Joint Center for Satellite Data Assimilation

CRTM: v2.0 User Guide

September 23, 2011; rev15674

Change History

Date	Author	Change
2009-01-30	P.van Delst	Initial release.
2009-02-03	P.van Delst	Updated chapter 2 with descriptions of the example code and coefficient tarballs. Added explanation of layering convention in chapter 4.
2010-03-12	P.van Delst	Updated for v2.0.
2010-05-18	P.van Delst	Updated for v2.0.1.
2010-06-01	P.van Delst	Updated for v2.0.2.
2011-09-23	P.van Delst	Updated for v2.0.4.

Contents

List of Figures

List of Tables

What's New in v2.0

New Science

Multiple transmittance algorithms There are now two transmittance models available for use in the CRTM: ODAS (Optical Depth in Absorber Space), which is equivalent to the previous CompactOPTRAN algorithm; and ODPS (Optical Depth in Pressure Space) which is similar to the RTTOV-type of transmittance algorithm, except here OPTRAN is used for water vapor line absorption.

The algorithm is selectable by the user via the transmittance coefficient (`TauCoeff`) data file used to initialise the CRTM. This method, rather than a switch argument in the `CRTM_Init()` function, was chosen to allow users to "mix-and-match" transmittance algorithms for different sensors in the same initialisation call.

SSU-specific transmittance model Similar to the multiple transmittance algorithm approach, a separate algorithm *just* for the SSU instrument has been constructed. The algorithm is based on the ODAS approach, but with elements to account for the time-dependence of the SSU CO_2 cell pressures.

Zeeman-splitting transmittance model for SSMIS upper-level channels A separate algorithm is available to account for the change in absorption at very low pressures due to the Zeeman-splitting of absorption lines. Currently this algorithm has only been applied to the affected channels in the SSMIS instrument, 19-22.

Visible sensor capability The CRTM now supports radiative transfer for visible instruments/channels. The treatment of visible channels was handled in the CRTM framework by considering them separate instruments. The sensor identifier for these instruments/channels are differentiated from their infrared counterparts by a "`v.`" prefix. For example, while `modis_aqua` is the sensor identifier for the infrared channels, `v.modis_aqua` identifies the visible channels.

Inclusion of Matrix Operator Method (MOM) in radiaitve transfer To handle visible wavelength radiative transfer in the prescence of aerosols, the Advanced Doubling-Adding (ADA) algorithm was adapted to use the MOM technique [Liu and Ruprecht, 1996].

Inclusion of additional infrared sea surface emissivity model Files containing the emissivity data (`EmisCoeff`) for the Nalli et al. [2008a] model are provided. Previously, only the `EmisCoeff` files for the Wu and Smith [1997] model were provided. Users can now select between the Nalli et al. [2008a] or Wu and Smith [1997] models by specifying the requisite filename in the call to `CRTM_Init()`.

Surface BRDF for solar-affected shortwave IR channels A bi-directional reflectance distribution function (BRDF) has been added to account for reflected solar in affected shortware infrared channels [Breon, 1993].

Reflectivity for downwelling infrared over water The reflectivity for downwelling infrared radiation over water surface has been changed from Lambertian to specular.

Aerosol type changes To account for changes in the handling of GOCART [Chin et al., 2002] aerosol model output, additional sea salt coarse modes were added to the list of allowed aerosol types. Also, the separate dry and wet types for organic and black carbon aerosols were combined, with a relatvie humidity of 0% used to indicate the previous "dry" aerosol type. See table A.5 for the new list of accepted aerosol types.

Interface Changes

CRTM Initialisation function The changes to the `CRTM_Init()` interface were relatively minor but do require calling codes to be modified:

- The `Sensor_Id` argument is now mandatory. This argument is used to construct the sensor-specific `SpcCoeff` and `TauCoeff` filename and in the past was optional to allow for "generic" filenames. This is no longer allowed and generic `SpcCoeff` and `TauCoeff` files are no longer used.
- The loading of the `CloudCoeff` and `AerosolCoeff` datafiles containing the optical properties of cloud and aerosol particulates is no longer mandatory. For cloud-free CRTM runs, the load of the `CloudCoeff` and `AerosolCoeff` datafiles can be disabled via the optional `Load_CloudCoeff` and `Load_AerosolCoeff` arguments which are logical switches (true or false).

User accessible structures The structures are defined as those that are used in the argument lists of the main CRTM functions (e.g. initialisation; the forward, tangent-linear, adjoint, and K-matrix models; and destruction). Changes were made to both the structure definitions and their procedures. To mitigate the possibility of memory leaks, the definitions of array members of structures have had their `POINTER` attribute replaced with `ALLOCATABLE`. This was a first step in preparation for use of Fortran2003 Object Oriented features in the CRTM (once Fortran2003 compiler become widely available), where the derived type structure definitions will be reclassified as objects and their procedures will be type-bound. To delineate this change from previous versions of CRTM the interfaces of the derived type procedures have been altered by:

- changing the procedure names to use the convention `CRTM_`*object*_*action* where an *object* can be any of the user accessible CRTM derived types (e.g. `CRTM_Atmosphere_type`, `CRTM_RTSolution_type` etc), and the *action* can be those defined operations for the structure (e.g. `Create`, `Destroy`, `Inspect`, etc).
- making the first dummy argument of the definition module procedures the derived type itself. This will eventually allow the procedures to be called via an instance of the derived type[1][2]

All of the current derived type definitions and their associated procedures and interfaces are shown in appendix A.

`GeometryInfo` **to** `Geometry` **structure name change** Previously, the `GeometryInfo` structure held both the user input to the CRTM as well as the internally computed geometry data. To separate these two sets of quantities, the name of the geometry information structure that is passed into the CRTM functions was changed from `CRTM_GeometryInfo_type` to `CRTM_Geometry_type`. This means that *all* of the user input structures are now strictly `INTENT(IN)` arguments.

`Options` **structure specific changes** The additional changes made to the `CRTM_Options_type` definition:

- all usage on/off switches have been changed from integers (0/1) to logicals (true/false),
- a logical switch to control input checking, `Check_Input`, has been added.
- structure components for SSU-specific and Zeeman model input have been added.

To migrate from the CRTM v1.2.x calling structures to those implemented in v2.0.x, see Appendix C, "Migration Path from REL-1.2 to REL-2.0."

[1] Interested readers can investigate the `PASS` attribute that can be used in the `PROCEDURE` statement within derived type definitions in Fortran2003.

[2] The I/O functions do not yet follow this convention, since they are considered secondary to the definition module procedures used to manipulate the derived types.

What's New in v2.0.1

The v2.0.1 update to the CRTM was done to

- Fix defects of varying severity
- Refactor some modules to work around compiler bugs
- Reorganise the testing/example program.

Bug Fixes

Replacing `CRTM_Atmosphere_IsValid` `WARNING` message for missing ozone with `FAILURE` The CRTM contains two different transmittance model algorithm: the Optical Depth in Absorber Space (ODAS) algorithm and the Optical Depth in Pressure Space (ODPS) algorithm. The ODPS algorithm was constructed to handle "missing" profiles of major trace gas absorbers (e.g. ozone). The ODAS algorithm, however, cannot yet handle a missing ozone profile. As such, we have switched back to missing ozone being a `FAILURE` error, regardless of whether or not the ODAS or ODPS transmittance algorithm is being used. See ticket 150[3].

Allowed for user profile top level pressures to be less than 0.005hPa in the ODAS algorithm. This corrected a bug that generated negative absorber amounts for the top layer when a user input a profile where the top level pressure is *less than* 0.005hPa. See ticket 151.

Fixed test of `SensorData%Tb` component The previous test (called within the `CRTM_Surface_IsValid` procedure) caused a `FAILURE` when *any* of the supplied brightness temperatures were less than zero. This test has been changed to fail only when *all* of the input brightness temperatures are less than zero to allow channel subsets of data to be passed. See ticket 110.

Corrected error mesage in `CRTM_Atmosphere_IsValid` function. The error message for invalid input absorber units was corrected. See ticket 141.

Coefficient load message suppression in the `CRTM_Init` function was not occurring correctly This problem was traced to a logic error in several of the coefficient load procedures when the optional MPI process identifier arguments were passed in. The logic has been corrected in the affected load procedures. See ticket 143.

Refactor for Compiler Defects

Memory leak in `CRTM_IRSSEM` module fixed This was a bug caused by apparent compiler bugs (in more than one compiler) where declaring the internals of a local (i.e. not `PUBLIC`) structure as `PRIVATE` caused a memory leak. Removal of the internal `PRIVATE` statement solved the problem. See ticket 144.

[3] The ticket references and links are included to allow CRTM developers to easily navigate to the CRTM Source Code Management system from this document

Modification of `Type_Kinds` module to allow for Intel ifort compilation This work around was necessary due to an ifort v11.1 compiler bug that surfaced due to the CRTM build switches for this compiler promote compiler warnings to errors. Rather than require users to modify their compilation setup to avoid this error, the `Type_Kinds` module was modified to avoind it entirely. See ticket 112.

Modification of `CRTM_Atmosphere_AddLayerCopy` procedure to allow for PGI pgf95 compilation The REL-2.0 version of the `CRTM_Atmosphere_AddLayerCopy` procedure was identified as a problem for the PGI pgf95 v10.2-1 compiler. A bug report was submitted to PGI Support and filed as TPR 16814. The bug is fixed in the v10.4 release of the pgf95 compiler, which does not have a problem with the original CRTM code. See ticket 114.

Reorganistion of Test/Example Programs

This update is probably the biggest change in REL-2.0.1. The CRTM tarball structure was updated to include the test/example codes – as opposed to supplying a separate tarball just for the example programs. The reasoning here was to establish the typical "`make`, `make test`" procedures for building packages, but be aware that the setup is still rather unsophisticated; we are still investigating ways to more easily configure the CRTM library and test/example programs (e.g. `autoconf`).

For a full description of the necessary steps to build the CRTM library and test/example programs, refer to the `README` file supplied with the CRTM release tarball.

What's New in v2.0.2

The v2.0.2 update to the CRTM was done to

- Fix two critical defects: one introduced in v2.0; another in the v2.0.1 update.
- Add additional tests.

Bug Fixes

Fix for specular reflection of IR sensors over water In v2.0, the reflectance behaviour for IR sensors over water was changed from Lambertian to specular. The problem with the update is due to the design of how sub-FOV surface differences are handled in the CRTM. Currently there is no way to handle a mixed land/water FOV where land reflectivity is assumed Lambertian and water reflectivity specular. The reflectivity behaviour ended up being that associated with the surface type having the largest FOV fraction. The temporary fix applied is that *all* IR sensor reflectivities are now treated as specular. See ticket 164.

Fix for invalid maximum number of azimuth angles for visible sensors To speed up visible sensor calculations in v2.0.1, the maximum number of azimuth angles used was switched from a fixed maximum to a dynamic one based on the number of Legendre terms required to properly simulate molecular scattering. However, the maximum number of azimuth angle assignment was being performed prior to the minimum acceptable value being set. This lead to an invalid value being specified for the number of azimuth angles in some cases. See ticket 165.

Addition of Test/Example Programs

An additional forward model test, `Example5_ClearSky`, was introduced to test the bug fixes mentioned above. All test comparison output files have been updated accordingly.

What's New in v2.0.4

The v2.0.4 update to the CRTM was done to

- Update the sensor coefficient files, including the renaming of the hyperspectral infrared sensor identifiers.
- Fix a number of defects.

Update of sensor coefficient files

An update of all the sensor coefficient files (the `SpcCoeff` and `TauCoeff` files) was carried out, introducing many new sensors. See table B.1 for the full listing of instruments for which there are CRTM datafiles.

Additionally, to facilitate use of a generic sensor identifier for CRTM datafiles containing channel subsets of hyperspectral instruments such as AIRS, IASI, and CrIS, the "all channel" files (constructed from the individual module or band files) are now tagged with the total channel count. The rename of the `Sensor_Id`'s for the current hyperspectral sensors are shown below.

Old `Sensor_Id`	**New** `Sensor_Id`
`airs_aqua`	`airs2378_aqua`
`iasi_metop-a`	`iasi8461_metop-a`
`iasi_metop-b`	`iasi8461_metop-b`
`cris_npp`	`cris1305_npp`

Bug Fixes

Fix an initialisation error in the `CRTM_IRSSEM` module In the adjoint procedure, `CRTM_Compute_IRSSEM_AD()` some local adjoint variable were not initialised prior to ttheir use. Under certain run conditions (based on compiler, platform, and sensors run), this was generated floating point exceptions and halting execution. The fix applied was to initialise the local adjoint variables in question. See ticket 259.

Fix a memory leak in the `ODPS_Predictor_Define` module A deallocation statement was missing a single structure component leading to a small memory leak. The fix applied was to ensure that the components of the allocation and deallocation statements matched. See ticket 260.

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 to allocate, destroy, copy etc. structures 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,

```
CRTM_Atmosphere_Define
CRTM_RTSolution_Define
```

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

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

1.1.3 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.

1.1.4 Naming of I/O Modules

Modules containing routines that read and write data from and to files are, naturally, termed I/O modules. Not all data structures have associated I/O modules. The naming convention adopted for these modules in the CRTM is,

`[CRTM_]`*name*`_Binary_IO`

or just

`[CRTM_]`*name*`_IO`

Some examples are,

```
CRTM_Atmosphere_IO
CRTM_RTSolution_IO
```

As with the other module types, the actual source code files for these modules have the same name with a "`.f90`" suffix.

In the context of the CRTM, the term "Binary" is a euphemisn for sequential, unformatted I/O in Fortran.

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, and ozone concentration[2] profiles.

[2] Additional trace gas absorption capabilities are being added.

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 A.4 and A.5 for the 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 gross surface types (land, water, snow, and ice). Each gross surface type has a specified number of specific surface types associated with it. See tables A.8, A.9, A.10, and A.11 for the land, water, snow, and ice surface types, respectively, that are valid in the CRTM.

The CRTM utilises separate models for each gross surface type for each spectral type (infrared and microwave). These models can be either physical models or database/LUT type of models.

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.

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,

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

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

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

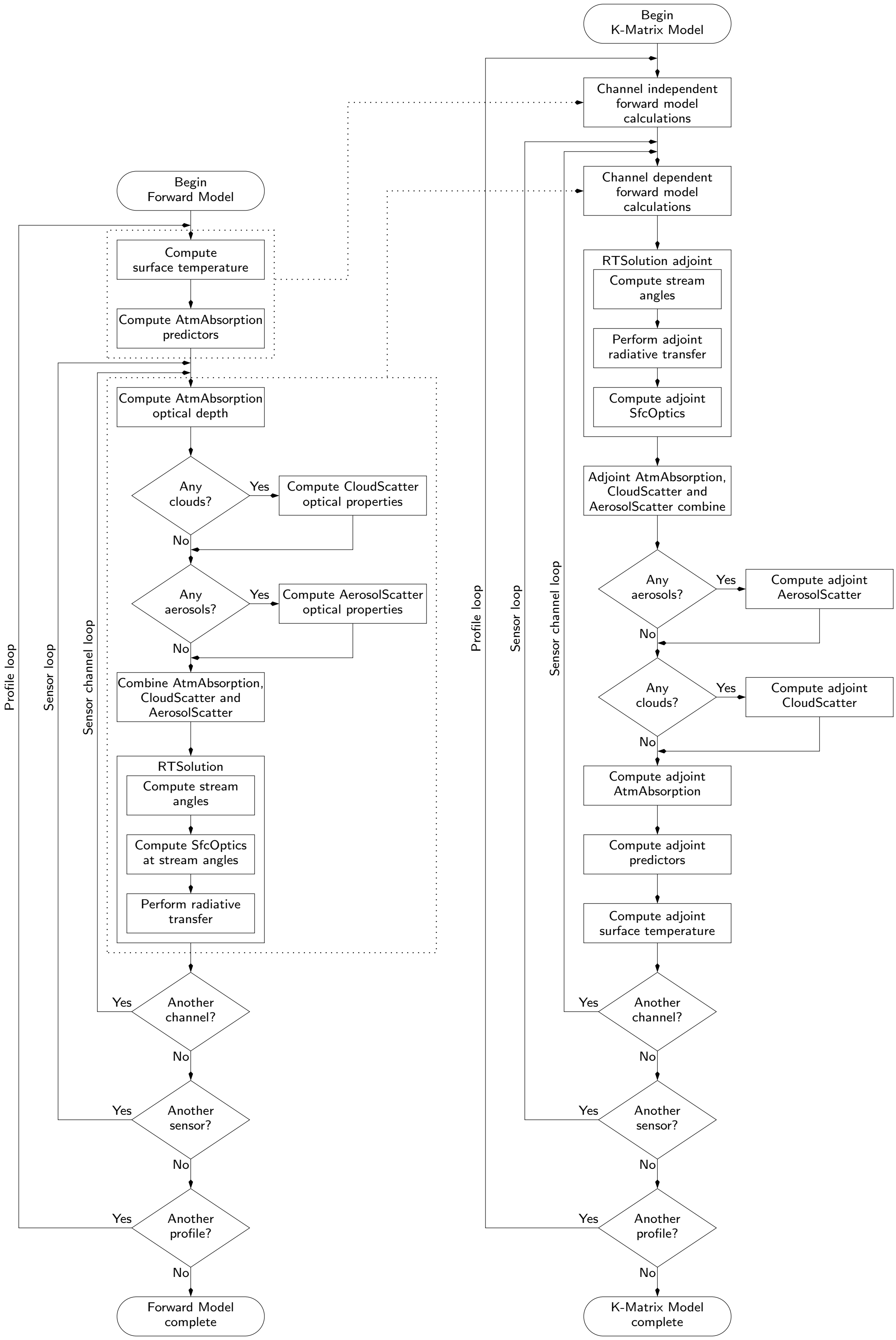
$$\delta^* x_l = \frac{\partial T_{B,l}}{\partial x} \tag{1.3}$$

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. 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.



(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 ftp download site

The CRTM source code and coefficients are released in a compressed tarball[1] via the CRTM ftp site:

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

The REL-2.0.4 release is available directly from

ftp://ftp.emc.ncep.noaa.gov/jcsda/CRTM/REL-2.0.4

Also note that additional releases, e.g. beta or experimental branches, may also made available on this ftp site.

2.2 Coefficient Data

All of the transmittance, spectral, cloud, aerosol, and emissivity coefficient data needed by the CRTM are available in the `fix/`[2] subdirectory. The coefficient directory structure is organised by coefficient and format type as shown in figure 2.1.

Both big- and little-endian format files are provided to save users the trouble of switching what they use for their system[3]. 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 same path (see the CRTM initialisation function description) so users will have to move/link the datafiles as required.

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

[2]The directory name "`fix`" is an NCEP standard name for a location containing files that do not change (frequently), i.e. they are "fixed".

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

```
CRTM_Coefficients/
├─ SpcCoeff/
│  ├─ Big_Endian/          } Spectral coefficients
│  └─ Little_Endian/
├─ TauCoeff/
│  ├─ ODAS/
│  │  ├─ Big_Endian/       } ODAS fast transmittance
│  │  └─ Little_Endian/      model coefficients
│  └─ ODPS/
│     ├─ Big_Endian/       } ODPS fast transmittance
│     └─ Little_Endian/      model coefficients
├─ AerosolCoeff/
│  ├─ Big_Endian/          } Aerosol optical properties
│  └─ Little_Endian/
├─ CloudCoeff/
│  ├─ Big_Endian/          } Cloud optical properties
│  └─ Little_Endian/
└─ EmisCoeff/
   ├─ Big_Endian/          } Surface emissivity model coefficients
   └─ Little_Endian/
```

Figure 2.1: The CRTM coefficients directory structure

3 How to build the CRTM library

3.1 Build Files

The build system for the CRTM is currently quite unsophisticated. It 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.

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`

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

3.3 Compilation Environment Setup

To set the compilation envars for your CRTM build, you need to source the required "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  -fconvert=big-endian  -ffast-math  -ffree-form
              -fno-second-underscore  -frecord-marker=4  -funroll-loops
              -ggdb  -static  -Wall
   FL:       gfortran
   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.

3.4 Building the library

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

```
make
```

If you are using the DEBUG compiler flags you may, unfortunately, see many warnings similar to:

```
Warning (137): Variable 'cosaz' at (1) is never used and never set
Warning (112): Variable 'rlongitude' at (1) is set but never used
Warning (140): Implicit conversion at (1) may cause precision loss
Warning: Unused dummy argument 'group_index' at (1)
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.

3.5 Testing the library

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 five 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
```

And there are four 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
```

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

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

Passed 14 of 14 tests.
Failed 0 of 14 tests.
```

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

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

Passed 10 of 14 tests.
Failed 4 of 14 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.

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
```

4 How to use the CRTM library

4.1 Step by Step Guide

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.

The examples shown here assume you are processing one sensor at a time. The CRTM can handle multiple sensors at once, but specifying the input information in a simple way is difficult; e.g. the `Geometry` structure that is used to specify the sensor viewing geometry – even sensors on the same platform typically have different numbers of fields-of-view (FOVs) per scan. For multiple sensor processing, we'll assume they will be separately processed in parallel.

Because 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), let's approach this via examples for a fixed number of atmospheric profiles, and a known sensors. 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 – refered 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.

4.1.1 Step 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.

4.1.2 Step 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 assume you want to process, say, 50 profiles for the NOAA-18 AMSU-A sensor which has 15 channels. The forward model declarations would look something like,

```
! Processing parameters
INTEGER      , PARAMETER :: N_SENSORS  =  1
INTEGER      , PARAMETER :: N_CHANNELS = 15
INTEGER      , PARAMETER :: N_PROFILES = 50
CHARACTER(*), PARAMETER :: SENSOR_ID(N_SENSORS) = (/'amsua_n18'/)
TYPE(CRTM_ChannelInfo_type)  :: chInfo(N_SENSORS)
TYPE(CRTM_Geometry_type)     :: geo(N_PROFILES)
TYPE(CRTM_Options_type)      :: opt(N_PROFILES)
! Forward declarations
TYPE(CRTM_Atmosphere_type)   :: atm(N_PROFILES)
TYPE(CRTM_Surface_type)      :: sfc(N_PROFILES)
TYPE(CRTM_RTSolution_type)   :: rts(N_CHANNELS,N_PROFILES)
```

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)   :: atm_K(N_CHANNELS,N_PROFILES)
TYPE(CRTM_Surface_type)      :: sfc_K(N_CHANNELS,N_PROFILES)
TYPE(CRTM_RTSolution_type)   :: rts_K(N_CHANNELS,N_PROFILES)
```

4.1.3 Step 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. We'll assume that all the required datafiles reside in the subdirectory `./coeff_data` and follow on from the example of Step 2. The CRTM initialisation is profile independent, so we're only dealing with sensor information here. The CRTM initialisation function call looks like,

```
INTEGER :: errStatus
....
errStatus = CRTM_Init( SENSOR_ID, chInfo, File_Path='./coeff_data' )
IF ( errStatus /= SUCCESS ) THEN
  handle error...
END IF
```

Here we see for the first time how the 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.

For a list of all the accepted sensor identifiers, see appendix B.

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

4.1.4 Step 4: Allocate the CRTM structures

Now we need to create instances of the various CRTM structures where necessary to hold the input or output data. Functions are used to perform any necessary component allocations allocations. The function 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.

Allocation of the Atmosphere structures

First, we'll allocate the atmosphere structures to the required dimensions. The forward variable is allocated like so:

```
! Allocate the forward atmosphere structure
CALL CRTM_Atmosphere_Create( atm        , &  ! Object
                             n_Layers   , &  ! Input
                             n_Absorbers, &  ! Input
                             n_Clouds   , &  ! Input
                             n_Aerosols   )  ! Input
! Check it was actually allocated
IF ( ANY(.NOT. CRTM_Atmosphere_Associated( atm )) ) THEN
  handle error...
END IF
```

and the K-matrix variable is allocated by looping over all profiles[2],

```
! Allocate the K-matrix atmosphere structure
DO m = 1, N_PROFILES
  CALL CRTM_Atmosphere_Create( atm_k(:,m) , &  ! Object
                               n_Layers   , &  ! Input
                               n_Absorbers, &  ! Input
                               n_Clouds   , &  ! Input
                               n_Aerosols   )  ! Input
  ! Check they were actually allocated
  IF ( ANY(.NOT. CRTM_Atmosphere_Associated( atm_k(:,m) )) ) THEN
    handle error...
  END IF
END DO
```

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 algoithm, CO_2 can also be specified. In future releases, trace gases such as CO, CH_4, and N_2O will also be accepted as input.

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     , &  ! Object
                             n_Layers  )  ! Input
```

[2]The `CRTM_Atmosphere_Create` function is defined as elemental so the profile loop is not strictly needed

```
! Check it was actually allocated
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.

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 structure
CALL CRTM_Options_Create( opt        , &  ! Object
                          n_Channels  )   ! Input
! Check it was actually allocated
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.

4.1.5 Step 5: Fill the CRTM input structures with data

This step simply entails filling the input `atm`, `sfc`, `geo`, and, if used, `opt` structures with the required information. However, there are some issues that need to be mentioned:

- 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.
- In the `Atmosphere` structure, the `Climatology` component is not yet used.
- In the `Atmosphere` structure, *both* the level and layer pressure profiles must be specified.
- In the `Atmosphere` structure, the absorber profile data units *must* be mass mixing ratio for water vapour and ppmv for other absorbers. The `Absorber_Units` component is not yet utilised to allow conversion of different user-supplied concentration units.
- In the `Atmosphere` structure, the `Absorber_Id` array must be set to the correct absorber identifiers (see table A.2) to allow the software to find a particular absorber. There is no necessary order in specifying the concentration profiles for different gaseous absorbers.
- In the `Surface` structure, the sum of the coverage types *must* add up to 1.0.
- In the `Geometry` structure, the sensor zenith and sensor scan angles should be consistent.
- Graphical definitions of the `Geometry` structure sensor scan, sensor zenith, sensor azimuth, source zenith, and source azimuth angles are shown in figures A.8, A.9, A.10, A.11, and A.12 respectively.
- The `Options` structure contains two "substructures", `SSU_Input` and `Zeeman_Input` to hold the necessary inputs for the SSU and Zeeman transmittance models. These substructures are private and can only be access via "`GetValue`" and "`SetValue`" functions as discussed further below.

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

Filling the Options substructures for SSU and Zeeman model input

As mentioned above, the SSU_Input and Zeeman_Input data structures are private. This means the contents of the structure cannot be accessed directly, but via helper subroutines. For example, to set the SSU instrument mission time, one would call the SSU_Input_SetValue subroutine,

```
! 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.

Similarly for the necessary Zeeman model parameters,

```
! 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.

4.1.6 Step 6: Call the required CRTM function

At this point, much of the prepatory heavy lifting has been done. The CRTM function calls themselves are quite simple. For the forward model we do,

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

and for the K-matrix model, the calling syntax is,

```
errStatus = 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 ( errStatus /= SUCCESS ) THEN
  handle error...
END IF
```

Note that the K-matrix model also returns the forward model radiances. The tangent-linear and adjoint models have similar call structures and will not be shown here.

4.1.7 Step 7: Destroy the CRTM and cleanup

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

```
errStatus = CRTM_Destroy( chInfo )
IF ( errStatus /= SUCCESS ) THEN
  handle error...
END IF
```

to deallocate all the shared coefficient data, as well as calling the individual structure destroy functions to deallocate as required. For the example here, that entails deallocating the various structure arrays that were created in Step 4.1.4. The cleanup mirrors that of the create step:

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

4.2 Interface Descriptions

4.2.1 CRTM_Init interface

```
NAME:
      CRTM_Init

PURPOSE:
      Function to initialise the CRTM.

CALLING SEQUENCE:
      Error_Status = CRTM_Init( Sensor_ID                            , &
                                ChannelInfo                          , &
                                CloudCoeff_File    = CloudCoeff_File   , &
                                AerosolCoeff_File = AerosolCoeff_File, &
```

```
                        EmisCoeff_File    = EmisCoeff_File   , &
                        File_Path         = File_Path        , &
                        Load_CloudCoeff   = Load_CloudCoeff  , &
                        Load_AerosolCoeff = Load_AerosolCoeff, &
                        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:
      CloudCoeff_File:    Name of the CRTM Binary format CloudCoeff file
                          containing the scattering coefficient data. If not
                          specified the default filename is "CloudCoeff.bin".
                          UNITS:      N/A
                          TYPE:       CHARACTER(*)
                          DIMENSION:  Scalar
                          ATTRIBUTES: INTENT(IN), OPTIONAL

      AerosolCoeff_File:  Name of the CRTM Binary format AerosolCoeff file
                          containing the aerosol absorption and scattering
                          coefficient data. If not specified the default
                          filename is "AerosolCoeff.bin".
                          UNITS:      N/A
                          TYPE:       CHARACTER(*)
                          DIMENSION:  Scalar
                          ATTRIBUTES: INTENT(IN), OPTIONAL

      EmisCoeff_File:     Name of the CRTM Binary format EmisCoeff file
                          containing the IRSSEM coefficient data. If not
                          specified the default filename is "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

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

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.

RESTRICTIONS:
      If specified, the length of the combined file path and filename strings
      cannot exceed 2000 characters.
```

4.2.2 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.
```

```
- The INTENT on the output RTSolution 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.
```

4.2.3 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.

      - The INTENT on the output RTSolution arguments are IN OUT rather
```

```
    than just OUT. This is necessary because the arguments may be defined
    upon input. To prevent memory leaks, the IN OUT INTENT is a must.
```

4.2.4 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.

      - The INTENT on the output RTSolution, Atmosphere_AD, and
        Surface_AD arguments are IN OUT rather than just OUT. This is
        necessary because the arguments should be defined upon input.
        To prevent memory leaks, the IN OUT INTENT is a must.
```

4.2.5 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.

        - The INTENT on the output RTSolution, Atmosphere_K, and Surface_K,
          arguments are IN OUT rather than just OUT. This is necessary because
          the arguments should be defined upon input. To prevent memory leaks,
          the IN OUT INTENT is a must.
```

4.2.6 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:  Scalar
                    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.
```

Bibliography

F.M. Breon. An analytical model for the cloud-free atmosphere/ocean system reflectance. *Remote Sens. Environ.*, 43(2):179–192, 1993.

M. Chin, P. Ginoux, S. Kinne, O. Torres, B.N. Holben, B.N. Duncan, R.V. Martin, J.A. Logan, A. Higurashi, and T. Nakajima. Tropospheric aerosol optical thickness from the GOCART model and comparisons with satellite and sun photometer meaasurements. *J. Atmos. Sci.*, 59:461–483, 2002.

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

N.R. Nalli, P.J. Minnett, and P. van Delst. Emissivity and reflection model for calculating unpolarized isotropic water surface-leaving radiance in the infrared. 1: Theoretical development and calculations. *Appl. Opt.*, 47 (21):3701–3721, 2008a.

X. Wu and W.L. Smith. Emissivity of rough sea surface for 8-13μm: modeling and verification. *Appl. Opt.*, 36 (12):2609–2619, 1997.

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           :: WMO_Satellite_ID = INVALID_WMO_SATELLITE_ID
  INTEGER           :: WMO_Sensor_ID    = INVALID_WMO_SENSOR_ID
  INTEGER           :: Sensor_Index     = 0
  ! Array data
  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_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.1.3 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.1.4 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 )

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

A.1.5 CRTM_ChannelInfo_n_Channels interface

```
NAME:
      CRTM_ChannelInfo_n_Channels

PURPOSE:
      Function to return the number of channels defined in a ChannelInfo
      structure or structure array

CALLING SEQUENCE:
      n_Channels = CRTM_ChannelInfo_n_Channels( ChannelInfo )

INPUTS:
      ChannelInfo: ChannelInfo structure or structure which is to have its
                   channels counted.
                   UNITS:      N/A
                   TYPE:       TYPE(CRTM_ChannelInfo_type)
                   DIMENSION:  Scalar
                                 or
                               Rank-1
                   ATTRIBUTES: INTENT(IN)

FUNCTION RESULT:
      n_Channels:   The number of defined channels in the input argument.
                   UNITS:      N/A
                   TYPE:       INTEGER
                   DIMENSION:  Scalar
```

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.

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 A.1: CRTM `Atmosphere` structure valid `Climatology` definitions. The same set as defined for LBLRTM is used.

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

Table A.2: CRTM `Atmosphere` structure valid `Absorber_ID` definitions. The same molecule set as defined for HITRAN is used.

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 A.3: CRTM `Atmosphere` structure valid `Absorber_Units` definitions. The same set as defined for LBLRTM is used.

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_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
            ATTRIBUTES: INTENT(IN)
```

A.2.8 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.2.9 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.2.10 CRTM_Atmosphere_IOVersion interface

```
NAME:
      CRTM_Atmosphere_IOVersion

PURPOSE:
      Subroutine to return the module version information.

CALLING SEQUENCE:
      CALL CRTM_Atmosphere_IOVersion( Id )

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

A.2.11 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.2.12 *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.2.13 *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.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.

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

Table A.4: CRTM Cloud structure valid Type definitions.

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_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
                     ATTRIBUTES: INTENT(IN)
```

A.3.8 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.3.9 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.3.10 CRTM_Cloud_IOVersion interface

```
NAME:
```

```
      CRTM_Cloud_IOVersion

PURPOSE:
      Subroutine to return the module version information.

CALLING SEQUENCE:
      CALL CRTM_Cloud_IOVersion( 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.3.11 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.3.12 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.3.13 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.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 :: Concentration(:)     ! K. Units are kg/m^2
END TYPE CRTM_Aerosol_type
```

Figure A.4: CRTM_Aerosol_type structure definition.

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 A.5: CRTM Aerosol structure valid Type definitions and effective radii. SSAM $\equiv$ Sea Salt Accumulation Mode, SSCM $\equiv$ Sea Salt Coarse Mode.

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_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
                     ATTRIBUTES: INTENT(IN)
```

A.4.8 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.4.9 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.4.10 CRTM_Aerosol_IOVersion interface

```
NAME:
```

```
      CRTM_Aerosol_IOVersion

PURPOSE:
      Subroutine to return the module version information.

CALLING SEQUENCE:
      CALL CRTM_Aerosol_IOVersion( 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.4.11 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.4.12 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.4.13 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.5 Surface Structure

```
TYPE :: CRTM_Surface_type
  ! Allocation indicator
  LOGICAL :: Is_Allocated = .TRUE.  ! Placeholder for future expansion
  ! Dimension values
  ! ...None yet
  ! 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
  ! 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.

Component	Description	Units	Dimensions
n_Sensors	The number of sensors for which data is provided inside the SensorData structure	N/A	Scalar
Land_Coverage	Fraction of the FOV that is land surface	N/A	Scalar
Water_Coverage	Fraction of the FOV that is water surface	N/A	Scalar
Snow_Coverage	Fraction of the FOV that is snow surface	N/A	Scalar
Ice_Coverage	Fraction of the FOV that is ice surface	N/A	Scalar
Wind_Speed	Surface wind speed	$m.s^{-1}$	Scalar
Wind_Direction	Surface wind direction	deg. E from N	Scalar
Land_Type	Land surface type	N/A	Scalar
Land_Temperature	Land surface temperature	Kelvin	Scalar
Soil_Moisture_Content	Volumetric water content of the soil	$g.cm^{-3}$	Scalar
Canopy_Water_Content	Gravimetric water content of the canopy	$g.cm^{-3}$	Scalar
Vegetation_Fraction	Vegetation fraction of the surface	%	Scalar
Soil_Temperature	Soil temperature	Kelvin	Scalar
Water_Type	Water surface type	N/A	Scalar
Water_Temperature	Water surface temperature	Kelvin	Scalar
Salinity	Water salinity	‰	Scalar
Snow_Type	Snow surface type	N/A	Scalar
Snow_Temperature	Snow surface temperature	Kelvin	Scalar
Snow_Depth	Snow depth	mm	Scalar
Snow_Density	Snow density	$g.m^{-3}$	Scalar
Snow_Grain_Size	Snow grain size	mm	Scalar
Ice_Type	Ice surface type	N/A	Scalar
Ice_Temperature	Ice surface temperature	Kelvin	Scalar
Ice_Thickness	Thickness of ice	mm	Scalar
Ice_Density	Density of ice	$g.m^{-3}$	Scalar
Ice_Roughness	Measure of the surface roughness of the ice	N/A	Scalar
SensorData	Satellite sensor data required for some surface emissivity algorithms	N/A	Scalar

Table A.6: CRTM `Surface` structure component description.

Parameter	Value	Units
Surface type independent data		
DEFAULT_WIND_SPEED	5.0	$m.s^{-1}$
DEFAULT_WIND_DIRECTION	0.0	deg. E from N
Land surface type data		
DEFAULT_LAND_TYPE	GRASS_SOIL	N/A
DEFAULT_LAND_TEMPERATURE	283.0	K
DEFAULT_SOIL_MOISTURE_CONTENT	0.05	$g.cm^{-3}$
DEFAULT_CANOPY_WATER_CONTENT	0.05	$g.cm^{-3}$
DEFAULT_VEGETATION_FRACTION	0.3	30%
DEFAULT_SOIL_TEMPERATURE	283.0	K
Water type data		
DEFAULT_WATER_TYPE	SEA_WATER	N/A
DEFAULT_WATER_TEMPERATURE	283.0	K
DEFAULT_SALINITY	33.0	ppmv
Snow surface type data		
DEFAULT_SNOW_TYPE	NEW_SNOW	N/A
DEFAULT_SNOW_TEMPERATURE	263.0	K
DEFAULT_SNOW_DEPTH	50.0	mm
DEFAULT_SNOW_DENSITY	0.2	$g.cm^{-3}$
DEFAULT_SNOW_GRAIN_SIZE	2.0	mm
Ice surface type data		
DEFAULT_ICE_TYPE	FRESH_ICE	N/A
DEFAULT_ICE_TEMPERATURE	263.0	K
DEFAULT_ICE_THICKNESS	10.0	mm
DEFAULT_ICE_DENSITY	0.9	$g.cm^{-3}$
DEFAULT_ICE_ROUGHNESS	0.0	N/A

Table A.7: CRTM `Surface` structure default values.

Land Type	Parameter
Compacted soil	COMPACTED_SOIL
Tilled soil	TILLED_SOIL
Sand	SAND
Rock	ROCK
Irrigated low vegetation	IRRIGATED_LOW_VEGETATION
Meadow grass	MEADOW_GRASS
Scrub	SCRUB
Broadleaf forest	BROADLEAF_FOREST
Pine forest	PINE_FOREST
Tundra	TUNDRA
Grass-soil	GRASS_SOIL
Broadleaf-pine forest	BROADLEAF_PINE_FOREST
Grass scrub	GRASS_SCRUB
Soil-grass-scrub	SOIL_GRASS_SCRUB
Urban concrete	URBAN_CONCRETE
Pine brush	PINE_BRUSH
Broadleaf brush	BROADLEAF_BRUSH
Wet soil	WET_SOIL
Scrub-soil	SCRUB_SOIL
Broadleaf(70)-Pine(30)	BROADLEAF70_PINE30

Table A.8: CRTM `Surface` structure valid `Land_Type` definitions.

Water Type	Parameter
Sea water	SEA_WATER
Fresh water	FRESH_WATER

Table A.9: CRTM `Surface` structure valid `Water_Type` definitions.

Snow Type	Parameter
Wet snow	WET_SNOW
Grass after snow	GRASS_AFTER_SNOW
Powder snow	POWDER_SNOW
RS snow(A)	RS_SNOW_A
RS snow(B)	RS_SNOW_B
RS snow(C)	RS_SNOW_C
RS snow(D)	RS_SNOW_D
RS snow(E)	RS_SNOW_E
Thin Crust snow	THIN_CRUST_SNOW
Thick crust snow	THICK_CRUST_SNOW
Shallow snow	SHALLOW_SNOW
Deep snow	DEEP_SNOW
Crust snow	CRUST_SNOW
Medium snow	MEDIUM_SNOW
Bottom crust snow(A)	BOTTOM_CRUST_SNOW_A
Bottom crust snow(B)	BOTTOM_CRUST_SNOW_B

Table A.10: CRTM `Surface` structure valid `Snow_Type` definitions.

Ice Type	Parameter
Fresh ice	FRESH_ICE
First year sea ice	FIRST_YEAR_SEA_ICE
Multiple year sea ice	MULTI_YEAR_SEA_ICE
Ice floe	ICE_FLOE
Ice ridge	ICE_RIDGE

Table A.11: CRTM Surface structure valid Ice_Type definitions.

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_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.5.8 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.5.9 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.5.10 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.5.11 CRTM_Surface_IOVersion interface

```
NAME:
      CRTM_Surface_IOVersion

PURPOSE:
      Subroutine to return the module version information.

CALLING SEQUENCE:
      CALL CRTM_Surface_IOVersion( Id )

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

A.5.12 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.5.13 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.5.14 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.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.

Component	Description	Units	Dimensions
n_Channels	Number of channels to use in SfcOptics emissivty algorithms (L)	N/A	Scalar
Sensor_Id	The sensor id	N/A	Scalar
WMO_Satellite_Id	The WMO satellite Id	N/A	Scalar
WMO_Sensor_Id	The WMO sensor Id	N/A	Scalar
Sensor_Channel	The channel numbers	N/A	L
Tb	The brightness temperature measurements for each channel	Kelvin	L

Table A.12: CRTM SensorData structure component description.

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_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.6.7 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.6.8 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.6.9 CRTM_SensorData_IOVersion interface

```
NAME:
      CRTM_SensorData_IOVersion

PURPOSE:
      Subroutine to return the module version information.

CALLING SEQUENCE:
      CALL CRTM_SensorData_IOVersion( 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.6.10 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.6.11 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.6.12 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.7 Geometry Structure

```
TYPE :: CRTM_Geometry_type
  ! Allocation indicator
  LOGICAL :: Is_Allocated = .TRUE.  ! Placeholder for future expansion
  ! 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 = ZERO
  ! 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.

Component	Description	Units	Dimensions
`iFOV`	The scan line FOV index	N/A	Scalar
`Longitude`	Earth longitude	deg. E (0→360)	Scalar
`Latitude`	Earth latitude	deg. N (-90→+90)	Scalar
`Surface_Altitude`	Altitude of the Earth's surface at the specified lon/lat location	metres (m)	Scalar
`Sensor_Scan_Angle`	The sensor scan angle from nadir. See fig.A.8	degrees	Scalar
`Sensor_Zenith_Angle`	The sensor zenith angle of the FOV. See fig.A.9	degrees	Scalar
`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.A.10	deg. from N	Scalar
`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.A.11	degrees	Scalar
`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.A.12	deg. from N	Scalar
`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	Scalar
`Year`	The year in 4-digit format	N/A	Scalar
`Month`	The month of year (1-12)	N/A	Scalar
`Day`	The day of month (1-28/29/30/31)	N/A	Scalar

Table A.13: CRTM `Geometry` structure component description.

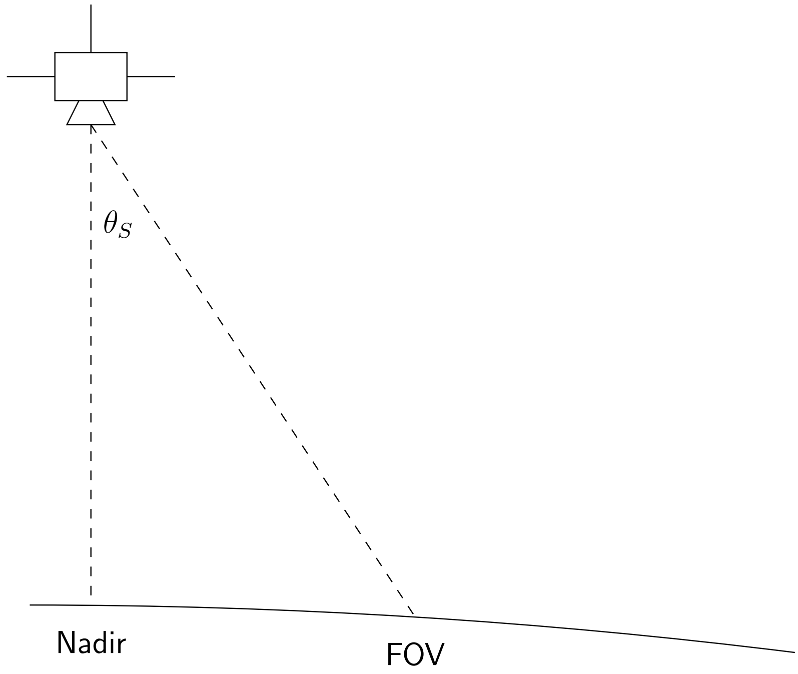


Figure A.8: Definition of `Geometry` sensor scan angle component.

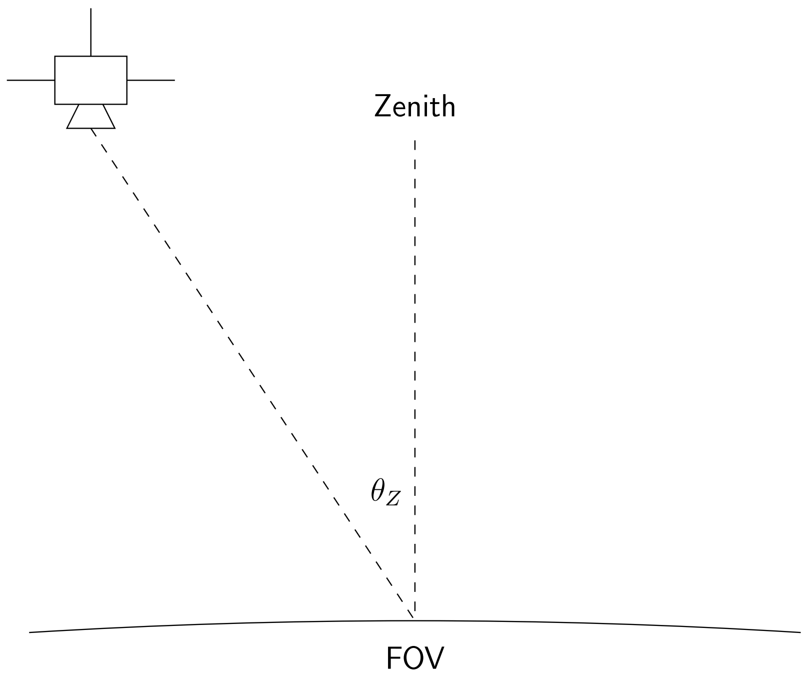


Figure A.9: Definition of `Geometry` sensor zenith angle component.

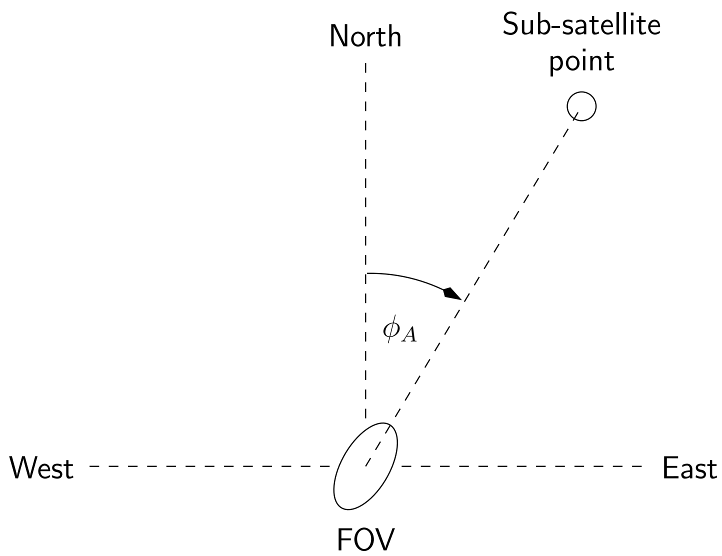


Figure A.10: Definition of `Geometry` sensor azimuth angle component.

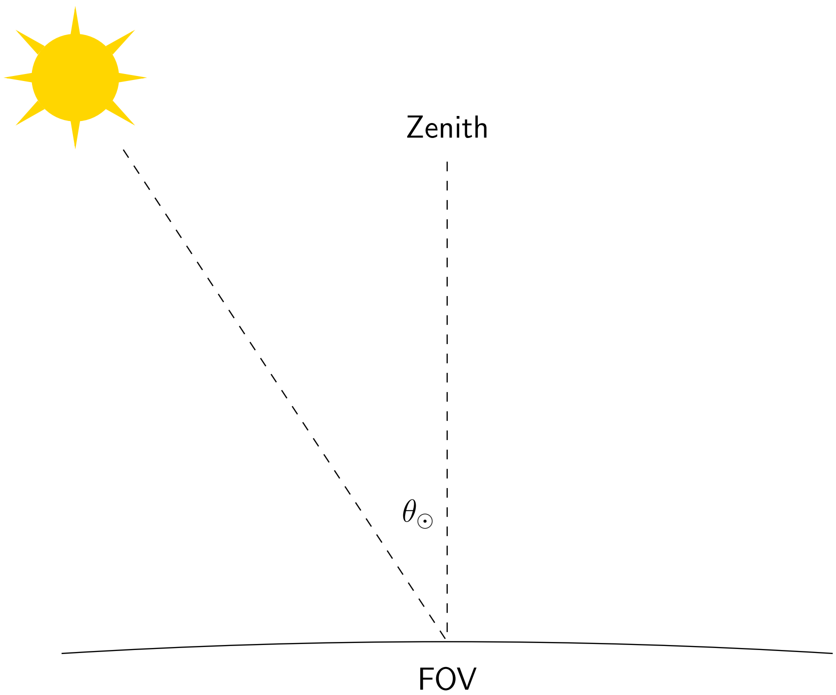


Figure A.11: Definition of `Geometry` source zenith angle component.

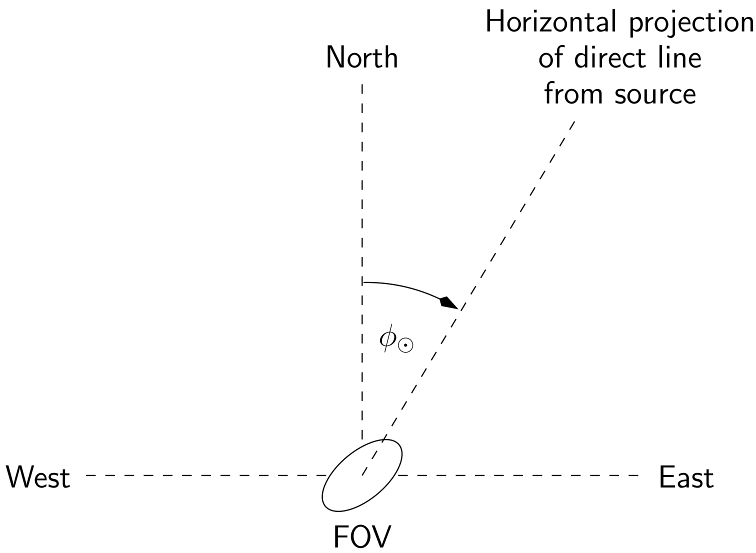


Figure A.12: Definition of `Geometry` source azimuth angle component.

A.7.1 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.7.2 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.7.3 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.7.4 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.7.5 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.7.6 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.7.7 CRTM_Geometry_IOVersion interface

```
NAME:
      CRTM_Geometry_IOVersion

PURPOSE:
      Subroutine to return the module version information.

CALLING SEQUENCE:
      CALL CRTM_Geometry_IOVersion( 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.7.8 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.7.9 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     , &
                                             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

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.7.10 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  )

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

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 RTSolution Structure

```
TYPE :: CRTM_RTSolution_type
  ! Allocation indicator
  LOGICAL :: Is_Allocated = .FALSE.
  ! Dimensions
  INTEGER :: n_Layers = 0  ! K
  ! Internal variables. Users do not need to worry about these.
  LOGICAL :: Scattering_Flag = .TRUE.
  INTEGER :: n_Full_Streams  = 0
  INTEGER :: n_Stokes        = 0
  ! 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) :: 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
  ! The layer optical depths
  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.13: CRTM_RTSolution_type structure definition.

Component	Description	Units	Dimensions
n_Layers	Number of atmospheric profile layers (K)	N/A	Scalar
Surface_Emissivity	The computed surface emissivity	N/A	Scalar
Up_Radiance	The atmospheric portion of the upwelling radiance	mW/(m^2.sr.cm^{-1})	Scalar
Down_Radiance	The atmospheric portion of the downwelling radiance	mW/(m^2.sr.cm^{-1})	Scalar
Down_Solar_Radiance	The downwelling direct solar radiance	mW/(m^2.sr.cm^{-1})	Scalar
Surface_Planck_Radiance	The surface radiance	mW/(m^2.sr.cm^{-1})	Scalar
Upwelling_Radiance	The upwelling radiance profile, including the reflected downwelling and surface contributions.	mW/(m^2.sr.cm^{-1})	K
Layer_Optical_Depth	The layer optical depth profile	N/A	K
Radiance	The sensor radiance	mW/(m^2.sr.cm^{-1})	Scalar
Brightness_Temperature	The sensor brightness temperature	Kelvin	Scalar

Table A.14: CRTM RTSolution structure component description

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:  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.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_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
                     ATTRIBUTES: INTENT(IN)
```

A.8.7 CRTM_RTSolution_IOVersion interface

```
NAME:
      CRTM_RTSolution_IOVersion

PURPOSE:
      Subroutine to return the module version information.

CALLING SEQUENCE:
      CALL CRTM_RTSolution_IOVersion( Id )

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

A.8.8 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.8.9 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.8.10 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.9 Options Structure

```
TYPE :: CRTM_Options_type
  ! Allocation indicator
  LOGICAL :: Is_Allocated = .FALSE.
  ! Input checking on by default
  LOGICAL :: Check_Input = .TRUE.
  ! User defined emissivity/reflectivity
  ! ...Dimensions
  INTEGER :: n_Channels = 0  ! L dimension
  ! ...Index into channel-specific components
  INTEGER :: Channel = 0
  ! ...Emissivity optional arguments
  LOGICAL :: Use_Emissivity = .FALSE.
  REAL(fp), ALLOCATABLE :: Emissivity(:)  ! L
  ! ...Direct reflectivity optional arguments
  LOGICAL :: Use_Direct_Reflectivity = .FALSE.
  REAL(fp), ALLOCATABLE :: Direct_Reflectivity(:) ! L
  ! Antenna correction application
  LOGICAL :: Use_Antenna_Correction = .FALSE.
  ! SSU instrument input
  TYPE(SSU_Input_type) :: SSU
  ! Zeeman-splitting input
  TYPE(Zeeman_Input_type) :: Zeeman
END TYPE CRTM_Options_type
```

Figure A.14: CRTM_Options_type structure definition.

Component	Description	Units	Dimensions
`Check_Input`	Logical switch to enable or disable input data checking. If: `.FALSE.`: No input data check. `.TRUE.` : Input data *is* checked [DEFAULT].	N/A	Scalar
`n_Channels`	Number of sensor channels (`L`).	N/A	Scalar
`Channel`	Index into channel-specific components.	N/A	Scalar
`Use_Emissivity`	Logical switch to apply user-defined surface emissivity. If: `.FALSE.`: Calculate emissivity [DEFAULT]. `.TRUE.` : Use user-defined emissivity	N/A	Scalar
`Emissivity`	User-defined surface emissivity for each sensor channel.	N/A	`L`
`Use_Direct_Reflectivity`	Logical switch to apply 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 [DEFAULT]. `.TRUE.` : Use user-defined reflectivity	N/A	Scalar
`Direct_Reflectivity`	User-defined direct reflectivity for downwelling source for each sensor channel.	N/A	`L`
`Use_Antenna_Correction`	Logical switch to apply 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 [DEFAULT]. `.TRUE.` : Apply antenna correction.	N/A	Scalar
`SSU`	Structure component containing optional SSU sensor-specific input. See section A.10.	N/A	Scalar
`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	Scalar

Table A.15: CRTM `Options` structure component description

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_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.9.6 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.10 SSU_Input Structure

The SSU_Input structure is a component of the Options input structure. Note in figure A.15 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
  ! Time in decimal year (e.g. 2009.08892694 corresponds to 11:00 Feb. 2, 2009)
  REAL(fp) :: Time = ZERO
  ! SSU CO2 cell pressures (hPa)
  REAL(fp) :: Cell_Pressure(MAX_N_CHANNELS) = ZERO
END TYPE SSU_Input_type
```

Figure A.15: SSU_Input_type structure definition.

Component	Description	Units	Dimensions
Time	Time in decimal year corresponding to SSU observation.	N/A	Scalar
Cell_Pressure	The SSU CO_2 cell pressures.	hPa	MAX_N_CHANNELS (3)

Table A.16: CRTM SSU_Input structure component description

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_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.11 Zeeman_Input Structure

The Zeeman_Input structure is a component of the Options input structure. Note in figure A.16 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
  ! Earth magnetic field strength in Gauss
  REAL(fp) :: Be = DEFAULT_MAGENTIC_FIELD
  ! Cosine of the angle between the Earth
  ! magnetic field and wave propagation direction
  REAL(fp) :: Cos_ThetaB = ZERO
  ! Cosine of the azimuth angle of the Be vector.
  REAL(fp) :: Cos_PhiB = ZERO
  ! Doppler frequency shift caused by Earth-rotation.
  REAL(fp) :: Doppler_Shift = ZERO
END TYPE Zeeman_Input_type
```

Figure A.16: Zeeman_Input_type structure definition.

Component	Description	Units	Dimensions
Be	Earth magnetic field strength.	Gauss	Scalar
Cos_ThetaB	Cosine of the angle between the Earth magnetic field and wave propagation direction.	N/A	Scalar
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. $\mathbf{h} = (\mathbf{z} \times \mathbf{k})/|\mathbf{z} \times \mathbf{k}|$. 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	Scalar
Doppler_Shift	Doppler frequency shift caused by Earth-rotation (positive towards sensor). A zero value means no frequency shift.	KHz	Scalar

Table A.17: CRTM Zeeman_Input structure component description

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_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
```

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
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
NOAA-16 AVHRR/3	avhrr3_n16	yes	yes

Continued on Next Page…

Table B.1 – Continued

Instrument	Sensor Id	ODAS available	ODPS available
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
GOES-14 Imager	`imgr_g14`	yes	yes

Continued on Next Page...

Table B.1 – Continued

Instrument	Sensor Id	ODAS available	ODPS available
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
GOES-11 Sounder (Detector 1)	sndrD1_g11	yes	yes

Continued on Next Page…

Table B.1 – Continued

Instrument	Sensor Id	ODAS available	ODPS available
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
TRMM TMI	tmi_trmm	yes	yes

Continued on Next Page...

Table B.1 – Continued

Instrument	Sensor Id	ODAS available	ODPS available
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-1.2.x to REL-2.0.x

This section details the user code changes that need to be made to migrate from using CRTM v1.2.x to v2.0.x.

C.1 CRTM Initialization

The `Sensor_Id` argument to the CRTM initialisation function identifies the sensors for which the CRTM will be initialised. In v1.2.x this argument was optional because generic `SpcCoeff` and `TauCoeff` files could be used. In v2.0.x, generic `SpcCoeff` and `TauCoeff` coefficient files are no longer accepted and, thus, a sensor identifier *must* be specified.

In v1.2.x the `CRTM_Init` interface looked like:

```
errStatus = CRTM_Init( ChannelInfo, Sensor_ID=Sensor_ID )
```

where `Sensor_Id` is optional. The v2.0.x interface is now,

```
errStatus = CRTM_Init( Sensor_ID, ChannelInfo )
```

where both the `Sensor_Id` and `ChannelInfo` arguments are mandatory. See the `CRTM_Init` section for complete details about the v2.0.x interface.

C.2 CRTM Structure Life Cycle Changes

As mentioned in the "What's New in v2.0" section, the user-accessible structures (i.e. those used to define the inputs to, and return the outputs from, the CRTM) and their associated life cycle procedures (i.e. allocation and deallocation) have been changed. To mitigate the possibility of memory leaks, the definitions of array members of structures have had their `POINTER` attribute replaced with `ALLOCATABLE`. This was a first step in preparation for use of Fortran2003 Object Oriented features in the CRTM (once Fortran2003 compiler become widely available), where the derived type structure definitions will be reclassified as objects and their procedures will be type-bound. The changes in the affected user-accessible structure procedures are shown below.

In addition to the general interface changes, all of the structure life cycle procedures are now elemental. That is, there is no longer a restriction on the dimensionality of the arguments as long as they are conformable.

C.2.1 Atmosphere

Creation

In v1.2.x the Atmosphere structure allocation was a function returning an error status,

```
errStatus = CRTM_Allocate_Atmosphere( n_Layers    , &
                                      n_Absorbers, &
                                      n_Clouds    , &
                                      n_Aerosols  , &
                                      Atmosphere    )
IF ( errStatus /= SUCCESS ) THEN
  ...
END IF
```

The v2.0.x interface was changed to an elemental subroutine,

```
CALL CRTM_Atmosphere_Create( Atmosphere , &
                             n_Layers    , &
                             n_Absorbers, &
                             n_Clouds    , &
                             n_Aerosols    )
IF ( .NOT. CRTM_Atmosphere_Associated( Atmosphere ) ) THEN
  ...
END IF
```

where the error checking is achieved via the `CRTM_Atmosphere_Associated` function call.

Destruction

In v1.2.x the Atmosphere structure destruction was a function returning an error status,

```
errStatus = CRTM_Destroy_Atmosphere( Atmosphere )
IF ( errStatus /= SUCCESS ) THEN
  ...
END IF
```

The v2.0.x interface was changed to an elemental subroutine,

```
CALL CRTM_Atmosphere_Destroy( Atmosphere )
IF ( CRTM_Atmosphere_Associated( Atmosphere ) ) THEN
  ...
END IF
```

where, again, the error checking is achieved via the `CRTM_Atmosphere_Associated` function call.

C.2.2 Surface

The Surface structure procedure changes only apply if you utilise the `SensorData` component.

Creation

In v1.2.x the Surface structure allocation was a function returning an error status,

```
errStatus = CRTM_Surface_Allocate( n_Channels, &
                                   Surface     )
IF ( errStatus /= SUCCESS ) THEN
  ...
END IF
```

The v2.0.x interface was changed to an elemental subroutine,

```
CALL CRTM_Surface_Create( Surface,    &
                          n_Channels )
IF ( .NOT. CRTM_Surface_Associated( Surface ) ) THEN
  ...
END IF
```

where the error checking is achieved via the CRTM_Surface_Associated function call.

Destruction

In v1.2.x the Surface structure destruction was a function returning an error status,

```
errStatus = CRTM_Destroy_Surface( Surface )
IF ( errStatus /= SUCCESS ) THEN
  ...
END IF
```

The v2.0.x interface was changed to an elemental subroutine,

```
CALL CRTM_Surface_Destroy( Surface )
IF ( CRTM_Surface_Associated( Surface ) ) THEN
  ...
END IF
```

where, again, the error checking is achieved via the CRTM_Surface_Associated function call.

C.2.3 Options

Creation

In v1.2.x the Options structure allocation was a function returning an error status,

```
errStatus = CRTM_Options_Allocate( n_Channels, &
                                   Options     )
IF ( errStatus /= SUCCESS ) THEN
  ...
END IF
```

The v2.0.x interface was changed to an elemental subroutine,

```
CALL CRTM_Options_Create( Options    , &
                          n_Channels  )
IF ( .NOT. CRTM_Options_Associated( Options ) ) THEN
  ...
END IF
```

where the error checking is achieved via the CRTM_Options_Associated function call.

Destruction

In v1.2.x the Options structure destruction was a function returning an error status,

```
errStatus = CRTM_Destroy_Options( Options )
IF ( errStatus /= SUCCESS ) THEN
  ...
END IF
```

The v2.0.x interface was changed to an elemental subroutine,

```
CALL CRTM_Options_Destroy( Options )
IF ( CRTM_Options_Associated( Options ) ) THEN
  ...
END IF
```

where, again, the error checking is achieved via the CRTM_Options_Associated function call.

C.2.4 RTSolution

Creation

In v1.2.x the RTSolution structure allocation was a function returning an error status,

```
errStatus = CRTM_RTSolution_Allocate( n_Layers   , &
                                      RTSolution   )
IF ( errStatus /= SUCCESS ) THEN
  ...
END IF
```

The v2.0.x interface was changed to an elemental subroutine,

```
CALL CRTM_RTSolution_Create( RTSolution, &
                             n_Layers    )
IF ( .NOT. CRTM_RTSolution_Associated( RTSolution ) ) THEN
  ...
END IF
```

where the error checking is achieved via the CRTM_RTSolution_Associated function call.

Destruction

In v1.2.x the RTSolution structure destruction was a function returning an error status,

```
errStatus = CRTM_Destroy_RTSolution( RTSolution )
IF ( errStatus /= SUCCESS ) THEN
  ...
END IF
```

The v2.0.x interface was changed to an elemental subroutine,

```
CALL CRTM_RTSolution_Destroy( RTSolution )
IF ( CRTM_RTSolution_Associated( RTSolution ) ) THEN
  ...
END IF
```

where, again, the error checking is achieved via the CRTM_RTSolution_Associated function call.

C.3 CRTM Structure Replacement

An additional change was the replacement of the `CRTM_GeometryInfo_type` input structure definition with that of `CRTM_Geometry_type`. This was done to strictly separate the user defined inputs from the derived values determined inside the main CRTM functions.

In v1.2.x the input structure definition would look something like:

```
TYPE(CRTM_GeometryInfo_type) :: geo(N_PROFILES)
```

for a predefined number of atmospheric profiles (via `N_PROFILES`). The v2.0.x definition would be,

```
TYPE(CRTM_Geometry_type) :: geo(N_PROFILES)
```

Users should check that they are assigning values to all the necessary structure components.