6. Required Libraries

The CMAQ programs require a set of third-party libraries that must be installed on the users system before CMAQ can be compiled and run. These libraries control the data flow through CMAQ, define the binary file formats used by the CMAQ input and output files, and control how CMAQ functions in a multiple-processor computing environment. The Input/Output Applications Programming Interface (I/O API) and the Network Common Data Form (netCDF) are required for all applications of CMAQ. The Message Passing Interface (MPI) is only required for multiple-processor applications of CCTM. Brief descriptions of these three libraries are provided in this section. For additional information, including how to compile and configure these libraries, refer to the documentation associated with each library.

Input/Output Applications Programming Interface (I/O API)

The Models-3 Input/Output Applications Programming Interface (I/O API) is an environmental software development library that provides an interface with the data involved in CMAQ applications (Coats, 2005). The I/O API is the core input/output framework of the CMAQ programs, providing a set of commonly used subroutines for passing information between source code modules and for reading and writing data files. Users should download the latest code for the I/O API from the website. In addition to providing the input/output framework for CMAQ, the I/O

API forms part of the binary file format used by the CMAQ programs. ***Starting with CMAQ version 5.0, the I/O API version 3.1 or newer is required to compile and run CMAQ.***

The CMAQ input and output files use a hybrid Network Common Data Form (netCDF)-I/O API file format. The netCDF is described below. The CMAQ data files all use the netCDF convention of self-describing, selective direct access, meaning the modeling system can be more efficient by reading only the necessary parts of the data files. Additionally, netCDF files are portable across computing platforms. This means that the same file can be read, for example, on a Sun workstation, a Red Hat Linux workstation, and on Mac OSX. The I/O API component of the file format is the way that spatial information is defined in the CMAQ data files. The I/O API convention of defining horizontal grids is to use a combination of the map projection and an offset from the projection center to the southwest corner of the modeling domain. After defining the southwest corner of the domain, or the “offset” from the projection center, the I/O API grid definition specifies the size of the horizontal grid cells and the number of cells in the X and Y directions. An additional benefit of the I/O API is that an expansive set of data manipulation utilities and statistical analysis programs is available to evaluate and postprocess the binary CMAQ input/output data files.

For CMAQ users using preconfigured applications of the model, the I/O API system can be essentially transparent. For users who plan to modify the code or implement updated modules for research purposes, a few key elements of the I/O API should be understood, and they are discussed below. This section covers only the barest of necessities in terms of a CMAQ user’s interaction with I/O API. For more detailed information about developing new modules for CMAQ using the I/O API code libraries, please refer to the I/O API User's Manual.

Files, Logical Names, and Physical Names

The I/O API stores and retrieves data using files and virtual files, which have (optionally) multiple time steps of multiple layers of multiple variables. Files are formatted internally so that they are machine- and network-independent. This behavior is unlike Fortran files, whose internal formats are platform-specific, which means that the files do not transfer using the File Transfer Protocol (FTP) or Network File System (NFS)-mount very well. Each I/O API file has an internal description, consisting of the file type, the grid and coordinate descriptions, and a set of descriptions for the file variables (i.e., names, unit specifications, and text descriptions). According to the I/O API format, files and variables are referred to by names, layers are referred to by numbers (from 1 to the greatest number of layers in the file), and dates and times are stored as integers, using the coding formats *YYYYDDD* (commonly called “JDATE”) and *HHMMSS* (commonly called “JTIME”), where

`YYYYDAY = (1000 * Year) + Julian Day`

`HHMMSS = (10000 * Hour) + (100 * Minute) + Seconds`

Rather than forcing the programmer and program-user to deal with hard-coded file names or hard-coded unit numbers, the I/O API utilizes the concept of logical file names. The modelers can define the logical names as properties of a program, and then at run-time the logical names can be linked to the actual file name using environment variables. For programming purposes, the only limitations are that file names cannot contain blank spaces and must be at most 16 characters long. When a modeler runs a program that uses the I/O API, environment variables must be used to set the values for the program’s logical file names. Additional details of how the CMAQ programs use I/O API environment variables are discussed in Chapter 7. The remainder of this section explains some of the rudimentary details of programming in an environment using I/O API data files.

I/O API Data Structure and Data File Types

Each CMAQ data file has internal file descriptions that contain the file type, the file start date and time, the file time step, the grid and coordinate descriptions, and a set of descriptions for the set of variables contained within the file (i.e., names, units specifications, and text descriptions). Some of the elements in a file description, such as the dates and times for file creation and update and the name of the program that created the file, are maintained automatically by the I/O API. The remainder of the descriptive information must be provided at the time of file creation.

All files manipulated by the I/O API may have multiple variables and multiple layers. Each file also has a time-step structure that is shared by all of its variables. There are three kinds of time-step structure supported (Table 6-1). Within a file, all the variables are data arrays with the same dimensions, number of layers, and data structure type, although possibly different basic types (e.g., gridded and boundary variables cannot be mixed within the same file, but real and integer variables can). The data type structures that are supported are listed in Table 6-2. GRDDED3 and BNDARY3 are the most prevalent file types in a CMAQ simulation. Magic number is an indicator associated with the files type.

Table 6-1. Possible Time Step Structures in I/O API Files

File Type	Description
Time-independent	The file's time-step attribute is set to zero. Routines that use time-independent files ignore the date and time arguments.

Time-stepped	The file has a starting date, a starting time, and a positive time step. Read and write requests must be for some positive integer multiple of the time step from the starting date and time.
Circular-buffer	This type of file keeps only two “records”, the “even” part and the “odd” part (useful, for example, for “restart” files where only the last data written in the file are used). The file’s description has a starting date, a starting time, and a negative time step (set to the negative of the actual time step). Read and write requests must be for some positive integer multiple of the time step from the starting date and time, and they must reflect a specific time step that is in the file.

Table 6-2. Possible Data Type Structures in I/O API Files

File Type	Magic Number	Data Type	Description
CUSTOM3	-1	Custom	User-dimensioned array of REAL*4s that the system reads/writes reliably
DCTNRY3	0	Dictionary	Data type stores and retrieves parts of an FDESC.EXT file description
GRDDED3	1	Gridded	Dimension as REAL*4 ARRAY (NCOLS, NROWS, NLAYS, NVARS)
BNDARY3	2	Boundary	Dimension as REAL*4 ARRAY (SIZE, NLAYS, NVARS)
IDDATA3	3	ID-reference	Used to store lists of data, such as pollution monitoring observations
		Vertical	Used to store lists of

PROFIL3	4	profile	vertical data, such as rawinsonde observations
GRNEST3	5	Nested grid	Preliminary and experimental implementation for storing multiple grids, which need not in fact have any particular relationship with each other beyond using the same coordinate system
SMATRX3	6	Sparse matrix	Sparse matrix data, which uses a "skyline-transpose" representation for sparse matrices, such as those found in SMOKE
KFEVNT3	-3	Cloud event	KF-Cloud files use the same file description data structures (from FDESC3.EXT) and defining parameters (from PARMS3.EXT); the usual I/O API DESC3() call may be used to retrieve file descriptions from the headers. KF-Cloud files, on the other hand, have their own specialized opening/creation, look-up/indexing, input, and output operations
TSRIES3	7	Hydrology Time Series	A hydrology time series file behaves much like a degenerate gridded file, except that the numbers of rows and columns are usually 1, and that there are additional file attributes

			found in the INCLUDE file ATDSC3.EXT
PTRFLY3	8	Pointer-flyer	A pointer-flyer observation file behaves much like a degenerate gridded file with NCOLS3D and NROWS3D set to 1, and certain mandatory variables and variable-naming conventions to be used by analysis and visualization software

Opening/Creating Data Files in I/O API

The I/O API function `OPEN3` is used to open both new and existing files. `OPEN3` is a Fortran logical function that returns TRUE when it succeeds and FALSE when it fails.

```
LOGICAL FUNCTION OPEN3( FNAME, FSTATUS, PGNAME )
```

where:

```
FNAME (CHARACTER) = file name for query
FSTATUS (INTEGER) = see possible values in Table 6-3
PGNAME (CHARACTER) = name of calling program
```

`OPEN3` maintains considerable audit trail information in the file header automatically, and automates various logging activities. The arguments to `OPEN3` are the name of the file, an integer FSTATUS indicating the type of open operation, and the caller's name for logging and audit-trail purposes. `OPEN3` can be called many times for the same file. FSTATUS values are defined for CMAQ in PARMS3.EXT and are also listed in Table 6-3.

Table 6-3. Possible values for OPEN(3) FSTATUS

FSTATUS	Value	Description
FSREAD3	1	for READ-ONLY access to an existing file
FSRDWR3	2	for READ/WRITE/UPDATE access to an existing file
FSNEW3	3	for READ/WRITE access to create a new file (file must not yet exist)
FSUNKN3	4	for READ/WRITE/UPDATE access to a file whose existence is unknown
FSCREA3	5	for CREATE/TRUNCATE/READ/WRITE access to files

In the last three cases, “new” “unknown” and “create/truncate,” the code developer may fill in the file description from the INCLUDE file FDESC3.EXT to define the structure for the file, and then call `OPEN3`. If the file does not exist in either of these cases, `OPEN3` will use the information to create a new file according to your specifications, and open it for read/write access. In the “unknown” case, if the file already exists, `OPEN3` will perform a consistency check between your supplied file description and the description found in the file’s own header, and will return TRUE (and leave the file open) only if the two are consistent.

An example of how to use the `OPEN3` function is shown below (from the CMAQ INITSCEN subroutine). This program segment checks for the existence of a CCTM concentration (CTM_CONC_1) file, which if found will be open read-write-update. If the CCTM CONC file is not found, a warning message will be generated.

```
IF ( .NOT. OPEN3( CTM_CONC_1, FSRDWR3, PNAME ) ) THEN
```

```
MSG = 'Could not open ' // CTM_CONC_1 // ' file for update - '
& // 'try to open new'
CALL M3MESG( MSG )
END IF
```

File descriptions (i.e., I/O API file type, dimensions, start date, start time, etc.) can be obtained by using `DESC3`, which is an I/O API Fortran logical function. When `DESC3` is called, the complete file description is placed in the standard file description data structures in FDESC3.EXT . Note that the file must have been opened prior to calling `DESC3`. A typical Fortran use of `DESC3` is:

```

IF ( .NOT. DESC3( ' myfile' ) ) THEN
... error message
ELSE
... DESC3 commons now contain the file description
END IF`
```

Reading Data Files in I/O API

There are four routines with varying kinds of selectivity used to read or otherwise retrieve data from files: `READ3`, `XTRACT3`, `INTERP3`, and `DDTVAR3`. All four are logical functions that return TRUE when they succeed, FALSE when they fail. The descriptions of the routines are listed in Table 6-4.

Table 6-4. IO API data retrieval routines

Routine	Description
READ3	reads one or all variables and layers from a file for a particular date and time.

XTRACT3	reads a windowed subgrid for one or all variables from a file for a particular date and time.
INTERP3	interpolates the requested variable from the requested file to the date/time
DDTVAR3	computes the time-derivative of the requested variable at the specified date/time

Because it optimizes the interpolation problem for the user, `INTERP3` is probably the most useful of these routines. An `INTERP3` call to read/interpolate the variable HNO3 to 1230 GMT on February 4, 1995, is outlined below.

```
CHARACTER*16 FNAME, VNAME
REAL*4 ARRAY( NCOLS, NROWS, NLAYS )
...
IF ( .NOT. INTERP3('myfile','HNO3',1995035,123000,NCOLS*NROWS*NLAY
... (some kind of error happened--deal with it here)
END IF
```

With `READ3` and `XTRACT3`, you can use the “magic values” `ALLVAR3` (= ‘ALL’, as defined in PARMS3.EXT) or `ALLAYS3` (= -1, as also defined in PARMS3.EXT) as the variable name and/or layer number to read all variables or all layers from the file, respectively. For time-independent files, the date and time arguments are ignored.

Writing Data Files in I/O API

CMAQ module developers should use the logical function *WRITE3* to write data to files. For gridded, boundary, and custom files, the code may write either one time step of one variable at a time, or one entire time step of data at a time (in which case, use the “magic value”

ALLVAR3 as the variable name). For ID-referenced, profile, and grid-nest files, the code must write an entire time step at a time.

`LOGICAL FUNCTION WRITE3( FNAME, VNAME, JDATE, JTIME, BUFFER)`

where:

`FNAME (CHARACTER) = file name for query`
`VNAME (CHARACTER) = variable name (or ALLVAR3 (='ALL'))`
`JDATE (INTEGER) = date, formatted YYYYDDD`
`JTIME (INTEGER) = time, formatted HHMMSS`
`BUFFER(*) = array holding output data`

`WRITE3` writes data for the variable with name VNAME, for the date and time (i.e., JDATE and JTIME) to an I/O API-formatted data file with logical name FNAME. For time-independent files, JDATE and JTIME are ignored. If VNAME is the "magic name" `ALLVAR3`, `WRITE3` writes all variables. If FNAME is a dictionary file, `WRITE3` treats VNAME as a dictionary index (and ignores JDATE and JTIME). A typical `WRITE3` call to write data for a given date and time might look like this:

```
REAL*4 ARRAY( NCOLS, NROWS, NLAYS, NVARS )
...
IF ( .NOT. WRITE3( 'myfile', 'HNO3', JDATE, JTIME, ARRAY ) ) THEN
...(some kind of error happened--deal with it here)
END IF
IF ( .NOT. WRITE3( 'afile', 'ALL', JDATE, JTIME, ARRAYB ) ) THEN
...(some kind of error happened--deal with it here)
END IF
```

CMAQ-Related I/O API Utilities

Data files in the CMAQ system can be easily manipulated by using the

I/O API utilities. The I/O API utilities (also known as m3tools) are a set of scriptable programs for manipulation and analysis of netCDF-I/O API formatted files. Information regarding the most commonly employed utility routines is listed in Table 6-5. Further information about how to use the utilities are available in the I/O API documentation.

Table 6-5. I/O API data manipulation utilities

Utility	Description
M3XTRACT	extract a subset of variables from a file for a specified time interval
M3DIFF	compute statistics for pairs of variables
M3STAT	compute statistics for variables in a file
BCWNDW	build a boundary-condition file for a sub-grid window of a gridded file
M3EDHDR	edit header attributes/file descriptive parameters
M3TPROC	compute time period aggregates and write them to an output file
M3TSHIFT	copy/time shift data from a file
M3WNDW	window data from a gridded file to a sub-grid
M3FAKE	build a file according to user specifications, filled with dummy data
VERTOT	compute vertical-column totals of variables in a file
UTMTOOL	coordinate conversions and grid-related computations for lat/lon, Lambert, and UTM

Network Common Data Form (netCDF)

The Network Common Data Form (netCDF) is a set of software libraries and machine-independent data formats that support the creation, access, and sharing of array-oriented scientific data (Unidata, 2009). The netCDF library provides an implementation of the netCDF interface for several different programming languages. The netCDF is used in CMAQ to define the format and data structure of the binary input and output files. CMAQ input and output files are self-describing netCDF-format files in which the file headers have all the dimensioning and descriptive information needed to define the resident data. Users should download the latest code for the NetCDF from the NetCDF website. Compilation and configuration information for the NetCDF is available through the Unidata website.

Message Passing Interface Library (MPI)

The Message Passing Interface (MPI) is a standard library specification for message passing, or intra-software communication, on both massively parallel computing hardware and workstation clusters. There are different open source MPI libraries available that work well with CMAQ.

- MVAPICH2 is a portable implementation of MPI that is available from Ohio State University.
- OpenMPI is an open-source MPI implelmentation that is developed and maintained by a consortium of academic, research, and industry partners.

References for Chapter 6: Required Libraries

Coats, C., 2005: The EDSS/Models-3 I/O API. [Available online at the I/O API website]

Unidata, 2009: NetCDF. [Available online at NetCDF website