NAME III Document
MD2/1(Version 2)
9/1/03

Code structure

David J. Thomson

1 Introduction

This document describes the overall structure of the code. The code structure uses a category-theory view of code design, using typed data structures (Fortran 90 derived types) and functions (Fortran 90 functions and subroutines) between these structures, with equal importance placed on these two aspects of the design. In keeping with the category-theoretic perspective, the types are generally treated as characterised by the functionality they support, and the type definitions and functions characterising the types are grouped together in Fortran 90 modules. To follow this philosophy strictly would require all the data items in the types to be declared with the 'private' attribute, but we have not done this everywhere — in effect this means we allow a direct reference to a data item as a short hand for what ought to be a function call. The absence of dynamic types and (categorical) coproducts of types in Fortran 90 means that the structure is not as simple as it could be — for example the various different types of met data cannot be designed to look the same and cannot be combined as elements of an array in situations where this would be desirable (i.e. those where the internal structure of the met data is irrelevant), at least without damaging the flexibility of the code. In the interest of maintaining flexibility for future design changes and code reuse, the design philosophy is prejudiced against use of 'saved' variables within routines and against use of instances of types other than constants (Fortran 90 parameters) declared globally within a module. In keeping with the category-theory perspective, we regard constants declared globally within a module as functions. For example π is regarded as the function $\pi : 1 \to R : ! \mapsto \pi$, where ! is the unique instance of the terminal type 1 and R is the set of reals (there is no single representation of the terminal type in Fortran 90, but in the C++ language the terminal type is the *void* type). Unlike in this example however, we don't always distinguish below between a data type and its value. For background information on category theory as applied to computer science, see e.g. Barr and Wells (1990), Walters (1991) and Crole (1993). The category theory approach has quite a lot in common with an object orientated approach; for the latter see e.g. Coad and Yourdan (1991).

2 Naming conventions

In this document we give full descriptive names to modules, data types, functions etc. However in many cases these names are abreviated in the code to avoid excessively long names. An exception is module names which are used in full in the code.

Within the code we adopt certain naming conventions.

- names of types end with an underscore
- variables giving the number of items in a collection start with n
- variables giving the index of an item in a collection start with i
- underscores are avoided (with capital and lower case letters being used instead to help make long variable names readable) with two exceptions:
  - parameters containing code numbers for things where the natural description is a character string (e.g. if 1 and 2 are the codes for lat-long and polar stereographic coord systems, it is convenient to introduce parameters with meaningful names which are set equal to these numbers); the names of these parameters take the form X_Y where X indicates the type of code (e.g. code for a horizontal coordinate system) and Y indicates the code meaning (e.g. latitude-longitude coord system)
  - names of 'set up' routines (discussed below) which take the form SetUpABC_XYZ where ABC are the items being set up and XYZ are any external infomation used

3 Units

SI units are used throughout except where this is clearly indicated. To avoid potential confusion, units are always given for pressure (usually Pa), for temperature (to distinguish K and °C), and for angles (to distinguish degrees and radians).

4 Cases, met modules, flow modules, calculation types and source strength dependencies

We discuss here some definitions, some general features of the code design, and some aspects of the range of calculations catered for.

Cases: In principle a run of the model consists of a series of 'cases'. For example one might consider an ensemble of realisations of the meteorology and wish to repeat the dispersion calculation for each. However in version 1 of NAME-PPM only one case will be allowed.

Met modules: Met data is input to the model via 'met modules'. A given met module may have multiple uses — for example the same module may read in global and mesoscale NWP data. This is made possible by ensuring that the module doesn't 'save' any data within its routines. To avoid confusion we will talk of a particular 'met module' and a particular 'met module instance' when we wish to distinguish between the module itself and a particular use of the module. A given met module instance takes as input a single source of met data, e.g. single site observations, mesoscale NWP data or radar rainfall data (possibly with supplementary data such as topographic height or roughness length — such information is regarded as part of the 'met data'), and makes it available to the flow modules (see below). A met module can perform some pre-processing of the data which is input (e.g. it might estimate boundary layer depth) but more extensive processing (e.g. estimating the flow around a building) is generally better performed in a flow module. Because each met module instance reads a single source of met data, any processing which requires information from more than one met source must be performed in a flow module.

Flow Modules: The information supplied by the met modules is processed further and supplied to the dispersion calculation via 'flow modules'. The processing might involve simply supplementing NWP data with turbulence estimates, or more extensive calculations such as calculating the flow around a building. As for met modules, a given flow module may be used more than once in a calculation, for example to provide both global and mesoscale NWP data or to calculate the flow around two different buildings. Again this is made possible by ensuring that the module doesn't 'save' any data within its routines and for clarity we will talk of a particular 'flow module' and a particular 'flow module instance' when a distinction is appropriate. A given flow module instance will get its input data from one or more met module instances and/or from other flow module instances and has a particular 'domain' associated with it which provides space, time and travel-time limits within which it supplies data. We will refer to the data supplied by the flow module instances to the dispersion calculation as flow information, although we are using the word 'flow' in a rather wide sense here to include e.g. cloud and rain.

The use of met and flow module in this way provides considerable flexibility, for example in using external or internal models to drive the dispersion calculation. The above discussion gives examples of both — namely external NWP models and internal flow-around-buildings models.

Calculation types and time types: There are four basic types of calculation which are catered for. These are

- calculations with time varying met/flow and an absolute time frame
- calculations with fixed met/flow and an absolute time frame
- calculations with time varying met/flow and a relative time frame
- calculations with fixed met/flow and a relative time frame.

The logical variables Fixed Met and Relative Time (in the Main Options data type) are used to indicate which of the four types of calculation is being carried out. Note that it would be possible in principle to have a varying flow with fixed met, for example if the flow module was a large eddy model (or even an NWP model) which was being run from fixed initial/boundary conditions. In the calculation types with fixed met/flow it is implied that both the met and flow are fixed. Calculations with fixed met and flow are often useful in short-range dispersion problems where the met/flow does not normally evolve significantly over the travel time from the source to receptor. In such cases one might wish to fix the met time at different values in the different cases. Calculations using a relative time frame are useful in situations where the time of the met data is not relevant or not known.

When the met/flow is fixed, the time variable Fixed Met Time (in the Main Options data type) can be used to specify the time at which the met and flow is fixed. However Fixed Met Time is not necessarily used by the met and flow modules — each module is responsible for its own method of determining the fixed met/flow (e.g. the module might take fixed met from a file or vary it with the case number).

The treatment of times in the four calculation types is a little involved. Four different types of time are used in specifying and performing the calculations, namely

- **type 1:** times which are defined absolutely
- **type 2:** times which are defined relative to some fixed but unknown reference time (which must be the same for all type 2 times)
- **type 3:** times which are defined relative to the release time of a particle or puff
- **type 4:** time differences.

These time types are discussed more fully when discussing the Time Module below. The time types used for the various times within the program vary according to the calculation type and are as follows:

	Time type for the case of			
Times describing or defining or associated with ...	Varying met/flow, absolute frame	Fixed met/flow, absolute frame	Varying met/flow, relative frame	Fixed met/flow, relative frame
the computational domain[1][2]	1 (absolute)	1 (absolute)	2 (relative)	2 (relative)
sources	1 (absolute)	1 (absolute)	2 (relative)	2 (relative)
receptors	1 (absolute)	1 (absolute)	2 (relative)	2 (relative)
met and flow module instances[1][3]	1 (absolute)	1 (absolute)	2 (relative)	2 (relative)
progress of the calculation	1 (absolute)	3 (rel to release)	2 (relative)	3 (rel to release)
particles and puffs	1 (absolute)	3 (rel to release)	2 (relative)	3 (rel to release)
maximum travel times[4]	3 (rel to release)	3 (rel to release)	3 (rel to release)	3 (rel to release)

[1] excluding times defining maximum travel times

[2] the computational domain is a separate domain which is not associated with a flow module instance and provides overall limits on the extent of the calculation

[3] times associated with the met and flow module instances include those defining the flow domains (excluding times defining maximum travel times), those describing the validity of the met and flow module instances, and those associated with the time of the met and flow, including the time variable Fixed Met Time (in the Main Options data type) which can be used to specify the time at which the met and flow is fixed (for fixed met/flow calculations)

[4] times defining maximum travel times are included in the definition of domains, and, through the domains, in each flow module instance

The data type used to store times keeps information on whether the time is of type 1, 2, 3 or 4. Infinite times constitute an exception — the data type used to store times can store positive and negative infinite times (useful e.g. for giving the start and stop times of a continuous source) but does not distinguish between infinite times of types 1, 2, 3 or 4.

If the met and flow are fixed, then the time limits (but not necessarily the travel time limits) associated with the flow domains must be $\pm\infty$. A consequence of this is that the validity of the met and flow module instances will never change (within one case) when the met and flow is fixed.

Other time information which is used internally by the met and flow modules, perhaps being read in as part of the met data, need not follow the types given in the above table. This could be complete time information or partial information (e.g. time of year or time of day). Such information will not however be made available outside the met and flow modules and it is up to these modules to ensure consistency as appropriate with the other time information in the model.

More on cases: In a run with multiple cases, each case will have a weight or frequency associated with it (which could be unity if the cases are all to have equal weight). Sometimes more detailed frequency information may be used to treat several different weightings in a single run, for example to represent the different frequencies of conditions for different seasons or different times of day. The frequency data is stored in a variable MetFr. Generally the frequency data will be input separately from the met data — however it can be useful to include the frequency data with the met data, especially in a run using only a single met module and single site data. Some, although not all, met modules will be able to handle frequency data in this way, and those that do will need to check for consistency to make sure multiple sources of frequency data do not lead to difficulties.

A number of things can vary from case to case. These are the met, the sources (through an offset to the time dependence) and the receptors (through an offset to their time locations). Also, for fixed met cases, the time of the met data (which is specified via the variable Fixed Met Time) can vary between cases – this provides an indirect route for the met to vary between cases. If detailed frequency data is used, the source strengths can take different values for each of the frequency weightings.

Source strength dependencies: As noted in passing above, the source strengths can depend on time, can vary between cases (via an offset to the time dependence), and can take different values for each of the detailed frequency weightings. In addition sources can have a dependence on the flow properties (e.g. dust sources might depend on wind speed and sources connected with heating may depend on temperature).

5 Input data items and collections of such items

Much of the input data used by the code is read in from 'the main set of headed input files'. The format of these files is described in detail in MD2/2. Here we consider some generic aspects of the way this data is stored in data types within the code. In this section we use upper and lower case symbols for types and their instances respectively where this distinction is important.

There are some data types Ψ where a number of instances of the type may be needed and where it is useful to have an additional data type $\hat{\Psi}$ which contains a collection of ψ's, possibly with some further information (such as the number of instances in the collection). There are also cases where the 'collective type' $\hat{\Psi}$ may contain collections of instances of several different types Ψ_1, Ψ_2 etc. As far as possible, we try to follow a similar code structure for all such cases when performing similar functions. The data types where this applies are as follows:

Ψ (or Ψ_1, Ψ_2 etc.)	$\hat{\Psi}$	Named Block?
Input File*	Input Files	No
Array	Arrays	Yes
Eta Definition	Eta Definitions	Yes
Horizontal Coordinate System	Coordinate Systems	No
Vertical Coordinate System	Coordinate Systems	No
Horizontal Grid	Grids	No
Vertical Grid	Grids	No
Temporal Grid	Grids	No
Domain	Domains	No
Specie	Species	No
Source	Sources	No
xx Requirement**	Requirements	No
xx Met Definition**	Met Definitions	No
xx Met**	Mets	No
xx Flow**	Flows	No
Flow Order	Flows	Yes
Flow Subset	Flows	Yes
Flow Attribute	Flows	No

* actually there is no Input File type — because the data stored is only a character string (the file name) we use a character string type instead; however the code is structured as if Input File was a separate type
** here x is wild — there are several different Requirement, Met Definition, Met and Flow data types corresponding to different sorts of requirement and different met and flow modules

These types are described in more detail in §6 below. The 'named block?' column refers to whether the data for the item which is contained in the 'headed input files' is read in from a named or unnamed block. This meaning of this is discussed in detail in MD2/2, but we note that items in unnamed blocks are read in with a single call while items in named blocks require a number of calls to read them in.

In all cases the instances of Ψ (or Ψ_i) are given names which enables the different instances to be distinguished and to be referred to by name from other parts of the code. We adopt different initialisation models for unnamed-block and named-block types. For the unnamed-block types we have the following routines:

- initialise $\hat{\psi}$: ! $\mapsto$ initialised $\hat{\psi}$ containing no ψ's [or ψ_i's]
- initialise ψ: (name of ψ, other information needed to initialise ψ) $\mapsto$ initialised ψ
- add ψ: (ψ, $\hat{\psi}$) $\mapsto$ $\hat{\psi}$ with ψ added
- find ψ index: (name of ψ, $\hat{\psi}$) $\mapsto$ index of ψ in $\hat{\psi}$

For the named-block types we have the following routines:

- initialise $\hat{\psi}$: ! $\mapsto$ initialised $\hat{\psi}$ containing no ψ's [or ψ_i's]
- add to ψ: (name of ψ, information to be added to ψ, $\hat{\psi}$) $\mapsto$ $\hat{\psi}$ with information added to the named ψ [if the named ψ doesn't exist within $\hat{\psi}$ it is created there]
- find ψ index: (name of ψ, $\hat{\psi}$) $\mapsto$ index of ψ in $\hat{\psi}$

The 'find ψ index' routine is not always needed (and is then omitted). For the named-block types we could in principle have routines to initialise and to add to an instance of Ψ outside of an instance of the collective data type $\hat{\Psi}$, but it turns out that in practice there is no role for such routines.

In addition to the data types listed in the table above, three further data types (Main Options, Dispersion Options and Output Options) are used in storing the data read from the main set of headed input files. However there is only one instance of each of these types and no corresponding collective types. These data types also have their own 'initialise ψ' routines. In addition they have 'pre-initialise ψ' routines for marking the data type as uninitialised. The reason for this is that otherwise it is impossible (i) to check that they have indeed been initialised and (ii) to trap any attempt to initialise them a second time (this contrasts with the case considered above where there is a collective type $\hat{\psi}$ which keeps track of how many ψ's have been initialised). For these types we therefore have the following routines:

- pre-initialise ψ: ! $\mapsto$ ψ marked as uninitialised
- initialise ψ: (uninitialised ψ, information to initialise ψ) $\mapsto$ initialised ψ

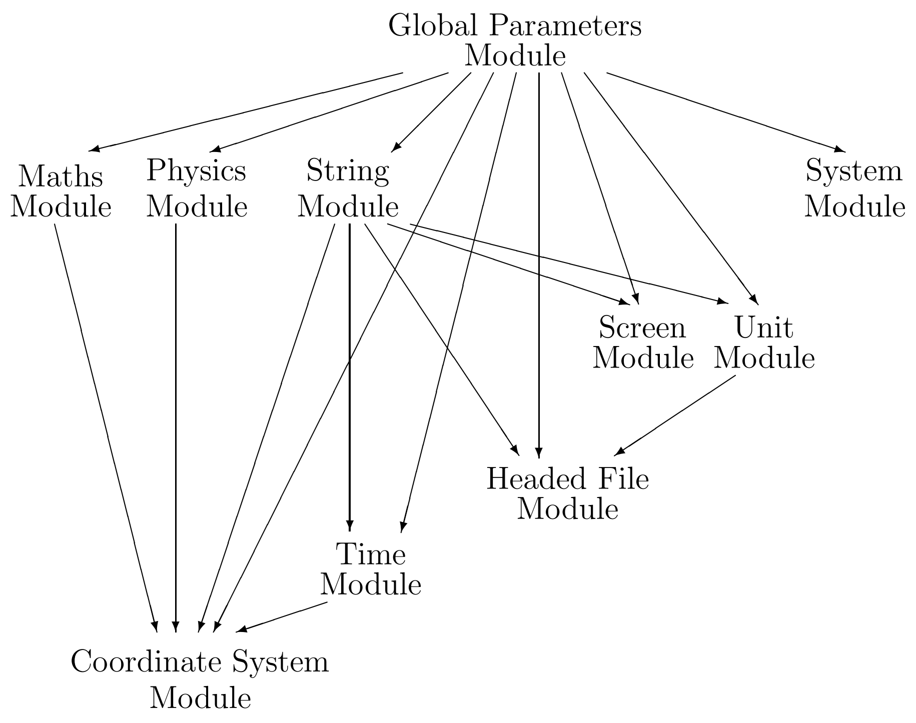
Once the data has been read in, there is a need for some data types to obtain information from or supply information to other data types in order to complete the preparation of the data types for the computation. In some cases this is done at the initialisation time, in which case the data types which are supplying the information need to be initialised first. In other cases this is done later in various ‘setup’ routines. Some of the data type instances used for inputting data from the main set of headed input files are used only as temporary stores of information which are then used in initialising or setting up other types. Those used only for initialising and/or setting up other types are Arrays, Eta Definitions, Domains and Met Definitions. Input Files is only used during the reading of the main set of headed input files.

During and/or after the initialisation and setting up of the data types, the data is checked as far as possible for consistency.

6 Modules, data types and functions

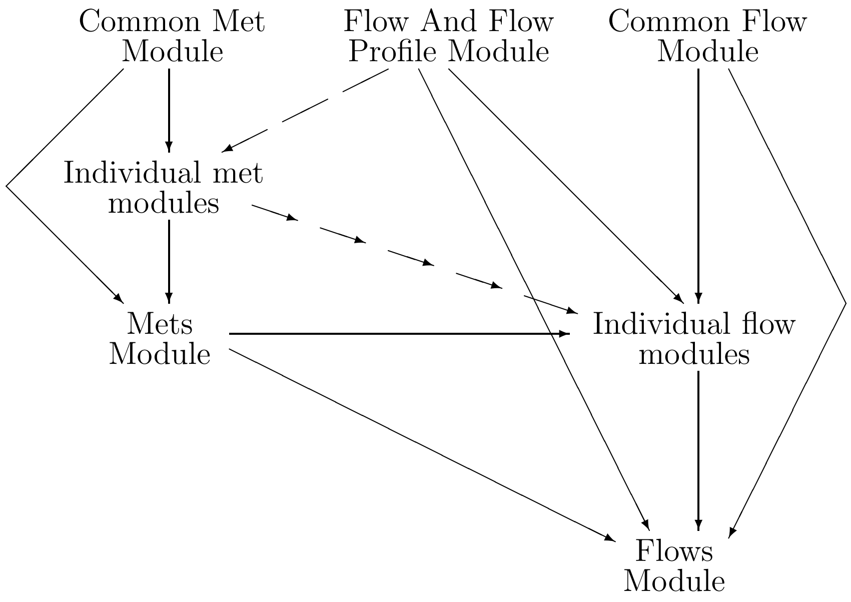
In this section we discuss the various modules used and the (public) data types and functions they contain. In line with the philosophy outlined in §1, all items in modules are regarded as either data types or functions.

The following diagram shows the modules which provide basic support services — in this diagram an arrow $A \longrightarrow B$ indicate that module A is used in module B.

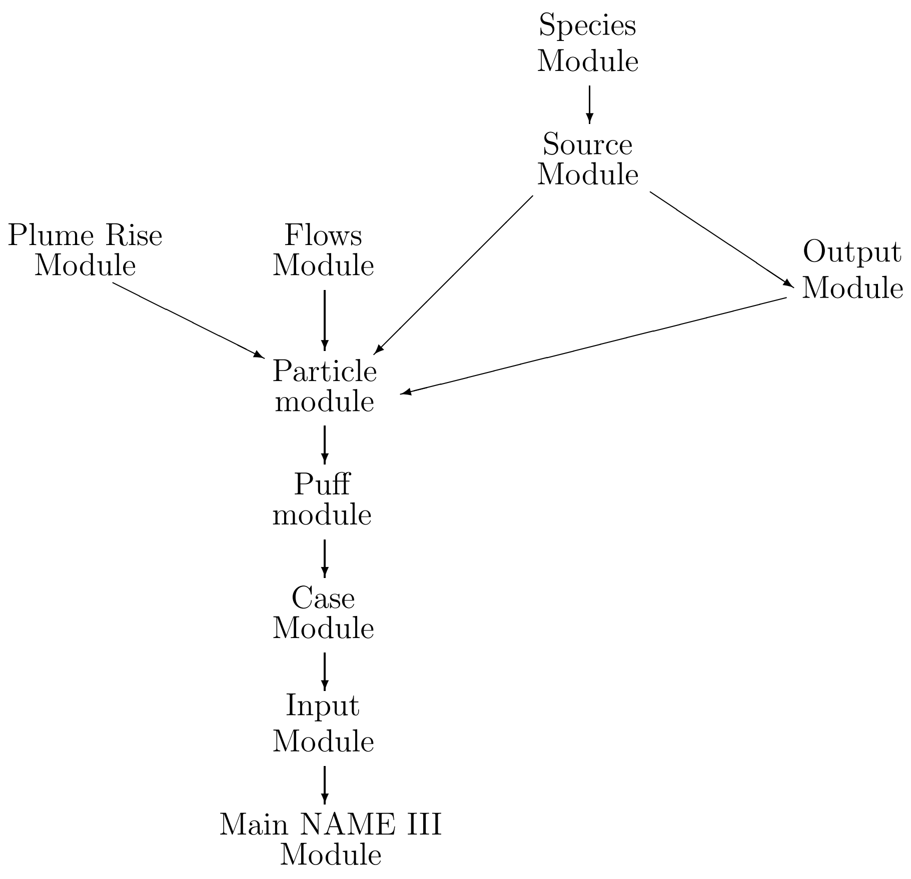


For convenience these modules are all combined into a single module, called the Service Module.

The flow and met modules form a complex cluster of modules. In the case of the met modules we have the Common Met Module which contains code common to all met modules, the individual met modules themselves (which might be internally structured as more than one module, but we don't consider this here) and the Mets Module which unifies the individual met modules so that they present a common interface to the rest of the code. The flow modules follow a similar pattern, but with the additional complication of the Flow And Flow Profile Module which provides support for transferring flow information from the flow modules to the dispersion calculation, for calculating boundary layer profiles and for allowing one individual flow module to obtain flow information from another. This is described in more detail when the modules are discussed below. The following diagram summarises the met and flow modules and their use of each other. The dashed lines are module uses which only apply to some of the individual met and flow modules. The individual met modules do not use each other and neither do the individual flow modules. All the modules in the diagram make use of the basic support services modules via the Service Module (but for clarity this is omitted from the diagram).



The following diagram summarises the remaining modules and how they relate to the met and flow modules. They all make use of the basic support services modules via the Service Module.



6.1 Global Parameters Module

This module provides code which defines various widely used parameters and checks them for consistency.

Functions:

- Parameters of the following types (there are too many parameters for it to be sensible to list them all here):
  - precision parameters (parameters for two types of real variable are defined to allow the possibility of altering the precision of horizontal coordinates independently from other real variables)
  - parameters defining Fortran unit numbers for input and output
  - lengths for various character strings
  - parameters for headed files and for input via the command line argument list
  - parameters associated with times
  - maximum number of various items (e.g. maximum horizontal coordinate systems allowed).
- *Check Global Parameters*: checks the parameters for consistency.

6.2 Maths Module

This module contains mathematical constants and functions.

Functions:

- *pi*: $1 \longrightarrow \pi$ in standard precision
- *pi Pos*: $1 \longrightarrow \pi$ in precision used for horizontal coordinates
- *Gauss*: $1 \longrightarrow$ Gaussian random number
- *erf*: $x \longrightarrow \mathrm{erf}(x)$
- *ATan2 with zero test*: $y, x \longrightarrow \begin{cases} 0 & \text{if } x = y = 0 \\ \mathrm{ATan2}(y, x) & \text{otherwise} \end{cases}$, where ATan2 is the Fortran function of that name.

6.3 Physics Module

This module contains physical constants and functions.

Functions:

- General physical constants:
  - *Gravity*: $1 \longrightarrow$ acceleration due to gravity
  - *Gas Constant*: $1 \longrightarrow$ gas constant for dry air
  - *Cp*: $1 \longrightarrow$ specific heat capacity for dry air
  - *Earth Radius*: $1 \longrightarrow$ Earth radius
  - *VK*: $1 \longrightarrow$ von Karman's constant
  - *Mole Mass Air*: $1 \longrightarrow$ molecular mass of dry air (g/mole)
  - *Mole Mass Water*: $1 \longrightarrow$ molecular mass of water (g/mole)
  - *TK At TC = 0*: $1 \longrightarrow$ thermodynamic temperature at 0 degrees Celsius (K).
- Constants for the ICAO standard atmosphere:
  - *T At 0km*: $1 \longrightarrow$ temperature at 0km above mean sea level (K)
  - *T At 11km*: $1 \longrightarrow$ temperature at 11km above mean sea level (K)
  - *T At 20km*: $1 \longrightarrow$ temperature at 20km above mean sea level (K)
  - *Lapse Rate 0 To 11km*: $1 \longrightarrow$ lapse rate from 0 to 11km above mean sea level
  - *Lapse Rate 20km Plus*: $1 \longrightarrow$ lapse rate at more than 20km above mean sea level
  - *P At 0km*: $1 \longrightarrow$ pressure at mean sea level (Pa)

- Reference values used in definitions:
  - *P Ref*: 1 $\longrightarrow$ reference pressure for potential temperature (Pa)
- Non-SI units (in SI units):
  - *Foot*: 1 $\longrightarrow$ one foot
- Humidity functions:
  - *CalcQ*: relative humidity, temperature, pressure $\longrightarrow$ specific humidity
  - *CalcRH*: specific humidity, temperature, pressure $\longrightarrow$ relative humidity
  - *CalcSatVapP*: temperature $\longrightarrow$ saturation vapour pressure
  - *CalcMixingRatio*: pressure, vapour pressure $\longrightarrow$ humidity mixing ratio

6.4 String Module

This module provides code for handling character strings and converting numbers to and from character strings in particular formats.

6.5 System Module

This module provdes code for interactions with the operating system and presents a consistent platform-independent interface for such interactions to the rest of the code.

Functions:

- *get command line arguments*: 1 $\longrightarrow$ number of command line arguments, the command line arguments
- *submit system command*: system command $\longrightarrow$ 1 [with the function submitting the system command]

6.6 Screen Module

This module provides code for producing screen output, including graphics.

6.7 Unit Module

This module provides code for handling Fortran unit numbers for input and output.

Data types:

- *units*: a collection of information on input/output unit numbers

Functions:

- *Init Units*: 1 $\longrightarrow$ initialised *units*
- *Get New Unit*: keep flag, *units* $\longrightarrow$ new unit number, *units* with the unit number marked according to the keep flag [the keep flag affects the behaviour of later calls to *Close Unit* or *Close Units*]
- *Close Unit*: unit number, *units* $\longrightarrow$ *units* with the specified unit number closed unless the unit number was obtained using *Get New Unit* with the keep flag set
- *Close Units*: *units* $\longrightarrow$ *units* with all unit numbers closed except for unit numbers that were obtained using *Get New Unit* with the keep flag set

6.8 Headed File Module

This module provides code for reading headed files.

Note headed files are described in more detail in the NAME III document 'Input' (document MD2/2).

6.9 Time Module

This module provides code for handling time.

Data types:

- *time*:

Functions:

- $\geq$:

6.10 Coordinate System Module

This module provides code for handling coordinate systems, grids and domains. We consider these three aspects separately.

6.10.1 Coordinate systems

Data types:

- *horizontal coord system*: information defining the coordinate system. Possible coordinate systems include latitude-longitude based systems with arbitrary orientation, Cartesian and polar coordinate systems defined on a polar stereographic projection, and Cartesian and polar coordinate systems defined on a transverse Mercator projection
- *vertical coord system*: information defining the coordinate system. Possible coordinate systems include height above ground, height above terrain, pressure, pressure converted to height above sea level using the ICAO standard atmosphere, pressure based eta coordinate, and height based eta coordinate.
- *coordinate systems*: a collection of coordinate systems

Functions:

- initialise horizontal coordinate system: information defining a horizontal coordinate system $\longrightarrow$ horizontal coordinate system
- initialise vertical coordinate system: information defining a vertical coordinate system $\longrightarrow$ vertical coordinate system
- convert horizontal coordinates: horizontal coordinate system 1, horizontal coordinate system 2, coordinates in horizontal coordinate system 1 $\longrightarrow$ coordinates in horizontal coordinate system 2
- convert height-based vertical coordinate: height-based vertical coordinate system 1, height-based vertical coordinate system 2, coordinates in height-based vertical coordinate system 1, topographic height $\longrightarrow$ coordinates in height-based vertical coordinate system 2
- convert pressure-based vertical coordinate: pressure-based vertical coordinate system 1, pressure-based vertical coordinate system 2, coordinates in pressure-based vertical coordinate system 1, surface pressure $\longrightarrow$ coordinates in pressure-based vertical coordinate system 2
- convert to a horizontal coordinate system: coordinate systems, index of a horizontal coord system, multi-coordinate position $\longrightarrow$ multi-coordinate position with the coordinates in the coordinate system with the given index computed

Note there are no ‘convert vertical coordinate’ and ‘convert to a vertical coordinate system’ functions (analogous to the ‘convert horizontal coordinates’ and ‘convert to a horizontal coordinate system’ functions) because these require knowledge of the flow and so are handled by the flow modules. Routines are provided however (‘convert height-based vertical coordinate’, ‘convert pressure-based vertical coordinate’, ...) to assist in the writing of such routines.

6.10.2 Grids

Data types:

- grid: information defining a grid of points
- grids: a collection of grids

Functions:

- initialise grid: information defining a grid $\longrightarrow$ grid.
- initialise grids: 1 $\longrightarrow$ grids. Initialises an instance of *grids* with no consituent grids.
- add grid: grids, grid $\longrightarrow$ grids. Adds a new grid to *grids*.

6.10.3 Domains

Note a point is regarded as being within a domain if it is within the domain according to the highest priority convert routine for which (according to that routine) the point is in.

Data types:

- time boundary: information defining a range of times of interest (e.g. duration of simulation, duration covered by a set of met data).
- boundary: information defining a spatial range of interest, time boundary.
- domain: domain type (natural or otherwise — natural domains have a terrain height associated with them whereas other domains will take their terrain height from a coarser natural domain — an example of a non-natural domain might be one used to calculate flow around a building), domain name (e.g. mesoscale, hill 1, building 2), boundary of domain, name of flow module (to be used in the domain), index indicating the set of dispersion module options to be used in the domain, information on whether the domain is currently active (i.e. whether the time is within the time boundary and the domain has all the information it needs to be fully functional — e.g. the associated flow module used in the domain must be functional and have access to any met data it needs).
- all domains: the collection of all the domains, blending height scales (these may be necessary if different overlapping domains have different terrain heights), boundary of the computational domain. The computational domain defined here in terms of its boundary should lie entirely within the 'global domain' introduced below. However we don't check this (it is hard to check whether two domains overlap if they are defined in different coordinate systems) and, if it is not satisfied, the actual computational domain is limited by both the computational domain defined here and the global domain. This type has only one instance.

- domains: a subset of the domains in (the one instance of) *all domains* together with a total order on these domains indicating which domains are coarser/finer than which. The data is stored as as an ordered list of indices pointing to the domains in *all domains*. This data type is used to indicate which domains are to be used for a particular source (e.g. one may wish to avoid treating the effects of a building on some distant sources but not on other nearby sources) and a particular instance of the data type is used to indicate which domains are to be used in defining the 'best value' of terrain height (the best value of terrain height is defined as the value given according to the finest of the natural domains [out of those in the particular instance of *domains*] which includes the location being considered). In all instances of *domains* the coarsest domain must be the same and must be a natural domain. We call this domain the global domain.

Functions:

- initialise boundary.
- initialise domain: domain type, domain name, boundary, flow module name, index to dispersion module options $\longrightarrow$ domain.
- horizontal distance to edge: horizontal coords, horizontal coord system, domain $\longrightarrow$ distance to edge of domain.
- which domain: location, all domains, domains $\longrightarrow$ domain, domain. Returns the finest active domain at the location and the finest active natural domain, the choice of domains being restricted to those in domains.

Note the treatment of boundaries requires care in implementation. Conceptually we can regard domains as including the boundaries (so that touching domains overlap) or not including the boundaries (so that touching domains do not overlap, with their common boundary belonging to another domain). In both cases one can't have domains which don't overlap but which have no "gap" (i.e. points not belonging to either domain) between them.

Terrain height is defined in every natural domain, but doesn't necessarily agree in overlapping domains. For any two natural domains n_1 and n_2 which overlap and have n_1 coarser than n_2 we define a blending height relating the two domains. This enables us to identify surface locations in the two domains (even if terrain heights are different) but also identify points high in the atmosphere which have equal heights above sea level. The terrain height data itself is not included in the data types in the domain module but is treated as a meteorological variable and handled by the flow and met modules.

6.11 Service Module

This module provides a route through which basic support services from a number of modules are made available.

All the public data types and functions from the following modules are made available:

- Global Parameters Module
- Maths Module
- Physics Module
- String Module
- System Module
- Screen Module
- Unit Module
- Headed File Module
- Time Module
- Coordinate System Module

6.12 Species Module

6.13 Source Module

Data types:

- source: source characteristics, domains
- sources: collection of sources

Functions:

- initialise: information on a source $\longrightarrow$ source
- update source: source, time, $\{$concentration fields, grid$\}$ $\longrightarrow$ source

6.14 Output Module

Data types:

- requirements: specification of what output is required, including whether to use output box averaging or kernels when particles are in use

- output: collection of output data

Defined relative to terrain height?

Points in buildings give average concentration over building surface.

6.15 Flow and Flow Profile Module

6.16 Met Modules

As noted above, the model structure supports use of a variety of met modules, each module handling only one data source.

Data types (generally these types are different for different met modules):

- met state: internal state of met module. This consists of the name of met module instance, the time boundary within which the met data is currently valid, information to identify the met source and how to process it, and met data.

Functions (generally these types are different for different met modules):

- initialise met state: name of met module instance, information to identify met source and how to process it $\longrightarrow$ met state
- update met: met state, time $\longrightarrow$ met state. Updates met state so as to be valid at the current time if possible.
- met valid: met state, time $\longrightarrow$ flag indicating if the met is valid at the time specified, earliest time at which the met may next become invalid.

6.16.1 Particular Met Modules

A number of met modules are provided:

- Prototype met module: This is a very simple module for testing purposes. It reads a number of met parameters from a file to characterise very simply the vertical profile of the meteorology in horizontally homogeneous conditions.
- Single site met module:
- NWP met module:
- Radar met module:

6.17 Flow Modules

As noted above, the model structure supports use of a variety of flow modules which may use data from met module instances or may use data from other flow module instances, or may use both. Formally there is no link between the coarser/finer order of domains and which domains the flow module might get data from, but usually of course it will be from coarser domains.

Data types (generally these types are different for different flow modules):

- flow state: internal state of a flow module instance. This consists of the name of the domain the flow module instance is being applied to (because flow module instances are linked bijectively with domains, we will identify them by the name of the domain), the time boundary within which the flow module instance is currently valid, information indicating the met modules and met module instances which can be used as sources of data, information on which (other) flow module instances (identified by their domain name) can be used as sources of data, and any information needed to control the module.

Functions (generally these types are different for different flow modules):

- initialise flow state: domain, information on data sources for the module, any information needed to control the module $\longrightarrow$ flow state.
- update flow: domain, flow state, met data that might be used as a data source, flow state information for flow module instances which might be used as a data source, all domains $\longrightarrow$ flow state. Updates flow state using the latest available input data.
- flow valid: flow state, time $\longrightarrow$ flag indicating if the flow is valid at the time specified, earliest time at which the flow may next become invalid.
- flow information: flow state, location, time $\longrightarrow$ flow information. This is a generic class of functions for returning a variety of information about the flow at a particular location, e.g. mean velocity, standard deviation of vertical velocity, or some collection of such information.

Flow information (including terrain height) should agree on the domain boundaries where appropriate. However the model cannot ensure this or easily test for it — it requires consistency of input data.

6.17.1 Particular Flow Modules

A number of flow modules are provided:

- Prototype flow module: This is a very simple module for testing purposes. It uses the data supplied by the prototype met module to construct a horizontally homogeneous flow.
- Single site flow module:
- NWP flow module: This is a flow module designed to interpolate gridded NWP data and add turbulence information.
- Building flow module:
- Radar met module:

6.18 Plume Rise Module

6.19 Particle Module

The following structures are used within the particle module

Data types:

- particle: particle characteristics, source name
- particles: collection of particles

Functions:

- initialise particles: particles, sources, time $\longrightarrow$ particles
- concentration: particles, grid $\longrightarrow$ instantaneous concentration field on grid
- update concentration: particles, output, requirements $\longrightarrow$ output

6.20 Puff Module

6.21 Case Module

As noted above, the model structure allows for the presence of a variety of dispersion modules, although only one is used in any one case.

Data types:

- dispersion state: internal state of dispersion module.

Functions:

- initialise dispersion state: dispersion options (e.g. how long or far to use $(\mathbf{x}, \mathbf{u})$ model, how long or far to use puff model, options for detemining puff scale Δ, release interval for continuous sources) $\longrightarrow$ dispersion state.
- disperse: dispersion state, domains, met states, flow states, sources, requirements, output $\longrightarrow$ dispersion state, output

6.22 Input Module

This module provides code for coordinating input from the command line arguments and from the main set of headed input files.

6.23 Main NAME III Module

7 The overall program structure

- Set up run info needed to characterise run, i.e. all domains, flow state (for each flow module instance), met state (for each met module instance), dispersion state (for each dispersion module), sources, output requirements, various miscellaneous variables (fixed met and flat earth flags).
- Check run info is sensible.
- Do cases:
  - Update run info for case
  - Calculate case:
    - Evaluate start time
    - Do until case complete
      - Check whether met and flow module instances are currently valid, and update if necessary
      - Evaluate the next time t when a met or flow module or a domain may change its validity
      - Call dispersion module to evaluate dispersion up to time t
  - Output information needed from case

  Output information needed from ensemble of cases

8 Structure of the dispersion calculations

- Loop over time with large time steps (chosen to avoid exceeding time when a met or flow module instance might change its validity, and any requirements internal to the dispersion module)
  - Loop over particles/puffs
    - Loop over time with small situation dependent time steps
      - Initialise particle/puffs as required using source information
      - Check which domain the particle/puff is in
      - Evolve particle/puff position, including plume rise effects
      - Evolve particle/puff state to account for deposition, radioactive-decay*
      - Calculate contribution to output*
      - If needed, calculate contribution to instantaneous concentration field for chemistry calcultions
    - End Loop
    - Chemistry
  - End Loop
- End Loop

* These items could go outside the innermost loop.

9 Pre-processing code for Mets and Flows modules

In order to enable the Mets and Flows modules to easily cope with a variety of particular met and flow modules, the code for the Mets and Flows modules are kept in .P90 files which are pre-processed to produce .F90 files. The code needed to do the preprocessing (which was written as part of the NAME III project) responds to various '$-directives' which instruct the pre-processor to repeat certain blocks of code with various changes.

References

Barr M. and Wells C., 1990, 'Category theory for computing science', Prentice Hall.

Coad P. and Yourdan E., 1991, 'Object-orientated analysis', Prentice Hall.

Crole R.L., 1991, 'Categories for types', Cambridge University Press.

Walters R.F.C., 1991, 'Categories and computer science', Cambridge University Press.