Proposed Application Program Interface Design for UKCA

John Hemmings, July 2020, v5

with thanks to Luke Abraham for code owner review

Contents

1. Introduction

UKCA is currently an integral part of the Unified Model, making use of many UM-specific procedures internally and communicating with other parts of the UM via a large number of shared modules. In preparation for its use in LFRic, it is in the process of being re-packaged as a standalone code with a well-defined API and will subsequently be moved to its own repository. This will also allow it to be coupled with other models or applications. A prospective parent application may be an atmospheric model (3-D or single column), a test harness or some other testbed system for model analysis.

The API implementation will preserve UM compatibility and the present functionality of UKCA within the UM rather than produce a divergent code base. This is necessary to ensure that future code improvements can benefit UM and non-UM applications alike. This document presents the proposed API design for UKCA and an implementation plan for refactoring UKCA as a standalone sub-model within the UM using this API. The text has been revised since implementation started at UM vn11.4 to include a number of design changes and to reflect the current situation at vn11.7. In addition, the scope has been extended to include GLOMAP-CLIM and RADAER.

GLOMAP-CLIM provides an alternative to the full GLOMAP-mode aerosol sub-model of UKCA that can be used for calculating cloud droplet condensation number based on a prescribed set of GLOMAP aerosol fields and/or for providing RADAER input data based on these fields. GLOMAP-CLIM will be treated as a simple UKCA configuration. It will share UKCA namespace and become part of the standalone code. It should be possible to build GLOMAP-CLIM from a minimal subset of UKCA source files.

RADAER is responsible for calculating aerosol optical property profiles consistent with a set of GLOMAP-mode aerosol profiles. It is called separately from UKCA, via the UM's radiation processing, but is dependent on GLOMAP-mode aerosol data from UKCA or from GLOMAP-CLIM. Like UKCA, it will need to be ported to LFRic and it would ideally be available in other parent applications too, alongside UKCA, for studying the direct radiative effects of UKCA's aerosol fields. UKCA is not currently dependent on RADAER. RADAER will not share UKCA's namespace but will be presented as a separate standalone code with its own API for interfacing with a parent application. It will be able to communicate with UKCA using the UKCA API but it should also be useable without UKCA so that applications calling RADAER will not necessarily need to be built with UKCA source files. It's close association with UKCA makes it sensible to keep it in the same repository as the core UKCA model when the new UKCA repository is set up.

All advection, diffusion and mixing of UKCA tracers will remain the responsibility of the UM or alternative parent model. Convective plume scavenging for UKCA aerosols is currently integrated with UM convection to improve the accuracy of the aerosol removal fluxes (although diagnostics related to plume scavenging are handled within the UKCA time step). The details of the processing required for plume scavenging are convection scheme dependent. For this reason and because the UKCA-specifics are relatively simple, plume scavenging (including the diagnostic handling) will also remain the responsibility of the UM (or alternative parent model) and not be supported within the new UKCA standalone code base.

A set of general design principles to be followed is given in Section 2. Section 3 outlines design considerations specific to UKCA and RADAER and Section 4 gives an outline of the outstanding refactoring work required to implement the new APIs within the UM with a brief overview of changes already made from vn11.4 to vn11.7.

2. General Design Principles

1. UKCA will be available to a parent application via a single API module (`ukca_api_mod` or `glomap_clim_api_mod`) as a minimal set of top-level subroutines. These will include subroutines to perform functions including setting up the UKCA configuration prior to a model run, executing a UKCA time step and doing housekeeping at the end of the model run. A number of other top-level subroutines will provide additional functionality. The API may also include fixed UKCA parameters. The inclusion of UKCA type definitions in the API is strongly discouraged since any parent access to members of such derived types would inhibit independent UKCA development.
2. Names of all subroutines and parameters presented via the API will start with `ukca_` to avoid polluting the parent's namespace.
3. All run-time communication between the parent model and UKCA will be via argument lists.
4. Where UKCA is required to perform processing that is parent model-specific, the processing will be encapsulated in a generic internal subroutine that will reference a parent model subroutine, or handler, conforming to a UKCA specification. The handler will be passed to UKCA via a top-level subroutine call. The alternative of providing parent versions of specific subroutines at compile-time will be supported by ensuring that each generic internal subroutine has its own module that could potentially be replaced in the build process.
5. It must be possible to run UKCA for a given period without doing any file IO during the run. This will be particularly important for future UKCA parameter perturbation/optimization experiments where a large number of integrations will be required with almost identical input data and no file output. For such experiments, the domain and/or time period are typically restricted compared with production cases and it is most efficient to keep all required input data in memory (managed by the parent application) between runs.
6. UKCA will not read any input data directly from external files. All input will be passed from the parent via API subroutine arguments. The parent will be responsible for sourcing these data but UKCA should provide library routines for reading file formats that are UKCA-specific. These library routines will be additional to the API, in that they will not use the UKCA namespace. They will instead pass the data read to the parent as output arguments. Each library routine will provide minimal functionality to allow the parent as much flexibility as possible in how (and from which files) the data are retrieved and handled before passing to UKCA.
7. Any output to external files from within UKCA subroutines will be under the control of the parent model. To achieve this, the parent will pass control data and/or a handler routine to UKCA via an appropriate top-level subroutine.
8. All UKCA state variables (tracers and non-transported prognostics) will be available to the parent model between time steps, as native FORTRAN arrays, for inspection and possible modification. The fields in these arrays, and those in any diagnostics arrays, will be in an order specified by UKCA independently of a particular parent application. Within the arrays, specific fields and their positions may differ between

different UKCA configurations. The actual lists of fields will be determined by field names retrieved by the parent from UKCA at run time.

9. The state variables will be passed to UKCA via the main time step subroutine. Separate top-level subroutines will be provided for initialising/updating other input data such as the physical environment, emissions and other drivers. These data will optionally be allowed to persist between time steps within UKCA until the next update. This will reduce overheads in cases where UKCA is run with a fixed or partially-varying environment. It should also help to avoid excessively long argument lists. However, it does rely on the use of module variables for spatial fields which is not LFRic-compatible since it is not thread-safe for multiple columns running in parallel on a single node. An alternative thread-safe approach will therefore be supported that handles all spatial fields that may vary horizontally in the parent application via a single UKCA API call at each time step.
10. The subroutine for executing a UKCA time step will only process spatial fields that are specified on the UKCA model grid (which is configurable by the parent) and have the appropriate time of validity. Any spatial or temporal interpolation that may be required will be the responsibility of the parent. This will allow a parent maximum flexibility for memory management where multiple instances of the model grid are distributed over processing elements in a parallel environment.
11. UKCA input fields will be expected to span the relevant dimensions of UKCA's spatial domain (as defined in the current configuration) but may extend beyond this (e.g. to allow for halos used by the parent). To allow for parent applications running UKCA in a 1-D or 0-D context, the relevant top-level subroutine arguments will be overloaded to allow use of actual arguments with appropriate dimensions.
12. It must be possible to run UKCA without internal persistence of horizontally varying spatial fields between time steps (as required for LFRic). Whether or not persistence is to be supported internally will be under the control of the parent via UKCA's configuration settings.
13. Whether or not temporary workspace used by UKCA is released between UKCA time steps will likewise be under the control of the parent model via UKCA's configuration settings.
14. The housekeeping subroutine, to be called by the parent in cases where multiple runs are supported, must ensure that all allocatable internal storage is released and any configuration or status information is cleared ready for the next run.
15. On encountering a fatal error condition, UKCA will optionally return control to the parent model with an appropriate UKCA-specific error code and message and the name of the routine where the error was trapped. This option, controlled via UKCA configuration settings, will allow the parent model to either abort or continue without the results of the current UKCA call as required. If this option is not selected or a warning condition arises, the same information will be passed to a parent model handler if one is provided. The parent handler may then output messages and/or abort.
16. The API should allow input data to be validated before starting a model run, at least to the extent that this is practical. This will be helpful when a parent application needs to run large ensembles with varying input data because it will allow the parent to perform input data checks for all ensemble members and trap errors early on before initiating any of the runs.

17. RADAER will be available to a parent model via a separate API module (`ukca_radaer_api_mod`) following the above principles (where applicable). Names will start with `ukca_radaer_`.

3. Elements of the Interfaces for UKCA and RADAER

This section describes the elements of UKCA and RADAER that must be supported by the new APIs (Sections 3.1 and 3.2). For the UKCA API, that is all data currently transferred between the UM or RADAER and UKCA and all UM procedures currently called from UKCA. For the RADAER API, it is all data currently transferred between the UM and RADAER and UM procedures currently called from RADAER. Some specific aspects of the API design are then discussed (Section 3.3) and a provisional list of API subroutines is given (Sections 3.4 and 3.5).

For the purposes of considering UKCA as a distinct entity within the UM, it will be defined by its core functionality as provided by the following subroutines.

- `ukca_setup`
- `ukca_set_environment`
- `ukca_set_emissions_from_nc`
- `ukca_set_oxidants_from_nc`
- `ukca_main1` (called as `ukca_step`)
- `glomap_clim_arg_act_get_cdnc`
- `glomap_clim_jones_act_get_cdnc`
- `prepare_fields_for_radaer`

RADAER will be defined by its core functionality as provided by

- `ukca_radaer_read_luts`
- `ukca_radaer_read_precalc`
- `ukca_radaer_set_aerosol_fields`
- `ukca_radaer_prepare`
- `ukca_radaer_band_average`
- `ukca_radaer_compute_aod`
- `ukca_radaer_3d_diags`

3.1 Input and Output Data

The different categories of input and output data are listed here with their sources/destinations, in the UM context, given in parentheses.

UKCA Input Only Data:

- Parameters, options and other user supplied configuration data (from `run_ukca` and `run_glomap_aeroclim` namelists via UM modules or `ukca_setup` argument list)
- Domain configuration data: field dimensions etc (from UM modules)
- Fast-JX photolysis specification data (from sequential files)

- Environmental driver fields: atmospheric physics and land surface, CLASSIC/aeroclim aerosols for heterogeneous chemistry, geographical location (from UM via `ukca_set_environment` argument list)
- GLOMAP-mode aerosol mixing ratio fields for GLOMAP-CLIM (from NetCDF files via `D1` array or `glomap_clim_jones_act_get_cdnc` argument list)
- Emissions (from NetCDF files)
- Offline oxidants (from NetCDF files)
- Reference data including aerosol surface area density climatology for heterogeneous chemistry, data for 2-D photolysis, Cambridge 2-D model data and ozone ancillary data for top boundary conditions and RCP scenario data for lower boundary conditions of long-lived gases (from sequential files)
- Concentrations of long-lived gases from radiation scheme (from UM via `ukca_set_environment` argument list).
- Requests for diagnostics (from STASH system data)
- Constants (from UM modules)

UKCA Input/Output Data (model state):

- Tracers (from/to UM via `ukca_main1` argument list)
- Non-transported prognostics, including GLOMAP-mode aerosol properties required by RADAER (from/to UM via `ukca_main1` argument list)

UKCA Output Only Data:

- GLOMAP-mode aerosol properties required by RADAER, if using GLOMAP-CLIM (to UM's `ukca_radaer` structure directly from `prepare_fields_for_radaer`)
- GLOMAP-mode configuration data required by RADAER (to UM's `ukca_radaer` structure from UKCA module)
- Diagnostics (to STASH system)
- Log output and error messages (to sequential files)
- Test output optionally produced by UKCA when lower BCs for long-lived gases are read (to sequential file)

RADAER Input Only Data:

- Option `l_ukca_radaer_sustrat` (from `run_ukca` and `run_glomap_aeroclim` namelists via UM modules)
- Domain configuration data: field dimensions etc (from UM via argument lists)
- Radiation scheme spectral configuration data for shortwave/longwave band (from UM's `spectrum` structure via `ukca_radaer_band_average` and `ukca_radaer_compute_aod` argument lists)
- Optical property look-up tables (from files in namelist format)
- Pre-computed variables (from sequential 'Pcalc' file)
- GLOMAP-mode configuration data (from UM's `ukca_radaer` structure via argument lists)
- GLOMAP-mode aerosol mixing ratio profiles (from UM via `ukca_radaer_set_aerosol_fields` argument list)

- GLOMAP-mode aerosol properties profiles: modal diameters and densities and partial volumes of components (from UM via `ukca_radaer_set_aerosol_fields` argument list)
- Other environment data: tropopause level, temperature, pressure (from UM via `ukca_radaer_set_aerosol_fields` and `ukca_radaer_prepare` argument lists)
- Constants (from UM modules)

RADAER Output Only Data:

- Number of aerosol modes (to UM via `ukca_radaer_set_aerosol_fields` argument list)
- Aerosol optical properties for shortwave/longwave band (to UM via `ukca_radaer_band_average` argument list)
- Modal mass mixing ratio fields (to UM via `ukca_radaer_prepare` argument list)
- Diagnostics (to UM via `ukca_radaer_3d_diags` argument list)
- Log output and error messages (to sequential files)

The above lists may not account for all variables that are transferred by direct usage from shared modules. It will be necessary to identify all such variables during the implementation work. The test output functionality relating to lower BCs for long-lived gases is no longer required as part of UKCA and can be removed to a standalone script.

UKCA provides much of the input data required for RADAER but not without some additional handling by the UM. In part, this is because the two models are called for different sub-domains. RADAER performs calculations on a set of vertical profiles at arbitrary horizontal locations. However, UKCA calculations are done on a regular 3-D spatial grid and the input data for RADAER is prepared on the UKCA grid. The main RADAER subroutines may be called multiple times to process different sets of profiles from this grid (and are called separately for shortwave and longwave radiation bands). A translation step is therefore required to extract and rearrange the required data for each RADAER call. In addition, when the data are provided by UKCA itself, rather than GLOMAP-CLIM, they are first written to the UM's `D1` array and not used as input to RADAER until the next time step. This is required to support the UM's restart capability.

The translation between data on the UKCA grid and the input arrays required by RADAER depends on details of the UM and/or SOCRATES configuration and reflects system-dependent memory management considerations. Requirements are likely to differ between applications so this data handling will remain the responsibility of the parent. For applications that do not need different handling for data from UKCA and GLOMAP-CLIM, ensuring that it is possible to get output from either in a consistent form would be beneficial. Also, in reduced dimension applications (e.g. a single column testbed), the overhead of translation might be avoided with careful API design. The UKCA and RADAER APIs should allow for the direct use of UKCA or GLOMAP-CLIM output arrays as RADAER inputs in such use cases but not at the expense of increasing complexity for other cases.

A further consideration regarding the use of the APIs in applications with domains of different dimensions is the need for the API subroutines to be able to handle spatial arrays of the appropriate dimension for each application. For example, it should be possible to call `ukca_step` with a tracer array having 3, 2, 1 or 0 spatial dimensions (2-D calls might be used in the context of stratospheric modelling). Likewise, RADAER might be presented with a single profile for each required variable, rather than an array containing profiles at multiple locations. (Note that such flexibility would be needed for the direct use of UKCA output arrays for RADAER input suggested above.)

3.2 Parent-specific Subroutines Required

At present, the following UM-specific subroutines are called from within both UKCA and RADAER.

- Print management for output to log file: `umprint`
- Error/warning handling: `ereport`
- Dr Hook timer calls: `dr_hook`

In addition, the following are called from within UKCA.

- NetCDF access: subroutines in `emiss_io_mod`
- Sequential file access: `ukca_2d_bc_read_interp`, `ukca_read_aerosol`, `ukca_read_reff`, `ukca_scenario_rcp, read2d_opt` and read subroutines in module `fastjx_specs`
- Boundary layer mixing and addition of emissions to model levels: `tr_mix`, `trsrce`
- Humidity routines: `qsat`, `qsat_wat_mix`, `lsp_qclear`
- Vertical integration of mass in UKCA for plume scavenging diagnostics, using an ENDGAME-specific scheme: `ukca_eg_tracers_total_mass_fix`
- Time interpolation of emissions/offline oxidants: `t_int`
- CLASSIC aerosol surface area and wet radii calculation: `calc_surf_area`
- Marine DMS emissions flux: `dms_flux_4a`

RADAER also calls `read_nml_ukcanml` to obtain the optical property look-up tables.

In many cases, the UM-specific subroutines are required because of the specifics of IO handling in the MPI parallel environment (i.e. reading on PE0 and broadcasting to other PEs). Sequential file access additionally involves calls to the UM file manager (`assign_file_unit` and `release_file_unit`).

All input from external files will remain the responsibility of the UM, or an alternative parent application, as indicated in Section 2 (Item 6). To avoid undue complexity and give the parent maximum flexibility for sourcing these data, all processing that involves file input will be moved outside the UKCA/RADAER subroutines and remain part of the UM code base. Appropriate UKCA library routines (additional to the API) should be provided for the parent to call where file formats are UKCA/RADAER specific. These would be minimal in terms of functionality, giving the parent as much flexibility as possible. In particular, they would not include opening and closing of files. This will allow for the continued use of the UM file manager calls for assigning and releasing file units. It also means that data that are currently read from multiple files could, for example, be read from a single file if required.

Time interpolation will likewise remain the responsibility of the parent. UKCA will expect to receive fields valid for the current time step.

The plume scavenging diagnostics code currently resides within `ukca_main1` and will also need to be moved out so that it remains part of the UM code base. The calculation of surface area and wet radii of CLASSIC aerosols, required for heterogenous chemistry in the RAQ chemistry scheme, could also remain the responsibility of the UM with only the results being passed to UKCA.

The Marine DMS flux routine is not UM specific, being based on simple calculations from the scientific literature and is probably best duplicated for use in UKCA.

Dr Hook timer calls from UKCA will be retained for use with the UM but will be disabled by use of a dummy library when the UM is unavailable. The required dummy library modules can be included in the UKCA code base.

Other UM-specific subroutine calls still in `ukca_main1` or in other UKCA subroutines will require replacement by more generic calls. These will allow indirect access to UM subroutines when used in the UM or to substitutes when used with a different parent. Each generic subroutine will have its own module that could potentially be replaced in the build process, as indicated in Section 2 (Item 4), as an alternative to run-time assignment of a parent handler. The remaining subroutines are `umprint`, `ereport`, `tr_mix`, `trsrce`, `qsat`, `qsat_wat_mix` and `lsp_qclear`.

3.3 Specific Design Aspects

Code structure

Calls to UKCA, GLOMAP-CLIM and RADAER from the parent application should be grouped into as small a number of API subroutines as possible without compromising flexibility. For each, there will be

- a set up subroutine to define details of the configuration that is called once, before the start of the run
- one or more subroutines to provide input data that may or may not change between time steps (including environment data, emissions and diagnostic requests)
- a subroutine to perform the core model calculations at each time step
- if required, a ‘housekeeping’ subroutine to do any final processing needed after the run completes.

UKCA already conforms to this structure (although there is no final subroutine at present as it is not needed for the UM).

For GLOMAP-CLIM, the set-up processing is performed by a specific GLOMAP-mode set up routine `ukca_setup_mode_sussbcoc_5mode` that is internal to UKCA and supports a single GLOMAP configuration. A GLOMAP-CLIM set up routine will be provided in the API to call `ukca_setup_mode_sussbcoc_5mode` and protect the parent from potential internal changes to UKCA. The GLOMAP-CLIM set up routine could then be extended in future to support multiple GLOMAP configurations. The provision of a GLOMAP-CLIM-specific set-up routine separate from that for UKCA is desirable to reduce the number of UKCA modules that need to be built for GLOMAP-CLIM runs.

GLOMAP-CLIM's core functionality is provided by subroutines `glomap_clim_arg_act_get_cdnc`, `glomap_clim_jones_act_get_cdnc` and `prepare_fields_for_radaer`. It should be possible to combine these under a single API subroutine call.

For RADAER, the set-up processing is divided between the routines `ukca_radaer_read_luts` and `ukca_radaer_read_precalc`. Some of this involves reading external files and will not be included within RADAER. Most of the remaining processing will be called via a single RADAER set-up routine but an additional setup subroutine may be required for handling look-up table data (details are given under the heading *RADAER Look-up Table and Pre-calculated Variables* below).

Core RADAER calculations in the UM are divided between subroutines `ukca_radaer_prepare`, `ukca_radaer_band_average`, `ukca_radaer_compute_aod` and `ukca_radaer_3d_diags`. These communicate with each-other via the UM subroutine `set_aer` that is called separately to process shortwave and longwave bands. The 4 subroutines will be combined to simplify the interface and avoid the need for the parent to handle variables that are only relevant to RADAER.

Configuration Data and Fast-JX Specifications

In the UM, UKCA configuration data are read in from a namelist which is updated via the Rose GUI. This functionality is UM-specific and will remain the responsibility of the parent. UKCA obtains configuration data via the `ukca_setup` subroutine call and the equivalent GLOMAP-CLIM subroutine call will receive GLOMAP_CLIM configuration data. Other configuration data such as domain dimensions are defined by the UM or input to the UM via other namelists. These will similarly be passed to UKCA as subroutine arguments.

Different UKCA configurations require different subsets of configuration variables so the values of individual variables are specified by optional arguments to avoid unnecessarily long argument lists. In future developments, the set of configuration variables used in the standalone code may change independently of a particular parent as new functionality is added. Use of keyword arguments in the subroutine call will avoid backward compatibility issues arising from the re-arrangement or addition of variables. Removal of variables should only be considered at major version changes. The API documentation should indicate that positional argument association is not explicitly supported and, if used, may produce unexpected results due to change in the number and/or position of arguments in future UKCA versions.

In addition to the user-defined configuration data, UKCA requires a set of specification data for the Fast-JX photolysis scheme including some species-specific data. In the UM, the specification data are read from sequential files at the first time step, using subroutines in module `fastjx_specs`. The details of the data set are specific to the Fast-JX scheme (part of UKCA) and are not configurable via the GUI. In accordance with Section 2 (Item 6), retrieval of the Fast-JX data from external files will remain the responsibility of the parent and will now be done before setting up UKCA. This means that the species for which UKCA will handle configuration data will be determined by the species present in the input file rather than those present in the chemistry scheme (but UKCA will ensure that data for all required species are present). UKCA will obtain the data via the `ukca_setup` call. The UKCA-specific nature of the data format in the Fast-JX files means that UKCA library routines should be provided for reading in the data.

RADAER configuration data comprise domain data, spectral data and GLOMAP-mode data (as well as some additional data provided by sequential files covered under the heading *RADAER Look-up Tables and Pre-calculated Variables* below). The domain configuration data can be handled in the same way as UKCA data via a RADAER setup subroutine. Spectral configuration data require different treatment since these inputs change according to the spectral band being processed. The required spectral configuration variables will therefore be passed to RADAER at each call to the main calculation subroutine instead of being fixed in the initial setup. GLOMAP-mode configuration data will typically be obtained direct from UKCA via a UKCA API call. This will hide unnecessary complexity from the parent. However, to allow for the possibility of running RADAER with offline data when neither UKCA or GLOMAP-CLIM are available, there will be an option to provide GLOMAP-mode configuration variables via the RADAER setup routine. For this use case, it must be possible to disable UKCA API calls from RADAER using a compiler directive when UKCA modules are not included in the build.

Parent access to configuration data will be provided via a UKCA or RADAER API subroutine call. Configuration variables accessed in this way will give the definitive values that are actually used within the application. These may be defaults where values are not supplied in the set-up call. Internal configuration variables may be made available via the API call as well as those set explicitly from parent supplied values.

RADAER Look-up Tables and Pre-calculated Variables

Some additional data from sequential files are required for setting up RADAER. There is a set of 6 files giving optical properties for different aerosol modal distributions and wavelengths and the 'Pcalc' file that gives pre-calculated values of a number of required variables.

`ukca_radaer_read_luts` calls `ukca_radaer_lut_in` for each file in the set of 6 optical properties files and `ukca_radaer_lut_in` reads the data and saves it to the appropriate part of the look-up table (LUT). Currently the LUT is passed from `ukca_radaer_read_luts`. The LUT is only used within RADAER and should be internal but the file input should remain the responsibility of the parent. The data flow will be re-arranged accordingly, transferring the data to RADAER using an API subroutine. Because the data from each file share a common format, multiple calls to the same subroutine may be preferable to transferring all the data in a single subroutine call since this will avoid unnecessary complexity. This does preclude the use of the main RADAER set-up subroutine for the purpose, so an additional LUT entry set-up routine would be required.

The subroutine `ukca_radaer_read_precalc` reads pre-calculated variables into a structure called `precalc` that is then used within RADAER. Like the LUT data, these data need to be read in by the parent before being transferred to RADAER. RADAER will obtain the data via its main set-up subroutine and will be responsible for validation checks that are currently done during the file read process.

In the case of both LUT data and pre-calculated variables, the file formats are RADAER-specific so RADAER should provide library routines for reading the data.

Tracers and Non-transported Prognostics

Formerly, the UKCA species in the tracer array passed to `ukca_main1` were determined with reference to the STASH system. (Their order matched the order of these fields in the UM STASH master file, with the actual indexing determined by which fields were active.) For UKCA to be developed independently, it must be possible to modify its tracer list in future work without reference to a particular parent model, so this is no longer the case. From UM vn11.5, the tracer species present in the array passed to UKCA are determined directly by UKCA on the basis of its configuration data, once the model has been set up for a particular run. The parent now retrieves a list of field names from UKCA prior to initiating a run and must provide an array of tracer field values matching this list. The same process is followed for non-transported prognostics. In both cases, the specific fields and their positions may differ between different UKCA configurations.

Parent field dimensions may differ from UKCA's internal field dimensions determined by model domain information. For example, there may be halos extending the horizontal dimensions or additional levels in the vertical that are not processed by UKCA (e.g. level 0 required in the UM for ENDGAME). UKCA does not make assumptions about the extent of any of its input fields but does ensure that the data at least span the required domain and that only the required extents are handled internally. This simplifies the internal processing and will avoid overheads associated with redundant regions that could potentially be large.

Some further changes to the handling of tracers and non-transported prognostics would improve the interaction between UKCA and RADAER as described below. These could be implemented at any time without affecting backwards compatibility.

There are two categories of non-transported prognostics: those which are output for use as prognostics in UKCA and those which are output for use as prognostic environmental driver fields in RADAER (i.e. the GLOMAP-mode aerosol properties fields). The latter are not actually required as UKCA inputs. For some applications, they would ideally be separated in the output so that they could be passed to RADAER directly. This is not required for the UM because the UM passes UKCA fields to RADAER via `D1` to support its restart capability and it is more convenient to handle them with other NTPs as a single array. Output of fields separately for RADAER will therefore be provided as an option (possibly controlled by whether optional arguments are supplied that correspond to the specific arrays required by RADAER).

The GLOMAP-mode aerosol mixing ratio fields required by RADAER are the GLOMAP-mode tracers in the UKCA tracer array. Splitting up the tracer array to present GLOMAP-mode tracers directly to RADAER is undesirable because a parent typically needs to handle tracers as a single array. However, additional optional arguments could hold copies of the GLOMAP-mode mixing ratio fields in the required form for RADAER, leaving the tracer array intact.

Environment Data

Environment data refers to input fields, other than the tracers and non-transported prognostics, that are provided on the model grid. These environmental 'drivers' may come from a variety of different sources and different fields might need to be updated at different times in some applications (or not updated at all). The fields can also have different dimensionality and extents. For these reasons, the API does not attempt to handle the environment data as a single array of fields but instead allows each field to be set separately by name. Like the tracers and NTP fields, environment fields passed in by the parent may extend beyond the required domain but only the required extents are handled internally.

For flexibility, UKCA environment data are set by calling a new top-level subroutine `ukca_set_environment` instead of being set in the call to `ukca_main1` and each call sets a single named field. In applications where some or all environment data are fixed, the fixed fields can be set at the beginning of a run and need not be updated. `ukca_main1` now has extra validation that ensures that all required fields are set. UKCA provides the parent with a list of required fields by name on request via an API call (`ukca_get_environment_varlist`). However, it will not fail to run if additional fields are set.

Scalar concentrations of long-lived gases are treated as a special case of environment data and are updatable by name in the same way.

At present, GLOMAP-CLIM gets its environment data directly from the UM's `D1` array or via the argument list to `glomap_clim_jones_act_get_cdnc`. These data comprise the GLOMAP aerosol mixing ratio fields that are handled as tracers in UKCA together with some physical fields. In the standalone code, GLOMAP-CLIM will handle environment data in the same way as UKCA, taking advantage of the existing API routines. This will require GLOMAP-CLIM to include checks like those at the beginning of `ukca_main1` to ensure that all required fields are available for the selected configuration. Advantages are that it will improve consistency with other UKCA configurations and allow more flexibility for individual GLOMAP aerosol fields to be updated as and when required (i.e. not necessarily at every time step). The capability for a parent to update fields individually will also make it easier to add different GLOMAP-mode configurations in future.

RADAER's environment data comprise the GLOMAP aerosol mixing ratio profiles, profiles of properties derived from the aerosol mixing ratio fields either by UKCA or by GLOMAP-CLIM and physical profiles. The aerosol profiles are handled as a set of mode-related variables (with mode index as an additional dimension) and a set of component-related variables (with component index as an additional dimension). If GLOMAP-CLIM is used to provide the

RADAER aerosol data, it will provide a set of output fields that are consistent with this but may have different spatial dimensions depending on configuration details. If UKCA is used, then the same set of RADAER aerosol arrays should be available as an option (as indicated in *Tracers and Non-transported Prognostics* above), although this will not be used by the UM.

Like UKCA and GLOMAP-CLIM, RADAER will obtain its environment data via a parent call to a separate API routine from the one that does the core calculations. This will mean that the data will typically only need to be passed once for use in both shortwave and longwave calculations. The list of environment profiles required by RADAER has some limited configuration dependency (tropopause is only needed if `l_ukca_radaer_sustrat` is true). An equivalent of `ukca_get_environment_varlist` will therefore be needed to provide the parent with the details. The specific profiles included for each mode and component are also configuration-dependent and the parent may need to use mode configuration data obtained from UKCA via an API call to determine how to compose the required arrays. This will be necessary in the UM when the data are extracted from `D1` but will not be necessary when using GLOMAP-CLIM or in applications where UKCA already outputs arrays in the required form.

The equivalent of `ukca_set_environment` for RADAER will be based on `ukca_radaer_set_aerosol_fields`. This will not do any sub-selection of profiles currently done therein: instead the parent will be expected to pass only the specific profiles that are to be processed in the subsequent RADAER call. The case for updating fields individually as done in UKCA does not apply to the aerosol fields because they are expected to be a consistent set. They could therefore be updated together in a single API call. However, there may be cases where physical environment profiles require updating while the aerosol fields do not, or vice versa. This, together with the configuration dependency of the physical fields, may be best handled by updating fields individually by name as in UKCA. A common UKCA-style approach is therefore recommended for both physics and aerosol fields. The resulting consistency throughout the code base should make the code clearer and easier to maintain and the API easier to understand. RADAER will need a validation step like that in UKCA to ensure that all required environment fields are set before use.

Reference Data Sets

Reference data, as referred to here, are distinct from environment data because they are defined on an arbitrary geographic grid, or sometimes throughout the model domain, and generally contain instances associated with multiple times. In accordance with Section 2 (Item 10), reference data defined on an arbitrary grid will not be processed directly by UKCA. Instead, they will need to be converted to fields on the model grid and interpolated to the current time by the parent. The resulting fields can then be treated as environment fields and passed to UKCA using `ukca_set_environment`.

Reference data that may be required, depending on the configuration, include top boundary condition data (currently read in `ukca_2d_bc_read_interp`) and aerosol climatology data (currently read in `ukca_read_aerosol` and `ukca_read_reff`) and yearly time series for long-lived gases read from an RCP data file (currently read in `ukca_scenario_rcp`). If the

2-D photolysis scheme is selected, reference data will also be required for species photolysis rates. These fields are currently read from external files and interpolated to the required grid and time within `ukca_main1` so that functionality must be moved outside the UKCA call and will remain part of the UM code base. If similar functionality is later needed by non-UM applications for running UKCA, it could potentially be supported by UKCA library routines additional to the API but these would not have access to UKCA's namespace (see Section 2, Item 6). For the long-lived gases, there is currently an option to write test output to a file after reading data from an RCP file. This functionality is best provided by a standalone script and will be removed from UKCA.

Emissions and Offline Oxidants

Currently, emissions and offline oxidants are read from NetCDF files and time-interpolated within the subroutines `ukca_set_emissions_from_nc` and `ukca_set_oxidants_from_nc`. As indicated in Section 3.2, all file input is to be moved outside UKCA and will become the responsibility of the parent. The parent will then obtain the set of fields from external files (or some other source) and pass individual fields to UKCA in a similar way to the standard environment fields. Fields passed to UKCA will be expected to map directly to the UKCA grid and be valid for the time step or time steps at which they will be applied. The two existing routines will be divided up to separate the UM and UKCA processing accordingly.

Emissions differ from standard environment fields in that the number of emission fields is determined by the parent. In addition, the fields have associated attributes. However, the number of offline oxidants is determined by UKCA and, although the offline oxidant fields do have associated attributes these attributes should not be needed for the processing that will remain the responsibility of UKCA. It should therefore be possible to handle the offline oxidant fields as standard environment fields passed to UKCA via calls to `ukca_set_environment`.

For emissions, a key attribute `tracer_name` gives the name of the emission that indicates which tracer it will be emitted into. There can be multiple fields with the same `tracer_name` value. The valid values of `tracer_name` depend on the UKCA configuration and are determined during the call to `ukca_setup`.

To determine which fields to pass to UKCA, the parent needs to obtain a list of the expected emission fields via an API subroutine call and select any matching fields from those that may be available. It must then indicate the total number of fields to be supplied before UKCA can allocate its internal emissions structure comprising a derived type array with one or more array entries for each field (aerosol emissions have multiple entries to represent emissions into different modes). Once the structure is allocated, a parent will be able to add emission fields individually via another API call. This will register the fields with UKCA and provide their attributes. A further API subroutine will be called to pass the actual field values for initialisation or update during the run.

Each field registered will be identified by an emission number. A parent will need to maintain a record that relates this number to its source. In the UM context, this will be a NetCDF filename and variable name. UKCA will allow a parent-supplied identity label to be held against each emission for use in any error messages generated. This might for example be a concatenation of the file and variable names.

Each time series of emission fields in UKCA can have additional spatial data associated with it in the form of time-invariant, level-dependent scaling factors. These are 3D fields used to spread 2D emissions over multiple levels. Their calculation is based on emissions field attributes (specifying the type of profile) and the model's horizontally varying level thickness. In the UM, the calculation is only required once for each field. It is done within `ukca_main1` at the first time step and the data persist throughout the run. However, as indicated in Section 2 (Item 12), it must be possible to run UKCA without any internal persistence of spatial fields. An alternative option is therefore required so that the 3D factors can either be re-calculated at each time step or, ideally, handled by the parent and passed back into UKCA at each time step with the corresponding emission field values. The latter solution requires the 3D factors to be obtained from UKCA after their initial calculation. The 3D scaling factor for each emission can be returned to the parent via the API call that initialises the emission field values.

Diagnostics

The parent needs to be able to check availability of diagnostics as determined by the UKCA configuration data, allowing pre-run validation or filtering of diagnostic requests. It also needs to pass diagnostic requests to UKCA that may vary between time steps. UKCA will use these requests to determine which diagnostics are to be included in the output. Diagnostic output can be 2-D or 3-D (equivalent to 0-D or 1-D in a single column model) so two separate arrays of diagnostic fields will be used. These will be referred to as 'flat' diagnostics and 'full-height' diagnostics.

The diagnostic requests will be updated separately from the time step and only diagnostics with active requests will be output at each time step. When the diagnostics required vary between time steps it may or may not be desirable for the length and indexing of the diagnostic output arrays to also vary. Retaining fixed indexing throughout a run may be convenient but would require space to be allocated in the arrays for all available diagnostics (possibly a much larger number than those actually used) or for the parent model to be forced to indicate all diagnostics that will be used in advance. The scheme proposed below will allow the parent maximum flexibility to control whether, and over what period, the indexing is fixed or variable.

The parent will set or update diagnostic requests by passing two 1-D arrays of field names, one for 'flat' diagnostics and one for 'full height' diagnostics. (Where names of 'full height' diagnostics are included in the 'flat' diagnostics list they will be taken to indicate surface level fields). The field name arrays will be accompanied by corresponding arrays of status flags to indicate whether each request is active. The status flags can be set 'on' or 'off' by the parent but will be set 'off' by UKCA on return for diagnostics that are unavailable. The output diagnostic fields will match the lengths and indexing of the field name arrays but only the fields corresponding to active requests will be valid. (Others should be set to NaN for safety.)

The status flags will be integers rather than logicals. This will allow further codes to be used during the run to indicate either that an output diagnostic field has indeed been updated since the last request or that an error has occurred and that the field is invalid. Note that all requests that remain active after UKCA's availability check against the configuration data should be valid in theory but the use of an explicit code that is set at the point of update provides additional confidence.

Parent-specific Subroutines

Where parent-specific subroutine calls are required, generic UKCA internal subroutines will be called that reference parent model subroutines via pointers declared in UKCA/RADAER modules. The appropriate parent model subroutines will be provided by calling a new subroutine `ukca_set_handler`, passing as arguments the subroutine and a 'handler type' label against which it is to be registered. The label will indicate its intended use. Where appropriate, default alternative processing will be performed by the generic subroutine if no parent handler is supplied. The parent's subroutine interface will need to conform to a UKCA specification as defined by the new API. In some cases, this may mean that existing routines need to be called via a parent-provided wrapper routine.

Error Handling

On encountering a fatal error condition UKCA (or RADAER) should be configurable to self-terminate without terminating the parent model (since the parent may want to continue without running the offending configuration). Instead, the procedure name, a UKCA-specific error code and an error message will be passed back to the parent in such configurations.

On encountering the error or warning condition, a procedure will call a UKCA error handler (replacing calls to the UM `ereport` routine). In the case of a fatal error, the UKCA error handler will determine whether to abort or return control to the calling procedure depending on the configuration. If aborting, it will either pass the error information to a parent-model error handler if one is supplied or it will write the error information to the standard error stream itself.

On encountering a warning, the UKCA error handler will either send warning information to a parent-model error handler if one is supplied or it will write the warning information to the standard error stream itself. It will then return control to the calling procedure.

After calling the UKCA error handler, the procedure trapping the error will in the case of fatal errors (not warnings) return control to the calling procedure after setting output arguments provided for the error information and calling `dr_hook` if applicable. Each call to a procedure that can potentially trap an error must be then be followed by a status check to determine whether the calling procedure can continue.

In the UM, UKCA will be configured to always call `ereport` via the UKCA handler on encountering an error or warning. However, it is good defensive programming practice for `ereport` calls to be included after UKCA API calls that may return an error.

Increasing the amount of error checking available to the parent application prior to starting UKCA integration would be desirable as indicated in Section 2 (Item 16) to support prior validation of input for ensemble runs but is not a high priority.

Thread-safe Build Option

The update of environment fields separately from the API call that executes the main time step (as described above) will be advantageous for some parent applications but is not compatible with LFRic due to its reliance on the use of public module variables, accessible throughout UKCA, for spatial fields. The issue arises because LFRic will typically use separate threads, each with its own UKCA time-step call, to process different vertical columns on the same node of a parallel architecture. This means that fields that may vary horizontally in LFRic cannot safely be shared between UKCA subroutines as module variables. They must only be accessed via argument lists to avoid memory conflicts between separate instances of the time-step call. The relevant column of each field will be passed to `ukca_step` at each call.

An LFRic-compatible, thread-safe build will be supported by

- Providing an alternative top-level subroutine in the API, to be referred to as the UKCA step control routine, that encapsulates the environment field updates (including emissions) and the `ukca_step` call as UKCA internal calls. This will take each environment field from the parent as a separate array argument.
- Transferring environment fields to `ukca_step` via the argument list or via USE statements depending on the build. This will be controlled by pre-processor directives.
- Ensuring that all spatial fields that may vary horizontally in the parent application are accessed internally via subroutine calls and not via USE statements, irrespective of the build directives. This can be enforced by making the relevant module variables private in “thread-safe” builds.
- Providing a mechanism to ensure that OpenMP directives within UKCA code are suppressed in “thread-safe” builds (by pre-processing the code or use of compiler options). These directives could otherwise interfere with OpenMP directives introduced by LFRic at a higher level in the calling chain.

In principle, this approach could be used to call multiple instances of UKCA in parallel with each instance processing 3-D data. However, there is no requirement for this (and none envisaged) so, to avoid unnecessary complexity in the subroutine interface, it will be assumed that all data passed to UKCA via the alternative top-level subroutine will be in reduced dimension form as appropriate for a single column model.

For ease of handling, the environment fields can be packaged as a derived type for internal use. In the thread-safe API call, they will be passed as separate arguments to avoid the need for the parent to use a UKCA derived type (see Section 2, point 1). Since the specific environment variables that are required depend on details of the UKCA configuration, it will be necessary to pass dummy arguments for unused fields. These can be kept to a minimum by use of optional keyword arguments and perhaps to some extent by using different calls for different configurations but is probably not practical to eliminate in applications that support many different configurations. The use of keyword arguments will also make it easier to change the argument list while maintaining backwards compatibility. The API documentation should indicate that positional argument association is not explicitly supported and may produce un results on upgrading to new UKCA versions (as for the configuration variables).

Emission fields will need to be supported in a similar way to the standard environment fields but will need special handling as the specific emissions are not known in advance. They can be grouped into 2 arrays, for 0D and 1D emissions with 2 corresponding arrays providing the identifying emission numbers. A further array will be needed for 1D scaling factors to be applied to 0D emission fields.

The general approach described here can be extended to GLOMAP-CLIM and RADAER, since each will have environment field handling consistent with UKCA and a similar calling sequence, `glomap_clim_calc` or `ukca_radaer_calc` calls being used in place of `ukca_step`.

Persistence of Fields Between Time Steps

Whether or not horizontally-varying spatial fields are allowed to persist between time steps must be under the control of the parent via the configuration settings as specified in Section 2 (Item 12). This is important for compatibility with LFRic since memory for spatial fields will be re-used by multiple calls to UKCA for different columns within a single time step. In other applications, it will be more efficient to retain certain field values between time steps as is done currently in the UM context.

The option to suppress persistence between time steps will apply to all fields that might vary horizontally in the parent model domain. It will not apply to any spatial fields that are treated as horizontally uniform. The values of the latter can be re-used safely for all UKCA column calls in LFRic.

For maximum flexibility, other non-trivial amounts of workspace allocated internally should similarly be under parent control (Section2, Item13). The temporary workspace allocated internally should be reviewed and deallocation statements added as appropriate if not already present. These and existing deallocation statements for such workspace will be put under the control of the parent via the configuration settings.

Field Names

Names used in the UKCA and RADAER APIs will include literals for prognostic and diagnostic fields and keywords for optional configuration variable arguments. These names should be given careful consideration because they will be difficult to change without causing loss of backwards compatibility in the event that the UKCA standalone code is used in many different parent applications.

In general, configuration variable names will be the same as the corresponding internal variable names unless a change is justified for clarity. For example, names for Fast-JX specification variables would have a '`fastjx_`' prefix added. A similar approach for field name literals will be taken initially but a review of these names, with community buy-in, is highly recommended before the standalone code is widely adopted in new applications.

The set of field names will be specific to the UKCA interface and the names therein need not necessarily match names used to refer to the same fields elsewhere in a parent model. This is important because the requirements of parent applications will vary (e.g. the UM identifies fields by STASH codes and other parents may use short field names for their user interfaces). In particular, it is unnecessary to consider potential name conflicts with similar fields (e.g. same quantity from a different source) that a parent may handle outside the context of its communication with UKCA. Handling such conflicts would be the responsibility of the parent.

Internally, field names literals will be defined by parameters. These should then be used throughout the UKCA code base to avoid the need for extensive changes if there is a need to change the names used externally.

3.4 Provisional UKCA API Subroutines

A provisional list of API subroutines is given here with brief descriptions. Subroutines used to set up the UKCA configuration are listed first, followed by subroutines that provide the parent with information about this configuration. The following subroutines then relate to setting up the emissions structure, setting up and updating diagnostic requests, providing information about these requests, setting or updating environment and other data required in the UKCA time steps, doing the time step processing and finally resetting UKCA to its un-initialised state.

A relatively large number of subroutines are to be provided with the aim of allowing the parent maximum flexibility. This does mean that executing a time step in the UM (and similar applications) will require multiple subroutine calls but these can easily be encapsulated within a parent-specific wrapper. If it is felt necessary to be able to hide this complexity from a parent in future, it would be possible to provide one or more UKCA-specific wrapper subroutines in the API at a later date to cover typical use cases without compromising backwards compatibility.

`ukca_set_handler`

Provides UKCA with a parent-specific subroutine to perform a particular function. The possible functions are listed in Section 3.2.

`ukca_setup`

Sets the user configuration variables, checks their validity and sets up everything required to establish full details of the configuration. This will determine which tracers and NTPs are used, which diagnostics are available and which emissions data and environmental input fields are required. Also copies Fast-JX specification data into UKCA and sets up or copies other data that will be fixed for the duration of the UKCA run but are currently used directly from UM modules (e.g. model grid information). In applications supporting ensemble runs, it may be called multiple times to validate different sets of input data and provide information to the parent about potential UKCA runs prior to any being executed.

`glomap_clim_setup`

Performs a limited setup (of a specific GLOMAP-mode configuration) required for using GLOMAP-CLIM.

`ukca_get_tracer_varlist`

Returns the list of field names of active tracers in the current configuration.

`ukca_get_ntp_varlist`

Returns the list of field names of active non-transported prognostics in the current configuration.

`ukca_get_environment_varlist`

Returns the list of field names for environment data required in the current configuration.

`ukca_get_emission_varlist`

Returns the list of names of active emissions in the current configuration with a corresponding array indicating the number of emission structure entries required for each field.

`ukca_get_config`

Returns other specific information about the current configuration. The information to be returned is controlled by which optional arguments are supplied.

`ukca_register_emission`

Adds a new emission entry holding the field attributes into UKCA's emissions structure and returns the emission number for identification. On the first call, a count giving the total number of required emissions entries will be mandatory for setting the size of the emissions structure.

`ukca_set_flat_diagnostic_requests`

Set up a new list of 'flat' diagnostic requests and the associated status flags. Includes availability check for each requested diagnostic.

`ukca_set_full_ht_diagnostic_requests`

Set up a new list of 'full height' diagnostic requests and the associated status flags. Includes availability check for each requested diagnostic.

`ukca_update_flat_diagnostic_requests`

Update one or more status flags associated with an existing list of ‘flat’ diagnostic requests. Includes availability check for any diagnostics to be activated.

`ukca_update_full_ht_diagnostic_requests`

Update one or more status flags associated with an existing list of ‘full height’ diagnostic requests. Includes availability check for any diagnostics to be activated.

`ukca_get_flat_diagnostic_varlist`

Returns the current list of diagnostic requests in the form of the list of field names corresponding to the ‘flat’ diagnostic output array and the associated status flags.

`ukca_get_full_ht_diagnostic_varlist`

Returns the current list of diagnostic requests in the form of the list of field names corresponding to the ‘full height’ diagnostic output array and the associated status flags.

`ukca_set_environment`

Sets or updates a named environmental input field. This is normally a 0, 1, 2 or 3D field that maps directly to the model grid and may be varied by the parent during the run. Will typically be called at each time step if that is the case. Will optionally return a copy of the updated field to support internal use by a UKCA or GLOMAP-CLIM control routine in thread-safe builds where environment module variables are private. (The interface for this routine, or possibly a separate API routine acting as a wrapper, will need to support the option of calling with reduced dimension arguments for 0D, 1D or 2D domains and convert to higher dimension arrays used internally if necessary.)

`ukca_set_emission`

Sets or updates a numbered emission field, These are 2 or 3D fields mapping directly to the model grid that may be varied by the parent during the run. Will typically be called at each time step if that is the case. On first setting a particular field, also calculates a 3D scaling factor for the emission if needed (based on the `vertical_scaling` attribute). If an optional 3D scaling factor argument is provided, returns the scaling field to the parent on first setting an emission or accepts a scaling field from the parent on update calls. Will also optionally return a copy of the updated emission field to support internal use by a UKCA step control routine in thread-safe builds where environment module variables are private. (The interface for this routine, or possibly a separate API routine acting as a wrapper, will need to support the option of calling with reduced dimension arguments for 0D, 1D or 2D domains and convert to higher dimension arrays used internally if necessary.)

`ukca_step`

Performs one UKCA time step by calling `ukca_main1`. (Will need to support the option of calling with reduced dimension arguments for 0D, 1D or 2D domains and convert to the higher dimension arrays expected by `ukca_main1` if necessary.)

`ukca_step_ctl`

Performs one thread-safe UKCA time step for a single vertical column with all spatial data potentially varying between different column calls being supplied via the argument list. Includes internal calls to `ukca_set_environment`, `ukca_set_emissions` and `ukca_step`.

`glomap_clim_calc`

Performs GLOMAP-CLIM calculations required for RADAER and/or for providing CDNC.

`glomap_clim_calc_ctl`

Performs thread-safe GLOMAP-CLIM calculations (as above) for a single vertical column with all spatial data potentially varying between different column calls being supplied via the argument list. Includes internal calls to `ukca_set_environment` and `glomap_clim_calc`.

`ukca_reset`

Resets UKCA to its un-initialised state to allow another configuration to be setup. (A separate `glomap_clim_reset` may be required to keep the build small for GLOMAP-CLIM, depending on the number of modules involved.)

3.5 Provisional RADAER API Subroutines

`ukca_radaer_setup`

Sets up the RADAER pre-calculated variables and domain configuration data. Also sets up the GLOMAP-mode configuration data via an API call to `ukca_get_config` or from the argument list if UKCA is unavailable and checks the validity of the resulting RADAER configuration. In applications supporting ensemble runs, it may be called multiple times to validate different sets of input data and provide information to the parent about potential UKCA runs prior to any being executed.

`ukca_radaer_get_environment_varlist`

Returns the list of field names for environment data (aerosol and physics data) required in the current configuration.

`ukca_radaer_get_config`

Returns other specific information about the current configuration. The information to be returned is controlled by which optional arguments are supplied.

`ukca_radaer_set_lut_entry`

Sets up an entry in the RADAER look-up table for a particular aerosol mode distribution and waveband.

`ukca_radaer_set_environment`

Sets or updates a named environmental input profile array. These are arrays of vertical profiles or single values (in the case of tropopause level) at one or more locations that may be varied by the parent during the run. Will typically be called at each time step if that is the case. Aerosol arrays will have an additional dimension for modes or components. Will optionally return a copy of the updated environment profile array to support internal use by a UKCA step control routine in thread-safe builds where environment module variables are private. (The interface for this routine, or possibly a separate API routine acting as a wrapper, will need to support the option of calling with reduced dimension arguments for 0D or 1D domains and convert to higher dimension arrays used internally if necessary.)

`ukca_radaer_calc`

Performs RADAER calculations for a given set of spectral data to provide aerosol optical properties and modal mixing ratios for parent model's radiation scheme.

`ukca_radaer_calc_ctl`

Performs thread-safe RADAER calculations (as above) for a single vertical column with all spatial data potentially varying between different column calls being supplied via the argument list. Includes internal calls to `ukca_radaer_set_environment` and `ukca_radaer_calc`.

`ukca_radaer_reset`

Resets RADAER to its un-initialised state to allow another configuration to be setup.

4. Implementation

The UKCA code is being refactored within the UM to conform to the new API design while at the same time maintaining the existing functionality. The refactored code should not change results and is required to pass all group UKCA and GLOMAP-CLIM rose stem tests. This section aims to identify the scope of the development work. Changes will also be required to LFRic for affected components that have already been ported. At present, this only applies to a partial port of GLOMAP-CLIM.

In the refactored code, the present UKCA modules (and any new modules created) will either become UKCA modules that are independent of the UM or UKCA-specific UM modules that prepare data for calling UKCA routines, perform the calls and handle the output. These UM modules will be considered to be external to UKCA and will not use UKCA's namespace directly. In the context of the interface, they will be referred to as UM-side modules and will communicate with UKCA by calling the top-level UKCA procedures provided by UKCA-side modules. These UKCA procedures will be made available via `USE` statements in an API module `ukca_api_mod`.

GLOMAP_CLIM will share the UKCA namespace but have a separate API module `glomap_clim_api_mod`. This will allow applications using GLOMAP-CLIM as an alternative to UKCA to be built with a much smaller set of UKCA source modules than applications using the full feature UKCA model.

RADAER will have a separate namespace from UKCA itself and a separate API module `ukca_radaer_api_mod`, to be used by the UM for access to top-level RADAER procedures. It will communicate with UKCA by using the UKCA API.

A number of modules currently held in `UKCA` and `GLOMAP_CLIM` directories in the UM repository contain UM-specific code that handles UKCA-related data but will not be directly involved in the interface. These will also be considered as external modules with respect to UKCA. A provisional categorisation of the modules from these directories (UKCA, RADAER or external) is given in the Appendix.

All communication between the UM and UKCA (or RADAER) modules should eventually be via procedures accessible via the API modules. One-way communication from UKCA (or RADAER) to the UM via use of internal modules might still be allowed in the short term as it will not prevent the standalone code from working. However, this is highly undesirable in the longer term because it makes the UM vulnerable to UKCA (or RADAER) changes that do not affect the defined API, making it impossible to maintain the UKCA code base independently.

The UM will continue to use the STASH system for UKCA sections but it should be possible to add or remove diagnostics, prognostics or even complete chemistry schemes in UKCA without necessarily needing to change UM code (including the STASH master file). UM code could of course be updated independently to take advantage of new UKCA fields or schemes. Similarly, it could be updated to remove support for any obsolete fields or schemes, and ideally should be, but it must at least allow for the fact that the presence of support for UKCA items in STASH can no longer be considered a robust indicator of their availability. It will be possible for the UM to check availability by enquiring of UKCA directly rather than using the STASH system (and fail cleanly if a missing item is requested) but this can only be done after calling `ukca_setup`. A different solution may be required with respect to the validation of data in the Rose GUI that is done a priori (See Section 4.9).

4.1 Overview of Work Completed

As of UM vn11.7, the following UKCA API tickets (trac system keyword: `UKCA_interface`) have been completed.

- #4367 Initial re-factoring of UM-UKCA interface
- #4819 Tracer handling for UKCA API
- #4822 Environment field handling for UKCA API
- #4876 Emissions and oxidants data handling for new UKCA API
- #5119 Latitude bugs affecting LAM configurations
- #4948 Handling of configuration data for UKCA API

Brief details are given here. See the Ticket Details pages on the UM trac system for full details.

Note that the new error handling scheme, as described in Version 4 of this document, is used for handling fatal errors in new modules and in other modules where there have been significant changes. This means that there is currently a mix of old error handling, where `ereport` is called to terminate the run immediately, and new error handling, where control is returned to the UM before `ereport` is called. In the present version of the design, the error handling (described in Section 3.3) has changed to include a configurable option to abort at the point where the error is trapped (to simplify traceback). This is the preferred option in the UM.

Initial Re-factoring of UM-UKCA Interface (#4367)

This ticket achieved the following aims.

1. Move `D1` access outside of UKCA (as this is UM-specific)
2. Move STASH handling outside of UKCA (as this is UM-specific)
3. Separate UM-specific and UKCA code currently in `ukca_setd1defs`
4. Separate UM-specific and UKCA processing of non-transported prognostics
5. Ensure early availability of NTP requirements to prepare for the verification of NTP output requests using UKCA data for reference rather than STASH

The UM call to `ukca_main1` was replaced by a call to a UM wrapper subroutine `atmos_ukca` that does the necessary UM-specific preparation and post-processing for calling the modified `ukca_main1` via the API subroutine `ukca_step`. (`ukca_step` is currently just a pseudonym for `ukca_main1` but will ultimately act as a wrapper calling `ukca_main1` as a separate subroutine to support the option of reduced dimension arguments for 1-D or 0-D domains.)

Two other UKCA API subroutines were introduced: `ukca_setup` and `ukca_get_ntp_varlist`. Much of the initialisation code that was formerly done at the first UKCA time step is now done within `ukca_setup` before the start of the model run.

Tracer Handling for UKCA API (#4819)

This ticket achieved the following aims.

1. Add a new tracer initialisation routine to be called during initial UKCA setup that will define the list of tracers required for the present configuration.
2. Add a new subroutine `ukca_get_tracer_varlist` to retrieve the list of tracers by name that are required for the present UKCA configuration.
3. Use this list of tracer names for the mapping between UM and UKCA tracer fields.
4. Move copying of tracers between UM and UKCA arrays outside `ukca_main1` so that it becomes the responsibility of the UM.

Environment Field Handling for UKCA API (#4822)

This ticket achieved the following aims.

1. Add a new API subroutine `ukca_get_environment_varlist` to retrieve the list of environment fields required for the present UKCA configuration.
2. Add a new API subroutine `ukca_set_environment` to allocate and set internal UKCA environment fields by name.
3. Change UM-side subroutine `ukca_setd1defs` to determine whether `D1` fields are required using the list returned by `ukca_get_environment_varlist`. Make equivalent changes to `ukca_set_trace_gas_mixratio` to determine which gas mixing ratio values are required.
4. Call `ukca_set_environment` for each field extracted from `D1` to update UKCA. Also call this subroutine for each of the required gas mixing ratio values.
5. Ensure all required internal environment fields are present for the UKCA time step.
6. Deallocate/reset the internal environment fields at the end of the UKCA time step.

Emissions and Oxidants Data Handling for New UKCA API (#4876)

This ticket achieved the following aims.

1. Move the emissions processing concerned with reading and interpolating data from NetCDF files out of `ukca_main1` into a new API subroutine `ukca_set_emissions_from_nc`.

2. Move the offline oxidants processing concerned with reading and interpolating data from NetCDF files out of `ukca_main1` into a new API subroutine `ukca_set_oxidants_from_nc`.
3. Call the new subroutines from the UM before `ukca_step` to setup/update the emissions and offline oxidants data as required.

Note that these subroutines were designated as API subroutines in Version 4 of this design document. However, owing to design changes that shift more of the responsibility for file IO to the parent, they will be replaced in the final API with part of their processing being retained in the UM.

Latitude Bugs Affecting LAM Configurations (#5119)

This ticket was raised as a bug fix but also involved a general rationalisation of the handling of latitude and longitude-related fields throughout UKCA. The ticket achieved the following aims.

1. Fix bugs affecting LAMs due to the use of incorrect UM location variables `sin_theta_latitude`, `fv_cos_theta_latitude` and `tan_theta_latitude`. These variables relate to grid latitude rather than true latitude but are, in most cases, interpreted as relating to true latitude within UKCA.
2. Fix bug affecting LAMs due to the use of the variable `f3u_by_omega` (based on UM variable `f3_at_u`) as a proxy for the true location variable `sin_latitude`.
3. Avoid the direct use of the UM module variables relating to true latitude and longitude within core UKCA modules. These variables should instead be passed from the UM to UKCA as arguments via the new UKCA API. An exception is made for the UKCA module `ukca_volcanic_so2` since this includes UM-specific spatial grid processing that will need to be moved out of UKCA in a future ticket.
4. Avoid the need for the values of the true location variables to persist within UKCA between time steps (for LFRic compatibility).
5. Avoid unnecessary recalculation of the true location variable values at each time step.

Following the changes in this ticket, true latitude and longitude and all trig. functions related to true latitude are passed to UKCA via the new API by calling `ukca_set_environment`.

Handling of configuration data for UKCA API (#4948)

This ticket achieved the following aims.

1. Pass configuration data from `ukca_option_mod` via the `ukca_setup` argument list, avoiding direct use of the UM module in UKCA.
2. Do the same for some additional configuration variables, related to environmental drivers, that are already handled by `ukca_setup`.
3. Manage configuration data in UKCA using well-organised data structures in the core configuration data module `ukca_config_specification_mod`.

4. Add new API subroutine `ukca_get_config` to return internal values of configuration variables.
5. Perform post-setup checks in the UM to ensure that the UKCA configuration matches requirements.
6. Move UM-specific validation of configuration variables out of UKCA modules.

4.2 Overview of Further Work Required

The UM-side subroutines in the UM vn11.7 code that will be affected by the re-factoring are shown in the tree below, highlighted in plain bold text, with their UM calling chains. The routines that are part of the current UKCA API are likewise shown in bold italics.

```
um_shell
      atmos_ukca_setup
            ukca_setup
            check_ukca_configuration
                  ukca_get_config
            ukca_get_config
      stash_proc
            prelim
                  tstmsk
                        tstmsk_ukca  (for STASH requests)
            addres
                  primary
                        tstmsk
                              tstmsk_ukca  (for tracers)
                  ukca_set_nmspec
      u_model_4a
            initial_4a
                  initphys
                        ukca_radaer_read_luts
                              ukca_radaer_lut_in
                        ukca_radaer_read_precalc
            atm_step_4a
                  ukca_mode_sussbcoc_5mode  (for GLOMAP_CLIM with RADAER)
                  ukca_radaer_init
                  glomap_clim_radaer_get
                        prepare_fields_for_radaer
                  allocate_ukca_cdnc
                        ukca_mode_sussbcoc_5mode  (for GLOMAP_CLIM, no RADAER)
                        glomap_clim_arg_act_get_cdnc
                        glomap_clim_jones_act_get_cdnc
                  atmos_physics1
                        ukca_set_trace_gas_mixratio
                              set_gas_mixratio
                                    ukca_set_environment
                        rad_ctl
                              sw_rad
                                    socrates_init
                                          set_aer
                              lw_rad
                                    set_aer
                  atmos_ukca
                        ukca_get_tracer_varlist
                        ukca_get_ntp_varlist
                        ukca_get_environment_varlist
                        ukca_um_d1_initialise
                              getd1flds
                                    getd1data
                                          ukca_set_environment (for fields in D1)
                        ukca_set_environment (for fields not in D1)
                        ukca_set_emissions_from_nc
                        ukca_set_oxidants_from_nc
                        ukca_step (a.k.a. ukca_main1)
```

Sub-tree for RADAER calls:

```
set_aer
      ukca_radaer_set_aerosol_fields
      ukca_radaer_prepare
      ukca_radaer_band_average
      ukca_radaer_compute_aod
      ukca_radaer_3d_diags
```

In the revised UM-UKCA interface:

- `ukca_setup` to be extended to handle configuration data such as domain dimensions and data from non-UKCA namelists used in UKCA.
- `tstmsk_ukca` to be replaced by using information on field requirement/availability from calls to `ukca_get_tracer_varlist`, `ukca_get_ntp_varlist` and `ukca_set_*_diagnostic_requests` (see Section 4.9 for details).
- `ukca_set_nmspec` to be called before `primary` to allow `primary` to check `D1` vs UKCA's required tracer list by STASH code (see Section 4.9 for details).
- `ukca_radaer_lut_in` calls to be partly replaced by `ukca_radaer_set_lut_entry`
- `ukca_radaer_read_precalc` to read data into temporary storage rather than directly into `precalc` structure; structure to be set-up via new `ukca_radaer_setup` routine.
- `ukca_mode_sussbcoc_5mode` to be called via `glomap_clim_setup` before `atmstep_4a`.
- `ukca_radaer_init` to be simplified to exclude mode configuration data other than that required by the UM (since RADAER will get these data via calls to `ukca_get_config` instead of via UM's `ukca_radaer` structure) and called from `ukca_radaer_setup` before `atmstep_4a`.

- `prepare_fields_for_radaer`, `glomap_clim_arg_act_get_cdnc` and `glomap_clim_jones_act_get_cdnc` to be combined under new subroutine `glomap_clim_calc`; `glomap_clim_radaer_get`, `allocate_ukca_cdnc` and calls to these to be refactored accordingly within a UM wrapper subroutine that calls `ukca_set_environment` to acquire the aerosol mixing ratio fields.
- `glomap_clim_calc` to pass fields to UM via argument list instead of writing direct to `ukca_radaer` structure and to include preparation of aerosol mixing ratio fields for RADAER currently done in `glomap_clim_radaer_get`.
- `set_aer` to call `ukca_radaer_set_environment` and `ukca_radaer_calc` in place of current RADAER routines.
- `ukca_set_emissions_from_nc` to communicate with UKCA using `ukca_get_emission_varlist`, `ukca_register_emission` and `ukca_set_emission`.
- `ukca_set_oxidants_from_nc` to communicate with UKCA using `ukca_set_environment`.
- `atmos_ukca` to include calls to `ukca_update_*_diagnostic_requests`.

A range of other tasks will be required in preparation for use of UKCA and RADAER in LFRic and other potential non-UM applications.

The required implementation tasks are listed in the following subsections with additional notes for individual tasks where appropriate. This should serve as a reference when raising tickets but it is not intended for there to be a 1:1 mapping between tickets and tasks in all cases. Tasks are listed in a logical sequence but the actual order of implementation may vary (subject to restrictions imposed by any task dependencies).

Priority will initially be given to work to support the implementation of prognostic aerosol in LFRic using the GLOMAP 5-mode setup (MS2) with offline oxidants. This will not need to be fully independent of the UM since LFRic will have access to UM modules. In addition, it has its own surrogate versions of `ereport` and `umPrint` that provide access to LFRic-specific routines.

4.3 UKCA Changes Required for Prognostic Aerosols in LFRic

The following tasks have been identified as preparation work for the porting of prognostic aerosols, with offline oxidants chemistry, to LFRic.

- Identify remaining UKCA inputs used directly from UM modules and handle via the API
- Revise the handling of spatial fields currently persisting between timesteps to be LFRic compatible
- Refactor API for emissions and offline oxidants
- Add API support for thread-safe UKCA calls (including modification of previously implemented API routines to handle reduced dimensions)
- Re-work internal flow of spatial data in UKCA for multi-thread compatibility

- Introduce new diagnostic handling scheme
- Make essential diagnostics available via new scheme

The scope of the above tasks will not generally extend to work that is required to support the full chemistry in LFRic at this stage (but see Section 4.5 below). Note that re-working the internal data flow will be a pre-requisite for running with multiple threads in LFRic and will be required for performance reasons but can be done after the initial coupling. Note also that a capability to output UKCA diagnostics is desirable for LFRic at an early stage but is a lower priority requirement that the other tasks listed.

The following notes relate to the diagnostic capability.

Introduce new diagnostic handling scheme

UKCA diagnostic requests will be in the form of two separate lists of field names, for ‘flat’ and ‘full height’ diagnostics, and associated lists of flags (see Section 3.3). The handling of these diagnostic requests requires new subroutines `ukca_set_flat_diagnostic_requests` and `ukca_set_full_ht_diagnostic_requests`. To cater for diagnostics that are not required at every time step, the routines `ukca_update_flat_diagnostic_requests` and `ukca_update_full_ht_diagnostic_requests` should also be provided. They can then be called between time steps to avoid unnecessary work within `ukca_step`. Subroutines `ukca_get_flat_diagnostic_varlist` and `ukca_get_full_ht_diagnostic_varlist` will be provided to retrieve the names and status flags associated with the diagnostic arrays.

The parent will need to establish the availability of diagnostics by checking status flags returned by `ukca_set_flat_diagnostic_requests` and/or `ukca_set_full_ht_diagnostic_requests` after calls to these routines with all required diagnostics set active. UKCA will require a mechanism for determining this availability. In the initial implementation of the standalone code, the UM-based subroutine `tstmsk_ukca` could be used if a table associating option codes with field names were provided within UKCA (to replicate the information available to the UM from the STASH master). This may prove to be the quickest method to implement but it is a rather UM-centric solution and may not be ideal in the long term. Certainly, UKCA should not be constrained to use this mechanism once it is maintained independently from the UM.

Within `ukca_step`, the new diagnostic processing will need to refer to the internal list of active diagnostic requests instead of referring to the STASH system to determine whether a diagnostic is needed. Instead of calling `copydiag` or `copydiag_3d` (which write the diagnostic values to a STASH work array) it will copy them to the 2-D or 3-D diagnostic output array passed to `ukca_main1`. It will also set the status flags as indicated in Section 3.3.

A disadvantage of identifying diagnostics by variable name rather than by number is that UKCA will not be able to refer to ranges of items. Association of appropriate group identifiers with related variables may provide an efficient alternative.

Make essential diagnostics available via new scheme

The systematic replacement of diagnostic processing is a major task that is unsuitable for a single ticket. It will be more practical to add API support for individual diagnostics as and when they are needed. Care must be taken to ensure that existing UM diagnostics are still supported via the old scheme in UM builds, pending conversion of the UM to using the new API (see Section 4.9). It should be possible to continue to satisfy requests for the same diagnostic via the new API in non-UM builds or via the UM-based scheme in UM builds. However, this dual support overhead would be a temporary measure that should only be supported for a small number of essential diagnostics during the transition period so that redundant code can easily be removed in due course.

4.4 Further UKCA Changes Required for Full Chemistry in LFRic

Some additional UKCA components are needed to implement full chemistry in LFRic, notably photolysis and some additional environmental fields that will need to be handled via the API instead of read from external files within `ukca_main1`. Also, the tasks in Section 4.3 will need to be reviewed and changes extended to support full chemistry where not already done.

- Handle Fast-JX specification data via `ukca_setup`
- Move reference data retrieval and interpolation out of UKCA
- Extensions to aerosol tasks above

4.5 RADAER and GLOMAP-CLIM Changes Required for LFRic

The following changes are required to prepare for porting RADAER to LFRic. They include modifications to GLOMAP-CLIM (already ported to LFRic for CDNC calculation). These modifications will provide support for RADAER independently from the UM while also improving consistency between UKCA and GLOMAP-CLIM interfaces.

- Create new RADAER environment fields module providing `ukca_radaer_get_environment_varlist` and `ukca_radaer_set_environment` API routines
- Combine main RADAER subroutines under `ukca_radaer_calc` routine and refactor UM routine `set_aer` accordingly to use this and the environment fields module
- Handle set-up of RADAER via new RADAER API routines `ukca_radaer_setup`, `ukca_radaer_set_lut_entry` and `ukca_radaer_get_config` (using `ukca_get_config` to obtain mode configuration data from UKCA)
- Handle GLOMAP-CLIM configuration via `glomap_clim_setup`
- Combine separate GLOMAP-CLIM routines under `glomap_clim_calc` (with fields destined for RADAER as output arguments) and provide input aerosol mixing ratios as environment fields

- Add API support for thread-safe RADAER and GLOMAP-CLIM calls (and review internal flow of spatial data to ensure compatibility)

It should be possible to implement the GLOMAP-CLIM changes in a way that preserves compatibility with the current implementation in LFRic, thus avoiding the need for a linked LFRic change.

4.6 Remaining UKCA Changes Required for Independence from UM

A comprehensive review of UM modules still used by UKCA modules is required. This may identify other processing that should be moved out of UKCA or encapsulated and performed by a UM handler when built with the UM. A number of tasks below have currently been identified. A basic test-harness will be required to perform simple tests with the UM modules excluded from the build to ensure true independence. This will also enable testing of non-UM functionality.

- Add handler registration mechanism for providing parent-specific routines to UKCA via `ukca_set_handler`.
- Replace internal UM subroutine calls (`umprint`, `ereport`, `tr_mix`, `trsrce`, `qsat`, `qsat_wat_mix` and `lsp_qclear`) with UKCA calls.
- Move code related to plume scavenging and advection out of UKCA (includes plume scavenging diagnostics and some validation checks still in `check_run_ukca`)
- Remove UM-scheme support for diagnostics available via new API (co-ordinated with UM changes detailed in Section 4.9)
- Disable unsupported diagnostics with pre-processor directives in non-UM builds
- Revise support for emissions from explosive volcanos (in `ukca_volcanic_so2` module) that is dependent on the UM grid or disable this in non-UM builds
- Remove redundant UM-specific code (e.g. calculation of zonal means in `ukca_calc_noy_zmeans` routine that cannot be executed without code modification)
- Create test harness for standalone code
- Fully document the independent code separately from existing UMDPs

It may be necessary to disable other UM-specific code not identified above in non-UM builds, depending on whether tasks listed in Section 4.3 – 4.6 have been completed at this stage. It will be important to include standard tests with the test harness as ‘UM’ standard jobs in rose stem to ensure that the independence of the code is not compromised by subsequent changes, pending the transfer of UKCA to its own repository.

4.7 UKCA/RADAER Changes Specifically for Testbeds

A single column testbed environment for UKCA is envisaged that would facilitate large ensemble runs such as parameter perturbation experiments efficiently, without having to resort to separate executions of the main program. This is one of a range of potential future developments to be made possible by the creation of a standalone code. The following API-related tasks will introduce design features specifically for supporting such applications.

- Apply new error handling protocol throughout UKCA.
- Introduce tracer and NTP output options for improving data flow to RADAER
- Add RADAER option to get GLOMAP-mode configuration data via `ukca_radaer_setup` argument list (required if UKCA is unavailable)
- Add reset capability (`ukca_reset` and `ukca_radaer_reset`)

Once the UKCA reset functionality has been implemented, standard tests with the test should include tests where multiple UKCA runs are performed in a single execution of the main program, in particular to ensure that a repeat run with an identical configuration following a reset produces the same results.

4.8 Further Priority Improvements to Standalone Code

A number of other priority UKCA modifications have been identified that will make it easier to develop the newly created standalone code independently of the UM and other specific applications.

- Give parent control of workspace persistence
- Convert remaining diagnostics to new scheme (as needed)
- Use parameters for field names throughout UKCA
- Review external names (literals and keywords) to improve clarity at interface and avoid need for subsequent changes

4.9 UM Changes Required for Compatibility with Future UKCA Development

The following UM changes are required to avoid compatibility issues arising as a result of independent UKCA development.

- Avoid direct use of all UKCA modules in UM other than `ukca_api_mod` , `glomap_clim_api_mod` & `ukca_radaer_api_mod` (use `ukca_get_config` to access configuration data)
- Address potential for tracer list compatibility issues
- Use new API for diagnostics where possible
- Review module and procedure names (for clarity and to avoid potential clashes, avoid using `ukca_` at the start of names on the UM side of the interface)

The notes below discuss the potential handling of STASH-related compatibility issues.

Address Potential for Tracer List Compatibility Issues

Maintaining and developing UKCA and UM codes separately will increase the risk of compatibility issues that may arise if the tracers that are required for existing UKCA configurations change. The ability to make such changes independently without being constrained by parent model requirements is important. In some cases, it may be necessary to modify tracers in the parent when upgrading to the new version but it other cases the parent may be able to adapt automatically and remain compatible. UM changes should be considered that reduce the potential for incompatibility.

The UM holds the active UKCA tracers in the array `tracer_ukca` that is a pointer to the relevant section of `D1`. This is set up within the call to `addres` (in `stash_proc`) to hold the active tracers by calling `tstmsk_ukca` for each of the primary fields in Section 34 to determine which are active. `tstmsk_ukca` checks the option code associated with an item obtained in the UM STASH master file against details of the requested UKCA configuration but there is a risk that the UKCA-specific logic used may become out-of-date if changes are made to UKCA itself.

The UM currently allows tracers to be present in `D1` that are not processed by UKCA. This avoids incompatibility arising in the event that a tracer is removed from a UKCA configuration, but it could result in unnecessary memory allocation that is not detected.

Two options should be considered for reducing the potential for compatibility issues, a basic non-adaptive option that relies on making UM changes in the event of a conflict or an adaptive option that would allow the UM to adapt its tracer list to the UKCA requirement subject to the availability of codes for all required tracers in the STASH master.

- Non-adaptive option: Add an error trap, checking the `D1` active tracer list (`nm_spec_active` defined in `ukca_set_nmspec`) against UKCA's required tracer list, that causes the run to fail if any redundant tracers are present. The opposite case when a tracer required by UKCA is missing from `D1` is already trapped by UKCA. If this solution is chosen, rose-stem tests would fail when upgrading to a new incompatible version of UKCA and appropriate UM changes would be required. Although this approach is simple to implement at the outset, the need to keep the `tstmsk_ukca` logic compatible with UKCA is a significant disadvantage.

- Adaptive option: Force the set of tracers in `D1` to match those specified in UKCA's required tracer list (subject to the availability in STASH) by using the latter to define the former. This approach would make the call to `tstmsk_ukca` below the STASH `addres` subroutine redundant and greatly reduce the potential for incompatibility. The new method would require the `primary` routine to check tracer items against STASH codes corresponding to the list of required tracers from a prior call to `ukca_get_tracer_varlist`, instead of relying on `tstmsk_ukca`. The translation between STASH codes and tracer names is defined by the `nm_spec` array set up in `ukca_set_nmspec` so the call to this subroutine would need to be moved to occur before the STASH processing in `primary`. (`nm_spec_active` set up therein would just be set up with reference to the list of required tracers.) Similar processing would be required in the reconfiguration code (in the subroutine `rcf_address`) to determine which fields are to be included in the output dump.

The changes outlined under the adaptive option above do not solve the potential problem of incompatibility with the validation in the Rose GUI since the availability of tracers would still be determined on the basis of STASH master option codes and `stash_testmask.py` logic. This would need to be addressed by changing `stash_testmask.py` to mirror the new UM method. Two options should be considered:

- Option 1: Call the UKCA routines `ukca_setup` and `ukca_get_tracer_varlist` directly from `stash_testmask.py` to obtain the UKCA required tracer list for the selected configuration. This would be the most robust way of ensuring compatibility of the Rose validation with the current version of UKCA. However, the need for compilation of the FORTRAN code may make this impractical.

- Option 2: Replicate the logic of the FORTRAN code in Python (as is currently done for the STASH-based method). This may introduce a requirement for UKCA to maintain such a Python code that could be called by the Rose validation. In this case it should be designed as a generic code that might also be useful in non-UM applications. Note that in principle Python code could be used to implement the internal workings of `ukca_setup` within UKCA, avoiding the need for code-replication in different languages. However, there is a risk that this could adversely affect UKCA performance and/or maintainability so replication of the code is probably preferable.

Use new API for diagnostics where possible

Until all UKCA diagnostics required by the UM are available via the new UKCA API, the UM will need to support both old and new diagnostic handling schemes. Which scheme is to be used for an individual STASH code will be determined by whether or not a corresponding UKCA name is available via a look-up table. Processing of STASH requests for UKCA prognostics will not require any dual support as these are already supported by the API.

STASH requests for API-supported diagnostics will be consolidated (by the UM), removing duplicates, to create UKCA diagnostic requests that will ensure the diagnostic fields are present in the output if they are available from the current configuration. The UKCA diagnostic requests will be in the form of two separate lists of field names, for ‘flat’ and ‘full height’ diagnostics, and associated lists of flags (see Section 3.3). These will be passed to UKCA by calling `ukca_set_flat_diagnostic_requests` and `ukca_set_full_ht_diagnostic_requests` respectively. Some diagnostics are not required at every time step so `ukca_update_flat_diagnostic_requests` and `ukca_update_full_ht_diagnostic_requests` may be called between time steps.

The validation of STASH requests is currently done by `tstmsk_ukca` during execution of the subroutine `prelim` (within `stash_proc`) which loops round all STASH requests calling `tstmsk` to check availability of each item. To use the new API, the UM must instead check for availability of prognostics by field name in the lists returned by `ukca_get_tracer_varlist` or `ukca_get_ntp_varlist`. The availability of API-supported diagnostics will be established by checking status flags returned by `ukca_set_flat_diagnostic_requests` and/or `ukca_set_full_ht_diagnostic_requests` after calls to these routines with all required diagnostics set active. Having established the availability or otherwise of all required output fields in the current UKCA configuration, individual STASH requests can then be processed in the existing `prelim` loop by checking against this field availability status. Once all diagnostics are available via the new API, the call to `tstmsk_ukca` below `prelim` will become redundant.

The UM must copy the requested fields from the UKCA output arrays to the appropriate section's STASH work array and call `stash` to do the handling. The variable names associated with the 2-D and 3-D diagnostic arrays will match those passed to the subroutines called to set the requests and can also be obtained by calling `ukca_get_flat_diagnostic_varlist` and `ukca_get_full_ht_diagnostic_varlist`. These subroutines can be called after `ukca_step` to access the status flags that indicate the validity of the fields before copying them to STASH.

As for the tracers, there is a risk that the Rose GUI test for availability of diagnostics, based on STASH master option codes and `stash_testmask.py` logic, may become incompatible with UKCA as UKCA is developed independently. This problem would be best addressed for all STASH requests (for tracers and other fields) in a consistent way.

Appendix: Categorisation of Modules

The following is a list of all modules in UKCA and GLOMAP_CLIM directories of the UM repository at vn11.7 with individual modules marked to indicate whether they are expected to be categorised as UKCA, RADAER or UM modules once the refactoring is complete (or whether they themselves require refactoring as a consequence of including code that will be split by the interface). This list is an important reference for any development work proceeding during the transition period. All such work should respect the intended separation of name spaces to avoid conflicting with the aims of the API development work.

Key to Category Codes

I	Internal to UKCA
R	Internal to RADAER
X	External (i.e UM-side module)
IX	Processing to be divided between UKCA and UM
RX	Processing to be between RADAER and UM
-	Expected to become redundant

Files in UKCA directory

`asad_bedriv.F90`	**`I`**
`asad_bimol.F90`	**`I`**
`asad_cdrive.F90`	**`I`**
`asad_chem_flux_diags.F90`	**`I`**
`asad_cinit.F90`	**`I`**
`asad_diffun.F90`	**`I`**
`asad_findreaction.F90`	**`I`**
`asad_flux_dat.F90`	**`I`**
`asad_ftoy.F90`	**`I`**
`asad_fuljac.F90`	**`I`**
`asad_fyfixr.F90`	**`I`**
`asad_fyinit.F90`	**`I`**
`asad_fyself.F90`	**`I`**
`asad_hetero.F90`	**`I`**
`asad_impact.F90`	**`I`**
`asad_inicnt.F90`	**`I`**
`asad_inicnt_col_mod.F90`	**`I`**
`asad_inijac.F90`	**`I`**
`asad_inimpct.F90`	**`I`**
`asad_inix.F90`	**`I`**

asad_inrats.F90	I
asad_jac.F90	I
asad_mod.F90	I
asad_posthet.F90	I
asad_prls.F90	I
asad_setsteady.F90	I
asad_sparse_vars.F90	I
asad_spimpmjp.F90	I
asad_spmjpdriv.F90	I
asad_steady.F90	I
asad_totnud.F90	I
asad_trimol.F90	I
atmos_ukca_mod.F90	X
atmos_ukca_setup_mod.F90	X
emiss_io_mod.F90	X
fastjx_data.F90	I
fastjx_extral.F90	I
fastjx_inphot.F90	IX
fastjx_jratet.F90	I
fastjx_miesct.F90	I
fastjx_opmie.F90	I
fastjx_photoj.F90	I
fastjx_set_aer.F90	I
fastjx_solar2.F90	I
fastjx_specs.F90	IX
fastjx_sphere.F90	I
get_emdiag_stash_mod.F90	X
get_molmass_mod.F90	I
get_nmvoc_mod.F90	I
get_noy_mod.F90	I
init_radukca.F90	X
o3intp_mod.F90	X
param2d_mod.F90	I
photolib/*.F90	I
spcrg3a_mod.F90	R
tstmsk_ukca_mod.F90	X
ukca_2d_bc_read_interp.F90	X
ukca_abdulrazzak_ghan.F90	I
ukca_activ_mini_snr_mod.F90	X
ukca_activ_mod.F90	I
ukca_activate.F90	I
ukca_add_emiss_mod.F90	I
ukca_aero_ctl.F90	I
ukca_aero_step.F90	I
ukca_aerod.F90	I
ukca_age_air_mod.F90	I
ukca_ageing.F90	I
ukca_all_tracers_copy_mod.F90	X

ukca_api_mod.F90	I
ukca_be_drydep.F90	I
ukca_be_wetdep.F90	I
ukca_binapara_mod.F90	I
ukca_calc_coag_kernel.F90	I
ukca_calc_drydiam.F90	I
ukca_calc_noy_zmeans.F90	-
ukca_calc_plev_diag_mod.F90	X
ukca_calcminmaxgc.F90	I
ukca_calcminmaxndmdt.F90	I
ukca_calcnucrate.F90	I
ukca_cdnc_jones_mod.F90	I
ukca_cdnc_mod.F90	X
ukca_ch4_stratloss.F90	I
ukca_check_md_nd.F90	I
ukca_check_radaer_coupling_mod.F90	I
ukca_chem1_dat.F90	I
ukca_chem_aer.F90	I
ukca_chem_defs_mod.F90	I
ukca_chem_diags_allts_mod.F90	I
ukca_chem_diags_mod.F90	I
ukca_chem_master.F90	I
ukca_chem_offline.F90	I
ukca_chem_raq.F90	I
ukca_chem_raqaero_mod.F90	I
ukca_chemco.F90	I
ukca_chemco_raq.F90	I
ukca_chemco_raq_init_mod.F90	I
ukca_chemistry_ctl.F90	I
ukca_chemistry_ctl_BE_mod.F90	I
ukca_chemistry_ctl_col_mod.F90	I
ukca_cloudproc.F90	I
ukca_coag_coff_v.F90	I
ukca_coagwithnucl.F90	I
ukca_cond_coff_v.F90	I
ukca_conden.F90	I
ukca_config_defs_mod.F90	I
ukca_config_specification_mod.F90	I
ukca_constants.F90	I
ukca_cspecies.F90	I
ukca_d1_defs.F90	X
ukca_dcoff_par_av_k.F90	I
ukca_ddcalc.F90	I
ukca_ddepaer_incl_sedi_mod.F90	I
ukca_ddepaer_mod.F90	I
ukca_ddepctl.F90	I
ukca_ddepo3_ocean_mod.F90	I
ukca_ddeprt.F90	I

ukca_deriv.F90	I
ukca_deriv_aero.F90	I
ukca_deriv_raq.F90	I
ukca_deriv_raqaero_mod.F90	I
ukca_dissoc.F90	I
ukca_diurnal_isop_ems.F90	I
ukca_diurnal_oxidant.F90	I
ukca_drydep.F90	I
ukca_drydiam_field_mod.F90	I
ukca_eg_tracers_total_mass_mod.F90	X
ukca_emdiags_struct_mod.F90	I
ukca_emiss_ctl_mod.F90	I
ukca_emiss_diags_mod.F90	I
ukca_emiss_diags_mode_mod.F90	I
ukca_emiss_factors.F90	I
ukca_emiss_mod.F90	I
ukca_emiss_mode_mod.F90	I
ukca_emiss_struct_mod.F90	I
ukca_environment_check_mod.F90	I
ukca_environment_fields_mod.F90	I
ukca_error_mod.F90	I
ukca_extract_d1_data_mod.F90	X
ukca_fastjx.F90	I
ukca_fdiss.F90	I
ukca_fdiss_constant_mod.F90	I
ukca_feedback_mod.F90	X
ukca_fieldname_mod.F90	I
ukca_fixeds.F90	I
ukca_flupj.F90	X
ukca_fracdiss.F90	I
ukca_hetero_mod.F90	I
ukca_impc_scav.F90	I
ukca_inddep.F90	I
ukca_ingridg.F90	I
ukca_iniasad.F90	I
ukca_init.F90	I
ukca_interp.F90	X
ukca_inwdep.F90	I
ukca_light.F90	I
ukca_light_ctl.F90	I
ukca_main1-ukca_main1.F90	I
ukca_mode_check_artefacts_mod.F90	I
ukca_mode_diags_mod.F90	I
ukca_mode_setup.F90	I
ukca_mode_tracer_maps_mod.F90	I
ukca_mode_verbose_mod.F90	I
ukca_nc_emiss_mod.F90	IX
ukca_nmspec_mod.F90	X

ukca_ntp_mod.F90	I
ukca_option_mod.F90	IX
ukca_phot2d.F90	X
ukca_photol.F90	I
ukca_plev_diags_mod.F90	X
ukca_pm_diags_mod.F90	I
ukca_pr_inputs_mod.F90	I
ukca_prim_du_mod.F90	I
ukca_prim_moc.F90	I
ukca_prim_ss.F90	I
ukca_radaer_3D_diags.F90	R
ukca_radaer_band_average.F90	R
ukca_radaer_compute_aod.F90	R
ukca_radaer_get.F90	X
ukca_radaer_get_specinfo.F90	R
ukca_radaer_init-ukca1.F90	X
ukca_radaer_lut_in.F90	RX
ukca_radaer_lut_mod.F90	R
ukca_radaer_precalc_mod.F90	R
ukca_radaer_prepare.F90	R
ukca_radaer_read_luts.F90	X
ukca_radaer_read_precalc.F90	RX
ukca_radaer_saved_mod.F90	X
ukca_radaer_set_aerosol_field.F90	RX
ukca_radaer_struct_mod.F90	X
ukca_radaer_tlut_mod.F90	R
ukca_rainout.F90	I
ukca_raq_diags_mod.F90	I
ukca_read_aerosol.F90	X
ukca_read_offline_oxidants_mod.F90	IX
ukca_read_reff.F90	X
ukca_remode.F90	I
ukca_scavenging_diags_mod.F90	X
ukca_scavenging_mod.F90	X
ukca_scenario_ctl_mod.F90	I
ukca_scenario_prescribed.F90	I
ukca_scenario_rcp_mod.F90	X
ukca_scenario_wmoa1.F90	I
ukca_sediment.F90	I
ukca_set_array_bounds.F90	X
ukca_setd1defs.F90	X
ukca_setup_chem_mod.F90	I
ukca_setup_indices.F90	I
ukca_setup_mod.F90	I
ukca_solang.F90	I
ukca_solflux.F90	I
ukca_solvecoagnucl_v.F90	I
ukca_strat_aero_clim_mmr.F90	I

`ukca_strat_update.F90`	I
`ukca_stratf.F90`	I
`ukca_surfddr.F90`	I
`ukca_top_boundary.F90`	I
`ukca_trace_gas_mixratio.F90`	X
`ukca_tracer_stash.F90`	X
`ukca_tracer_vars.F90`	I
`ukca_tracers_mod.F90`	I
`ukca_transform_halogen.F90`	I
`ukca_trop_hetchem.F90`	I
`ukca_tropopause.F90`	I
`ukca_um_interf_mod.F90`	X
`ukca_um_surf_wet_mod.F90`	X
`ukca_update_emdiagstruct_mod.F90`	I
`ukca_vapour.F90`	I
`ukca_vgrav_av_k.F90`	I
`ukca_volcanic_so2.F90`	IX
`ukca_volume_mode.F90`	I
`ukca_water_content_v.F90`	I
`ukca_wdeprt.F90`	I
`ukca_wetdep.F90`	I
`ukca_wetox.F90`	I

Files in GLOMAP_CLIM directory

`get_gc_aerosol_fields_mod.F90`	I
`glomap_clim_calc_aird_mod.F90`	I
`glomap_clim_calc_drydiam_mod.F90`	I
`glomap_clim_cdnc_mod.F90`	IX
`glomap_clim_drydp_nd_out_mod.F90`	I
`glomap_clim_fields_mod.F90`	I
`glomap_clim_get_netcdffile_rec_mod.F90`	X
`glomap_clim_identify_fields_mod.F90`	X
`glomap_clim_netcdf_ancil_mod.F90`	X
`glomap_clim_netcdf_anclist_mod.F90`	X
`glomap_clim_netcdf_io_mod.F90`	X
`glomap_clim_netcdf_parameter_mod.F90`	X
`glomap_clim_option_mod.F90`	IX
`glomap_clim_pop_md_mdt_nd_mod.F90`	I
`glomap_clim_radaer_get.F90`	IX
`prepare_fields_for_radaer_mod.F90`	IX
`tstmsk_glomap_clim_mod.F90`	X