Rocman Developers Guide

Xiangmin Jiao

December 12th, 2003

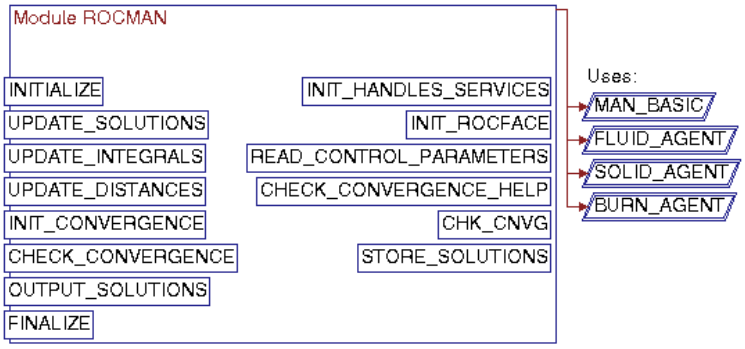

Figure 1: Pictorial declaration of F90 Module ROCMAN.

1 Introduction

This document is intended to serve as a guide for developers who desire to modify or extend *Rocman*. Developers of physics codes who would to learn how to interact with *Rocman* should look into the design document of *Rocman* – Interface Design of GEN2.5, which describe the algorithms, features, and API to physics codes of *Rocman*.

2 Organization of *Rocman*

2.1 *Rocman*'s Core Modules

At the time of this writing, *Rocman* is composed of the following F90 modules: ROCMAN (providing the interface routines called by the driver of GENx), three agents for different physics (FLUID_AGENT, SOLID_AGENT, and BURN_AGENT), MAN_BASIC (defining some derived data types and support utility routines for the aforementioned higher-level modules), and MAN_PARAM (defining some constant parameters; used by MAN_BASIC). These modules are defined in their corresponding f90 files with lowercase letters (i.e., rocman.f90, fluid_agent.f90, solid_agent.f90, burn_agent.f90, man_basic.f90, and man_param.f90).

Rocman also has two header files, configf90.h, and rocmanf90.h. The former defines a character string identifying the version of GENx, and an integer identifying the build number, which is to be incremented automatically by *Rocbuild*. The latter header file defines some constant parameters that are included by physics modules (for example, MAN_DBL, which defines the kind of double precision numbers used by GENx, and some indices of the array for checking integrals).

2.1.1 F90 Module ROCMAN

Figure 1 shows the declaration of module ROCMAN. The functions in the left column are the public functions, and those in the right column are private functions. ROCMAN uses module MAN_BASIC and the agents in its

implementation.

2.1.2 Agents for Physics Modules

Figure 2 shows the declaration of the three agents. Each agent constructs some windows inside *Roccom* by inheriting data from the windows of its corresponding physics solver, and the window names are given by the variables contained in the rounded boxes in the figure. The names contained in the rectangles are public subroutines, of which the INITIALIZE_* and UPDATE_INBUFF_* are callback routines provided to physics modules, and the others are to be called by module ROCMAN.

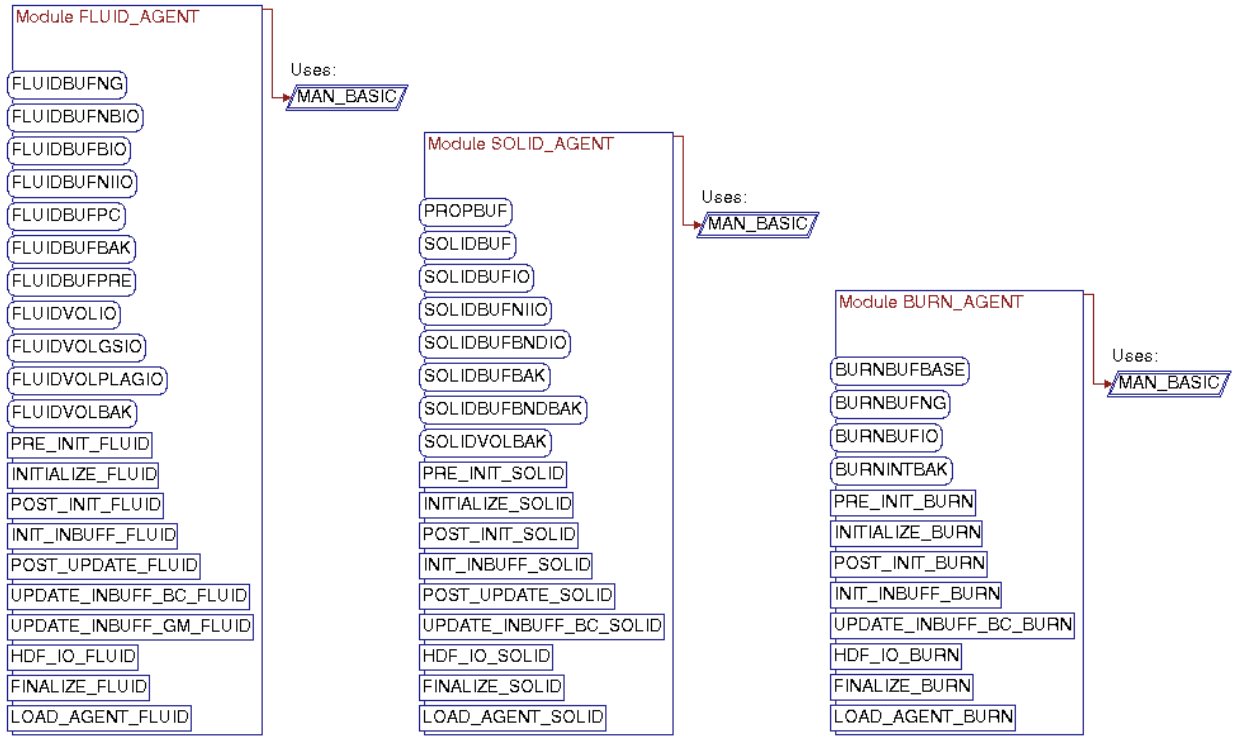

Figure 2: Pictorial declarations of agents for physics solvers.

2.1.3 Utility Routines, Derived Data Types, and Parameters

Module MAN_BASIC defines some basic utility routines to facilitate the implementation of the agents and the ROCMAN module. The left graph of Figure 3 shows the declaration of the module. The names in the rectangles in the left column are public subroutines, and the one in the right column is a private subroutine used by INTERPOLATE. The names in the hexahedra are derived data types, among which MAN_GLOBAL is the context object of ROCMAN. MAN_GLOBAL contains a pointer to MAN_GLOBAL_FLUID, MAN_GLOBAL_SOLID, and MAN_GLOBAL_BURN, which are the context objects of FLUID_AGENT, SOLID_AGENT, and BURN_AGENT, respectively. All subroutines in module ROCMAN and most subroutines in the agents and MAN_BASIC take a pointer to MAN_GLOBAL as their first arguments.

Module MAN_PARAM defines some constant parameters. In particular, it defines the input/output mode for I/O routines, the maximum lengths of strings, the modes of load transfer, the modes of interpolation in time, the file units used by *Rocman*, and some file names. The last integer MAN_GLOBAL_COOKIE is an arbitrary integer used for runtime verification of the MAN_GLOBAL object.

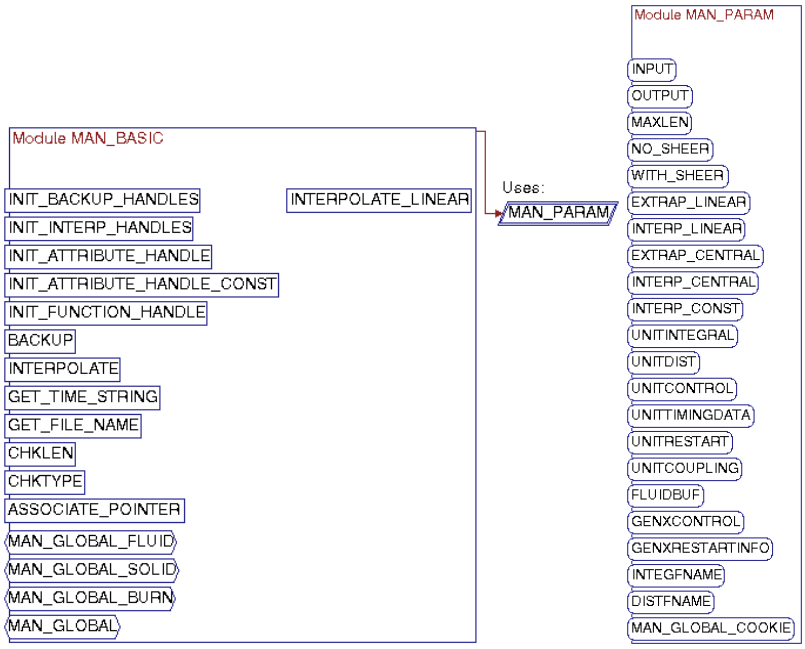

Figure 3: Pictorial declarations of F90 Modules MAN_BASIC and MAN_PARAM.

2.2 Loading and Unloading of Modules

Rocman's modules are loaded into *Roccom* by the subroutine ROCMAN_LOAD_MODULE, defined in file rocman_load_module.F90, which uses some C preprocessor directives to accommodate both static and dynamic linking of modules by conditional compilation. Figure 4 shows the top three levels of the call graph of ROCMAN_LOAD_MODULE. This procedure creates a window in *Roccom*, allocates an MAN_GLOBAL object and registers it with *Roccom* by calling COM_set_pointer, which takes ASSOCIATE_POINTER as a callback routine. Then, ROCMAN_LOAD_MODULE calls COM_INIT_MEMBER_FUNCTION to register the subroutines in module ROCMAN that are children of ROCMAN_LOAD_MODULE in this call graph, and those in module X_AGENT that are children of LOAD_AGENT_X. These registered subroutines will call their child procedures in the graph when they are invoked later through *Roccom* during the time marching schemes of GENx. Note that ROCMAN_LOAD_MODULE loads not only ROCMAN and the agents into *Roccom*, but also the physics and service modules specified by its input argument, by calling COM_LOAD_MODULE. The file rocman_load_module.F90 also a subroutine ROCMAN_UNLOAD_MODULE, which obtains a reference to the MAN_GLOBAL object through COM_get_pointer, deallocates the object, and then unloads the modules by calling COM_DELETE_WINDOW and COM_UNLOAD_MODULE.

2.3 Driver Routine of GENx

In Rocman's source directory, there are also two additional files: genx_driver.f90 and genx_coupling.f90. The former file contains the main entry subroutine of GENx, GENX_DRIVER, which is implemented by calling the registered functions of the ROCMAN module, together with some support utilities implemented in genx_coupling.f90. Figure 5 shows the top three levels of the call graph of GENX_DRIVER, and the declaration of its helper module GENX_COUPLING. In particlar, the initialization routine of GENX_COUPLING invokes ROCMAN_LOAD_MODULE to load the modules, and the finalization routines invokes ROCMAN_UNLOAD_MODULE to unload the modules. The calls to COM_CALL_FUNCTION in GENX_DRIVER are indirect calls to ROCMAN's interface functions through *Roccom*.

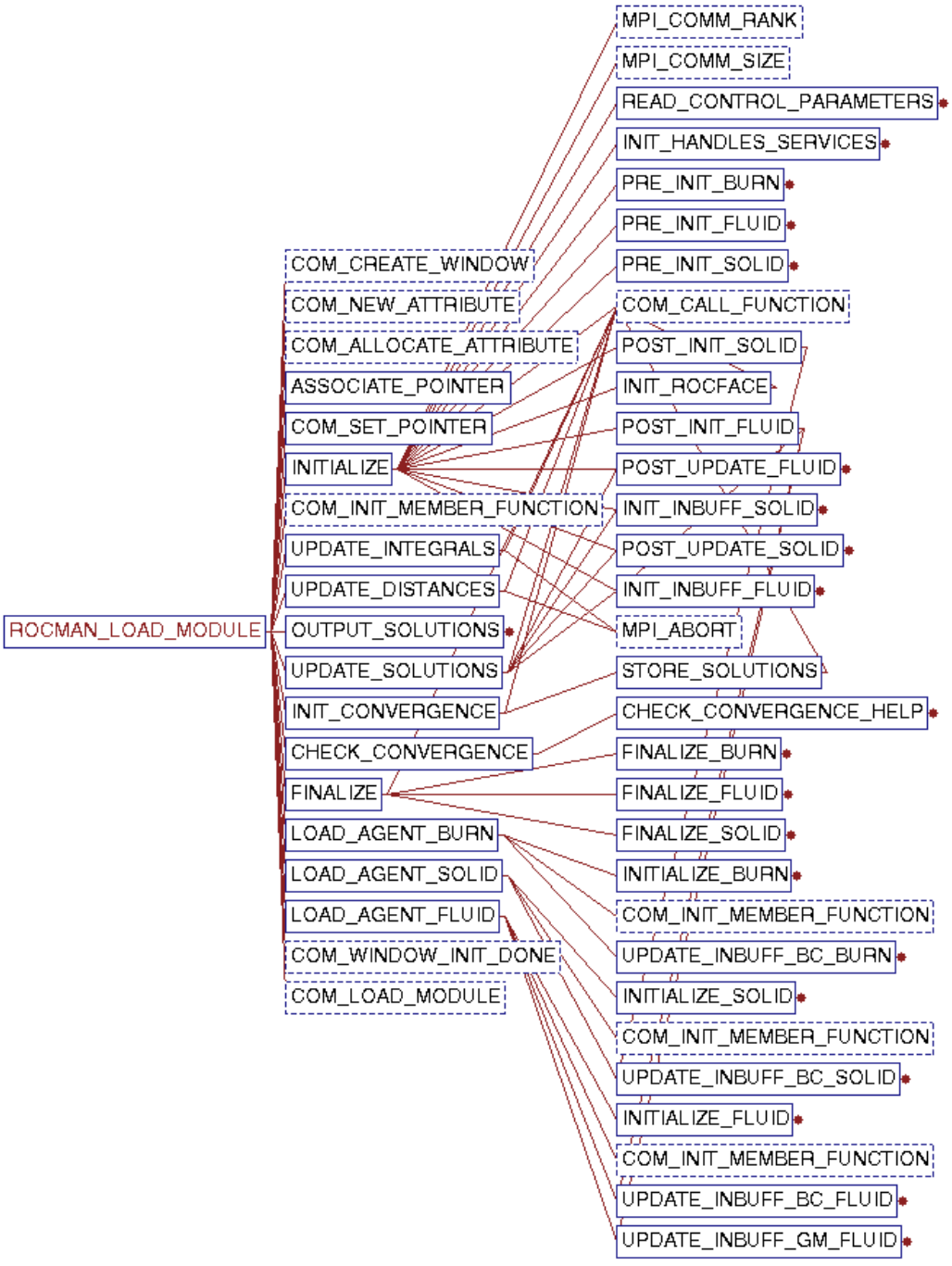

Figure 4: Top three levels of call graph of ROCMAN_LOAD_MODULE.

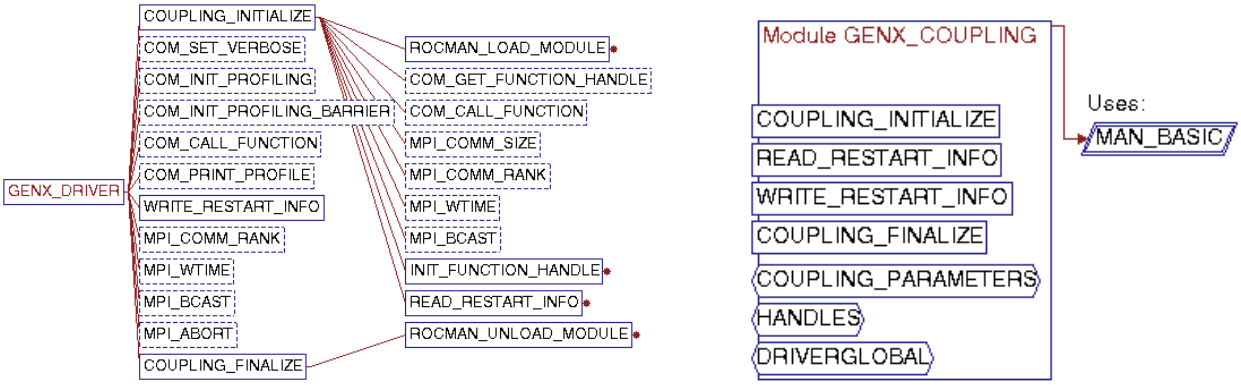

Figure 5: Top three levels of call graph of GENX_DRIVER and declaration of its helper module GENX_COUPLING.

3 Data Structure

3.1 Data Types

As described in Section 2.1.3, *Rocman* has four derived data types: MAN_GLOBAL, MAN_GLOBAL_FLUID, MAN_GLOBAL_SOLID, and MAN_GLOBAL_BURN. These data types encapsulate the states of ROCMAN, FLUID_AGENT, SOLID_AGENT, and BURN_AGENT, respectively. MAN_GLOBAL contains some variables for controlling the behavior of data transfer (the order of interpolation in time, the mode of load transfer, and predictor-corrector iterations), the names of the main window of *Rocman* and of the physics modules, some state variables (such as MPI communicators, the time stamp and time steps, the I/O mode for integral and distance files, and an integer ID of the data type for runtime checking), followed by the function handles of service modules and a pointer to each of FLUID_AGENT, SOLID_AGENT, and BURN_AGENT. The agent-specific objects share the following structure: first a few control parameters, followed by names of the interface window and output directory, some internal states and temporary variables, and finally the attribute and function handles.

3.2 Intermediate Windows

Besides these derived data types, each agent also needs to construct some windows in *Roccom* by inheriting the windows of its corresponding physics code for data manipulation or I/O. There are four types of windows in *Rocman*:

1. Interface windows for data manipulation such as data transfer or surface propagation. Such windows must contain the interacting panes but must not contain ghost nodes/cells. These windows include:

   (a) Fluid: FLUIDBUFNG (for interacting panes without ghosts).

   (b) Solid: PROPBUF(for surface propagation) and SOLIDBUF (for data transfer).

   (c) Burn: BURNBUFNG (for burning panes without ghosts).

2. Interface or volume windows for I/O. An agent may need to split a window of a physics module based on the values of the pane attribute bcflag, so that the panes with different bcflags can be written into different files. *Rocman* outputs the volume windows of solid and burn solvers as is without splitting, but splits other windows into the following windows whose names all end with "IO":

   (a) Fluid surface: FLUIDBUFNBIO (for nonburning), FLUIDBUFBIO (for burning), and FLUIDBUFNIIO (for noninteracting).

   (b) Fluid volume: FLUIDVOLIO (for main part), FLUIDVOLGSIO (for grid speed), and FLUIDVOLPLAGIO (for particles).

   (c) Solid surface: SOLIDBUFIO (for interacting), SOLIDBUFNIIO (for noninteracting), and SOLIDBUFBNDIO (for boundary flags).

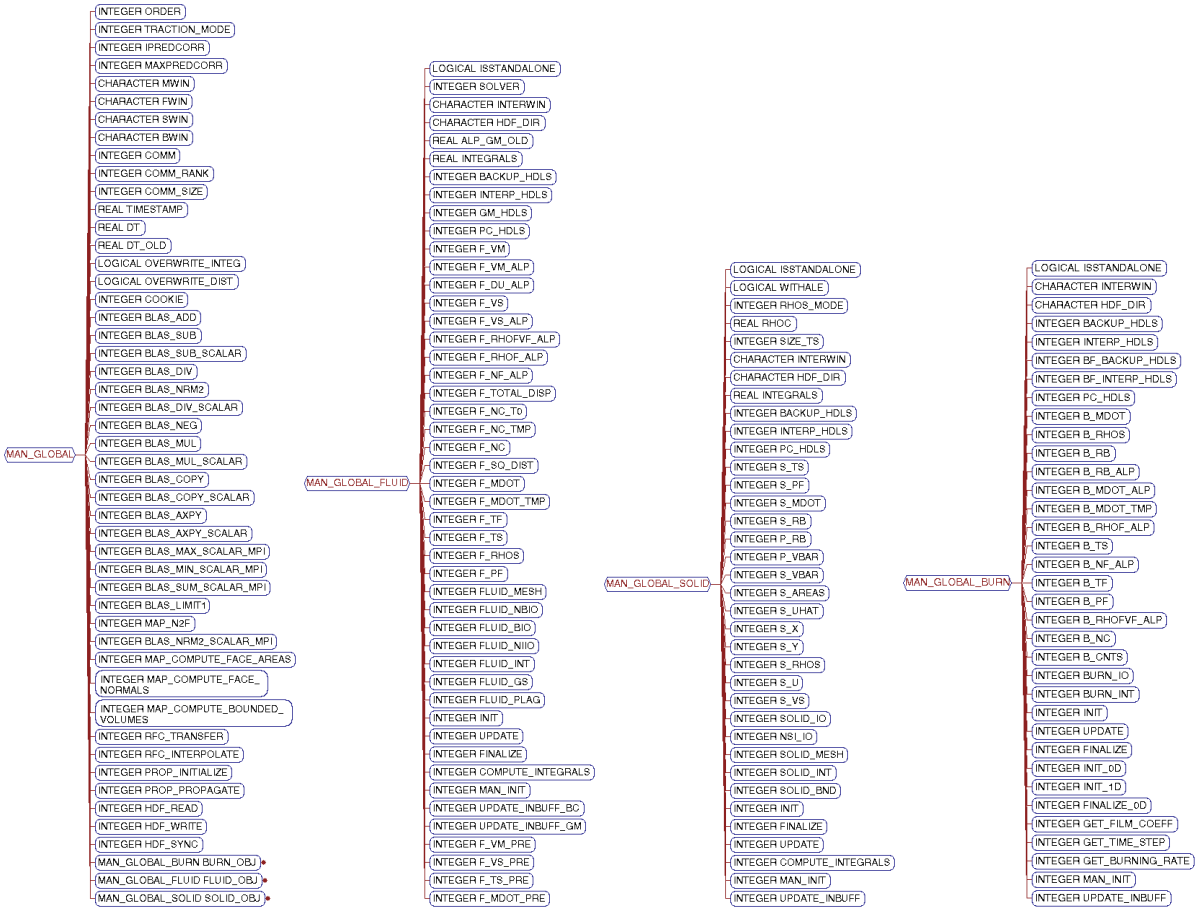

Figure 6: Declarations of derived data types of *Rocman*.

(d) Burn surface: BURNBUFIO.

3. Interface or volume windows for predictor-corrector iterations, including:

   (a) Fluid: FLUIDBUFPC (surface data to be backed up), FLUIDBUFBAK (backup of surface data), FLUIDBUFPRE (previous solution for convergence check), and FLUIDVOLBAK (backup of volume data).

   (b) Solid: SOLIDBUFBAK (backup of surface data), SOLIDBUFBNDBAK (backup of boundary flags), and SOLIDBUFBNDBAK (backup of volume data).

   (c) Burn: BURNINTBAK (backup of internal states of burn module).

4. Special-purpose intermediate windows. There is only one such window: BURNBUFBASE, which inherits data from a surface mesh of with fluid or solid with ghost nodes/cells, and is then inherited by BURNBUFIO with ghost nodes/cells for IO and by BURNBUFNG without ghost nodes for data transfer.

4 Interface

This section describes the interfaces of the functions in module ROCMAN that are registered with *Roccom*. These functions are called indirectly by GENX_DRIVER through COM_CALL_FUNCTION. The registered functions of the agents are described in the Design Document.

```
subroutine INITIALIZE (g, initialTime, maxPredCorr, comm, dt)
   type (MAN_Global), POINTER :: g
   real (kind=DBL), INTENT(IN) :: initialTime
   integer, INTENT(IN) :: maxPredCorr
   integer, INTENT(IN) :: comm
   real (kind=DBL), INTENT(IN) :: dt

subroutine UPDATE_SOLUTIONS (g, timestamp, dt, iPredCorr)
   type (MAN_Global), POINTER :: g
   real (kind=DBL), INTENT(IN) :: timestamp
   real (kind=DBL), INTENT(IN) :: dt
   integer, INTENT(IN) :: iPredCorr

subroutine UPDATE_INTEGRALS (g, currentTime)
   type (MAN_Global), POINTER :: g
   real (kind=DBL), INTENT(IN) :: currentTime

subroutine UPDATE_DISTANCES (g, currentTime)
   type (MAN_Global), POINTER :: g
   real (kind=DBL), INTENT(IN) :: currentTime

subroutine INIT_CONVERGENCE (g, iPredCorr)
   type (MAN_Global), POINTER :: g
   integer, INTENT(IN) :: iPredCorr

subroutine CHECK_CONVERGENCE (g, iPredCorr, TolerMass, &
           TolerTract, TolerVelo, InterfaceConverged)
   type (MAN_Global), POINTER :: g
   integer, INTENT(IN) :: iPredCorr
   real (kind=DBL), INTENT(IN) :: TolerMass
   real (kind=DBL), INTENT(IN) :: TolerTract
   real (kind=DBL), INTENT(IN) :: TolerVelo
   logical, INTENT(OUT) :: InterfaceConverged

subroutine OUTPUT_SOLUTIONS (g, currentTime)
   type (MAN_Global), POINTER :: g
   real (kind=DBL), INTENT(IN) :: currentTime

subroutine FINALIZE (g)
   type (MAN_Global), POINTER :: g
```

5 Future Plans

In the near future, *Rocman* needs to be extended to allow the combustion model to sit on either fluid or solid meshes. Similarly, we will need to extend *Rocman* to allow surface propagation to be on either fluid or solid meshes, so that we can apply surface propagation in the FluidAlone mode.

Acknowledgments

This document was typeset using LyX Version 1.3.2 (`http://www.lyx.org`) on Suse Linux 9.0. The declaration and call graphs were produced using STI Understanding for Fortran Version 1.4 of Scientific Toolworks, Inc. (`http://www.scitools.com/`).