

**CCSM Coupler Requirements**  
Version 6, the “Next Generation” Coupler

*May 2001*

*CCSM Coupler 6 Development Team*

**Community Climate System Model**  
National Center for Atmospheric Research, Boulder, CO  
<http://www.cgd.ucar.edu/csm/models>

---

CVS tag \$Name: \$ Build date: June 21, 2001

# Contents

<b>1 Synopsis</b>	<b>2</b>
<b>2 Requirements</b>	<b>2</b>
2.1 Scientific Requirements . . . . .	2
2.1.1 Component Model Control . . . . .	2
2.1.2 Flexibility . . . . .	2
2.1.3 Extensibility . . . . .	2
2.1.4 Time Stepping Method . . . . .	3
2.1.5 Computations . . . . .	3
2.1.6 Grids . . . . .	3
2.1.7 Fault Detection . . . . .	3
2.2 Computational Requirements . . . . .	4
2.2.1 Target Platforms . . . . .	4
2.2.2 Language . . . . .	4
2.2.3 Parallel Implementation . . . . .	4
2.2.4 Parallel Reproducibility . . . . .	4
2.2.5 Sequencing . . . . .	4
2.2.6 Performance . . . . .	4
2.2.7 Single Vs. Multiple Executables . . . . .	5
2.2.8 Processor Allocation . . . . .	5
2.2.9 Communications . . . . .	5
2.2.10 Mapping . . . . .	5
2.2.11 I/O . . . . .	5
2.2.12 Fault Detection . . . . .	6
2.3 Imposed Functionality Requirements . . . . .	6
2.3.1 Single vs. Multiple Executable . . . . .	6
2.3.2 Fault Detection . . . . .	6
2.3.3 Handshaking . . . . .	6
<b>3 Review Status</b>	<b>7</b>
<b>4 Glossary</b>	<b>7</b>
<b>Bibliography</b>	<b>8</b>

# 1 Synopsis

The primary function of the CCSM coupler is to coordinate the execution of, and interactions between, the various component models which comprise a single configuration of the CCSM. Critical functionality in the coupler includes synchronization of the time integration of the coupled system, managing inter-model data communication, conservatively mapping data between the various model grids, and computing certain interfacial fluxes between components. The coupler must perform these functions whenever and wherever performance, software engineering, or scientific requirements dictate. The design and implementation of CCSM coupler version 6, the “next generation” coupler, will represent significant advances over previous coupler implementations, in particular, the CCSM and PCM couplers.

The primary motivations for creating a new coupler are (1) to create a single coupler that will replace the existing PCM and CCSM couplers, thus unifying the PCM and CCSM modeling efforts, and (2) to support the creation of a performance-portable CCSM, one which can execute efficiently on a range of platforms, in a variety of configurations. Thus, the coupler requirements should clearly address the goal of delivering performance portability across scalable parallel supercomputers, commodity clusters, shared memory multiprocessors, and vector supercomputers.

Continuing the naming sequence familiar to the CCSM community, this coupler is called "cpl6", or "the CCSM coupler, version six." Among developers, cpl6 is also known as the "next generation coupler" or "ngc."

## 2 Requirements

### 2.1 Scientific Requirements

#### 2.1.1 Component Model Control

The coupler steers the execution of the components of the CCSM, and coordinates and supports the transfer of data between the components. Each component model is unaware of the timestep or grid used by the other component models, and the component models communicate only through the coupler at a specified communication interval.

#### 2.1.2 Flexibility

The coupler’s interfaces to the component models must be well defined, such that existing component models can be swapped for new models, or benign substitutes, in a reasonably straightforward manner.

#### 2.1.3 Extensibility

The coupler must allow the user to modify (add/delete/change) the types and amounts of data which are being exchanged between the coupler and the com-

ponent models, or add a new model to the configuration, with minimal changes to the coupler code.

#### 2.1.4 Time Stepping Method

The coupler is required to support either process split or time split advancement of the component models. This requirement is closely associated with the requirement contained in Section 2.2.5.

#### 2.1.5 Computations

- **Fluxes** The coupler is required to compute and combine interfacial fluxes, as needed, among the various component models.
- **Mapping** The coupler must perform the necessary mapping operations required to transform data between different component model grids. Each individual component model is unaware about the grid structures of the other component models. When mapping fluxes, conservation must be insured (using the LANL SCRIP package or similar software).
- **Merging** When data is being prepared for a destination grid, the coupler must be able to merge (spatially average) data originating from multiple source grids. For example, a flux to the atmosphere, within an atmospheric grid cell, may be composed of a combination of the flux from the land, sea ice, and ocean models.
- **Time-Accumulation and Time-Averaging** The coupler must be able to do time accumulation and time averaging of data during the integration for subsequent distribution to other component models. This may be necessary for flux conservation when different models use different communication intervals.
- **Diagnostics** The coupler must be able to perform and report basic diagnostic calculations, such as spatial and temporal averages of the quantities exchanged between component models.

#### 2.1.6 Grids

The coupler is required to support general grid shapes (not just logically rectangular grids). The coupler must have the ability to work around masked areas.

#### 2.1.7 Fault Detection

The coupler must perform an initial coherence check to insure that all component models are set-up in the proper manner for the expected integration. The coupler must allow periodic coherence checks to assure proper synchronization. In situations where the component models fail, the coupler is required to have a mechanism to detect the failure and shutdown the entire system.

## **2.2 Computational Requirements**

### **2.2.1 Target Platforms**

The target platform for the coupler design will be clusters of RISC based multiprocessors. This architecture is a superset of traditional one processor per node MPPs (Cray-T3E) and large scalable shared memory systems (SGI Origin 2000). Examples of such clusters include the IBM SP and Compaq ES-40 clusters. Achieving high performance on both vector and RISC processors is complicated. Experience has shown that in many circumstances redundant pieces of code must be generated, which will necessarily slow the coupler development effort. Therefore, the goal of achieving efficient vector processing performance, although desirable, is not a requirement at this time.

### **2.2.2 Language**

All of the general purpose libraries written for the coupler are required to have a Fortran 90 API or a Fortran 90 module interface. All physics packages written explicitly for the coupler are required to be written in Fortran 90.

### **2.2.3 Parallel Implementation**

The coupler is required to give the user control in selecting pure message passing, pure shared memory multithreading, or hybrid parallel algorithms to take advantage of different platforms and the relative performance of parallel modes on those platforms.

### **2.2.4 Parallel Reproducibility**

Under certain conditions, bit-for-bit reproducibility is required under the different parallel implementations listed in Section 2.2.3. At a minimum the coupler must give bitwise reproducibility when a statically loaded executable (same model with identical linked libraries) is rerun on the same machine at the same site on the same number of processors. It is noted that a strict requirement for bitwise parallel reproducibility may have a negative impact on performance.

### **2.2.5 Sequencing**

The coupler functions will support both sequential and concurrent execution of the component models, in a fashion such that the user can configure the model/coupler with the sequencing method which makes the most sense for a particular problem/platform. This requirement is closely associated with the requirement contained in Section 2.1.4.

### **2.2.6 Performance**

At a minimum the coupler is required to meet or exceed the performance of the current (March 2000) Climate System Model, Version 1.2 and Parallel Climate

Model, Version 1.0 when configured (component model resolutions and communication intervals) similarly. That is, for concurrent execution, the coupler must at least outperform the most time intensive component model in terms of seconds per model day; for sequential execution, the coupler must not comprise more than 20% of the entire CCSM execution time. It is noted that the user can select a model/coupler configuration which easily can violate this requirement (such as configuring a shorter coupling interval requiring more mappings and communication).

### **2.2.7 Single Vs. Multiple Executables**

The coupler is required to accommodate executing in an environment where the coupler and the component models comprise a single executable, or where the coupler and each component model run as separate executables. It is worth noting that single vs. multiple executables is independent of the sequencing method selected (e.g., concurrent execution can be accomplished with a single executable).

### **2.2.8 Processor Allocation**

For flexibility, the coupler must be capable of running decomposed into an arbitrary numbers of processors. This is transparent to the component models.

### **2.2.9 Communications**

Only communications between the coupler and the component models are allowed. Component model to component model communications are disallowed. Functionally, n coupler nodes should be able to communicate in parallel with m model nodes, for an arbitrary (n,m). Parallel communications between coupler and component models should be explored, developed and optimized, if possible.

### **2.2.10 Mapping**

The regridded must be capable of performing arbitrary regriddings; i.e., any mapping from a source grid vector into a destination grid vector, representable by a transformation matrix, should be supported. The regridded must be capable of performing fast, efficient regriddings in parallel either through threading, or message passing, or both. The regridded must be able to perform a masked regridding operation.

### **2.2.11 I/O**

The coupler must be able to read/write restart datasets that insure CCSM bit-for-bit restartability, and write history datasets consisting of user selected sets and subsets of data being exchanged between the component models. If possible, the NCAR/Unidata I/O Library will be used.

### **2.2.12 Fault Detection**

Similar to the scientific functionality requirement in Section 2.1.7, the coupler is required to have a mechanism which will recognize in a rudimentary fashion that a fatal computational problem has arisen and perform the necessary operations to shutdown the entire model system.

## **2.3 Imposed Functionality Requirements**

In order to meet the desired requirements of the coupler, it is apparent that the component models will need to have a priori knowledge of, and must be able to fulfill, some coupler requirements. A list of possible requirements of the component models, imposed by the coupler requirements, is given in this section. Note that the reverse is also true - the models may impose a requirement of the coupler. Any requirements of this latter type are unknown at the present time, but this document recognizes and notes the possibility.

### **2.3.1 Single vs. Multiple Executable**

The component models are required to make accommodations for system configurations of either a single or multiple executable running environment. Furthermore, when the system is configured as concurrently running component models within a single executable, the component models must have the flexibility to execute within a designated processor space.

### **2.3.2 Fault Detection**

The coupler is required to detect and/or react to a system problem, and therefore must receive appropriate fault detection handshaking signals from the component models. Thus, the component models are required to provide such signals. The component models may also be required to detect a coupler problem and act appropriately.

### **2.3.3 Handshaking**

Other handshaking requirements between the coupler and the component models include:

- interface design
- on-command AUTO-restart writing capability
- possibly use the CCSM shared code (physical constants, orbital parameters, etc.)
- calendar manager

This list is certainly not all-inclusive.

### 3 Review Status

#### Requirements Review

**Review Date:** May 5, 2001

<b>Reviewers:</b>	Byron Boville	NCAR/CGD
	Peter Gent	NCAR/CGD
	Steve Hammond	NCAR/SCD
	Ian Foster	ANL
	Phil Jones	LANL
	John Weatherly	US Army Cold Regions Research and Engineering Lab

### 4 Glossary

**bundle** Several data fields, all of which share the same domain, grouped together in one data structure.

**combine** Add together two or more fields that are on the same domain; e.g., latent heat flux plus sensible heat flux on a T42 grid with the same land mask

**communication interval** The time interval at which a component model exchanges data with the coupler.

**concurrent execution** Running two or more coupled system components at the same time.

**domain** Coordinates arrays together with mask, cell area, and processor decomposition information.

**grid** Coordinate arrays without a mask, cell area, decomposition information; e.g., a T42 grid.

**hybrid parallel** A combination of messaging passing and threading.

**map** Regrid; move a data field from one domain to another.

**mask** True/false flags associated with a specific grid; e.g., a land mask.

**merge** Take two or more fields of the same type (e.g., land, ocean, and ice surface temperatures), which are all on the same domain (e.g., all on a T42 grid, global domain), and the same number of corresponding cell fractions (fraction of land, ocean, and ice, which together sum to unity), and create one field ( e.g., (merged surface temperature) = (ice frac)\*(ice T) + (ocn frac)\*(ocn T) + (lnd frac)\*(lnd T) ).

**process split** Time stepping strategy whereby when component models integrate thru a time interval, say  $[t_1, t_2]$ , the only data available to the models corresponds to a previous time interval,  $[t_0, t_1]$ ; generally associated with concurrent component model execution.

**reentrant** A characteristic of a function which can be called concurrently by multiple threads without mishap.

**regrid** Map; move a data field from one domain to another.

**sequential execution** Running only one coupled system component at a time.

**time split** Time stepping strategy whereby when component models integrate thru a time interval, say  $[t_1, t_2]$ , the data made available to some models may correspond to the time interval  $[t_1, t_2]$ ; generally associated with sequential component model execution.

**timestep** A model's internal integration increment in time.

## References