc_K`, and `rts_K`? How many channels should be used in their allocation?

The answer is simple, even if mildly unsatisfying: while there is nothing to preclude you from allocating the channel-dependent structure arrays for *all* the channels ...... the number of channels for the `rts` allocation should be for a single sensor. Why? Well, primarily because it is unlikely that the data in the other input structure arrays can (should?) be considered the same for the other sensors – even if they are on the same platform. The simplest example is the `Geometry` structure array, `geo`, where the sensor scan geometry is going to be quite different for different sensors on the same platform. Similarly for the `Surface` structure array, `sfc`, where different sensor field-of-view (FOV) geometries will lead to different surface properties.

So now we introduce a channel-dependence to the usage of the CRTM input structure arrays. Starting with their allocation, let's put these in a loop over sensor, and use the `CRTM_ChannelInfo_n_Channels` from the previous section,

```
INTEGER :: n
....
Sensor_Loop: DO n = 1, n_sensors
  ....
  ! Get the number of channels to process for current sensor
  n_channels = CRTM_ChannelInfo_n_Channels( chinfo(n) )

  ! Allocate channel-dependent arrays
  ALLOCATE( rts(n_channels, n_profiles)  , &
            atm_K(n_channels, n_profiles), &
            sfc_K(n_channels, n_profiles), &
            rts_K(n_channels, n_profiles), &
            STAT = alloc_stat )
  IF ( alloc_stat /= 0 ) THEN
    handle error...
  END IF
  ....
END DO Sensor_Loop
```

4.5 Create the CRTM structures

Now we need to create instances of the various CRTM structures where necessary to hold the input or output data.

Subroutines are used to perform the necessary creation of the CRTM structures by allocating the internal components. The procedure naming convention is `CRTM_`*object*`_Create` where, for typical usage, the CRTM structures that need to be allocated are the `Atmosphere`, `RTSolution` and, if used, `Options` structures. Potentially, the `SensorData` component of the `Surface` structure may also need to be allocated to allow for input of sensor observations for some of the NESDIS microwave surface emissivity models.

The `CRTM_`*object*`_Create` procedures are always elemental and can be invoked for scalar or conformable arrays arguments.

4.5.1 Allocation of the Atmosphere structures

First, we'll allocate the atmosphere structures to the required dimensions. For simplicity, let's assume that the number of layers, gaseous absorbers, clouds, and aerosols are the same for all the profiles. The creation of the forward atmosphere structures is done like so,

```
INTEGER :: n_layers, n_absorbers
INTEGER :: n_clouds, n_aerosols
....
! Some default dimensions
n_layers    = 64
n_absorbers = 2
n_clouds    = 1
n_aerosols  = 2

! Allocate the forward atmosphere structures
CALL CRTM_Atmosphere_Create( atm        , &
                             n_layers   , &
                             n_absorbers, &
                             n_clouds   , &
                             n_aerosols   )
! Check they were created successfully
IF ( ANY(.NOT. CRTM_Atmosphere_Associated( atm )) ) THEN
  handle error...
END IF
```

and the K-matrix structures can be allocated by looping over all profiles,

```
INTEGER :: m
....
! Allocate the K-matrix atmosphere structures
DO m = 1, n_profiles
  CALL CRTM_Atmosphere_Create( atm_k(:,m) , &
                               n_layers   , &
                               n_absorbers, &
                               n_clouds   , &
                               n_aerosols   )
  ! Check they were created successfully
  IF ( ANY(.NOT. CRTM_Atmosphere_Associated( atm_k(:,m) )) ) THEN
    handle error...
  END IF
END DO
```

The CRTM_Atmosphere_Create function is defined as elemental so the profile loop is not strictly needed. The above K-matrix creation example is equivalent to

```
! Allocate the K-matrix atmosphere structures
CALL CRTM_Atmosphere_Create( atm_k      , &
                             n_layers   , &
                             n_absorbers, &
                             n_clouds   , &
                             n_aerosols   )
! Check they were created successfully
IF ( ANY(.NOT. CRTM_Atmosphere_Associated( atm_k )) ) THEN
  handle error...
END IF
```

Note that for the ODAS algorithm the allowed number of absorbers is *at most* two: that of H_2O and O_3. For the ODPS algorithm CO_2 can also be specified. For the infrared hyperspectral sensors (AIRS, IASI, and CrIS) the trace gases CH_4, N_2O, and CO can also be specified as absorbers.

4.5.2 Allocation of the RTSolution structure

To return additional information used in the radiative transfer calculations, such as upwelling radiance and layer optical depth profiles, the RTSolution structure must be allocated to the number of atmospheric layers used,

```
! Allocate the RTSolution structure
CALL CRTM_RTSolution_Create( rts     , &
                             n_layers  )
! Check they were created successfully
IF ( ANY(.NOT. CRTM_RTSolution_Associated( rts )) ) THEN
  handle error...
END IF
```

Note that internal checks are performed in the CRTM to determine if the RTSolution structure has been allocated before its array components are accessed. Thus, if the additional information is not required, the RTSolution structure does not need to be allocated. Also, the extra information returned is only applicable to the forward model, not any of the tangent-linear, adjoint, or K-matrix models.

4.5.3 *Allocation of the Options structure*

If user-supplied surface emissivity data is to be used, then the options structure must first be allocated to the necessary number of channels:

```
! Allocate the options structures
CALL CRTM_Options_Create( opt        , &
                          n_channels  )
! Check they were created successfully
IF ( ANY(.NOT. CRTM_Options_Associated( opt )) ) THEN
  handle error...
END IF
```

If no emissivities are to be input, the options structure does not need to be allocated.

4.6 Fill the CRTM input structures with data

This step simply entails filling the input `Atmosphere` (including `Cloud` and `Aerosol`), `Surface`, `Geometry`, and, if used, `Options` structures with the required information. Sound simple? Read on...

4.6.1 *Filling the Atmosphere structure with data*

The elements of the `Atmosphere` structure, and their description, are shown in table 4.2. The modifiers such as "(1:J)" and "(1:nA)" are an indication of the allocatable range of the components. Similar descriptions of the `Cloud` and `Aerosol` structures are show in tables 4.3 and 4.4 respectively.

Table 4.2: CRTM `Atmosphere` structure component description.

Component	Description	Units	Default value
`n_Layers`	Number of atmospheric layers, K	N/A	N/A
`n_Absorbers`	Number of gaseous absorbers, J	N/A	N/A
`n_Clouds`	Number of clouds, nC	N/A	N/A
`n_Aerosols`	Number of aerosol species, nA	N/A	N/A
`Climatology`	Climatology model associated with the profile. See table 4.5.	N/A	`US_STANDARD_ATMOSPHERE`
`Absorber_ID`(1:J)	Absorber identifiers. See table 4.6.	N/A	N/A
`Absorber_Units`(1:J)	Absorber concentration unit identifiers. See table 4.7.	N/A	N/A
`Level_Pressure`(0:K)	*Level* pressure profile	hPa	N/A
`Pressure`(1:K)	Layer pressure profile	hPa	N/A
`Temperature`(1:K)	Layer temperature profile	Kelvin	N/A
`Absorber`(1:K,1:J)	Layer absorber concentraton profiles	Variable	N/A
`Cloud`(1:nC)	Clouds associated with the profile	N/A	N/A
`Aerosol`(1:nA)	Aerosol species associatedwith the profile	N/A	N/A

Some issues to mention with populating the `Atmosphere` structure

Table 4.3: CRTM `Cloud` structure component description.

Component	Description	Units	Default value
`n_Layers`	Number of atmospheric layers, K	N/A	N/A
`Type`	The supported cloud type. See table 4.8.	N/A	`INVALID_CLOUD`
`Effective_Radius(1:`K`)`	Cloud particle effective radius profile	μm	N/A
`Water_Content(1:`K`)`	Cloud water content profile	kg.m^{-2}	N/A

Table 4.4: CRTM `Aerosol` structure component description.

Component	Description	Units	Default value
`n_Layers`	Number of atmospheric layers, K	N/A	N/A
`Type`	The supported aerosol type. See tables 4.9 and 4.10.	N/A	`INVALID_AEROSOL`
`Effective_Radius(1:`K`)`	Aerosol particle effective radius profile	μm	N/A
`Effective_Variance(1:`K`)`	Aerosol particle radius standard deviation profile. Required only if using CMAQ aerosol coefficient look-up table.	μm	N/A
`Concentration(1:`K`)`	Aerosol concentration profile	kg.m^{-2}	N/A

Table 4.5: CRTM `Atmosphere` structure valid `Climatology` definitions. The same set as defined for LBLRTM is used.

Climatology Type	Parameter
Tropical	`TROPICAL`
Midlatitude summer	`MIDLATITUDE_SUMMER`
Midlatitude winter	`MIDLATITUDE_WINTER`
Subarctic summer	`SUBARCTIC_SUMMER`
Subarctic winter	`SUBARCTIC_WINTER`
U.S. Standard Atmosphere	`US_STANDARD_ATMOSPHERE`

Table 4.6: CRTM `Atmosphere` structure valid `Absorber_ID` definitions. The same molecule set as defined for HITRAN is used.

Molecule	Parameter	Molecule	Parameter	Molecule	Parameter
H_2O	`H2O_ID`	OH	`OH_ID`	H_2O_2	`H2O2_ID`
CO_2	`CO2_ID`	HF	`HF_ID`	C_2H_2	`C2H2_ID`
O_3	`O3_ID`	HCl	`HCl_ID`	C_2H_6	`C2H6_ID`
N_2O	`N2O_ID`	HBr	`HBr_ID`	PH_3	`PH3_ID`
CO	`CO_ID`	HI	`HI_ID`	COF_2	`COF2_ID`
CH_4	`CH4_ID`	ClO	`ClO_ID`	SF_6	`SF6_ID`
O_2	`O2_ID`	OCS	`OCS_ID`	H_2S	`H2S_ID`
NO	`NO_ID`	H_2CO	`H2CO_ID`	HCOOH	`HCOOH_ID`
SO_2	`SO2_ID`	HOCl	`HOCl_ID`		
NO_2	`NO2_ID`	N_2	`N2_ID`		
NH_3	`NH3_ID`	HCN	`HCN_ID`		
HNO_3	`HNO3_ID`	CH_3l	`CH3l_ID`		

Table 4.7: CRTM `Atmosphere` structure valid `Absorber_Units` definitions. The same set as defined for LBLRTM is used.

Absorber Units	Parameter
Volume mixing ratio, ppmv	`VOLUME_MIXING_RATIO_UNITS`
Number density, cm^{-3}	`NUMBER_DENSITY_UNITS`
Mass mixing ratio, g/kg	`MASS_MIXING_RATIO_UNITS`
Mass density, g.m^{-3}	`MASS_DENSITY_UNITS`
Partial pressure, hPa	`PARTIAL_PRESSURE_UNITS`
Dewpoint temperature, K **(H_2O ONLY)**	`DEWPOINT_TEMPERATURE_K_UNITS`
Dewpoint temperature, C **(H_2O ONLY)**	`DEWPOINT_TEMPERATURE_C_UNITS`
Relative humidity, % **(H_2O ONLY)**	`RELATIVE_HUMIDITY_UNITS`
Specific amount, g/g	`SPECIFIC_AMOUNT_UNITS`
Integrated path, mm	`INTEGRATED_PATH_UNITS`

Table 4.8: CRTM `Cloud` structure valid `Type` definitions.

Cloud Type	Parameter
Water	`WATER_CLOUD`
Ice	`ICE_CLOUD`
Rain	`RAIN_CLOUD`
Snow	`SNOW_CLOUD`
Graupel	`GRAUPEL_CLOUD`
Hail	`HAIL_CLOUD`

Table 4.9: CRTM `Aerosol` structure valid `Type` definitions and effective radii, based on the GOCART model. SSAM ≡ Sea Salt Accumulation Mode, SSCM ≡ Sea Salt Coarse Mode.

Aerosol Type	Parameter	r_{eff} Range (μm)
Dust	`DUST_AEROSOL`	0.01 - 8
Sea salt SSAM	`SEASALT_SSAM_AEROSOL`	0.3 - 1.45
Sea salt SSCM1	`SEASALT_SSCM1_AEROSOL`	1.0 - 4.8
Sea salt SSCM2	`SEASALT_SSCM2_AEROSOL`	3.25 - 17.3
Sea salt SSCM3	`SEASALT_SSCM3_AEROSOL`	7.5 - 89
Organic carbon	`ORGANIC_CARBON_AEROSOL`	0.09 - 0.21
Black carbon	`BLACK_CARBON_AEROSOL`	0.036 - 0.074
Sulfate	`SULFATE_AEROSOL`	0.24 - 0.8

Table 4.10: CRTM `Aerosol` structure valid `Type` definitions, effective radii (0.01 -7.5 μm), and radius standard deviations (1.05 - 2.5 μm), based on the aerosol size specification in CMAQ model.

Aerosol Type	**Index Parameter**
Dust	1
Soot	2
Water soluble	3
Sulfate	4
Sea salt	5
Water	6
Insoluble	7
Dust-like	8

- In the CRTM, all profile layering is from top-of-atmosphere (TOA) to surface (SFC). So, for an atmospheric profile layered as $k = 1, 2, ..., K$, layer 1 is the TOA layer and layer K is the SFC layer.
- *Both* the level and layer pressure profiles must be specified.
- The absorber profile data units *must* be mass mixing ratio for water vapour and volume mixing ratio (ppmv) for other absorbers. The `Absorber_Units` component is not yet utilized to allow conversion of different user-supplied concentration units.
- The `Absorber_Id` array must be set to the correct absorber identifiers (see table 4.6) to allow the software to find a particular absorber. There is no necessary order in specifying the concentration profiles for different gaseous absorbers.

An example of assigning values to an `Atmosphere` structure is shown below, adapted and abridged from one of the test/example programs supplied with the CRTM,

```
! ...Profile and absorber definitions
atm(1)%Climatology         = US_STANDARD_ATMOSPHERE
atm(1)%Absorber_Id(1:2)    = (/ H2O_ID                 , O3_ID /)
atm(1)%Absorber_Units(1:2) = (/ MASS_MIXING_RATIO_UNITS, VOLUME_MIXING_RATIO_UNITS /)

! ...Profile data
atm(1)%Level_Pressure = &
(/ 0.714_fp, 0.975_fp, .... , 1070.917_fp, 1100.000_fp /)

atm(1)%Pressure = &
(/ 0.838_fp, 1.129_fp, .... , 1056.510_fp, 1085.394_fp /)

atm(1)%Temperature = &
(/ 256.186_fp, 252.608_fp, .... , 273.356_fp, 273.356_fp /)

atm(1)%Absorber(:,1) = &
(/ 4.187e-03_fp, 4.401e-03_fp, .... , 3.172_fp, 3.087_fp /)

atm(1)%Absorber(:,2) = &
(/ 3.035_fp, 3.943_fp, .... , 1.428e-02_fp, 1.428e-02_fp /)

! ...Load CO2 absorber data if there are three absorrbers
IF ( atm(1)%n_Absorbers > 2 ) THEN
  atm(1)%Absorber_Id(3)    = CO2_ID
```

```
    atm(1)%Absorber_Units(3) = VOLUME_MIXING_RATIO_UNITS
    atm(1)%Absorber(:,3)     = 380.0_fp
  END IF
```

The allowable definitions of the `Climatology`, `Absorber_Id`, and `Absorber_Units` components are shown in tables 4.5, 4.6, and 4.7 respectively. Even though the `Absorber_Units` component is not currently used in the v2.1 CRTM it is recommended that it still be set in `Atmosphere` structures to accommodate future CRTM versions that do utilise it.

The cloud and aerosol data for a given atmospheric profile are specified via the contained `Cloud` and `Aerosol` structure arrays. Continuing with the example assignment, we could do the following for our single cloud,

```
  INTEGER :: k1, k2
  ....
  ! Assign cloud data
  k1 = 55  ! Begin cloud layer
  k2 = 62  ! End cloud layer
  atm(1)%Cloud(1)%Type = WATER_CLOUD

  atm(1)%Cloud(1)%Effective_Radius(k1:k2) = &
    (/ 20.14_fp, 19.75_fp, .... , 12.49_fp, 11.17_fp /)  ! microns
  atm(1)%Cloud(1)%Water_Content(k1:k2)    = &
    (/ 5.09_fp, 3.027_fp, .... , 1.56_fp, 2.01_fp /)     ! kg/m^2
```

and for our multiple aerosols with the default GOCART look-up table:

```
  ! Assign aerosol data
  ! ...First aerosol
  k1 = 21  ! Begin aerosol layer
  k2 = 64  ! End aerosol layer
  atm(1)%Aerosol(1)%Type = DUST_AEROSOL

  atm(1)%Aerosol(1)%Effective_Radius(k1:k2) = &
    (/7.340409e-16_fp, 1.037097e-15_fp, .... , 2.971053e-03_fp, 8.218245e-04_fp/) ! microns
  atm(1)%Aerosol(1)%Concentration(k1:k2) = &
    (/2.458105E-18_fp, 1.983430E-16_fp, .... , 7.418821E-05_fp, 1.172680E-05_fp/) ! kg/m^2

  ! ...Second aerosol
  k1 = 48  ! Begin aerosol layer
  k2 = 64  ! End aerosol layer
  atm(1)%Aerosol(2)%Type = SULFATE_AEROSOL

  atm(1)%Aerosol(2)%Effective_Radius(k1:k2) = &
    (/3.060238E-01_fp, 3.652677E-01_fp, .... , 5.570077E-01_fp, 3.828734E-01_fp/) ! microns
  atm(1)%Aerosol(2)%Concentration(k1:k2) = &
    (/2.609907E-05_fp, 2.031620E-05_fp, .... , 1.095622E-04_fp, 7.116027E-05_fp/) ! kg/m^2
```

or if using the CMAQ look-up table:

```
  ! Assign aerosol data
  ! ...First aerosol
  k1 = 21  ! Begin aerosol layer
  k2 = 64  ! End aerosol layer
  atm(1)%Aerosol(1)%Type = 1 !dust

  atm(1)%Aerosol(1)%Effective_Radius(k1:k2) = &
    (/7.340409E-16_fp, 1.037097E-15_fp, .... , 2.971053E-03_fp, 8.218245E-04_fp/) ! microns
  atm(1)%Aerosol(1)%Effective_Variance(k1:k2) = &
    (/1.800000E+00_fp, 1.400000E+00_fp, .... , 2.200000E+00_fp, 1.0500000E+00_fp,/) ! microns
  atm(1)%Aerosol(1)%Concentration(k1:k2) = &
    (/2.458105E-18_fp, 1.983430E-16_fp, .... , 7.418821E-05_fp, 1.172680E-05_fp/) ! kg/m^2

  ! ...Second aerosol
  k1 = 48  ! Begin aerosol layer
  k2 = 64  ! End aerosol layer
  atm(1)%Aerosol(2)%Type = 4

  atm(1)%Aerosol(2)%Effective_Radius(k1:k2) = &
    (/3.060238E-01_fp, 3.652677E-01_fp, .... , 5.570077E-01_fp, 3.828734E-01_fp/) ! microns
  atm(1)%Aerosol(2)%Effective_Variance(k1:k2) = &
    (/1.300000E+00_fp, 1.100000E+00_fp, .... , 2.500000E+00_fp, 1.0800000E+00_fp,/) ! microns
  atm(1)%Aerosol(2)%Concentration(k1:k2) = &
    (/2.609907E-05_fp, 2.031620E-05_fp, .... , 1.095622E-04_fp, 7.116027E-05_fp/) ! kg/m^2
```

The allowable definitions of the cloud and aerosol type components are shown in tables 4.8, 4.9, and 4.10

One final note regarding clouds and aerosols (although we'll use just clouds as an example here). Let's assume for a given atmospheric profile we have cloud data specifying a water cloud near the surface (say from layers 60–64) and the same type of cloud higher in the troposphere (say from layers 52–57). You could define this as a *single* cloud like so,

```
  ! Assign multiple level cloud data in a single cloud structure
  atm(1)%Cloud(1)%Type = WATER_CLOUD
  k1 = 52  ! Begin cloud layer 1
  k2 = 57  ! End cloud layer 1
  atm(1)%Cloud(1)%Effective_Radius(k1:k2) = ....
  atm(1)%Cloud(1)%Water_Content(k1:k2)    = ....
  k1 = 60  ! Begin cloud layer 2
  k2 = 64  ! End cloud layer 2
  atm(1)%Cloud(1)%Effective_Radius(k1:k2) = ....
  atm(1)%Cloud(1)%Water_Content(k1:k2)    = ....
```

or you could define it in *separate* cloud structures like so,

```
  ! Assign multiple level cloud data in separate cloud structures
  k1 = 52  ! Begin cloud 1 layer
  k2 = 57  ! End cloud 1 layer
  atm(1)%Cloud(1)%Type = WATER_CLOUD
  atm(1)%Cloud(1)%Effective_Radius(k1:k2) = ....
  atm(1)%Cloud(1)%Water_Content(k1:k2)    = ....
  k1 = 60  ! Begin cloud 2 layer
  k2 = 64  ! End cloud 2 layer
  atm(1)%Cloud(2)%Type = WATER_CLOUD
  atm(1)%Cloud(2)%Effective_Radius(k1:k2) = ....
  atm(1)%Cloud(2)%Water_Content(k1:k2)    = ....
```

That is, for the same type of cloud there is no difference between specifying multiple layers in a single structure, or specifying multiple structures that contain a single layer. The two "styles" of definition are equivalent. Similarly for aerosols.

4.6.2 Filling the Surface structure with data

The `Surface` structure is designed around four main surface types: Land, Water, Snow, and Ice. As you can see in table 4.11, for each of these main surface types there are components that define the surface characteristics. This division of surface types and the required surface characteristics are based upon the way surface emissivity and reflectivity models have been constructed in the past. It is also complicated by the fact that for the different spectral regions that the CRTM models – infrared, microwave, and visible – the surface emissivity and reflectivity modeling has to be handled differently as different processes are more important in different spectral regions. As such, it is important that users understand what needs to set in a `Surface` structure for a given surface type and spectral region. We will also assume that a `Surface` structure corresponds to a sensor field-of-view (FOV).

Table 4.11: CRTM `Surface` structure component description.

Component	Description	Units	Default value
`Land_Coverage`	Fraction of the FOV that is land surface	N/A	0.0
`Water_Coverage`	Fraction of the FOV that is water surface	N/A	0.0
`Snow_Coverage`	Fraction of the FOV that is snow surface	N/A	0.0
`Ice_Coverage`	Fraction of the FOV that is ice surface	N/A	0.0
`Land_Type`	Land surface type	N/A	1
`Land_Temperature`	Land surface temperature	Kelvin	283.0
`Soil_Moisture_Content`	Volumetric water content of the soil	$g.cm^{-3}$	0.05
`Canopy_Water_Content`	Gravimetric water content of the canopy	$g.cm^{-3}$	0.05
`Vegetation_Fraction`	Vegetation fraction of the surface	%	0.3
`Soil_Temperature`	Soil temperature	Kelvin	283.0
`LAI`	Leaf area index	m^2/m^2	3.5
`Soil_Type`	Soil type	N/A	1
`Vegetation_Type`	Vegetation type	N/A	1
`Water_Type`	Water surface type	N/A	1
`Water_Temperature`	Water surface temperature	Kelvin	283.0
`Wind_Speed`	Surface wind speed	$m.s^{-1}$	5.0
`Wind_Direction`	Surface wind direction	deg. E from N	0.0
`Salinity`	Water salinity	‰	33.0
`Snow_Type`	Snow surface type	N/A	1
`Snow_Temperature`	Snow surface temperature	Kelvin	263.0
`Snow_Depth`	Snow depth	mm	50.0
`Snow_Density`	Snow density	$g.m^{-3}$	0.2
`Snow_Grain_Size`	Snow grain size	mm	2.0
`Ice_Type`	Ice surface type	N/A	1
`Ice_Temperature`	Ice surface temperature	Kelvin	263.0
`Ice_Thickness`	Thickness of ice	mm	10.0
`Ice_Density`	Density of ice	$g.m^{-3}$	0.9
`Ice_Roughness`	Measure of the surface roughness of the ice	N/A	0.0
`SensorData`	Satellite sensor data required for empirical microwave snow and ice emissivity algorithms	N/A	N/A

The specification of the actual physical surface characteristics in a `Surface` structure (e.g. temperature, wind speed, soil moisture, etc) is relatively straightforward and won't be covered in detail here. What we'll look into are those items that are specific (or peculiar?) to the CRTM implementation of emissivity and reflectivity models and how they influence the definition of the `Surface` structure.

The first thing to address are the coverage fractions. The CRTM allows the specification of a combination of the main surface types. Let's say we have a FOV that consists of 10% land, 50% water, 25% snow, and 15% ice. The specification of these fractions in the surface structure would look like so:

```
! Assign main surface type coverage fractions
sfc(1)%Land_Coverage  = 0.1_fp
sfc(1)%Water_Coverage = 0.5_fp
sfc(1)%Snow_Coverage  = 0.25_fp
sfc(1)%Ice_Coverage   = 0.15_fp
```

Whatever the surface coverage combination, the sum of the coverage fractions *must* add up to 1.0. Otherwise the CRTM will issue an error message and return with a `FAILURE` error status.

Now we'll look at the specification of the subtypes of the main surface types, with a particular focus on the land surface subtypes. Table 4.12 shows the number of valid surface subtypes available for the different surface and spectral categories in v2.1. As can be seen for land surfaces, some care is required to ensure correct specification of the subtype specification(s). The situation is much simpler for the other surface types (water, snow and ice) and, for microwave sensors, is simplified further since no subtype even need be defined due to the surface optics models used.

Table 4.12: Number of valid surface types available for the different surface and spectral categories. [a]Same IR and VIS reflectivity source, NPOESS. [b]Surface type reflectivities mapped from NPOESS classification. [c]Different land classifications for IR and VIS defined at CRTM initialisation. [d]These are specified separately from the generic surface type in the input `Surface`structure and are used to index arrays containing various physical quantities for the soil/vegetation type – **both** must be specified.

Spectral category	**Land**[c]	**Water**	**Snow**	**Ice**
Infrared	NPOESS(20)[a] USGS(27)[a,b] IGBP(20)[a,b]	CRTM(1)	CRTM(2)[a]	CRTM(1)[a]
Microwave	Soil type(9)[d] Vegetation type(13)[4]	Parameterized physical model	Empirical model	Empirical model
Visible	NPOESS(20)[a] USGS(27)[a,b] IGBP(20)[a,b]	CRTM(1)	CRTM(2)[a]	CRTM(1)[a]

Land surface subtypes for infrared and visible sensors

In the v2.0.x CRTM releases, there was only one allowable set of surface subtypes allowed. For the land surface type in the infrared and visible spectral regions, that was the NPOESS[4] set. However, different land surface classification schemes (USGS[5] and IGBP[6]) were being used in various applications that called the CRTM, requiring users to generate a mapping from their surface classification scheme to that of the CRTM (i.e. the

[4]National Polar-orbiting Operational Environmental Satellite System. Now called the Joint Polar Satellite System, or JPSS.
[5]U.S. Geological Survey
[6]International Geosphere-Biosphere Programme

NPOESS classification). In an effort to simplify the use of different land subtype classification systems with the CRTM, separate datafiles contining the reflectivity data for the different classification schemes are now provided (see section 4.3 regarding the use of these data files during CRTM initialisation). Thus you need only initialise the CRTM with the data files for your land subtype classification scheme of choice to use that scheme.

The downside of this change is that parameterised values of the surface subtypes can no longer be used since, depending on how the CRTM was initialised, the same parameterised value can be used as an index for different classification schemes – in which the index may not exist, or – even worse –refer to a different land subtype giving a plausibly wrong result. Thus, you should study the allowable subtype index values for the NPOESS, USGS, and IGBP classifications schemes shown in tables 4.13, 4.14, and 4.15 respectively to ensure you are selecting the correct land subtype.

Table 4.13: Surface type names and their index value for the NPOESS land surface classification scheme. Applicable for infrared and visible spectral regions only.

NPOESS Classification Scheme	
Surface Type Name	Classification Index
compacted soil	1
tilled soil	2
sand	3
rock	4
irrigated low vegetation	5
meadow grass	6
scrub	7
broadleaf forest	8
pine forest	9
tundra	10
grass soil	11
broadleaf pine forest	12
grass scrub	13
soil grass scrub	14
urban concrete	15
pine brush	16
broadleaf brush	17
wet soil	18
scrub soil	19
broadleaf70 pine30	20

As an example, if the CRTM was initialised with the NPOESS classification data and the surface type was considered "urban", consultation of table 4.13 would yield the following assignment,

```
! Assign urban land surface subtype for NPOESS classification
sfc(1)%Land_Type = 15
```

Similarly, if the CRTM was initialised with the USGS classification data, the same assignment would be (see table 4.14)

```
! Assign urban land surface subtype for USGS classification
sfc(1)%Land_Type = 1
```

For completeness, here is the same for the IGBP classification (see table 4.15)

```
! Assign urban land surface subtype for IGBP classification
sfc(1)%Land_Type = 13
```

Table 4.14: Surface type names and their index value for the USGS land surface classification scheme. Note that the "non-land" surface types in the context of the CRTM (water, snow, or ice at indices 16 and 24) are still included but are empty entries in the reflectivity database. Applicable for infrared and visible spectral regions only.

USGS Classification Scheme	
Surface Type Name	Classification Index
urban and built-up land	1
dryland cropland and pasture	2
irrigated cropland and pasture	3
mixed dryland/irrigated cropland and pasture	4
cropland/grassland mosaic	5
cropland/woodland mosaic	6
grassland	7
shrubland	8
mixed shrubland/grassland	9
savanna	10
deciduous broadleaf forest	11
deciduous needleleaf forest	12
evergreen broadleaf forest	13
evergreen needleleaf forest	14
mixed forest	15
water bodies **(empty)**	16
herbaceous wetland	17
wooded wetland	18
barren or sparsely vegetated	19
herbaceous tundra	20
wooded tundra	21
mixed tundra	22
bare ground tundra	23
snow or ice **(empty)**	24
playa	25
lava	26
white sand	27

Table 4.15: Surface type names and their index value for the IGBP land surface classification scheme. Note that the "non-land" surface types in the context of the CRTM (water, snow, or ice at indices 15 and 17) are still included but are empty entries in the reflectivity database. Applicable for infrared and visible spectral regions only.

IGBP Classification Scheme	
Surface Type Name	Classification Index
evergreen needleleaf forest	1
evergreen broadleaf forest	2
deciduous needleleaf forest	3
deciduous broadleaf forest	4
mixed forests	5
closed shrublands	6
open shrublands	7
woody savannas	8
savannas	9
grasslands	10
permanent wetlands	11
croplands	12
urban and built-up	13
cropland/natural vegetation mosaic	14
snow and ice **(empty)**	15
barren or sparsely vegetated	16
water **(empty)**	17
wooded tundra	18
mixed tundra	19
bare ground tundra	20

Land surface subtypes for microwave sensors

For the land surface/microwave spectral region case, the situation is a little different. The emissivity model uses specification of the soil and vegetation type to drive the calculation; that is, *both* must be specified. The valid soil and vegetation types in this case are defined by their definitions in the NCEP Global Forecast System (GFS) and are shown in tables 4.16 and 4.17 respectively.

Table 4.16: Soil type textures and descriptions, along with their index value for the GFS classification scheme. Applicable for the microwave spectral regions only.

GFS Soil Type Classification Scheme		
Texture	Description	Classification Index
coarse	loamy sand	1
medium	silty clay loam	2
fine	light clay	3
coarse-medium	sandy loam	4
coarse-fine	sandy clay	5
medium-fine	clay loam	6
coarse-med-fine	sandy clay loam	7
organic	farmland	8
glacial land ice	ice over land	9

Table 4.17: Vegetation type names and their index value for the GFS classification scheme. Applicable for the microwave spectral regions only.

GFS Vegetation Type Classification Scheme	
Vegetation Type	Classification Index
broadleaf-evergreen (tropical forest)	1
broad-deciduous trees	2
broadleaf and needleleaf trees (mixed forest)	3
needleleaf-evergreen trees	4
needleleaf-deciduous trees (larch)	5
broadleaf trees with ground cover (savanna)	6
ground cover only (perennial)	7
broad leaf shrubs w/ ground cover	8
broadleaf shrubs with bare soil	9
dwarf trees & shrubs w/ground cover (tundra)	10
bare soil	11
cultivations	12
glacial	13

An example of assigning these two types for use with the microwave land emissivity model would be,

```
! Assign farmland soil and vegetation types for
! the microwave land emissivity model
sfc(1)%Soil_Type       = 8
sfc(1)%Vegetation_Type = 12
```

Water, snow, and ice surface subtypes for infrared and visible sensors

The situation for the water, snow, and ice surface subtypes in the infrared and visible spectral regions is much simpler. There are only at most two variations for these main surface types and, for ice, there is only one. Table 4.18 lists the available subtype indices in these cases.

Table 4.18: Water, snow, and ice surface subtypes and their index value. Applicable for infrared and visible spectral regions only.

IR/VIS Water, Snow, and Ice Classification Scheme		
Surface Type	Description	Classification Index
Water	sea water	1
Snow	old snow	1
	new snow	2
Ice	new ice	1

An example of assigning these types for use with the infrared or visible water, snow, or ice emissivity models would be,

```
! Assign water, snow and ice types for the
! infrared and visible emissivity models
sfc(1)%Water_Type = 1  ! Sea water
sfc(1)%Snow_Type  = 2  ! New snow
sfc(1)%Ice_Type   = 1  ! New ice
```

Water, snow, and ice surface subtypes for microwave sensors

The specification of the water, snow, and ice surface subtypes is not necessary in the microwave spectral region. Consultation of table 4.12 reveals why: for the water case, the emissivity model is a parameterised physical model and for the snow and ice surfaces the CRTM uses empirical models. In fact, in the latter case, the snow and ice subtypes are actually *output* from the models.

Specification of SensorData for microwave snow and ice emissivity models

Recall from table 4.12 that the snow and ice emissivity models for microwave sensors are empirical, i.e. they use input sensor measurements to estimate the snow and/or ice emissivities for particular sensors[7]. To supply the brightness temperatures used by the empirical emissivity model, the `SensorData` structure component of the main `Surface` structure is used. The components of the `SensorData` structure are shown in table 4.19 where the modifier "(1:L)" is the indication of the allocatable range of those components.

The values of the WMO satellite and sensor identifiers are those defined in the WMO Common Code Tables C-5 and C-8 respectively.[8] The WMO sensor identifier is used to select the particular sensor algorithm so you should endeavour to correctly specify it in the `SensorData` structure. If an unrecognised WMO identifier is encountered then, for snow surfaces, a default physical model is used. For ice surfaces the default is to use a fixed emissivity of 0.92.

The sensors for which empirical snow and ice emissivity models exist, along with their WMO sensor identifiers, are shown in table 4.20

[7]Supplied by NESDIS/STAR for use in the CRTM

[8]See http://www.wmo.int/pages/prog/www/WMOCodes/WMO306_vI2/VolumeI.2.html to access the WMO Part C Common Code Tables in various languages.

Table 4.19: CRTM SensorData structure component description.

Component	Description	Units	Default value
n_Channels	Number of sensor channels, L	N/A	0
Sensor_Id	The sensor id	N/A	empty string
WMO_Satellite_Id	The WMO satellite Id	N/A	INVALID_WMO_SATELLITE_ID
WMO_Sensor_Id	The WMO sensor Id	N/A	INVALID_WMO_SENSOR_ID
Sensor_Channel(1:L)	The channel numbers	N/A	N/A
Tb(1:L)	The brightness temperature measurements for each channel	Kelvin	N/A

Table 4.20: Microwave sensors and their associated WMO sensor identifiers for which the CRTM has empirical snow and ice emissivity models.

Sensor	WMO Sensor Id	Sensor	WMO Sensor Id	Sensor	WMO Sensor Id
AMSR-E	345	AMSU-B	574	SSMIS	908
AMSU-A	570	MHS	203	SSM/I	905

Using the sensor-loop example of section 4.4, an example of specifying the brightness temperature data for the NOAA-19 AMSU-A to use for its empirical snow or ice emissivity module would be,

```
INTEGER :: m, n
....
Sensor_Loop: DO n = 1, n_sensors
  ....
  ! Get the number of channels for the SensorData structure for current sensor
  n_channels = chinfo(n)%n_Channels
  ....
  ! Allocate the SensorData structure for this sensor to use its empirical emissivity model
  CALL CRTM_SensorData_Create( sfc%SensorData, &
                               n_channels   )
  ! Check they were created successfully
  IF ( ANY(.NOT. CRTM_SensorData_Associated( sfc%SensorData )) ) THEN
    handle error...
  END IF
  ....
  ! Specify the sensor identifiers for all the profiles
  sfc%SensorData%Sensor_Id        = 'amsua_n19'
  sfc%SensorData%WMO_Satellite_Id = 223  ! From Common Code Table C-5
  sfc%SensorData%WMO_Sensor_Id    = 570  ! From Common Code Table C-8
  ....
  ! Specify the brightness temperature data for the various profiles/FOVs in the Sensordata structure
  Profile_Loop: DO m = 1, n_profiles
    sfc(m)%SensorData%Tb = ...assign appropriate data...
  END DO Profile_Loop
  ....
END DO Sensor_Loop
```

Note the use of the "`n_channels = chinfo(n)%n_Channels`" statement. The empirical snow and ice models do not recognise the channel subsetting feature implemented in the CRTM (see section 4.3.6) and thus, to correctly index the brightness temperature array, *all* of a particular sensor's channels must be specified.

4.6.3 *Filling the Geometry structure with data*

Descriptions of the components of the `Geometry` structure are shown in table 4.21. They are relatively self-explanatory, but visualisations of some of the angle descriptions are shown in figures 4.1 to 4.5.

The one note that should be made is that the sensor zenith (θ_Z) and sensor scan (θ_S) angles should be consistent. They are related by equation:

$$\frac{\sin\theta_Z}{R+h} = \frac{\sin\theta_S}{R} \tag{4.1}$$

with the quantity definitions shown in figure 4.6

Table 4.21: CRTM `Geometry` structure component description.

Component	Description	Units	Default value
`iFOV`	The scan line FOV index	N/A	0
`Longitude`	Earth longitude for FOV	deg. E (0→360)	0.0
`Latitude`	Earth latitude for FOV	deg. N (-90→+90)	0.0
`Surface_Altitude`	Altitude of the Earth's surface at the specified lon/lat location	metres (m)	0.0
`Sensor_Scan_Angle`	The sensor scan angle from nadir. See fig.4.1	degrees	0.0
`Sensor_Zenith_Angle`	The sensor zenith angle of the FOV. See fig.4.2	degrees	0.0
`Sensor_Azimuth_Angle`	The sensor azimuth angle is the angle subtended by the horizontal projection of a direct line from the satellite to the FOV and the North-South axis measured clockwise from North. See fig.4.3	deg. from N	999.9
`Source_Zenith_Angle`	The source zenith angle. The source is typically the Sun (IR/VIS) or Moon (MW/VIS) [only solar source valid in current release] See fig.4.4	degrees	100.0
`Source_Azimuth_Angle`	The source azimuth angle is the angle subtended by the horizontal projection of a direct line from the source to the FOV and the North-South axis measured clockwise from North. See fig.4.5	deg. from N	0.0
`Flux_Zenith_Angle`	The zenith angle used to approximate downwelling flux transmissivity. If not set, the default value is that of the diffusivity approximation, such that $\sec(F) = 5/3$. Maximum allowed value is determined from $\sec(F) = 9/4$	degrees	$cos^{-1}(3/5)$
`Year`	The year in 4-digit format	N/A	2001
`Month`	The month of year (1-12)	N/A	1
`Day`	The day of month (1-28/29/30/31)	N/A	1

4.6.4 *Filling the Options structure with data*

Descriptions of the components of the `Options` structure are shown in table 4.22. If the `Options` structure is not even specified in the CRTM function call (since it is itself an optional argument), the default values specified in table 4.22 are used.

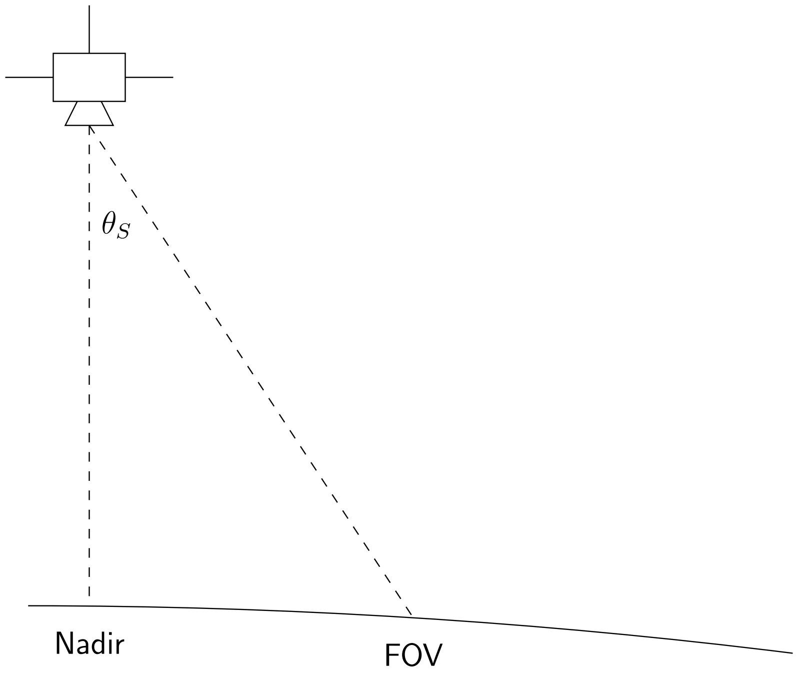

Figure 4.1: Definition of `Geometry` sensor scan angle component.

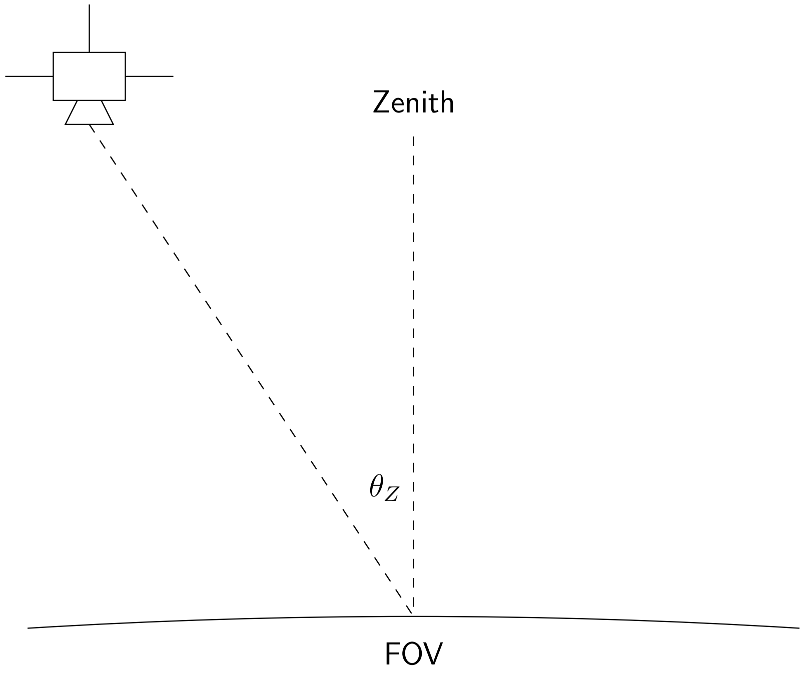

Figure 4.2: Definition of `Geometry` sensor zenith angle component.

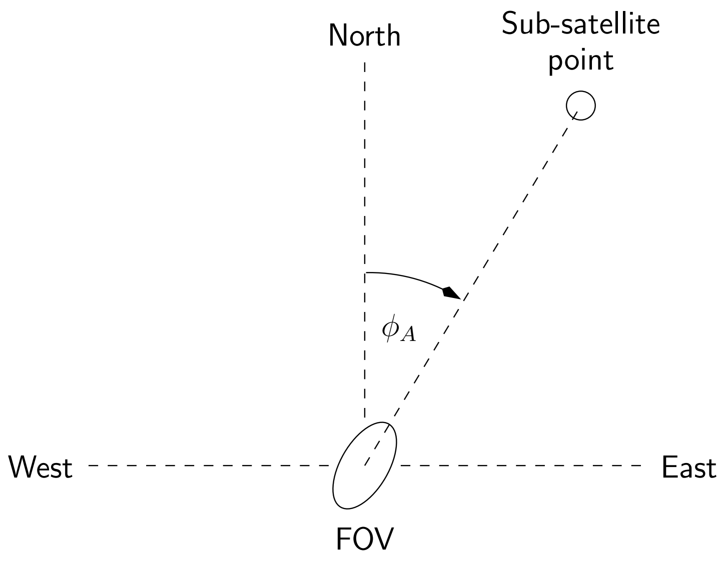

Figure 4.3: Definition of `Geometry` sensor azimuth angle component.

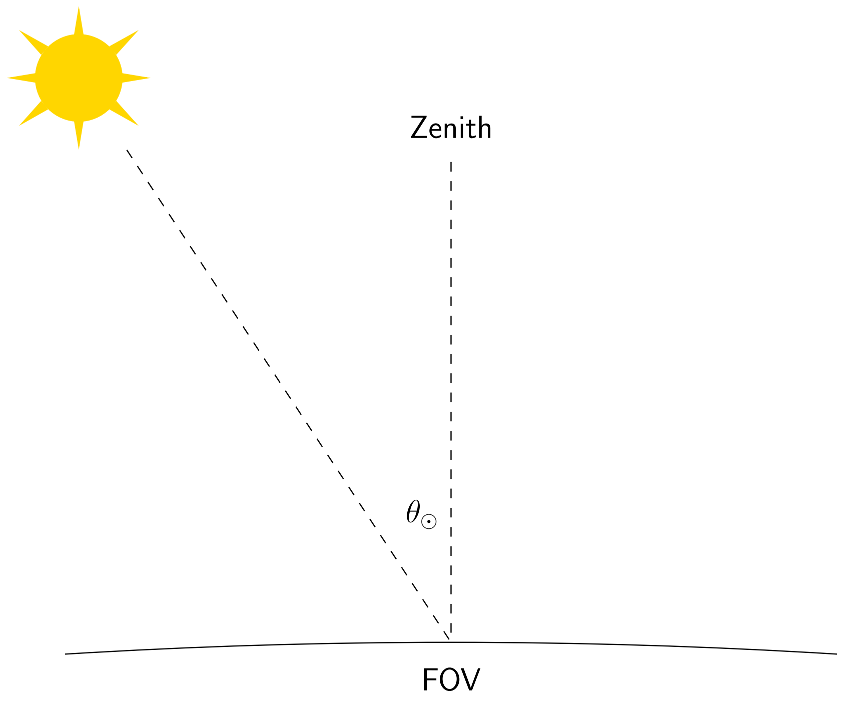

Figure 4.4: Definition of `Geometry` source zenith angle component.

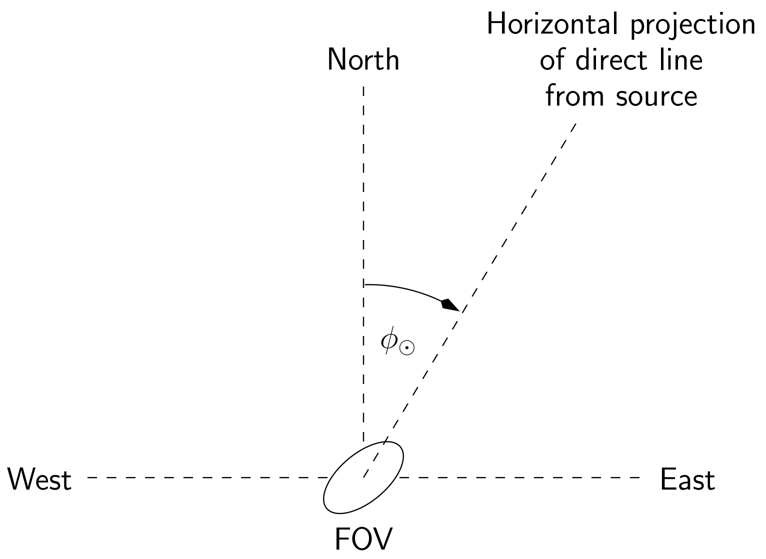

Figure 4.5: Definition of `Geometry` source azimuth angle component.

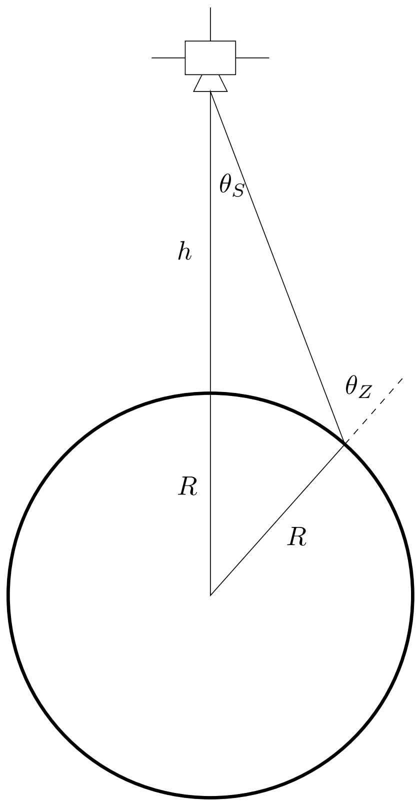

Figure 4.6: Geometry definitions for equation 4.1.

For the allocatable components, the modifier "(1:L)" is an indication of the range of the array indices. Note that if user-defined surface emissivities are not going to be used there is no need to allocate the internals of the `Options` structure.

Table 4.22: CRTM `Options` structure component description

Component	Description	Units	Default value
`Check_Input`	Logical switch to enable or disable input data checking. If: `.FALSE.`: No input data check. `.TRUE.` : Input data *is* checked.	N/A	`.TRUE.`
`Use_Old_MWSSEM`	Logical switch to enable or disable the v2.0.x microwave sea surface emissivity model. If: `.FALSE.`: Use FASTEM5. `.TRUE.` : Use LFMWSSEM/FASTEM1.	N/A	`.FALSE.`
`Use_Antenna_Correction`	Logical switch to enable or disable the application of the antenna correction for the AMSU-A, AMSU-B, and MHS sensors. Note that for this switch to be effective in the CRTM call, the FOV field of the input `Geometry` structure must be set and the antenna correction coefficients must be present in the sensor `SpcCoeff` datafile. If: `.FALSE.`: No correction. `.TRUE.` : Apply antenna correction.	N/A	`.FALSE.`
`Apply_NLTE_Correction`	Logical switch to enable or disable the application of the non-LTE radiance correction. Note that for this switch to be effective in the CRTM call, the non-LTE correction coefficients must be present in the sensor `SpcCoeff` datafile. If: `.FALSE.`: No correction. `.TRUE.` : Apply non-LTE correction.	N/A	`.TRUE.`
`RT_Algorithm_Id`	Integer switch (using parameterised values) to select the scattering radiative transfer model. If: `RT_ADA`: Use ADA algorithm. `RT_SOI`: Use SOI algorithm.	N/A	`RT_ADA`
`Aircraft_Pressure`	Real value specifying an aircraft flight level pressure. If: $<$`0.0`: Satellite simulation. $>$`0.0`: Aircraft simulation.	hPa	`-1.0`
`Use_n_Streams`	Logical switch to enable or disable the use of a user-defined number of RT streams for scattering calculations. If: `.FALSE.`: Use internally calculated `n_Streams`. `.TRUE.` : Use specified `n_Streams`.	N/A	`.FALSE.`
`n_Streams`	Number of streams to use for scattering calculations if the `Use_n_Streams` is set to `.TRUE.`. Valid values for `n_Streams` are 2, 4, 6, 8, and 16.	N/A	`0`

Continued on Next Page...

Table 4.22 – Continued

Component	Description	Units	Default value
`Include_Scattering`	Logical switch to enable or disable scattering calculations for clouds and aerosols. If: `.FALSE.`: Only cloud and/or aerosol absorption is computed. `.TRUE.` : Cloud and/or aerosol absorption and scattering is computed.	N/A	`.TRUE.`
`n_Channels`	Number of sensor channels, L.	N/A	N/A
`Channel`	Index into channel-specific components.	N/A	0
`Use_Emissivity`	Logical switch to enable or disable the use of user-defined surface emissivity. If: `.FALSE.`: Calculate emissivity. `.TRUE.` : Use user-defined emissivity.	N/A	`.FALSE.`
`Emissivity(1:`L`)`	Allocatable array containing the user-defined surface emissivity for each sensor channel.	N/A	N/A
`Use_Direct_Reflectivity`	Logical switch to enable or disable the use of user-defined reflectivity for downwelling source (e.g. solar). This switch is ignored unless the `Use_Emissivity` switch is also set. If: `.FALSE.`: Calculate reflectivity. `.TRUE.` : Use user-defined reflectivity.	N/A	`.FALSE.`
`Direct_Reflectivity(1:`L`)`	Allocatable array containing the user-defined direct reflectivity for downwelling source for each sensor channel.	N/A	N/A
`SSU`	Structure component containing optional SSU sensor-specific input. See section A.10.	N/A	N/A
`Zeeman`	Structure component containing optional input for those sensors where Zeeman-splitting is an issue for high-peaking channels. See section A.11.	N/A	N/A

Some examples of assigning values to an `Options` structure are shown below.

Options influencing CRTM behaviour

To check the validity of input data within the CRTM, you can set the `Check_Input` logical component. Note that enabling this option could increase execution time.

```
! Check the input for profile #1...
opt(1)%Check_Input = .TRUE.
! ...but not for profile #2
opt(2)%Check_Input = .FALSE.
```

The default microwave sea surface emissivity model implemented in this release is FASTEM5 (or FASTEM4 if you initialise the CRTM using the requisite file). To switch back to the previous (i.e. "old") microwave sea surface emissivity model, a combination of the low-frequency model and FASTEM1, you can set the Use_Old_MWSSEM option,

```
! Use the old microwave sea surface emissivity model (MWSSEM) for profile #2
opt(2)%Use_Old_MWSSEM = .TRUE.
```

The default radiative transfer algorithm used for scattering calculation is the Advanced Doubling-Adding (ADA) algorithm with the Matrix Operator Method (MOM) for calculating layer quantities. To select an alternative algorithm, you can set the RT_Algorithm_Id option. Currently this is done by specifying a parameterised value identifying the algorithm. For example, to select the Successive Order of Interation (SOI) algorithm, the option is set to the parameter RT_SOI,

```
! Use the SOI algorithm for all scattering RT
opt%RT_Algorithm_Id = RT_SOI
```

To explicitly select the default RT algorithm, you can set the option to the parameter RT_ADA. The use of a parameterised integer value rather than a logical switch is to accommodate the implementation of additional algorithms in future releases.

If you wish to do simulations for aircraft instruments, you can enable this option by setting the aircraft flight level pressure,

```
! Specify an aircraft flight level pressure for profile #1
opt(1)%Aircraft_Pressure = 325.0_fp
```

Of course, doing aircraft sensor simulations requires the various sensor and transmittance models coefficients to be available for your instrument. To get that process started, contact CRTM Support[9]

This release of the CRTM also allows you to turn off cloud and aerosol scattering, performing only the absorption calculations, via the Include_Scattering option,

```
! Only perform cloud/aerosol absorption calculations for profile #1...
opt(1)%Include_Scattering = .FALSE.
```

If you do require the scattering calculations to be done, you can now also specify the number of streams you wish to be used for the calculations via the Use_n_Streams and n_Streams options,

```
! ...and do 4-stream scattering calculations for profile #2
opt(2)%Include_Scattering = .TRUE.
opt(2)%Use_n_Streams      = .TRUE.
opt(2)%n_Streams          = 4
```

Options for user-defined emissivities

You can also specify emissivity spectra for each input profile. For simplicity the example shown below assigns fixed values for all channels allocated in the Options structure,

[9]We'll need instrument information, e.g. spectral response or instrument line functions, to generate the CRTM transmittance coefficient data files.

```
! Specify the use of user-defined emissivities...
opt%Use_Emissivity = .TRUE.
! ...defining different "grey-body" fixed emissivities for each profile
opt(1)%Emissivity = 0.9525_fp
opt(2)%Emissivity = 0.8946_fp
additional profiles...
```

This setup, however, is problematical when you have multiple sensors (it's a actually an historical failure of the specification of the CRTM interface... but let's not go there.) Recall in section 4.4 that a loop over sensor was introduced to correctly allocate the channel-dependent arrays. This should be extended to the allocation of the `Options` structure itself (see 4.5.3) to allow emissivity spectra to be specified for the different sensors. Extending the sensor-loop example of section 4.4 with the specification of user-defined emissivities, we could do something like:

```
INTEGER :: m, n
....
Sensor_Loop: DO n = 1, n_sensors
  ....
  ! Get the number of channels to process for current sensor
  n_channels = CRTM_ChannelInfo_n_Channels( chinfo(n) )
  ....
  ! Allocate the options structure for this sensor to specify emissivity
  CALL CRTM_Options_Create( opt        , &
                            n_channels  )
  ! Check they were created successfully
  IF ( ANY(.NOT. CRTM_Options_Associated( opt )) ) THEN
    handle error...
  END IF
  ....
  ! Specify the use of user-defined emissivities in the options structure
  opt%Use_Emissivity = .TRUE.
  Profile_Loop: DO m = 1, n_profiles
    opt(m)%Emissivity(1:n_channels) = ...assign appropriate data...
  END DO Profile_Loop
  ....
END DO Sensor_Loop
```

Options for SSU and Zeeman models

The `SSU_Input` and `Zeeman_Input` structures are included in the `Options` input structure.

The components of the `SSU_Input` data structure are shown in table 4.23.

Table 4.23: CRTM `SSU_Input` structure component description

Component	Description	Units	Default value
`Time`	Time in decimal year corresponding to SSU observation.	N/A	0.0
`Cell_Pressure`	The SSU CO_2 cell pressures.	hPa	0.0

The SSU_Input data structure itself is declared as PRIVATE (see figure A.10). As such, the only way to set values in, or get values from, the structure is via the SSU_Input_SetValue or SSU_Input_GetValue subroutines respectively.

For example, to set the SSU instrument mission time, one would call the SSU_Input_SetValue subroutine like so,

```
! Set the SSU input data in the options substructure
CALL SSU_Input_SetValue( opt%SSU_Input    , &  ! Object
                         Time=mission_time  )  ! Optional input
```

where the local variable mission_time contains the required time.

The contents of the Zeeman_Input data structure are shown in table 4.23. similarly to the SSU_Input data structure, the Zeeman_Input data structure is also declared as PRIVATE and the corresponding Zeeman_Input_SetValue or Zeeman_Input_GetValue subroutines must be used to assign or retrieve values from the structure.

Table 4.24: CRTM Zeeman_Input structure component description

Component	Description	Units	Default value
Be	Earth magnetic field strength.	Gauss	0.3
Cos_ThetaB	Cosine of the angle between the Earth magnetic field and wave propagation direction.	N/A	0.0
Cos_PhiB	Cosine of the azimuth angle of the $\mathbf{B}_e$ vector in the $(\mathbf{v}, \mathbf{h}, \mathbf{k})$ coordinates system, where $\mathbf{v}$, $\mathbf{h}$ and $\mathbf{k}$ comprise a right-hand orthogonal system, similar to the $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ Cartesian coordinates. The $\mathbf{h}$ vector is normal to the plane containing the $\mathbf{k}$ and $\mathbf{z}$ vectors, where $\mathbf{k}$ points to the wave propagation direction and $\mathbf{z}$ points to the zenith. $\mathbf{h} = (\mathbf{z} \times \mathbf{k})/\lvert\mathbf{z} \times \mathbf{k}\rvert$. The azimuth angle is the angle on the $(\mathbf{v}, \mathbf{h})$ plane from the positive $\mathbf{v}$ axis to the projected line of the $\mathbf{B}_e$ vector on this plane, positive counterclockwise.	N/A	0.0
Doppler_Shift	Doppler frequency shift caused by Earth-rotation (positive towards sensor). A zero value means no frequency shift.	KHz	0.0

Setting the Earth's magnetic field strength and θ_B cosine in the Zeeman_Input structure is done via the Zeeman_Input_SetValue subroutine like so,

```
! Set the Zeeman input data in the options substructure
CALL Zeeman_Input_SetValue( opt%Zeeman_Input    , &  ! Object
                            Field_Strength=Be   , &  ! Optional input
                            Cos_ThetaB    =angle  )  ! Optional input
```

where, again, Be and angle are the local variables for the necessary data.

4.6.5 Initialising the K-matrix input and outputs

For the K-matrix structures, you should zero the K-matrix *outputs*, atm_K and sfc_K,

```
! Zero the K-matrix OUTPUT structures
CALL CRTM_Atmosphere_Zero( atm_K )
CALL CRTM_Surface_Zero( sfc_K )
```

and initialise the K-matrix *input*, `rts_K`, to provide you with the derivatives you want. For example, if you want the `atm_K`, `sfc_K` outputs to contain brightness temperature derivatives $\partial T_B/\partial x$, you should initialise `rts_K` like so,

```
! Initialise the K-Matrix INPUT to provide dTb/dx derivatives
rts_K%Radiance = ZERO
rts_K%Brightness_Temperature = ONE
```

Alternatively, if you want radiance derivatives returned in `atm_K` and `sfc_K`, the `rts_K` structure should be initialised like so,

```
! Initialise the K-Matrix INPUT to provide dR/dx derivatives
rts_K%Radiance = ONE
rts_K%Brightness_Temperature = ZERO
```

Note that, for visible channels, one should always set the K-Matrix input to provide $\partial R/\partial x$ derivatives since the generated brightness temperatures are for solar temperatures.

4.7 Call the required CRTM function

At this point, much of the preparatory heavy lifting has been done. The CRTM function calls themselves are quite simple.

4.7.1 The CRTM Forward model

The calling syntax for the CRTM forward model is,

```
err_stat = CRTM_Forward( atm        , & ! Input
                         sfc        , & ! Input
                         geo        , & ! Input
                         chInfo     , & ! Input
                         rts        , & ! Output
                         Options=opt  ) ! Optional input
IF ( err_stat /= SUCCESS ) THEN
  handle error...
END IF
```

Let's also specify the forward model call in the context of the sensor-loop example of section 4.4. It might look something like,

```
INTEGER :: m, n
....
Sensor_Loop: DO n = 1, n_sensors
  ....
  ! Get the number of channels to process for current sensor
```

```
    n_channels = CRTM_ChannelInfo_n_Channels( chinfo(n) )
    ....
    ! Allocate channel-dependent arrays
    ALLOCATE( rts(n_channels, n_profiles), &
              STAT = alloc_stat )
    IF ( alloc_stat /= 0 ) THEN
      handle error...
    END IF
    ....
    ! Call the forward model, processing ALL profiles at once.
    err_stat = CRTM_Forward( atm        , & ! Input
                             sfc        , & ! Input
                             geo        , & ! Input
                             chinfo(n:n), & ! Input
                             rts        , & ! Output
                             Options=opt  ) ! Optional input
    IF ( err_stat /= SUCCESS ) THEN
      handle error...
    END IF
    ....
    ! Deallocate channel-dependent arrays
    DEALLOCATE( rts, STAT = alloc_stat )
    IF ( alloc_stat /= 0 ) THEN
      handle error...
    END IF
  END DO Sensor_Loop
```

where we are procesing a single sensor at a time. Note the specification of the `ChannelInfo` argument, `chInfo(n:n)`. The use of the `(n:n)` modifier is required to ensure that a single element *array* is passed in to the forward model. If one simply wrote `chInfo(n)`, this specifies a scalar and the calling code would not compile[10].

4.7.2 The CRTM K-Matrix model

The calling syntax for the CRTM K-matrix model is,

```
  err_stat = CRTM_K_Matrix( atm        , & ! Forward  input
                            sfc        , & ! Forward  input
                            rts_K      , & ! K-matrix input
                            geo        , & ! Input
                            chinfo     , & ! Input
                            atm_K      , & ! K-matrix output
                            sfc_K      , & ! K-matrix output
                            rts        , & ! Forward  output
                            Options=opt  ) ! Optional input
  IF ( err_stat /= SUCCESS ) THEN
    handle error...
  END IF
```

Note that the K-matrix model also returns the forward model radiances.

[10]If you think this quirk is annoying and should be corrected, please email CRTM Support with your vote! ncep.list.emc.jcsda_crtm.support@noaa.gov

Similarly to the forward model example, let's recast the call within a sensor-loop,

```
INTEGER :: m, n
....
Sensor_Loop: DO n = 1, n_sensors
  ....
  ! Get the number of channels to process for current sensor
  n_channels = CRTM_ChannelInfo_n_Channels( chinfo(n) )
  ....
  ! Allocate channel-dependent arrays
  ALLOCATE( rts(n_channels, n_profiles)  , &
            atm_K(n_channels, n_profiles), &
            sfc_K(n_channels, n_profiles), &
            rts_K(n_channels, n_profiles), &
            STAT = alloc_stat )
  IF ( alloc_stat /= 0 ) THEN
    handle error...
  END IF
  ....
  ! Call the forward model, processing ALL profiles at once.
  err_stat = CRTM_K_Matrix( atm        , & ! Forward  input
                            sfc        , & ! Forward  input
                            rts_K      , & ! K-matrix input
                            geo        , & ! Input
                            chinfo(n:n), & ! Input
                            atm_K      , & ! K-matrix output
                            sfc_K      , & ! K-matrix output
                            rts        , & ! Forward  output
                            Options=opt  ) ! Optional input
  IF ( err_stat /= SUCCESS ) THEN
    handle error...
  END IF
  ....
  ! Deallocate channel-dependent arrays
  DEALLOCATE( rts, atm_K, sfc_K, rts_K, &
              STAT = alloc_stat )
  IF ( alloc_stat /= 0 ) THEN
    handle error...
  END IF
END DO Sensor_Loop
```

4.7.3 The CRTM Tangent-linear and Adjoint models

The tangent-linear and adjoint models have similar call structures and will not be shown here. Refer to their interface descriptions for details.

4.7.4 The CRTM Aerosol Optical Depth (AOD) functions

There is a separate module containing forward, tangent-linear, adjoint and K-matrix function to *just* compute aerosol optical depths. The calling syntax for these functions are similar to the main function, but with fewer argument.

The calling syntax for the CRTM forward AOD model is,

```
err_stat = CRTM_AOD( atm         , & ! Input
                     chInfo      , & ! Input
                     rts         , & ! Output
                     Options=opt  ) ! Optional input
IF ( err_stat /= SUCCESS ) THEN
  handle error...
END IF
```

A important note: the computed aerosol optical depth is stored in the `Layer_Optical_Depth` component of the `RTSolution` output so you must allocate the internals of the `RTSolution` structure. Using the call in the context of the sensor-loop example of section 4.4, we would do,

```
INTEGER :: m, n
....
Sensor_Loop: DO n = 1, n_sensors
  ....
  ! Get the number of channels to process for current sensor
  n_channels = CRTM_ChannelInfo_n_Channels( chinfo(n) )
  ....
  ! Allocate channel-dependent arrays
  ALLOCATE( rts(n_channels, n_profiles), &
            STAT = alloc_stat )
  IF ( alloc_stat /= 0 ) THEN
    handle error...
  END IF
  ....
  ! Allocate RTSolution structure to store optical depth output
  CALL CRTM_RTSolution_Create( rts, n_layers )
  IF ( .NOT. ALL(CRTM_RTSolution_Associated(rts)) ) THEN
    handle error...
  END IF
  ...
  ! Call the forward AOD model, processing ALL profiles at once.
  err_stat = CRTM_AOD( atm         , & ! Input
                       chinfo(n:n), & ! Input
                       rts         , & ! Output
                       Options=opt  ) ! Optional input
  IF ( err_stat /= SUCCESS ) THEN
    handle error...
  END IF
  ....
  ! Deallocate channel-dependent arrays
  DEALLOCATE( rts, STAT = alloc_stat )
  IF ( alloc_stat /= 0 ) THEN
    handle error...
  END IF
END DO Sensor_Loop
```

The aerosol optical depth tangent-linear, adjoint, and K-matrix functions have call structures similar to the main function and will not be shown here. Refer to their interface descriptions for details.

4.8 Inspect the CRTM output structures

Regardless of whether you have called the forward or K-matrix model, you will want to have a look at the results in the `RTSolution` structure. The components of this structure are shown in table 4.25. The modifier "(1:K)" indicates the range of the allocatable components.

Table 4.25: CRTM `RTSolution` structure component description. †Only defined for *forward* radiative transfer computations.

Component	Description	Units	Default value
`n_Layers`	Number of atmospheric profile layers, `K`	N/A	0
`Sensor_Id`	The sensor id string	N/A	empty string
`WMO_Satellite_Id`	The WMO satellite Id	N/A	`INVALID_WMO_SATELLITE_ID`
`WMO_Sensor_Id`	The WMO sensor Id	N/A	`INVALID_WMO_SENSOR_ID`
`Sensor_Channel`	The channel number	N/A	0
`RT_Algorithm_Name`	Character string containing the name of the radiative transfer algorithm used.	N/A	empty string
`SOD`†	The scattering optical depth	N/A	0.0
`Surface_Emissivity`†	The surface emissivity (computed or user-defined)	N/A	0.0
`Up_Radiance`†	The atmospheric portion of the upwelling radiance	$mW/(m^2.sr.cm^{-1})$	0.0
`Down_Radiance`†	The atmospheric portion of the downwelling radiance	$mW/(m^2.sr.cm^{-1})$	0.0
`Down_Solar_Radiance`†	The downwelling direct solar radiance	$mW/(m^2.sr.cm^{-1})$	0.0
`Surface_Planck_Radiance`†	The surface radiance	$mW/(m^2.sr.cm^{-1})$	0.0
`Upwelling_Radiance(1:`K`)`†	The upwelling radiance profile, including the reflected downwelling and surface contributions.	$mW/(m^2.sr.cm^{-1})$	N/A
`Layer_Optical_Depth(1:`K`)`†	The layer optical depth profile	N/A	N/A
`Radiance`	The sensor radiance	$mW/(m^2.sr.cm^{-1})$	0.0
`Brightness_Temperature`	The sensor brightness temperature	Kelvin	0.0

Although most people are interested in using the radiance or brightness temperature component, you can dump the entire contents of the `RTSolution` structure directly to screen using the `CRTM_RTSolution_Inspect` procedure,

```
CALL CRTM_RTSolution_Inspect(rts_K)
```

4.9 Destroy the CRTM and cleanup

The last step is to cleanup. This involves calling the CRTM destruction function

```
err_stat = CRTM_Destroy( chinfo )
IF ( err_stat /= SUCCESS ) THEN
  handle error...
END IF
```

to deallocate all the shared coefficient data that was read during the intialisation step.

Note that one can also call the individual CRTM structure subroutines as well to deallocate the internals of the various structure arrays that were created in section 4.5. The cleanup mirrors that of the create step:

```
CALL CRTM_Options_Destroy(opt)
CALL CRTM_RTSolution_Destroy(rts)
CALL CRTM_Atmosphere_Destroy(atm)
```

If you also have K-matrix structures, you also call the destruction subroutines for htem too:

```
CALL CRTM_RTSolution_Destroy(rts_K)
CALL CRTM_Atmosphere_Destroy(atm_K)
```

However, it should be pointed out that deallocating the structure arrays also deallocates the internals of each element of a structure. To use the `Atmosphere` array, `atm`, as an example; doing the following,

```
DEALLOCATE( atm, STAT = alloc_stat )
IF ( alloc_stat /= 0 ) THEN
  handle error...
END IF
```

is equivalent to,

```
! Deallocate the array element internals
CALL CRTM_Atmosphere_Destroy(atm)
! Deallocate the array itself
DEALLOCATE( atm, STAT = alloc_stat )
IF ( alloc_stat /= 0 ) THEN
  handle error...
END IF
```

since, in Fortran95+TR15581 and Fortran2003 the array deallocation will also deallocate any structure components that have an `ALLOCATABLE` attribute.

5 Interface Descriptions

5.1 Initialisation functions

5.1.1 `CRTM_Init` interface

```
NAME:
      CRTM_Init

PURPOSE:
      Function to initialise the CRTM.

CALLING SEQUENCE:
      Error_Status = CRTM_Init( Sensor_ID  , &
                                ChannelInfo, &
                                Aerosol_Model       = Aerosol_Model       , &
                                AerosolCoeff_Format = AerosolCoeff_Format , &
                                AerosolCoeff_File   = AerosolCoeff_File   , &
                                Cloud_Model         = Cloud_Model         , &
                                CloudCoeff_Format   = CloudCoeff_Format   , &
                                CloudCoeff_File     = CloudCoeff_File     , &
                                Load_CloudCoeff     = Load_CloudCoeff     , &
                                Load_AerosolCoeff   = Load_AerosolCoeff   , &
                                IRwaterCoeff_File   = IRwaterCoeff_File   , &
                                IRlandCoeff_File    = IRlandCoeff_File    , &
                                IRsnowCoeff_File    = IRsnowCoeff_File    , &
                                IRiceCoeff_File     = IRiceCoeff_File     , &
                                VISwaterCoeff_File  = VISwaterCoeff_File  , &
                                VISlandCoeff_File   = VISlandCoeff_File   , &
                                VISsnowCoeff_File   = VISsnowCoeff_File   , &
                                VISiceCoeff_File    = VISiceCoeff_File    , &
                                MWwaterCoeff_File   = MWwaterCoeff_File   , &
                                File_Path           = File_Path           , &
                                NC_File_Path        = NC_File_Path        , &
                                Quiet               = Quiet               , &
                                Process_ID          = Process_ID          , &
                                Output_Process_ID   = Output_Process_ID   )

INPUTS:
      Sensor_ID:          List of the sensor IDs (e.g. hirs3_n17, amsua_n18,
                          ssmis_f16, etc) with which the CRTM is to be
                          initialised. These sensor ids are used to construct
                          the sensor specific SpcCoeff and TauCoeff filenames
                          containing the necessary coefficient data, i.e.
                            <Sensor_ID>.SpcCoeff.bin
                          and
                            <Sensor_ID>.TauCoeff.bin
                          for each sensor Id in the list.
                          UNITS:      N/A
                          TYPE:       CHARACTER(*)
                          DIMENSION:  Rank-1 (n_Sensors)
                          ATTRIBUTES: INTENT(IN), OPTIONAL

OUTPUTS:
      ChannelInfo:        ChannelInfo structure array populated based on
                          the contents of the coefficient files and the
                          user inputs.
                          UNITS:      N/A
                          TYPE:       CRTM_ChannelInfo_type
                          DIMENSION:  Same as input Sensor_Id argument
                          ATTRIBUTES: INTENT(OUT)

OPTIONAL INPUTS:
      Aerosol_Model:      Name of the aerosol scheme for scattering calculation
                          Available aerosol scheme:
                          - GOCART  [DEFAULT]
                          - CMAQ
                          UNITS:      N/A
                          TYPE:       CHARACTER(*)
                          DIMENSION:  Scalar
                          ATTRIBUTES: INTENT(IN), OPTIONAL

      AerosolCoeff_Format: Format of the aerosol optical properties data
                           Available options:
                           - Binary  [DEFAULT]
                           - netCDF
                           UNITS:      N/A
                           TYPE:       CHARACTER(*)
                           DIMENSION:  Scalar
                           ATTRIBUTES: INTENT(IN), OPTIONAL

      AerosolCoeff_File:  Name of the data file containing the aerosol optical
                          properties data for scattering calculations.
                          Available datafiles:
                          GOCART:
                          - AerosolCoeff.bin      [DEFAULT, Binary]
                          - AerosolCoeff.nc/nc4   [netCDF-Classic/4]
                          CMAQ:
                          - AerosolCoeff.CMAQ.bin      [Binary]
                          - AerosolCoeff.CMAQ.nc/nc4   [netCDF-Classic/4]
                          UNITS:      N/A
                          TYPE:       CHARACTER(*)
                          DIMENSION:  Scalar
```

```
                    ATTRIBUTES: INTENT(IN), OPTIONAL

Cloud_Model:        Name of the cloud scheme for scattering calculation
                    Available cloud scheme:
                    - CRTM  [DEFAULT]
                    UNITS:      N/A
                    TYPE:       CHARACTER(*)
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN), OPTIONAL

CloudCoeff_Format:  Format of the cloud optical properties data
                    Available options
                    - Binary  [DEFAULT]
                    - netCDF
                    UNITS:      N/A
                    TYPE:       CHARACTER(*)
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN), OPTIONAL

CloudCoeff_File:    Name of the data file containing the cloud optical
                    properties data for scattering calculations.
                    Available datafiles:
                    - CloudCoeff.bin  [DEFAULT, Binary]
                    - CloudCoeff.nc     [netCDF-Classic/4]
                    UNITS:      N/A
                    TYPE:       CHARACTER(*)
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN), OPTIONAL

Load_CloudCoeff:    Set this logical argument for not loading the CloudCoeff data
                    to save memory space under the clear conditions
                    If == .FALSE., the CloudCoeff data will not be loaded;
                       == .TRUE.,  the CloudCoeff data will be loaded.
                    If not specified, default is .TRUE. (will be loaded)
                    UNITS:      N/A
                    TYPE:       LOGICAL
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN), OPTIONAL

Load_AerosolCoeff:  Set this logical argument for not loading the AerosolCoeff data
                    to save memory space under the clear conditions
                    If == .FALSE., the AerosolCoeff data will not be loaded;
                       == .TRUE.,  the AerosolCoeff data will be loaded.
                    If not specified, default is .TRUE. (will be loaded)
                    UNITS:      N/A
                    TYPE:       LOGICAL
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN), OPTIONAL

MWwaterCoeff_File:  Name of the data file containing the coefficient
                    data for the microwave water emissivity model.
                    Available datafiles:
                    - FASTEM5.MWwater.EmisCoeff.bin  [DEFAULT]
                    - FASTEM4.MWwater.EmisCoeff.bin
```

```
                  UNITS:      N/A
                  TYPE:       CHARACTER(*)
                  DIMENSION:  Scalar
                  ATTRIBUTES: INTENT(IN), OPTIONAL

IRwaterCoeff_File:  Name of the data file containing the coefficient
                    data for the infrared water emissivity model.
                    Available datafiles:
                    - Nalli.IRwater.EmisCoeff.bin  [DEFAULT]
                    - WuSmith.IRwater.EmisCoeff.bin
                    If not specified the Nalli datafile is read.
                    UNITS:      N/A
                    TYPE:       CHARACTER(*)
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN), OPTIONAL

IRlandCoeff_File:   Name of the data file containing the coefficient
                    data for the infrared land emissivity model.
                    Available datafiles:
                    - NPOESS.IRland.EmisCoeff.bin  [DEFAULT]
                    - IGBP.IRland.EmisCoeff.bin
                    - USGS.IRland.EmisCoeff.bin
                    UNITS:      N/A
                    TYPE:       CHARACTER(*)
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN), OPTIONAL

IRsnowCoeff_File:   Name of the data file containing the coefficient
                    data for the infrared snow emissivity model.
                    Available datafiles:
                    - NPOESS.IRsnow.EmisCoeff.bin  [DEFAULT]
                    - IGBP.IRsnow.EmisCoeff.bin
                    - USGS.IRsnow.EmisCoeff.bin
                    UNITS:      N/A
                    TYPE:       CHARACTER(*)
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN), OPTIONAL

IRiceCoeff_File:    Name of the data file containing the coefficient
                    data for the infrared ice emissivity model.
                    Available datafiles:
                    - NPOESS.IRice.EmisCoeff.bin  [DEFAULT]
                    - IGBP.IRice.EmisCoeff.bin
                    - USGS.IRice.EmisCoeff.bin
                    UNITS:      N/A
                    TYPE:       CHARACTER(*)
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN), OPTIONAL

VISwaterCoeff_File: Name of the data file containing the coefficient
                    data for the visible water emissivity model.
                    Available datafiles:
                    - NPOESS.VISwater.EmisCoeff.bin  [DEFAULT]
                    - IGBP.VISwater.EmisCoeff.bin
                    - USGS.VISwater.EmisCoeff.bin
                    UNITS:      N/A
                    TYPE:       CHARACTER(*)
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN), OPTIONAL

VISlandCoeff_File:  Name of the data file containing the coefficient
                    data for the visible land emissivity model.
                    Available datafiles:
                    - NPOESS.VISland.EmisCoeff.bin  [DEFAULT]
                    - IGBP.VISland.EmisCoeff.bin
                    - USGS.VISland.EmisCoeff.bin
                    UNITS:      N/A
                    TYPE:       CHARACTER(*)
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN), OPTIONAL

VISsnowCoeff_File:  Name of the data file containing the coefficient
                    data for the visible snow emissivity model.
                    Available datafiles:
                    - NPOESS.VISsnow.EmisCoeff.bin  [DEFAULT]
                    - IGBP.VISsnow.EmisCoeff.bin
                    - USGS.VISsnow.EmisCoeff.bin
                    UNITS:      N/A
                    TYPE:       CHARACTER(*)
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN), OPTIONAL

VISiceCoeff_File:   Name of the data file containing the coefficient
                    data for the visible ice emissivity model.
                    Available datafiles:
                    - NPOESS.VISice.EmisCoeff.bin  [DEFAULT]
                    - IGBP.VISice.EmisCoeff.bin
                    - USGS.VISice.EmisCoeff.bin
                    UNITS:      N/A
                    TYPE:       CHARACTER(*)
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN), OPTIONAL

File_Path:          Character string specifying a file path for the
                    input data files. If not specified, the current
                    directory is the default.
                    UNITS:      N/A
                    TYPE:       CHARACTER(*)
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN), OPTIONAL

NC_File_Path:       Character string specifying a file path for the
                    input data files in netCDF format. If not specified,
                    the current directory is the default.
                    UNITS:      N/A
                    TYPE:       CHARACTER(*)
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN), OPTIONAL
```

```
      Quiet:              Set this logical argument to suppress INFORMATION
                          messages being printed to stdout
                          If == .FALSE., INFORMATION messages are OUTPUT [DEFAULT].
                             == .TRUE.,  INFORMATION messages are SUPPRESSED.
                          If not specified, default is .FALSE.
                          UNITS:      N/A
                          TYPE:       LOGICAL
                          DIMENSION:  Scalar
                          ATTRIBUTES: INTENT(IN), OPTIONAL

      Process_ID:         Set this argument to the MPI process ID that this
                          function call is running under. This value is used
                          solely for controlling INFORMATION message output.
                          If MPI is not being used, ignore this argument.
                          This argument is ignored if the Quiet argument is set.
                          UNITS:      N/A
                          TYPE:       INTEGER
                          DIMENSION:  Scalar
                          ATTRIBUTES: INTENT(IN), OPTIONAL

      Output_Process_ID:  Set this argument to the MPI process ID in which
                          all INFORMATION messages are to be output. If
                          the passed Process_ID value agrees with this value
                          the INFORMATION messages are output.
                          This argument is ignored if the Quiet argument
                          is set.
                          UNITS:      N/A
                          TYPE:       INTEGER
                          DIMENSION:  Scalar
                          ATTRIBUTES: INTENT(IN), OPTIONAL

FUNCTION RESULT:
      Error_Status:       The return value is an integer defining the error
                          status. The error codes are defined in the
                          Message_Handler module.
                          If == SUCCESS the CRTM initialisation was successful
                             == FAILURE an unrecoverable error occurred.
                          UNITS:      N/A
                          TYPE:       INTEGER
                          DIMENSION:  Scalar

SIDE EFFECTS:
      All public data arrays accessed by this module and its dependencies
      are overwritten.
```

5.2 Main functions

5.2.1 CRTM_Forward interface

```
NAME:
      CRTM_Forward

PURPOSE:
      Function that calculates top-of-atmosphere (TOA) radiances
      and brightness temperatures for an input atmospheric profile or
      profile set and user specified satellites/channels.

CALLING SEQUENCE:
      Error_Status = CRTM_Forward( Atmosphere       , &
                                   Surface          , &
                                   Geometry         , &
                                   ChannelInfo      , &
                                   RTSolution       , &
                                   Options = Options  )

INPUTS:
      Atmosphere:     Structure containing the Atmosphere data.
                      UNITS:      N/A
                      TYPE:       CRTM_Atmosphere_type
                      DIMENSION:  Rank-1 (n_Profiles)
                      ATTRIBUTES: INTENT(IN)

      Surface:        Structure containing the Surface data.
                      UNITS:      N/A
                      TYPE:       CRTM_Surface_type
                      DIMENSION:  Same as input Atmosphere structure
                      ATTRIBUTES: INTENT(IN)

      Geometry:       Structure containing the view geometry
                      information.
                      UNITS:      N/A
                      TYPE:       CRTM_Geometry_type
                      DIMENSION:  Same as input Atmosphere structure
                      ATTRIBUTES: INTENT(IN)

      ChannelInfo:    Structure returned from the CRTM_Init() function
                      that contains the satellite/sensor channel index
                      information.
                      UNITS:      N/A
                      TYPE:       CRTM_ChannelInfo_type
                      DIMENSION:  Rank-1 (n_Sensors)
                      ATTRIBUTES: INTENT(IN)

OUTPUTS:
      RTSolution:     Structure containing the soluition to the RT equation
                      for the given inputs.
                      UNITS:      N/A
```

```
                    TYPE:       CRTM_RTSolution_type
                    DIMENSION:  Rank-2 (n_Channels x n_Profiles)
                    ATTRIBUTES: INTENT(IN OUT)

OPTIONAL INPUTS:
      Options:      Options structure containing the optional arguments
                    for the CRTM.
                    UNITS:      N/A
                    TYPE:       CRTM_Options_type
                    DIMENSION:  Same as input Atmosphere structure
                    ATTRIBUTES: INTENT(IN), OPTIONAL

FUNCTION RESULT:
      Error_Status: The return value is an integer defining the error status.
                    The error codes are defined in the Message_Handler module.
                    If == SUCCESS the computation was sucessful
                       == FAILURE an unrecoverable error occurred
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Scalar

COMMENTS:
      - The Options optional input structure argument contains
        spectral information (e.g. emissivity) that must have the same
        spectral dimensionality (the "L" dimension) as the output
        RTSolution structure.
```

5.2.2 CRTM_Tangent_Linear interface

```
NAME:
      CRTM_Tangent_Linear

PURPOSE:
      Function that calculates tangent-linear top-of-atmosphere (TOA)
      radiances and brightness temperatures for an input atmospheric
      profile or profile set and user specified satellites/channels.

CALLING SEQUENCE:
      Error_Status = CRTM_Tangent_Linear( Atmosphere       , &
                                          Surface          , &
                                          Atmosphere_TL    , &
                                          Surface_TL       , &
                                          Geometry         , &
                                          ChannelInfo      , &
                                          RTSolution       , &
                                          RTSolution_TL    , &
                                          Options = Options  )

INPUTS:
      Atmosphere:     Structure containing the Atmosphere data.
                      UNITS:      N/A
                      TYPE:       CRTM_Atmosphere_type
                      DIMENSION:  Rank-1 (n_Profiles)
                      ATTRIBUTES: INTENT(IN)

      Surface:        Structure containing the Surface data.
                      UNITS:      N/A
                      TYPE:       CRTM_Surface_type
                      DIMENSION:  Same as input Atmosphere structure
                      ATTRIBUTES: INTENT(IN)

      Atmosphere_TL:  Structure containing the tangent-linear Atmosphere data.
                      UNITS:      N/A
                      TYPE:       CRTM_Atmosphere_type
                      DIMENSION:  Same as input Atmosphere structure
                      ATTRIBUTES: INTENT(IN)

      Surface_TL:     Structure containing the tangent-linear Surface data.
                      UNITS:      N/A
                      TYPE:       CRTM_Surface_type
                      DIMENSION:  Same as input Atmosphere structure
                      ATTRIBUTES: INTENT(IN)

      Geometry:       Structure containing the view geometry
                      information.
                      UNITS:      N/A
                      TYPE:       CRTM_Geometry_type
                      DIMENSION:  Same as input Atmosphere structure
                      ATTRIBUTES: INTENT(IN)
      ChannelInfo:    Structure returned from the CRTM_Init() function
                      that contains the satellite/sensor channel index
                      information.
                      UNITS:      N/A
                      TYPE:       CRTM_ChannelInfo_type
                      DIMENSION:  Rank-1 (n_Sensors)
                      ATTRIBUTES: INTENT(IN)

OUTPUTS:
      RTSolution:     Structure containing the solution to the RT equation
                      for the given inputs.
                      UNITS:      N/A
                      TYPE:       CRTM_RTSolution_type
                      DIMENSION:  Rank-2 (n_Channels x n_Profiles)
                      ATTRIBUTES: INTENT(IN OUT)

      RTSolution_TL:  Structure containing the solution to the tangent-
                      linear RT equation for the given inputs.
                      UNITS:      N/A
                      TYPE:       CRTM_RTSolution_type
                      DIMENSION:  Rank-2 (n_Channels x n_Profiles)
                      ATTRIBUTES: INTENT(IN OUT)

OPTIONAL INPUTS:
      Options:        Options structure containing the optional forward model
                      arguments for the CRTM.
                      UNITS:      N/A
                      TYPE:       CRTM_Options_type
                      DIMENSION:  Same as input Atmosphere structure
                      ATTRIBUTES: INTENT(IN), OPTIONAL

FUNCTION RESULT:
      Error_Status:   The return value is an integer defining the error status.
                      The error codes are defined in the Message_Handler module.
                      If == SUCCESS the computation was sucessful
                         == FAILURE an unrecoverable error occurred
                      UNITS:      N/A
                      TYPE:       INTEGER
                      DIMENSION:  Scalar

COMMENTS:
      - The Options optional input structure arguments contain
        spectral information (e.g. emissivity) that must have the same
        spectral dimensionality (the "L" dimension) as the output
        RTSolution structures.
```

5.2.3 CRTM_Adjoint interface

```
NAME:
      CRTM_Adjoint

PURPOSE:
      Function that calculates the adjoint of top-of-atmosphere (TOA)
      radiances and brightness temperatures for an input atmospheric
      profile or profile set and user specified satellites/channels.

CALLING SEQUENCE:
      Error_Status = CRTM_Adjoint( Atmosphere       , &
                                   Surface          , &
                                   RTSolution_AD    , &
                                   Geometry         , &
                                   ChannelInfo      , &
                                   Atmosphere_AD    , &
                                   Surface_AD       , &
                                   RTSolution       , &
                                   Options = Options  )

INPUTS:
      Atmosphere:     Structure containing the Atmosphere data.
                      UNITS:      N/A
                      TYPE:       CRTM_Atmosphere_type
                      DIMENSION:  Rank-1 (n_Profiles)
                      ATTRIBUTES: INTENT(IN)

      Surface:        Structure containing the Surface data.
                      UNITS:      N/A
                      TYPE:       CRTM_Surface_type
                      DIMENSION:  Same as input Atmosphere structure
                      ATTRIBUTES: INTENT(IN)

      RTSolution_AD:  Structure containing the RT solution adjoint inputs.
                      **NOTE: On EXIT from this function, the contents of
                              this structure may be modified (e.g. set to
                              zero.)
                      UNITS:      N/A
                      TYPE:       CRTM_RTSolution_type
                      DIMENSION:  Rank-2 (n_Channels x n_Profiles)
                      ATTRIBUTES: INTENT(IN OUT)

      Geometry:       Structure containing the view geometry
                      information.
                      UNITS:      N/A
                      TYPE:       CRTM_Geometry_type
                      DIMENSION:  Same as input Atmosphere argument
                      ATTRIBUTES: INTENT(IN)

      ChannelInfo:    Structure returned from the CRTM_Init() function
                      that contains the satellite/sensor channel index
                      information.
```

```
                    UNITS:      N/A
                    TYPE:       CRTM_ChannelInfo_type
                    DIMENSION:  Rank-1 (n_Sensors)
                    ATTRIBUTES: INTENT(IN)

OPTIONAL INPUTS:
      Options:      Options structure containing the optional forward model
                    arguments for the CRTM.
                    UNITS:      N/A
                    TYPE:       CRTM_Options_type
                    DIMENSION:  Same as input Atmosphere structure
                    ATTRIBUTES: INTENT(IN), OPTIONAL

OUTPUTS:
      Atmosphere_AD: Structure containing the adjoint Atmosphere data.
                    **NOTE: On ENTRY to this function, the contents of
                            this structure should be defined (e.g.
                            initialized to some value based on the
                            position of this function in the call chain.)
                    UNITS:      N/A
                    TYPE:       CRTM_Atmosphere_type
                    DIMENSION:  Same as input Atmosphere argument
                    ATTRIBUTES: INTENT(IN OUT)

      Surface_AD:   Structure containing the tangent-linear Surface data.
                    **NOTE: On ENTRY to this function, the contents of
                            this structure should be defined (e.g.
                            initialized to some value based on the
                            position of this function in the call chain.)
                    UNITS:      N/A
                    TYPE:       CRTM_Surface_type
                    DIMENSION:  Same as input Atmosphere argument
                    ATTRIBUTES: INTENT(IN OUT)

      RTSolution:   Structure containing the solution to the RT equation
                    for the given inputs.
                    UNITS:      N/A
                    TYPE:       CRTM_RTSolution_type
                    DIMENSION:  Same as input RTSolution_AD argument
                    ATTRIBUTES: INTENT(IN OUT)

FUNCTION RESULT:
      Error_Status: The return value is an integer defining the error status.
                    The error codes are defined in the Message_Handler module.
                    If == SUCCESS the computation was sucessful
                       == FAILURE an unrecoverable error occurred
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Scalar

SIDE EFFECTS:
     Note that the input adjoint arguments are modified upon exit, and
     the output adjoint arguments must be defined upon entry. This is
     a consequence of the adjoint formulation where, effectively, the
     chain rule is being used and this function could reside anywhere
     in the chain of derivative terms.

COMMENTS:
      - The Options optional structure arguments contain
        spectral information (e.g. emissivity) that must have the same
        spectral dimensionality (the "L" dimension) as the RTSolution
        structures.
```

5.2.4 CRTM_K_Matrix interface

```
NAME:
      CRTM_K_Matrix

PURPOSE:
      Function that calculates the K-matrix of top-of-atmosphere (TOA)
      radiances and brightness temperatures for an input atmospheric
      profile or profile set and user specified satellites/channels.

CALLING SEQUENCE:
      Error_Status = CRTM_K_Matrix( Atmosphere       , &
                                    Surface          , &
                                    RTSolution_K     , &
                                    Geometry         , &
                                    ChannelInfo      , &
                                    Atmosphere_K     , &
                                    Surface_K        , &
                                    RTSolution       , &
                                    Options = Options  )

INPUTS:
      Atmosphere:     Structure containing the Atmosphere data.
                      UNITS:      N/A
                      TYPE:       CRTM_Atmosphere_type
                      DIMENSION:  Rank-1 (n_Profiles)
                      ATTRIBUTES: INTENT(IN)

      Surface:        Structure containing the Surface data.
                      UNITS:      N/A
                      TYPE:       CRTM_Surface_type
                      DIMENSION:  Same as input Atmosphere argument.
                      ATTRIBUTES: INTENT(IN)

      RTSolution_K:   Structure containing the RT solution K-matrix inputs.
                      **NOTE: On EXIT from this function, the contents of
                              this structure may be modified (e.g. set to
                              zero.)
                      UNITS:      N/A
                      TYPE:       CRTM_RTSolution_type
                      DIMENSION:  Rank-2 (n_Channels x n_Profiles)
                      ATTRIBUTES: INTENT(IN OUT)

      Geometry:       Structure containing the view geometry
                      information.
                      UNITS:      N/A
                      TYPE:       CRTM_Geometry_type
                      DIMENSION:  Same as input Atmosphere argument
                      ATTRIBUTES: INTENT(IN)

      ChannelInfo:    Structure returned from the CRTM_Init() function
                      that contains the satellite/sesnor channel index
                      information.
```

```
                  UNITS:      N/A
                  TYPE:       CRTM_ChannelInfo_type
                  DIMENSION:  Rank-1 (n_Sensors)
                  ATTRIBUTES: INTENT(IN)

OPTIONAL INPUTS:
      Options:    Options structure containing the optional forward model
                  arguments for the CRTM.
                  UNITS:      N/A
                  TYPE:       CRTM_Options_type
                  DIMENSION:  Same as input Atmosphere structure
                  ATTRIBUTES: INTENT(IN), OPTIONAL

OUTPUTS:
      Atmosphere_K:   Structure containing the K-matrix Atmosphere data.
                  **NOTE: On ENTRY to this function, the contents of
                          this structure should be defined (e.g.
                          initialized to some value based on the
                          position of this function in the call chain.)
                  UNITS:      N/A
                  TYPE:       CRTM_Atmosphere_type
                  DIMENSION:  Same as input RTSolution_K argument
                  ATTRIBUTES: INTENT(IN OUT)

      Surface_K:      Structure containing the tangent-linear Surface data.
                  **NOTE: On ENTRY to this function, the contents of
                          this structure should be defined (e.g.
                          initialized to some value based on the
                          position of this function in the call chain.)
                  UNITS:      N/A
                  TYPE:       CRTM_Surface_type
                  DIMENSION:  Same as input RTSolution_K argument
                  ATTRIBUTES: INTENT(IN OUT)

      RTSolution:     Structure containing the solution to the RT equation
                  for the given inputs.
                  UNITS:      N/A
                  TYPE:       CRTM_RTSolution_type
                  DIMENSION:  Same as input RTSolution_K argument
                  ATTRIBUTES: INTENT(IN OUT)

FUNCTION RESULT:
      Error_Status:   The return value is an integer defining the error status.
                  The error codes are defined in the Message_Handler module.
                  If == SUCCESS the computation was sucessful
                     == FAILURE an unrecoverable error occurred
                  UNITS:      N/A
                  TYPE:       INTEGER
                  DIMENSION:  Scalar

SIDE EFFECTS:
     Note that the input K-matrix arguments are modified upon exit, and
     the output K-matrix arguments must be defined upon entry. This is
     a consequence of the K-matrix formulation where, effectively, the
     chain rule is being used and this funtion could reside anywhere
     in the chain of derivative terms.

COMMENTS:
      - The Options optional structure arguments contain
        spectral information (e.g. emissivity) that must have the same
        spectral dimensionality (the "L" dimension) as the RTSolution
        structures.
```

5.3 Aerosol optical depth functions

5.3.1 CRTM_AOD interface

```
NAME:
      CRTM_AOD

PURPOSE:
      Function that calculates layer total optical depth profile at nadir.

CALLING SEQUENCE:
      Error_Status = CRTM_AOD( Atmosphere       , &
                               ChannelInfo      , &
                               RTSolution       , &
                               Options = Options  )

INPUTS:
      Atmosphere:     Structure containing the Atmosphere data.
                      UNITS:      N/A
                      TYPE:       CRTM_Atmosphere_type
                      DIMENSION:  Rank-1 (n_Profiles)
                      ATTRIBUTES: INTENT(IN)

      ChannelInfo:    Structure returned from the CRTM_Init() function
                      that contains the satellite/sensor channel index
                      information.
                      UNITS:      N/A
                      TYPE:       CRTM_ChannelInfo_type
                      DIMENSION:  Rank-1 (n_Sensors)
                      ATTRIBUTES: INTENT(IN)

OUTPUTS:
      RTSolution:     Structure containing the layer aerosol optical
                      profile for the given inputs.
                      UNITS:      N/A
                      TYPE:       CRTM_RTSolution_type
                      DIMENSION:  Rank-2 (n_Channels x n_Profiles)
                      ATTRIBUTES: INTENT(IN OUT)

OPTIONAL INPUTS:
      Options:        Options structure containing the optional arguments
                      for the CRTM.
                      UNITS:      N/A
                      TYPE:       CRTM_Options_type
                      DIMENSION:  Same as input Atmosphere structure
                      ATTRIBUTES: INTENT(IN), OPTIONAL

FUNCTION RESULT:
      Error_Status:   The return value is an integer defining the error status.
                      The error codes are defined in the Message_Handler module.
                      If == SUCCESS the computation was sucessful
                         == FAILURE an unrecoverable error occurred
```

```
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Scalar

COMMENTS:
      - Many of the components of the Options optional input structure
        are not used in this function. Consult the CRTM User Guide for
        which Options components are usable for AOD calculations.
```

5.3.2 CRTM_AOD_TL interface

```
NAME:
      CRTM_AOD_TL

PURPOSE:
      Function that calculates tangent-linear layer total optical depth.

CALLING SEQUENCE:
      Error_Status = CRTM_AOD_TL( Atmosphere       , &
                                  Atmosphere_TL    , &
                                  ChannelInfo      , &
                                  RTSolution       , &
                                  RTSolution_TL    , &
                                  Options = Options  )

INPUTS:
      Atmosphere:     Structure containing the Atmosphere data.
                      UNITS:      N/A
                      TYPE:       CRTM_Atmosphere_type
                      DIMENSION:  Rank-1 (n_Profiles)
                      ATTRIBUTES: INTENT(IN)

      Atmosphere_TL:  Structure containing the tangent-linear Atmosphere data.
                      UNITS:      N/A
                      TYPE:       CRTM_Atmosphere_type
                      DIMENSION:  Same as input Atmosphere structure
                      ATTRIBUTES: INTENT(IN)

      ChannelInfo:    Structure returned from the CRTM_Init() function
                      that contains the satellite/sensor channel index
                      information.
                      UNITS:      N/A
                      TYPE:       CRTM_ChannelInfo_type
                      DIMENSION:  Rank-1 (n_Sensors)
                      ATTRIBUTES: INTENT(IN)

OUTPUTS:
      RTSolution:     Structure containing the layer aerosol optical
                      profile for the given inputs.
                      UNITS:      N/A
                      TYPE:       CRTM_RTSolution_type
                      DIMENSION:  Rank-2 (n_Channels x n_Profiles)
                      ATTRIBUTES: INTENT(IN OUT)

      RTSolution_TL:  Structure containing the tangent-linear aerosol
                      optical depth profile for the given inputs.
                      UNITS:      N/A
                      TYPE:       CRTM_RTSolution_type
                      DIMENSION:  Same as RTSolution output
                      ATTRIBUTES: INTENT(IN OUT)

OPTIONAL INPUTS:
      Options:        Options structure containing the optional arguments
                      for the CRTM.
                      UNITS:      N/A
                      TYPE:       CRTM_Options_type
                      DIMENSION:  Same as input Atmosphere structure
                      ATTRIBUTES: INTENT(IN), OPTIONAL

FUNCTION RESULT:
      Error_Status:   The return value is an integer defining the error status.
                      The error codes are defined in the Message_Handler module.
                      If == SUCCESS the computation was sucessful
                         == FAILURE an unrecoverable error occurred
                      UNITS:      N/A
                      TYPE:       INTEGER
                      DIMENSION:  Scalar

COMMENTS:
      - Many of the components of the Options optional input structure
        are not used in this function. Consult the CRTM User Guide for
        which Options components are usable for AOD calculations.
```

5.3.3 CRTM_AOD_AD interface

```
NAME:
      CRTM_AOD_AD

PURPOSE:
      Function that calculates the adjoint nadir aerosol optical depth.

CALLING SEQUENCE:
      Error_Status = CRTM_AOD_AD( Atmosphere       , &
                                  RTSolution_AD    , &
                                  ChannelInfo      , &
                                  RTSolution       , &
                                  Atmosphere_AD    , &
                                  Options = Options  )

INPUTS:
      Atmosphere:     Structure containing the Atmosphere data.
                      UNITS:      N/A
                      TYPE:       CRTM_Atmosphere_type
                      DIMENSION:  Rank-1 (n_Profiles)
                      ATTRIBUTES: INTENT(IN)

      RTSolution_AD:  Structure containing the RT solution adjoint inputs.
                      **NOTE: On EXIT from this function, the contents of
                              this structure may be modified (e.g. set to
                              zero.)
                      UNITS:      N/A
                      TYPE:       CRTM_RTSolution_type
                      DIMENSION:  Rank-2 (n_Channels x n_Profiles)
                      ATTRIBUTES: INTENT(IN OUT)

      ChannelInfo:    Structure returned from the CRTM_Init() function
                      that contains the satellite/sensor channel index
                      information.
                      UNITS:      N/A
                      TYPE:       CRTM_ChannelInfo_type
                      DIMENSION:  Rank-1 (n_Sensors)
                      ATTRIBUTES: INTENT(IN)

OUTPUTS:
      RTSolution:     Structure containing the soluition to the RT equation
                      for the given inputs.
                      UNITS:      N/A
                      TYPE:       CRTM_RTSolution_type
                      DIMENSION:  Rank-2 (n_Channels x n_Profiles)
                      ATTRIBUTES: INTENT(IN OUT)

      Atmosphere_AD:  Structure containing the adjoint Atmosphere data.
                      **NOTE: On ENTRY to this function, the contents of
                              this structure should be defined (e.g.
                              initialized to some value based on the
                              position of this function in the call chain.)
                      UNITS:      N/A
                      TYPE:       CRTM_Atmosphere_type
                      DIMENSION:  Same as input Atmosphere argument
                      ATTRIBUTES: INTENT(IN OUT)

OPTIONAL INPUTS:
      Options:        Options structure containing the optional arguments
                      for the CRTM.
                      UNITS:      N/A
                      TYPE:       CRTM_Options_type
                      DIMENSION:  Same as input Atmosphere structure
                      ATTRIBUTES: INTENT(IN), OPTIONAL

FUNCTION RESULT:
      Error_Status:   The return value is an integer defining the error status.
                      The error codes are defined in the Message_Handler module.
                      If == SUCCESS the computation was sucessful
                         == FAILURE an unrecoverable error occurred
                      UNITS:      N/A
                      TYPE:       INTEGER
                      DIMENSION:  Scalar

COMMENTS:
      - Many of the components of the Options optional input structure
        are not used in this function. Consult the CRTM User Guide for
        which Options components are usable for AOD calculations.
```

5.3.4 CRTM_AOD_K interface

```
NAME:
      CRTM_AOD_K

PURPOSE:
      Function that calculates the K-matrix nadir aerosol optical depth.

CALLING SEQUENCE:
      Error_Status = CRTM_AOD_K( Atmosphere        , &
                                 RTSolution_K      , &
                                 ChannelInfo       , &
                                 RTSolution        , &
                                 Atmosphere_K      , &
                                 Opttions = Options  )

INPUTS:
      Atmosphere:     Structure containing the Atmosphere data.
                      UNITS:      N/A
                      TYPE:       CRTM_Atmosphere_type
                      DIMENSION:  Rank-1 (n_Profiles)
                      ATTRIBUTES: INTENT(IN)

      RTSolution_K:   Structure containing the aerosol optical depth
                      profile K-matrix input.
                      **NOTE: On EXIT from this function, the contents of
                              this structure may be modified (e.g. set to
                              zero.)
                      UNITS:      N/A
                      TYPE:       CRTM_RTSolution_type
                      DIMENSION:  Rank-2 (n_Channels x n_Profiles)
                      ATTRIBUTES: INTENT(IN OUT)

      ChannelInfo:    Structure returned from the CRTM_Init() function
                      that contains the satellite/sensor channel index
                      information.
                      UNITS:      N/A
                      TYPE:       CRTM_ChannelInfo_type
                      DIMENSION:  Rank-1 (n_Sensors)
                      ATTRIBUTES: INTENT(IN)

OUTPUTS:
      RTSolution:     Structure containing the layer aerosol optical
                      depth profile for the given inputs.
                      UNITS:      N/A
                      TYPE:       CRTM_RTSolution_type
                      DIMENSION:  Rank-2 (n_Channels x n_Profiles)
                      ATTRIBUTES: INTENT(IN OUT)

      Atmosphere_K:   Structure containing the K-matrix Atmosphere data.
                      **NOTE: On ENTRY to this function, the contents of
                              this structure should be defined (e.g.
                              initialized to some value based on the
```

```
                        position of this function in the call chain.)
                    UNITS:      N/A
                    TYPE:       CRTM_Atmosphere_type
                    DIMENSION:  Same as input RTSolution_K argument
                    ATTRIBUTES: INTENT(IN OUT)

OPTIONAL INPUTS:
      Options:      Options structure containing the optional arguments
                    for the CRTM.
                    UNITS:      N/A
                    TYPE:       CRTM_Options_type
                    DIMENSION:  Same as input Atmosphere structure
                    ATTRIBUTES: INTENT(IN), OPTIONAL

FUNCTION RESULT:
      Error_Status:   The return value is an integer defining the error status.
                    The error codes are defined in the Message_Handler module.
                    If == SUCCESS the computation was sucessful
                       == FAILURE an unrecoverable error occurred
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Scalar

COMMENTS:
      - Many of the components of the Options optional input structure
        are not used in this function. Consult the CRTM User Guide for
        which Options components are usable for AOD calculations.
```

5.4 Destruction functions

5.4.1 CRTM_Destroy interface

```
NAME:
      CRTM_Destroy

PURPOSE:
      Function to deallocate all the shared data arrays allocated and
      populated during the CRTM initialization.

CALLING SEQUENCE:
      Error_Status = CRTM_Destroy( ChannelInfo             , &
                                   Process_ID = Process_ID  )

OUTPUTS:
      ChannelInfo:  Reinitialized ChannelInfo structure.
                    UNITS:      N/A
                    TYPE:       CRTM_ChannelInfo_type
                    DIMENSION:  Rank-1
                    ATTRIBUTES: INTENT(IN OUT)

OPTIONAL INPUTS:
      Process_ID:   Set this argument to the MPI process ID that this
                    function call is running under. This value is used
                    solely for controlling message output. If MPI is not
                    being used, ignore this argument.
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN), OPTIONAL

FUNCTION RESULT:
      Error_Status: The return value is an integer defining the error
                    status. The error codes are defined in the
                    Message_Handler module.
                    If == SUCCESS the CRTM deallocations were successful
                       == FAILURE an unrecoverable error occurred.
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Scalar

SIDE EFFECTS:
      All CRTM shared data arrays and structures are deallocated.

COMMENTS:
      Note the INTENT on the output ChannelInfo argument is IN OUT rather than
      just OUT. This is necessary because the argument may be defined upon
      input. To prevent memory leaks, the IN OUT INTENT is a must.
```

5.5 Utility functions

5.5.1 CRTM_Version interface

```
NAME:
      CRTM_Version

PURPOSE:
      Subroutine to the CRTM version information.

CALLING SEQUENCE:
      CALL CRTM_Version( version )

OUTPUTS:
      version:       Character string identifying the CRTM release version.
                     UNITS:      N/A
                     TYPE:       CHARACTER(*)
                     DIMENSION:  Scalar
                     ATTRIBUTES: INTENT(OUT)
```

5.5.2 CRTM_IsInitialized interface

```
NAME:
      CRTM_IsInitialized

PURPOSE:
      Logical function to test if the CRTM has been correctly initialized.

CALLING SEQUENCE:
      status = CRTM_IsInitialized( ChannelInfo )

INPUTS:
      ChannelInfo:  ChannelInfo structure array.
                    UNITS:      N/A
                    TYPE:       CRTM_ChannelInfo_type
                    DIMENSION:  Rank-1
                    ATTRIBUTES: INTENT(IN)

FUNCTION RESULT:
      Status:       The return value is a logical result indicating if the
                    CRTM has been correctly initialised.
                    If == .TRUE., all the ChannelInfo entries are valid.
                       == .FALSE., any of the ChannelInfo entries are invalid.
                    UNITS:      N/A
                    TYPE:       LOGICAL
                    DIMENSION:  Scalar
```

5.5.3 CRTM_LifeCycleVersion interface

```
NAME:
      CRTM_LifeCycleVersion

PURPOSE:
      Subroutine to return the module version information.

CALLING SEQUENCE:
      CALL CRTM_LifeCycleVersion( Id )

OUTPUT ARGUMENTS:
      Id:            Character string containing the version Id information
                     for the module.
                     UNITS:      N/A
                     TYPE:       CHARACTER(*)
                     DIMENSION:  Scalar
                     ATTRIBUTES: INTENT(OUT)
```

5.5.4 CRTM_Forward_Version interface

```
NAME:
      CRTM_Forward_Version

PURPOSE:
      Subroutine to return the module version information.

CALLING SEQUENCE:
      CALL CRTM_Forward_Version( Id )

OUTPUTS:
      Id:            Character string containing the version Id information
                     for the module.
                     UNITS:      N/A
                     TYPE:       CHARACTER(*)
                     DIMENSION:  Scalar
                     ATTRIBUTES: INTENT(OUT)
```

5.5.5 CRTM_Tangent_Linear_Version interface

```
NAME:
      CRTM_Tangent_Linear_Version

PURPOSE:
      Subroutine to return the module version information.
```

```
CALLING SEQUENCE:
      CALL CRTM_Tangent_Linear_Version( Id )

OUTPUTS:
      Id:            Character string containing the version Id information
                     for the module.
                     UNITS:      N/A
                     TYPE:       CHARACTER(*)
                     DIMENSION:  Scalar
                     ATTRIBUTES: INTENT(OUT)
```

5.5.6 CRTM_Adjoint_Version interface

```
NAME:
      CRTM_Adjoint_Version

PURPOSE:
      Subroutine to return the module version information.

CALLING SEQUENCE:
      CALL CRTM_Adjoint_Version( Id )

OUTPUTS:
      Id:            Character string containing the version Id information
                     for the module.
                     UNITS:      N/A
                     TYPE:       CHARACTER(*)
                     DIMENSION:  Scalar
                     ATTRIBUTES: INTENT(OUT)
```

5.5.7 CRTM_K_Matrix_Version interface

```
NAME:
      CRTM_K_Matrix_Version

PURPOSE:
      Subroutine to return the module version information.

CALLING SEQUENCE:
      CALL CRTM_K_Matrix_Version( Id )

OUTPUTS:
      Id:            Character string containing the version Id information
                     for the module.
                     UNITS:      N/A
```

```
                      TYPE:       CHARACTER(*)
                      DIMENSION:  Scalar
                      ATTRIBUTES: INTENT(OUT)
```

5.5.8 CRTM_AOD_Version interface

```
NAME:
      CRTM_AOD_Version

PURPOSE:
      Subroutine to return the module version information.

CALLING SEQUENCE:
      CALL CRTM_AOD_Version( Id )

OUTPUTS:
      Id:            Character string containing the version Id information
                     for the module.
                     UNITS:      N/A
                     TYPE:       CHARACTER(*)
                     DIMENSION:  Scalar
                     ATTRIBUTES: INTENT(OUT)
```

Bibliography

S. English and T. Hewison. A fast generic millimeter-wave emissivity model. In *Microwave Remote Sensing of the Atmosphere and Environment*, volume 3503, pages 288–300. SPIE, Sep. 1998. doi: 10.1117/12.319490.

A.K. Heidinger, C. O'Dell, R. Bennartz, and T. Greenwald. The Successive-Order-of-Interaction Radiative Transfer Model. Part I: Model Development. *J. Appl. Meteorol. Climatol.*, 45(10):1388–1402, 2006. doi: 10.1175/JAM2387.1.

M. Kazumori, Q. Liu, R. Treadon, and J.C. Derber. Impact study of AMSR-E radiances in the NCEP Global Data Assimilation System. *Mon. Wea. Rev.*, 136(2):541–559, 2008. doi: 10.1175/2007MWR2147.1.

Q. Liu and E. Ruprecht. Radiative transfer model: matrix operator method. *Appl. Opt.*, 35(21):4229–4237, 1996.

Q. Liu and F. Weng. Advanced doubling-adding method for radiative transfer in planetary atmosphere. *J. Atmos. Sci.*, 63(12):3459–3465, 2006. doi: 10.1175/JAS3808.1.

A
Structure and procedure interface definitions

A.1 `ChannelInfo` Structure

```
TYPE :: CRTM_ChannelInfo_type
  ! Allocation indicator
  LOGICAL :: Is_Allocated = .FALSE.
  ! Dimensions
  INTEGER :: n_Channels = 0  ! L dimension
  ! Scalar data
  CHARACTER(STRLEN) :: Sensor_ID        = ''
  INTEGER           :: Sensor_Type      = INVALID_SENSOR
  INTEGER           :: WMO_Satellite_ID = INVALID_WMO_SATELLITE_ID
  INTEGER           :: WMO_Sensor_ID    = INVALID_WMO_SENSOR_ID
  INTEGER           :: Sensor_Index     = 0
  ! Array data
  LOGICAL, ALLOCATABLE :: Process_Channel(:)  ! L
  INTEGER, ALLOCATABLE :: Sensor_Channel(:)   ! L
  INTEGER, ALLOCATABLE :: Channel_Index(:)    ! L
END TYPE CRTM_ChannelInfo_type
```

Figure A.1: CRTM_ChannelInfo_type structure definition.

A.1.1 CRTM_ChannelInfo_Associated interface

```
NAME:
      CRTM_ChannelInfo_Associated

PURPOSE:
      Elemental function to test the status of the allocatable components
      of a CRTM ChannelInfo object.

CALLING SEQUENCE:
      Status = CRTM_ChannelInfo_Associated( ChannelInfo )

OBJECTS:
      ChannelInfo:   ChannelInfo object which is to have its member's
                     status tested.
                     UNITS:      N/A
                     TYPE:       TYPE(CRTM_ChannelInfo_type)
                     DIMENSION:  Scalar or any rank
                     ATTRIBUTES: INTENT(IN)

FUNCTION RESULT:
      Status:        The return value is a logical value indicating the
                     status of the ChannelInfo members.
                       .TRUE.  - if the array components are allocated.
                       .FALSE. - if the array components are not allocated.
                     UNITS:      N/A
                     TYPE:       LOGICAL
                     DIMENSION:  Same as input ChannelInfo argument
```

A.1.2 CRTM_ChannelInfo_Channels interface

```
NAME:
      CRTM_ChannelInfo_Channels

PURPOSE:
      Pure function to return the list of channels to be processed in a
      ChannelInfo object.

CALLING SEQUENCE:
      Channels = CRTM_ChannelInfo_Channels( ChannelInfo )

OBJECTS:
      ChannelInfo: ChannelInfo object which is to have its channel list queried.
                   UNITS:      N/A
                   TYPE:       TYPE(CRTM_ChannelInfo_type)
                   DIMENSION:  Scalar
                   ATTRIBUTES: INTENT(IN)

FUNCTION RESULT:
      Channels:    The list of channels to be processed in the ChannelInfo
                   object.
                   UNITS:      N/A
                   TYPE:       INTEGER
                   DIMENSION:  Rank-1
```

A.1.3 CRTM_ChannelInfo_DefineVersion interface

```
NAME:
      CRTM_ChannelInfo_DefineVersion

PURPOSE:
      Subroutine to return the module version information.

CALLING SEQUENCE:
      CALL CRTM_ChannelInfo_DefineVersion( Id )

OUTPUTS:
      Id:    Character string containing the version Id information
             for the module.
             UNITS:      N/A
             TYPE:       CHARACTER(*)
             DIMENSION:  Scalar
             ATTRIBUTES: INTENT(OUT)
```

A.1.4 CRTM_ChannelInfo_Destroy interface

```
NAME:
      CRTM_ChannelInfo_Destroy

PURPOSE:
      Elemental subroutine to re-initialize CRTM ChannelInfo objects.

CALLING SEQUENCE:
      CALL CRTM_ChannelInfo_Destroy( ChannelInfo )

OBJECTS:
      ChannelInfo:    Re-initialized ChannelInfo object.
                      UNITS:      N/A
                      TYPE:       TYPE(CRTM_ChannelInfo_type)
                      DIMENSION:  Scalar or any rank
                      ATTRIBUTES: INTENT(OUT)
```

A.1.5 CRTM_ChannelInfo_Inspect interface

```
NAME:
      CRTM_ChannelInfo_Inspect

PURPOSE:
      Subroutine to print the contents of a CRTM ChannelInfo object
      to stdout.

CALLING SEQUENCE:
      CALL CRTM_ChannelInfo_Inspect( ChannelInfo )

OBJECTS:
      ChannelInfo:   ChannelInfo object to display.
                     UNITS:      N/A
                     TYPE:       TYPE(CRTM_ChannelInfo_type)
                     DIMENSION:  Scalar
                     ATTRIBUTES: INTENT(IN)
```

A.1.6 CRTM_ChannelInfo_Subset interface

```
NAME:
      CRTM_ChannelInfo_Subset

PURPOSE:
      Function to specify a channel subset for processing in the CRTM.
      By default, ALL channels are processed. This function allows the
      list of channels that are to be processed to be altered.

CALLING SEQUENCE:
      Error_Status = CRTM_ChannelInfo_Subset( ChannelInfo   , &
                                              Channel_Subset, &
                                              Reset           )

OBJECTS:
      ChannelInfo:    Valid ChannelInfo object for which a channel subset is
                      to be specified.
                      UNITS:      N/A
                      TYPE:       TYPE(CRTM_ChannelInfo_type)
                      DIMENSION:  Scalar
                      ATTRIBUTES: INTENT(IN OUT)

OPTIONAL INPUTS:
      Channel_Subset: An integer array containing the subset list of channels.
                      Future calls to the CRTM main functions using the passed
                      ChannelInfo object will process ONLY the channels
                      specified in this list.
                      *** NOTE: This argument is ignored if the Reset optional
                      *** argument is specified with a .TRUE. value.
```

```
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Rank-1
                    ATTRIBUTES: INTENT(IN), OPTIONAL

      Reset:        Logical flag to reset the ChannelInfo object channel
                    processing subset to ALL channels.
                    If == .TRUE.  Future calls to the CRTM main functions using
                                  the passed ChannelInfo object will process
                                  ALL the channels
                       == .FALSE. Procedure execution is equivalent to the Reset
                                  argument not being specified at all.
                    UNITS:      N/A
                    TYPE:       LOGICAL
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN), OPTIONAL

FUNCTION RESULT:
      Error_Status: The return value is an integer defining the error status.
                    The error codes are defined in the Message_Handler module.
                    If == SUCCESS the channel subset setting was sucessful
                       == FAILURE an error occurred
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Scalar

COMMENTS:
      - The ChannelInfo object can be modified by this procedure.
      - An error in this procedure will DISABLE processing for ALL channels.
```

A.1.7 CRTM_ChannelInfo_n_Channels interface

```
NAME:
      CRTM_ChannelInfo_n_Channels

PURPOSE:
      Elemental function to return the number of channels flagged for
      processing in a ChannelInfo object.

CALLING SEQUENCE:
      n_Channels = CRTM_ChannelInfo_n_Channels( ChannelInfo )

OBJECTS:
      ChannelInfo: ChannelInfo object which is to have its processed
                   channels counted.
                   UNITS:      N/A
                   TYPE:       TYPE(CRTM_ChannelInfo_type)
                   DIMENSION:  Scalar or any rank
                   ATTRIBUTES: INTENT(IN)
```

```
FUNCTION RESULT:
      n_Channels:  The number of channels to be processed in the ChannelInfo
                   object.
                   UNITS:      N/A
                   TYPE:       INTEGER
                   DIMENSION:  Same as input ChannelInfo argument.
```

A.2 Atmosphere Structure

```
TYPE :: CRTM_Atmosphere_type
  ! Allocation indicator
  LOGICAL :: Is_Allocated = .FALSE.
  ! Dimension values
  INTEGER :: Max_Layers   = 0  ! K dimension
  INTEGER :: n_Layers     = 0  ! Kuse dimension
  INTEGER :: n_Absorbers  = 0  ! J dimension
  INTEGER :: Max_Clouds   = 0  ! Nc dimension
  INTEGER :: n_Clouds     = 0  ! NcUse dimension
  INTEGER :: Max_Aerosols = 0  ! Na dimension
  INTEGER :: n_Aerosols   = 0  ! NaUse dimension
  ! Number of added layers
  INTEGER :: n_Added_Layers = 0
  ! Climatology model associated with the profile
  INTEGER :: Climatology = US_STANDARD_ATMOSPHERE
  ! Absorber ID and units
  INTEGER, ALLOCATABLE :: Absorber_ID(:)    ! J
  INTEGER, ALLOCATABLE :: Absorber_Units(:) ! J
  ! Profile LEVEL and LAYER quantities
  REAL(fp), ALLOCATABLE :: Level_Pressure(:)  ! 0:K
  REAL(fp), ALLOCATABLE :: Pressure(:)        ! K
  REAL(fp), ALLOCATABLE :: Temperature(:)     ! K
  REAL(fp), ALLOCATABLE :: Absorber(:,:)      ! K x J
  ! Clouds associated with each profile
  TYPE(CRTM_Cloud_type),   ALLOCATABLE :: Cloud(:)    ! Nc
  ! Aerosols associated with each profile
  TYPE(CRTM_Aerosol_type), ALLOCATABLE :: Aerosol(:)  ! Na
END TYPE CRTM_Atmosphere_type
```

Figure A.2: CRTM_Atmosphere_type structure definition.

A.2.1 CRTM_Atmosphere_AddLayerCopy interface

```
NAME:
      CRTM_Atmosphere_AddLayerCopy

PURPOSE:
      Elemental function to copy an instance of the CRTM Atmosphere object
      with additional layers added to the TOA of the input.

CALLING SEQUENCE:
      Atm_out = CRTM_Atmosphere_AddLayerCopy( Atm, n_Added_Layers )

OBJECTS:
      Atm:              Atmosphere structure to copy.
                        UNITS:      N/A
                        TYPE:       CRTM_Atmosphere_type
                        DIMENSION:  Scalar or any rank
                        ATTRIBUTES: INTENT(OUT)

INPUTS:
      n_Added_Layers:   Number of layers to add to the function result.
                        UNITS:      N/A
                        TYPE:       INTEGER
                        DIMENSION:  Same as atmosphere object
                        ATTRIBUTES: INTENT(IN)

FUNCTION RESULT:
      Atm_out:          Copy of the input atmosphere structure with space for
                        extra layers added to TOA.
                        UNITS:      N/A
                        TYPE:       CRTM_Atmosphere_type
                        DIMENSION:  Same as input.
                        ATTRIBUTES: INTENT(OUT)
```

A.2.2 CRTM_Atmosphere_Associated interface

```
NAME:
      CRTM_Atmosphere_Associated

PURPOSE:
      Elemental function to test the status of the allocatable components
      of a CRTM Atmosphere object.

CALLING SEQUENCE:
      Status = CRTM_Atmosphere_Associated( Atm )

OBJECTS:
      Atm:       Atmosphere structure which is to have its member's
```

```
                status tested.
                UNITS:      N/A
                TYPE:       CRTM_Atmosphere_type
                DIMENSION:  Scalar or any rank
                ATTRIBUTES: INTENT(IN)

FUNCTION RESULT:
      Status:   The return value is a logical value indicating the
                status of the Atmosphere members.
                  .TRUE.  - if the array components are allocated.
                  .FALSE. - if the array components are not allocated.
                UNITS:      N/A
                TYPE:       LOGICAL
                DIMENSION:  Same as input
```

A.2.3 CRTM_Atmosphere_Compare interface

```
NAME:
      CRTM_Atmosphere_Compare

PURPOSE:
      Elemental function to compare two CRTM_Atmosphere objects to within
      a user specified number of significant figures.

CALLING SEQUENCE:
      is_comparable = CRTM_Atmosphere_Compare( x, y, n_SigFig=n_SigFig )

OBJECTS:
      x, y:         Two CRTM Atmosphere objects to be compared.
                    UNITS:      N/A
                    TYPE:       CRTM_Atmosphere_type
                    DIMENSION:  Scalar or any rank
                    ATTRIBUTES: INTENT(IN)

OPTIONAL INPUTS:
      n_SigFig:     Number of significant figure to compare floating point
                    components.
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Scalar or same as input
                    ATTRIBUTES: INTENT(IN), OPTIONAL

FUNCTION RESULT:
      is_equal:     Logical value indicating whether the inputs are equal.
                    UNITS:      N/A
                    TYPE:       LOGICAL
                    DIMENSION:  Same as inputs.
```

A.2.4 CRTM_Atmosphere_Create interface

```
NAME:
      CRTM_Atmosphere_Create

PURPOSE:
      Elemental subroutine to create an instance of the CRTM Atmosphere object.

CALLING SEQUENCE:
      CALL CRTM_Atmosphere_Create( Atm        , &
                                   n_Layers   , &
                                   n_Absorbers, &
                                   n_Clouds   , &
                                   n_Aerosols   )

OBJECTS:
      Atm:           Atmosphere structure.
                     UNITS:      N/A
                     TYPE:       CRTM_Atmosphere_type
                     DIMENSION:  Scalar or any rank
                     ATTRIBUTES: INTENT(OUT)

INPUTS:
      n_Layers:      Number of layers dimension.
                     Must be > 0.
                     UNITS:      N/A
                     TYPE:       INTEGER
                     DIMENSION:  Same as atmosphere object
                     ATTRIBUTES: INTENT(IN)

      n_Absorbers:   Number of absorbers dimension.
                     Must be > 0.
                     UNITS:      N/A
                     TYPE:       INTEGER
                     DIMENSION:  Same as atmosphere object
                     ATTRIBUTES: INTENT(IN)

      n_Clouds:      Number of clouds dimension.
                     Can be = 0 (i.e. clear sky).
                     UNITS:      N/A
                     TYPE:       INTEGER
                     DIMENSION:  Same as atmosphere object
                     ATTRIBUTES: INTENT(IN)

      n_Aerosols:    Number of aerosols dimension.
                     Can be = 0 (i.e. no aerosols).
                     UNITS:      N/A
                     TYPE:       INTEGER
                     DIMENSION:  Same as atmosphere object
                     ATTRIBUTES: INTENT(IN)
```

A.2.5 CRTM_Atmosphere_DefineVersion interface

```
NAME:
      CRTM_Atmosphere_DefineVersion

PURPOSE:
      Subroutine to return the module version information.

CALLING SEQUENCE:
      CALL CRTM_Atmosphere_DefineVersion( Id )

OUTPUTS:
      Id:            Character string containing the version Id information
                     for the module.
                     UNITS:      N/A
                     TYPE:       CHARACTER(*)
                     DIMENSION:  Scalar
                     ATTRIBUTES: INTENT(OUT)
```

A.2.6 CRTM_Atmosphere_Destroy interface

```
NAME:
      CRTM_Atmosphere_Destroy

PURPOSE:
      Elemental subroutine to re-initialize CRTM Atmosphere objects.

CALLING SEQUENCE:
      CALL CRTM_Atmosphere_Destroy( Atm )

OBJECTS:
      Atm:          Re-initialized Atmosphere structure.
                    UNITS:      N/A
                    TYPE:       CRTM_Atmosphere_type
                    DIMENSION:  Scalar or any rank
                    ATTRIBUTES: INTENT(OUT)
```

A.2.7 CRTM_Atmosphere_InquireFile interface

```
NAME:
      CRTM_Atmosphere_InquireFile

PURPOSE:
      Function to inquire CRTM Atmosphere object files.
```

```
CALLING SEQUENCE:
      Error_Status = CRTM_Atmosphere_InquireFile( Filename               , &
                                                  n_Channels = n_Channels, &
                                                  n_Profiles = n_Profiles  )

INPUTS:
      Filename:       Character string specifying the name of a
                      CRTM Atmosphere data file to read.
                      UNITS:      N/A
                      TYPE:       CHARACTER(*)
                      DIMENSION:  Scalar
                      ATTRIBUTES: INTENT(IN)

OPTIONAL OUTPUTS:
      n_Channels:     The number of spectral channels for which there is
                      data in the file. Note that this value will always
                      be 0 for a profile-only dataset-- it only has meaning
                      for K-matrix data.
                      UNITS:      N/A
                      TYPE:       INTEGER
                      DIMENSION:  Scalar
                      ATTRIBUTES: OPTIONAL, INTENT(OUT)

      n_Profiles:     The number of profiles in the data file.
                      UNITS:      N/A
                      TYPE:       INTEGER
                      DIMENSION:  Scalar
                      ATTRIBUTES: OPTIONAL, INTENT(OUT)

FUNCTION RESULT:
      Error_Status:   The return value is an integer defining the error status.
                      The error codes are defined in the Message_Handler module.
                      If == SUCCESS, the file inquire was successful
                         == FAILURE, an unrecoverable error occurred.
                      UNITS:      N/A
                      TYPE:       INTEGER
                      DIMENSION:  Scalar
```

A.2.8 CRTM_Atmosphere_Inspect interface

```
NAME:
      CRTM_Atmosphere_Inspect

PURPOSE:
      Subroutine to print the contents of a CRTM Atmosphere object to stdout.

CALLING SEQUENCE:
      CALL CRTM_Atmosphere_Inspect( Atm )

INPUTS:
      Atm:  CRTM Atmosphere object to display.
            UNITS:      N/A
            TYPE:       CRTM_Atmosphere_type
            DIMENSION:  Scalar, Rank-1, or Rank-2 array
            ATTRIBUTES: INTENT(IN)
```

A.2.9 CRTM_Atmosphere_IsValid interface

```
NAME:
      CRTM_Atmosphere_IsValid

PURPOSE:
      Non-pure function to perform some simple validity checks on a
      CRTM Atmosphere object.

      If invalid data is found, a message is printed to stdout.

CALLING SEQUENCE:
      result = CRTM_Atmosphere_IsValid( Atm )

        or

      IF ( CRTM_Atmosphere_IsValid( Atm ) ) THEN....

OBJECTS:
      Atm:       CRTM Atmosphere object which is to have its
                 contents checked.
                 UNITS:      N/A
                 TYPE:       CRTM_Atmosphere_type
                 DIMENSION:  Scalar
                 ATTRIBUTES: INTENT(IN)

FUNCTION RESULT:
      result:    Logical variable indicating whether or not the input
                 passed the check.
                 If == .FALSE., Atmosphere object is unused or contains
                                invalid data.
                    == .TRUE.,  Atmosphere object can be used in CRTM.
                 UNITS:      N/A
                 TYPE:       LOGICAL
                 DIMENSION:  Scalar
```

A.2.10 CRTM_Atmosphere_ReadFile interface

```
NAME:
      CRTM_Atmosphere_ReadFile
```

```
PURPOSE:
      Function to read CRTM Atmosphere object files.

CALLING SEQUENCE:
      Error_Status = CRTM_Atmosphere_ReadFile( Filename                , &
                                               Atmosphere              , &
                                               Quiet      = Quiet      , &
                                               n_Channels = n_Channels , &
                                               n_Profiles = n_Profiles , &

INPUTS:
      Filename:     Character string specifying the name of an
                    Atmosphere format data file to read.
                    UNITS:      N/A
                    TYPE:       CHARACTER(*)
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN)

OUTPUTS:
      Atmosphere:   CRTM Atmosphere object array containing the Atmosphere
                    data. Note the following meanings attributed to the
                    dimensions of the object array:
                    Rank-1: M profiles.
                            Only profile data are to be read in. The file
                            does not contain channel information. The
                            dimension of the structure is understood to
                            be the PROFILE dimension.
                    Rank-2: L channels  x  M profiles
                            Channel and profile data are to be read in.
                            The file contains both channel and profile
                            information. The first dimension of the
                            structure is the CHANNEL dimension, the second
                            is the PROFILE dimension. This is to allow
                            K-matrix structures to be read in with the
                            same function.
                    UNITS:      N/A
                    TYPE:       CRTM_Atmosphere_type
                    DIMENSION:  Rank-1 (M) or Rank-2 (L x M)
                    ATTRIBUTES: INTENT(OUT)

OPTIONAL INPUTS:
      Quiet:        Set this logical argument to suppress INFORMATION
                    messages being printed to stdout
                    If == .FALSE., INFORMATION messages are OUTPUT [DEFAULT].
                       == .TRUE.,  INFORMATION messages are SUPPRESSED.
                    If not specified, default is .FALSE.
                    UNITS:      N/A
                    TYPE:       LOGICAL
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN), OPTIONAL

OPTIONAL OUTPUTS:
      n_Channels:   The number of channels for which data was read. Note that
```

```
                  this value will always be 0 for a profile-only dataset--
                  it only has meaning for K-matrix data.
                  UNITS:      N/A
                  TYPE:       INTEGER
                  DIMENSION:  Scalar
                  ATTRIBUTES: OPTIONAL, INTENT(OUT)

      n_Profiles: The number of profiles for which data was read.
                  UNITS:      N/A
                  TYPE:       INTEGER
                  DIMENSION:  Scalar
                  ATTRIBUTES: OPTIONAL, INTENT(OUT)


FUNCTION RESULT:
      Error_Status: The return value is an integer defining the error status.
                  The error codes are defined in the Message_Handler module.
                  If == SUCCESS, the file read was successful
                     == FAILURE, an unrecoverable error occurred.
                  UNITS:      N/A
                  TYPE:       INTEGER
                  DIMENSION:  Scalar
```

A.2.11 CRTM_Atmosphere_SetLayers interface

```
NAME:
      CRTM_Atmosphere_SetLayers

PURPOSE:
      Elemental subroutine to set the working number of layers to use
      in a CRTM Atmosphere object.

CALLING SEQUENCE:
     CALL CRTM_Atmosphere_SetLayers( Atmosphere, n_Layers )

OBJECT:
      Atmosphere: CRTM Atmosphere object which is to have its working number
                  of layers updated.
                  UNITS:      N/A
                  TYPE:       CRTM_Atmosphere_type
                  DIMENSION:  Scalar or any rank
                  ATTRIBUTES: INTENT(IN OUT)

INPUTS:
      n_Layers:   The value to set the n_Layers component of the
                  Atmosphere object.
                  UNITS:      N/A
                  TYPE:       CRTM_Atmosphere_type
                  DIMENSION:  Conformable with the Atmosphere object argument
                  ATTRIBUTES: INTENT(IN)
```

```
COMMENTS:
      - The object is zeroed upon output.

      - If n_Layers <= Atmosphere%Max_Layers, then only the n_Layers dimension
        value of the object, as well as any contained objects, is changed.

      - If n_Layers > Atmosphere%Max_Layers, then the object is reallocated
        to the required number of layers. No other dimensions of the object
        or contained objects are altered.
```

A.2.12 CRTM_Atmosphere_WriteFile interface

```
NAME:
      CRTM_Atmosphere_WriteFile

PURPOSE:
      Function to write CRTM Atmosphere object files.

CALLING SEQUENCE:
      Error_Status = CRTM_Atmosphere_WriteFile( Filename     , &
                                                Atmosphere   , &
                                                Quiet = Quiet  )

INPUTS:
      Filename:     Character string specifying the name of the
                    Atmosphere format data file to write.
                    UNITS:      N/A
                    TYPE:       CHARACTER(*)
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN)

      Atmosphere:   CRTM Atmosphere object array containing the Atmosphere
                    data. Note the following meanings attributed to the
                    dimensions of the Atmosphere array:
                    Rank-1: M profiles.
                            Only profile data are to be read in. The file
                            does not contain channel information. The
                            dimension of the array is understood to
                            be the PROFILE dimension.
                    Rank-2: L channels  x  M profiles
                            Channel and profile data are to be read in.
                            The file contains both channel and profile
                            information. The first dimension of the
                            array is the CHANNEL dimension, the second
                            is the PROFILE dimension. This is to allow
                            K-matrix structures to be read in with the
                            same function.
                    UNITS:      N/A
                    TYPE:       CRTM_Atmosphere_type
                    DIMENSION:  Rank-1 (M) or Rank-2 (L x M)
                    ATTRIBUTES: INTENT(IN)

  OPTIONAL INPUTS:
        Quiet:      Set this logical argument to suppress INFORMATION
                    messages being printed to stdout
                    If == .FALSE., INFORMATION messages are OUTPUT [DEFAULT].
                       == .TRUE.,  INFORMATION messages are SUPPRESSED.
                    If not specified, default is .FALSE.
                    UNITS:      N/A
                    TYPE:       LOGICAL
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN), OPTIONAL

  FUNCTION RESULT:
        Error_Status: The return value is an integer defining the error status.
                    The error codes are defined in the Message_Handler module.
                    If == SUCCESS, the file write was successful
                       == FAILURE, an unrecoverable error occurred.
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Scalar

  SIDE EFFECTS:
        - If the output file already exists, it is overwritten.
        - If an error occurs during *writing*, the output file is deleted before
          returning to the calling routine.
```

A.2.13 CRTM_Atmosphere_Zero interface

```
  NAME:
        CRTM_Atmosphere_Zero

  PURPOSE:
        Elemental subroutine to zero out the data arrays
        in a CRTM Atmosphere object.

  CALLING SEQUENCE:
        CALL CRTM_Atmosphere_Zero( Atm )

  OUTPUTS:
        Atm:        CRTM Atmosphere structure in which the data arrays
                    are to be zeroed out.
                    UNITS:      N/A
                    TYPE:       CRTM_Atmosphere_type
                    DIMENSION:  Scalar or any rank
                    ATTRIBUTES: INTENT(IN OUT)

  COMMENTS:
        - The dimension components of the structure are *NOT* set to zero.
```

```
      - The Climatology, Absorber_ID, and Absorber_Units components are
        *NOT* reset in this routine.
```

A.2.14 CRTM_Get_AbsorberIdx interface

```
NAME:
      CRTM_Get_AbsorberIdx

PURPOSE:
      Function to determine the index of the requested absorber in the
      CRTM Atmosphere structure absorber component.

CALLING SEQUENCE:
      Idx = CRTM_Get_AbsorberIdx(Atm, AbsorberId)

INPUTS:
      Atm:           CRTM Atmosphere structure.
                     UNITS:      N/A
                     TYPE:       CRTM_Atmosphere_type
                     DIMENSION:  Scalar
                     ATTRIBUTES: INTENT(IN)

      AbsorberId:    Integer value used to identify absorbing molecular
                     species. The accepted absorber Ids are defined in
                     this module.
                     UNITS:      N/A
                     TYPE:       INTEGER
                     DIMENSION:  Scalar
                     ATTRIBUTES: INTENT(IN)

FUNCTION RESULT:
      Idx:           Index of the requested absorber in the
                     Atm%Absorber array component.
                     If the requested absorber cannot be found,
                     a value of -1 is returned.
                     UNITS:      N/A
                     TYPE:       INTEGER
                     DIMENSION:  Scalar
```

A.2.15 CRTM_Get_PressureLevelIdx interface

```
NAME:
      CRTM_Get_PressureLevelIdx

PURPOSE:
      Function to determine the index in the CRTM Atmosphere structure
      pressure level array component that corresponds to the value
      closest to the requested level pressure.

CALLING SEQUENCE:
      Idx = CRTM_Get_PressureLevelIdx(Atm, Level_Pressure)

INPUTS:
      Atm:             CRTM Atmosphere structure.
                       UNITS:      N/A
                       TYPE:       CRTM_Atmosphere_type
                       DIMENSION:  Scalar
                       ATTRIBUTES: INTENT(IN)

      Level_Pressure:  Level pressure for which the index in the atmosphere
                       structure level pressure profile is required.
                       UNITS:      N/A
                       TYPE:       REAL(fp)
                       DIMENSION:  Scalar
                       ATTRIBUTES: INTENT(IN)

FUNCTION RESULT:
      Idx:             Index of the level in the Atm%Level_Pressure
                       array component for the closest value to the
                       input level pressure.
                       UNITS:      N/A
                       TYPE:       INTEGER
                       DIMENSION:  Scalar
```

A.3 `Cloud` Structure

```
TYPE :: CRTM_Cloud_type
  ! Allocation indicator
  LOGICAL :: Is_Allocated = .FALSE.
  ! Dimension values
  INTEGER :: Max_Layers = 0  ! K dimension.
  INTEGER :: n_Layers   = 0  ! Kuse dimension.
  ! Number of added layers
  INTEGER :: n_Added_Layers = 0
  ! Cloud type
  INTEGER :: Type = INVALID_CLOUD
  ! Cloud state variables
  REAL(fp), ALLOCATABLE :: Effective_Radius(:)   ! K. Units are microns
  REAL(fp), ALLOCATABLE :: Effective_Variance(:) ! K. Units are microns^2
  REAL(fp), ALLOCATABLE :: Water_Content(:)      ! K. Units are kg/m^2
END TYPE CRTM_Cloud_type
```

Figure A.3: CRTM_Cloud_type structure definition.

A.3.1 CRTM_Cloud_AddLayerCopy interface

```
NAME:
      CRTM_Cloud_AddLayerCopy

PURPOSE:
      Elemental function to copy an instance of the CRTM Cloud object
      with additional layers added to the TOA of the input.

CALLING SEQUENCE:
      cld_out = CRTM_Cloud_AddLayerCopy( cld, n_Added_Layers )

OBJECTS:
      cld:              Cloud structure to copy.
                        UNITS:      N/A
                        TYPE:       CRTM_Cloud_type
                        DIMENSION:  Scalar or any rank
                        ATTRIBUTES: INTENT(OUT)

INPUTS:
      n_Added_Layers:   Number of layers to add to the function result.
                        UNITS:      N/A
                        TYPE:       INTEGER
                        DIMENSION:  Same as Cloud object
                        ATTRIBUTES: INTENT(IN)

FUNCTION RESULT:
      cld_out:          Copy of the input Cloud structure with space for
                        extra layers added to TOA.
                        UNITS:      N/A
                        TYPE:       CRTM_Cloud_type
                        DIMENSION:  Same as input.
                        ATTRIBUTES: INTENT(OUT)
```

A.3.2 CRTM_Cloud_Associated interface

```
NAME:
      CRTM_Cloud_Associated

PURPOSE:
      Elemental function to test the status of the allocatable components
      of a CRTM Cloud object.

CALLING SEQUENCE:
      Status = CRTM_Cloud_Associated( Cloud )

OBJECTS:
      Cloud:   Cloud structure which is to have its member's
```

```
               status tested.
               UNITS:      N/A
               TYPE:       CRTM_Cloud_type
               DIMENSION:  Scalar or any rank
               ATTRIBUTES: INTENT(IN)

FUNCTION RESULT:
      Status:  The return value is a logical value indicating the
               status of the Cloud members.
                 .TRUE.  - if the array components are allocated.
                 .FALSE. - if the array components are not allocated.
               UNITS:      N/A
               TYPE:       LOGICAL
               DIMENSION:  Same as input Cloud argument
```

A.3.3 CRTM_Cloud_Compare interface

```
NAME:
      CRTM_Cloud_Compare

PURPOSE:
      Elemental function to compare two CRTM_Cloud objects to within
      a user specified number of significant figures.

CALLING SEQUENCE:
      is_comparable = CRTM_Cloud_Compare( x, y, n_SigFig=n_SigFig )

OBJECTS:
      x, y:          Two CRTM Cloud objects to be compared.
                     UNITS:      N/A
                     TYPE:       CRTM_Cloud_type
                     DIMENSION:  Scalar or any rank
                     ATTRIBUTES: INTENT(IN)

OPTIONAL INPUTS:
      n_SigFig:      Number of significant figure to compare floating point
                     components.
                     UNITS:      N/A
                     TYPE:       INTEGER
                     DIMENSION:  Scalar or same as input
                     ATTRIBUTES: INTENT(IN), OPTIONAL

FUNCTION RESULT:
      is_equal:      Logical value indicating whether the inputs are equal.
                     UNITS:      N/A
                     TYPE:       LOGICAL
                     DIMENSION:  Same as inputs.
```

A.3.4 CRTM_Cloud_Create interface

```
NAME:
      CRTM_Cloud_Create

PURPOSE:
      Elemental subroutine to create an instance of the CRTM Cloud object.

CALLING SEQUENCE:
      CALL CRTM_Cloud_Create( Cloud, n_Layers )

OBJECTS:
      Cloud:        Cloud structure.
                    UNITS:      N/A
                    TYPE:       CRTM_Cloud_type
                    DIMENSION:  Scalar or any rank
                    ATTRIBUTES: INTENT(OUT)

INPUTS:
      n_Layers:     Number of layers for which there is cloud data.
                    Must be > 0.
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Same as Cloud object
                    ATTRIBUTES: INTENT(IN)
```

A.3.5 CRTM_Cloud_DefineVersion interface

```
NAME:
      CRTM_Cloud_DefineVersion

PURPOSE:
      Subroutine to return the module version information.

CALLING SEQUENCE:
      CALL CRTM_Cloud_DefineVersion( Id )

OUTPUTS:
      Id:            Character string containing the version Id information
                     for the module.
                     UNITS:      N/A
                     TYPE:       CHARACTER(*)
                     DIMENSION:  Scalar
                     ATTRIBUTES: INTENT(OUT)
```

A.3.6 CRTM_Cloud_Destroy interface

```
NAME:
      CRTM_Cloud_Destroy

PURPOSE:
      Elemental subroutine to re-initialize CRTM Cloud objects.

CALLING SEQUENCE:
      CALL CRTM_Cloud_Destroy( Cloud )

OBJECTS:
      Cloud:        Re-initialized Cloud structure.
                    UNITS:      N/A
                    TYPE:       CRTM_Cloud_type
                    DIMENSION:  Scalar OR any rank
                    ATTRIBUTES: INTENT(OUT)
```

A.3.7 CRTM_Cloud_InquireFile interface

```
NAME:
      CRTM_Cloud_InquireFile

PURPOSE:
      Function to inquire CRTM Cloud object files.

CALLING SEQUENCE:
      Error_Status = CRTM_Cloud_InquireFile( Filename           , &
                                             n_Clouds = n_Clouds  )

INPUTS:
      Filename:       Character string specifying the name of a
                      CRTM Cloud data file to read.
                      UNITS:      N/A
                      TYPE:       CHARACTER(*)
                      DIMENSION:  Scalar
                      ATTRIBUTES: INTENT(IN)

OPTIONAL OUTPUTS:
      n_Clouds:       The number of Cloud profiles in the data file.
                      UNITS:      N/A
                      TYPE:       INTEGER
                      DIMENSION:  Scalar
                      ATTRIBUTES: OPTIONAL, INTENT(OUT)

FUNCTION RESULT:
      Error_Status:   The return value is an integer defining the error status.
                      The error codes are defined in the Message_Handler module.
                      If == SUCCESS, the file inquire was successful
```

```
                     == FAILURE, an unrecoverable error occurred.
                  UNITS:      N/A
                  TYPE:       INTEGER
                  DIMENSION:  Scalar
```

A.3.8 CRTM_Cloud_Inspect interface

```
NAME:
      CRTM_Cloud_Inspect

PURPOSE:
      Subroutine to print the contents of a CRTM Cloud object to stdout.

CALLING SEQUENCE:
      CALL CRTM_Cloud_Inspect( Cloud )

INPUTS:
      Cloud:         CRTM Cloud object to display.
                     UNITS:      N/A
                     TYPE:       CRTM_Cloud_type
                     DIMENSION:  Scalar, Rank-1, or Rank-2 array
                     ATTRIBUTES: INTENT(IN)
```

A.3.9 CRTM_Cloud_IsValid interface

```
NAME:
      CRTM_Cloud_IsValid

PURPOSE:
      Non-pure function to perform some simple validity checks on a
      CRTM Cloud object.

      If invalid data is found, a message is printed to stdout.

CALLING SEQUENCE:
      result = CRTM_Cloud_IsValid( cloud )

        or

      IF ( CRTM_Cloud_IsValid( cloud ) ) THEN....

OBJECTS:
      cloud:         CRTM Cloud object which is to have its
                     contents checked.
                     UNITS:      N/A
                     TYPE:       CRTM_Cloud_type
```

```
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN)

FUNCTION RESULT:
      result:       Logical variable indicating whether or not the input
                    passed the check.
                    If == .FALSE., Cloud object is unused or contains
                                   invalid data.
                       == .TRUE.,  Cloud object can be used in CRTM.
                    UNITS:      N/A
                    TYPE:       LOGICAL
                    DIMENSION:  Scalar
```

A.3.10 CRTM_Cloud_ReadFile interface

```
NAME:
      CRTM_Cloud_ReadFile

PURPOSE:
      Function to read CRTM Cloud object files.

CALLING SEQUENCE:
      Error_Status = CRTM_Cloud_ReadFile( Filename           , &
                                          Cloud              , &
                                          Quiet    = Quiet   , &
                                          No_Close = No_Close, &
                                          n_Clouds = n_Clouds  )

INPUTS:
      Filename:       Character string specifying the name of a
                      Cloud format data file to read.
                      UNITS:      N/A
                      TYPE:       CHARACTER(*)
                      DIMENSION:  Scalar
                      ATTRIBUTES: INTENT(IN)

OUTPUTS:
      Cloud:          CRTM Cloud object array containing the Cloud data.
                      UNITS:      N/A
                      TYPE:       CRTM_Cloud_type
                      DIMENSION:  Rank-1
                      ATTRIBUTES: INTENT(OUT)

OPTIONAL INPUTS:
      Quiet:          Set this logical argument to suppress INFORMATION
                      messages being printed to stdout
                      If == .FALSE., INFORMATION messages are OUTPUT [DEFAULT].
                         == .TRUE.,  INFORMATION messages are SUPPRESSED.
                      If not specified, default is .FALSE.
                      UNITS:      N/A
```

```
                     TYPE:       LOGICAL
                     DIMENSION:  Scalar
                     ATTRIBUTES: INTENT(IN), OPTIONAL

      No_Close:      Set this logical argument to NOT close the file upon exit.
                     If == .FALSE., the input file is closed upon exit [DEFAULT]
                        == .TRUE.,  the input file is NOT closed upon exit.
                     If not specified, default is .FALSE.
                     UNITS:      N/A
                     TYPE:       LOGICAL
                     DIMENSION:  Scalar
                     ATTRIBUTES: INTENT(IN), OPTIONAL

OPTIONAL OUTPUTS:
      n_Clouds:      The actual number of cloud profiles read in.
                     UNITS:      N/A
                     TYPE:       INTEGER
                     DIMENSION:  Scalar
                     ATTRIBUTES: OPTIONAL, INTENT(OUT)

FUNCTION RESULT:
      Error_Status:  The return value is an integer defining the error status.
                     The error codes are defined in the Message_Handler module.
                     If == SUCCESS, the file read was successful
                        == FAILURE, an unrecoverable error occurred.
                     UNITS:      N/A
                     TYPE:       INTEGER
                     DIMENSION:  Scalar
```

A.3.11 CRTM_Cloud_SetLayers interface

```
NAME:
      CRTM_Cloud_SetLayers

PURPOSE:
      Elemental subroutine to set the working number of layers to use
      in a CRTM Cloud object.

CALLING SEQUENCE:
     CALL CRTM_Cloud_SetLayers( Cloud, n_Layers )

OBJECT:
      Cloud:       CRTM Cloud object which is to have its working number
                   of layers updated.
                   UNITS:      N/A
                   TYPE:       CRTM_Cloud_type
                   DIMENSION:  Scalar or any rank
                   ATTRIBUTES: INTENT(IN OUT)

INPUTS:
      n_Layers:     The value to set the n_Layers component of the
                    Cloud object.
                    UNITS:      N/A
                    TYPE:       CRTM_Cloud_type
                    DIMENSION:  Conformable with the Cloud object argument
                    ATTRIBUTES: INTENT(IN)

COMMENTS:
      - The object is zeroed upon output.

      - If n_Layers <= Cloud%Max_Layers, then only the dimension value
        of the object is changed.

      - If n_Layers > Cloud%Max_Layers, then the object is reallocated
        to the required number of layers.
```

A.3.12 CRTM_Cloud_WriteFile interface

```
NAME:
      CRTM_Cloud_WriteFile

PURPOSE:
      Function to write CRTM Cloud object files.

CALLING SEQUENCE:
      Error_Status = CRTM_Cloud_WriteFile( Filename           , &
                                           Cloud              , &
                                           Quiet    = Quiet   , &
                                           No_Close = No_Close  )

INPUTS:
      Filename:       Character string specifying the name of the
                      Cloud format data file to write.
                      UNITS:      N/A
                      TYPE:       CHARACTER(*)
                      DIMENSION:  Scalar
                      ATTRIBUTES: INTENT(IN)

      Cloud:          CRTM Cloud object array containing the Cloud data.
                      UNITS:      N/A
                      TYPE:       CRTM_Cloud_type
                      DIMENSION:  Rank-1
                      ATTRIBUTES: INTENT(IN)

OPTIONAL INPUTS:
      Quiet:          Set this logical argument to suppress INFORMATION
                      messages being printed to stdout
                      If == .FALSE., INFORMATION messages are OUTPUT [DEFAULT].
                         == .TRUE.,  INFORMATION messages are SUPPRESSED.
                      If not specified, default is .FALSE.
```

```
                    UNITS:      N/A
                    TYPE:       LOGICAL
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN), OPTIONAL

      No_Close:     Set this logical argument to NOT close the file upon exit.
                    If == .FALSE., the input file is closed upon exit [DEFAULT]
                       == .TRUE.,  the input file is NOT closed upon exit.
                    If not specified, default is .FALSE.
                    UNITS:      N/A
                    TYPE:       LOGICAL
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN), OPTIONAL

FUNCTION RESULT:
      Error_Status: The return value is an integer defining the error status.
                    The error codes are defined in the Message_Handler module.
                    If == SUCCESS, the file write was successful
                       == FAILURE, an unrecoverable error occurred.
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Scalar

SIDE EFFECTS:
      - If the output file already exists, it is overwritten.
      - If an error occurs during *writing*, the output file is deleted before
        returning to the calling routine.
```

A.3.13 CRTM_Cloud_Zero interface

```
NAME:
      CRTM_Cloud_Zero

PURPOSE:
      Elemental subroutine to zero out the data arrays in a CRTM Cloud object.

CALLING SEQUENCE:
      CALL CRTM_Cloud_Zero( Cloud )

OBJECTS:
      Cloud:         CRTM Cloud structure in which the data arrays are
                     to be zeroed out.
                     UNITS:      N/A
                     TYPE:       CRTM_Cloud_type
                     DIMENSION:  Scalar or any rank
                     ATTRIBUTES: INTENT(IN OUT)

COMMENTS:
      - The dimension components of the structure are *NOT* set to zero.
      - The cloud type component is *NOT* reset.
```

A.4 Aerosol Structure

```
TYPE :: CRTM_Aerosol_type
  ! Allocation indicator
  LOGICAL :: Is_Allocated = .FALSE.
  ! Dimension values
  INTEGER :: Max_Layers = 0  ! K dimension.
  INTEGER :: n_Layers   = 0  ! Kuse dimension
  ! Number of added layers
  INTEGER :: n_Added_Layers = 0
  ! Aerosol type
  INTEGER :: Type = INVALID_AEROSOL
  ! Aerosol state variables
  REAL(fp), ALLOCATABLE :: Effective_Radius(:)    ! K. Units are microns
  REAL(fp), ALLOCATABLE :: Effective_Variance(:)  ! K. Units are microns, CMAQ only
  REAL(fp), ALLOCATABLE :: Concentration(:)       ! K. Units are kg/m^2
END TYPE CRTM_Aerosol_type
```

Figure A.4: CRTM_Aerosol_type structure definition.

A.4.1 CRTM_Aerosol_AddLayerCopy interface

```
NAME:
      CRTM_Aerosol_AddLayerCopy

PURPOSE:
      Elemental function to copy an instance of the CRTM Aerosol object
      with additional layers added to the TOA of the input.

CALLING SEQUENCE:
      aer_out = CRTM_Aerosol_AddLayerCopy( aer, n_Added_Layers )

OBJECTS:
      aer:             Aerosol structure to copy.
                       UNITS:      N/A
                       TYPE:       CRTM_Aerosol_type
                       DIMENSION:  Scalar or any rank
                       ATTRIBUTES: INTENT(OUT)

INPUTS:
      n_Added_Layers:  Number of layers to add to the function result.
                       UNITS:      N/A
                       TYPE:       INTEGER
                       DIMENSION:  Same as Aerosol object
                       ATTRIBUTES: INTENT(IN)

FUNCTION RESULT:
      aer_out:         Copy of the input Aerosol structure with space for
                       extra layers added to TOA.
                       UNITS:      N/A
                       TYPE:       CRTM_Aerosol_type
                       DIMENSION:  Same as input.
                       ATTRIBUTES: INTENT(OUT)
```

A.4.2 CRTM_Aerosol_Associated interface

```
NAME:
      CRTM_Aerosol_Associated

PURPOSE:
      Elemental function to test the status of the allocatable components
      of a CRTM Aerosol object.

CALLING SEQUENCE:
      Status = CRTM_Aerosol_Associated( Aerosol )

OBJECTS:
      Aerosol: Aerosol structure which is to have its member's
```

```
               status tested.
               UNITS:      N/A
               TYPE:       CRTM_Aerosol_type
               DIMENSION:  Scalar or any rank
               ATTRIBUTES: INTENT(IN)

FUNCTION RESULT:
      Status:  The return value is a logical value indicating the
               status of the Aerosol members.
                 .TRUE.  - if the array components are allocated.
                 .FALSE. - if the array components are not allocated.
               UNITS:      N/A
               TYPE:       LOGICAL
               DIMENSION:  Same as input Aerosol argument
```

A.4.3 CRTM_Aerosol_Compare interface

```
NAME:
      CRTM_Aerosol_Compare

PURPOSE:
      Elemental function to compare two CRTM_Aerosol objects to within
      a user specified number of significant figures.

CALLING SEQUENCE:
      is_comparable = CRTM_Aerosol_Compare( x, y, n_SigFig=n_SigFig )

OBJECTS:
      x, y:          Two CRTM Aerosol objects to be compared.
                     UNITS:      N/A
                     TYPE:       CRTM_Aerosol_type
                     DIMENSION:  Scalar or any rank
                     ATTRIBUTES: INTENT(IN)

OPTIONAL INPUTS:
      n_SigFig:      Number of significant figure to compare floating point
                     components.
                     UNITS:      N/A
                     TYPE:       INTEGER
                     DIMENSION:  Scalar or same as input
                     ATTRIBUTES: INTENT(IN), OPTIONAL

FUNCTION RESULT:
      is_equal:      Logical value indicating whether the inputs are equal.
                     UNITS:      N/A
                     TYPE:       LOGICAL
                     DIMENSION:  Same as inputs.
```

A.4.4 CRTM_Aerosol_Create interface

```
NAME:
      CRTM_Aerosol_Create

PURPOSE:
      Elemental subroutine to create an instance of the CRTM Aerosol object.

CALLING SEQUENCE:
      CALL CRTM_Aerosol_Create( Aerosol, n_Layers )

OBJECTS:
      Aerosol:      Aerosol structure.
                    UNITS:      N/A
                    TYPE:       CRTM_Aerosol_type
                    DIMENSION:  Scalar or any rank
                    ATTRIBUTES: INTENT(OUT)

INPUTS:
      n_Layers:     Number of layers for which there is Aerosol data.
                    Must be > 0.
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Same as Aerosol object
                    ATTRIBUTES: INTENT(IN)
```

A.4.5 CRTM_Aerosol_DefineVersion interface

```
NAME:
      CRTM_Aerosol_DefineVersion

PURPOSE:
      Subroutine to return the module version information.

CALLING SEQUENCE:
      CALL CRTM_Aerosol_DefineVersion( Id )

OUTPUTS:
      Id:   Character string containing the version Id information
            for the module.
            UNITS:      N/A
            TYPE:       CHARACTER(*)
            DIMENSION:  Scalar
            ATTRIBUTES: INTENT(OUT)
```

A.4.6 CRTM_Aerosol_Destroy interface

```
NAME:
      CRTM_Aerosol_Destroy

PURPOSE:
      Elemental subroutine to re-initialize CRTM Aerosol objects.

CALLING SEQUENCE:
      CALL CRTM_Aerosol_Destroy( Aerosol )

OBJECTS:
      Aerosol:      Re-initialized Aerosol structure.
                    UNITS:      N/A
                    TYPE:       CRTM_Aerosol_type
                    DIMENSION:  Scalar OR any rank
                    ATTRIBUTES: INTENT(OUT)
```

A.4.7 CRTM_Aerosol_InquireFile interface

```
NAME:
      CRTM_Aerosol_InquireFile

PURPOSE:
      Function to inquire CRTM Aerosol object files.

CALLING SEQUENCE:
      Error_Status = CRTM_Aerosol_InquireFile( Filename               , &
                                               n_Aerosols = n_Aerosols  )

INPUTS:
      Filename:       Character string specifying the name of a
                      CRTM Aerosol data file to read.
                      UNITS:      N/A
                      TYPE:       CHARACTER(*)
                      DIMENSION:  Scalar
                      ATTRIBUTES: INTENT(IN)

OPTIONAL OUTPUTS:
      n_Aerosols:     The number of Aerosol profiles in the data file.
                      UNITS:      N/A
                      TYPE:       INTEGER
                      DIMENSION:  Scalar
                      ATTRIBUTES: OPTIONAL, INTENT(OUT)

FUNCTION RESULT:
      Error_Status:   The return value is an integer defining the error status.
                      The error codes are defined in the Message_Handler module.
                      If == SUCCESS, the file inquire was successful
```

```
                        == FAILURE, an unrecoverable error occurred.
                     UNITS:      N/A
                     TYPE:       INTEGER
                     DIMENSION:  Scalar
```

A.4.8 CRTM_Aerosol_Inspect interface

```
NAME:
      CRTM_Aerosol_Inspect

PURPOSE:
      Subroutine to print the contents of a CRTM Aerosol object to stdout.

CALLING SEQUENCE:
      CALL CRTM_Aerosol_Inspect( Aerosol )

INPUTS:
      Aerosol:       CRTM Aerosol object to display.
                     UNITS:      N/A
                     TYPE:       CRTM_Aerosol_type
                     DIMENSION:  Scalar, Rank-1, or Rank-2 array
                     ATTRIBUTES: INTENT(IN)
```

A.4.9 CRTM_Aerosol_IsValid interface

```
NAME:
      CRTM_Aerosol_IsValid

PURPOSE:
      Non-pure function to perform some simple validity checks on a
      CRTM Aerosol object.

      If invalid data is found, a message is printed to stdout.

CALLING SEQUENCE:
      result = CRTM_Aerosol_IsValid( Aerosol )

        or

      IF ( CRTM_Aerosol_IsValid( Aerosol ) ) THEN....

OBJECTS:
      Aerosol:       CRTM Aerosol object which is to have its
                     contents checked.
                     UNITS:      N/A
                     TYPE:       CRTM_Aerosol_type
```

```
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN)

FUNCTION RESULT:
      result:       Logical variable indicating whether or not the input
                    passed the check.
                    If == .FALSE., Aerosol object is unused or contains
                                   invalid data.
                       == .TRUE.,  Aerosol object can be used in CRTM.
                    UNITS:      N/A
                    TYPE:       LOGICAL
                    DIMENSION:  Scalar
```

A.4.10 CRTM_Aerosol_ReadFile interface

```
NAME:
      CRTM_Aerosol_ReadFile

PURPOSE:
      Function to read CRTM Aerosol object files.

CALLING SEQUENCE:
      Error_Status = CRTM_Aerosol_ReadFile( Filename               , &
                                            Aerosol                , &
                                            Quiet      = Quiet     , &
                                            No_Close   = No_Close  , &
                                            n_Aerosols = n_Aerosols  )

INPUTS:
      Filename:      Character string specifying the name of a
                     Aerosol format data file to read.
                     UNITS:      N/A
                     TYPE:       CHARACTER(*)
                     DIMENSION:  Scalar
                     ATTRIBUTES: INTENT(IN)

OUTPUTS:
      Aerosol:       CRTM Aerosol object array containing the Aerosol data.
                     UNITS:      N/A
                     TYPE:       CRTM_Aerosol_type
                     DIMENSION:  Rank-1
                     ATTRIBUTES: INTENT(OUT)

OPTIONAL INPUTS:
      Quiet:         Set this logical argument to suppress INFORMATION
                     messages being printed to stdout
                     If == .FALSE., INFORMATION messages are OUTPUT [DEFAULT].
                        == .TRUE.,  INFORMATION messages are SUPPRESSED.
                     If not specified, default is .FALSE.
                     UNITS:      N/A
```

```
                    TYPE:       LOGICAL
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN), OPTIONAL

      No_Close:     Set this logical argument to NOT close the file upon exit.
                    If == .FALSE., the input file is closed upon exit [DEFAULT]
                       == .TRUE.,  the input file is NOT closed upon exit.
                    If not specified, default is .FALSE.
                    UNITS:      N/A
                    TYPE:       LOGICAL
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN), OPTIONAL

OPTIONAL OUTPUTS:
      n_Aerosols:   The actual number of aerosol profiles read in.
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Scalar
                    ATTRIBUTES: OPTIONAL, INTENT(OUT)

FUNCTION RESULT:
      Error_Status: The return value is an integer defining the error status.
                    The error codes are defined in the Message_Handler module.
                    If == SUCCESS, the file read was successful
                       == FAILURE, an unrecoverable error occurred.
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Scalar
```

A.4.11 CRTM_Aerosol_SetLayers interface

```
NAME:
      CRTM_Aerosol_SetLayers

PURPOSE:
      Elemental subroutine to set the working number of layers to use
      in a CRTM Aerosol object.

CALLING SEQUENCE:
     CALL CRTM_Aerosol_SetLayers( Aerosol, n_Layers )

OBJECT:
      Aerosol:      CRTM Aerosol object which is to have its working number
                    of layers updated.
                    UNITS:      N/A
                    TYPE:       CRTM_Aerosol_type
                    DIMENSION:  Scalar or any rank
                    ATTRIBUTES: INTENT(IN OUT)

INPUTS:
      n_Layers:     The value to set the n_Layers component of the
                    Aerosol object.
                    UNITS:      N/A
                    TYPE:       CRTM_Aerosol_type
                    DIMENSION:  Conformable with the Aerosol object argument
                    ATTRIBUTES: INTENT(IN)

COMMENTS:
      - The object is zeroed upon output.

      - If n_Layers <= Aerosol%Max_Layers, then only the dimension value
        of the object is changed.

      - If n_Layers > Aerosol%Max_Layers, then the object is reallocated
        to the required number of layers.
```

A.4.12 CRTM_Aerosol_WriteFile interface

```
NAME:
      CRTM_Aerosol_WriteFile

PURPOSE:
      Function to write CRTM Aerosol object files.

CALLING SEQUENCE:
      Error_Status = CRTM_Aerosol_WriteFile( Filename           , &
                                             Aerosol            , &
                                             Quiet    = Quiet   , &
                                             No_Close = No_Close  )

INPUTS:
      Filename:       Character string specifying the name of the
                      Aerosol format data file to write.
                      UNITS:      N/A
                      TYPE:       CHARACTER(*)
                      DIMENSION:  Scalar
                      ATTRIBUTES: INTENT(IN)

      Aerosol:        CRTM Aerosol object array containing the Aerosol data.
                      UNITS:      N/A
                      TYPE:       CRTM_Aerosol_type
                      DIMENSION:  Rank-1
                      ATTRIBUTES: INTENT(IN)

OPTIONAL INPUTS:
      Quiet:          Set this logical argument to suppress INFORMATION
                      messages being printed to stdout
                      If == .FALSE., INFORMATION messages are OUTPUT [DEFAULT].
                         == .TRUE.,  INFORMATION messages are SUPPRESSED.
                      If not specified, default is .FALSE.
```

```
                    UNITS:      N/A
                    TYPE:       LOGICAL
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN), OPTIONAL

      No_Close:     Set this logical argument to NOT close the file upon exit.
                    If == .FALSE., the input file is closed upon exit [DEFAULT]
                       == .TRUE.,  the input file is NOT closed upon exit.
                    If not specified, default is .FALSE.
                    UNITS:      N/A
                    TYPE:       LOGICAL
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN), OPTIONAL

FUNCTION RESULT:
      Error_Status: The return value is an integer defining the error status.
                    The error codes are defined in the Message_Handler module.
                    If == SUCCESS, the file write was successful
                       == FAILURE, an unrecoverable error occurred.
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Scalar

SIDE EFFECTS:
      - If the output file already exists, it is overwritten.
      - If an error occurs during *writing*, the output file is deleted before
        returning to the calling routine.
```

A.4.13 CRTM_Aerosol_Zero interface

```
NAME:
      CRTM_Aerosol_Zero

PURPOSE:
      Elemental subroutine to zero out the data arrays in a CRTM Aerosol object.

CALLING SEQUENCE:
      CALL CRTM_Aerosol_Zero( Aerosol )

OBJECTS:
      Aerosol:     CRTM Aerosol object in which the data arrays are
                   to be zeroed out.
                   UNITS:      N/A
                   TYPE:       CRTM_Aerosol_type
                   DIMENSION:  Scalar or any rank
                   ATTRIBUTES: INTENT(IN OUT)

COMMENTS:
      - The dimension components of the structure are *NOT* set to zero.
      - The Aerosol type component is *NOT* reset.
```

A.5 Surface Structure

```
TYPE :: CRTM_Surface_type
  ! Gross type of surface determined by coverage
  REAL(fp) :: Land_Coverage  = ZERO
  REAL(fp) :: Water_Coverage = ZERO
  REAL(fp) :: Snow_Coverage  = ZERO
  REAL(fp) :: Ice_Coverage   = ZERO
  ! Land surface type data
  INTEGER  :: Land_Type             = DEFAULT_LAND_TYPE
  REAL(fp) :: Land_Temperature      = DEFAULT_LAND_TEMPERATURE
  REAL(fp) :: Soil_Moisture_Content = DEFAULT_SOIL_MOISTURE_CONTENT
  REAL(fp) :: Canopy_Water_Content  = DEFAULT_CANOPY_WATER_CONTENT
  REAL(fp) :: Vegetation_Fraction   = DEFAULT_VEGETATION_FRACTION
  REAL(fp) :: Soil_Temperature      = DEFAULT_SOIL_TEMPERATURE
  REAL(fp) :: LAI                   = DEFAULT_LAI
  INTEGER  :: Soil_Type             = DEFAULT_SOIL_TYPE
  INTEGER  :: Vegetation_Type       = DEFAULT_VEGETATION_TYPE
  ! Water type data
  INTEGER  :: Water_Type        = DEFAULT_WATER_TYPE
  REAL(fp) :: Water_Temperature = DEFAULT_WATER_TEMPERATURE
  REAL(fp) :: Wind_Speed        = DEFAULT_WIND_SPEED
  REAL(fp) :: Wind_Direction    = DEFAULT_WIND_DIRECTION
  REAL(fp) :: Salinity          = DEFAULT_SALINITY
  ! Snow surface type data
  INTEGER  :: Snow_Type        = DEFAULT_SNOW_TYPE
  REAL(fp) :: Snow_Temperature = DEFAULT_SNOW_TEMPERATURE
  REAL(fp) :: Snow_Depth       = DEFAULT_SNOW_DEPTH
  REAL(fp) :: Snow_Density     = DEFAULT_SNOW_DENSITY
  REAL(fp) :: Snow_Grain_Size  = DEFAULT_SNOW_GRAIN_SIZE
  ! Ice surface type data
  INTEGER  :: Ice_Type        = DEFAULT_ICE_TYPE
  REAL(fp) :: Ice_Temperature = DEFAULT_ICE_TEMPERATURE
  REAL(fp) :: Ice_Thickness   = DEFAULT_ICE_THICKNESS
  REAL(fp) :: Ice_Density     = DEFAULT_ICE_DENSITY
  REAL(fp) :: Ice_Roughness   = DEFAULT_ICE_ROUGHNESS
  ! SensorData containing channel brightness temperatures
  TYPE(CRTM_SensorData_type) :: SensorData
END TYPE CRTM_Surface_type
```

Figure A.5: CRTM_Surface_type structure definition.

A.5.1 CRTM_Surface_Associated interface

```
NAME:
      CRTM_Surface_Associated

PURPOSE:
      Elemental function to test the status of the allocatable components
      of a CRTM Surface object.

CALLING SEQUENCE:
      Status = CRTM_Surface_Associated( Sfc )

OBJECTS:
      Sfc:       Surface structure which is to have its member's
                 status tested.
                 UNITS:      N/A
                 TYPE:       CRTM_Surface_type
                 DIMENSION:  Scalar or any rank
                 ATTRIBUTES: INTENT(IN)

FUNCTION RESULT:
      Status:    The return value is a logical value indicating the
                 status of the Surface members.
                   .TRUE.  - if the array components are allocated.
                   .FALSE. - if the array components are not allocated.
                 UNITS:      N/A
                 TYPE:       LOGICAL
                 DIMENSION:  Same as input
```

A.5.2 CRTM_Surface_Compare interface

```
NAME:
      CRTM_Surface_Compare

PURPOSE:
      Elemental function to compare two CRTM_Surface objects to within
      a user specified number of significant figures.

CALLING SEQUENCE:
      is_comparable = CRTM_Surface_Compare( x, y, n_SigFig=n_SigFig )

OBJECTS:
      x, y:          Two CRTM Surface objects to be compared.
                     UNITS:      N/A
                     TYPE:       CRTM_Surface_type
                     DIMENSION:  Scalar or any rank
                     ATTRIBUTES: INTENT(IN)

OPTIONAL INPUTS:
      n_SigFig:      Number of significant figure to compare floating point
```

```
                    components.
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Scalar or same as input
                    ATTRIBUTES: INTENT(IN), OPTIONAL

FUNCTION RESULT:
      is_equal:     Logical value indicating whether the inputs are equal.
                    UNITS:      N/A
                    TYPE:       LOGICAL
                    DIMENSION:  Same as inputs.
```

A.5.3 CRTM_Surface_CoverageType interface

```
NAME:
      CRTM_Surface_CoverageType

PURPOSE:
      Elemental function to return the gross surface type based on coverage.

CALLING SEQUENCE:
      type = CRTM_Surface_CoverageType( sfc )

INPUTS:
      Sfc:  CRTM Surface object for which the gross surface type is required.
            UNITS:      N/A
            TYPE:       CRTM_Surface_type
            DIMENSION:  Scalar or any rank
            ATTRIBUTES: INTENT(IN)

FUNCTION:
      type: Surface type indicator for the passed CRTM Surface object.
            UNITS:      N/A
            TYPE:       INTEGER
            DIMENSION:  Same as input

COMMENTS:
      For a scalar Surface object, this function result can be used to
      determine what gross surface types are included by using it to
      index the SURFACE_TYPE_NAME parameter arrays, e.g.

        WRITE(*,*) SURFACE_TYPE_NAME(CRTM_Surface_CoverageType(sfc))
```

A.5.4 CRTM_Surface_Create interface

```
NAME:
      CRTM_Surface_Create
```

```
PURPOSE:
      Elemental subroutine to create an instance of the CRTM Surface object.

CALLING SEQUENCE:
      CALL CRTM_Surface_Create( Sfc        , &
                                n_Channels  )

OBJECTS:
      Sfc:          Surface structure.
                    UNITS:      N/A
                    TYPE:       CRTM_Surface_type
                    DIMENSION:  Scalar or any rank
                    ATTRIBUTES: INTENT(OUT)

INPUT ARGUMENTS:
      n_Channels:   Number of channels dimension of SensorData
                    substructure
                    ** Note: Can be = 0 (i.e. no sensor data). **
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Same as Surface object
                    ATTRIBUTES: INTENT(IN)
```

A.5.5 CRTM_Surface_DefineVersion interface

```
NAME:
      CRTM_Surface_DefineVersion

PURPOSE:
      Subroutine to return the module version information.

CALLING SEQUENCE:
      CALL CRTM_Surface_DefineVersion( Id )

OUTPUT ARGUMENTS:
      Id:            Character string containing the version Id information
                     for the module.
                     UNITS:      N/A
                     TYPE:       CHARACTER(*)
                     DIMENSION:  Scalar
                     ATTRIBUTES: INTENT(OUT)
```

A.5.6 CRTM_Surface_Destroy interface

```
NAME:
      CRTM_Surface_Destroy
```

```
PURPOSE:
      Elemental subroutine to re-initialize CRTM Surface objects.

CALLING SEQUENCE:
      CALL CRTM_Surface_Destroy( Sfc )

OBJECTS:
      Sfc:          Re-initialized Surface structure.
                    UNITS:      N/A
                    TYPE:       CRTM_Surface_type
                    DIMENSION:  Scalar or any rank
                    ATTRIBUTES: INTENT(OUT)
```

A.5.7 CRTM_Surface_InquireFile interface

```
NAME:
      CRTM_Surface_InquireFile

PURPOSE:
      Function to inquire CRTM Surface object files.

CALLING SEQUENCE:
      Error_Status = CRTM_Surface_InquireFile( Filename               , &
                                               n_Channels = n_Channels, &
                                               n_Profiles = n_Profiles  )

INPUTS:
      Filename:       Character string specifying the name of a
                      CRTM Surface data file to read.
                      UNITS:      N/A
                      TYPE:       CHARACTER(*)
                      DIMENSION:  Scalar
                      ATTRIBUTES: INTENT(IN)

OPTIONAL OUTPUTS:
      n_Channels:     The number of spectral channels for which there is
                      data in the file. Note that this value will always
                      be 0 for a profile-only dataset-- it only has meaning
                      for K-matrix data.
                      UNITS:      N/A
                      TYPE:       INTEGER
                      DIMENSION:  Scalar
                      ATTRIBUTES: OPTIONAL, INTENT(OUT)

      n_Profiles:     The number of profiles in the data file.
                      UNITS:      N/A
                      TYPE:       INTEGER
                      DIMENSION:  Scalar
                      ATTRIBUTES: OPTIONAL, INTENT(OUT)
```

```
FUNCTION RESULT:
      Error_Status:   The return value is an integer defining the error status.
                      The error codes are defined in the Message_Handler module.
                      If == SUCCESS, the file inquire was successful
                         == FAILURE, an unrecoverable error occurred.
                      UNITS:      N/A
                      TYPE:       INTEGER
                      DIMENSION:  Scalar
```

A.5.8 CRTM_Surface_Inspect interface

```
NAME:
      CRTM_Surface_Inspect

PURPOSE:
      Subroutine to print the contents of a CRTM Surface object to stdout.

CALLING SEQUENCE:
      CALL CRTM_Surface_Inspect( Sfc )

INPUTS:
      Sfc:  CRTM Surface object to display.
            UNITS:      N/A
            TYPE:       CRTM_Surface_type
            DIMENSION:  Scalar
            ATTRIBUTES: INTENT(IN)
```

A.5.9 CRTM_Surface_IsCoverageValid interface

```
NAME:
      CRTM_Surface_IsCoverageValid

PURPOSE:
      Function to determine if the coverage fractions are valid
      for a CRTM Surface object.

CALLING SEQUENCE:
      result = CRTM_Surface_IsCoverageValid( Sfc )

OBJECTS:
      Sfc:       CRTM Surface object which is to have its
                 coverage fractions checked.
                 UNITS:      N/A
                 TYPE:       CRTM_Surface_type
                 DIMENSION:  Scalar
                 ATTRIBUTES: INTENT(IN)

FUNCTION RESULT:
      result:    Logical variable indicating whether or not the input
                 passed the check.
                 If == .FALSE., Surface object coverage fractions are invalid.
                    == .TRUE.,  Surface object coverage fractions are valid.
                 UNITS:      N/A
                 TYPE:       LOGICAL
                 DIMENSION:  Scalar
```

A.5.10 CRTM_Surface_IsValid interface

```
  NAME:
        CRTM_Surface_IsValid

  PURPOSE:
        Non-pure function to perform some simple validity checks on a
        CRTM Surface object.

        If invalid data is found, a message is printed to stdout.

  CALLING SEQUENCE:
        result = CRTM_Surface_IsValid( Sfc )

          or

        IF ( CRTM_Surface_IsValid( Sfc ) ) THEN....

  OBJECTS:
        Sfc:       CRTM Surface object which is to have its
                   contents checked.
                   UNITS:      N/A
                   TYPE:       CRTM_Surface_type
                   DIMENSION:  Scalar
                   ATTRIBUTES: INTENT(IN)

  FUNCTION RESULT:
        result:    Logical variable indicating whether or not the input
                   passed the check.
                   If == .FALSE., Surface object is unused or contains
                                  invalid data.
                      == .TRUE.,  Surface object can be used in CRTM.
                   UNITS:      N/A
                   TYPE:       LOGICAL
                   DIMENSION:  Scalar
```

A.5.11 CRTM_Surface_ReadFile interface

```
NAME:
      CRTM_Surface_ReadFile

PURPOSE:
      Function to read CRTM Surface object files.

CALLING SEQUENCE:
      Error_Status = CRTM_Surface_ReadFile( Filename                , &
                                            Surface                 , &
                                            Quiet      = Quiet      , &
                                            n_Channels = n_Channels, &
                                            n_Profiles = n_Profiles  )

INPUTS:
      Filename:     Character string specifying the name of an
                    Surface format data file to read.
                    UNITS:      N/A
                    TYPE:       CHARACTER(*)
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN)

OUTPUTS:
      Surface:      CRTM Surface object array containing the Surface
                    data. Note the following meanings attributed to the
                    dimensions of the object array:
                    Rank-1: M profiles.
                            Only profile data are to be read in. The file
                            does not contain channel information. The
                            dimension of the structure is understood to
                            be the PROFILE dimension.
                    Rank-2: L channels  x  M profiles
                            Channel and profile data are to be read in.
                            The file contains both channel and profile
                            information. The first dimension of the
                            structure is the CHANNEL dimension, the second
                            is the PROFILE dimension. This is to allow
                            K-matrix structures to be read in with the
                            same function.
                    UNITS:      N/A
                    TYPE:       CRTM_Surface_type
                    DIMENSION:  Rank-1 (M) or Rank-2 (L x M)
                    ATTRIBUTES: INTENT(OUT)

OPTIONAL INPUTS:
      Quiet:        Set this logical argument to suppress INFORMATION
                    messages being printed to stdout
                    If == .FALSE., INFORMATION messages are OUTPUT [DEFAULT].
                       == .TRUE.,  INFORMATION messages are SUPPRESSED.
                    If not specified, default is .FALSE.
                    UNITS:      N/A
                    TYPE:       LOGICAL
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN), OPTIONAL

OPTIONAL OUTPUTS:
      n_Channels:   The number of channels for which data was read. Note that
                    this value will always be 0 for a profile-only dataset--
                    it only has meaning for K-matrix data.
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Scalar
                    ATTRIBUTES: OPTIONAL, INTENT(OUT)

      n_Profiles:   The number of profiles for which data was read.
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Scalar
                    ATTRIBUTES: OPTIONAL, INTENT(OUT)


FUNCTION RESULT:
      Error_Status: The return value is an integer defining the error status.
                    The error codes are defined in the Message_Handler module.
                    If == SUCCESS, the file read was successful
                       == FAILURE, an unrecoverable error occurred.
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Scalar
```

A.5.12 CRTM_Surface_WriteFile interface

```
  NAME:
        CRTM_Surface_WriteFile

  PURPOSE:
        Function to write CRTM Surface object files.

  CALLING SEQUENCE:
        Error_Status = CRTM_Surface_WriteFile( Filename     , &
                                               Surface      , &
                                               Quiet = Quiet  )

  INPUTS:
        Filename:     Character string specifying the name of the
                      Surface format data file to write.
                      UNITS:      N/A
                      TYPE:       CHARACTER(*)
                      DIMENSION:  Scalar
                      ATTRIBUTES: INTENT(IN)

        Surface:      CRTM Surface object array containing the Surface
```

```
                    data. Note the following meanings attributed to the
                    dimensions of the Surface array:
                    Rank-1: M profiles.
                            Only profile data are to be read in. The file
                            does not contain channel information. The
                            dimension of the array is understood to
                            be the PROFILE dimension.
                    Rank-2: L channels  x  M profiles
                            Channel and profile data are to be read in.
                            The file contains both channel and profile
                            information. The first dimension of the
                            array is the CHANNEL dimension, the second
                            is the PROFILE dimension. This is to allow
                            K-matrix structures to be read in with the
                            same function.
                    UNITS:      N/A
                    TYPE:       CRTM_Surface_type
                    DIMENSION:  Rank-1 (M) or Rank-2 (L x M)
                    ATTRIBUTES: INTENT(IN)

OPTIONAL INPUTS:
      Quiet:        Set this logical argument to suppress INFORMATION
                    messages being printed to stdout
                    If == .FALSE., INFORMATION messages are OUTPUT [DEFAULT].
                       == .TRUE.,  INFORMATION messages are SUPPRESSED.
                    If not specified, default is .FALSE.
                    UNITS:      N/A
                    TYPE:       LOGICAL
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN), OPTIONAL

FUNCTION RESULT:
      Error_Status: The return value is an integer defining the error status.
                    The error codes are defined in the Message_Handler module.
                    If == SUCCESS, the file write was successful
                       == FAILURE, an unrecoverable error occurred.
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Scalar

SIDE EFFECTS:
      - If the output file already exists, it is overwritten.
      - If an error occurs during *writing*, the output file is deleted before
        returning to the calling routine.
```

A.5.13 CRTM_Surface_Zero interface

```
NAME:
      CRTM_Surface_Zero
```

```
PURPOSE:
      Elemental subroutine to zero out the data arrays
      in a CRTM Surface object.

CALLING SEQUENCE:
      CALL CRTM_Surface_Zero( Sfc )

OUTPUT ARGUMENTS:
      Sfc:          CRTM Surface structure in which the data arrays
                    are to be zeroed out.
                    UNITS:      N/A
                    TYPE:       CRTM_Surface_type
                    DIMENSION:  Scalar or any rank
                    ATTRIBUTES: INTENT(IN OUT)

COMMENTS:
      - The various surface type indicator flags are
        *NOT* reset in this routine.
```

A.6 `SensorData` Structure

```
TYPE :: CRTM_SensorData_type
  ! Allocation indicator
  LOGICAL :: Is_Allocated = .FALSE.
  ! Dimension values
  INTEGER :: n_Channels = 0  ! L
  ! The data sensor IDs
  CHARACTER(STRLEN) :: Sensor_Id        = ' '
  INTEGER           :: WMO_Satellite_ID = INVALID_WMO_SATELLITE_ID
  INTEGER           :: WMO_Sensor_ID    = INVALID_WMO_SENSOR_ID
  ! The sensor channels and brightness temperatures
  INTEGER , ALLOCATABLE :: Sensor_Channel(:)   ! L
  REAL(fp), ALLOCATABLE :: Tb(:)               ! L
END TYPE CRTM_SensorData_type
```

Figure A.6: CRTM_SensorData_type structure definition.

A.6.1 CRTM_SensorData_Associated interface

```
NAME:
      CRTM_SensorData_Associated

PURPOSE:
      Elemental function to test the status of the allocatable components
      of a CRTM SensorData object.

CALLING SEQUENCE:
      Status = CRTM_SensorData_Associated( SensorData )

OBJECTS:
      SensorData:  SensorData structure which is to have its member's
                   status tested.
                   UNITS:      N/A
                   TYPE:       CRTM_SensorData_type
                   DIMENSION:  Scalar or any rank
                   ATTRIBUTES: INTENT(IN)

FUNCTION RESULT:
      Status:      The return value is a logical value indicating the
                   status of the SensorData members.
                     .TRUE.  - if the array components are allocated.
                     .FALSE. - if the array components are not allocated.
                   UNITS:      N/A
                   TYPE:       LOGICAL
                   DIMENSION:  Same as input SensorData argument
```

A.6.2 CRTM_SensorData_Compare interface

```
NAME:
      CRTM_SensorData_Compare

PURPOSE:
      Elemental function to compare two CRTM_SensorData objects to within
      a user specified number of significant figures.

CALLING SEQUENCE:
      is_comparable = CRTM_SensorData_Compare( x, y, n_SigFig=n_SigFig )

OBJECTS:
      x, y:          Two CRTM SensorData objects to be compared.
                     UNITS:      N/A
                     TYPE:       CRTM_SensorData_type
                     DIMENSION:  Scalar or any rank
                     ATTRIBUTES: INTENT(IN)

OPTIONAL INPUTS:
      n_SigFig:      Number of significant figure to compare floating point
```

```
                     components.
                     UNITS:      N/A
                     TYPE:       INTEGER
                     DIMENSION:  Scalar or same as input
                     ATTRIBUTES: INTENT(IN), OPTIONAL

FUNCTION RESULT:
      is_equal:      Logical value indicating whether the inputs are equal.
                     UNITS:      N/A
                     TYPE:       LOGICAL
                     DIMENSION:  Same as inputs.
```

A.6.3 CRTM_SensorData_Create interface

```
NAME:
      CRTM_SensorData_Create

PURPOSE:
      Elemental subroutine to create an instance of the CRTM SensorData object.

CALLING SEQUENCE:
      CALL CRTM_SensorData_Create( SensorData, n_Channels )

OBJECTS:
      SensorData:   SensorData structure.
                    UNITS:      N/A
                    TYPE:       CRTM_SensorData_type
                    DIMENSION:  Scalar or any rank
                    ATTRIBUTES: INTENT(OUT)

INPUTS:
      n_Channels:   Number of sensor channels.
                    Must be > 0.
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Same as SensorData object
                    ATTRIBUTES: INTENT(IN)
```

A.6.4 CRTM_SensorData_DefineVersion interface

```
NAME:
      CRTM_SensorData_DefineVersion

PURPOSE:
      Subroutine to return the module version information.

CALLING SEQUENCE:
      CALL CRTM_SensorData_DefineVersion( Id )

OUTPUT ARGUMENTS:
      Id:            Character string containing the version Id information
                     for the module.
                     UNITS:      N/A
                     TYPE:       CHARACTER(*)
                     DIMENSION:  Scalar
                     ATTRIBUTES: INTENT(OUT)
```

A.6.5 CRTM_SensorData_Destroy interface

```
NAME:
      CRTM_SensorData_Destroy

PURPOSE:
      Elemental subroutine to re-initialize CRTM SensorData objects.

CALLING SEQUENCE:
      CALL CRTM_SensorData_Destroy( SensorData )

OBJECTS:
      SensorData:   Re-initialized SensorData structure.
                    UNITS:      N/A
                    TYPE:       CRTM_SensorData_type
                    DIMENSION:  Scalar OR any rank
                    ATTRIBUTES: INTENT(OUT)
```

A.6.6 CRTM_SensorData_InquireFile interface

```
NAME:
      CRTM_SensorData_InquireFile

PURPOSE:
      Function to inquire CRTM SensorData object files.

CALLING SEQUENCE:
      Error_Status = CRTM_SensorData_InquireFile( Filename           , &
                                                  n_DataSets = n_DataSets  )

INPUTS:
      Filename:       Character string specifying the name of a
                      CRTM SensorData data file to read.
                      UNITS:      N/A
                      TYPE:       CHARACTER(*)
                      DIMENSION:  Scalar
                      ATTRIBUTES: INTENT(IN)

OPTIONAL OUTPUTS:
      n_DataSets:     The number of datasets in the file.
                      UNITS:      N/A
                      TYPE:       INTEGER
                      DIMENSION:  Scalar
                      ATTRIBUTES: OPTIONAL, INTENT(OUT)

FUNCTION RESULT:
      Error_Status:   The return value is an integer defining the error status.
                      The error codes are defined in the Message_Handler module.
                      If == SUCCESS, the file inquire was successful
                         == FAILURE, an unrecoverable error occurred.
                      UNITS:      N/A
                      TYPE:       INTEGER
                      DIMENSION:  Scalar
```

A.6.7 CRTM_SensorData_Inspect interface

```
NAME:
      CRTM_SensorData_Inspect

PURPOSE:
      Subroutine to print the contents of a CRTM SensorData object to stdout.

CALLING SEQUENCE:
      CALL CRTM_SensorData_Inspect( SensorData )

INPUTS:
      SensorData:    CRTM SensorData object to display.
                     UNITS:      N/A
                     TYPE:       CRTM_SensorData_type
                     DIMENSION:  Scalar
                     ATTRIBUTES: INTENT(IN)
```

A.6.8 CRTM_SensorData_IsValid interface

```
NAME:
      CRTM_SensorData_IsValid

PURPOSE:
      Non-pure function to perform some simple validity checks on a
      CRTM SensorData object.

      If invalid data is found, a message is printed to stdout.
```

```
CALLING SEQUENCE:
      result = CRTM_SensorData_IsValid( SensorData )

        or

      IF ( CRTM_SensorData_IsValid( SensorData ) ) THEN....

OBJECTS:
      SensorData:    CRTM SensorData object which is to have its
                     contents checked.
                     UNITS:      N/A
                     TYPE:       CRTM_SensorData_type
                     DIMENSION:  Scalar
                     ATTRIBUTES: INTENT(IN)

FUNCTION RESULT:
      result:        Logical variable indicating whether or not the input
                     passed the check.
                     If == .FALSE., SensorData object is unused or contains
                                    invalid data.
                        == .TRUE.,  SensorData object can be used in CRTM.
                     UNITS:      N/A
                     TYPE:       LOGICAL
                     DIMENSION:  Scalar
```

A.6.9 CRTM_SensorData_ReadFile interface

```
NAME:
      CRTM_SensorData_ReadFile

PURPOSE:
      Function to read CRTM SensorData object files.

CALLING SEQUENCE:
      Error_Status = CRTM_SensorData_ReadFile( Filename              , &
                                               SensorData            , &
                                               Quiet      = Quiet    , &
                                               No_Close   = No_Close , &
                                               n_DataSets = n_DataSets )

INPUTS:
      Filename:       Character string specifying the name of a
                      SensorData format data file to read.
                      UNITS:      N/A
                      TYPE:       CHARACTER(*)
                      DIMENSION:  Scalar
                      ATTRIBUTES: INTENT(IN)

OUTPUTS:
      SensorData:     CRTM SensorData object array containing the sensor data.
                      UNITS:      N/A
                      TYPE:       CRTM_SensorData_type
                      DIMENSION:  Rank-1
                      ATTRIBUTES: INTENT(OUT)

OPTIONAL INPUTS:
      Quiet:          Set this logical argument to suppress INFORMATION
                      messages being printed to stdout
                      If == .FALSE., INFORMATION messages are OUTPUT [DEFAULT].
                         == .TRUE.,  INFORMATION messages are SUPPRESSED.
                      If not specified, default is .FALSE.
                      UNITS:      N/A
                      TYPE:       LOGICAL
                      DIMENSION:  Scalar
                      ATTRIBUTES: INTENT(IN), OPTIONAL

      No_Close:       Set this logical argument to NOT close the file upon exit.
                      If == .FALSE., the input file is closed upon exit [DEFAULT]
                         == .TRUE.,  the input file is NOT closed upon exit.
                      If not specified, default is .FALSE.
                      UNITS:      N/A
                      TYPE:       LOGICAL
                      DIMENSION:  Scalar
                      ATTRIBUTES: INTENT(IN), OPTIONAL

OPTIONAL OUTPUTS:
      n_DataSets:     The actual number of datasets read in.
                      UNITS:      N/A
                      TYPE:       INTEGER
                      DIMENSION:  Scalar
                      ATTRIBUTES: OPTIONAL, INTENT(OUT)

FUNCTION RESULT:
      Error_Status:   The return value is an integer defining the error status.
                      The error codes are defined in the Message_Handler module.
                      If == SUCCESS, the file read was successful
                         == FAILURE, an unrecoverable error occurred.
                      UNITS:      N/A
                      TYPE:       INTEGER
                      DIMENSION:  Scalar
```

A.6.10 CRTM_SensorData_WriteFile interface

```
NAME:
      CRTM_SensorData_WriteFile

PURPOSE:
      Function to write CRTM SensorData object files.
```

```
CALLING SEQUENCE:
      Error_Status = CRTM_SensorData_WriteFile( Filename           , &
                                                SensorData         , &
                                                Quiet    = Quiet   , &
                                                No_Close = No_Close  )

INPUTS:
      Filename:       Character string specifying the name of the
                      SensorData format data file to write.
                      UNITS:      N/A
                      TYPE:       CHARACTER(*)
                      DIMENSION:  Scalar
                      ATTRIBUTES: INTENT(IN)

      SensorData:     CRTM SensorData object array containing the datasets.
                      UNITS:      N/A
                      TYPE:       CRTM_SensorData_type
                      DIMENSION:  Rank-1
                      ATTRIBUTES: INTENT(IN)

OPTIONAL INPUTS:
      Quiet:          Set this logical argument to suppress INFORMATION
                      messages being printed to stdout
                      If == .FALSE., INFORMATION messages are OUTPUT [DEFAULT].
                         == .TRUE.,  INFORMATION messages are SUPPRESSED.
                      If not specified, default is .FALSE.
                      UNITS:      N/A
                      TYPE:       LOGICAL
                      DIMENSION:  Scalar
                      ATTRIBUTES: INTENT(IN), OPTIONAL

      No_Close:       Set this logical argument to NOT close the file upon exit.
                      If == .FALSE., the input file is closed upon exit [DEFAULT]
                         == .TRUE.,  the input file is NOT closed upon exit.
                      If not specified, default is .FALSE.
                      UNITS:      N/A
                      TYPE:       LOGICAL
                      DIMENSION:  Scalar
                      ATTRIBUTES: INTENT(IN), OPTIONAL

FUNCTION RESULT:
      Error_Status:   The return value is an integer defining the error status.
                      The error codes are defined in the Message_Handler module.
                      If == SUCCESS, the file write was successful
                         == FAILURE, an unrecoverable error occurred.
                      UNITS:      N/A
                      TYPE:       INTEGER
                      DIMENSION:  Scalar

SIDE EFFECTS:
      - If the output file already exists, it is overwritten.
      - If an error occurs during *writing*, the output file is deleted before
        returning to the calling routine.
```

A.6.11 CRTM_SensorData_Zero interface

```
NAME:
      CRTM_SensorData_Zero

PURPOSE:
      Elemental subroutine to zero out the data arrays in a
      CRTM SensorData object.

CALLING SEQUENCE:
      CALL CRTM_SensorData_Zero( SensorData )

OBJECTS:
      SensorData:    CRTM SensorData structure in which the data arrays are
                     to be zeroed out.
                     UNITS:      N/A
                     TYPE:       CRTM_SensorData_type
                     DIMENSION:  Scalar or any rank
                     ATTRIBUTES: INTENT(IN OUT)

COMMENTS:
      - The dimension components of the structure are *NOT* set to zero.
      - The SensorData sensor id and channel components are *NOT* reset.
```

A.7 `Geometry` Structure

```
TYPE :: CRTM_Geometry_type
  ! Allocation indicator
  LOGICAL :: Is_Allocated = .FALSE.
  ! Field of view index (1-nFOV)
  INTEGER  :: iFOV = 0
  ! Earth location
  REAL(fp) :: Longitude        = ZERO
  REAL(fp) :: Latitude         = ZERO
  REAL(fp) :: Surface_Altitude = ZERO
  ! Sensor angle information
  REAL(fp) :: Sensor_Scan_Angle    = ZERO
  REAL(fp) :: Sensor_Zenith_Angle  = ZERO
  REAL(fp) :: Sensor_Azimuth_Angle = 999.9_fp  ! Invalid marker
  ! Source angle information
  REAL(fp) :: Source_Zenith_Angle  = 100.0_fp  ! Below horizon
  REAL(fp) :: Source_Azimuth_Angle = ZERO
  ! Flux angle information
  REAL(fp) :: Flux_Zenith_Angle = DIFFUSIVITY_ANGLE
  ! Date for geometry calculations
  INTEGER :: Year  = 2001
  INTEGER :: Month = 1
  INTEGER :: Day   = 1
END TYPE CRTM_Geometry_type
```

Figure A.7: CRTM_Geometry_type structure definition.

A.7.1 CRTM_Geometry_Associated interface

```
NAME:
      CRTM_Geometry_Associated

PURPOSE:
      Elemental function to test the status of the allocatable components
      of a CRTM Geometry object.

CALLING SEQUENCE:
      Status = CRTM_Geometry_Associated( geo )

OBJECTS:
      geo:          Geometry structure which is to have its member's
                    status tested.
                    UNITS:      N/A
                    TYPE:       CRTM_Geometry_type
                    DIMENSION:  Scalar or any rank
                    ATTRIBUTES: INTENT(IN)

FUNCTION RESULT:
      Status:       The return value is a logical value indicating the
                    status of the Geometry members.
                      .TRUE.  - if the array components are allocated.
                      .FALSE. - if the array components are not allocated.
                    UNITS:      N/A
                    TYPE:       LOGICAL
                    DIMENSION:  Same as input Geometry argument
```

A.7.2 CRTM_Geometry_Compare interface

```
NAME:
      CRTM_Geometry_Compare

PURPOSE:
      Elemental function to compare two CRTM_Geometry objects to within
      a user specified number of significant figures.

CALLING SEQUENCE:
      is_comparable = CRTM_Geometry_Compare( x, y, n_SigFig=n_SigFig )

OBJECTS:
      x, y:          Two CRTM Geometry objects to be compared.
                     UNITS:      N/A
                     TYPE:       CRTM_Geometry_type
                     DIMENSION:  Scalar or any rank
                     ATTRIBUTES: INTENT(IN)

OPTIONAL INPUTS:
      n_SigFig:      Number of significant figure to compare floating point
```

```
                    components.
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Scalar or same as input
                    ATTRIBUTES: INTENT(IN), OPTIONAL

FUNCTION RESULT:
      is_equal:     Logical value indicating whether the inputs are equal.
                    UNITS:      N/A
                    TYPE:       LOGICAL
                    DIMENSION:  Same as inputs.
```

A.7.3 CRTM_Geometry_Create interface

```
NAME:
      CRTM_Geometry_Create

PURPOSE:
      Elemental subroutine to create an instance of the CRTM Geometry object.

CALLING SEQUENCE:
      CALL CRTM_Geometry_Create( geo )

OBJECTS:
      geo:         Geometry structure.
                   UNITS:      N/A
                   TYPE:       CRTM_Geometry_type
                   DIMENSION:  Scalar or any rank
                   ATTRIBUTES: INTENT(OUT)
```

A.7.4 CRTM_Geometry_DefineVersion interface

```
NAME:
      CRTM_Geometry_DefineVersion

PURPOSE:
      Subroutine to return the module version information.

CALLING SEQUENCE:
      CALL CRTM_Geometry_DefineVersion( Id )

OUTPUT ARGUMENTS:
      Id:           Character string containing the version Id information
                    for the module.
                    UNITS:      N/A
                    TYPE:       CHARACTER(*)
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(OUT)
```

A.7.5 CRTM_Geometry_Destroy interface

```
 NAME:
       CRTM_Geometry_Destroy

 PURPOSE:
       Elemental subroutine to re-initialize CRTM Geometry objects.

 CALLING SEQUENCE:
       CALL CRTM_Geometry_Destroy( geo )

 OBJECTS:
       geo:          Re-initialized Geometry structure.
                     UNITS:      N/A
                     TYPE:       CRTM_Geometry_type
                     DIMENSION:  Scalar or any rank
                     ATTRIBUTES: INTENT(OUT)
```

A.7.6 CRTM_Geometry_GetValue interface

```
 NAME:
       CRTM_Geometry_GetValue

 PURPOSE:
       Elemental subroutine to get the values of CRTM Geometry
       object components.

 CALLING SEQUENCE:
       CALL CRTM_Geometry_GetValue( geo, &
                                    iFOV                 = iFOV                 , &
                                    Longitude            = Longitude            , &
                                    Latitude             = Latitude             , &
                                    Surface_Altitude     = Surface_Altitude     , &
                                    Sensor_Scan_Angle    = Sensor_Scan_Angle    , &
                                    Sensor_Zenith_Angle  = Sensor_Zenith_Angle  , &
                                    Sensor_Azimuth_Angle = Sensor_Azimuth_Angle, &
                                    Source_Zenith_Angle  = Source_Zenith_Angle  , &
                                    Source_Azimuth_Angle = Source_Azimuth_Angle, &
                                    Flux_Zenith_Angle    = Flux_Zenith_Angle    , &
                                    Year                 = Year                 , &
                                    Month                = Month                , &
                                    Day                  = Day                    )

 OBJECTS:
       geo:                  Geometry object from which component values
                             are to be retrieved.
                             UNITS:      N/A
                             TYPE:       CRTM_Geometry_type
                             DIMENSION:  Scalar or any rank
                             ATTRIBUTES: INTENT(IN OUT)

 OPTIONAL OUTPUTS:
       iFOV:                 Sensor field-of-view index.
                             UNITS:      N/A
                             TYPE:       INTEGER
                             DIMENSION:  Scalar or same as geo input
                             ATTRIBUTES: INTENT(OUT), OPTIONAL

       Longitude:            Earth longitude
                             UNITS:      degrees East (0->360)
                             TYPE:       REAL(fp)
                             DIMENSION:  Scalar or same as geo input
                             ATTRIBUTES: INTENT(OUT), OPTIONAL

       Latitude:             Earth latitude.
                             UNITS:      degrees North (-90->+90)
                             TYPE:       REAL(fp)
                             DIMENSION:  Scalar or same as geo input
                             ATTRIBUTES: INTENT(OUT), OPTIONAL

       Surface_Altitude:     Altitude of the Earth's surface at the specifed
                             lon/lat location.
                             UNITS:      metres (m)
                             TYPE:       REAL(fp)
                             DIMENSION:  Scalar or same as geo input
                             ATTRIBUTES: INTENT(OUT), OPTIONAL

       Sensor_Scan_Angle:    The sensor scan angle from nadir.
                             UNITS:      degrees
                             TYPE:       REAL(fp)
                             DIMENSION:  Scalar or same as geo input
                             ATTRIBUTES: INTENT(OUT), OPTIONAL

       Sensor_Zenith_Angle:  The zenith angle from the field-of-view
                             to the sensor.
                             UNITS:      degrees
                             TYPE:       REAL(fp)
                             DIMENSION:  Scalar or same as geo input
                             ATTRIBUTES: INTENT(OUT), OPTIONAL

       Sensor_Azimuth_Angle: The azimuth angle subtended by the horizontal
                             projection of a direct line from the satellite
                             to the FOV and the North-South axis measured
                             clockwise from North.
                             UNITS:      degrees from North (0->360)
                             TYPE:       REAL(fp)
                             DIMENSION:  Scalar or same as geo input
                             ATTRIBUTES: INTENT(OUT), OPTIONAL
       Source_Zenith_Angle:  The zenith angle from the field-of-view
                             to a source (sun or moon).
                             UNITS:      degrees
                             TYPE:       REAL(fp)
                             DIMENSION:  Scalar or same as geo input
                             ATTRIBUTES: INTENT(OUT), OPTIONAL

       Source_Azimuth_Angle: The azimuth angle subtended by the horizontal
                             projection of a direct line from the source
                             to the FOV and the North-South axis measured
                             clockwise from North.
                             UNITS:      degrees from North (0->360)
                             TYPE:       REAL(fp)
                             DIMENSION:  Scalar or same as geo input
                             ATTRIBUTES: INTENT(OUT), OPTIONAL

       Flux_Zenith_Angle:    The zenith angle used to approximate downwelling
                             flux transmissivity
                             UNITS:      degrees
                             TYPE:       REAL(fp)
                             DIMENSION:  Scalar or same as geo input
                             ATTRIBUTES: INTENT(OUT), OPTIONAL

       Year:                 The year in 4-digit format, e.g. 1997.
                             UNITS:      N/A
                             TYPE:       INTEGER
                             DIMENSION:  Scalar or same as geo input
                             ATTRIBUTES: INTENT(OUT), OPTIONAL

       Month:                The month of the year (1-12).
                             UNITS:      N/A
                             TYPE:       INTEGER
                             DIMENSION:  Scalar or same as geo input
                             ATTRIBUTES: INTENT(OUT), OPTIONAL

       Day:                  The day of the month (1-28/29/30/31).
                             UNITS:      N/A
                             TYPE:       INTEGER
                             DIMENSION:  Scalar or same as geo input
                             ATTRIBUTES: INTENT(OUT), OPTIONAL
```

A.7.7 CRTM_Geometry_InquireFile interface

```
NAME:
      CRTM_Geometry_InquireFile

PURPOSE:
      Function to inquire CRTM Geometry object files.
```

```
CALLING SEQUENCE:
      Error_Status = CRTM_Geometry_InquireFile( Filename               , &
                                                n_Profiles = n_Profiles  )

INPUTS:
      Filename:       Character string specifying the name of a
                      CRTM Geometry data file to read.
                      UNITS:      N/A
                      TYPE:       CHARACTER(*)
                      DIMENSION:  Scalar
                      ATTRIBUTES: INTENT(IN)

OPTIONAL OUTPUTS:
      n_Profiles:     The number of profiles for which their is geometry
                      information in the data file.
                      UNITS:      N/A
                      TYPE:       INTEGER
                      DIMENSION:  Scalar
                      ATTRIBUTES: OPTIONAL, INTENT(OUT)

FUNCTION RESULT:
      Error_Status:   The return value is an integer defining the error status.
                      The error codes are defined in the Message_Handler module.
                      If == SUCCESS, the file inquire was successful
                         == FAILURE, an unrecoverable error occurred.
                      UNITS:      N/A
                      TYPE:       INTEGER
                      DIMENSION:  Scalar
```

A.7.8 CRTM_Geometry_Inspect interface

```
NAME:
      CRTM_Geometry_Inspect

PURPOSE:
      Subroutine to print the contents of a CRTM Geometry object to stdout.

CALLING SEQUENCE:
      CALL CRTM_Geometry_Inspect( geo )

INPUTS:
      geo:  CRTM Geometry object to display.
            UNITS:      N/A
            TYPE:       CRTM_Geometry_type
            DIMENSION:  Scalar
            ATTRIBUTES: INTENT(IN)
```

A.7.9 CRTM_Geometry_IsValid interface

```
NAME:
      CRTM_Geometry_IsValid

PURPOSE:
      Non-pure function to perform some simple validity checks on a
      CRTM Geometry object.

      If invalid data is found, a message is printed to stdout.

CALLING SEQUENCE:
      result = CRTM_Geometry_IsValid( geo )

        or

      IF ( CRTM_Geometry_IsValid( geo ) ) THEN....

OBJECTS:
      geo:       CRTM Geometry object which is to have its
                 contents checked.
                 UNITS:      N/A
                 TYPE:       CRTM_Geometry_type
                 DIMENSION:  Scalar
                 ATTRIBUTES: INTENT(IN)

FUNCTION RESULT:
      result:    Logical variable indicating whether or not the input
                 passed the check.
                 If == .FALSE., Geometry object is unused or contains
                                invalid data.
                    == .TRUE.,  Geometry object can be used in CRTM.
                 UNITS:      N/A
                 TYPE:       LOGICAL
                 DIMENSION:  Scalar
```

A.7.10 CRTM_Geometry_ReadFile interface

```
NAME:
      CRTM_Geometry_ReadFile

PURPOSE:
      Function to read CRTM Geometry object files.

CALLING SEQUENCE:
      Error_Status = CRTM_Geometry_ReadFile( Filename               , &
                                             Geometry               , &
                                             Quiet      = Quiet     , &
                                             No_Close   = No_Close  , &
                                             n_Profiles = n_Profiles  )

INPUTS:
      Filename:     Character string specifying the name of an
                    a Geometry data file to read.
                    UNITS:      N/A
                    TYPE:       CHARACTER(*)
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN)

OUTPUTS:
      Geometry:     CRTM Geometry object array containing the
                    data read from file.
                    UNITS:      N/A
                    TYPE:       CRTM_Geometry_type
                    DIMENSION:  Rank-1
                    ATTRIBUTES: INTENT(OUT)

OPTIONAL INPUTS:
      Quiet:        Set this logical argument to suppress INFORMATION
                    messages being printed to stdout
                    If == .FALSE., INFORMATION messages are OUTPUT [DEFAULT].
                       == .TRUE.,  INFORMATION messages are SUPPRESSED.
                    If not specified, default is .FALSE.
                    UNITS:      N/A
                    TYPE:       LOGICAL
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN), OPTIONAL

      No_Close:     Set this logical argument to NOT close the file upon exit.
                    If == .FALSE., the input file is closed upon exit [DEFAULT]
                       == .TRUE.,  the input file is NOT closed upon exit.
                    If not specified, default is .FALSE.
                    UNITS:      N/A
                    TYPE:       LOGICAL
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN), OPTIONAL

OPTIONAL OUTPUTS:
      n_Profiles:   The number of profiles for which data was read.
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Scalar
                    ATTRIBUTES: OPTIONAL, INTENT(OUT)

FUNCTION RESULT:
      Error_Status: The return value is an integer defining the error status.
                    The error codes are defined in the Message_Handler module.
                    If == SUCCESS, the file read was successful
                       == FAILURE, an unrecoverable error occurred.
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Scalar
```

A.7.11 CRTM_Geometry_ReadRecord interface

```
NAME:
      CRTM_Geometry_ReadRecord

PURPOSE:
      Utility function to read a single Geometry data record

CALLING SEQUENCE:
      Error_Status = CRTM_Geometry_ReadRecord( FileID, Geometry )

INPUTS:
      FileID:       Logical unit number from which to read data.
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN)

OUTPUTS:
      Geometry:     CRTM Geometry object containing the data read in.
                    UNITS:      N/A
                    TYPE:       CRTM_Geometry_type
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(OUT)

FUNCTION RESULT:
      Error_Status: The return value is an integer defining the error status.
                    The error codes are defined in the Message_Handler module.
                    If == SUCCESS, the read was successful
                       == FAILURE, an unrecoverable error occurred.
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Scalar
```

A.7.12 CRTM_Geometry_SetValue interface

```
NAME:
      CRTM_Geometry_SetValue

PURPOSE:
      Elemental subroutine to set the values of CRTM Geometry
      object components.

CALLING SEQUENCE:
      CALL CRTM_Geometry_SetValue( geo, &
                                   iFOV                = iFOV               , &
                                   Longitude           = Longitude          , &
                                   Latitude            = Latitude            , &
                                   Surface_Altitude    = Surface_Altitude    , &
                                   Sensor_Scan_Angle   = Sensor_Scan_Angle   , &
                                   Sensor_Zenith_Angle = Sensor_Zenith_Angle , &
                                   Sensor_Azimuth_Angle = Sensor_Azimuth_Angle, &
                                   Source_Zenith_Angle = Source_Zenith_Angle , &
                                   Source_Azimuth_Angle = Source_Azimuth_Angle, &
                                   Flux_Zenith_Angle   = Flux_Zenith_Angle   , &
                                   Year                = Year                , &
                                   Month               = Month               , &
                                   Day                 = Day                   )

OBJECTS:
      geo:                  Geometry object for which component values
                            are to be set.
                            UNITS:      N/A
                            TYPE:       CRTM_Geometry_type
                            DIMENSION:  Scalar or any rank
                            ATTRIBUTES: INTENT(IN OUT)

OPTIONAL INPUTS:
      iFOV:                 Sensor field-of-view index.
                            UNITS:      N/A
                            TYPE:       INTEGER
                            DIMENSION:  Scalar or same as geo input
                            ATTRIBUTES: INTENT(IN), OPTIONAL

      Longitude:            Earth longitude
                            UNITS:      degrees East (0->360)
                            TYPE:       REAL(fp)
                            DIMENSION:  Scalar or same as geo input
                            ATTRIBUTES: INTENT(IN), OPTIONAL

      Latitude:             Earth latitude.
                            UNITS:      degrees North (-90->+90)
                            TYPE:       REAL(fp)
                            DIMENSION:  Scalar or same as geo input
                            ATTRIBUTES: INTENT(IN), OPTIONAL

      Surface_Altitude:     Altitude of the Earth's surface at the specifed
                            lon/lat location.
                            UNITS:      metres (m)
                            TYPE:       REAL(fp)
                            DIMENSION:  Scalar or same as geo input
                            ATTRIBUTES: INTENT(IN), OPTIONAL

      Sensor_Scan_Angle:    The sensor scan angle from nadir.
                            UNITS:      degrees
                            TYPE:       REAL(fp)
                            DIMENSION:  Scalar or same as geo input
                            ATTRIBUTES: INTENT(IN), OPTIONAL

      Sensor_Zenith_Angle:  The zenith angle from the field-of-view
                            to the sensor.
```

```
                        UNITS:      degrees
                        TYPE:       REAL(fp)
                        DIMENSION:  Scalar or same as geo input
                        ATTRIBUTES: INTENT(IN), OPTIONAL

Sensor_Azimuth_Angle: The azimuth angle subtended by the horizontal
                      projection of a direct line from the satellite
                      to the FOV and the North-South axis measured
                      clockwise from North.
                      UNITS:      degrees from North (0->360)
                      TYPE:       REAL(fp)
                      DIMENSION:  Scalar or same as geo input
                      ATTRIBUTES: INTENT(IN), OPTIONAL

Source_Zenith_Angle:  The zenith angle from the field-of-view
                      to a source (sun or moon).
                      UNITS:      degrees
                      TYPE:       REAL(fp)
                      DIMENSION:  Scalar or same as geo input
                      ATTRIBUTES: INTENT(IN), OPTIONAL

Source_Azimuth_Angle: The azimuth angle subtended by the horizontal
                      projection of a direct line from the source
                      to the FOV and the North-South axis measured
                      clockwise from North.
                      UNITS:      degrees from North (0->360)
                      TYPE:       REAL(fp)
                      DIMENSION:  Scalar or same as geo input
                      ATTRIBUTES: INTENT(IN), OPTIONAL

Flux_Zenith_Angle:    The zenith angle used to approximate downwelling
                      flux transmissivity
                      UNITS:      degrees
                      TYPE:       REAL(fp)
                      DIMENSION:  Scalar or same as geo input
                      ATTRIBUTES: INTENT(IN), OPTIONAL

Year:                 The year in 4-digit format, e.g. 1997.
                      UNITS:      N/A
                      TYPE:       INTEGER
                      DIMENSION:  Scalar or same as geo input
                      ATTRIBUTES: INTENT(IN), OPTIONAL

Month:                The month of the year (1-12).
                      UNITS:      N/A
                      TYPE:       INTEGER
                      DIMENSION:  Scalar or same as geo input
                      ATTRIBUTES: INTENT(IN), OPTIONAL

Day:                  The day of the month (1-28/29/30/31).
                      UNITS:      N/A
                      TYPE:       INTEGER
                      DIMENSION:  Scalar or same as geo input
                      ATTRIBUTES: INTENT(IN), OPTIONAL
```

A.7.13 CRTM_Geometry_WriteFile interface

```
NAME:
      CRTM_Geometry_WriteFile

PURPOSE:
      Function to write CRTM Geometry object files.

CALLING SEQUENCE:
      Error_Status = CRTM_Geometry_WriteFile( Filename           , &
                                              Geometry           , &
                                              Quiet    = Quiet   , &
                                              No_Close = No_Close  )

INPUTS:
      Filename:     Character string specifying the name of the
                    Geometry format data file to write.
                    UNITS:      N/A
                    TYPE:       CHARACTER(*)
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN)

      Geometry:     CRTM Geometry object array containing the Geometry
                    data to write.
                    UNITS:      N/A
                    TYPE:       CRTM_Geometry_type
                    DIMENSION:  Rank-1
                    ATTRIBUTES: INTENT(IN)

OPTIONAL INPUTS:
      Quiet:        Set this logical argument to suppress INFORMATION
                    messages being printed to stdout
                    If == .FALSE., INFORMATION messages are OUTPUT [DEFAULT].
                       == .TRUE.,  INFORMATION messages are SUPPRESSED.
                    If not specified, default is .FALSE.
                    UNITS:      N/A
                    TYPE:       LOGICAL
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN), OPTIONAL

      No_Close:     Set this logical argument to NOT close the file upon exit.
                    If == .FALSE., the input file is closed upon exit [DEFAULT]
                       == .TRUE.,  the input file is NOT closed upon exit.
                    If not specified, default is .FALSE.
                    UNITS:      N/A
                    TYPE:       LOGICAL
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN), OPTIONAL
```

```
FUNCTION RESULT:
      Error_Status: The return value is an integer defining the error status.
                    The error codes are defined in the Message_Handler module.
                    If == SUCCESS, the file write was successful
                       == FAILURE, an unrecoverable error occurred.
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Scalar

SIDE EFFECTS:
      - If the output file already exists, it is overwritten.
      - If an error occurs during *writing*, the output file is deleted before
        returning to the calling routine.
```

A.7.14 CRTM_Geometry_WriteRecord interface

```
NAME:
      CRTM_Geometry_WriteRecord

PURPOSE:
      Function to write a single Geometry data record

CALLING SEQUENCE:
      Error_Status = CRTM_Geometry_WriteRecord( FileID, Geometry )

INPUTS:
      FileID:       Logical unit number to which data is written
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN)

      Geometry:     CRTM Geometry object containing the data to write.
                    UNITS:      N/A
                    TYPE:       CRTM_Geometry_type
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN)

FUNCTION RESULT:
      Error_Status: The return value is an integer defining the error status.
                    The error codes are defined in the Message_Handler module.
                    If == SUCCESS the record write was successful
                       == FAILURE an unrecoverable error occurred.
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Scalar
```

A.8 RTSolution Structure

```
TYPE :: CRTM_RTSolution_type
  ! Allocation indicator
  LOGICAL :: Is_Allocated = .FALSE.
  ! Dimensions
  INTEGER :: n_Layers = 0  ! K
  ! Sensor information
  CHARACTER(STRLEN) :: Sensor_ID        = ''
  INTEGER           :: WMO_Satellite_ID = INVALID_WMO_SATELLITE_ID
  INTEGER           :: WMO_Sensor_ID    = INVALID_WMO_SENSOR_ID
  INTEGER           :: Sensor_Channel   = 0
  ! RT algorithm information
  CHARACTER(STRLEN) :: RT_Algorithm_Name = ''
  ! Forward radiative transfer intermediate results for a single channel
  !    These components are not defined when they are used as TL, AD
  !    and K variables
  REAL(fp) :: SOD                     = ZERO  ! Scattering Optical Depth
  REAL(fp) :: Surface_Emissivity      = ZERO
  REAL(fp) :: Up_Radiance             = ZERO
  REAL(fp) :: Down_Radiance           = ZERO
  REAL(fp) :: Down_Solar_Radiance     = ZERO
  REAL(fp) :: Surface_Planck_Radiance = ZERO
  REAL(fp), ALLOCATABLE :: Upwelling_Radiance(:)   ! K
  REAL(fp), ALLOCATABLE :: Layer_Optical_Depth(:)  ! K
  ! Radiative transfer results for a single channel/node
  REAL(fp) :: Radiance               = ZERO
  REAL(fp) :: Brightness_Temperature = ZERO
END TYPE CRTM_RTSolution_type
```

Figure A.8: CRTM_RTSolution_type structure definition.

A.8.1 CRTM_RTSolution_Associated interface

```
NAME:
      CRTM_RTSolution_Associated

PURPOSE:
      Elemental function to test the status of the allocatable components
      of a CRTM RTSolution object.

CALLING SEQUENCE:
      Status = CRTM_RTSolution_Associated( RTSolution )

OBJECTS:
      RTSolution:   RTSolution structure which is to have its member's
                    status tested.
                    UNITS:      N/A
                    TYPE:       CRTM_RTSolution_type
                    DIMENSION:  Scalar or any rank
                    ATTRIBUTES: INTENT(IN)

FUNCTION RESULT:
      Status:       The return value is a logical value indicating the
                    status of the RTSolution members.
                      .TRUE.  - if the array components are allocated.
                      .FALSE. - if the array components are not allocated.
                    UNITS:      N/A
                    TYPE:       LOGICAL
                    DIMENSION:  Same as input RTSolution argument
```

A.8.2 CRTM_RTSolution_Compare interface

```
NAME:
      CRTM_RTSolution_Compare

PURPOSE:
      Elemental function to compare two CRTM_RTSolution objects to within
      a user specified number of significant figures.

CALLING SEQUENCE:
      is_comparable = CRTM_RTSolution_Compare( x, y, n_SigFig=n_SigFig )

OBJECTS:
      x, y:          Two CRTM RTSolution objects to be compared.
                     UNITS:      N/A
                     TYPE:       CRTM_RTSolution_type
                     DIMENSION:  Scalar or any rank
                     ATTRIBUTES: INTENT(IN)

OPTIONAL INPUTS:
      n_SigFig:      Number of significant figure to compare floating point
```

```
                     components.
                     UNITS:      N/A
                     TYPE:       INTEGER
                     DIMENSION:  Conformable with inputs
                     ATTRIBUTES: INTENT(IN), OPTIONAL

FUNCTION RESULT:
      is_comparable: Logical value indicating whether the inputs are
                     comparable.
                     UNITS:      N/A
                     TYPE:       LOGICAL
                     DIMENSION:  Same as inputs.
```

A.8.3 CRTM_RTSolution_Create interface

```
NAME:
      CRTM_RTSolution_Create

PURPOSE:
      Elemental subroutine to create an instance of the CRTM RTSolution object.

CALLING SEQUENCE:
      CALL CRTM_RTSolution_Create( RTSolution, n_Layers )

OBJECTS:
      RTSolution:   RTSolution structure.
                    UNITS:      N/A
                    TYPE:       CRTM_RTSolution_type
                    DIMENSION:  Scalar or any rank
                    ATTRIBUTES: INTENT(OUT)

INPUTS:
      n_Layers:     Number of layers for which there is RTSolution data.
                    Must be > 0.
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Same as RTSolution object
                    ATTRIBUTES: INTENT(IN)
```

A.8.4 CRTM_RTSolution_DefineVersion interface

```
NAME:
      CRTM_RTSolution_DefineVersion

PURPOSE:
      Subroutine to return the module version information.
```

```
CALLING SEQUENCE:
      CALL CRTM_RTSolution_DefineVersion( Id )

OUTPUTS:
      Id:            Character string containing the version Id information
                     for the module.
                     UNITS:      N/A
                     TYPE:       CHARACTER(*)
                     DIMENSION:  Scalar
                     ATTRIBUTES: INTENT(OUT)
```

A.8.5 CRTM_RTSolution_Destroy interface

```
NAME:
      CRTM_RTSolution_Destroy

PURPOSE:
      Elemental subroutine to re-initialize CRTM RTSolution objects.

CALLING SEQUENCE:
      CALL CRTM_RTSolution_Destroy( RTSolution )

OBJECTS:
      RTSolution:   Re-initialized RTSolution structure.
                    UNITS:      N/A
                    TYPE:       CRTM_RTSolution_type
                    DIMENSION:  Scalar OR any rank
                    ATTRIBUTES: INTENT(OUT)
```

A.8.6 CRTM_RTSolution_InquireFile interface

```
NAME:
      CRTM_RTSolution_InquireFile

PURPOSE:
      Function to inquire CRTM RTSolution object files.

CALLING SEQUENCE:
      Error_Status = CRTM_RTSolution_InquireFile( Filename               , &
                                                  n_Channels = n_Channels, &
                                                  n_Profiles = n_Profiles  )

INPUTS:
      Filename:       Character string specifying the name of a
                      CRTM RTSolution data file to read.
                      UNITS:      N/A
```

```
                    TYPE:       CHARACTER(*)
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN)

OPTIONAL OUTPUTS:
      n_Channels:   The number of spectral channels for which there is
                    data in the file.
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Scalar
                    ATTRIBUTES: OPTIONAL, INTENT(OUT)

      n_Profiles:   The number of profiles in the data file.
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Scalar
                    ATTRIBUTES: OPTIONAL, INTENT(OUT)

FUNCTION RESULT:
      Error_Status: The return value is an integer defining the error status.
                    The error codes are defined in the Message_Handler module.
                    If == SUCCESS, the file inquire was successful
                       == FAILURE, an unrecoverable error occurred.
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Scalar
```

A.8.7 CRTM_RTSolution_Inspect interface

```
NAME:
      CRTM_RTSolution_Inspect

PURPOSE:
      Subroutine to print the contents of a CRTM RTSolution object to stdout.

CALLING SEQUENCE:
      CALL CRTM_RTSolution_Inspect( RTSolution )

INPUTS:
      RTSolution:   CRTM RTSolution object to display.
                    UNITS:      N/A
                    TYPE:       CRTM_RTSolution_type
                    DIMENSION:  Scalar or Rank-2 (n_channels x n_profiles)
                    ATTRIBUTES: INTENT(IN)
```

A.8.8 CRTM_RTSolution_ReadFile interface

```
NAME:
      CRTM_RTSolution_ReadFile

PURPOSE:
      Function to read CRTM RTSolution object files.

CALLING SEQUENCE:
      Error_Status = CRTM_RTSolution_ReadFile( Filename                , &
                                               RTSolution              , &
                                               Quiet      = Quiet      , &
                                               n_Channels = n_Channels , &
                                               n_Profiles = n_Profiles , &

INPUTS:
      Filename:     Character string specifying the name of an
                    RTSolution format data file to read.
                    UNITS:      N/A
                    TYPE:       CHARACTER(*)
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN)

OUTPUTS:
      RTSolution:   CRTM RTSolution object array containing the RTSolution
                    data.
                    UNITS:      N/A
                    TYPE:       CRTM_RTSolution_type
                    DIMENSION:  Rank-2 (n_Channels x n_Profiles)
                    ATTRIBUTES: INTENT(OUT)

OPTIONAL INPUTS:
      Quiet:        Set this logical argument to suppress INFORMATION
                    messages being printed to stdout
                    If == .FALSE., INFORMATION messages are OUTPUT [DEFAULT].
                       == .TRUE.,  INFORMATION messages are SUPPRESSED.
                    If not specified, default is .FALSE.
                    UNITS:      N/A
                    TYPE:       LOGICAL
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN), OPTIONAL

OPTIONAL OUTPUTS:
      n_Channels:   The number of channels for which data was read.
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Scalar
                    ATTRIBUTES: OPTIONAL, INTENT(OUT)

      n_Profiles:   The number of profiles for which data was read.
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Scalar
                    ATTRIBUTES: OPTIONAL, INTENT(OUT)


FUNCTION RESULT:
      Error_Status: The return value is an integer defining the error status.
                    The error codes are defined in the Message_Handler module.
                    If == SUCCESS, the file read was successful
                       == FAILURE, an unrecoverable error occurred.
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Scalar
```

A.8.9 CRTM_RTSolution_WriteFile interface

```
  NAME:
        CRTM_RTSolution_WriteFile

  PURPOSE:
        Function to write CRTM RTSolution object files.

  CALLING SEQUENCE:
        Error_Status = CRTM_RTSolution_WriteFile( Filename     , &
                                                  RTSolution   , &
                                                  Quiet = Quiet  )

  INPUTS:
        Filename:     Character string specifying the name of the
                      RTSolution format data file to write.
                      UNITS:      N/A
                      TYPE:       CHARACTER(*)
                      DIMENSION:  Scalar
                      ATTRIBUTES: INTENT(IN)

        RTSolution:   CRTM RTSolution object array containing the RTSolution
                      data.
                      UNITS:      N/A
                      TYPE:       CRTM_RTSolution_type
                      DIMENSION:  Rank-2 (n_Channels x n_Profiles)
                      ATTRIBUTES: INTENT(IN)

  OPTIONAL INPUTS:
        Quiet:        Set this logical argument to suppress INFORMATION
                      messages being printed to stdout
                      If == .FALSE., INFORMATION messages are OUTPUT [DEFAULT].
                         == .TRUE.,  INFORMATION messages are SUPPRESSED.
                      If not specified, default is .FALSE.
                      UNITS:      N/A
                      TYPE:       LOGICAL
                      DIMENSION:  Scalar
                      ATTRIBUTES: INTENT(IN), OPTIONAL
```

```
FUNCTION RESULT:
      Error_Status: The return value is an integer defining the error status.
                    The error codes are defined in the Message_Handler module.
                    If == SUCCESS, the file write was successful
                       == FAILURE, an unrecoverable error occurred.
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Scalar

SIDE EFFECTS:
      - If the output file already exists, it is overwritten.
      - If an error occurs during *writing*, the output file is deleted before
        returning to the calling routine.
```

A.8.10 CRTM_RTSolution_Zero interface

```
NAME:
      CRTM_RTSolution_Zero

PURPOSE:
      Elemental subroutine to zero out the data components
      in a CRTM RTSolution object.

CALLING SEQUENCE:
      CALL CRTM_RTSolution_Zero( rts )

OUTPUTS:
      rts:          CRTM RTSolution structure in which the data components
                    are to be zeroed out.
                    UNITS:      N/A
                    TYPE:       CRTM_RTSolution_type
                    DIMENSION:  Scalar or any rank
                    ATTRIBUTES: INTENT(IN OUT)

COMMENTS:
      - The dimension components of the structure are *NOT* set to zero.
      - The sensor infomration and RT algorithm components are
        *NOT* reset in this routine.
```

A.9 `Options` Structure

```
TYPE :: CRTM_Options_type
  ! Allocation indicator
  LOGICAL :: Is_Allocated = .FALSE.

  ! Input checking on by default
  LOGICAL :: Check_Input = .TRUE.

  ! User defined MW water emissivity algorithm
  LOGICAL :: Use_Old_MWSSEM = .FALSE.

  ! Antenna correction application
  LOGICAL :: Use_Antenna_Correction = .FALSE.

  ! NLTE radiance correction is ON by default
  LOGICAL :: Apply_NLTE_Correction = .TRUE.

  ! RT Algorithm is set to ADA by default
  INTEGER(Long) :: RT_Algorithm_Id = RT_ADA

  ! Aircraft flight level pressure
  ! Value > 0 turns "on" the aircraft option
  REAL(Double) :: Aircraft_Pressure = -ONE

  ! User defined number of RT solver streams (streams up + streams down)
  LOGICAL       :: Use_n_Streams = .FALSE.
  INTEGER(Long) :: n_Streams = 0

  ! Scattering switch. Default is for
  ! Cloud/Aerosol scattering to be included.
  LOGICAL :: Include_Scattering = .TRUE.

  ! User defined emissivity/reflectivity
  ! ...Dimensions
  INTEGER(Long) :: n_Channels = 0  ! L dimension
  ! ...Index into channel-specific components
  INTEGER(Long) :: Channel = 0
  ! ...Emissivity optional arguments
  LOGICAL :: Use_Emissivity = .FALSE.
  REAL(Double), ALLOCATABLE :: Emissivity(:)  ! L
  ! ...Direct reflectivity optional arguments
  LOGICAL :: Use_Direct_Reflectivity = .FALSE.
  REAL(Double), ALLOCATABLE :: Direct_Reflectivity(:) ! L

  ! SSU instrument input
  TYPE(SSU_Input_type) :: SSU

  ! Zeeman-splitting input
  TYPE(Zeeman_Input_type) :: Zeeman
END TYPE CRTM_Options_type
```

Figure A.9: CRTM_Options_type structure definition.

A.9.1 CRTM_Options_Associated interface

```
NAME:
      CRTM_Options_Associated

PURPOSE:
      Elemental function to test the status of the allocatable components
      of a CRTM Options object.

CALLING SEQUENCE:
      Status = CRTM_Options_Associated( Options )

OBJECTS:
      Options:      Options structure which is to have its member's
                    status tested.
                    UNITS:      N/A
                    TYPE:       CRTM_Options_type
                    DIMENSION:  Scalar or any rank
                    ATTRIBUTES: INTENT(IN)

FUNCTION RESULT:
      Status:       The return value is a logical value indicating the
                    status of the Options members.
                      .TRUE.  - if the array components are allocated.
                      .FALSE. - if the array components are not allocated.
                    UNITS:      N/A
                    TYPE:       LOGICAL
                    DIMENSION:  Same as input Options argument
```

A.9.2 CRTM_Options_Create interface

```
NAME:
      CRTM_Options_Create

PURPOSE:
      Elemental subroutine to create an instance of the CRTM Options object.

CALLING SEQUENCE:
      CALL CRTM_Options_Create( Options, n_Channels )

OBJECTS:
      Options:      Options structure.
                    UNITS:      N/A
                    TYPE:       CRTM_Options_type
                    DIMENSION:  Scalar or any rank
                    ATTRIBUTES: INTENT(OUT)

INPUTS:
      n_Channels:   Number of channels for which there is Options data.
```

```
                    Must be > 0.
                    This dimension only applies to the emissivity-related
                    components.
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Same as Options object
                    ATTRIBUTES: INTENT(IN)
```

A.9.3 CRTM_Options_DefineVersion interface

```
NAME:
      CRTM_Options_DefineVersion

PURPOSE:
      Subroutine to return the module version information.

CALLING SEQUENCE:
      CALL CRTM_Options_DefineVersion( Id )

OUTPUTS:
      Id:            Character string containing the version Id information
                     for the module.
                     UNITS:      N/A
                     TYPE:       CHARACTER(*)
                     DIMENSION:  Scalar
                     ATTRIBUTES: INTENT(OUT)
```

A.9.4 CRTM_Options_Destroy interface

```
NAME:
      CRTM_Options_Destroy

PURPOSE:
      Elemental subroutine to re-initialize CRTM Options objects.

CALLING SEQUENCE:
      CALL CRTM_Options_Destroy( Options )

OBJECTS:
      Options:      Re-initialized Options structure.
                    UNITS:      N/A
                    TYPE:       CRTM_Options_type
                    DIMENSION:  Scalar OR any rank
                    ATTRIBUTES: INTENT(OUT)
```

A.9.5 CRTM_Options_InquireFile interface

```
NAME:
      CRTM_Options_InquireFile

PURPOSE:
      Function to inquire CRTM Options object files.

CALLING SEQUENCE:
      Error_Status = CRTM_Options_InquireFile( &
                       Filename               , &
                       n_Profiles = n_Profiles  )

INPUTS:
      Filename:       Character string specifying the name of a
                      CRTM Options data file to read.
                      UNITS:      N/A
                      TYPE:       CHARACTER(*)
                      DIMENSION:  Scalar
                      ATTRIBUTES: INTENT(IN)

OPTIONAL OUTPUTS:
      n_Profiles:     The number of profiles in the data file.
                      UNITS:      N/A
                      TYPE:       INTEGER
                      DIMENSION:  Scalar
                      ATTRIBUTES: OPTIONAL, INTENT(OUT)

FUNCTION RESULT:
      Error_Status:   The return value is an integer defining the error status.
                      The error codes are defined in the Message_Handler module.
                      If == SUCCESS, the file inquire was successful
                         == FAILURE, an unrecoverable error occurred.
                      UNITS:      N/A
                      TYPE:       INTEGER
                      DIMENSION:  Scalar
```

A.9.6 CRTM_Options_Inspect interface

```
NAME:
      CRTM_Options_Inspect

PURPOSE:
      Subroutine to print the contents of a CRTM Options object to stdout.

CALLING SEQUENCE:
      CALL CRTM_Options_Inspect( Options )

INPUTS:
      Options:       CRTM Options object to display.
                     UNITS:      N/A
                     TYPE:       CRTM_Options_type
                     DIMENSION:  Scalar
                     ATTRIBUTES: INTENT(IN)
```

A.9.7 CRTM_Options_IsValid interface

```
NAME:
      CRTM_Options_IsValid

PURPOSE:
      Non-pure function to perform some simple validity checks on a
      CRTM Options object.

      If invalid data is found, a message is printed to stdout.

CALLING SEQUENCE:
      result = CRTM_Options_IsValid( opt )

        or

      IF ( CRTM_Options_IsValid( opt ) ) THEN....

OBJECTS:
      opt:       CRTM Options object which is to have its
                 contents checked.
                 UNITS:      N/A
                 TYPE:       CRTM_Options_type
                 DIMENSION:  Scalar
                 ATTRIBUTES: INTENT(IN)

FUNCTION RESULT:
      result:    Logical variable indicating whether or not the input
                 passed the check.
                 If == .FALSE., Options object is unused or contains
                                invalid data.
                    == .TRUE.,  Options object can be used in CRTM.
                 UNITS:      N/A
                 TYPE:       LOGICAL
                 DIMENSION:  Scalar
```

A.9.8 CRTM_Options_ReadFile interface

```
NAME:
      CRTM_Options_ReadFile
```

```
PURPOSE:
      Function to read CRTM Options object files.

CALLING SEQUENCE:
      Error_Status = CRTM_Options_ReadFile( &
                       Filename              , &
                       Options               , &
                       Quiet      = Quiet     , &
                       n_Profiles = n_Profiles  )

INPUTS:
      Filename:     Character string specifying the name of an
                    Options format data file to read.
                    UNITS:      N/A
                    TYPE:       CHARACTER(*)
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN)

OUTPUTS:
      Options:      CRTM Options object array containing the Options
                    data.
                    UNITS:      N/A
                    TYPE:       CRTM_Options_type
                    DIMENSION:  Rank-1 (n_Profiles)
                    ATTRIBUTES: INTENT(OUT)

OPTIONAL INPUTS:
      Quiet:        Set this logical argument to suppress INFORMATION
                    messages being printed to stdout
                    If == .FALSE., INFORMATION messages are OUTPUT [DEFAULT].
                       == .TRUE.,  INFORMATION messages are SUPPRESSED.
                    If not specified, default is .FALSE.
                    UNITS:      N/A
                    TYPE:       LOGICAL
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN), OPTIONAL

OPTIONAL OUTPUTS:
      n_Profiles:   The number of profiles for which data was read.
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Scalar
                    ATTRIBUTES: OPTIONAL, INTENT(OUT)


FUNCTION RESULT:
      Error_Status: The return value is an integer defining the error status.
                    The error codes are defined in the Message_Handler module.
                    If == SUCCESS, the file read was successful
                       == FAILURE, an unrecoverable error occurred.
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Scalar
```

A.9.9 CRTM_Options_WriteFile interface

```
NAME:
      CRTM_Options_WriteFile

PURPOSE:
      Function to write CRTM Options object files.

CALLING SEQUENCE:
      Error_Status = CRTM_Options_WriteFile( Filename     , &
                                             Options      , &
                                             Quiet = Quiet  )

INPUTS:
      Filename:     Character string specifying the name of the
                    Options format data file to write.
                    UNITS:      N/A
                    TYPE:       CHARACTER(*)
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN)

      Options:      CRTM Options object array containing the Options
                    data.
                    UNITS:      N/A
                    TYPE:       CRTM_Options_type
                    DIMENSION:  Rank-1 (n_Profiles)
                    ATTRIBUTES: INTENT(IN)

OPTIONAL INPUTS:
      Quiet:        Set this logical argument to suppress INFORMATION
                    messages being printed to stdout
                    If == .FALSE., INFORMATION messages are OUTPUT [DEFAULT].
                       == .TRUE.,  INFORMATION messages are SUPPRESSED.
                    If not specified, default is .FALSE.
                    UNITS:      N/A
                    TYPE:       LOGICAL
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN), OPTIONAL

FUNCTION RESULT:
      Error_Status: The return value is an integer defining the error status.
                    The error codes are defined in the Message_Handler module.
                    If == SUCCESS, the file write was successful
                       == FAILURE, an unrecoverable error occurred.
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Scalar

SIDE EFFECTS:
```

```
- If the output file already exists, it is overwritten.
- If an error occurs during *writing*, the output file is deleted before
  returning to the calling routine.
```

A.10 SSU_Input Structure

The SSU_Input structure is a component of the Options input structure. Note in figure A.10 that the structure is declared as PRIVATE. As such, the only way to set values in, or get values from, the structure is via the SSU_Input_SetValue or SSU_Input_GetValue subroutines respectively.

```
TYPE :: SSU_Input_type
  PRIVATE
  ! Release and version information
  INTEGER(Long) :: Release = SSU_INPUT_RELEASE
  INTEGER(Long) :: Version = SSU_INPUT_VERSION
  ! Time in decimal year (e.g. 2009.08892694 corresponds to 11:00 Feb. 2, 2009)
  REAL(Double) :: Time = ZERO
  ! SSU CO2 cell pressures (hPa)
  REAL(Double) :: Cell_Pressure(MAX_N_CHANNELS) = ZERO
END TYPE SSU_Input_type
```

Figure A.10: SSU_Input_type structure definition.

A.10.1 SSU_Input_CellPressureIsSet interface

```
NAME:
      SSU_Input_CellPressureIsSet

PURPOSE:
      Elemental function to determine if SSU_Input object cell pressures
      are set (i.e. > zero).

CALLING SEQUENCE:
      result = SSU_Input_CellPressureIsSet( ssu )

        or

      IF ( SSU_Input_CellPressureIsSet( ssu ) ) THEN
        ...
      END IF

OBJECTS:
      ssu:       SSU_Input object for which the cell pressures
                 are to be tested.
                 UNITS:      N/A
                 TYPE:       SSU_Input_type
                 DIMENSION:  Scalar or any rank
                 ATTRIBUTES: INTENT(IN)

FUNCTION RESULT:
      result:    Logical variable indicating whether or not all the
                 SSU cell pressures are set.
                 If == .FALSE., cell pressure values are <= 0.0hPa and
                                thus are considered to be NOT set or valid.
                    == .TRUE.,  cell pressure values are > 0.0hPa and
                                thus are considered to be set and valid.
                 UNITS:      N/A
                 TYPE:       LOGICAL
                 DIMENSION:  Scalar
```

A.10.2 SSU_Input_DefineVersion interface

```
NAME:
      SSU_Input_DefineVersion

PURPOSE:
      Subroutine to return the module version information.

CALLING SEQUENCE:
      CALL SSU_Input_DefineVersion( Id )

OUTPUTS:
      Id:            Character string containing the version Id information
                     for the module.
                     UNITS:      N/A
                     TYPE:       CHARACTER(*)
                     DIMENSION:  Scalar
                     ATTRIBUTES: INTENT(OUT)
```

A.10.3 SSU_Input_GetValue interface

```
NAME:
      SSU_Input_GetValue

PURPOSE:
      Elemental subroutine to Get the values of SSU_Input
      object components.

CALLING SEQUENCE:
      CALL SSU_Input_GetValue( SSU_Input                    , &
                               Channel       = Channel      , &
                               Time          = Time         , &
                               Cell_Pressure = Cell_Pressure, &
                               n_Channels    = n_Channels     )

OBJECTS:
      SSU_Input:            SSU_Input object for which component values
                            are to be set.
                            UNITS:      N/A
                            TYPE:       SSU_Input_type
                            DIMENSION:  Scalar or any rank
                            ATTRIBUTES: INTENT(IN OUT)

OPTIONAL INPUTS:
      Channel:              SSU channel for which the CO2 cell pressure
                            is required.
                            UNITS:      N/A
                            TYPE:       INTEGER
                            DIMENSION:  Scalar or same as SSU_Input
                            ATTRIBUTES: INTENT(IN), OPTIONAL

OPTIONAL OUTPUTS:
      Time:                 SSU instrument mission time.
                            UNITS:      decimal year
                            TYPE:       REAL(fp)
                            DIMENSION:  Scalar or same as SSU_Input
                            ATTRIBUTES: INTENT(OUT), OPTIONAL

      Cell_Pressure:        SSU channel CO2 cell pressure. Must be
                            specified with the Channel optional input
                            dummy argument.
                            UNITS:      hPa
```

```
                              TYPE:       REAL(fp)
                              DIMENSION:  Scalar or same as SSU_Input
                              ATTRIBUTES: INTENT(OUT), OPTIONAL

      n_Channels:             Number of SSU channels..
                              UNITS:      N/A
                              TYPE:       INTEGER
                              DIMENSION:  Scalar or same as SSU_Input
                              ATTRIBUTES: INTENT(OUT), OPTIONAL
```

A.10.4 SSU_Input_Inspect interface

```
NAME:
      SSU_Input_Inspect

PURPOSE:
      Subroutine to print the contents of an SSU_Input object to stdout.

CALLING SEQUENCE:
      CALL SSU_Input_Inspect( ssu )

INPUTS:
      ssu:           SSU_Input object to display.
                     UNITS:      N/A
                     TYPE:       SSU_Input_type
                     DIMENSION:  Scalar
                     ATTRIBUTES: INTENT(IN)
```

A.10.5 SSU_Input_IsValid interface

```
NAME:
      SSU_Input_IsValid

PURPOSE:
      Non-pure function to perform some simple validity checks on a
      SSU_Input object.

      If invalid data is found, a message is printed to stdout.

CALLING SEQUENCE:
      result = SSU_Input_IsValid( ssu )

        or

      IF ( SSU_Input_IsValid( ssu ) ) THEN....
```

```
OBJECTS:
      ssu:       SSU_Input object which is to have its
                 contents checked.
                 UNITS:      N/A
                 TYPE:       SSU_Input_type
                 DIMENSION:  Scalar
                 ATTRIBUTES: INTENT(IN)

FUNCTION RESULT:
      result:    Logical variable indicating whether or not the input
                 passed the check.
                 If == .FALSE., object is unused or contains
                                invalid data.
                    == .TRUE.,  object can be used.
                 UNITS:      N/A
                 TYPE:       LOGICAL
                 DIMENSION:  Scalar
```

A.10.6 SSU_Input_ReadFile interface

```
NAME:
      SSU_Input_ReadFile

PURPOSE:
      Function to read SSU_Input object files.

CALLING SEQUENCE:
      Error_Status = SSU_Input_ReadFile( &
                       SSU_Input          , &
                       Filename           , &
                       No_Close = No_Close, &
                       Quiet    = Quiet     )

OBJECTS:
      SSU_Input:      SSU_Input object containing the data read from file.
                      UNITS:      N/A
                      TYPE:       SSU_Input_type
                      DIMENSION:  Scalar
                      ATTRIBUTES: INTENT(OUT)

INPUTS:
      Filename:       Character string specifying the name of a
                      SSU_Input data file to read.
                      UNITS:      N/A
                      TYPE:       CHARACTER(*)
                      DIMENSION:  Scalar
                      ATTRIBUTES: INTENT(IN)

OPTIONAL INPUTS:
      No_Close:       Set this logical argument to *NOT* close the datafile
```

```
                    upon exiting this routine. This option is required if
                    the SSU_Input data is embedded within another file.
                    If == .FALSE., File is closed upon function exit [DEFAULT].
                       == .TRUE.,  File is NOT closed upon function exit
                    If not specified, default is .FALSE.
                    UNITS:      N/A
                    TYPE:       LOGICAL
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN), OPTIONAL

      Quiet:        Set this logical argument to suppress INFORMATION
                    messages being printed to stdout
                    If == .FALSE., INFORMATION messages are OUTPUT [DEFAULT].
                       == .TRUE.,  INFORMATION messages are SUPPRESSED.
                    If not specified, default is .FALSE.
                    UNITS:      N/A
                    TYPE:       LOGICAL
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN), OPTIONAL

FUNCTION RESULT:
      Error_Status: The return value is an integer defining the error status.
                    The error codes are defined in the Message_Handler module.
                    If == SUCCESS, the file read was successful
                       == FAILURE, an unrecoverable error occurred.
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Scalar
```

A.10.7 SSU_Input_SetValue interface

```
NAME:
      SSU_Input_SetValue

PURPOSE:
      Elemental subroutine to set the values of SSU_Input
      object components.

CALLING SEQUENCE:
      CALL SSU_Input_SetValue( SSU_Input                    , &
                               Time          = Time         , &
                               Cell_Pressure = Cell_Pressure, &
                               Channel       = Channel        )

OBJECTS:
      SSU_Input:            SSU_Input object for which component values
                            are to be set.
                            UNITS:      N/A
                            TYPE:       SSU_Input_type
                            DIMENSION:  Scalar or any rank
                            ATTRIBUTES: INTENT(IN OUT)

OPTIONAL INPUTS:
      Time:                 SSU instrument mission time.
                            UNITS:      decimal year
                            TYPE:       REAL(fp)
                            DIMENSION:  Scalar or same as SSU_Input
                            ATTRIBUTES: INTENT(IN), OPTIONAL

      Cell_Pressure:        SSU channel CO2 cell pressure. Must be
                            specified with the Channel optional dummy
                            argument.
                            UNITS:      hPa
                            TYPE:       REAL(fp)
                            DIMENSION:  Scalar or same as SSU_Input
                            ATTRIBUTES: INTENT(IN), OPTIONAL

      Channel:              SSU channel for which the CO2 cell pressure
                            is to be set. Must be specified with the
                            Cell_Pressure optional dummy argument.
                            UNITS:      N/A
                            TYPE:       INTEGER
                            DIMENSION:  Scalar or same as SSU_Input
                            ATTRIBUTES: INTENT(IN), OPTIONAL
```

A.10.8 SSU_Input_ValidRelease interface

```
NAME:
      SSU_Input_ValidRelease

PURPOSE:
      Function to check the SSU_Input Release value.

CALLING SEQUENCE:
      IsValid = SSU_Input_ValidRelease( SSU_Input )

INPUTS:
      SSU_Input:    SSU_Input object for which the Release component
                     is to be checked.
                     UNITS:      N/A
                     TYPE:       SSU_Input_type
                     DIMENSION:  Scalar
                     ATTRIBUTES: INTENT(IN)

FUNCTION RESULT:
      IsValid:       Logical value defining the release validity.
                     UNITS:      N/A
                     TYPE:       LOGICAL
                     DIMENSION:  Scalar
```

A.10.9 SSU_Input_WriteFile interface

```
NAME:
      SSU_Input_WriteFile

PURPOSE:
      Function to write SSU_Input object files.

CALLING SEQUENCE:
      Error_Status = SSU_Input_WriteFile( &
                       SSU_Input          , &
                       Filename           , &
                       No_Close = No_Close, &
                       Quiet    = Quiet     )

OBJECTS:
      SSU_Input:      SSU_Input object containing the data to write to file.
                      UNITS:      N/A
                      TYPE:       SSU_Input_type
                      DIMENSION:  Scalar
                      ATTRIBUTES: INTENT(IN)

INPUTS:
      Filename:       Character string specifying the name of a
                      SSU_Input format data file to write.
                      UNITS:      N/A
                      TYPE:       CHARACTER(*)
                      DIMENSION:  Scalar
                      ATTRIBUTES: INTENT(IN)

OPTIONAL INPUTS:
      No_Close:       Set this logical argument to *NOT* close the datafile
                      upon exiting this routine. This option is required if
                      the SSU_Input data is to be embedded within another file.
                      If == .FALSE., File is closed upon function exit [DEFAULT].
                         == .TRUE.,  File is NOT closed upon function exit
                      If not specified, default is .FALSE.
                      UNITS:      N/A
                      TYPE:       LOGICAL
                      DIMENSION:  Scalar
                      ATTRIBUTES: INTENT(IN), OPTIONAL

      Quiet:          Set this logical argument to suppress INFORMATION
                      messages being printed to stdout
                      If == .FALSE., INFORMATION messages are OUTPUT [DEFAULT].
                         == .TRUE.,  INFORMATION messages are SUPPRESSED.
                      If not specified, default is .FALSE.
                      UNITS:      N/A
                      TYPE:       LOGICAL
                      DIMENSION:  Scalar
```

```
                     ATTRIBUTES: INTENT(IN), OPTIONAL

FUNCTION RESULT:
      Error_Status:  The return value is an integer defining the error status.
                     The error codes are defined in the Message_Handler module.
                     If == SUCCESS, the file write was successful
                        == FAILURE, an unrecoverable error occurred.
                     UNITS:      N/A
                     TYPE:       INTEGER
                     DIMENSION:  Scalar
```

A.11 Zeeman_Input Structure

The Zeeman_Input structure is a component of the Options input structure. Note in figure A.11 that the structure is declared as PRIVATE. As such, the only way to set values in, or get values from, the structure is via the Zeeman_Input_SetValue or Zeeman_Input_GetValue subroutines respectively.

```
TYPE :: Zeeman_Input_type
  PRIVATE
  ! Release and version information
  INTEGER(Long) :: Release = ZEEMAN_INPUT_RELEASE
  INTEGER(Long) :: Version = ZEEMAN_INPUT_VERSION
  ! Earth magnetic field strength in Gauss
  REAL(Double) :: Be = DEFAULT_MAGENTIC_FIELD
  ! Cosine of the angle between the Earth
  ! magnetic field and wave propagation direction
  REAL(Double) :: Cos_ThetaB = ZERO
  ! Cosine of the azimuth angle of the Be vector.
  REAL(Double) :: Cos_PhiB = ZERO
  ! Doppler frequency shift caused by Earth-rotation.
  REAL(Double) :: Doppler_Shift = ZERO
END TYPE Zeeman_Input_type
```

Figure A.11: Zeeman_Input_type structure definition.

A.11.1 Zeeman_Input_DefineVersion interface

```
NAME:
      Zeeman_Input_DefineVersion

PURPOSE:
      Subroutine to return the module version information.

CALLING SEQUENCE:
      CALL Zeeman_Input_DefineVersion( Id )

OUTPUTS:
      Id:            Character string containing the version Id information
                     for the module.
                     UNITS:      N/A
                     TYPE:       CHARACTER(*)
                     DIMENSION:  Scalar
                     ATTRIBUTES: INTENT(OUT)
```

A.11.2 Zeeman_Input_GetValue interface

```
NAME:
      Zeeman_Input_GetValue

PURPOSE:
      Elemental subroutine to get the values of Zeeman_Input
      object components.

CALLING SEQUENCE:
      CALL Zeeman_Input_GetValue( Zeeman_Input                   , &
                                  Field_Strength = Field_Strength, &
                                  Cos_ThetaB     = Cos_ThetaB    , &
                                  Cos_PhiB       = Cos_PhiB      , &
                                  Doppler_Shift  = Doppler_Shift   )

OBJECTS:
      Zeeman_Input:         Zeeman_Input object for which component values
                            are to be set.
                            UNITS:      N/A
                            TYPE:       Zeeman_Input_type
                            DIMENSION:  Scalar or any rank
                            ATTRIBUTES: INTENT(IN OUT)

OPTIONAL OUTPUTS:
      Field_Strength:       Earth's magnetic filed strength
                            UNITS:      Gauss
                            TYPE:       REAL(fp)
                            DIMENSION:  Scalar or same as Zeeman_Input
                            ATTRIBUTES: INTENT(OUT), OPTIONAL
      Cos_ThetaB:           Cosine of the angle between the Earth magnetic
                            field and wave propagation vectors.
                            UNITS:      N/A
                            TYPE:       REAL(fp)
                            DIMENSION:  Scalar or same as Zeeman_Input
                            ATTRIBUTES: INTENT(OUT), OPTIONAL

      Cos_PhiB:             Cosine of the azimuth angle of the Earth magnetic
                            field vector.
                            UNITS:      N/A
                            TYPE:       REAL(fp)
                            DIMENSION:  Scalar or same as Zeeman_Input
                            ATTRIBUTES: INTENT(OUT), OPTIONAL

      Doppler_Shift:        Doppler frequency shift caused by Earth-rotation.
                            Positive towards sensor.
                            UNITS:      KHz
                            TYPE:       REAL(fp)
                            DIMENSION:  Scalar or same as Zeeman_Input
                            ATTRIBUTES: INTENT(OUT), OPTIONAL
```

A.11.3 Zeeman_Input_Inspect interface

```
NAME:
      Zeeman_Input_Inspect

PURPOSE:
      Subroutine to print the contents of an Zeeman_Input object to stdout.

CALLING SEQUENCE:
      CALL Zeeman_Input_Inspect( z )

INPUTS:
      z:             Zeeman_Input object to display.
                     UNITS:      N/A
                     TYPE:       Zeeman_Input_type
                     DIMENSION:  Scalar
                     ATTRIBUTES: INTENT(IN)
```

A.11.4 Zeeman_Input_IsValid interface

```
NAME:
      Zeeman_Input_IsValid

PURPOSE:
      Non-pure function to perform some simple validity checks on a
      Zeeman_Input object.

      If invalid data is found, a message is printed to stdout.

CALLING SEQUENCE:
      result = Zeeman_Input_IsValid( z )

        or

      IF ( Zeeman_Input_IsValid( z ) ) THEN....

OBJECTS:
      z:         Zeeman_Input object which is to have its
                 contents checked.
                 UNITS:      N/A
                 TYPE:       Zeeman_Input_type
                 DIMENSION:  Scalar
                 ATTRIBUTES: INTENT(IN)

FUNCTION RESULT:
      result:    Logical variable indicating whether or not the input
                 passed the check.
                 If == .FALSE., object is unused or contains
                                invalid data.
                    == .TRUE.,  object can be used.
                 UNITS:      N/A
                 TYPE:       LOGICAL
                 DIMENSION:  Scalar
```

A.11.5 Zeeman_Input_ReadFile interface

```
NAME:
      Zeeman_Input_ReadFile

PURPOSE:
      Function to read Zeeman_Input object files.

CALLING SEQUENCE:
      Error_Status = Zeeman_Input_ReadFile( &
                       Zeeman_Input      , &
                       Filename          , &
                       No_Close = No_Close, &
                       Quiet    = Quiet     )

OBJECTS:
      Zeeman_Input:   Zeeman_Input object containing the data read from file.
                      UNITS:      N/A
                      TYPE:       Zeeman_Input_type
                      DIMENSION:  Scalar
```

```
                    ATTRIBUTES: INTENT(OUT)

INPUTS:
      Filename:       Character string specifying the name of a
                      Zeeman_Input data file to read.
                      UNITS:      N/A
                      TYPE:       CHARACTER(*)
                      DIMENSION:  Scalar
                      ATTRIBUTES: INTENT(IN)

OPTIONAL INPUTS:
      No_Close:       Set this logical argument to *NOT* close the datafile
                      upon exiting this routine. This option is required if
                      the Zeeman_Input data is embedded within another file.
                      If == .FALSE., File is closed upon function exit [DEFAULT].
                         == .TRUE.,  File is NOT closed upon function exit
                      If not specified, default is .FALSE.
                      UNITS:      N/A
                      TYPE:       LOGICAL
                      DIMENSION:  Scalar
                      ATTRIBUTES: INTENT(IN), OPTIONAL

      Quiet:          Set this logical argument to suppress INFORMATION
                      messages being printed to stdout
                      If == .FALSE., INFORMATION messages are OUTPUT [DEFAULT].
                         == .TRUE.,  INFORMATION messages are SUPPRESSED.
                      If not specified, default is .FALSE.
                      UNITS:      N/A
                      TYPE:       LOGICAL
                      DIMENSION:  Scalar
                      ATTRIBUTES: INTENT(IN), OPTIONAL

FUNCTION RESULT:
      Error_Status:   The return value is an integer defining the error status.
                      The error codes are defined in the Message_Handler module.
                      If == SUCCESS, the file read was successful
                         == FAILURE, an unrecoverable error occurred.
                      UNITS:      N/A
                      TYPE:       INTEGER
                      DIMENSION:  Scalar
```

A.11.6 Zeeman_Input_SetValue interface

```
NAME:
      Zeeman_Input_SetValue

PURPOSE:
      Elemental subroutine to set the values of Zeeman_Input
      object components.
```

```
CALLING SEQUENCE:
      CALL Zeeman_Input_SetValue( Zeeman_Input                   , &
                                  Field_Strength = Field_Strength, &
                                  Cos_ThetaB     = Cos_ThetaB    , &
                                  Cos_PhiB       = Cos_PhiB      , &
                                  Doppler_Shift  = Doppler_Shift   )

OBJECTS:
      Zeeman_Input:         Zeeman_Input object for which component values
                            are to be set.
                            UNITS:      N/A
                            TYPE:       Zeeman_Input_type
                            DIMENSION:  Scalar or any rank
                            ATTRIBUTES: INTENT(IN OUT)

OPTIONAL INPUTS:
      Field_Strength:       Earth's magnetic filed strength
                            UNITS:      Gauss
                            TYPE:       REAL(fp)
                            DIMENSION:  Scalar or same as Zeeman_Input
                            ATTRIBUTES: INTENT(IN), OPTIONAL

      Cos_ThetaB:           Cosine of the angle between the Earth magnetic
                            field and wave propagation vectors.
                            UNITS:      N/A
                            TYPE:       REAL(fp)
                            DIMENSION:  Scalar or same as Zeeman_Input
                            ATTRIBUTES: INTENT(IN), OPTIONAL

      Cos_PhiB:             Cosine of the azimuth angle of the Earth magnetic
                            field vector.
                            UNITS:      N/A
                            TYPE:       REAL(fp)
                            DIMENSION:  Scalar or same as Zeeman_Input
                            ATTRIBUTES: INTENT(IN), OPTIONAL

      Doppler_Shift:        Doppler frequency shift caused by Earth-rotation.
                            Positive towards sensor.
                            UNITS:      KHz
                            TYPE:       REAL(fp)
                            DIMENSION:  Scalar or same as Zeeman_Input
                            ATTRIBUTES: INTENT(IN), OPTIONAL
```

A.11.7 Zeeman_Input_ValidRelease interface

```
NAME:
      Zeeman_Input_ValidRelease

PURPOSE:
      Function to check the Zeeman_Input Release value.
```

```
CALLING SEQUENCE:
      IsValid = Zeeman_Input_ValidRelease( Zeeman_Input )

INPUTS:
      Zeeman_Input:  Zeeman_Input object for which the Release component
                     is to be checked.
                     UNITS:      N/A
                     TYPE:       Zeeman_Input_type
                     DIMENSION:  Scalar
                     ATTRIBUTES: INTENT(IN)

FUNCTION RESULT:
      IsValid:       Logical value defining the release validity.
                     UNITS:      N/A
                     TYPE:       LOGICAL
                     DIMENSION:  Scalar
```

A.11.8 Zeeman_Input_WriteFile interface

```
NAME:
      Zeeman_Input_WriteFile

PURPOSE:
      Function to write Zeeman_Input object files.

CALLING SEQUENCE:
      Error_Status = Zeeman_Input_WriteFile( &
                       Zeeman_Input       , &
                       Filename           , &
                       No_Close = No_Close, &
                       Quiet    = Quiet     )

OBJECTS:
      Zeeman_Input:   Zeeman_Input object containing the data to write to file.
                      UNITS:      N/A
                      TYPE:       Zeeman_Input_type
                      DIMENSION:  Scalar
                      ATTRIBUTES: INTENT(IN)

INPUTS:
      Filename:       Character string specifying the name of a
                      Zeeman_Input format data file to write.
                      UNITS:      N/A
                      TYPE:       CHARACTER(*)
                      DIMENSION:  Scalar
                      ATTRIBUTES: INTENT(IN)

OPTIONAL INPUTS:
      No_Close:       Set this logical argument to *NOT* close the datafile
```

```
                    upon exiting this routine. This option is required if
                    the Zeeman_Input data is to be embedded within another file.
                    If == .FALSE., File is closed upon function exit [DEFAULT].
                       == .TRUE.,  File is NOT closed upon function exit
                    If not specified, default is .FALSE.
                    UNITS:      N/A
                    TYPE:       LOGICAL
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN), OPTIONAL

      Quiet:        Set this logical argument to suppress INFORMATION
                    messages being printed to stdout
                    If == .FALSE., INFORMATION messages are OUTPUT [DEFAULT].
                       == .TRUE.,  INFORMATION messages are SUPPRESSED.
                    If not specified, default is .FALSE.
                    UNITS:      N/A
                    TYPE:       LOGICAL
                    DIMENSION:  Scalar
                    ATTRIBUTES: INTENT(IN), OPTIONAL

FUNCTION RESULT:
      Error_Status:   The return value is an integer defining the error status.
                    The error codes are defined in the Message_Handler module.
                    If == SUCCESS, the file write was successful
                       == FAILURE, an unrecoverable error occurred.
                    UNITS:      N/A
                    TYPE:       INTEGER
                    DIMENSION:  Scalar
```

B
Valid Sensor Identifiers

This section contains a table detailing the instruments for which there are CRTM coefficients. For most sensors there are transmittance coefficient (`TauCoeff`) datafiles for both the Optical Depth in Absorber Space (ODAS; also known as Compact-OPTRAN) and Optical Depth in Pressure Space (ODPS) transmittance algorithms. All visible and SSU channels have only ODAS coefficients.

Table B.1: CRTM sensor identifiers and the availability of ODAS or ODPS TauCoeff files

Instrument	Sensor Id	ODAS available	ODPS available
Envisat AATSR	`aatsr_envisat`	yes	yes
GOES-R ABI	`abi_gr`	yes	yes
Aqua AIRS (281ch. subset)	`airs281_aqua`	yes	yes
Aqua AIRS (324ch. subset)	`airs324_aqua`	yes	yes
Aqua AIRS (all channels)	`airs2378_aqua`	yes	yes
Aqua AIRS Module-1a	`airsM1a_aqua`	yes	yes
Aqua AIRS Module-1b	`airsM1b_aqua`	yes	yes
Aqua AIRS Module-2a	`airsM2a_aqua`	yes	yes
Aqua AIRS Module-2b	`airsM2b_aqua`	yes	yes
Aqua AIRS Module-3	`airsM3_aqua`	yes	yes
Aqua AIRS Module-4a	`airsM4a_aqua`	yes	yes
Aqua AIRS Module-4b	`airsM4b_aqua`	yes	yes
Aqua AIRS Module-4c	`airsM4c_aqua`	yes	yes
Aqua AIRS Module-4d	`airsM4d_aqua`	yes	yes
Aqua AIRS Module-5	`airsM5_aqua`	yes	yes
Aqua AIRS Module-6	`airsM6_aqua`	yes	yes
Aqua AIRS Module-7	`airsM7_aqua`	yes	yes
Aqua AIRS Module-8	`airsM8_aqua`	yes	yes
Aqua AIRS Module-9	`airsM9_aqua`	yes	yes
Aqua AIRS Module-10	`airsM10_aqua`	yes	yes
Aqua AIRS Module-11	`airsM11_aqua`	yes	yes
Aqua AIRS Module-12	`airsM12_aqua`	yes	yes
Aqua AMSR-E	`amsre_aqua`	yes	yes
GCOM-W1 AMSR-2	`amsr2_gcom-w1`	yes	yes
Aqua AMSU-A	`amsua_aqua`	yes	yes
NOAA-15 AMSU-A	`amsua_n15`	yes	yes
NOAA-16 AMSU-A	`amsua_n16`	yes	yes
NOAA-17 AMSU-A	`amsua_n17`	yes	yes
NOAA-18 AMSU-A	`amsua_n18`	yes	yes
NOAA-19 AMSU-A	`amsua_n19`	yes	yes
MetOp-A AMSU-A	`amsua_metop-a`	yes	yes
MetOp-B AMSU-A	`amsua_metop-b`	yes	yes
MetOp-C AMSU-A	`amsua_metop-c`	yes	yes
NOAA-15 AMSU-B	`amsub_n15`	yes	yes
NOAA-16 AMSU-B	`amsub_n16`	yes	yes
NOAA-17 AMSU-B	`amsub_n17`	yes	yes
NPP ATMS	`atms_npp`	yes	yes
ERS-1 ATSR	`atsr1_ers1`	yes	yes
ERS-2 ATSR	`atsr2_ers2`	yes	yes
TIROS-N AVHRR/2	`avhrr2_tirosn`	yes	yes
NOAA-06 AVHRR/2	`avhrr2_n06`	yes	yes
NOAA-07 AVHRR/2	`avhrr2_n07`	yes	yes
NOAA-08 AVHRR/2	`avhrr2_n08`	yes	yes
NOAA-09 AVHRR/2	`avhrr2_n09`	yes	yes
NOAA-10 AVHRR/2	`avhrr2_n10`	yes	yes
NOAA-11 AVHRR/2	`avhrr2_n11`	yes	yes
NOAA-12 AVHRR/2	`avhrr2_n12`	yes	yes
NOAA-14 AVHRR/2	`avhrr2_n14`	yes	yes
NOAA-15 AVHRR/3	`avhrr3_n15`	yes	yes

Continued on Next Page...

Table B.1 – Continued

Instrument	Sensor Id	ODAS available	ODPS available
NOAA-16 AVHRR/3	`avhrr3_n16`	yes	yes
NOAA-17 AVHRR/3	`avhrr3_n17`	yes	yes
NOAA-18 AVHRR/3	`avhrr3_n18`	yes	yes
NOAA-19 AVHRR/3	`avhrr3_n19`	yes	yes
MetOp-A AVHRR/3	`avhrr3_metop-a`	yes	yes
MetOp-B AVHRR/3	`avhrr3_metop-b`	yes	yes
NPP CrIS (374ch. subset)	`cris374_npp`	yes	yes
NPP CrIS (399ch. subset)	`cris399_npp`	yes	yes
NPP CrIS (all channels)	`cris1305_npp`	yes	yes
NPP CrIS Band 1	`crisB1_npp`	yes	yes
NPP CrIS Band 2	`crisB2_npp`	yes	yes
NPP CrIS Band 3	`crisB3_npp`	yes	yes
GPM GMI	`gmi_gpm`	yes	yes
TIROS-N HIRS/2	`hirs2_tirosn`	yes	yes
NOAA-06 HIRS/2	`hirs2_n06`	yes	yes
NOAA-07 HIRS/2	`hirs2_n07`	yes	yes
NOAA-08 HIRS/2	`hirs2_n08`	yes	yes
NOAA-09 HIRS/2	`hirs2_n09`	yes	yes
NOAA-10 HIRS/2	`hirs2_n10`	yes	yes
NOAA-11 HIRS/2	`hirs2_n11`	yes	yes
NOAA-12 HIRS/2	`hirs2_n12`	yes	yes
NOAA-14 HIRS/2	`hirs2_n14`	yes	yes
NOAA-15 HIRS/3	`hirs3_n15`	yes	yes
NOAA-16 HIRS/3	`hirs3_n16`	yes	yes
NOAA-17 HIRS/3	`hirs3_n17`	yes	yes
NOAA-18 HIRS/4	`hirs4_n18`	yes	yes
NOAA-19 HIRS/4	`hirs4_n19`	yes	yes
MetOp-A HIRS/4	`hirs4_metop-a`	yes	yes
MetOp-B HIRS/4	`hirs4_metop-b`	yes	yes
Aqua HSB	`hsb_aqua`	yes	yes
MetOp-A IASI (300ch. subset)	`iasi300_metop-a`	yes	yes
MetOp-A IASI (316ch. subset)	`iasi316_metop-a`	yes	yes
MetOp-A IASI (616ch. subset)	`iasi616_metop-a`	yes	yes
MetOp-A IASI (all channels)	`iasi8461_metop-a`	yes	yes
MetOp-A IASI Band 1	`iasiB1_metop-a`	yes	yes
MetOp-A IASI Band 2	`iasiB2_metop-a`	yes	yes
MetOp-A IASI Band 3	`iasiB3_metop-a`	yes	yes
MetOp-B IASI (300ch. subset)	`iasi300_metop-b`	yes	yes
MetOp-B IASI (316ch. subset)	`iasi316_metop-b`	yes	yes
MetOp-B IASI (616ch. subset)	`iasi616_metop-b`	yes	yes
MetOp-B IASI (all channels)	`iasi8461_metop-b`	yes	yes
MetOp-B IASI Band 1	`iasiB1_metop-b`	yes	yes
MetOp-B IASI Band 2	`iasiB2_metop-b`	yes	yes
MetOp-B IASI Band 3	`iasiB3_metop-b`	yes	yes
GOES-08 Imager	`imgr_g08`	yes	yes
GOES-09 Imager	`imgr_g09`	yes	yes
GOES-10 Imager	`imgr_g10`	yes	yes
GOES-11 Imager	`imgr_g11`	yes	yes
GOES-12 Imager	`imgr_g12`	yes	yes
GOES-13 Imager	`imgr_g13`	yes	yes

Continued on Next Page...

Table B.1 – Continued

Instrument	Sensor Id	ODAS available	ODPS available
GOES-14 Imager	`imgr_g14`	yes	yes
GOES-15 Imager	`imgr_g15`	yes	yes
MTSAT-1R Imager	`imgr_mt1r`	yes	yes
MTSAT-2 Imager	`imgr_mt2`	yes	yes
Fengyun-3a IRAS	`iras_fy3a`	yes	yes
Fengyun-3b IRAS	`iras_fy3b`	yes	yes
Megha-Tropiques MADRAS	`madras_meghat`	yes	yes
Fengyun-3a MERSI	`mersi_fy3a`	yes	yes
NOAA-18 MHS	`mhs_n18`	yes	yes
NOAA-19 MHS	`mhs_n19`	yes	yes
MetOp-A MHS	`mhs_metop-a`	yes	yes
MetOp-B MHS	`mhs_metop-b`	yes	yes
MetOp-C MHS	`mhs_metop-c`	yes	yes
COMS-1 MI (low patch)	`mi-l_coms`	yes	yes
COMS-1 MI (medium patch)	`mi-m_coms`	yes	yes
Aqua MODIS	`modis_aqua`	yes	yes
Terra MODIS	`modis_terra`	yes	yes
TIROS-N MSU	`msu_tirosn`	yes	yes
NOAA-06 MSU	`msu_n06`	yes	yes
NOAA-07 MSU	`msu_n07`	yes	yes
NOAA-08 MSU	`msu_n08`	yes	yes
NOAA-09 MSU	`msu_n09`	yes	yes
NOAA-10 MSU	`msu_n10`	yes	yes
NOAA-11 MSU	`msu_n11`	yes	yes
NOAA-12 MSU	`msu_n12`	yes	yes
NOAA-14 MSU	`msu_n14`	yes	yes
Meteosat-3 MVIRI (backup)	`mviriBKUP_m03`	no	yes
Meteosat-4 MVIRI (backup)	`mviriBKUP_m04`	no	yes
Meteosat-5 MVIRI (backup)	`mviriBKUP_m05`	no	yes
Meteosat-6 MVIRI (backup)	`mviriBKUP_m06`	no	yes
Meteosat-7 MVIRI (backup)	`mviriBKUP_m07`	no	yes
Meteosat-3 MVIRI (nominal)	`mviriNOM_m03`	no	yes
Meteosat-4 MVIRI (nominal)	`mviriNOM_m04`	no	yes
Meteosat-5 MVIRI (nominal)	`mviriNOM_m05`	no	yes
Meteosat-6 MVIRI (nominal)	`mviriNOM_m06`	no	yes
Meteosat-7 MVIRI (nominal)	`mviriNOM_m07`	no	yes
Fengyun-3a MWHS	`mwhs_fy3a`	yes	yes
Fengyun-3b MWHS	`mwhs_fy3b`	yes	yes
Fengyun-3a MWRI	`mwri_fy3a`	yes	yes
Fengyun-3b MWRI	`mwri_fy3b`	yes	yes
Fengyun-3a MWTS	`mwts_fy3a`	yes	yes
Fengyun-3b MWTS	`mwts_fy3b`	yes	yes
Megha-Tropiques SAPHIR	`saphir_meghat`	yes	yes
Meteosat-08 SEVIRI	`seviri_m08`	yes	yes
Meteosat-09 SEVIRI	`seviri_m09`	yes	yes
Meteosat-10 SEVIRI	`seviri_m10`	yes	yes
GOES-10 Sounder (Detector 1)	`sndrD1_g10`	yes	yes
GOES-10 Sounder (Detector 2)	`sndrD2_g10`	yes	yes
GOES-10 Sounder (Detector 3)	`sndrD3_g10`	yes	yes
GOES-10 Sounder (Detector 4)	`sndrD4_g10`	yes	yes

Continued on Next Page…

Table B.1 – Continued

Instrument	Sensor Id	ODAS available	ODPS available
GOES-11 Sounder (Detector 1)	sndrD1_g11	yes	yes
GOES-11 Sounder (Detector 2)	sndrD2_g11	yes	yes
GOES-11 Sounder (Detector 3)	sndrD3_g11	yes	yes
GOES-11 Sounder (Detector 4)	sndrD4_g11	yes	yes
GOES-12 Sounder (Detector 1)	sndrD1_g12	yes	yes
GOES-12 Sounder (Detector 2)	sndrD2_g12	yes	yes
GOES-12 Sounder (Detector 3)	sndrD3_g12	yes	yes
GOES-12 Sounder (Detector 4)	sndrD4_g12	yes	yes
GOES-13 Sounder (Detector 1)	sndrD1_g13	yes	yes
GOES-13 Sounder (Detector 2)	sndrD2_g13	yes	yes
GOES-13 Sounder (Detector 3)	sndrD3_g13	yes	yes
GOES-13 Sounder (Detector 4)	sndrD4_g13	yes	yes
GOES-14 Sounder (Detector 1)	sndrD1_g14	yes	yes
GOES-14 Sounder (Detector 2)	sndrD2_g14	yes	yes
GOES-14 Sounder (Detector 3)	sndrD3_g14	yes	yes
GOES-14 Sounder (Detector 4)	sndrD4_g14	yes	yes
GOES-15 Sounder (Detector 1)	sndrD1_g15	yes	yes
GOES-15 Sounder (Detector 2)	sndrD2_g15	yes	yes
GOES-15 Sounder (Detector 3)	sndrD3_g15	yes	yes
GOES-15 Sounder (Detector 4)	sndrD4_g15	yes	yes
GOES-08 Sounder	sndr_g08	yes	yes
GOES-09 Sounder	sndr_g09	yes	yes
GOES-10 Sounder	sndr_g10	yes	yes
GOES-11 Sounder	sndr_g11	yes	yes
GOES-12 Sounder	sndr_g12	yes	yes
GOES-13 Sounder	sndr_g13	yes	yes
GOES-14 Sounder	sndr_g14	yes	yes
GOES-15 Sounder	sndr_g15	yes	yes
DMSP-08 SSM/I	ssmi_f08	yes	yes
DMSP-10 SSM/I	ssmi_f10	yes	yes
DMSP-11 SSM/I	ssmi_f11	yes	yes
DMSP-13 SSM/I	ssmi_f13	yes	yes
DMSP-14 SSM/I	ssmi_f14	yes	yes
DMSP-15 SSM/I	ssmi_f15	yes	yes
DMSP-16 SSMIS	ssmis_f16	yes	yes
DMSP-17 SSMIS	ssmis_f17	yes	yes
DMSP-18 SSMIS	ssmis_f18	yes	yes
DMSP-19 SSMIS	ssmis_f19	yes	yes
DMSP-20 SSMIS	ssmis_f20	yes	yes
DMSP-13 SSM/T-1	ssmt1_f13	yes	yes
DMSP-15 SSM/T-1	ssmt1_f15	yes	yes
DMSP-14 SSM/T-2	ssmt2_f14	yes	yes
DMSP-15 SSM/T-2	ssmt2_f15	yes	yes
TIROS-N SSU	ssu_tirosn	yes	yes
NOAA-06 SSU	ssu_n06	yes	yes
NOAA-07 SSU	ssu_n07	yes	yes
NOAA-08 SSU	ssu_n08	yes	yes
NOAA-09 SSU	ssu_n09	yes	yes
NOAA-11 SSU	ssu_n11	yes	yes
NOAA-14 SSU	ssu_n14	yes	yes

Continued on Next Page…

Table B.1 – Continued

Instrument	Sensor Id	ODAS available	ODPS available
TRMM TMI	`tmi_trmm`	yes	yes
GOES-R ABI (visible)	`v.abi_gr`	yes	no
NOAA-15 AVHRR/3 (visible)	`v.avhrr3_n15`	yes	no
NOAA-16 AVHRR/3 (visible)	`v.avhrr3_n16`	yes	no
NOAA-17 AVHRR/3 (visible)	`v.avhrr3_n17`	yes	no
NOAA-18 AVHRR/3 (visible)	`v.avhrr3_n18`	yes	no
NOAA-19 AVHRR/3 (visible)	`v.avhrr3_n19`	yes	no
MetOp-A AVHRR/3 (visible)	`v.avhrr3_metop-a`	yes	no
MetOp-B AVHRR/3 (visible)	`v.avhrr3_metop-b`	yes	no
GOES-11 Imager (visible)	`v.imgr_g11`	yes	no
GOES-12 Imager (visible)	`v.imgr_g12`	yes	no
GOES-13 Imager (visible)	`v.imgr_g13`	yes	no
GOES-14 Imager (visible)	`v.imgr_g14`	yes	no
GOES-15 Imager (visible)	`v.imgr_g15`	yes	no
MTSAT-2 Imager (visible)	`v.imgr_mt2`	yes	no
Aqua MODIS (visible)	`v.modis_aqua`	yes	no
Terra MODIS (visible)	`v.modis_terra`	yes	no
Meteosat-08 SEVIRI (visible)	`v.seviri_m08`	yes	no
Meteosat-09 SEVIRI (visible)	`v.seviri_m09`	yes	no
Meteosat-10 SEVIRI (visible)	`v.seviri_m10`	yes	no
NPP VIIRS Imager, HiRes (visible)	`v.viirs-i_npp`	yes	no
NPP VIIRS Imager, ModRes (visible)	`v.viirs-m_npp`	yes	no
GOES-4 VAS	`vas_g04`	no	yes
GOES-5 VAS	`vas_g05`	no	yes
GOES-6 VAS	`vas_g06`	no	yes
GOES-7 VAS	`vas_g07`	no	yes
NPP VIIRS Imager, HiRes	`viirs-i_npp`	yes	yes
NPP VIIRS Imager, ModRes	`viirs-m_npp`	yes	yes
Fengyun-3a VIRR	`virr_fy3a`	yes	yes
GMS-5 VISSR (Detector A)	`vissrDetA_gms5`	yes	yes
GMS-5 VISSR (Detector B)	`vissrDetB_gms5`	no	yes
Kalpana-1 VHRR	`vhrr_kalpana1`	yes	yes
ITOS VTPR-S1	`vtprS1_itos`	yes	yes
ITOS VTPR-S2	`vtprS2_itos`	yes	yes
ITOS VTPR-S3	`vtprS3_itos`	yes	yes
ITOS VTPR-S4	`vtprS4_itos`	yes	yes
Coriolis WindSat	`windsat_coriolis`	yes	yes

C Migration Path from REL-2.0.x to REL-2.1

This section details the user code changes that need to be made to migrate from using CRTM v2.0.x to v2.1. It is assumed that you've read chapter 4 and aware of the various other changes to the CRTM that can (will?) cause differences in any before/after result comparisons.

C.1 CRTM Initialisation: Emissivity/Reflectivity model datafile arguments

New, *optional*, arguments have been added to the CRTM initialisation function to allow different data files (referred to as "`EmisCoeff`" files) for the various emissivity/reflectivity models to be loaded during initialisation.

C.1.1 Old v2.0.x Calling Syntax

In the v2.0.x CRTM the only emissivity/reflectivity model data loaded during initialisation was that for the infrared sea surface emissivity model (IRSSEM). The v2.0.x CRTM initialisation function used a generic name, "`EmisCoeff.bin`", as the data file to load. Generally this file was symbolically linked to a specific dataset file (for the Nalli or Wu-Smith model). Alternatively, you could specify the actual file name via the optional `EmisCoeff_File` argument. To load the supplied Nalli emissivity model dataset, the v2.0.x CRTM initialisation called looked like,

```
INTEGER :: err_stat
....
err_stat = CRTM_Init( sensor_id, chinfo, &
                      EmisCoeff_File = 'Nalli.IRwater.EmisCoeff.bin' )
IF ( err_stat /= SUCCESS ) THEN
  handle error...
END IF
```

C.1.2 New v2.1 Calling Syntax

Now, in v2.1, emissivity/reflectivity model datafiles are loaded for each spectral type (infrared, microwave, and visible) as well as each main surface type (land, water, snow, and ice). This was done to get set up for planned future changes and updates to the emissivity and reflectivity models for various spectral regions and surface types. because of the need for separate arguments for the different cases, the use of the generic `EmisCoeff_File` argument to refer to the IRSSEM data is deprecated in favour of the specific `IRwaterCoeff_File` optional argument[1]. The equivalent v2.1 initialisation call is now,

[1]The `EmisCoeff_File` argument is deprecated, but still available. However, it will be removed in a future release.

```
INTEGER :: err_stat
....
err_stat = CRTM_Init( sensor_id, chinfo, &
                      IRwaterCoeff_File = 'Nalli.IRwater.EmisCoeff.bin' )
IF ( err_stat /= SUCCESS ) THEN
  handle error...
END IF
```

Note that the Nalli model is the default so the above call is equivalent to not specifying the `IRwaterCoeff_File` argument at all.

In general you can rely on the default data files loaded. See table 4.1 for a list of available data files where different data are available and you have a choice to specify something other than the default. See the `CRTM_Init()` documentation for a complete list of optional arguments to specify the various `EmisCoeff` datafiles.

C.2 CRTM Surface: Infrared/Visible Land surface type specification

The v2.1 updates to the land surface type specifications, along with examples of how to use them, are described in detail in section 4.6.2. As such, in this section we'll simply mention the changes you need to make to your CRTM calling code to replicate the same functionality.

C.2.1 Old v2.0.x Assignment Syntax

In v2.0.x, when specifying land surface types in the `Surface` structure, a number of parameterised values were made available for assignment. For example, one could do something like,

```
TYPE(CRTM_Surface_type) :: sfc(2)
...
! Assign tundra land surface subtype in v2.0.x CRTM
sfc(1)%Land_Type = TUNDRA
```

where the `TUNDRA` was made available and referenced a particular reflectivity spectrum. This approach is possible only when a single land surface classification scheme is used. In the case of the v2.0.x CRTM that was the NPOESS classification. In v2.1 additional land surface classifications, such as USGS and IGBP, are available so a simple parameter to reference a reflectivity spectrum index becomes more difficult to maintain.

C.2.2 New v2.1 Assignment Syntax

Rather than parameterise all the land surface subtypes for all the available classifications, what you need to do is to refer to the particular table defining the subtypes for the land surface classification scheme you are using and select the `numerical value` for the subtype you want.

So, in v2.1, the equivalent assignment for the above tundra land surface subtype would begin by referrring to the NPOESS classification subtype table, table 4.13, find the tundra entry, and use the associated "classification index" (in this case `10`) in the surface structure assignment,

```
TYPE(CRTM_Surface_type) :: sfc(2)
...
! Assign tundra land surface subtype for NPOESS classification in v2.1 CRTM
sfc(1)%Land_Type = 10
```

C.3 CRTM Surface: Microwave Land surface type specification

The v2.1 updates to the land surface type specifications for use with the microwave land surface emissivity model involve the specification of the soil and vegetation types as well as the leaf area index (LAI). The available soil and vegetation types, along with examples of how to use them, are described in detail in section 4.6.2.

C.3.1 Old v2.0.x Assignment Syntax

In v2.0.x, there was no means to specify the soil type, vegetation type, or LAI as they were not used in the microwave land emissivity algorithm.

C.3.2 New v2.1 Assignment Syntax

New components were added to the `Surface` structure to allow specification of the soil type, vegetation type, and LAI. The structure is initialised to default values so *not* specifying values is equivalent to the following,

```
! Default values for new inputs to microwave land surface emissivity algorithm
sfc(1)%LAI             = 3.5_fp
sfc(1)%Soil_Type       = 1
sfc(1)%Vegetation_Type = 1
```

See tables 4.16 and 4.17 for the valid soil and vegetation types accepted by the CRTM v2.1.

D REL-2.4.0: Known Issues and Troubleshooting

D.1 Known Issues

General users:

1. Any 'Transmitance Coefficient' generation codes included in `src/` are not functional. Contact CRTM support above for details.
2. No testing was done on PGI, XLF, or other less popular compilers.
3. Compiler setup files do not contain 'generic' ways to point to netCDF libraries - you need to edit those files and ensure that the paths point to the correct place. This is the netCDF life. Note: Building inside of a JEDI environment (e.g., singularity container) using ecbuild makes this part much easier.

JEDI users:

1. Any 'Transmitance Coefficient' generation codes included in `src/` are not functional. Contact CRTM support above for details.
2. Testing was only done on modern gfortran compilers.

D.2 Troubleshooting

```
==========================================
Installing /crtm based scripts...
  crtm_install_scripts.sh(INFORMATION): CRTM root directory is crtm
  crtm_install_scripts.sh(INFORMATION): /bin exists...
  crtm_install_scripts.sh(INFORMATION): Your $PATH does NOT contain /bin...
  crtm_install_scripts.sh(INFORMATION): Creating a crtmrc file with $PATH modification. For a permanent c
==========================================
```

This uncommon error message relates to the fact that you do not have a `$HOME` environment variable set. You'll also need a `$HOME/bin` directory. Typically something like:

```
export HOME="/home/users/username/
```

or

```
export HOME="~"
```

may work as well. However, usually `$HOME` is set automatically by your system. If you're having this problem, you're likely to have even more problems later – contact your Sysadmin first.

```
=========================================
checking whether the Fortran compiler works... no
configure: error: in '/src/Build':
configure: error: Fortran compiler cannot create executables
=========================================
```

Bash users, type `export | grep "FC"`, it should be set to the name of a compiler, e.g., `declare -x FC="ifort"`. next simply type `ifort` (or whatever FC is trying to use) at the command prompt to see if it's accessible. If it says `command not found`, then you're missing the path to your compiler. This could be a `module` command that needs to be run, or a valid compiler needs to be installed. This varies based on operating system.

```
=========================================
When issuing the 'make' command in src/ :

File Type_Kinds.f90 not found in CRTM_Module.F90 hierarchy.
File File_Utility.f90 not found in CRTM_Module.F90 hierarchy.
<...> dozens of similar lines <...>
File FitCoeff_WriteFile.inc not found in CRTM_Module.F90 hierarchy.
File FitCoeff_Equal.inc not found in CRTM_Module.F90 hierarchy.

Returning to directory <directory>/crtm/src
=========================================
```

You forgot to `. ./Set_CRTM_Environment.sh` in the `crtm/` directory, paying close attention to that leading `.`, then `cd src/` and `make`.