MAPL Tutorial

Max Suarez, Atanas Trayanov, Arlindo da Silva, Purnendu Chakraborty

October 18, 2012

Contents

Chapter 1

Introduction

1.1 ESMF

The Earth System Modeling Framework (ESMF) is a software package designed to provide some of the essential functions needed by parallel, scalable earth system models in a machine-independent way.

The simplest ESMF implementation consists of building a Gridded Component (an ESMF superstructure class) that encapsulates the user code, interfacing it to the framework by defining the ESMF callable methods (Initialize, Run and Finalize, hereafter `IRF` methods). This can actually be done without using any of the ESMF Infrastructure - a strategy that fails to capitalize on some of ESMFs greatest strengths. Such encapsulation implementations have dominated the early adoptions of ESMF.

More sophisticated implementations put user data in ESMF infrastructure objects (primarily `ESMF_Field`, `ESMF_Bundle`) which can then be manipulated by a wide array of ESMF methods to facilitate the coupling of components with different data structures (i.e., that are on different grids) and to insulate the user from the architecture-specific implementation layers that are used for inter-process or inter-processor communication, I/O, etc.

In designing ESMF, a deliberate decision was made to have the framework provide services in a very *general* way, and not to prejudge how future models would use it or what programming models would best suit future computer architectures. This generality is an important strength of ESMF, but it is also an impediment to many users that would prefer a more specific formulation for porting existing codes or a better defined recipe for building new codes with ESMF. The generality also impacts the interoperability of applications, since the ESMF interfaces to the IRF methods are general purpose, and they carry little information (other than the grid definition) about the physical content of the data moving in and out of the gridded component.

Also, it was found that users *repeatedly* needed functions that provided higher level functionality than that provided by basic ESMF tools, and that the *core methods of ESMF components* (`Run`, `Initialize` and `Finalize`) *looked very similar in their implementations.*

1.2 MAPL

The MAPL package arose as a response to this early experience, particularly during the construction of GEOS-5. It is based on the observation that much of the work done in these initial implementations can be standardized; thus, reducing the labor of constructing ESMF applications in the future, as well as increasing their interoperability. In its initial implementation, MAPL provides:

- Specific conventions and best practices for the utilization of ESMF in climate models
- A middle-ware layer (between the model and ESMF) that facilitates the adoption of ESMF by climate models.

This enhancement in usability of ESMF comes at the cost of reduced generality. MAPL works *on top of* ESMF and as a separate layer through which the application uses ESMF for some of its functions (although for most things, applications will continue to use ESMF directly). We feel that this middle-ware-layer approach is the right way to get the usability and interoperability that climate model components require of the framework, without sacrificing ESMF's generality and extensibility.

The goal of this tutorial is to introduce the reader quickly to MAPL through examples (in Chapter 2) increasing in complexity and demonstrating various features of MAPL. The corresponding source codes are listed in the Appendix (A). Much of the code presented in the latter examples can be easily adapted to write application specific gridded components.

Chapter 2

Examples

Contents

This chapter consists of examples intended to get the reader started tith MAPL. The first four examples deal with ESMF infrastructure (read/write arrays and bundles, use of configuration files etc.) leading to a stand-alone gridded component and a more complete MAPL example.

It is assumed that the user already has an account on NCCS discover with access to the source code repository (see progress repository quickstart - the link requires NCCS username and password). Also the shell is assumed to be `csh`/`tcsh`.

2.1 Build and Run

The first step is to check out the tutorial module from the CVS repository. Once the CVS environment has been set up, the module can be pulled from the repository.

```
$ setenv CVS_RSH ssh
$ setenv CVSROOT :ext:USERID@cvsacldirect:/cvsroot/esma
$ cd /discover/nobackup/USERID
$ cvs co -r pc_G5tutorial-1_0_9 G5tutorial
```

Here `USERID` is the NCCS username.

Next step is to compile the module. For NCCS discover, a script (`g5_modules`) is provided to create the necessary environment for running and compiling. Once the required modules have been loaded and the environment variable `BASEDIR` has been set, the examples can be built.
[**NEED TO ADD g5_modules TO THE src DIRECTORY**.]

```
$ cd G5tutorial/src
$ source g5_modules
$ make install
```

This creates the following executables in the `G5tutorial/src/Applications/Examples` directory:

```
ex001-CFIO_Array.x
ex001a-CFIO_Array.x
ex002-CFIO_Bundle.x
ex002a-Exceptions.x
ex003-FieldBundle.x
ex004-Config.x
ex005-GridComp.x
ex006-Complete.x
```

All of these examples assume 2 processors, and so are typically run as

```
$ mpirun -np 2 ./executable_name
```

after starting an interactive session (with x forwarding) with

```
$ xsub -V -I
```

2.2 Example - Creating grids and reading arrays from files

This section demonstrates, using a simple ESMF/MAPL example, how to

- Initialize ESMF
- Create a grid using `MAPL_LatLonGridCreate` - retrieve (local) dimensions and latitudes/longitudes from the grid
- Use time manager to create time objects
- Read 2D and 3D arrays from a file

- Read data and test time interpolation

The source code is available at:
`G5tutorial/src/Applications/Examples/ex001-CFIO_Array.F90` and is listed in the Appendix (A.1). Below, we provide a description of the salient features of this code.

We always have the following `Use`s for ESMF/MAPL applications.

```
use ESMF_Mod
use MAPL_Mod
```

Declare variables: 2D/3D grids, time objects, file names, arrays to hold data read from file, coordinate variables, traceback handle (Iam) **explain!?!?!**). It is important to declare both return codes `rc` and `status` - they are required by the error logging and exceptions handling macros.

Initialize ESMF. For performance reasons, it is important to turn of ESMF's automatic logging feature

```
call ESMF_Initialize (DefaultLogType=ESMF_LOG_NONE, vm=vm, rc=status)
VERIFY_(status)
```

where `vm` is an ESMF virtual machine (**EXPLAIN!?!?!**) and `VERIFY_` is an error logging macro. In the following, for the sake of brevity, we skip the `VERIFY_(status)` statement following an ESMF/MAPL call.

After ensuring that we are running on only 2 PETs (an artificial constraint on the tutorial examples), we create a global Lat/Lon grid on a 2×1 layout (2 PEs for decomposing longitudes and 1 for latitudes). Here, we use the origin defaults, date-line and south pole.

```
myGrid = MAPL_LatLonGridCreate (               &
     Name='Max', Nx = 2, Ny = 1,               &
     IM_World=288, JM_World=181, LM_World=72, &
     rc=status)
```

where `myGrid` is an instance of an `ESMF_Grid`.

If needed, one can retrieve the grid co-ordinates (lats/lons) as well as the local (MPI process specific) dimensions (in MAPL terminology, the global dimensions are called `IM_World`, `JM_World` whereas the local dimensions are called `IM`, `JM`. There being no decomposition in the vertical direction, vertical dimension (number of levels) is simply referred to as `LM`.

```
call ESMF_GridGetCoord (myGrid, fptr=lons, coordDim=1, localDE=0, rc=status)
call ESMF_GridGetCoord (myGrid, fptr=lats, coordDim=2, localDE=0, rc=status)

call MAPL_GridGet (myGrid, localCellCountPerDim=dims, rc=status)
IM = dims(1);   JM = dims(2);   LM = dims(3)
```

Next, the time objects are created after setting the default calendar to Gregorian. The data at these times are to be read in from the files.

```
call ESMF_CalendarSetDefault ( ESMF_CAL_GREGORIAN, rc=status )

! March 16 2001, 12:00:00
!------------------------
call ESMF_TimeSet(Time(1), yy=2001, mm=3, dd=16, h=12, m=0, s=0, rc=status )
```

Once all the required arrays have been allocated, the data are read from the files into these arrays. The statement below reads the 3D variable `CH4` from the NetCDF data file defined by `f3d_filename` at `Time(1)`. For more details about the arguments, please refer to the API section of the MAPL manual.

```
call MAPL_CFIORead  ( 'CH4', f3d_Filename, Time(1), myGrid, xa,         &
                      verbose=.true., force_regrid=.true., rc=status )
```

Havin read the `CH4` data at 3 different times (3/16/2001 12:00:00, 3/31/2001 18:00:00 and 4/16/2001 00:00:00) into the variables `xa`, `xb` and `xc`, the absolute and RSM error between `xm` [= `mean(xa,xc)`] and `xb` is computed and printed out.

2.2.1 Using a configuration file

This example reads data and tests time interpolation as in 2.2 while illustrating the following

- Use of a configuration file to read layout and grid dimensions
- Use of MAPL object and the generic `MAPL_GridCreate`
- Use of exception handling macro `__RC__`

The source code is available at:
`G5tutorial/src/Applications/Examples/ex001a-CFIO Array.F90` and is listed in the Appendix(A.1.1). The main features are highlighted below.

2.2.1.1 Configuration/Resource file

The simplest resource file consists of layout parameters `NX:`, `NY:` (number of Parallel Environments for decomposing longitudes/latitudes), `LM:` (number of vertical levels), `GRIDNAME:` (consisting of the global grid dimensions (`IM_WORLD`, `JM_WORLD`), pole and dateline locations). The content of the config file `ex001a.rc` used by this example is:

```
# Resource file for ex001a-CFIO_Array.F90
# Information necessary for creating a global Lat/Lon GEOS-5 grid
# --------------------------------------------------------------

# Layout
# ------
```

```
NX: 2                 # number of PEs for decomposing longitudes
NY: 1                 # number of PEs for decomposing latitudes

# 3D Grid dimensions
# ------------------
LM: 72                # number of vertical levels
GRIDNAME: PE288x91-DE # IM_WORLD=288, JM_WORLD=91
```

2.2.1.2 MAPL object and MAPL_GridCreate

MAPLobj is an instance of the derived type MAPL_MetaComp.

```
type(MAPL_MetaComp) :: MAPLobj
```

It is an opaque object set by MAPL_Set and stores information read from the configuration file which can then be used to create grid by the generic grid creation routine.

```
call MAPL_Set(MAPLobj, ConfigFile=ConfigFile, __RC__)
call MAPL_GridCreate(MAPLOBJ=MAPLobj, ESMFGRID=myGrid, __RC__)
```

The macro __RC__ is defined in 2.2.1.3.

MAPLobj can be queried using MAPL_Get to retrieve the local (process-specific) grid dimensions (IM, JM).

```
call MAPL_Get(STATE=MAPLobj, IM=IM, JM=JM, LM=LM, __RC__)
```

As before, we query the grid to retrieve the co-ordinates. (For MAPL-ized gridded components, where MAPL_GenericInitialize has already been called, the MAPL object also stores the co-ordinate information and can be queried using MAPL_Get). (**CONFIRM!?!?!**) co-ordinates.

```
call ESMF_GridGetCoord (myGrid, fptr=lons, coordDim=1, localDE=0, __RC__)
call ESMF_GridGetCoord (myGrid, fptr=lats, coordDim=2, localDE=0, __RC__)
```

2.2.1.3 Exception handling

MAPL provides a macro __RC__ which is an automatic exception handler for rc=status. Consequently, the calls

```
call MAPL_GridCreate(MAPLOBJ=MAPLobj, ESMFGRID=myGrid, rc=status)
VERIFY_(status)
```

can be reduced to

```
call MAPL_GridCreate(MAPLOBJ=MAPLobj, ESMFGRID=myGrid, __RC__)
```

resulting in fewer lines of code.

The rest of the code is identical to that in 2.2.

2.3 Example - Bundle operations including read/write

This example illustrates the use of `ESMF_Bundle` s

- Read bundle from a file
- Create a new Bundle and populate it with a new field

The source code is available at:
`G5tutorial/src/Applications/Examples/ex003-FieldBundle.F90` and is listed in the Appendix (A.2). The main features of the code are described below.

As before, we initialize ESMF, create grid, retrieve the co-ordinates and (local) dimensions and set the time object as the one in the data file.

2.3.0.4 Read an `ESMF_Bundle`

An `ESMF_Bundle` is first created and all variables from the data file are read into it

```
myBundle = ESMF_FieldBundleCreate (name='Steven', grid=myGrid, __RC__)
call MAPL_CFIORead (f2d_Filename, Time, myBundle, verbose=.true., __RC__)
```

where `myBundle` is an instance of `ESMF_Bundle`.

Number of fields inside the bundle is easily obtained by

```
call ESMF_FieldBundleGet(myBundle, fieldCount=nBndls, __RC__)
```

One can loop over each field to retireve the field name and the Fortran array inside that field.

```
do iBndl = 1, nBndls
   call ESMF_FieldBundleGet(myBundle, fieldIndex=iBndl, field=ithField, __RC__)
   call ESMF_FieldGet(ithField, name=name, __RC__)
   call ESMF_FieldGet(ithField, fArrayptr=fArr, localDE=0, __RC__)
   fArr = 2. * fArr
end do
```

where `ithField` is an instance of `ESMF_Field` and `fArr` is a pointer to a 2D Fortran array. NOTE: modifying this array will modify the what is inside the bundle.

MAPL even has handly routines for retrieving an array out of a bundle.

```
call ESMFL_BundleGetPointerToData(myBundle, 'ch4', fArr, __RC__)
```

NOTE: extra `L` in the routine name. MAPL provides an `ESMFL` module consisting of functions/subroutines that are expected to become a part of ESMF.

2.3.0.5 Create a new ESMF_Bundle

A Fortran array and an ESMF_Grid are first combined to form an ESMF_Field.

```
newField = ESMF_FieldCreate (name='Rolf', grid=myGrid, fArrayptr=newfArr, __RC__)
```

where newfArr is a Fortran array.

Next, a new ESMF_Bundle is create and newField is stored inside it.

```
newBundle = ESMF_FieldBundleCreate (name='Denis', grid=myGrid, __RC__ )
call ESMF_FieldBundleAdd (newBundle, newField, __RC__)
```

Once stored, the Fortran array can always be retrieved using ESMFL_BundleGetPointerToData.

And of course, we clean up after ourselves

```
call ESMF_FieldBundleDestroy(myBundle)
call ESMF_FieldBundleDestroy(newBundle)
call ESMF_FieldDestroy(newField)
call ESMF_GridDestroy(myGrid)
```

2.4 Example - ESMF_Config

A configuration/resource file typically contains a lot more information than layout and grid name. In this example, using a more detailed resource file, we illustrate how to

- load the file
- read and modify scalar attributes
- read vector parameters
- use tables

The source code is available at:
G5tutorial/src/Applications/Examples/ex004-Config.F90 and is listed in the Appendix (A.3). The resource file is available at G5tutorial/src/Applications/Examples/ex004.rc. Below, we provide a description of the code.

2.4.0.6 Load resource file

After initializing ESMF and verifying that we are running on 2 PETs, we start by loading the resource file.

```
cf = ESMF_ConfigCreate()
call ESMF_ConfigLoadFile(cf, fileName=ConfigFile, __RC__)
```

where cf is an instance of ESMF_Config.

2.4.0.7 Read and modify scalar resources

We can retrieve the integers NX, NY as

```
call ESMF_ConfigGetAttribute(cf, Nx, 'NX:', __RC__)
call ESMF_ConfigGetAttribute(cf, Ny, 'NY:', default=2)
```

We can even specify the default values for the retrieved variables.

To modify the value of an attribute, we first set its value in the ESMF_Config object and then query the object for the updated value.

```
call ESMF_ConfigSetAttribute(cf, 4, label='NX:', __RC__)
call ESMF_ConfigGetAttribute(cf, Nx, label='NX:', __RC__)
```

2.4.0.8 Read vector resources

The vector resources in the resource file are accessed as

```
call ESMF_ConfigGetAttribute(cf, BegDate, 2, label='BEG_DATE:', default=0, __RC__)
call ESMF_ConfigGetAttribute(cf, EndDate, 2, lable='END_DATE:', default=0, __RC__)
```

where BegDate and EndDate are integer arrays of length 2. The third argument is the number of returned values expected, 2 in our case.

2.4.0.9 Reading tables

First, we read a table with real numbers. We get the size of the table and then read the table itself by its label.

```
call ESMF_ConfigGetDim(cf, nLines, nCols, 'Statistics::',__RC__)
allocate(rTable(nLines,nCols))
call ESMF_ConfigFindLabel(cf, 'Statistics::',__RC__)
do i = 1, nLines
   call ESMF_ConfigNextLine(cf, __RC__)
   call ESMF_ConfigGetAttribute(cf, rTable(i,:), nCols, __RC__)
end do
```

where rTable is a pointer to a 2D real array.

We can read a variable column table, that too with the table entries being strings.

```
call ESMF_ConfigGetDim(cf, nLines, nCols, 'diag_sfc.fields::',__RC__)
allocate(cTable(nLines,nCols))
call ESMF_ConfigFindLabel(cf, 'diag_sfc.fields::',__RC__)
do i = 1, nLines
   call ESMF_ConfigNextLine(cf, rc=rc)
   do j = 1, nCols
      call ESMF_ConfigGetAttribute(cf, cTable(i,j), default='-')
   end do
end do
```

where `ctable` is a pointer to a 2D character array. NOTE: we need two `do` loops to read the variable sized array. The missing entries are denoted by '-'.

As always, we clean up

```
call ESMF_ConfigDestroy(cf)
```

2.5 Example - Stand-alone gridded component

This example illustrates the use MAPL to write a gridded component (referred to as a 'gridcomp'). The steps are

- Initialize ESMF
- Create a grid
- Create clock
- Create `Import`/`Export` states and the gridcomp
- Set the component's `SetServices`
- Initialize/Run/Finalize component

The source codes for this example are:
`G5tutorial/src/Applications/Examples/ex005-GridComp.F90` and
`G5tutorial/src/Applications/Examples/Example_GridComp/Example_GridCompMod.F90`.

`ex005-GridComp.F90` is the *driver* (or `CAP`) routine that creates a component, and performs its `IRF` functions thorugh the ESMF framework. `Example_GridCompMod.F90` implements the component's internals, its *only* public routine `SetServices` (registering the user-defined `IRF` routine with ESMF) and the user-defined `IRF` methods themselves. Below, we provide a description of the code which is listed in the Appendix (A.4).

Note the statement

```
use Example_GridCompMod, only: ExSetServices => SetServices
```

in addition to our regular `use` statements. In a MAPL application, this implies that the current module (either the main program or a gridcomp) spawns a gridcomp which is coded in the module `Example_GridCompMod`. The *only* public routine `SetServices` of this gridcomp is made accessible to the current module and renamed as `ExSetServices`.

As before, we initialize ESMF, populate MAPL object from the resource file, create a Lat/Lon grid using this object and retrieve the (local) dimensions as well as the co-ordinates. We create an `ESMF_Clock` `CLOCK` starting at 1/1/2001 with a 30 minute time step

```
call ESMF_CalendarSetDefault (ESMF_CAL_GREGORIAN, __RC__)
call ESMF_TimeSet (Time, yy=2001, mm=1, dd=1,  h=0,  m=0, s=0, __RC__)
call ESMF_TimeIntervalSet (TimeStep, h=0, m=30, s=0, __RC__)
CLOCK = ESMF_ClockCreate ("Clovis", timeStep=TimeStep, startTime=Time, __RC__)
```

Next, we create empty `ESMF_States` `IMPORT`, `EXPORT` and the ESMF gridcomp `GC`, of type `ESMF_ATM` (**explain !?!?!**) named Gilbert. Gilbert inherits `myGrid` from the present module and reads the resource file `ex005_GridComp.rc` available at `G5tutorial/src/Applications/Examples/ex005_GridComp.rc`

```
IMPORT = ESMF_StateCreate ( 'Ivan', __RC__ )
EXPORT = ESMF_StateCreate ( 'Evan', __RC__ )
GC = ESMF_GridCompCreate ( &
     name='Gilbert', Grid=myGrid, GridCompType = ESMF_ATM, &
     ConfigFile='ex005_GridComp.rc', __RC__  )
```

The component's `IRF` methods are then registered with the ESMF framework

```
call ESMF_GridCompSetServices ( GC, ExSetServices, __RC__ )
```

and run through calls to the framework.

```
call ESMF_GridCompInitialize ( GC, IMPORT, EXPORT, CLOCK, __RC__ )
call ESMF_GridCompRun ( GC, IMPORT, EXPORT, CLOCK, __RC__ )
call ESMF_GridCompFinalize ( GC, IMPORT, EXPORT, CLOCK, __RC__ )
```

2.5.1 Implementation of the gridded component (Gilbert)

Every non-trivial MAPL-ized ESMF gridcomp has a `SetServices` and at least one phase of each of the registered `IRF` methods. MAPL provides a recipe for each of these routines (please refer to the MAPL User's Guide for a more detailed discussion). We follow these recipes to create our gridcomp.

2.5.1.1 SetServices

The `SetServices` routine is the *only public interface* of the component. It takes the instantiated gridcomp as the first argument and an integer return code as the second

```
subroutine SetServices ( GC, RC )

   type(ESMF_GridComp), intent(INOUT) :: GC  ! gridded component
   integer, optional                  :: RC  ! return code
```

The steps to construct `SetServices` are:

1. Get instance name and setup traceback handle

   ```
   __Iam__('SetServices')
   call ESMF_GridCompGet( GC, name=comp_name, __RC__ )
   Iam = TRIM(comp_name) // '::' // TRIM(Iam)
   ```

2. Register custom `Run` method (`Run_`) with MAPL. MAPL in turn registers them with ESMF. Since, we are not explicitly setting the `Initialize` and `Finalize` routines, MAPL calls `MAPL_GenericInitialize` and `MAPL_GenericFinalize` by default. Custom `Initialize` and `Finalize` methods (if defined) need to invoke these generic methods explicitly.

```
call MAPL_GridCompSetEntryPoint (GC, ESMF_SETRUN, Run_, __RC__)
```

3. Set data services: MAPL's philosophy is that a component, in addition to giving the ESMF framework access to its `IRF` methods, should also tell the framework about its data services - the data it needs from others (`Import`) as well as what it can provide (`Export`). The `SetServices` method of a MAPL-ized gridcomp contains *spec calls* like:

```
! Import state
! ------------
call MAPL_AddImportSpec (                       &
     GC,                                        &
     SHORT_NAME         = 'im_st',              &
     LONG_NAME          = 'import state',       &
     UNITS              = 'uim',                &
     DIMS               = MAPL_DimsHorzOnly,    &
     VLOCATION          = MAPL_VLocationCenter, &
     __RC__ )

! Internal state
! --------------
call MAPL_AddInternalSpec (                     &
     GC,                                        &
     SHORT_NAME         = 'in_st',              &
     LONG_NAME          = 'internal state',     &
     UNITS              = 'uin',                &
     DIMS               = MAPL_DimsHorzVert,    &
     VLOCATION          = MAPL_VLocationCenter, &
     __RC__ )

! Export state
! ------------
call MAPL_AddExportSpec (                       &
     GC,                                        &
     SHORT_NAME         = 'ex_st',              &
     LONG_NAME          = 'export state',       &
     UNITS              = 'u ex',               &
     DIMS               = MAPL_DimsHorzVert,    &
     VLOCATION          = MAPL_VLocationCenter, &
     __RC__ )
```

These calls describe the 3 (ESMF_States) IM/EX/INand allows MAPL to create, initialize and otherwise manipulate these data. The Import/Export states are passed in the calls to the IRF methods whereas the IN state is *attached* to the MAPL object by MAPL. In SetServices we must describe all items in all the three states.

4. Call MAPL_GenericSetServices. This is a setup routine for MAPL and is one of the last things called from the gridcomp's SetServices.

```
call MAPL_GenericSetServices ( GC, __RC__ )
```

Among other things, MAPL_GenericSetServices sets any of the IRF methods that have not been registered to the generic versions and also deals with children (discussed later in the tutorial).

2.5.1.2 Initialize/Finalize

Every MAPL gridcomp must invoke MAPL_GenericInitialize and MAPL_GenericFinalize. MAPL_GenericInitialize, among others, allocates/deallocates memory for the states based on their dimensions provided in the spec calls as well as creates the MAPL object associated with the gridcomp, which can be queried using MAPL_Get. This can be done by either letting the method *default* or by writing a component-specific Initialize function that invokes these generic functions. Finalize parallels Initialize. In the current example, we do not explicitly register the Initialize and Finalize methods, so the generic ones are called.

2.5.1.3 Run

This example defines a custom Run method (Run_) as is evident from step 2 of SetServices. The Run method is simple and does the following:

As always, get the instance name and set up traceback handle.

Get MAPL object for querying the internal state of the gridcomp. If required, one can retrieve grid co-ordinates as well from the MAPL object

```
call MAPL_GetObjectFromGC ( GC, MAPLobj, __RC__ )
call MAPL_Get(STATE=MAPLobj, INTERNAL_ESMF_STATE=INTERNAL, __RC__)
call MAPL_Get(MAPLobj, LONS=lons, LATS=lats, __RC__)
```

Get pointers to data in IM/EX/IN states

```
call MAPL_GetPointer ( IMPORT,   p_im, 'im_st', __RC__ )
call MAPL_GetPointer ( EXPORT,   p_ex, 'ex_st', __RC__ )
call MAPL_GetPointer ( INTERNAL, p_in, 'in_st', __RC__ )
```

Print the first element of the data in each state. Note that EX data array is not printed when the program is run. This is explained in the following section.

```
if (associated(p_im)) print *, 'Import   spec im_st: ', p_im(1,1)
if (associated(p_ex)) print *, 'Export   spec ex_st: ', p_ex(1,1,1)
if (associated(p_in)) print *, 'Internal spec in_st: ', p_in(1,1,1)
```

Most simple components follow the template provided in this example: define a custom `Run` method, a `SetServices` that registers it and calls the generic `SetServices` and default the `Initialize` and `Finalize` methods. Note that the

2.6 Example - More complete Mapl example

This example illustrates a more typical use of MAPL in building complex applications. The source code for the main program is available at
`G5tutorial/src/Applications/Examples/ex006-Complete.F90`.
Most of the functionalities of the main program (the `CAP`) of a MAPL application is provided by the MAPL library and is called `MAPL_Cap`. Consequently, the main program in its entirety looks like:

```
#define I_AM_MAIN
#include "MAPL_Generic.h"

program Example006

  use MAPL_Mod
  use RootGridCompMod, only: Root_SetServices => SetServices

  implicit none

  integer           :: rc, status
  character(len=18) :: Iam="Example006_MAIN"

  call MAPL_CAP(Root_SetServices, __RC__)

  call exit(0)

end Program Example006
```

The main purpose of this driver program is to identify the `Root` component of the MAPL hierarchy and call the `MAPL_Cap` method. The `MAPL_Cap` has exactly *three* children: a single instance each of the (a) `Root` component of a MAPL hierarchy (identified as above) (b) MAPL's `History` component and its (c) `ExtData` component. The `History` and `ExtData` components are created automatically by MAPL.

Additionally, in this example, the `Root` spawns two children `ChildOne` and `ChildTwo` with `ChildTwo` further spawning two children of its own `ChildTwoOne` and `ChildTwoTwo`. The entire hierarchy of this MAPL application (along with each component's states) is represented in Figure 2.1.

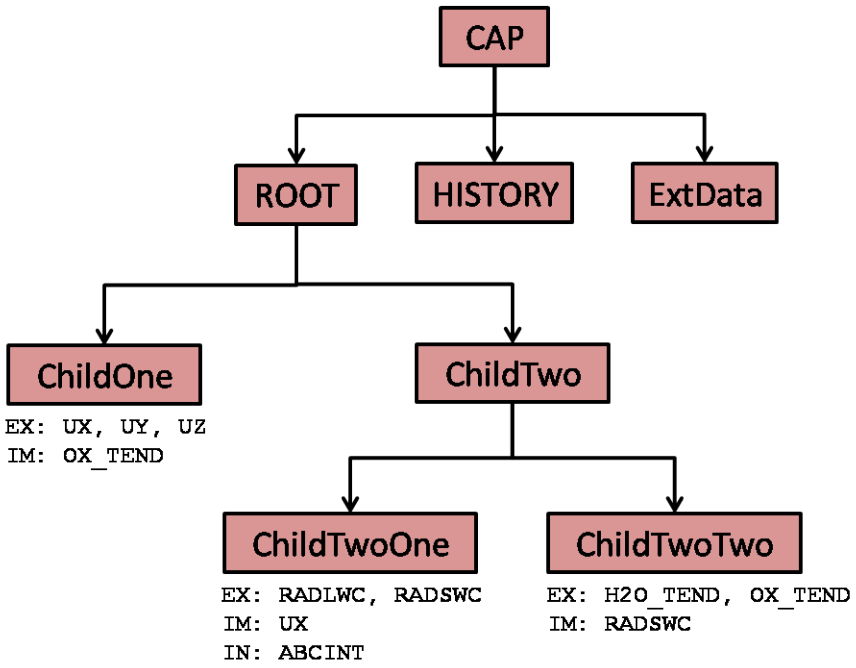

Figure 2.1: Hierarchy of the MAPL application - each box represents a valid `ESMF_GridComp`

To use this code in another application, only the name `RootGridCompMod` needs to be modified. In the following, we describe the application in detail using the bottom-up approach - starting with the *leaf* components (no children) at the bottom and moving up the tree to describe the *composite* components.

2.6.1 ChildTwoTwo

The source code for the `ChildTwoTwo` gridcomp is available at
`G5tutorial/src/Applications/Examples/Ex006_GridComp/ChildTwo_GridComp/ChildTwoTwo_GridComp/ChildTwoTwo_GridCompMod.F90`
and is listed in the Appendix (A.5.1). The gridcomp is similar to the one in the previous example.

2.6.1.1 SetServices

The steps are:

1. Get instance name and setup traceback handle
2. Register custom `Run` method `ChildTwoTwoRun` with MAPL.
3. Set data services: `H2O_TEND` and `OX_TEND` are the `Export` states while `RADSWC` is the `Import` state.
4. Invoke `MAPL_GenericSetServices`.

2.6.1.2 Initialize/Finalize

We let the initialize/finalize routine default to their generic versions.

2.6.1.3 Run

This custom `Run` method illustrates the use of `Export` states. Recall that an `EX` state of a gridcomp contains data that it produces and can share with other components. Memory for the array in an `EX` state is allocated by `MAPL_GenericInitialize` only if the `EX` state appears as an `IM` in another gridcomp. In this example, two temporary arrays (`tmp_h2o`, `tmp_ox`) are allocated and initialized. The pointer to the `EX` states are then retrieved and if *associated* (i.e. if memory for this `EX` has been allocated), the `EX` states are filled out with data from these temporary arrays.

```
! Allocate temporary arrays
! --------------------------
allocate(tmp_h2o(imloc, jmloc, lm)); tmp_h2o = 0.0
allocate(tmp_ox(imloc, jmloc));      tmp_ox  = 0.0

! Do something
! ------------
do ii=1,imloc
   do jj=1,jmloc
      tmp_ox(ii,jj) = 4.44
      do ll=1,lm
         tmp_h2o(ii,jj,ll) = 5.55
      end do
   end do
end do

! Get pointers to export states and fill out
! ------------------------------------------
call MAPL_GetPointer(Export, ptr_ox,  'OX_TEND',  __RC__)
if(associated(pox))  ptr_ox  = tmp_ox
call MAPL_GetPointer(Export, ptr_h2o, 'H2O_TEND', __RC__)
if(associated(ph2o)) ptr_h2o = tmp_h2o

deallocate(tmp_ox, tmp_h2o)
```

NOTE that

- while `OX_TEND` appears as an `Import` state in another gridcomp (`ChildOne`), `H2O_TEND` does not (Figure 2.1). Consequently, memory for this `EX` is not allocated, saving memory. As a result, `ptr_h2o` is not *associated*. If a new gridcomp is added that requires `H2O_TEND`, this `Export` state will get filled automatically making it available to the new gridcomp.

- Memory for data in `Import` state `RADSWC` is automatically allocated by `MAPL_GenericInitialize`.
- Allocate/deallocate memory and copying data (from temporary array) is expensive. Since we know apriori that `OX_TEND` is an `IM` to another gridcomp, it would be preferable to do the following instead

```
! Allocate temp array for H2O_TEND
! Get pointer to OX_TEND and assert that it is associated
! ------------------------------------------------------
allocate(tmp_h2o(imloc, jmloc, lm))
call MAPL_GetPointer(Export, ptr_ox,  'OX_TEND',  __RC__)
ASSERT_(associated(ptr_ox))

! Do something
! ------------
do ii=1,imloc
   do jj=1,jmloc
      ptr_ox(ii,jj) = 4.44
      do ll=1,lm
         tmp_h2o(ii,jj,ll) = 5.55
      end do
   end do
end do

! Get pointer to H2O_TEND and fill out (if needed)
! ------------------------------------------------
call MAPL_GetPointer(Export, ptr_h2o, 'H2O_TEND', __RC__)
if(associated(ptr_h2o)) ptr_h2o = tmp_h2o

deallocate(tmp_h2o)
```

2.6.2 ChildTwoOne

The source code for the `ChildTwoOne` gridcomp is available at
`G5tutorial/src/Applications/Examples/Ex006_GridComp/ChildTwo_GridComp/ChildTwoOne_GridComp/ChildTwoOne_GridCompMod.F90`
and is listed in the Appendix (A.5.2). Much of the code is similar to the gridcomp `ChildTwoTwo`.

2.6.2.1 SetServices

1. Register custom `Run` method `ChildTwoOneRun` with MAPL.
2. Set data services: `RADSWC`, `RADLWC` are `Export` states, `UX` is an `Import` state while `ABCINT` is an `Internal` state. Recall that MAPL expands the concept of `ESMF_State` to a component's `Internal` state to *manage its persistent data*. This `IN` state does

not appear explicitly in the argument list of `IRF` methods, instead is *attached* to the `ESMF_GridComp` and can be queried.

2.6.2.2 Initialize

Since a custom `Initialize` method is not registered with MAPL, `MAPL_GenericInitialize` is called by default which, among others, allocates memory for the `IN` state.

2.6.2.3 Run

The internal state is retrieved from the MAPL object (the MAPL object literally knows everything about the gridcomp), stored in `IntState` and the elements of its array are accessed.

```
call MAPL_Get(MAPLobj, INTERNAL_ESMF_STATE=IntState, __RC__)
call MAPL_GetPointer(IntState, ptr_abc, 'ABCINT', __RC__)
do ii=1,imloc
   do jj=1,jmloc
      do ll=1,lm
         ptr_abc(ii,jj,ll) = 3.33
      end do
   end do
end do
```

2.6.3 ChildTwo

The source code for the `ChildTwo` gridcomp is available at
`G5tutorial/src/Applications/Examples/Ex006_GridComp/ChildTwo_GridComp/`
`/ChildTwo_GridCompMod.F90`
and is listed in the Appendix (A.5.3). Since this is a composite gridcomp (spawns children), we have two additional `use` statements identifying the children's `SetServices` and two *global* integers representing the children.

```
use ChildTwoOneGridCompMod, only: TwoOne_SetServices => SetServices
use ChildTwoTwoGridCompMod, only: TwoTwo_SetServices => SetServices

integer                                 :: ChildTwoOne
integer                                 :: ChildTwoTwo
```

2.6.3.1 SetServices

1. Register custom `Run` method `ChildTwoRun` with MAPL.

2. Create children, identify and invoke their `SetServices` methods.

```
ChildTwoOne = MAPL_AddChild(GC, name='ChildTwoOne', ss=TwoOne_SetServices, __RC__)
ChildTwoTwo = MAPL_AddChild(GC, name='ChildTwoTwo', ss=TwoTwo_SetServices, __RC__)
```

3. Since a parent component *owns* its children components, it has access to their IM/EX states. MAPL takes advantage of this by having the parent explicitly declare the connections between the children's Import and Export states. Note that *all connections occur between siblings*. Each parent's constituent components (its children) are connected to each other though the MAPL_AddConnectivity method (part of MAPL libray). Through this call, MAPL makes RADSWC available to ChildTwoTwo by extracting some of the needed information from the data services provided by ChildTwoOne.

```
call MAPL_AddConnectivity(        &
     GC,                          &
     SHORT_NAME = (/ 'RADSWC' /), &
     SRC_ID     = ChildTwoOne,    &
     DST_ID     = ChildTwoTwo,    &
     __RC__)
```

4. An Import state is *required* by a gridcomp to function. In case, an IM is not satified at the parent level (e.g. UX of ChildTwoOne) it bubbles up the hierarchy, i.e. this state becomes an IM state of its parent gridcomp and enabling *coupling between cousins*. So UX become an IM of ChildTwo.

5. Unlike Imports, the Export states of a child gridcomp do not bubble up the hierarchy. A parent needs to make a particular Export state of a child available for coupling at a higher level of hierarchy. For example, the EX state OX_TEND of ChildTwoTwo is an IM of ChildOne. Since connections can only be made between siblings (by the parent), OX_TEND needs to appear as an EX of ChildTwo (and subsequently the connection can be made by Root). This is done through the following spec call in ChildTwo's SetServices.

```
call MAPL_AddExportSpec(          &
     GC,                          &
     SHORT_NAME = 'OX_TEND',      &
     CHILD_ID   = ChildTwoTwo,    &
     __RC__)
```

2.6.3.2 Run

Typically the *leaf* components (appear towards the bottom of the hierarchy and do not spawn children) contain the bulk of the computational code (dynamical cores or computation of physical quantities) whereas the composite (parent) components group the children together and decide the order in which the children's Run methods are executed

```
! Get MAPL object from GC
! ------------------------
call MAPL_GetObjectFromGC(GC, MAPLobj, __RC__)

! Get children and their IM/EX states from MAPL object
```

```
! ------------------------------------------------------
call MAPL_Get(MAPLobj, GCS=GCS, GIM=GIM, GEX=GEX, __RC__)

! Run children
! ------------
call ESMF_GridCompRun(GCS(ChildTwoOne), GIM(ChildTwoOne), GEX(ChildTwoOne), Clock, __RC__)
call ESMF_GridCompRun(GCS(ChildTwoTwo), GIM(ChildTwoTwo), GEX(ChildTwoTwo), Clock, __RC__)
```

2.6.4 ChildOne

The source code for the `ChildOne` gridcomp is available at
`G5tutorial/src/Applications/Examples/Ex006_GridComp/ChildOne_GridComp/`
`/ChildOne_GridCompMod.F90`
and is listed in the Appendix (A.5.4). Unlike `ChildTwo`, this is a leaf component with its own `EX`/`IM` states.

2.6.4.1 SetServices

1. Register custom `Run` method `ChildOneRun` with MAPL.
2. Set data services: `UX`, `UY`, `UZ` are the `Export` states and `OX_TEND` (originally from `ChildTwoTwo` but also available as an `EX` of `ChildTwo`) is the `Import` state.

2.6.5 Root

The source code for the `Root` gridcomp is available at
`G5tutorial/src/Applications/Examples/Ex006_GridComp/Root_GridComp.F90`
and is listed in the Appendix (A.5.5). This is the topmost user-defined gridcomp and is almost always a composite component spawning children.

2.6.5.1 SetServices

1. Register custom `Initialize` and `Run` methods (`RootInitialize`, `RootRun`) with MAPL. The `Root` component *requires* (**Is this correct !?!?!?**) a user-defined `Initialize` method, the main purpose of which is to create a grid.
2. Make `ChildTwo`'s `Export` state `OX_TEND` (manually moved up the hierarchy from `ChildTwoTwo` to `ChildTwo`) available to `ChildOne` as an `Import` state.

```
call MAPL_AddConnectivity(         &
     GC,                           &
     SHORT_NAME = (/ 'OX_TEND' /), &
     SRC_ID   =  ChildTwo,         &
     DST_ID   =  ChildOne,         &
     __RC__)
```

   Also connect `ChildOne`'s `Export` state `UX` to `ChildTwo`'s `IM` (`UX` bubbled up from `ChildTwoOne` to `ChildTwo`).

```
call MAPL_AddConnectivity(         &
     GC,                           &
     SHORT_NAME = (/ 'UX' /),      &
     SRC_ID   =  ChildOne,         &
     DST_ID   =  ChildTwo,         &
     __RC__)
```

2.6.5.2 Initialize

The main purpose of the user-define `Initialize` method (`RootInitialize`) is to create the underlying grid. Also, since we are using a custom `Initialize` method, we *have to* call its generic version explicitly.

```
call MAPL_GridCreate(GC, __RC__)
call MAPL_GenericInitialize(GC, Import, Export, Clock, __RC__)
```

2.6.5.3 Run

The parent gridcomp executes the `Run` methods of its children

```
! Get children and their IM/EX states from MAPL object
! -----------------------------------------------------
call MAPL_Get(MAPLobj, GCS=GCS, GIM=GIM, GEX=GEX, __RC__)

! Run children
! ------------
call ESMF_GridCompRun(GCS(ChildOne), GIM(ChildOne), GEX(ChildOne), Clock, __RC__)
call ESMF_GridCompRun(GCS(ChildTwo), GIM(ChildTwo), GEX(ChildTwo), Clock, __RC__)
```

2.6.6 Configuration/Resource files

The behavior of a MAPL application is controlled through resource or configuration files. MAPL treats the application configuration like a UNIX environment that is available to all components in the application by propagating it down the hierarchy. `MAPL_Cap` opens configuration files for itself and its three children (`Root`, `History` and `ExtData`) and these *must be present* in the run directory at run time. The name `MAPL_Cap`'s own resource file is fixed as `CAP.rc`. The default names for the other three are `ROOT.rc`, `HISTORY.rc` and `ExtData.rc` but may be renamed in `CAP.rc`.

Below, we briefly describe the contents of these files. For a more detailed description, please refer to the MAPL user's guide.

2.6.6.1 Cap config

- The config files for `Root`, `History` and `ExtData` are renamed.
- Carry out a 1-day simulation starting at 03/01/1991 00:00:00 (also see `cap_restart`) and ending at 03/02/1991 00:00:00 with a time step of 7200 seconds.

- Enable MAPL timers and disable MAPL memutils.
- Do not print `ESMF_State` specs (a non-zero value of `PRINTSPEC` simply prints the corresponding spec (`Import`, `Export` or both) and does not execute the program.

2.6.6.2 Root config

At the very minimum this config file contains

- Layout (number of parallel environments for decomposing longitudes/latitudes) - 2×1 for this example
- Gridname (`PC72x46-DC`) specifying `IM_WORLD` (72), `JM_WORLD` (46) (global grid dimensions) and the number of vertical levels (7).

2.6.6.3 History config

The history config controls the output (output file names, variables to be written, frequency etc.).

- Set the experiment ID (`EXPID` and a description `EXPDSC`.
- Name the *Collection* to be written (`myCollection`)
- Add attributes of each collection (`template`, `format` etc.). The attribute 'fields' is the label for the list opf fields in the collection. Each line in this field consists of a comma separated list: '`Export` state name, component that produces it, the alias to use for it in the file'. The alias may be omitted, in which case it defaults to the true name.

In this example, NetCDF-4 (self-describing format) files are written out for variables `UZ` (in `ChildOne`) and `RADLWC` in `ChildTwoOne` every 4 hours.

Question: So `UZ` and `RADLWC` are `Import`s of `History`. But these `EX`s have not been manually bubbled up. How are they connected then **!?!?!**.
Question: What does `MAPL_ExportStateGet` do **!?!?!**.

2.6.6.4 ExtData

Since all `Import`s are satisfied, we do not need any external data and hence this file is empty.

2.6.6.5 cap_restart???

Appendix A

Source code for tutorials

Contents

A.1 Create grids and read arrays from files

```
#include "MAPL_Generic.h"

Program Example_001

  ! Example 001 - Simple ESMF/MAPL example demonstrating how to
  !
  ! 1. Initialize the ESMF
  ! 2. Create a grid (using MAPL_LatLonGridCreate) - retrieve
  !    (local) dimensions and longitudes/latitudes
  ! 3. Use the time manager to create a time object
  ! 3. Read 2D and 3D arrays from a file
  ! 4. Read data with automatic time interpolation
  !
  ! It assumes 2 processors, so typically you will run it as
  !
  ! % mprirun -np 2 ex001-CFIO_Array.x
```

```
17  !
    ! Arlindo da Silva <arlindo.dasilva@nasa.gov>, October 2008
19  !-----------------------------------------------------------------


21
    use ESMF_Mod
23  use MAPL_Mod

25  implicit NONE


27

29  ! Basic ESMF objects being used in this example
    ! ---------------------------------------------
31  type(ESMF_Grid)       :: myGrid, myGrid2d    ! 3D and 2D Grids
    type(ESMF_Time)       :: Time(4)             ! Time objects
33  type(ESMF_VM)         :: vm                  ! ESMF Virtual Machine

35  ! Hardwired file names
    ! --------------------
37  character(len=*), parameter ::  f2d_Filename = 'data/emissions.nc'
    character(len=*), parameter ::  f3d_Filename = 'data/tracers.nc'
39
    ! Holds arrays being read from file
41  ! --------------------------------
    real, pointer, dimension(:,:)    :: terp, voc
43  real, pointer, dimension(:,:,:) :: oh, ch4, xa, xb, xc, xm, xe

45  ! Coordinate variables
    ! --------------------
47  real(kind=8), pointer, dimension(:,:) :: lons, lats

49  ! Basic information about the parallel environment
    !        PET = Persistent Execution Threads
51  ! In the current implementation, a PET is equivalent
    ! to an MPI process
53  ! -----------------------------------------------
    integer :: myPET   ! The local PET number
55  integer :: nPET    ! The total number of PETs you are running on

57  integer :: rc, status, dims(3)
    integer :: i, j, k, IM, JM, LM
59
    real, parameter :: r2d = 180. / MAPL_PI
61  character(len=*), parameter :: Iam = 'Example-001'

63  call Main()

65 CONTAINS

67    subroutine Main()

69    ! Initialize the ESMF. For performance reasons, it is important
      ! to turn OFF ESMF's automatic logging feature
71    ! ---------------------------------------------------------------
```

```
      call ESMF_Initialize (DefaultLogType=ESMF_LOG_NONE, vm=vm, rc=status)
73    VERIFY_(status)

75    ! Check the number of processors, if not 2, exit
      ! ------------------------------------------------
77    call ESMF_VMGet(vm, localPET=myPET, PETcount=nPET)
      if ( nPET /= 2 ) then
79       if ( MAPL_am_I_root() ) then
            print *, 'Error: expecting 2 PETs but found ', nPET, 'PETs'
81          print *, 'Try:   mpirun -np 2 ex001-CFIO_Array.x'
         end if
83       ASSERT_(.FALSE.)
      end if
85
      if ( MAPL_am_I_root() ) then
87       print *
         print *, 'Starting ' // Iam // ' with ', nPET, ' PETs ...'
89       print *
      end if
91
      ! Create a global 3D Lat-Lon grid on a 2x1 layout
93    ! Uses origin defaults: date-line and South Pole
      ! ----------------------------------------------
95    myGrid = MAPL_LatLonGridCreate (                &
           Name='Max', Nx = 2, Ny = 1,                &
97         IM_World=288, JM_World=181, LM_World=72, &
           rc=status)
99    VERIFY_(status)

101   ! Create a global 2D Lat-Lon grid on a 2x1 layout
      ! LM_World=0 implies no vertical dimension
103   ! ----------------------------------------------
      myGrid2d = MAPL_LatLonGridCreate (              &
105        Name='Larry', Nx = 2, Ny = 1,              &
           IM_World=288, JM_World=181, LM_World=0,  &
107        rc=status)
      VERIFY_(status)
109
      ! Retrieve Lons/Lats from Grid
111   ! ----------------------------
      call ESMF_GridGetCoord (myGrid, fptr=lons, coordDim=1, localDE=0, rc=status)
113   VERIFY_(status)
      call ESMF_GridGetCoord (myGrid, fptr=lats, coordDim=2, localDE=0, rc=status)
115   VERIFY_(status)

117   ! Get Local dimensions
      ! --------------------
119   call MAPL_GridGet (myGrid, localCellCountPerDim=dims, rc=status)
      VERIFY_(status)
121   IM = dims(1);    JM = dims(2);    LM = dims(3)

123   ! Set the time as the one on the hardwired file name
      ! --------------------------------------------------
125   call ESMF_CalendarSetDefault ( ESMF_CAL_GREGORIAN, rc=status )
      VERIFY_(status)
```

```
127     call ESMF_TimeSet(Time(1), yy=2001, mm=1, dd=1,  h=0,  m=0, s=0, rc=status )
        VERIFY_(status)
129     call ESMF_TimeSet(Time(2), yy=2001, mm=7, dd=6,  h=0,  m=0, s=0, rc=status )
        VERIFY_(status)
131     call ESMF_TimeSet(Time(3), yy=2001, mm=1, dd=14, h=12, m=0, s=0, rc=status )
        VERIFY_(status)
133     call ESMF_TimeSet(Time(4), yy=2001, mm=3, dd=16, h=12, m=0, s=0, rc=status )
        VERIFY_(status)
135
        ! Allocate arrays to receive data to be read
137     ! ------------------------------------------
        allocate( terp(IM,JM), voc(IM,JM), oh(IM,JM,LM), ch4(IM,JM,LM), stat=status)
139
        ! Read 2D arrays from file
141     ! ------------------------
        call MAPL_CFIORead ( 'emcoterp', f2d_Filename, Time(1), myGrid2d, terp, &
143                          verbose=.true., force_regrid=.true.,               &
                             time_is_cyclic = .true., rc=status )
145     VERIFY_(status)
        call MAPL_CFIORead ( 'emconvoc', f2d_Filename, Time(2), myGrid2d, voc,  &
147                           verbose=.true., force_regrid=.true.,              &
                              time_is_cyclic = .true., rc=status )
149     VERIFY_(status)

151     ! Read 3D arrays from file
        ! ------------------------
153     call MAPL_CFIORead ( 'OH', f3d_Filename, Time(3), myGrid, oh,           &
                             verbose=.true., force_regrid=.true., rc=status )
155     VERIFY_(status)
        call MAPL_CFIORead ( 'CH4', f3d_Filename, Time(4), myGrid, ch4,         &
157                          verbose=.true., force_regrid=.true., rc=status )
        VERIFY_(status)
159

161     ! Testing time interpolation
        ! --------------------------
163     call ESMF_TimeSet(Time(1), yy=2001, mm=3, dd=16, h=12, m=0, s=0, rc=status )
        VERIFY_(status)
165     call ESMF_TimeSet(Time(2), yy=2001, mm=3, dd=31, h=18, m=0, s=0, rc=status )
        VERIFY_(status)
167     call ESMF_TimeSet(Time(3), yy=2001, mm=4, dd=16, h=0,  m=0, s=0, rc=status )
        VERIFY_(status)
169
        allocate( xa(IM,JM,LM), xb(IM,JM,LM), xc(IM,JM,LM), &
171               xm(IM,JM,LM), xe(IM,JM,LM), stat=status )

173     call MAPL_CFIORead  ( 'CH4', f3d_Filename, Time(1), myGrid, xa,          &
                              verbose=.true., force_regrid=.true., rc=status )
175     VERIFY_(status)
        call MAPL_CFIORead  ( 'CH4', f3d_Filename, Time(2), myGrid, xb,          &
177                           verbose=.true., force_regrid=.true.,              &
                              time_interp = .true., rc=status )
179     VERIFY_(status)
        call MAPL_CFIORead  ( 'CH4', f3d_Filename, Time(3), myGrid, xc,          &
181                            verbose=.true., force_regrid=.true., rc=status )
```

```
      VERIFY_(status)

      ! Summary
      ! -------
      if ( MAPL_am_I_root() ) then
         print *
         print *, 'Grid Information: '
         print *
      end if
      write(*,'('' ['',i2,''] Longitudinal range: '',2f10.2)') &
           myPET, r2d*lons(1,1), r2d*lons(IM,1)
      write(*,'('' ['',i2,''] Latitudinal range: '',2f10.2)') &
           myPET, r2d*lats(1,1), r2d*lats(1,JM)

      if ( MAPL_am_I_root() ) then
         xm = ( xa + xc ) / 2.
         xe = abs ( xb - xm )
         i = IM/2

         print *
         print *, 'Time interpolation statistics: '
         print *
         do j = 1, JM, JM/4
            write(*,'(1x,a,f8.2)') 'Lat = ', lats(i,j) * 180. / MAPL_PI
            print *, 'CH4 at time a: ', (xa(i,j,k), k=1,LM,LM/4)
            print *, 'CH4 at time b: ', (xb(i,j,k), k=1,LM,LM/4)
            print *, 'CH4 m e a n  : ', (xm(i,j,k), k=1,LM,LM/4)
            print *, 'CH4 at time c: ', (xc(i,j,k), k=1,LM,LM/4)
            print *, 'CH4 e r r o r: ', (xe(i,j,k), k=1,LM,LM/4)
            print *
         end do

         print *, 'CH4 RMS error: ', sqrt(sum(xe*xe)/(IM*JM))

      endif

      deallocate(terp,voc,oh,ch4)
      deallocate(xa, xb, xc, xm, xe)

      ! All done
      ! --------
      call ESMF_Finalize (rc=status)
      VERIFY_(status)

   end subroutine Main

end Program Example_001
```

A.1.1 Use of configuration file

```
#include "MAPL_Generic.h"

Program Example_001a

   ! Example 001 - Simple ESMF/MAPL example demonstrating how to
   !
```

```
! 1. Initialize the ESMF
! 2. Create a grid (using the generic MAPL_GridCreate) -
!    retrieve (local) dimensions and longitudes/latitudes
!    from the grid
! 3. Use time manager to create time objects
! 3. Read 2D and 3D arrays from a file at specified times
! 4. Read data and test time interpolation
!
! It assumes 2 processors, so typically you will run it as
!
! % mprirun -np 2 ex001a-CFIO_Array.x
!
! Arlindo da Silva <arlindo.dasilva@nasa.gov>, October 2008
!-------------------------------------------------------------------------


use ESMF_Mod
use MAPL_Mod

implicit NONE

! Basic ESMF objects being used in this example
! ---------------------------------------------
type(ESMF_Grid)       :: myGrid          ! The grid
type(ESMF_Time)       :: Time(4)         ! Time objects
type(ESMF_VM)         :: vm              ! ESMF Virtual Machine

! All important MAPL object
! -------------------------
type(MAPL_MetaComp) :: MAPLobj

! Hardwired file names
! --------------------
character(len=*), parameter ::  f2d_Filename = 'data/emissions.nc'
character(len=*), parameter ::  f3d_Filename = 'data/tracers.nc'

! Holds arrays being read from file
! ---------------------------------
real, pointer, dimension(:,:)    :: terp, voc
real, pointer, dimension(:,:,:) :: oh, ch4, xa, xb, xc, xm, xe

! Coordinate variables
! --------------------
real(kind=8), pointer, dimension(:,:)    :: lons, lats

! Basic information about the parallel environment
!       PET = Persistent Execution Threads
! In the current implementation, a PET is equivalent
! to an MPI process
! -----------------------------------------------
integer :: myPET    ! Local PET number
integer :: nPET     ! Total number of PETs

integer :: status, rc, dims(3)
integer :: i, j, k, IM, JM, LM
```

```
62    type(ESMF_TimeInterval) :: TINV

64    real, parameter :: r2d = 180. / MAPL_PI

66    character(len=*), parameter :: Iam        = 'Example-001a'
      character(len=*), parameter :: ConfigFile = 'ex001a.rc'
68

70    call Main()

72  contains

74    subroutine Main()

76      ! Initialize the ESMF. For performance reasons, it is important
        ! to turn OFF ESMF's automatic logging feature
78      ! ------------------------------------------------------------
        call ESMF_Initialize (DefaultLogType=ESMF_LOG_NONE, vm=vm, __RC__)
80
        ! Check the number of processors. If not 2, exit
82      ! ----------------------------------------------
        call ESMF_VMGet(vm, localPET=myPET, PETcount=nPET)
84      if ( nPET /= 2 ) then
           if ( MAPL_am_I_root() ) then
86            print *, 'Error: expecting 2 PETs but found ', nPET, 'PETs'
              print *, 'Try:   mpirun -np 2 ex001-CFIO_Array.x'
88         end if
           ASSERT_(.false.)
90      end if

92      if ( MAPL_am_I_root() ) then
           print *
94         print *, 'Starting ' // Iam // ' with ', nPET, ' PETs ...'
           print *
96      end if

98      ! MAPL object is populated from config file
        ! -----------------------------------------
100     call MAPL_Set(MAPLobj, ConfigFile=ConfigFile, __RC__)

102     ! Create grid from info in config file
        ! ------------------------------------
104     call MAPL_GridCreate(MAPLOBJ=MAPLobj, ESMFGRID=myGrid, __RC__)
        call ESMF_GridValidate(myGrid)
106
        ! Get local dimensions from MAPL object
108     ! Note: Cannot use MAPL_Get to get the coords lons, lats.
        !       That would have been possible if MAPL_Get was called
110     !       from inside a MAPL component, whose
        !       MAPL_GenericInitialize has already been called
112     ! ------------------------------------------------------------
        call MAPL_Get(STATE=MAPLobj, IM=IM, JM=JM, LM=LM, __RC__)
114
        ! Retrieve lons/lats from grid
116     ! ----------------------------
```

```
      call ESMF_GridGetCoord (myGrid, fptr=lons, coordDim=1, localDE=0, __RC__)
      call ESMF_GridGetCoord (myGrid, fptr=lats, coordDim=2, localDE=0, __RC__)

      ASSERT_(IM==size(LONS,1))
      ASSERT_(JM==size(LONS,2))

      ! Set the time as the one on the hardwired file name
      ! ---------------------------------------------------
      call ESMF_CalendarSetDefault ( ESMF_CAL_GREGORIAN, __RC__ )
      call ESMF_TimeSet(Time(1), yy=2001, mm=1, dd=1,  h=0,  m=0, s=0, __RC__ )
      call ESMF_TimeSet(Time(2), yy=2001, mm=7, dd=6,  h=0,  m=0, s=0, __RC__ )
      call ESMF_TimeSet(Time(3), yy=2001, mm=1, dd=14, h=12, m=0, s=0, __RC__ )
      call ESMF_TimeSet(Time(4), yy=2001, mm=3, dd=16, h=12, m=0, s=0, __RC__ )

      ASSERT_(Time(1)<Time(2))

      ! What does this do really?
      ! -------------------------
      if ( MAPL_am_I_root() ) then
         TINV = Time(2)-Time(1)
         print *
         call ESMF_TimeIntervalPrint(TINV, options="string",__RC__)
         VERIFY_(STATUS)
         print *
      end if

      ! Allocate arrays to receive data to be read
      ! ------------------------------------------
      allocate( terp(IM,JM), voc(IM,JM), oh(IM,JM,LM), ch4(IM,JM,LM), stat=STATUS )
      VERIFY_(STATUS)

      ! Read 2D arrays from file
      ! ------------------------
      call MAPL_CFIORead  ( 'emcoterp', f2d_Filename, Time(1), myGrid, terp,   &
                            verbose=.true., force_regrid=.true.,               &
                            time_is_cyclic = .true., __RC__ )
      call MAPL_CFIORead  ( 'emconvoc', f2d_Filename, Time(2), myGrid, voc,    &
                            verbose=.true., force_regrid=.true.,               &
                            time_is_cyclic = .true., __RC__ )

      ! Read 3D arrays from file
      ! ------------------------
      call MAPL_CFIORead  ( 'OH', f3d_Filename, Time(3), myGrid, oh,           &
                            verbose=.true., force_regrid=.true., __RC__ )
      call MAPL_CFIORead  ( 'CH4', f3d_Filename, Time(4), myGrid, ch4,         &
                            verbose=.true., force_regrid=.true., __RC__ )

      ! Testing time interpolation
      ! --------------------------
      call ESMF_TimeSet(Time(1), yy=2001, mm=3, dd=16, h=12, m=0, s=0, __RC__ )
      call ESMF_TimeSet(Time(2), yy=2001, mm=3, dd=31, h=18, m=0, s=0, __RC__ )
      call ESMF_TimeSet(Time(3), yy=2001, mm=4, dd=16, h=0,  m=0, s=0, __RC__ )

      allocate( xa(IM,JM,LM), xb(IM,JM,LM), xc(IM,JM,LM), &
                xm(IM,JM,LM), xe(IM,JM,LM), stat=status )
```

```
172     VERIFY_(STATUS)

174     call MAPL_CFIORead  ( 'CH4', f3d_Filename, Time(1), myGrid, xa,          &
                              verbose=.true., force_regrid=.true., __RC__ )
176     call MAPL_CFIORead  ( 'CH4', f3d_Filename, Time(2), myGrid, xb,          &
                              verbose=.true., force_regrid=.true.,            &
178                           time_interp = .true., __RC__ )
        call MAPL_CFIORead  ( 'CH4', f3d_Filename, Time(3), myGrid, xc,          &
180                           verbose=.true., force_regrid=.true., __RC__ )

182     ! Summary
        ! -------
184     if ( MAPL_am_I_root() ) then
           print *
186        print *, 'Grid Information: '
           print *
188     end if
        write(*,'('' ['',i2,''] Longitudinal range: '',2f10.2)') &
190          myPET, r2d*lons(1,1), r2d*lons(IM,1)
        write(*,'('' ['',i2,''] Latitudinal range: '',2f10.2)') &
192          myPET, r2d*lats(1,1), r2d*lats(1,JM)

194    if ( MAPL_am_I_root() ) then
          xm = ( xa + xc ) / 2.
196       xe = abs ( xb  - xm )
          i = IM/2
198
          print *
200       print *, 'Time interpolation statistics: '
          print *
202       do j = 1, JM, JM/4
             write(*,'(1x,a,f8.2)') 'Lat = ', lats(i,j) * 180. / MAPL_PI
204          print *, 'CH4 at time a: ', (xa(i,j,k), k=1,LM,LM/4)
             print *, 'CH4 at time b: ', (xb(i,j,k), k=1,LM,LM/4)
206          print *, 'CH4 m e a n  : ', (xm(i,j,k), k=1,LM,LM/4)
             print *, 'CH4 at time c: ', (xc(i,j,k), k=1,LM,LM/4)
208          print *, 'CH4 e r r o r: ', (xe(i,j,k), k=1,LM,LM/4)
             print *
210       end do

212       print *, 'CH4 RMS error: ', sqrt(sum(xe*xe)/(IM*JM))

214    endif

216    deallocate(terp,voc,oh,ch4)
       deallocate(xa, xb, xc, xm, xe)
218
       ! All done
220    ! --------
       call ESMF_Finalize ( __RC__ )
222
   end subroutine Main
224
end Program Example_001a
```

A.2 Bundle operations including read/write

```
#include "MAPL_Generic.h"

Program Example_003
  ! Example 003 - Simple ESMF/MAPL example demonstrating how to
  !
  ! 1. Initialize ESMF
  ! 2. Create a grid
  ! 3. Use time manager to create a time object
  ! 3. Read bundles from a file
  ! 4. Iterate through items in a bundle
  ! 5. Get Fortran arrays out of a Bundle
  ! 6. Create a new Bundle and populate it with a new Field.
  !
  ! It assumes 2 processors, so typically you will run it as
  !
  ! % mprirun -np 2 ex003-FieldBundle.x
  !
  ! Arlindo da Silva <arlindo.dasilva@nasa.gov>, October 2008
  !-----------------------------------------------------------------------

  use ESMF_Mod
  use MAPL_Mod

  implicit NONE

  ! Basic ESMF objects being used in this example
  ! ---------------------------------------------
  type(ESMF_Grid)                 :: myGrid   ! Grid
  type(ESMF_Time)                 :: Time     ! Time object
  type(ESMF_VM)                   :: vm       ! ESMF Virtual Machine

  ! All important MAPL object
  ! -------------------------
  type(MAPL_MetaComp) :: MAPLobj

  ! Bundles to hold data to be read in
  ! ----------------------------------
  type(ESMF_Field)                :: ithField
  type(ESMF_FieldBundle)          :: myBundle

  type(ESMF_FieldBundle)          :: newBundle
  type(ESMF_Field)                :: newField

  ! Hardwired file names
  ! --------------------
  character(len=*), parameter     ::  f2d_Filename = 'data/emissions.nc'

  ! Basic information about the parallel environment
  !       PET = Persistent Execution Threads
  ! In the current implementation, a PET is equivalent
  ! to an MPI process
  ! ------------------------------------------------
  integer                         :: myPET    ! The local PET number
```

```
   integer                         :: nPET    ! The total number of PETs you are running on
55
   integer                         :: status, rc
57 integer                         :: iBndl, nBndls, IM, JM

59 ! Coordinate variables
   ! ---------------------
61 real(kind=8), pointer, dimension(:,:) :: lons, lats

63 character(len=ESMF_MAXSTR)      :: name
   real, pointer, dimension(:,:)   :: fArr, newfArr
65
   character(len=*), parameter     :: Iam = 'Example-003'
67 character(len=*), parameter     :: ConfigFile = 'ex003.rc'

69 call Main()

71 CONTAINS

73 subroutine Main()

75   ! Initialize ESMF. For performance reasons, it is important
     ! to turn OFF ESMF's automatic logging feature
77   ! ------------------------------------------------------------
     call ESMF_Initialize (DefaultLogType=ESMF_LOG_NONE, vm=vm, __RC__)
79
     ! Check the number of processors
81   ! ------------------------------
     call ESMF_VMGet(vm, localPET=myPET, PETcount=nPET)
83   if ( nPET /= 2 ) then
        if ( MAPL_am_I_root() ) then
85         print *, 'Error: expecting 2 PETs but found ', nPET, 'PETs'
           print *, 'Try:   mpirun -np 2 ex001-CFIO_Array.x'
87      end if
        ASSERT_(.FALSE.)
89   end if

91   if ( MAPL_am_I_root() ) then
        print *
93      print *, 'Starting ' // Iam // ' with ', nPET, ' PETs ...'
        print *
95   end if

97   ! MAPL object is populated from config file
     ! -----------------------------------------
99   call MAPL_Set(MAPLobj, ConfigFile=ConfigFile, __RC__)

101  ! Create grid from info in config file
     ! ------------------------------------
103  call MAPL_GridCreate(MAPLOBJ=MAPLobj, ESMFGRID=myGrid, __RC__)
     call ESMF_GridValidate(myGrid)
105

107  ! Get local dimensions from MAPL object
     ! Note: Cannot use MAPL_Get to get the coords lons, lats.
```

```
109     !       That would have been possible if MAPL_Get was called
        !       from inside a MAPL component, whose
111     !       MAPL_GenericInitialize has already been called
        ! ------------------------------------------------------------
113     call MAPL_Get(STATE=MAPLobj, IM=IM, JM=JM, __RC__)

115
        ! Retrieve co-ordinates
117     ! ---------------------
        call ESMF_GridGetCoord (myGrid, fptr=lons, coordDim=1, localDE=0, __RC__)
119     call ESMF_GridGetCoord (myGrid, fptr=lats, coordDim=2, localDE=0, __RC__)

121     ASSERT_(IM==size(LONS,1))
        ASSERT_(JM==size(LONS,2))
123
        ! Set the time as the one on the hardwired file name
125     ! --------------------------------------------------
        call ESMF_CalendarSetDefault (ESMF_CAL_GREGORIAN, __RC__)
127     call ESMF_TimeSet (Time, yy=2001, mm=1, dd=1,  h=0,  m=0, s=0, __RC__)

129     ! Create empty Bundle and read all variables from file into it
        ! ------------------------------------------------------------
131     myBundle = ESMF_FieldBundleCreate (name='Steven', grid=myGrid, __RC__)
        call MAPL_CFIORead (f2d_Filename, Time, myBundle, verbose=.true., __RC__)
133
        !                       ---------------------
135     !                       Looking into a Bundle
        !                       ---------------------
137
        ! Print a summary of what is inside this bound
139     ! --------------------------------------------
        if ( MAPL_Am_I_Root() ) then
141        call ESMF_FieldBundlePrint(myBundle, __RC__)
        end if
143
        ! Get the number of fields in this Bundle...
145     ! ------------------------------------------
        call ESMF_FieldBundleGet(myBundle, fieldCount=nBndls, __RC__)
147
        ! And loop over each field
149     ! ------------------------
        do iBndl = 1, nBndls
151
           ! Get the i-th field
153        ! ------------------
           call ESMF_FieldBundleGet(myBundle, fieldIndex=iBndl, field=ithField, __RC__)
155
           ! Get the field name
157        ! ------------------
           call ESMF_FieldGet(ithField, name=name, __RC__)
159
           ! Get the fortran array inside this field
161        ! ---------------------------------------
           call ESMF_FieldGet(ithField, fArrayptr=fArr, localDE=0, __RC__)
163
```

```
        ! Print summary
        ! -------------
        write(*,'('' ['',i1,''] min, max: '',2(1pe10.2),2x,i3,''x'',i3,3x,a)') &
             myPET, minval(fArr), maxval(fArr), &
             size(fArr,1), size(fArr,2), trim(name)

        ! Notice that modifying the array will modify what is
        ! inside the Bundle
        ! ------------------------------------------------
        fArr = 2. * fArr

      end do

      ! MAPL has handy routines for getting an array out of Bundle
      ! ----------------------------------------------------------
      call ESMFL_BundleGetPointerToData(myBundle, 'ch4', fArr, __RC__)

      ! Print summary - notice factor of 2
      ! ----------------------------------
      write(*,'('' <'',i1,''> min, max: '',2(1pe10.2),2x,i3,''x'',i3,3x,a)') &
           myPET, minval(fArr), maxval(fArr), &
           size(fArr,1), size(fArr,2), 'ch4 (x2)'


      !                      ---------------------
      !                       Making a New Bundle
      !                      ---------------------

      ! Create a new array to go into the new Bundle
      ! --------------------------------------------
      allocate(newfArr(IM,JM),stat=STATUS); VERIFY_(STATUS)
      newfArr = cos(lats) * sin(lons) ! make something up

      ! New array
      ! ---------
      write(*,'('' {'',i1,''} min, max: '',2(1pe10.2),2x,i3,''x'',i3,3x,a)') &
           myPET, minval(newfArr), maxval(newfArr), &
           size(newfArr,1), size(newfArr,2), 'Rolf (newfArr)'

      ! Combine the new fArr and grid into a new Field
      ! ----------------------------------------------
      newField = ESMF_FieldCreate (name='Rolf', grid=myGrid, fArrayptr=newfArr, __RC__)

      ! Create an empty Bundle
      ! ----------------------
      newBundle = ESMF_FieldBundleCreate (name='Denis', grid=myGrid, __RC__ )

      ! Store Field into the Bundle
      ! ---------------------------
      call ESMF_FieldBundleAdd (newBundle, newField, __RC__)

      ! Get fArr back and make sure we got it right
      ! -------------------------------------------
      call ESMFL_BundleGetPointerToData (newBundle, 'Rolf', fArr, __RC__)
```

```
219     newfArr = newfArr - fArr ! should be zero

221     ! Print zeros
        ! -----------
223     write(*,'('' |'',i1,''| min, max: '',2(1pe10.2),2x,i3,''x'',i3,3x,a)') &
            myPET, minval(newfArr), maxval(newfArr), &
225         size(newfArr,1), size(newfArr,2), 'zeros'


227
        ! Clean up (is order important)
229     ! -----------------------------
        call ESMF_FieldBundleDestroy(myBundle)
231     call ESMF_FieldBundleDestroy(newBundle)
        call ESMF_FieldDestroy(newField)
233     call ESMF_GridDestroy(myGrid)
        deallocate(newfArr)
235
        ! All done
237     ! --------
        call ESMF_Finalize(__RC__)
239
    end subroutine Main
241
end Program Example_003
```

A.3 Basic ESMF_Config usage

```
1  #include "MAPL_Generic.h"

3  Program Example_004

5    ! Example 002 - Simple ESMF/MAPL example demonstrating how to
     !
7    ! 1. Initialize ESMF
     ! 2. Create an ESMF Config object and illustrate scalar attributes
9    ! 3. Illustrate vector parameters
     ! 4. Illustrate the use of tables
11   !
     ! It assumes 2 processors, so typically you will run it as
13   !
     ! % mprirun -np 2 ex004-Config.x
15   !
     ! Arlindo da Silva <arlindo.dasilva@nasa.gov>, November 2008
17   !-----------------------------------------------------------------------


19
     use ESMF_Mod
21   use MAPL_Mod

23   implicit NONE

25   ! Basic ESMF objects being used in this example
     ! ---------------------------------------------
27   type(ESMF_Config)       :: cf
     type(ESMF_VM)           :: vm              ! ESMF Virtual Machine
```

```
29
   ! Basic information about the parallel environment
31 !        PET = Persistent Execution Threads
   ! In the current implementation, a PET is equivalent
33 ! to an MPI process
   ! ------------------------------------------------
35 integer :: myPET   ! The local PET number
   integer :: nPET    ! The total number of PETs you are running on
37
   ! Information to be retrieved from resource (config) file
39 ! ------------------------------------------------------
   character(len=ESMF_MAXSTR), pointer :: cTable(:,:)
41 real, pointer                       :: rTable(:,:)
   integer                             :: Nx, Ny, Nz, nLines, nCols
43 integer                             :: BegDate(2), EndDate(2)

45 integer :: status, rc
   integer :: i, j, n
47
   character(len=*), parameter :: Iam = 'Example-004'
49 character(len=*), parameter :: ConfigFile = 'ex004.rc'

51 call Main()

53 CONTAINS

55 subroutine Main()

57   ! Initialize the ESMF. For performance reasons, it is important
     ! to turn OFF ESMF's automatic logging feature
59   ! ------------------------------------------------------------
     call ESMF_Initialize (DefaultLogType=ESMF_LOG_NONE, vm=vm, __RC__)
61
     ! Check the number of processors
63   ! ------------------------------
     call ESMF_VMGet(vm, localPET=myPET, PETcount=nPET)
65   if ( nPET /= 2 ) then
        if ( MAPL_am_I_root() ) then
67         print *, 'Error: expecting 2 PETs but found ', nPET, 'PETs'
           print *, 'Try:   mpirun -np 2 ex004-Config.x'
69      end if
        ASSERT_(.FALSE.)
71   end if

73   if ( MAPL_am_I_root() ) then
        print *
75      print *, 'Starting ' // Iam // ' with ', nPET, ' PETs ...'
        print *
77   end if

79   ! Start by load a simple resource file
     ! ------------------------------------
81   cf = ESMF_ConfigCreate()
     call ESMF_ConfigLoadFile(cf, fileName=ConfigFile, __RC__)
83
```

```
!                                 ------------------
!                                 Scalar Resources
!                                 ------------------

! Get a couple of integers
! ------------------------
call ESMF_ConfigGetAttribute(cf, Nx, label='NX:', __RC__)
call ESMF_ConfigGetAttribute(cf, Ny, label='NY:', default=2, __RC__)  ! 2 is the default
call ESMF_ConfigGetAttribute(cf, Nz, label='NZ:', default=1, __RC__)  ! 1 is the default

write(*,'('' ['',i1,''] Layout: '',i2,'' x'',i2,'' x'',i2,2x,a)') &
     myPET, Nx, Ny, Nz, '(Nx x Ny x Nz)'


! Can also modify the value of an attribute (that is present in cf)
! -----------------------------------------------------------------
call ESMF_ConfigSetAttribute(cf, 4, label='NX:', __RC__)
call ESMF_ConfigSetAttribute(cf, 2, label='NY:', __RC__)

call ESMF_ConfigGetAttribute(cf, Nx, label='NX:', __RC__)
call ESMF_ConfigGetAttribute(cf, Ny, label='NY:', __RC__)

write(*,'('' ['',i1,''] Modified Layout: '',i2,'' x'',i2,'' x'',i2,2x,a)') &
     myPET, Nx, Ny, Nz, '(Nx x Ny x Nz)'



!                                 ------------------
!                                 Vector Resources
!                                 ------------------

call ESMF_ConfigGetAttribute(cf, BegDate, 2, 'BEG_DATE:', default=0 )
call ESMF_ConfigGetAttribute(cf, EndDate, 2, 'END_DATE:', default=0 )

write(*,'('' ['',i1,''] Beginning Date: '',2i10)') myPET, BegDate
write(*,'('' ['',i1,''] Ending    Date: '',2i10)') myPET, EndDate

!                                 ------------------
!                                      Tables
!                                 ------------------

! 1) 7x3 Table with real numbers
!    ---------------------------

! First get the size of the table
! ...............................
call ESMF_ConfigGetDim(cf, nLines, nCols, 'Statistics::',__RC__)
write(*,'('' ['',i1,''] Table size: '',i2,'' x'',i2,a)') &
     myPET, nLines, nCols, ' [Statistics]'

! Read the Table
! ..............
allocate(rTable(nLines,nCols))
call ESMF_ConfigFindLabel(cf, 'Statistics::',__RC__)
do i = 1, nLines
```

```
139         call ESMF_ConfigNextLine(cf, __RC__)
            call ESMF_ConfigGetAttribute(cf, rTable(i,:), nCols, __RC__)
141      end do

143      if ( MAPL_Am_I_Root() ) then
            do i = 1, nLines
145            print *, rTable(i,:)
            end do
147      end if

149      ! 2) Variable column table with strings
         !    ---------------------------------

151
         ! First get the size of the table
153      ! ...............................
         call ESMF_ConfigGetDim(cf, nLines, nCols, 'diag_sfc.fields::',__RC__)
155      write(*,'('' ['',i1,''] Table size: '',i2,'' x'',i2,a)') &
              myPET, nLines, nCols, '[diag_sfc.fields]'
157
         ! Read the Table
159      ! ..............
         allocate(cTable(nLines,nCols))
161      call ESMF_ConfigFindLabel(cf, 'diag_sfc.fields::',__RC__)
         do i = 1, nLines
163         call ESMF_ConfigNextLine(cf, rc=rc)
            do j = 1, nCols
165            call ESMF_ConfigGetAttribute(cf, cTable(i,j), default='-')
            end do
167      end do

169      if ( MAPL_Am_I_Root() ) then
            do i = 1, nLines
171            write(*,'(1x,3a12)') (trim(cTable(i,j)), j=1,nCols)
            end do
173      end if


175
         ! Clean up
177      ! --------
         call ESMF_ConfigDestroy(cf)
179      deallocate(rTable,cTable)

181      ! All done
         ! --------
183      call ESMF_Finalize(__RC__)

185   end subroutine Main

187 end Program Example_004
```

A.4 Stand-alone gridded component

```
#include "MAPL_Generic.h"
2

Program Example_005
```

```
4
   ! Example 005 - Simple ESMF/MAPL example demonstrating how to
6  !
   ! 1. Initialize ESMF
8  ! 2. Create a 2D grid, validate it
   ! 3. Create clock
10 ! 4. Create IM/EX states and the gridded component
   ! 5. Run component's SetServices
12 ! 6. Initialize the component
   ! 7. Run the component
14 ! 8. Clean up
   !
16 ! It assumes 2 processors, so typically you will run it as
   !
18 ! % mpirun -np 2 ex005-GridComp.x
   !
20 ! Arlindo da Silva <arlindo.dasilva@nasa.gov>, October 2008
   !-----------------------------------------------------------------------
22
   use ESMF_Mod
24 use MAPL_Mod
   use Example_GridCompMod, only: ExSetServices => SetServices
26
   implicit NONE
28
   ! Basic ESMF objects being used in this example
30 ! ---------------------------------------------
   type(ESMF_Grid)           :: myGrid     ! Grid
32 type(ESMF_VM)             :: vm         ! ESMF Virtual Machine
   type(ESMF_Time)           :: Time       ! Time objects
34 type(ESMF_TimeInterval)   :: TimeStep   ! used to define a clock

36 ! All important MAPL object
   ! ----------------------
38 type(MAPL_MetaComp) :: MAPLobj


40
   ! Grid Component Objects
42 ! ----------------------
   type(ESMF_GridComp) :: GC
44 type(ESMF_State)    :: IMPORT
   type(ESMF_State)    :: EXPORT
46 type(ESMF_Clock)    :: CLOCK

48 ! Basic information about the parallel environment
   !       PET = Persistent Execution Threads
50 ! In the current implementation, a PET is equivalent
   ! to an MPI process
52 ! ---------------------------------------------------
   integer :: myPET   ! The local PET number
54 integer :: nPET    ! The total number of PETs you are running on

56 integer :: status, rc
   integer :: i, j, n, im, jm
58
```

```
   ! Coordinate variables
60 ! ---------------------
   real(kind=8), pointer, dimension(:,:) :: lons, lats
62
   character(len=ESMF_MAXSTR)      :: name
64 real, pointer, dimension(:,:) :: Array, newArray

66 character(len=*), parameter :: Iam = 'Example-005'
   character(len=*), parameter :: ConfigFile = 'ex005.rc'
68
   call Main()
70
CONTAINS
72
   subroutine Main()
74
     ! Initialize the ESMF. For performance reasons, it is important
76   ! to turn OFF ESMF's automatic logging feature
     ! ------------------------------------------------------------
78   call ESMF_Initialize (DefaultLogType=ESMF_LOG_NONE, vm=vm, __RC__)

80   ! Check the number of processors
     ! ------------------------------
82   call ESMF_VMGet(vm, localPET=myPET, PETcount=nPET, __RC__)
     if ( nPET /= 2 ) then
84      if ( MAPL_am_I_root() ) then
           print *, 'Error: expecting 2 PETs but found ', nPET, 'PETs'
86         print *, 'Try:   mpirun -np 2 ./ex005-GridComp.x'
        end if
88      ASSERT_(.FALSE.)
     end if
90
     if ( MAPL_am_I_root() ) then
92      print *
        print *, 'Starting ' // Iam // ' with ', nPET, ' PETs ...'
94      print *
     end if
96
     ! MAPL object is populated from config file
98   ! -----------------------------------------
     call MAPL_Set(MAPLobj, ConfigFile=ConfigFile, __RC__)
100
     ! Create grid from info in MAPL object and validate it
102  ! ----------------------------------------------------
     call MAPL_GridCreate(MAPLOBJ=MAPLobj, ESMFGRID=myGrid, __RC__)
104  call ESMF_GridValidate(myGrid, __RC__)


106
     ! Get local dimensions from MAPL object
108  ! Note: Cannot use MAPL_Get to get the coords lons, lats.
     !       That would have been possible if MAPL_Get was called
110  !       from inside a MAPL component, whose
     !       MAPL_GenericInitialize has already been called
112  ! ------------------------------------------------------------
     call MAPL_Get(STATE=MAPLobj, IM=IM, JM=JM, __RC__)
```

```
114
      ! Retrieve co-ordinates
116   ! ----------------------
      call ESMF_GridGetCoord (myGrid, fptr=lons, coordDim=1, localDE=0, __RC__)
118   call ESMF_GridGetCoord (myGrid, fptr=lats, coordDim=2, localDE=0, __RC__)

120   ASSERT_(IM==size(LONS,1))
      ASSERT_(JM==size(LONS,2))
122
      ! Create a clock starting at 1/1/2001 0Z with a 30 min time step
124   ! --------------------------------------------------------------
      call ESMF_CalendarSetDefault (ESMF_CAL_GREGORIAN, __RC__)
126   call ESMF_TimeSet (Time, yy=2001, mm=1, dd=1,  h=0,  m=0, s=0, __RC__)
      call ESMF_TimeIntervalSet (TimeStep, h=0, m=30, s=0, __RC__)
128   CLOCK = ESMF_ClockCreate ("Clovis", timeStep=TimeStep, startTime=Time, __RC__)

130
      ! Create states and the component
132   ! -------------------------------
      IMPORT = ESMF_StateCreate ( 'Ivan', __RC__ )
134   EXPORT = ESMF_StateCreate ( 'Evan', __RC__ )
      GC = ESMF_GridCompCreate ( &
136        name='Gilbert', Grid=myGrid, GridCompType = ESMF_ATM, &
           ConfigFile='ex005_GridComp.rc', __RC__  )
138
      ! Set component services
140   ! ----------------------
      call ESMF_GridCompSetServices ( GC, ExSetServices, __RC__ )
142
      ! Initialize component
144   ! --------------------
      call ESMF_GridCompInitialize ( GC, IMPORT, EXPORT, CLOCK, __RC__ )
146
      ! Look at states
148   ! --------------
      if ( MAPL_AM_I_ROOT() ) then
150      !call ESMF_StatePrint(IMPORT)
         !call ESMF_StatePrint(EXPORT)
152   end if

154   ! Run component
      ! -------------
156   call ESMF_GridCompRun ( GC, IMPORT, EXPORT, CLOCK, __RC__ )

158   ! Finalize component
      ! ------------------
160   call ESMF_GridCompFinalize ( GC, IMPORT, EXPORT, CLOCK, __RC__ )

162   ! All done
      ! --------
164   call ESMF_Finalize(__RC__)

166 end subroutine Main

168 end Program Example_005
```

A.4.1 Module Example_GridCompMod

USES:

```
use ESMF_Mod
use MAPL_Mod
```

PUBLIC MEMBER FUNCTIONS:

```
public SetServices
```

DESCRIPTION:
`Example_GridComp` is an ESMF gridded component implementing the Example chemical processes.

A.4.1.1 SetServices — Sets IRF services for the Example Grid Component

INTERFACE:

```
subroutine SetServices ( GC, RC )
```

USES:

ARGUMENTS:

```
type(ESMF_GridComp), intent(INOUT) :: GC  ! gridded component
integer, optional                  :: RC  ! return code
```

DESCRIPTION:
Sets Initialize, Run and Finalize services. *STATES:*

The following is a list of `Import`, `Export` and `Internal` states (second column specifies the type):

Short Name	Type	Units	Dims	Vert Loc	Long name
im_st	IM	uim	HorzOnly	Center	importstate
in_st	IN	uin	HorzVert	Center	internalstate
ex_st	EX	u ex	HorzVert	Center	exportstate

A.4.1.2 Run_ — Runs Example

INTERFACE:

```
subroutine Run_ ( gc, IMPORT, EXPORT, CLOCK, rc )
```

USES:

ARGUMENTS:

```
type(ESMF_GridComp), intent(inout) :: GC      ! Grid Component
type(ESMF_State),    intent(inout) :: IMPORT  ! Import State
type(ESMF_State),    intent(inout) :: EXPORT  ! Export State
type(ESMF_Clock),    intent(inout) :: CLOCK   ! The clock
integer,             intent(out)   :: rc      ! Error return code:
                                              !  0 - all is well
                                              !  1 -
```

DESCRIPTION:
This is a simple ESMF wrapper.

A.5 Complete Mapl example

A.5.1 Gridded Component: ChildTwoTwo

```
#include "MAPL_Generic.h"

module ChildTwoTwoGridCompMod

  use ESMF_Mod
  use MAPL_Mod

  implicit none
  private

  public SetServices

contains

  !-----------------------------------------------------------------------------!

  subroutine SetServices(GC, RC)

    type(ESMF_GridComp), intent(INOUT) :: GC  ! gridded component
    integer, optional,   intent(  OUT) :: RC  ! return code

    ! ErrLog
    ! ------
```

```
      character(len=ESMF_MAXSTR)              :: Iam
25    integer                                 :: status
      character(len=ESMF_MAXSTR)              :: comp_name
27
      ! Get my name and set-up traceback handle
29    ! ---------------------------------------
      call ESMF_GridCompGet(GC, name=comp_name, __RC__)
31    Iam = trim(COMP_NAME)//'::'//"SetServices"

33    ! Register services for this component (initialize/finalize
      ! default to their generic versions)
35    ! ----------------------------------------------------------
      call MAPL_GridCompSetEntryPoint(GC, ESMF_SETRUN, ChildTwoTwoRun, __RC__)
37
      ! Import states
39    ! -------------
      call MAPL_AddImportSpec(                    &
41         GC,                                    &
           SHORT_NAME = 'RADSWC',                 &
43         LONG_NAME  = 'R A D S W C',            &
           UNITS      = 'u',                      &
45         DIMS       = MAPL_DimsHorzVert,        &
           VLOCATION  = MAPL_VLocationCenter,     &
47         __RC__)

49    ! Export states
      ! -------------
51    call MAPL_AddExportSpec(                    &
           GC,                                    &
53         SHORT_NAME = 'H2O_TEND',               &
           LONG_NAME  = 'H 2 O T E N D',          &
55         UNITS      = 'u',                      &
           DIMS       = MAPL_DimsHorzVert,        &
57         VLOCATION  = MAPL_VLocationCenter,     &
           __RC__)
59
      call MAPL_AddExportSpec(                    &
61         GC,                                    &
           SHORT_NAME = 'OX_TEND',                &
63         LONG_NAME  = 'O X T E N D',            &
           UNITS      = 'u',                      &
65         DIMS       = MAPL_DimsHorzOnly,        &
           VLOCATION  = MAPL_VLocationCenter,     &
67         __RC__)

69    ! Generic SetServices. Among other things, this allocates an
      ! instance of MAPL object (also called generic state), wraps it
71    ! and sets it as the GC's internal state, which can be retrieved
      ! later through a call to MAPL_GetObjectFromGC.
73    ! ------------------------------------------------------------
      call MAPL_GenericSetServices(GC, __RC__)
75
      ! All done
77    ! --------
      RETURN_(ESMF_SUCCESS)
```

```
79
   end subroutine SetServices
81
   !-------------------------------------------------------------------------!
83
   subroutine ChildTwoTwoRun(GC, Import, Export, Clock, rc)
85
     type(ESMF_GridComp), intent(inout) :: GC       ! Child TwoTwo Grid Comp
87   type(ESMF_State),    intent(inout) :: Import   ! Import state
     type(ESMF_State),    intent(inout) :: Export   ! Export state
89   type(ESMF_Clock),    intent(inout) :: Clock    ! The clock
     integer, optional,   intent(  out) :: rc       ! Error code:
91                                                  ! = 0 all is well
                                                    ! otherwise, error
93
     ! ErrLog
95   ! ------
     character(len=ESMF_MAXSTR)          :: Iam
97   integer                             :: status
     character(len=ESMF_MAXSTR)          :: comp_name
99
     ! MAPL object
101  ! -----------
     type(MAPL_MetaComp), pointer        :: MAPLobj  ! MAPL object
103
     ! Pointers to import/export/internal states
105  ! -----------------------------------------
     real, pointer                       :: ptr_h2o(:,:,:) => null() ! export
107  real, pointer                       :: ptr_ox(:,:)    => null() ! export
     real, pointer                       :: ptr_swc(:,:,:) => null()    ! import
109
     ! temporary arrays (to be copied to export)
111  ! -----------------------------------------
     real, allocatable                   :: tmp_h2o(:,:,:)
113  real, allocatable                   :: tmp_ox(:,:)

115  ! grid, config, internal state, dimensions and counters
     ! -----------------------------------------------------
117  type(ESMF_Grid)                     :: myGrid
     integer                             :: imloc, jmloc, lm
119  integer                             :: ii, jj, ll

121  ! Get my name and set-up traceback handle
     ! ---------------------------------------
123  call ESMF_GridCompGet(GC, name=comp_name, grid=myGrid, __RC__)
     Iam = trim(comp_name)//'::'//"Run"
125
     ! Get MAPL object from GC
127  ! -----------------------
     call MAPL_GetObjectFromGC(GC, MAPLobj, __RC__)
129
     ! Get local dimensions from MAPL obj
131  ! ----------------------------------
     call MAPL_Get(STATE=MAPLobj, IM = imloc, JM = jmloc, LM = lm)
133
```

```
      ! Allocate temporary arrays
135   ! Allocate/deallocate at every step?? Maybe use an internal state
      ! Or, maybe this can be done in Initialize
137   ! ---------------------------------------------------------------
      allocate(tmp_h2o(imloc, jmloc, lm)); tmp_h2o = 0.0
139   allocate(tmp_ox(imloc, jmloc));      tmp_ox  = 0.0

141   ! Do something with these temp arrays
      ! -----------------------------------
143   do ii=1,imloc
         do jj=1,jmloc
145         tmp_ox(ii,jj) = 4.44
            do ll=1,lm
147            tmp_h2o(ii,jj,ll) = 5.55
            end do
149      end do
      end do
151
      ! Get pointers to export states and fill out
153   ! ------------------------------------------
      call MAPL_GetPointer(Export, ptr_ox,  'OX_TEND',  __RC__)
155   if(associated(ptr_ox))  ptr_ox  = tmp_ox
      call MAPL_GetPointer(Export, ptr_h2o, 'H2O_TEND', __RC__)
157   if(associated(ptr_h2o)) ptr_h2o = tmp_h2o

159   ! We can deallocate temporary arrays. They have been
      ! deep-copied into the variable referenced by 'ee'..
161   ! --------------------------------------------------
      deallocate(tmp_ox, tmp_h2o)
163
      ! We should have access to import states
165   ! --------------------------------------
      call MAPL_GetPointer(Import, ptr_swc, 'RADSWC', __RC__)
167
      if (MAPL_am_I_root()) then
169      print *, ''
         print *, '---------------------------------------'
171      print *, 'ChildTwoTwo:'
         print *, 'Import spec   RADSWC:', ptr_swc(1,1,1)
173      if (associated(ptr_h2o)) print *, 'Export spec H2O_TEND:', ptr_h2o(1,1,1)
         if (associated(ptr_ox))  print *, 'Export spec  OX_TEND:', ptr_ox (1,1)
175      print *, '---------------------------------------'
      end if
177

179   ! All done
      ! --------
181   RETURN_(ESMF_SUCCESS)

183 end subroutine ChildTwoTwoRun

185 !-----------------------------------------------------------------------------!

187 end module ChildTwoTwoGridCompMod
```

A.5.2 Gridded Component: ChildTwoOne

```
#include "MAPL_Generic.h"

module ChildTwoOneGridCompMod

   use ESMF_Mod
   use MAPL_Mod

   implicit none
   private

   public SetServices

contains

   !-----------------------------------------------------------------------------!

   subroutine SetServices(GC, RC)

     type(ESMF_GridComp), intent(INOUT)  :: GC  ! gridded component
     integer, optional,   intent(  OUT)  :: RC  ! return code

     ! ErrLog
     ! ------
     character(len=ESMF_MAXSTR)          :: Iam
     integer                             :: status
     character(len=ESMF_MAXSTR)          :: comp_name

     ! Get my name and set-up traceback handle
     ! ---------------------------------------
     call ESMF_GridCompGet(GC, name=comp_name, __RC__)
     Iam = trim(COMP_NAME)//'::'//"SetServices"

     ! Register services for this component (initialize/finalize
     ! default to their generic versions)
     ! ----------------------------------------------------------
     call MAPL_GridCompSetEntryPoint(GC, ESMF_SETRUN, ChildTwoOneRun, __RC__)

     ! Import states
     ! -------------
     call MAPL_AddImportSpec(                    &
          GC,                                    &
          SHORT_NAME = 'UX',                     &
          LONG_NAME  = 'U X',                    &
          UNITS      = 'u',                      &
          DIMS       = MAPL_DimsHorzOnly,        &
          VLOCATION  = MAPL_VLocationCenter,     &
          __RC__)

     ! Export states
     ! -------------
     call MAPL_AddExportSpec(                    &
          GC,                                    &
          SHORT_NAME = 'RADLWC',                 &
```

```
        LONG_NAME  = 'R A D L W C',              &
55      UNITS      = 'u',                        &
        DIMS       = MAPL_DimsHorzVert,          &
57      VLOCATION  = MAPL_VLocationCenter,       &
        __RC__)
59
    call MAPL_AddExportSpec(                     &
61      GC,                                      &
        SHORT_NAME = 'RADSWC',                   &
63      LONG_NAME  = 'R A D S W C',              &
        UNITS      = 'u',                        &
65      DIMS       = MAPL_DimsHorzVert,          &
        VLOCATION  = MAPL_VLocationCenter,       &
67      __RC__)

69  ! Internal state
    ! --------------
71  call MAPL_AddInternalSpec(                   &
        GC,                                      &
73      SHORT_NAME = 'ABCINT',                   &
        LONG_NAME  = 'A B C I N T',              &
75      UNITS      = 'u',                        &
        DIMS       = MAPL_DimsHorzVert,          &
77      VLOCATION  = MAPL_VLocationCenter,       &
        __RC__)
79
    ! Generic SetServices. Among other things, this allocates an
81  ! instance of MAPL object (also called generic state), wraps it
    ! and sets it as the GC's internal state, which can be retrieved
83  ! later through a call to MAPL_GetObjectFromGC.
    ! ------------------------------------------------------------
85  call MAPL_GenericSetServices(GC, __RC__)

87  ! All done
    ! --------
89  RETURN_(ESMF_SUCCESS)

91 end subroutine SetServices

93 !-------------------------------------------------------------------------!

95 subroutine ChildTwoOneRun(GC, Import, Export, Clock, rc)

97   type(ESMF_GridComp), intent(inout) :: GC     ! Child TwoOne Grid Comp
     type(ESMF_State),    intent(inout) :: Import ! Import state
99   type(ESMF_State),    intent(inout) :: Export ! Export state
     type(ESMF_Clock),    intent(inout) :: Clock  ! The clock
101  integer, optional,   intent(  out) :: rc     ! Error code:
                                                  ! = 0 all is well
103                                               ! otherwise, error

105  ! ErrLog
     ! ------
107  character(len=ESMF_MAXSTR)         :: Iam
     integer                            :: status
```

```
109     character(len=ESMF_MAXSTR)          :: comp_name

111     ! MAPL object
        ! -------------
113     type(MAPL_MetaComp), pointer        :: MAPLobj  ! MAPL object

115     ! Pointers to import/export/internal states
        ! -----------------------------------------
117     real, pointer                       :: ptr_lwc(:,:,:) => null() ! export
        real, pointer                       :: ptr_swc(:,:,:) => null() ! export
119     real, pointer                       :: ptr_ux(:,:)    => null() ! import
        real, pointer                       :: ptr_abc(:,:,:) => null() ! internal
121
        ! temporary arrays (to be copied to export)
123     ! -----------------------------------------
        real, allocatable                   :: tmp_lwc(:,:,:)
125     real, allocatable                   :: tmp_swc(:,:,:)

127     ! grid, config, internal state, dimensions and counters
        ! -----------------------------------------------------
129     type(ESMF_Grid)                     :: myGrid
        integer                             :: imloc, jmloc, lm
131     integer                             :: ii, jj, ll
        type(ESMF_State)                    :: IntState
133
        ! Get my name and set-up traceback handle
135     ! ---------------------------------------
        call ESMF_GridCompGet(GC, name=comp_name, grid=myGrid, __RC__)
137     Iam = trim(comp_name)//'::'//"Run"

139     ! Get MAPL object from GC
        ! -----------------------
141     call MAPL_GetObjectFromGC(GC, MAPLobj, __RC__)

143     ! Get local dimensions from MAPL obj
        ! ----------------------------------
145     call MAPL_Get(STATE=MAPLobj, IM = imloc, JM = jmloc, LM = lm)

147     ! Allocate temporary arrays
        ! Allocate/deallocate at every step?? Maybe use an internal state
149     ! or maybe this can be done in initialize
        ! -----------------------------------------------------------
151     allocate(tmp_lwc(imloc, jmloc, lm))
        allocate(tmp_swc(imloc, jmloc, lm))
153
        ! Do something with these temp arrays
155     ! -----------------------------------
        do ii=1,imloc
157        do jj=1,jmloc
              do ll=1,lm
159              tmp_lwc(ii,jj,ll) = 1.11
                 tmp_swc(ii,jj,ll) = 2.22
161           end do
           end do
163     end do
```

```
165     ! Get pointers to export states and fill out
        ! ------------------------------------------
167     call MAPL_GetPointer(Export, ptr_lwc, 'RADLWC', __RC__)
        if(associated(ptr_lwc)) ptr_lwc = tmp_lwc
169     call MAPL_GetPointer(Export, ptr_swc, 'RADSWC', __RC__)
        if(associated(ptr_swc)) ptr_swc = tmp_swc
171
        ! We can deallocate temporary arrays. They have been
173     ! deep-copied into the variable referenced by 'ee'..
        ! --------------------------------------------------
175     deallocate(tmp_lwc,tmp_swc)

177     ! Access the internal states. Recall that an internal
        ! state is 'attached' to a gridded component and can
179     ! be retrieved from the gc by querying its MAPL object
        ! ----------------------------------------------------
181     call MAPL_Get(MAPLobj, INTERNAL_ESMF_STATE=IntState, __RC__)
        call MAPL_GetPointer(IntState, ptr_abc, 'ABCINT', __RC__)
183     do ii=1,imloc
           do jj=1,jmloc
185           do ll=1,lm
                 ptr_abc(ii,jj,ll) = 3.33
187           end do
           end do
189     end do

191     ! We should have access to import states
        ! --------------------------------------
193     call MAPL_GetPointer(Import, ptr_ux,  'UX',  __RC__)

195     ! Print out some values at every time step
        ! ----------------------------------------
197     if (MAPL_am_I_root()) then
           print *, ''
199        print *, '---------------------------------------'
           print *, 'ChildTwoOne:'
201        print *, 'Import   spec     UX:', ptr_ux (1,1)
           print *, 'IntState spec ABCINT:', ptr_abc(1,1,1)
203        if (associated(ptr_lwc)) print *, 'Export   spec RADLWC:', ptr_lwc(1,1,1)
           if (associated(ptr_swc)) print *, 'Export   spec RADSWC:', ptr_swc(1,1,1)
205        print *, '---------------------------------------'
        end if
207

209     ! All done
        ! --------
211     RETURN_(ESMF_SUCCESS)

213   end subroutine ChildTwoOneRun

215   !-----------------------------------------------------------------------------!

217 end module ChildTwoOneGridCompMod
```

A.5.3 Gridded Component: ChildTwo

```
#include "MAPL_Generic.h"

module ChildTwoGridCompMod
   use ESMF_Mod
   use MAPL_Mod

   use ChildTwoOneGridCompMod, only: TwoOne_SetServices => SetServices
   use ChildTwoTwoGridCompMod, only: TwoTwo_SetServices => SetServices

   implicit none
   private

   public SetServices

   ! (Global) integers representing children components
   ! ---------------------------------------------------
   integer                                 :: ChildTwoOne
   integer                                 :: ChildTwoTwo

contains

   !-----------------------------------------------------------------------------!

   subroutine SetServices(GC, RC)

     type(ESMF_GridComp), intent(INOUT)  :: GC  ! gridded component
     integer, optional,   intent(  OUT)  :: RC  ! return code

     ! ErrLog
     ! ------
     character(len=ESMF_MAXSTR)          :: Iam
     integer                             :: status
     character(len=ESMF_MAXSTR)          :: comp_name

     ! Get my name and set-up traceback handle
     ! ---------------------------------------
     call ESMF_GridCompGet(GC, name=comp_name, __RC__)
     Iam = trim(COMP_NAME)//'::'//"SetServices"

     ! Register services for this component
     ! -------------------------------------
     call MAPL_GridCompSetEntryPoint(GC, ESMF_SETRUN, ChildTwoRun, __RC__)

     ! Create children
     ! ---------------
     ChildTwoOne = MAPL_AddChild(GC, name='ChildTwoOne', ss=TwoOne_SetServices, __RC__)
     ChildTwoTwo = MAPL_AddChild(GC, name='ChildTwoTwo', ss=TwoTwo_SetServices, __RC__)

     ! Register connectivities between children
     ! ----------------------------------------
     call MAPL_AddConnectivity(          &
          GC,                            &
          SHORT_NAME = (/ 'RADSWC' /), &
```

```
         SRC_ID      = ChildTwoOne,     &
         DST_ID      = ChildTwoTwo,     &
         __RC__)

    ! Unlike imports, exports don't bubble up. Since ChildOne
    ! has an import 'OX_TEND' that is ChildTwoTwo's export, we
    ! need to make this variable available to ChildOne. See
    ! GEOS_SuperdynGridComp.F90 for examples.
    ! ------------------------------------------------------
    call MAPL_AddExportSpec(            &
         GC,                            &
         SHORT_NAME = 'OX_TEND',        &
         CHILD_ID   = ChildTwoTwo,      &
         __RC__)

    ! Generic SetServices. Among other things, this allocatates an
    ! instance of MAPL object (also called generic state), wraps it
    ! and sets it as the GC's internal state, which can be retrieved
    ! later through a call to MAPL_GetObjectFromGC.
    ! ------------------------------------------------------------
    call MAPL_GenericSetServices(GC, __RC__)

    ! All done
    ! --------
    RETURN_(ESMF_SUCCESS)

end subroutine SetServices

!-----------------------------------------------------------------------------!

subroutine ChildTwoRun(GC, Import, Export, Clock, rc)

  type(ESMF_GridComp), intent(inout) :: GC       ! Child Two  Grid Comp
  type(ESMF_State),    intent(inout) :: Import   ! Import state
  type(ESMF_State),    intent(inout) :: Export   ! Export state
  type(ESMF_Clock),    intent(inout) :: Clock    ! The clock
  integer, optional,   intent(  out) :: rc       ! Error code:
                                                 ! = 0 all is well
                                                 ! otherwise, error

    ! ErrLog and local variables
    ! --------------------------
    character(len=ESMF_MAXSTR)          :: Iam
    integer                             :: status
    character(len=ESMF_MAXSTR)          :: comp_name
    type(MAPL_MetaComp), pointer        :: MAPLobj  ! MAPL object
    type(ESMF_Grid)                     :: myGrid
    type(ESMF_GridComp), pointer        :: GCS(:)
    type(ESMF_State),    pointer        :: GIM(:), GEX(:)

    ! Get my name and set-up traceback handle
    ! ---------------------------------------
    call ESMF_GridCompGet(GC, name=comp_name, grid=myGrid, __RC__)
    Iam = trim(comp_name)//'::'//"Run"
```

```
109     ! Get MAPL object from GC
        ! ------------------------
111     call MAPL_GetObjectFromGC(GC, MAPLobj, __RC__)

113     ! Get children and their IM/EX states from MAPL object
        ! ----------------------------------------------------
115     call MAPL_Get(MAPLobj, GCS=GCS, GIM=GIM, GEX=GEX, __RC__)

117     ! Do we need to do something about clock??

119     ! Run children
        ! ------------
121     call ESMF_GridCompRun(GCS(ChildTwoOne), GIM(ChildTwoOne), GEX(ChildTwoOne), Clock, __RC__)
        call ESMF_GridCompRun(GCS(ChildTwoTwo), GIM(ChildTwoTwo), GEX(ChildTwoTwo), Clock, __RC__)
123
        ! All done
125     ! --------
        RETURN_(ESMF_SUCCESS)
127
    end subroutine ChildTwoRun
129
    !-----------------------------------------------------------------------------!
131
end module ChildTwoGridCompMod
```

A.5.4 Gridded Component: ChildOne

```
#include "MAPL_Generic.h"
2
module ChildOneGridCompMod
4
   use ESMF_Mod
6  use MAPL_Mod

8  implicit none
   private
10
   public SetServices
12
contains
14
   !-----------------------------------------------------------------------------!
16
   subroutine SetServices(GC, RC)
18
     type(ESMF_GridComp), intent(INOUT)  :: GC  ! gridded component
20   integer, optional,   intent(  OUT)  :: RC  ! return code

22   ! ErrLog and Local Variables
     ! --------------------------
24   character(len=ESMF_MAXSTR)          :: Iam
     integer                             :: status
26   character(len=ESMF_MAXSTR)          :: comp_name

28   ! Get my name and set-up traceback handle
     ! ---------------------------------------
```

```
30      call ESMF_GridCompGet(GC, name=comp_name, __RC__)
        Iam = trim(COMP_NAME)//'::'//"SetServices"
32
        ! Register services for this component
34      ! ------------------------------------
        call MAPL_GridCompSetEntryPoint(GC, ESMF_SETRUN, ChildOneRun, __RC__)
36
        ! Import states
38      ! -------------
        call MAPL_AddImportSpec(                    &
40           GC,                                    &
             SHORT_NAME = 'OX_TEND',                &
42           LONG_NAME  = 'O X T E N D',            &
             UNITS      = 'u',                      &
44           DIMS       = MAPL_DimsHorzOnly,        &
             VLOCATION  = MAPL_VLocationCenter,     &
46           __RC__)

48      ! Export states
        ! -------------
50      call MAPL_AddExportSpec(                    &
             GC,                                    &
52           SHORT_NAME = 'UX',                     &
             LONG_NAME  = 'U X',                    &
54           UNITS      = 'u',                      &
             DIMS       = MAPL_DimsHorzOnly,        &
56           VLOCATION  = MAPL_VLocationCenter,     &
             __RC__)
58
        call MAPL_AddExportSpec(                    &
60           GC,                                    &
             SHORT_NAME = 'UY',                     &
62           LONG_NAME  = 'U Y',                    &
             UNITS      = 'u',                      &
64           DIMS       = MAPL_DimsHorzVert,        &
             VLOCATION  = MAPL_VLocationCenter,     &
66           __RC__)

68      call MAPL_AddExportSpec(                    &
             GC,                                    &
70           SHORT_NAME = 'UZ',                     &
             LONG_NAME  = 'U Z',                    &
72           UNITS      = 'u',                      &
             DIMS       = MAPL_DimsHorzVert,        &
74           VLOCATION  = MAPL_VLocationCenter,     &
             __RC__)
76
        ! Generic SetServices. Among other things, this allocates an
78      ! instance of MAPL object (also called generic state), wraps it
        ! and sets it as the GC's internal state, which can be retrieved
80      ! later through a call to MAPL_GetObjectFromGC.
        ! ------------------------------------------------------------
82      call MAPL_GenericSetServices(GC, __RC__)

84      ! All done
```

```
    ! ________
    RETURN_(ESMF_SUCCESS)

end subroutine SetServices

!-------------------------------------------------------------------------!

subroutine ChildOneRun(GC, Import, Export, Clock, rc)

  type(ESMF_GridComp), intent(inout) :: GC       ! Child One  Grid Comp
  type(ESMF_State),    intent(inout) :: Import   ! Import state
  type(ESMF_State),    intent(inout) :: Export   ! Export state
  type(ESMF_Clock),    intent(inout) :: Clock    ! The clock
  integer, optional,   intent(  out) :: rc       ! Error code:
                                                 ! = 0 all is well
                                                 ! otherwise, error

  ! ErrLog
  ! ______
  character(len=ESMF_MAXSTR)         :: Iam
  integer                            :: status
  character(len=ESMF_MAXSTR)         :: comp_name

  ! MAPL object
  ! ___________
  type(MAPL_MetaComp), pointer       :: MAPLobj  ! MAPL object

  ! Pointers to import/export/internal states
  ! _________________________________________
  real, pointer                      :: ptr_ux(:,:)   => null() ! export
  real, pointer                      :: ptr_uy(:,:,:) => null() ! export
  real, pointer                      :: ptr_uz(:,:,:) => null() ! export
  real, pointer                      :: ptr_ox(:,:)   => null() ! import

  ! temporary arrays (to be copied to export)
  ! _________________________________________
  real, allocatable                  :: tmp_ux(:,:)
  real, allocatable                  :: tmp_uy(:,:,:)
  real, allocatable                  :: tmp_uz(:,:,:)

  ! grid, config, dimensions and counters
  ! _____________________________________
  type(ESMF_Grid)                    :: myGrid
  integer                            :: imloc, jmloc, lm
  integer                            :: ii, jj, ll

  ! Get my name and set-up traceback handle
  ! _______________________________________
  call ESMF_GridCompGet(GC, name=comp_name, grid=myGrid, __RC__)
  Iam = trim(comp_name)//'::'//"Run"

  ! Get MAPL object from GC
  ! _______________________
  call MAPL_GetObjectFromGC(GC, MAPLobj, __RC__)
```

```
140     ! Get local dimensions
        ! ---------------------
142     call MAPL_Get(STATE=MAPLobj, IM = imloc, JM = jmloc, LM = lm)

144     ! Allocate temporary arrays
        ! -------------------------
146     allocate(tmp_ux(imloc, jmloc))
        allocate(tmp_uy(imloc, jmloc, lm))
148     allocate(tmp_uz(imloc, jmloc, lm))

150     ! Do something with these temp arrays
        ! -----------------------------------
152     do ii=1,imloc
           do jj=1,jmloc
154           tmp_ux(ii,jj) = 6.66
              do ll=1,lm
156              tmp_uy(ii,jj,ll) = 7.77
                 tmp_uz(ii,jj,ll) = 8.88
158           end do
           end do
160     end do

162     ! Get pointers to export states and fill out
        ! Note that we are NOT filling out UY since
164     ! it is not an import state anywhere (this is
        ! to save memory)
166     ! ------------------------------------------
        call MAPL_GetPointer(Export, ptr_ux, 'UX', __RC__)
168     if(associated(ptr_ux)) ptr_ux = tmp_ux
        call MAPL_GetPointer(Export, ptr_uy, 'UY', __RC__)
170     if(associated(ptr_uz)) ptr_uy = tmp_uy
        call MAPL_GetPointer(Export, ptr_uz, 'UZ', __RC__)
172     if(associated(ptr_uz)) ptr_uz = tmp_uz

174     ! We can deallocate temporary arrays. They have been
        ! deep-copied into the variable referenced by 'AA'
176     ! ------------------------------------------------
        deallocate(tmp_ux, tmp_uy, tmp_uz)
178
        ! We should have access to the import states
180     ! ------------------------------------------
        call MAPL_GetPointer(Import, ptr_ox, 'OX_TEND', __RC__)
182
        ! Print out
184     ! ---------
        if (MAPL_am_I_root()) then
186        print *, ''
           print *, '-------------------------------------'
188        print *, 'ChildOne:'
           print *, 'Import spec OX_TEND: ', ptr_ox(1,1)
190        if (associated(ptr_ux)) print *, 'Export spec      UX: ', ptr_ux(1,1)
           if (associated(ptr_uy)) print *, 'Export spec      UY: ', ptr_uy(1,1,1)
192        if (associated(ptr_uz)) print *, 'Export spec      UZ: ', ptr_uz(1,1,1)
           print *, '-------------------------------------'
194     end if
```

```
      ! All done
196   ! ---------
      RETURN_(ESMF_SUCCESS)
198
    end subroutine ChildOneRun
200
    !-----------------------------------------------------------------------------!
202
  end module ChildOneGridCompMod
```

A.5.5 Gridded Component: Root

```
1 #include "MAPL_Generic.h"

3 module RootGridCompMod

5   use ESMF_Mod
    use MAPL_Mod
7
    use ChildOneGridCompMod, only: ChildOne_SetServices => SetServices
9   use ChildTwoGridCompMod, only: ChildTwo_SetServices => SetServices

11  implicit none
    private
13
    public SetServices
15
    ! (Global) integers representing children components
17  ! ------------------------------------------------
    integer                                 :: ChildOne
19  integer                                 :: ChildTwo


21
  contains
23
    !-----------------------------------------------------------------------------!
25
    subroutine SetServices(GC, RC)
27
      type(ESMF_GridComp), intent(INOUT) :: GC  ! gridded component
29    integer, optional,   intent(  OUT) :: RC  ! return code

31    ! ErrLog
      ! ------
33    character(len=ESMF_MAXSTR)          :: Iam
      integer                             :: status
35    character(len=ESMF_MAXSTR)          :: comp_name

37    ! Get my name and set-up traceback handle
      ! ---------------------------------------
39    call ESMF_GridCompGet(GC, name=comp_name, __RC__)
      Iam = trim(COMP_NAME)//'::'//"SetServices"
41
      ! Set the Initialize, Run and Finalize entry points
43    ! -------------------------------------------------
      call MAPL_GridCompSetEntryPoint(GC, ESMF_SETINIT, RootInitialize, __RC__)
```

```
45      call MAPL_GridCompSetEntryPoint(GC, ESMF_SETRUN,   RootRun,          __RC__)

47      ! Create children and invoke their setservices
        ! ------------------------------------------
49      ChildOne = MAPL_AddChild(GC, name='ChildOne', ss=ChildOne_SetServices, __RC__)
        ChildTwo = MAPL_AddChild(GC, name='ChildTwo', ss=ChildTwo_SetServices, __RC__)
51
        ! Register connections between children
53      ! ------------------------------------------------------------------
        call MAPL_AddConnectivity(          &
55           GC,                            &
             SHORT_NAME = (/ 'OX_TEND' /), &
57           SRC_ID    =  ChildTwo,         &
             DST_ID    =  ChildOne,         &
59           __RC__)

61      ! Register connections between children
        ! ------------------------------------------------------------------
63      call MAPL_AddConnectivity(          &
             GC,                            &
65           SHORT_NAME = (/ 'UX' /),       &
             SRC_ID    =  ChildOne,         &
67           DST_ID    =  ChildTwo,         &
             __RC__)
69
        ! Generic setservices is the last one to be called
71      ! ------------------------------------------------
        call MAPL_GenericSetServices(GC, __RC__)
73
      RETURN_(ESMF_SUCCESS)
75
    end subroutine SetServices
77
    !-----------------------------------------------------------------------------!
79
    subroutine RootInitialize(GC, Import, Export, Clock, rc)
81
      type(ESMF_GridComp), intent(inout) :: GC     ! Gridded component
83    type(ESMF_State),    intent(inout) :: IMPORT ! Import state
      type(ESMF_State),    intent(inout) :: EXPORT ! Export state
85    type(ESMF_Clock),    intent(inout) :: CLOCK  ! The clock
      integer, optional,   intent(  out) :: RC     ! Error code
87
      ! ErrLog and local variables
89    ! --------------------------
      character(len=ESMF_MAXSTR)         :: IAm
91    integer                            :: status
      character(len=ESMF_MAXSTR)         :: comp_name
93    type(MAPL_MetaComp), pointer       :: MAPLobj
      type(ESMF_Config)                  :: cf
95    integer                            :: Nx, Ny
      type(ESMF_VM)                      :: vm
97    integer                            :: myPET, nPETs

99
```

```
    ! Get the target components name and set-up traceback handle
101 ! -----------------------------------------------------------
    call ESMF_GridCompGet(GC, name=comp_name, config=cf, __RC__)
103 Iam = trim(COMP_NAME)//'::'//"Initialize"

105 ! Get my MAPL object
    ! --------------------
107 call MAPL_GetObjectFromGC(GC, MAPLobj, __RC__)

109 ! Get Nx, Ny from MAPL object
    ! ----------------------------
111 call MAPL_GetResource(MAPLobj, Nx, label="NX:", __RC__)
    call MAPL_GetResource(MAPLobj, Ny, label="NY:", __RC__)
113
    call ESMF_VMGetCurrent(vm, __RC__)
115 call ESMF_VMGet(vm, localPET=myPET, petCount=nPETs, __RC__)
    if (nPETs /= Nx*Ny) then
117    if (MAPL_am_I_root()) then
          write(*, '(a16, 1x, i4, 1x, a14, 1x, i4, 1x, a4)') &
119            'ERROR: expecting', Nx*Ny, 'PETs but found', nPETs, 'PETs'
       end if
121    ASSERT_(.FALSE.)
    end if
123
    ! Create grid, this has to be done in the parent gridcomp
125 ! All children inherit this grid which may be overwritten
    ! in a child gridcomp
127 !-----------------------------------------------------------
    call MAPL_GridCreate(GC, __RC__)
129
    ! Generic initialize (call initialize for each child)
131 ! ---------------------------------------------------
    call MAPL_GenericInitialize(GC, Import, Export, Clock, __RC__)
133
    ! All done
135 ! --------
    RETURN_(ESMF_SUCCESS)
137
end subroutine RootInitialize
139
!-----------------------------------------------------------------------------!
141
subroutine RootRun(GC, Import, Export, Clock, rc)
143
  type(ESMF_GridComp), intent(inout) :: GC     ! Gridded component
145 type(ESMF_State),    intent(inout) :: IMPORT ! Import state
  type(ESMF_State),    intent(inout) :: EXPORT ! Export state
147 type(ESMF_Clock),    intent(inout) :: CLOCK  ! The clock
  integer, optional,   intent(  out) :: RC     ! Error code
149
  ! ErrLog and local variables
151 ! --------------------------
  character(len=ESMF_MAXSTR)         :: IAm
153 integer                            :: status
  character(len=ESMF_MAXSTR)         :: comp_name
```

```
155     type(MAPL_MetaComp), pointer       :: MAPLobj
        type(ESMF_GridComp), pointer       :: GCS(:)
157     type(ESMF_State),    pointer       :: GIM(:), GEX(:)

159     ! Get the target components name and set-up traceback handle
        ! ------------------------------------------------------------
161     call ESMF_GridCompGet(GC, name=comp_name, __RC__)
        Iam = trim(COMP_NAME)//'::'//"Run"
163
        ! Get my MAPL object
165     ! ------------------
        call MAPL_GetObjectFromGC(GC, MAPLobj, __RC__)
167
        ! Get children and their IM/EX states from MAPL object
169     ! ----------------------------------------------------
        call MAPL_Get(MAPLobj, GCS=GCS, GIM=GIM, GEX=GEX, __RC__)
171
        ! Do we need to do something about clock??
173
        ! Run children
175     ! ------------
        call ESMF_GridCompRun(GCS(ChildOne), GIM(ChildOne), GEX(ChildOne), Clock, __RC__)
177     call ESMF_GridCompRun(GCS(ChildTwo), GIM(ChildTwo), GEX(ChildTwo), Clock, __RC__)

179     ! All done
        ! --------
181     RETURN_(ESMF_SUCCESS)

183   end subroutine RootRun

185   !-----------------------------------------------------------------------------!

187 end module RootGridCompMod
```