But first…

Please consider submitting your manuscripts to the AGU’s modeling journal

JAMES: Journal of Advances in Modeling Earth Systems (james.agu.org)
User interface issues for RRTMGP

“Intra-component coupling”

Robert Pincus, University of Colorado
reflecting valuable input from NCAR, ECWMF, DWD
Can’t we all just get along?

RRTMGP is a component rather than a stand-alone model. Ease of implementation will affect adoption rates and how much people still like us in the morning.

Models have pre-existing infrastructure for input/output, error reporting…

We are writing general code

We seek to share as much infrastructure as possible between RRMTGP (broadband radiative heating rates) and OSS (synthetic observations for data assimilation).

RRTMGP itself is designed for generality e.g. in other climates, on Mars…

We’ve spent time refining the coupling between host models and RRTMGP.
Black boxes for the mysterious bits

The core of RRTMGP is a $k$-distribution: a mapping from the physical state of the atmosphere to “spectrally”-resolved optical properties of the clear, clean sky.

We defined a gas_optics class useable for RRTMGP and OSS. Some procedures are mandatory e.g. `load()`, `gas_optics()`. Others are informative e.g. `get_gases()`

```fortran
  type(ty_gas_optics) :: k_dist
  ....
  !< Initialize k-distribution
  error = k_dist%load('rrtmgp-lw-coefficients.nc')

  !< Use k-distribution
  error = rrtmgp_lw(k_dist, ...)
```
Black boxes for the mysterious bits

Another class describes gas concentration specified as scalars, profiles, or 2-D fields.

```
type(ty_gas_conc) :: gas_concentrations
    ...
    !< Set water volume mixing ratio
    error = gas_concs%set_vmr('h2o', wv_vmr)
```

Minimizes data transfer

Allows looser coupling between host model and RRTMGP (e.g. persistence)

These two classes are gateway code.
Static infrastructure

We isolated code-level coupling of RRTMGP in a single directory

REAL kinds

Physical and mathematical constants

Error reporting (needs revision for parallel environments)

NetCDF I/O to come (needed for parallel environments)

We’re guided by hard experience: e.g. inconsistent physical constants have led to conservation-of-energy errors in the past.

Users will edit these files when they integrate RRTMGP.
Dynamic descriptions

To be flexible RRTMGP relies heavily on Fortran classes. Users can’t escape them (and might hate us in the morning).

Classes describe either the optical or physical properties of clouds and aerosols:

- We provide an optical properties type that can be used directly
- We provide abstract (and example) types for aerosols, clouds, random numbers
  (A carrot: physical types localize memory use by RRTMGP)
- An extensible class describes desired output (spectral and/or spatial reduction)
  Pointers minimize copying. We provide useful examples.
User’s-eye view

! Function interface — optical description
rrtmgp_lw(k_dist, gas_concs, ..., cloud_props,&
    allsky_fluxes, clrsky_fluxes, ..., aer Props)

! derived type with spectral information
type(ty_gas_optics_specification), &
    intent(in ) :: k_dist

! derived type encapsulating gas concentrations
type(ty_gas_desc),
    intent(in ) :: gas_concs

! derived type for optical properties
type(ty_optical_properties), intent(in ) :: cloud_props
type(ty_optical_properties), intent(in ) :: aer_props

! user-provided class for spectral/spatial reduction
class(ty_fluxes),
    intent(inout) :: allsky_fluxes
class(ty_fluxes),
    intent(inout) :: clrsky_fluxes
Class for user-defined flux outputs

```fortran
module user_fluxes

type :: ty_fluxes
    real(wp), dimension(:,,:), pointer :: &
       flux_up => NULL(), &
       flux_dn => NULL()
contains
    procedure :: reduce => reduce_base
    procedure :: are_desired => are_desired_base
end type ty_fluxes

this%are_desired() checks that memory has been allocated for outputs

this%reduce() performs spectral or spatial reduction (e.g. surface PAR)
```

User’s-eye view
User’s-eye view

\begin{verbatim}
<! Function interface — physical description
rrtmgp_lw(k_dist, gas_concs, ...arrays..., &
    clouds, rngs, allsky_fluxes, clrsky_fluxes,... aerosols)

<! derived type with spectral information
type(ty_gas_optics_specification), &
    intent(in ) :: k_dist

<! derived type encapsulating gas concentrations
type(ty_gas_desc),     intent(in ) :: gas_concs

<! user-provided types for clouds and aerosols
type(ty_cloud_desc),   intent(in ) :: clouds
type(ty_aerosol_desc), intent(in ) :: aerosols
class(ty_rng),
    dimension(:,), intent(inout) :: rngs

<! user-provided class for spectral/spatial reduction
class(ty_fluxes),       intent(inout) :: allsky_fluxes
class(ty_fluxes),       intent(inout) :: clrsky_fluxes
\end{verbatim}
type, abstract, public :: ty_cloud_desc
! Abstract class
! Users add pointers or data as needed
contains
  procedure(abstract_cloud_optics), deferred :: cloud_optics
  generic, public :: sample_and_optics => cloud_optics
end type

abstract interface
! Deferred procedure: users must implement to this interface
  subroutine abstract_cloud_optics(this, rngs, spec_cfg, cloud_props)
    class(ty_cloud_desc), intent(in) :: this
    class(ty_rng), dimension(:), intent(inout) :: rngs
    type (ty_gas_optics_specification), intent(in) :: spec_cfg
    class(ty_optical_props), intent(out) :: cloud_props
  end subroutine abstract_cloud_optics
end interface
We’re building a model component that needs rather a lot of information about host model implementation to be efficient and useful.

We’ve wrapped the specifics in Fortran classes. This will be new for most (all?) models.

In the next year we’ll see how much our users like it and/or use it, or don’t.