Package Variables: Requirements and Design

MPAS Development Team

October 24, 2013

Contents

Chapter 1

Summary

This document introduces package variables, and describes requirements and design specifications for the implementation and use of package variables.

Package variables can most easily be explained through the concept of optional physics packages. For example, one simulation might have physics package A on while the next might have physics package A off. During a simulation we might not want to have all variables allocated for this physics package when it's not being used.

As such, package variables are introduced. These are groupings of variables whos allocation depends on the choices of namelist options.

Chapter 2

Requirements

To support the increasing complexity and breadth of options in MPAS cores, while keeping memory usage to a minimum, the package variable capability in MPAS must meet the following requirements.

1. MPAS must be capable of enabling or disabling individual variables, constituents of variable arrays (super-arrays), and variable groups at run-time. "Enabling" a variable means that the variable should be allocated and fully usable within an MPAS core; "disabling" means that a variable should use a little memory as possible while still allowing an MPAS core to compile and run using a set of options that do not require the variable. Packages are the means by which MPAS variables will be enabled and disabled.

2. It must be possible to include arbitrary sets of variables in a package. A package may, therefore, include a mix of regular variables, constituent variables, and variable groups.

3. Variables are not required to belong to any package, and the behavior of variables that do not belong to any package should not change from current behavior in MPAS.

4. MPAS must support the ability to enable packages using arbitrary logic.

5. MPAS core and infrastructure code must be able to determine whether a variable is enabled or disabled.

6. MPAS I/O should only read/write variables and constituents that are enabled. If a variable is disabled, and therefore has no associated storage, reading the variable makes no sense; and it wouldn't appear to be useful to write garbage or default values for a variable that is never used during a particular execution of an MPAS core.

7. MPAS must define packages in the XML Registry files.

8. Packages must allow documentation describing the pacakge and it's intended use.

Chapter 3

Design and Implementation

3.1 Implementation: Package Variables

Date last modified: 10/24/2013
Contributors: (Doug Jacobsen)

Package variables are defined by modifying the persistence of a particular variable. Previously variables could have their persistence defined as persistent, or scratch, and now can additionally be defined as package.

The persistence option is added to the var_struct, and var (nested under a var_array) constructs within the Registry.xml file. When persistence is speified on a var_struct it does not cause the entire var_struct to be persistent, scratch, or package, it simply defines the default persistence for all variables defined within that particular var_struct. This default can be overridden by specifying the persistence on a per var, var_array, or constituent var basis.

Additionally, when persistence is set to package, another attribute "package_name" is required. This defines the name of the package the var, var_struct, var_array, or constituent var belongs to.

Packages also need to be defined. Witihin registry, at the same level as var_struct a new construct is created called packages. All packages must be defined at this level before being used throughout the var_struct groups. Below is an example of a package definition:

```
   ...
</nml_record>
<packages>
       <package name=''package_a'' description=''Description of package a''/>
       <package name=''package_b'' description=''Description of package b''/>
</packages>
<var_struct name=''mesh'' time_levs=''0''>
   ...
```

After the package is defined, any var_struct, var, var_array, or constituent var constructs can be attached to it as follows:

```
<var_struct name=''physicsA'' time_levs=''0'' persistence=''package'' package_name=''package_a''>
       <var name=''physVar1'' dimensions=''nCells Time'' .../>
</var_struct>
<var_struct name=''physicsB'' time_levs=''0''>
       <var_array name=''physVarArray'' dimensions=''nVertLevels nCells Time'' persistence=''package'' package_name=''packa
              <var ... />
              ...
       </var_array>
```

```
        <var array name=‘‘physVarArray2’’ dimensions=‘‘nVertLevels nCells Time’’>
               <var name=‘‘consVar1’’ persistence=‘‘package’’ package_name=‘‘package_b’’ ... />
               <var name=‘‘consVar2’’ ... />
               ...
        </var_array>
</var_struct>
```

Within the shared framework, a module named mpas_packages is created that contains logicals of the format "package_name_on". These logicals are set to ".false." by default, but when set to true at the beginning of a simulation, they enable the allocation and I/O of the package variables.

Cores are responsible for the proper initialization of package logicals. This is done through a routine called core_setup_packages, which should only have an error argument. This subroutine is written on a per-core basis, and can contain arbitrary logic to enable packages.

For example:

```
subroutine mpas_core_setup_package(ierr)
  use mpas_configure

  implicit none

  integer, intent(out) :: ierr

  if(config_physics_option == trim('A')) then
    physics_a_on = .true.
  else if(config_physics_option == trim('B') .and. config_num_halos .ge. 3) then
    physics_b_on = .true.
  end if
end subroutine mpas_core_setup_packages
```

The core should not try to allocate or deallocate the variables contained in the package. The core is responsible for ensuring that variables in a package that is not active are not used.

There are two methods of determining if a variable is part of an active package or not. The first is to check the "package_name_on" logical in the mpas_packages module. The second depends on what type of variable is being queried. For all variables that are not constituents of a var_array, the isActive attribute can be queried. If field % isActive is equal to .true. then the package controling that variables allocation is active. For constituent variables that don't conatin an isActive attribute the index_varname index is set to a value of -1.

Chapter 4

Testing

4.1 Testing and Validation: Package Variables

Date last modified: 10/03/2013
Contributors: (Doug Jacobsen)

Package variables will be added to a component.

A run with the package on and off will be performed, and then should both run to completion and produce bit-idential results to runs where the package variables are defined as persistent.