Proposed Application Program Interface Design for UKCA

John Hemmings, March 2019, v4

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 to be 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.

RADAER is called separately from UKCA, by the UM's radiation processing, but is dependent on GLOMAP-mode data from UKCA (obtained via the UM's `D1` array). 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 though and the work to separate it from the UM is outside the scope of this document. It is envisaged that RADAER will have a separate API for interfacing with a parent model but here it is effectively treated as an integral part of the UM. Like the rest of the UM, it will be modified to communicate with UKCA through the UKCA API.

The GLOMAP-mode climatology scheme, GLOMAP_CLIM, comprises other UKCA-related code that will need to be ported to LFRic. This code is dependent on both UKCA and RADAER. However, UKCA has no dependencies on GLOMAP_CLIM and the separation of GLOMAP_CLIM from the UM is also considered out-of-scope in this document. As with RADAER, the UKCA API design will support GLOMAP_CLIM's use of UKCA.

A set of general design principles to be followed is given in Section 2. Section 3 outlines design considerations specific to UKCA and Section 4 gives an outline of the refactoring work required to implement the new API within the UM.

2. General Design Principles

- UKCA will be available to a parent application via a single API module 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.
- Names of all subroutines and parameters presented via the API will start with `ukca_` to avoid polluting the parent's namespace.
- All run-time communication between the parent model and UKCA will be via argument lists.
- 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.
- 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. 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.
- Any IO performed within UKCA subroutines will be under the control of the parent model. Where this IO occurs in an internal UKCA subroutine, the parent will pass control data and/or a handler routine to UKCA via an appropriate top-level subroutine. Other IO will be handled by top-level UKCA subroutines that perform specific input or output tasks (e.g. reading emissions data from a NetCDF file). Such subroutines should not perform other processing so could be replaced by substitute calls in the parent model (or suppressed) without side effects. This will allow flexibility for the parent model to get input data from different sources or at different intervals or handle output data differently.
- 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.
- The state variables will be passed to UKCA via the main time step subroutine. Other input data (physical environment, emissions, etc.) will instead be initialised/updated within UKCA via separate top-level subroutine calls and optionally allowed to persist between time steps until the next update (as emissions do at present). This will

reduce overheads in cases where UKCA is run in a fixed environment. It should also help to avoid excessively long argument lists.

- The subroutine for executing a UKCA time step will only process fields specified on the UKCA model grid (which is configurable by the parent). Any interpolation to this grid that may be required will be handled by other top-level subroutines. This will help to support efficient memory management where multiple instances of the model grid are distributed over processing elements in a parallel environment.
- 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.
- It must be possible to run UKCA without internal persistence of fields between time steps (as required for LFRic). Whether or not persistence is supported internally will therefore be under the control of the parent via UKCA's configuration settings. Any persistence of fields that is required must be provided for by top-level subroutines to pass the data to and from the parent in the event that internal persistence is disabled.
- 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
- 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.
- On encountering a fatal error condition, UKCA will 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 will allow the parent model to either abort or continue without UKCA as required. Warning conditions will provide the same information to a generic internal subroutine that calls a parent model subroutine if one is provided. Any output messages will be under the control of the parent model via this handler routine.
- 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.

3. Elements of the UKCA Interface

This section describes the elements of UKCA that must be supported by the new API (i.e. all data currently transferred between the UM and UKCA and all UM procedures currently called from within UKCA). It then describes some specific aspects of the API design and gives a provisional list of subroutines for its implementation. 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 initialisation subroutine `ukca_init`, the time_step subroutine `ukca_main1` and the plume scavenging subroutine `ukca_plume_scav` embedded in the UM convection code.

3.1 UKCA Input and Output Data

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

Input Only Data:

- Parameters, options and other configuration data (from `run_ukca` namelist)
- Domain configuration data: field dimensions etc (from parent)
- Environment: atmospheric physics and land surface, CLASSIC aerosols for heterogeneous chemistry (from `D1` array)
- Emissions (from NetCDF files)
- Offline oxidants (from NetCDF files)
- Climatologies and other reference data including AeroClim climatology, data for Fast-JX photolysis and 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 parent via `ukca_set_trace_gas_mixratio`).
- Requests for diagnostics (from STASH system data)

Input/Output Data (model state):

- Tracers (from parent via `ukca_main1` argument list)
- Non-transported prognostics (from `D1` array)

Output Only Data:

- Diagnostics (to STASH system)
- Log output and error messages (to files)
- Test output optionally produced when lower BCs for long-lived gases are read (to file)

3.2 Parent-specific Subroutines Required by UKCA

At present, the following UM-specific subroutines (listed here according to their general function) 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`)
- 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`)
- Print management for output to log file (`umprint`)
- Warning handling (`ereport`)
- Dr Hook timer calls (`dr_hook`)[1]

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`). In other cases, the requirement is for compatibility with the UM's numerical schemes. Some of the subroutines include significant amounts of non-UM specific processing that can be separated out.

The reading of data from NetCDF and sequential files in `ukca_main1` will be moved out of that subroutine. This will give the parent maximum flexibility to control how these data are provided and ensure that file access during the run can be avoided if necessary. Where the reading of data does not involve a significant amount of UKCA-specific processing, it can be left to the parent to do the file access and pass the fields to UKCA subsequently. However, any non-trivial UKCA-specific processing involved in reading data should remain the responsibility of UKCA. Such processing should be supported by new UKCA top-level subroutines for use by any parent application.

UM-specific subroutine calls still in `ukca_main1` or in other UKCA subroutines (including the new top-level 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.

Note 1: Dr Hook is a candidate for migration to the SHUMlib library which would make it available to non-UM applications and may therefore not need replacement. (See https://code.metoffice.gov.uk/trac/um/wiki/ProjectDocumentation/SharedFFLibrary/CandidatesForMigration)

3.3 Specific Design Aspects

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. This means that UKCA will obtain its configuration data via a subroutine call rather than from a namelist. Different UKCA configurations require different subsets of configuration variables so the values of individual variables will be 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 supported and will produce undefined results.

In addition to the user-defined configuration data, UKCA requires a set of specification data for the FAST-JX photolysis scheme. In the UM, these 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, so the file access will be the responsibility of UKCA. However, the existing subroutines are UM-specific because they read the data on PE0 and broadcast them to the other PEs. To support this (or similar requirements for parent handling of the data), the API must provide subroutines that allow the data set to be passed to the parent and back to UKCA, although the parent should not need to know the details of the data. The file access will be moved out of `ukca_main1` as indicated in Section 3.2.

Tracers and Non-transported Prognostics

At present, the UKCA species in the tracer array passed to `ukca_main1` are determined with reference to the STASH system. Their order corresponds to the order of these fields in the UM STASH master file, with the actual indexing determined by which fields are active. It is not possible to preserve this relationship in a standalone code: it must be possible to modify the UKCA tracer list in future UKCA development work without reference to a particular parent model.

In the new UKCA code, the tracer species and their order will instead be determined directly by UKCA. The parent will then retrieve an arbitrary 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 will be followed for non-transported prognostics. In both cases, the specific fields and their positions may differ between different UKCA configurations.

A complication arises with UKCA’s plume scavenging functionality that is integrated with UM convection to improve the accuracy of aerosol removal fluxes. It involves UKCA-specific processing outside the main UKCA time step that accesses the UM-specific tracer array.

The way this is done needs to be made more generic so that plume scavenging can be used in other parent models.

Environment Data

Environment data refers to input fields, other than the tracers and non-transported prognostics, that are provided on the model grid. These data 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 will not attempt to handle the environment data as a single array of fields but will instead allow each field to be set separately by name.

For flexibility, environment data will be set by calling a new top-level subroutine instead of being set in the call to `ukca_main1` and each call will set a single named field. In applications where some or all environment data are fixed, the fixed fields can then be set at the beginning of a run and need not be updated. This means that `ukca_main1` will need extra validation that ensures all required environment fields are set. UKCA should be able to provide the parent with a list of required fields by name on request. However, it should not fail to run if additional fields are set.

Scalar concentrations of long-lived gases can be treated as a special case of environment data and would be updatable by name in the same way. A subroutine, equivalent to `ukca_set_trace_gas_mixratio`, that treats these as a collection of environment variables, with default values, should also be provided in the API as an alternative.

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, reference data defined on an arbitrary grid will not be processed directly by `ukca_main1`. They will instead be converted to fields on the model grid and interpolated to the current time externally to `ukca_main1`. The functionality for accessing these data from reference data sets will not be required for all applications. (Some may instead require them to be provided by the parent as environment variables on the model grid.) However, it is likely to be useful for some non-UM applications, so ideally should be supported by the API rather than leaving the code as part of the UM.

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`). These data can be treated as environment data once the necessary interpolation has been done. The top-level subroutines that do the interpolation will just be alternatives to those for setting environment data. That is, they will give the same outputs but have different input data.

In the UM, each of these reference data sets are read by routines that perform a UM-specific read on the first time step and process it in a UM-independent way on subsequent time steps. The read functionality is specific to the file format rather than to UKCA and is relatively trivial, so need not be included in the standalone UKCA code base. The read step will therefore be separated from the subsequent processing and become the responsibility of the parent. For the long-lived gases, there is an option to write test output to a file after reading data from an RCP file. This functionality will remain the responsibility of the UM for now but could be implemented in UKCA at a later date if needed in other applications.

If the 2-D photolysis scheme is selected, reference data will also be required for species photolysis rates. Again, these data are read only on the first time step. However, the saved data are interpolated in time daily, with only the data for the current day being broadcast across the different PEs (the broadcast step being UM-specific). This is desirable because of the relatively large size of the complete data set. To replicate this functionality requires a slightly more complicated division of responsibility than for the other reference data. Ideally, UKCA should have the responsibility for knowing when to update and would do the time interpolation but should allow the option of calling procedures provided by the parent to support the distribution of the data in a parallel environment. When prioritizing this work, it should be noted that the 2-D photolysis scheme is only used for testing and debugging now, having been succeeded operationally by FAST-JX. It is unclear at present whether it will be needed in non-UM applications.

A disadvantage of the present schemes is that they rely on the whole data set (or, in the case of the 2-D photolysis, its whole spatial extent) being held on each PE. This does impose an unnecessary memory overhead that is likely to be unsuitable for LFRic. LFRic might instead provide the data to UKCA as environment data on the model grid at each time step. In that case, additional top-level subroutines could be provided in the API at a later date to support UKCA-specific interpolation and return interpolated fields to the parent instead of updating UKCA directly.

Emissions and Offline Oxidants

In the UM, the emissions of each tracer from various sources are controlled by emissions data and metadata from a NetCDF data set and the way in which these data are handled is prescribed within the UKCA time step processing. Oxidant species for the offline oxidants chemistry scheme are treated similarly. The functionality of this NetCDF-based emissions system should be preserved for the UM and is generally recommended for non-UM applications too, but is likely to be restrictive for some. Testbed applications, for example, would ideally have the option of setting up simple emissions fields without necessarily reading any external data at all.

To allow flexibility, the extraction of NetCDF data will be moved out of `ukca_main1` and performed by a new top-level UKCA subroutine that will be called separately by the UM and can be called by other parent models as required. This will involve separating the data extraction from subsequent state update processing which will remain within `ukca_main1`. The new top-level subroutine will need to call parent routines to gain access to UM-specific low-level IO functions in module `emiss_io_mod` or alternatives provided by other parent applications.

In the NetCDF emission system, the frequency at which the internal emissions data are updated is controlled by metadata in the file for each species and the data are allowed to persist between time step. Internal persistence of fields between time steps may be disallowed by the parent (e.g. LFRic) as indicated in Section 2. If the NetCDF emission system is to be useable in this context with the same method of controlling update frequency, subroutines will be needed for transferring the current emissions data to and from the parent.

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 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.

Field Names

The CF naming convention (http://cfconventions.org/standard-names.html) will be adopted for the field names to be used by UKCA for prognostic and diagnostic fields. This will avoid any ambiguity at the interface level and reduce reliance on external documentation at the expense of a small overhead of processing long character arrays. (Use of parameters for these names in the code will ensure readability is not compromised.)

The use of CF names will be specific to the UKCA interface and 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.

3.4 Provisional UKCA API Subroutines

A provisional list of API subroutines is given here. 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 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_set_fastjx_specs_from_file`

Reads the FAST-JX specification data from sequential files. Normally called once at the beginning of a run.

`ukca_get_fastjx_specifications`

Returns the FAST-JX specification variables (e.g. for broadcasting to PEs)

`ukca_set_fastjx_specifications`

Sets the values of the FAST-JX specification variables (e.g. after broadcasting).

`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. Will also set up or copy to UKCA other data that are currently used directly from UM modules and will be fixed for the duration of the UKCA run (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.

`ukca_setup_mode_sussbcoc_5mode`

Performs a limited setup (of a specific GLOMAP_mode configuration) required for using RADAER and/or GLOMAP climatology scheme. Also called as an internal subroutine within `ukca_setup`.

`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_config_*`

Returns other specific information about the current configuration as indicated by a suffix replacing `*`. A small set of subroutines will retrieve various types of information required by a parent application (e.g. selected configuration options). The set can be added to as needed to support different parents without compromising backwards compatibility.

`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. These are fields on 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.

`ukca_set_env_from_lat_ht`

Sets or updates a named environmental input field by selecting or interpolating from reference data comprising a time series of 2-D fields on a latitude-height grid. Will typically be called at each time step.

`ukca_set_gas_mixratio`

A synonym in the API for `ukca_set_trace_gas_mixratio`: Sets up or updates mixing ratios of long-lived gases (primarily trace gases) for use as lower boundary conditions (and in some cases for the atmosphere as a whole). These are fields that may be varied by the parent during the run. Will typically be called at each time step if that is the case.

`ukca_set_gas_mr_from_tseries`

Sets or updates mixing ratios of long-lived gases by interpolating from a reference time series (e.g. from RCP scenario data). Will typically be called at each time step.

`ukca_set_emissions_from_nc`

Sets or updates emissions input data using a list of NetCDF data files defined in the current configuration. May be called at each time step but the actual update frequency is controlled by metadata in the NetCDF data sets.

`ukca_get_emission_fields`

Returns the current set of emission fields. Required when the internal persistence of emission fields between time steps is undesirable and persistence must be handled by the parent.

`ukca_set_emission_fields`

Sets the emissions fields by copying in fields obtained using `ukca_get_emission_fields`.

`ukca_set_oxidants_from_nc`

Sets or updates oxidants input data (for offline oxidants chemistry scheme) using a NetCDF data file defined in the current configuration. May be called at each time step but the actual update frequency is controlled by metadata in the NetCDF data set.

`ukca_get_oxidant_fields`

Returns the current set of oxidant fields. Required when the internal persistence of emission fields between time steps is undesirable and persistence must be handled by the parent.

`ukca_set_oxidant_fields`

Sets the oxidant fields by copying in fields obtained using `ukca_get_oxidant_fields`.

`ukca_set_photol_rates_from_lat_ht`

Sets or updates photolysis rates for 2-D photolysis scheme by interpolating from a multi-species latitude-height time series. These are reference rates for the current day, varying by latitude, height, time of day and time of year. May be called at each time step but the actual update frequency (daily) is controlled by UKCA.

`ukca_step`

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

`ukca_activate`

Calculates number concentration of aerosol particles which become activated into cloud droplets. Required for the GLOMAP climatology scheme but also called as an internal routine within `ukca_step.`

`ukca_plume_scav`

Determines the change in tracer content in a model layer due to scavenging by precipitation.

`ukca_reset`

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

4. Implementation

The UKCA code will be 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 rose stem tests. This section aims to identify the scope of the development work required.

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 UKCA output. From a UKCA perspective, these UM modules will be UM-specific and will not do any UKCA processing other than that required for interfacing with the UM. The top-level UKCA procedures that are required by the UM (or may be required by another parent) will be made available via `USE` statements in an API module `ukca_api_mod`.

All communication between the UM and UKCA modules should eventually be via procedures accessible via the API module. One-way communication from UKCA to the UM via use of internal UKCA 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 changes that do not affect the defined API, making it impossible to maintain the UKCA code 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 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 FORTRAN code 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 is not possible with respect to the validation of data in the Rose GUI.

The UKCA-specific calls from the UM code that will be affected by the re-factoring are shown in the tree below (highlighted in bold with the UM calling chains shown), followed by notes outlining the changes needed to use the new API.

```
um_shell
      readlsta
            read_nml_run_ukca
            ukca_init
      stash_proc
            Prelim
                  tstmsk
                        tstmsk_ukca  (for STASH requests)
            addres
                  primary
                        tstmsk
                              tstmsk_ukca  (for tracers)
                  ukca_set_nmspec
                  ukca_set_conv_indices
      u_model_4a
            atm_step_4a
                  ukca_mode_sussbcoc_5mode  (for RADAER runs)
                  allocate_ukca_cdnc
                        ukca_mode_sussbcoc_5mode  (for GLOMAP_CLIM runs)
                        glomap_clim_arg_act_cdnc
                              ukca_activate
                  atmos_physics1
                        ukca_set_trace_gas_mixratio
                  atmos_physics2
                        ni_conv_ctl
                              tracer_copy
                                    ukca_plume_scav_initial
                              glue_conv_5a OR glue_conv_6a
                                    misc. convection subroutines
                                          convec2_4a5a OR convec_6a
                                                ukca_plume_scav
                              tracer_restore
                                    ukca_plume_scav_final
                  ukca_main1
```

In the revised UM-UKCA interface:

- `ukca_init` call to become an internal UKCA subroutine and replaced by a call to `ukca_setup`. This will copy the user configuration data (obtained by `read_nml_run_ukca`) into UKCA, call `ukca_init` and also initialise tracer and NTP data. Requires moving some processing from `ukca_main1`.
- `tstmsk_ukca` to be replaced by using information on field availability from calls to `ukca_get_tracer_varlist`, `ukca_get_ntp_varlist` and `ukca_set_*_diagnostic_requests` (see 4.2 & 4.3 for details).

- `ukca_set_nmspec` & `ukca_set_conv_indices` to be retained in modified form as UM subroutines to provide mapping data for plume scavenging (see 4.2 & 4.3). Should ideally be renamed not to start with `ukca_` to clarify that they are not UKCA subroutines.
- `ukca_mode_sussbcoc_5mode` to be presented as a UKCA API subroutine for RADAER and GLOMAP climatology runs.
- `ukca_activate` to be presented as a UKCA API subroutine for GLOMAP climatology runs.
- `ukca_set_trace_gas_mixratio` (alias `ukca_set_gas_mixratio`) to be presented as a UKCA API subroutine.
- `ukca_plume_scav` to be presented as a UKCA API subroutine.
- `ukca_main1` to become an internal UKCA subroutine called by `ukca_step`, `ukca_step` will be encapsulated in a UM wrapper with calls to `ukca_set_fastjx_specs_from_file`, `ukca_get_fastjx_specifications`, `ukca_set_fastjx_specifications`, `ukca_set_environment`, `ukca_set_env_from_lat_ht`, `ukca_set_gas_mr_from_tseries`, `ukca_set_emissions_from_nc`, `ukca_set_oxidants_from_nc`, `ukca_set_photol_rates_from_lat_ht`, `ukca_update_*_diagnostic_requests` as required to setup data in preparation for the time step.

Since the UM only performs a single UKCA integration, it should not need to call `ukca_reset` to do housekeeping and this subroutine could be a later addition to the API.

The required implementation tasks are outlined in the following subsections. 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 subsections. Priority will initially be given to work required for a 'prototype' standalone code that at least supports the GLOMAP 5-mode setup (MS2) with offline oxidants running independently of the UM.

4.1 Moving `D1` Access and STASH Handling out of UKCA

The UM call to `ukca_main1` will be replaced by a call to a UM wrapper subroutine that does the necessary UM-specific preparation for calling the modified `ukca_main1` via the API subroutine `ukca_step`. The UM-specific initialisation code related to communications with `D1` can then be moved outside `ukca_main1` into the UM wrapper routine, as can the copying of non-transported prognostics back to `D1`. Calls to the STASH handling subroutine `stash` for processing diagnostics at the end of the time step will also be moved out, including those for the pressure level diagnostic sections that occur within the internal `ukca_plev_diags` subroutine. The whole subroutine can be moved as it is simply a UM reprocessing of other UKCA diagnostics.

Moving `D1` access outside `ukca_main1` requires other initialisation code in `ukca_main1` on which `D1` access is dependent to be moved out too. In general, this is initialisation that may be needed by any parent model to initiate a UKCA run. In the standalone code, it will be done in `ukca_setup`. Some significant re-working of the initialisation code moved out of `ukca_main1` is required to separate data structures and processing into those that are UM-specific and those that are UM-independent, since only the latter can go into the UKCA subroutines. The former will become part of the UM module containing the wrapper subroutine (or UM modules used therein).

The wrapper subroutine will determine which non-transported prognostics are required from `D1` by calling the new subroutine `ukca_get_ntp_varlist` and converting the field names returned in the list to STASH codes using a new lookup table. Non-transported prognostics extracted from `D1` will be passed to UKCA via the `ukca_step` call as a 4-D array holding 3-D fields corresponding to the field name list.

The remaining data extracted from `D1` is environment data that will be passed to UKCA via calls to `ukca_set_environment` for each field. Which environment fields are required should now be determined by calling the new subroutine `ukca_get_environment_varlist` instead of by using UKCA configuration data. This will reduce the amount of configuration data that needs to be made accessible to the parent. UKCA will need to record the status of each required environment field for validation purposes (e.g. by maintaining status flags for each field to show whether or not it has been set). A new configuration setting will determine whether or not UKCA's internal copies of the environment fields are deallocated at the end of the time step.

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 will not make assumptions about the extent of any of its input fields but will ensure that the data at least span the required domain and that only the required extents are handled internally. This will avoid overheads associated with redundant regions that could potentially be very large.

In cases where reduced dimension fields are used in actual arguments (i.e. in applications running UKCA in 1-D or 0-D), fields will be copied to the equivalent higher dimensional arrays that are processed in `ukca_main1`. Copying between parent fields and internal fields will be handled by overloading the `ukca_step` subroutine interface with alternative versions of the subroutine that call `ukca_main1` with appropriately modified fields. Other top-level subroutines handling parent fields will be implemented likewise.

4.2 Handling Tracers

The UM holds 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 UKCA configuration.)

To use the new API, the UM must instead get the list of active tracer fields from UKCA by calling `ukca_get_tracer_varlist`. The way that UKCA determines this list according to the details of the configuration will be independent of the UM and will be determined within `ukca_setup`. The UM will use the list as a basis for setting up the tracer fields in `D1` but must check that each of the fields is supported in the STASH master. This can be done in the STASH subroutine `addres` that loops through the items in the STASH sections. Each item will be checked against the active tracer list and a final check will then ensure that all active tracers are accounted for at the end. This will make the call to `tstmsk_ukca` below `addres` redundant. The UM will need to provide a lookup table for converting the names of supported UKCA fields to STASH codes.

Note that similar processing to the above will be required in the reconfiguration code (in the subroutine `rcf_address`) to determine which fields are to be included in the output dump.

In the initial implementation of the standalone code, `tstmsk_ukca` could still be employed to determine the list of active tracers within the `ukca_setup` call if a separate association between field names and option codes is provided by UKCA. This may be the quickest method to implement. However, replacing it with a more direct method like that currently used within UKCA for non-transported prognostics would make the code clearer and easier to maintain.

In the UM, two subroutines `ukca_set_nmspec` and `ukca_set_conv_indices` are called to set up UM-specific information for tracer access that is used within UKCA. `ukca_set_nmspec` sets up an array of field names `nm_spec` that correspond to STASH Section 34 item numbers and another array of field names `nm_spec_active` that corresponds to the UM array `tracer_ukca` used to hold UKCA-specific tracer fields. `ukca_set_conv_indices` sets up the indices of GLOMAP-mode fields in the `tracer_ukca` array (in `nmr_index_um` and `mmr_index_um`). Use of this UM-specific information within UKCA should be eliminated in the standalone code where practical but this may not always be the case (see details of convection processing below and in Section 4.3). Any such information that is required will need to be passed via argument lists.

The main requirement for the UM-specific data described above is the use of `nm_spec_active` to inform the copying of tracers between the UM's `tracer_ukca` array and UKCA's internal tracer array used within `ukca_main1`. In the standalone code, the tracer array passed to UKCA cannot be specific to a particular parent model, so this copying will be moved outside UKCA. The tracer array passed to UKCA will then be the array used internally.

It is less practical to eliminate the use of parent model specific indexing where UKCA-specific processing occurs outside the `ukca_main1` call, as required for plume scavenging. UKCA-specific processing of GLOMAP-mode tracers occurs in the UM-convection processing via calls to the UKCA subroutine `ukca_plume_scav`. Indexing of UKCA tracers within this call uses the UM tracer order (with an offset into a composite array of all scavenged tracers). The required mapping between parent tracer indices and UKCA field names is currently derived from `nm_spec_active`, accessed via a shared module. This will need to be replaced by passing similar information via a UKCA subroutine call. Changes to `ukca_plume_scav` are also needed because the processing is slightly different for two different UM convection schemes and needs to be re-packaged in a non-UM specific way for potential use by other parent models.

Including `ukca_plume_scav` in the standalone code is a lesser priority than implementation of the main time step routine since UKCA can run without plume scavenging. The processing may therefore remain the responsibility of the UM until actually required in another application.

4.3 Handling STASH Requests and Diagnostics

The UM needs to validate all STASH requests present in the Rose configuration file to ensure that the requested prognostics and/or diagnostics will be available in the UKCA configuration. Prognostics will always be available in the `ukca_step` input/output arrays, if used by the model. Diagnostics must be requested individually.

UM diagnostic requests will be consolidated (by the UM) 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 to avoid unnecessary work within `ukca_step`.

The validation of STASH requests is currently done by `tstmsk_ukca` during execution of the subroutine `prelim`, within `stash_proc`, which loops round all of the 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 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. The call to `tstmsk_ukca` below `prelim` will become redundant.

Calls to get the field lists for the prognostics will use information already prepared by `ukca_setup`. Calls to set the diagnostic requests must use the STASH system data to determine the required field name lists as requested via Rose. These lists should be based on the set of all diagnostic item requests with duplicates removed. A translation from STASH code to the corresponding field name, based on a UM lookup table, will be required to translate the original STASH requests into a list of field names.

In the initial implementation of the standalone code, `tstmsk_ukca` could still be used to check validity of diagnostic requests in the set/update diagnostic request subroutines, given an association of option codes with field names within UKCA. As for tracer setup, this is not an ideal long term solution.

Within `ukca_step`, the 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. Processing will need to be re-worked to use variable names rather than STASH codes and 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. The UM must then translate variable names to STASH codes, copy the fields in the 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 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.

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.

Plume scavenging diagnostics calculations require a tracer difference array as input. This requires capture of the difference in UKCA tracers before and after convection in the parent model (via calls to `ukca_plume_scav_initial` and `ukca_plume_scav_final`, see calling tree in Section 4 introduction). It is reasonable for this simple differencing to be the responsibility of the parent but the details of the calculation should be the responsibility of UKCA. However, the difference in the way the tracer array is ordered between the parent model and UKCA must be considered. If UKCA is to access a parent-specific tracer difference array (as is currently done within the `ukca_main1` call), the required mapping information should be passed to UKCA alongside the difference array. As already noted (Section 3.2), the diagnostic calculations use a vertical integration scheme that requires calling a parent subroutine.

The systematic replacement of diagnostic processing is a major task that is unlikely to be suitable for a single ticket. However, it should be possible to create a working standalone code for non-UM applications without replicating all UM diagnostics by disabling the processing for unsupported diagnostics using pre-processor directives. This would allow diagnostics to be added after the initial implementation phase as and when they are required.

4.4 Moving Input from External Files out of UKCA Time Step Subroutine

The main file input processing within UKCA is that associated with emissions. NetCDF access occurs in `ukca_emiss_init` and `ukca_emiss_update`. `ukca_emiss_init` loops through the list of NetCDF input files to count emission fields before setting up an emissions structure that is used as a reference database for the rest of the emissions processing. It then loops through the files again to add metadata to the data structure. These loops will need to be moved to a separate top-level emissions processing subroutine `ukca_set_emissions_from_nc` that can be called before `ukca_step`. The new subroutine will be responsible for setting up the emissions structure on its first call and reading the field data (at intervals specified in the metadata) by calling `ukca_emiss_update` on first and subsequent calls. The data structure will also include online emissions as before but these will still be processed within `ukca_main1`. `ukca_set_emissions_from_nc` will need to allow space for the online emissions when setting up the emissions structure. (This is possible because the number of online emissions required is fully determined by the UKCA configuration details.)

To support LFRic, data exchange between the parent and the UKCA emissions structure will need to be supported by new subroutines `ukca_get_emission_fields` and `ukca_set_emission_fields` (since LFRic will not allow internal persistence of fields). However, these are not required in the initial UM implementation. Other parents that do not use the present NetCDF-based system to retrieve emissions data from external files might use `ukca_set_emission_fields` instead of `ukca_set_emissions_from_nc`. This routine could eventually include functionality for setting up metadata in the emissions structure, via optional arguments, but this could be added as and when required without affecting backwards compatibility.

Other file-based input processing includes that for offline oxidants, various reference data sets as listed in Section 3.2 (including the FAST-JX specifications). Offline oxidants are treated in the same way as emissions and the NetCDF access will likewise be moved out of `ukca_main1` and be handled by a top-level subroutine `ukca_set_oxidants_from_nc`. Access to the other data sets will also need to be moved outside `ukca_main1` and handled as described in Section 3.3.

Implementation of the functionality to retrieve environment data from reference data sets is of relatively low priority and could remain the responsibility of the UM initially, with the UM calling `ukca_set_environment` and `ukca_set_gas_mixing_ratio` to pass the data to UKCA in the interim. To replicate existing functionality for 2-D photolysis with the retrieval remaining the responsibility of the UM, a subroutine to set/update these data directly would be required in place of `ukca_set_photolysis_rates_from_lat_ht`.

4.5 Error Handling

Fatal errors should terminate UKCA but not the parent model (since the parent may want to continue without running the offending UKCA configuration). Instead, the procedure name, a UKCA-specific error code and an error message will be passed back to the parent. The UM can use this information to call `ereport`. Although the transfer of responsibility for deciding whether to terminate to the parent is not required for the UM, the internal `ereport` calls will need to be replaced in all UKCA modules anyway so it is appropriate to introduce the new error handling scheme at the same time.

On encountering a potentially fatal error condition, a procedure will return control to the calling procedure after setting output arguments provided for the error information and calling `dr_hook` (or the equivalent generic routine) 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.

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

4.6 Replacing Internal UM Subroutine Calls

All calls to UM subroutines (see Section 3.2) must be systematically replaced by calls to the equivalent generic routines. Each generic subroutine will have its own module that could potentially be replaced in the build process, as indicated in Section 2, as an alternative to run-time assignment of parent handlers.

Generic UKCA internal subroutines will either accept parent model subroutines (or pointers to parent model subroutines) as arguments or use pointers declared in UKCA modules that will reference parent model subroutines. In either case, 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 and UKCA will use the labels to check availability of particular parent model subroutines.

In some cases, such as reading particular input data, the UM subroutine will be UKCA-specific. However, any UKCA-specific processing that it is practical to separate out should be moved to the UKCA internal subroutine calling it, so that other parent models can take advantage of it.

In other cases, such as printing to a log file, the appropriate UM subroutine will be general purpose. Such subroutines will not necessarily conform to a UKCA specification as defined by the new API. If this is the case, the UM will need to provide a UKCA-compatible wrapper and pass the wrapper subroutine to UKCA. For example, the UM might pass a subroutine called `um_print_handler` to UKCA (via `ukca_set_handler`). In UKCA, a subroutine `ukca_print` would call this handler which would then call `umPrint`.

A substitute for the UM `ereport` subroutine will only be required for non-fatal errors. This UKCA warning handler should be able to receive an optional parent model warning handler and call it if present or do nothing other than possibly reset the warning code if not. In the UM case, warnings would be printed by the UM warning handler via a call to `ereport`.

4.7 Giving Parent Control of Workspace Persistence

The workspace allocated internally should be reviewed and deallocation statements added if not already present. These and existing deallocation statements for such workspace will be put under the control of the parent via one or more switches. This is not required for the UM or for the initial standalone code but will be essential for LFRic where persistence of fields allocated for the model domain will not be possible.

4.8 Handling Configuration Data

At the end of the refactoring process, the only UKCA module used by the UM should be the UKCA API module and the final version of the API module should contain procedures and fixed parameters only. No UM modules should be used within UKCA.

The module `ukca_option_mod` containing the subroutine `read_nml_run_ukca` and the namelist variables will remain part of the UM (and should be renamed accordingly). `ukca_setup` will need to accept all namelist variables as optional input arguments and provide default values. These default values would probably be the same as or equivalent to the namelist defaults. However, this is not a strict requirement and they could be changed in future versions without reference to the UM.

The UM, in common with other parent applications, will require access to some configuration data other than that provided by the prognostic and diagnostic field name lists. In the UM, these could be obtained from the renamed `ukca_option_mod` module, however this would not be good practice. The configuration data held within UKCA define the actual configuration and these should be accessed instead. A review of the UM code (including GLOMAP_CLIM), excluding that now in modules that have been formally identified as UKCA modules, will be needed to determine exactly what data are required. The data can then be provided by API subroutines of the form `ukca_get_config_*`. Each such subroutine should return a logical group of variables. In choosing these, consideration should be given to what variables might be generally useful to other parent applications.

4.9 Test Harness for Testing Non-UM Functionality

Testing outside the UM will be required to ensure that the new UKCA code is truly independent. A basic test harness will be needed for setting up some simple tests. These should include tests that are not covered by the UM application, such as running with a fixed environment. 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.

Once the UKCA reset functionality has been implemented, standard tests should also 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 call to `ukca_reset` produces the same results.

Appendix: Provisional UM Tickets for Initial Standalone Code

A provisional list of UM tickets is given below for the priority changes that will create the initial standalone code. Suggested target UM release versions are given against each with the code submission review deadlines.

Initial re-factoring of UM-UKCA interface (#4367 for vn11.4, 31-MAY-2019)

Move D1 access and STASH handling out of UKCA and re-arrange initialisation code as indicated in Section 4.1. This includes separating UM-specific and UKCA code currently in `ukca_setd1defs` and separating UM-specific and UKCA processing of non-transported prognostics. Ensure early availability of NTP requirements to prepare for the verification of NTP output requests using UKCA data for reference rather than STASH. Add `ukca_setup` and `ukca_get_ntp_varlist`.

UKCA interface refactoring: environment fields (for vn11.4)

Add `ukca_set_environment` and `ukca_get_environment_varlist`. Use these to determine environment variables required and supply from D1.

UKCA interface refactoring: tracers (for vn11.4)

Add `ukca_get_tracer_varlist` and implement tracer handling as described in Section 4.2 so that the array of tracers passed to UKCA is independent of the UM. Exclude plume scavenging.

UKCA interface refactoring: emissions and reference data (for vn11.5, 04-OCT-2019)

Divide emissions processing between `ukca_set_emissions_from_nc` and `ukca_main1` as indicated in Section 4.4. Likewise, divide offline oxidants processing between `ukca_set_oxidants_from_nc` and `ukca_main1`. Remove reference data retrieval from UKCA and pass data to UKCA via `ukca_set_environment` and `ukca_set_gas_mixratio`. Exclude 2-D photolysis support.

UKCA interface refactoring: diagnostics (for vn11.5)

Implement the diagnostic handling scheme described in Section 4.3 with a focus on its mechanics rather than on making all diagnostics available outside the UM. Only a small range of diagnostics will be included initially. Others will be disabled by pre-processor directives unless built with the UM. Validation of UM diagnostic requests will use the new method for diagnostics available outside the UM and the old method for all other fields requested. Future tickets can add diagnostics as and when needed. Add `ukca_set_*_diagnostic_requests`, `ukca_update_*_diagnostic_requests` and `ukca_get_*_diagnostic_varlist`.

UKCA interface refactoring: configuration data and FAST-JX specs (for vn11.5)

Modify `ukca_setup` to set configuration data from namelist variables read by UM (see Section 4.8). Add `ukca_set_fastjx_specs_from_file`, `ukca_get_fastjx_specifications`, `ukca_set_fastjx_specifications`. Do final UM-side changes to remove all use of modules now designated as UKCA modules except API. Add `ukca_get_config_*` routines as needed.

UKCA interface refactoring: generic handlers (for vn11.5)

Add `ukca_set_handler` and generic UKCA routines (see Section 4.6) to replace `umPrint`, `ereport`, `tr_mix`, `trsrce`, `t_int` and NetCDF access subroutines in `emiss_io_mod`. Exclude wholesale replacement of `umPrint` and `ereport`: just do enough to test handler.

UKCA interface refactoring: `umPrint` and `ereport` replacement (for vn11.5)

Systematically replace `umPrint` and `ereport` throughout the designated UKCA modules, implementing fatal error handling as described in Section 4.5.

Standalone UKCA and test harness (for vn11.6, 31-JAN-2020)

Disable any remaining UM-specifics in UKCA modules when built outside the UM. Overload API subroutine interfaces as needed for 0-D, 1-D or 3-D domains. Construct a basic test harness to test main functionality (see Section 4.9) and ensure that UKCA works within this test harness independently of the UM.