ICFERST

22_06

Generated by Doxygen 1.8.17

Chapter 1

Modules Index

1.1 Modules List

Here is a list of all documented modules with brief descriptions:

Chapter 2

Data Type Index

2.1 Data Types List

Here are the data types with brief descriptions:

Chapter 3

Module Documentation

3.1 multi_machine_learning Module Reference

Module to load and call a XGBoost model.

Functions/Subroutines

- subroutine xgboost_load_model ()

  Load the XGBoost model as -> xgb_model (private module variable)
- subroutine xgboost_predict (raw_input, out_result)

  Predict using the loaded XGBoost model xgboost_load_model() needs to be run first.
- subroutine xgboost_free_model ()

  Free the loaded XGBoost model from memory.
- subroutine test_xgboost ()

  Teste the XGBoost coupling.

3.1.1 Detailed Description

Module to load and call a XGBoost model.

Author

Vinicius L S Silva

3.1.2 Function/Subroutine Documentation

3.1.2.1 test_xgboost()

```
subroutine multi_machine_learning::test_xgboost
```

Teste the XGBoost coupling.

Author

Vinicius L S Silva

3.1.2.2 xgboost_free_model()

```
subroutine multi_machine_learning::xgboost_free_model
```

Free the loaded XGBoost model from memory.

Author

Vinicius L S Silva

3.1.2.3 xgboost_load_model()

```
subroutine multi_machine_learning::xgboost_load_model
```

Load the XGBoost model as -> xgb_model (private module variable)

Author

Vinicius L S Silva

3.1.2.4 xgboost_predict()

```
subroutine multi_machine_learning::xgboost_predict (
            real(c_float), dimension(17), intent(in) raw_input,
            real(c_float), dimension(:), intent(inout), pointer out_result )
```

Predict using the loaded XGBoost model xgboost_load_model() needs to be run first.

Author

Vinicius L S Silva

3.2 multiphase_time_loop Module Reference

Time-loop module of IC-FERST. This module contains the time-loop and the non-linear loop. The time-loop consists many steps: 1) Data initialisation including shape functions, memory allocation, sparsity, porous media properties, etc. 2) Initialisation of the actual time loop. 3) Non-linear loop. IC-FERST uses a modified Anderson-acceleration non-linear solver, which is based in a Picard iterative non-linear solver. In this way, the different equations are solved independently and coupled through the Anderson non-linear solver. First the momentum and continuity equations are assembled and solved for, next the different transport equations are solved, for example: saturation, temperature, concentration, etc. ActiveTracers and Species are solved within the non-linear solver while Passive↩ Tracers are solved outside the non-linear solver. 4) Once the non-linear solver has converged, we proceed to jump to the next time-level but first we check if we need to adapt the mesh and/or generate a vtu file. etc.

Functions/Subroutines

- subroutine, public multifluids_solvetimeloop (state, dt, nonlinear_iterations, dump_no)

  This is the main subroutine from which everything is called. It performs the time-loop and therefore calls all the necessary blocks to solve for the system of equations, adapt the mesh etc.

3.2.1 Detailed Description

Time-loop module of IC-FERST. This module contains the time-loop and the non-linear loop. The time-loop consists many steps: 1) Data initialisation including shape functions, memory allocation, sparsity, porous media properties, etc. 2) Initialisation of the actual time loop. 3) Non-linear loop. IC-FERST uses a modified Anderson-acceleration non-linear solver, which is based in a Picard iterative non-linear solver. In this way, the different equations are solved independently and coupled through the Anderson non-linear solver. First the momentum and continuity equations are assembled and solved for, next the different transport equations are solved, for example: saturation, temperature, concentration, etc. ActiveTracers and Species are solved within the non-linear solver while Passive↩ Tracers are solved outside the non-linear solver. 4) Once the non-linear solver has converged, we proceed to jump to the next time-level but first we check if we need to adapt the mesh and/or generate a vtu file. etc.

3.2.2 Function/Subroutine Documentation

3.2.2.1 multifluids_solvetimeloop()

```
subroutine, public multiphase_time_loop::multifluids_solvetimeloop (
             type(state_type), dimension(:), intent(inout), pointer state,
             real, intent(inout) dt,
             integer, intent(inout) nonlinear_iterations,
             integer, intent(inout) dump_no )
```

This is the main subroutine from which everything is called. It performs the time-loop and therefore calls all the necessary blocks to solve for the system of equations, adapt the mesh etc.

the number of items of the coupling term coefficients stored in the system

3.3 phreeqcrm Module Reference

Fortran Documentation for the geochemical reaction module PhreeqcRM.

Functions/Subroutines

- integer function rm_abort (id, irm_result, err_str)

  Abort the program. irm_result will be interpreted as an IRM_RESULT value and decoded; err_str will be printed; and the reaction module will be destroyed. If using MPI, an MPI_Abort message will be sent before the reaction module is destroyed. If the id is an invalid instance, RM_Abort will return a value of IRM_BADINSTANCE, otherwise the program will exit with a return code of 4.
- integer function rm_closefiles (id)

  Close the output and log files.
- integer function rm_concentrations2utility (id, c, n, tc, p_atm)

  N sets of component concentrations are converted to SOLUTIONs numbered 1-n in the Utility IPhreeqc. The solutions can be reacted and manipulated with the methods of IPhreeqc. If solution concentration units (RM_SetUnitsSolution) are per liter, one liter of solution is created in the Utility instance; if solution concentration units are mass fraction, one kilogram of solution is created in the Utility instance. The motivation for this method is the mixing of solutions in wells, where it may be necessary to calculate solution properties (pH for example) or react the mixture to form scale minerals. The code fragments below make a mixture of concentrations and then calculate the pH of the mixture.
- integer function rm_create (nxyz, nthreads)

  Creates a reaction module. If the code is compiled with the preprocessor directive USE_OPENMP, the reaction module is multithreaded. If the code is compiled with the preprocessor directive USE_MPI, the reaction module will use MPI and multiple processes. If neither preprocessor directive is used, the reaction module will be serial (unparallelized).
- integer function rm_createmapping (id, grid2chem)

  Provides a mapping from grid cells in the user's model to reaction cells in PhreeqcRM. The mapping is used to eliminate inactive cells and to use symmetry to decrease the number of cells for which chemistry must be run. The array grid2chem of size nxyz (the number of grid cells, RM_GetGridCellCount) must contain the set of all integers 0 <= i < count_chemistry, where count_chemistry is a number less than or equal to nxyz. Inactive cells are assigned a negative integer. The mapping may be many-to-one to account for symmetry. Default is a one-to-one mapping–all user grid cells are reaction cells (equivalent to grid2chem values of 0,1,2,3,...,nxyz-1).
- integer function rm_decodeerror (id, e)

  If e is negative, this method prints an error message corresponding to IRM_RESULT e. If e is non-negative, no action is taken.
- integer function rm_destroy (id)

  Destroys a reaction module.
- integer function rm_dumpmodule (id, dump_on, append)

  Writes the contents of all workers to file in _RAW formats, including SOLUTIONs and all reactants.
- integer function rm_errormessage (id, errstr)

  Send an error message to the screen, the output file, and the log file.
- integer function rm_findcomponents (id)

  Returns the number of items in the list of all elements in the InitialPhreeqc instance. Elements are those that have been defined in a solution or any other reactant (EQUILIBRIUM_PHASE, KINETICS, and others). The method can be called multiple times and the list that is created is cummulative. The list is the set of components that needs to be transported. By default the list includes water, excess H and excess O (the H and O not contained in water); alternatively, the list may be set to contain total H and total O (RM_SetComponentH2O), which requires transport results to be accurate to eight or nine significant digits. If multicomponent diffusion (MCD) is to be modeled, there is a capability to retrieve aqueous species concentrations (RM_GetSpeciesConcentrations) and to set new solution concentrations after MCD by using individual species concentrations (RM_SpeciesConcentrations2Module). To use these methods the save-species property needs to be turned on (RM_SetSpeciesSaveOn). If the save-species property is on, RM_FindComponents will generate a list of aqueous species (RM_GetSpeciesCount, RM_Get↩ SpeciesName), their diffusion coefficients at 25 C (RM_GetSpeciesD25), their charge (RM_GetSpeciesZ).
- integer function rm_getbackwardmapping (id, n, list, size)

  Fills an array with the cell numbers in the user's numbering sytstem that map to a cell in the PhreeqcRM numbering system. The mapping is defined by RM_CreateMapping.
- integer function rm_getchemistrycellcount (id)

  Returns the number of chemistry cells in the reaction module. The number of chemistry cells is defined by the set of non-negative integers in the mapping from user grid cells (RM_CreateMapping). The number of chemistry cells is less than or equal to the number of cells in the user's model.
- integer function rm_getcomponent (id, num, comp_name)

Retrieves an item from the reaction-module component list that was generated by calls to RM_FindComponents.

- integer function rm_getcomponentcount (id)

  Returns the number of components in the reaction-module component list. The component list is generated by calls to RM_FindComponents. The return value from the last call to RM_FindComponents is equal to the return value from RM_GetComponentCount.

- integer function rm_getconcentrations (id, c)

  Transfer solution concentrations from each reaction cell to the concentration array given in the argument list (c). Units of concentration for c are defined by RM_SetUnitsSolution. For concentration units of per liter, the solution volume is used to calculate the concentrations for c. For mass fraction concentration units, the solution mass is used to calculate concentrations for c. Two options are available for the volume and mass of solution that are used in converting to transport concentrations: (1) the volume and mass of solution are calculated by PHREEQC, or (2) the volume of solution is the product of saturation (RM_SetSaturation), porosity (RM_SetPorosity), and representative volume (R↩ M_SetRepresentativeVolume), and the mass of solution is volume times density as defined by RM_SetDensity. R↩ M_UseSolutionDensityVolume determines which option is used. For option 1, the databases that have partial molar volume definitions needed to accurately calculate solution volume are phreeqc.dat, Amm.dat, and pitzer.dat.

- integer function rm_getdensity (id, density)

  Transfer solution densities from the reaction cells to the array given in the argument list (density). Densities are those calculated by the reaction module. Only the following databases distributed with PhreeqcRM have molar volume information needed to accurately calculate density: phreeqc.dat, Amm.dat, and pitzer.dat.

- integer function rm_getendcell (id, ec)

  Returns an array with the ending cell numbers from the range of cell numbers assigned to each worker.

- integer function rm_getequilibriumphasescount (id)

  Returns the number of equilibrium phases in the initial-phreeqc module. RM_FindComponents must be called before RM_GetEquilibriumPhasesCount. This method may be useful when generating selected output definitions related to equilibrium phases.

- integer function rm_getequilibriumphasesname (id, num, name)

  Retrieves an item from the equilibrium phase list. The list includes all phases included in any EQUILIBRIUM_PHASES definitions in the initial-phreeqc module. RM_FindComponents must be called before RM_GetEquilibriumPhases↩ Name. This method may be useful when generating selected output definitions related to equilibrium phases.

- integer function rm_geterrorstring (id, errstr)

  Returns a string containing error messages related to the last call to a PhreeqcRM method to the character argument (errstr).

- integer function rm_geterrorstringlength (id)

  Returns the length of the string that contains error messages related to the last call to a PhreeqcRM method.

- integer function rm_getexchangename (id, num, name)

  Retrieves an item from the exchange name list. RM_FindComponents must be called before RM_GetExchange↩ Name. The exchange names vector is the same length as the exchange species names vector and provides the corresponding exchange site (for example, X corresponing to NaX). This method may be useful when generating selected output definitions related to exchangers.

- integer function rm_getexchangespeciescount (id)

  Returns the number of exchange species in the initial-phreeqc module. RM_FindComponents must be called before RM_GetExchangeSpeciesCount. This method may be useful when generating selected output definitions related to exchangers.

- integer function rm_getexchangespeciesname (id, num, name)

  Retrieves an item from the exchange species list. The list of exchange species (such as "NaX") is derived from the list of components (RM_FindComponents) and the list of all exchange names (such as "X") that are included in EX↩ CHANGE definitions in the initial-phreeqc module. RM_FindComponents must be called before RM_GetExchange↩ SpeciesName. This method may be useful when generating selected output definitions related to exchangers.

- integer function rm_getfileprefix (id, prefix)

  Returns the reaction-module file prefix to the character argument (prefix).

- integer function rm_getgascomponentscount (id)

  Returns the number of gas phase components in the initial-phreeqc module. RM_FindComponents must be called before RM_GetGasComponentsCount. This method may be useful when generating selected output definitions related to gas phases.

- integer function rm_getgascomponentsname (id, num, name)

Retrieves an item from the gas components list. The list includes all gas components included in any GAS_PHASE definitions in the initial-phreeqc module. RM_FindComponents must be called before RM_GetGasComponentsName. This method may be useful when generating selected output definitions related to gas phases.

- integer function rm_getgascompmoles (id, gas_moles)

  Transfer moles of gas components from each reaction cell to the array given in the argument list (gas_moles).

- integer function rm_getgascomppressures (id, gas_p)

  Transfer pressures of gas components from each reaction cell to the array given in the argument list (gas_p).

- integer function rm_getgascompphi (id, gas_phi)

  Transfer fugacity coefficients (phi) of gas components from each reaction cell to the array given in the argument list (gas_phi). Fugacity of a gas component is equal to the pressure of the component times the fugacity coefficient.

- integer function rm_getgasphasevolume (id, gas_volume)

  Transfer volume of gas from each reaction cell to the vector given in the argument list (gas_volume).

- integer function rm_getgfw (id, gfw)

  Returns the gram formula weights (g/mol) for the components in the reaction-module component list.

- integer function rm_getgridcellcount (id)

  Returns the number of grid cells in the user's model, which is defined in the call to RM_Create. The mapping from grid cells to reaction cells is defined by RM_CreateMapping. The number of reaction cells may be less than the number of grid cells if there are inactive regions or symmetry in the model definition.

- integer function rm_getiphreeqcid (id, i)

  Returns an IPhreeqc id for the ith IPhreeqc instance in the reaction module. For the threaded version, there are nthreads + 2 IPhreeqc instances, where nthreads is defined in the constructor (RM_Create). The number of threads can be determined by RM_GetThreadCount. The first nthreads (0 based) instances will be the workers, the next (nthreads) is the InitialPhreeqc instance, and the next (nthreads + 1) is the Utility instance. Getting the IPhreeqc pointer for one of these instances allows the user to use any of the IPhreeqc methods on that instance. For MPI, each process has exactly three IPhreeqc instances, one worker (number 0), one InitialPhreeqc instance (number 1), and one Utility instance (number 2).

- integer function rm_getkineticreactionscount (id)

  Returns the number of kinetic reactions in the initial-phreeqc module. RM_FindComponents must be called before RM_GetKineticReactionsCount. This method may be useful when generating selected output definitions related to kinetic reactions.

- integer function rm_getkineticreactionsname (id, num, name)

  Retrieves an item from the kinetic reactions list. The list includes all kinetic reactions included in any KINETICS definitions in the initial-phreeqc module. RM_FindComponents must be called before RM_GetKineticReactions↩ Name. This method may be useful when generating selected output definitions related to kinetic reactions.

- integer function rm_getmpimyself (id)

  Returns the MPI task number. For the OPENMP version, the task number is always zero and the result of RM_↩ GetMpiTasks is one. For the MPI version, the root task number is zero, and all workers have a task number greater than zero. The number of tasks can be obtained with RM_GetMpiTasks. The number of tasks and computer hosts are determined at run time by the mpiexec command, and the number of reaction-module processes is defined by the communicator used in constructing the reaction modules (RM_Create).

- integer function rm_getmpitasks (id)

  Returns the number of MPI processes (tasks) assigned to the reaction module. For the OPENMP version, the number of tasks is always one (although there may be multiple threads, RM_GetThreadCount), and the task number returned by RM_GetMpiMyself is zero. For the MPI version, the number of tasks and computer hosts are determined at run time by the mpiexec command. An MPI communicator is used in constructing reaction modules for MPI. The communicator may define a subset of the total number of MPI processes. The root task number is zero, and all workers have a task number greater than zero.

- integer function rm_getnthselectedoutputusernumber (id, n)

  Returns the user number for the nth selected-output definition. Definitions are sorted by user number. Phreeqc allows multiple selected-output definitions, each of which is assigned a nonnegative integer identifier by the user. The number of definitions can be obtained by RM_GetSelectedOutputCount. To cycle through all of the definitions, RM_GetNthSelectedOutputUserNumber can be used to identify the user number for each selected-output definition in sequence. RM_SetCurrentSelectedOutputUserNumber is then used to select that user number for selected-output processing.

- integer function rm_getsaturation (id, sat_calc)

Returns a vector of saturations (sat_calc) as calculated by the reaction module. Reactions will change the volume of solution in a cell. The transport code must decide whether to ignore or account for this change in solution volume due to reactions. Following reactions, the cell saturation is calculated as solution volume (RM_GetSolutionVolume) divided by the product of representative volume (RM_SetRepresentativeVolume) and the porosity (RM_SetPorosity). The cell saturation returned by RM_GetSaturation may be less than or greater than the saturation set by the transport code (RM_SetSaturation), and may be greater than or less than 1.0, even in fully saturated simulations. Only the following databases distributed with PhreeqcRM have molar volume information needed to accurately calculate solution volume and saturation: phreeqc.dat, Amm.dat, and pitzer.dat.

- integer function rm_getselectedoutput (id, so)

  Populates an array with values from the current selected-output definition. RM_SetCurrentSelectedOutputUser↩ Number determines which of the selected-output definitions is used to populate the array.

- integer function rm_getselectedoutputcolumncount (id)

  Returns the number of columns in the current selected-output definition. RM_SetCurrentSelectedOutputUserNumber determines which of the selected-output definitions is used.

- integer function rm_getselectedoutputcount (id)

  Returns the number of selected-output definitions. RM_SetCurrentSelectedOutputUserNumber determines which of the selected-output definitions is used.

- integer function rm_getselectedoutputheading (id, icol, heading)

  Returns a selected output heading. The number of headings is determined by RM_GetSelectedOutputColumnCount. RM_SetCurrentSelectedOutputUserNumber determines which of the selected-output definitions is used.

- integer function rm_getselectedoutputrowcount (id)

  Returns the number of rows in the current selected-output definition. However, the method is included only for convenience; the number of rows is always equal to the number of grid cells in the user's model, and is equal to RM_GetGridCellCount.

- integer function rm_getsicount (id)

  Returns the number of phases in the initial-phreeqc module for which saturation indices can be calculated. RM↩ _FindComponents must be called before RM_GetSICount. This method may be useful when generating selected output definitions related to saturation indices.

- integer function rm_getsiname (id, num, name)

  Retrieves an item from the list of all phases for which saturation indices can be calculated. The list includes all phases that contain only elements included in the components in the initial-phreeqc module. The list assumes that all components are present to be able to calculate the entire list of SIs; it may be that one or more components are missing in any specific cell. RM_FindComponents must be called before RM_GetSIName. This method may be useful when generating selected output definitions related to saturation indices.

- integer function rm_getsolidsolutioncomponentscount (id)

  Returns the number of solid solution components in the initial-phreeqc module. RM_FindComponents must be called before RM_GetSolidSolutionComponentsCount. This method may be useful when generating selected output definitions related to solid solutions.

- integer function rm_getsolidsolutioncomponentsname (id, num, name)

  Retrieves an item from the solid solution components list. The list includes all solid solution components included in any SOLID_SOLUTIONS definitions in the initial-phreeqc module. RM_FindComponents must be called before RM_GetSolidSolutionComponentsName. This method may be useful when generating selected output definitions related to solid solutions.

- integer function rm_getsolidsolutionname (id, num, name)

  Retrieves an item from the solid solution names list. The list includes solid solution names included in SOLID_SO↩ LUTIONS definitions in the initial-phreeqc module. The solid solution names vector is the same length as the solid solution components vector and provides the corresponding name of solid solution containing the component. RM↩ _FindComponents must be called before RM_GetSolidSolutionName. This method may be useful when generating selected output definitions related to solid solutions.

- integer function rm_getsolutionvolume (id, vol)

  Transfer solution volumes from the reaction cells to the array given in the argument list (vol). Solution volumes are those calculated by the reaction module. Only the following databases distributed with PhreeqcRM have molar volume information needed to accurately calculate solution volume: phreeqc.dat, Amm.dat, and pitzer.dat.

- integer function rm_getspeciesconcentrations (id, species_conc)

  Transfer concentrations of aqueous species to the array argument (species_conc) This method is intended for use with multicomponent-diffusion transport calculations, and RM_SetSpeciesSaveOn must be set to true. The list of aqueous species is determined by RM_FindComponents and includes all aqueous species that can be made from

the set of components. Solution volumes used to calculate mol/L are calculated by the reaction module. Only the following databases distributed with PhreeqcRM have molar volume information needed to accurately calculate solution volume: phreeqc.dat, Amm.dat, and pitzer.dat.

- integer function rm_getspeciescount (id)

  The number of aqueous species used in the reaction module. This method is intended for use with multicomponent-diffusion transport calculations, and RM_SetSpeciesSaveOn must be set to true. The list of aqueous species is determined by RM_FindComponents and includes all aqueous species that can be made from the set of components.

- integer function rm_getspeciesd25 (id, diffc)

  Transfers diffusion coefficients at 25C to the array argument (diffc). This method is intended for use with multicomponent-diffusion transport calculations, and RM_SetSpeciesSaveOn must be set to true. Diffusion coefficients are defined in SOLUTION_SPECIES data blocks, normally in the database file. Databases distributed with the reaction module that have diffusion coefficients defined are phreeqc.dat, Amm.dat, and pitzer.dat.

- integer function rm_getspecieslog10gammas (id, species_log10gammas)

  Transfer log10 aqueous-species activity coefficients to the array argument (species_log10gammas) This method is intended for use with multicomponent-diffusion transport calculations, and RM_SetSpeciesSaveOn must be set to true. The list of aqueous species is determined by RM_FindComponents and includes all aqueous species that can be made from the set of components.

- integer function rm_getspecieslog10molalities (id, species_log10molalities)

  Transfer log10 aqueous-species log10 molalities to the array argument (species_log10molalities) To use this method RM_SetSpeciesSaveOn must be set to true. The list of aqueous species is determined by RM_FindComponents and includes all aqueous species that can be made from the set of components.

- integer function rm_getspeciesname (id, i, name)

  Transfers the name of the ith aqueous species to the character argument (name). This method is intended for use with multicomponent-diffusion transport calculations, and RM_SetSpeciesSaveOn must be set to true. The list of aqueous species is determined by RM_FindComponents and includes all aqueous species that can be made from the set of components.

- integer function rm_getspeciessaveon (id)

  Returns the value of the species-save property. By default, concentrations of aqueous species are not saved. Setting the species-save property to true allows aqueous species concentrations to be retrieved with RM_GetSpecies↩ Concentrations, and solution compositions to be set with RM_SpeciesConcentrations2Module.

- integer function rm_getspeciesz (id, z)

  Transfers the charge of each aqueous species to the array argument (z). This method is intended for use with multicomponent-diffusion transport calculations, and RM_SetSpeciesSaveOn must be set to true.

- integer function rm_getstartcell (id, sc)

  Returns an array with the starting cell numbers from the range of cell numbers assigned to each worker.

- integer function rm_getsurfacename (id, num, name)

  Retrieves the surface name (such as "Hfo") that corresponds with the surface species name. The lists of surface species names and surface names are the same length. RM_FindComponents must be called before RM_Get↩ SurfaceName. This method may be useful when generating selected output definitions related to surfaces.

- integer function rm_getsurfacespeciescount (id)

  Returns the number of surface species (such as "Hfo_wOH") in the initial-phreeqc module. RM_FindComponents must be called before RM_GetSurfaceSpeciesCount. This method may be useful when generating selected output definitions related to surfaces.

- integer function rm_getsurfacespeciesname (id, num, name)

  Retrieves an item from the surface species list. The list of surface species (for example, "Hfo_wOH") is derived from the list of components (RM_FindComponents) and the list of all surface types (such as "Hfo_w") that are included in SURFACE definitions in the initial-phreeqc module. RM_FindComponents must be called before RM_GetSurface↩ SpeciesName. This method may be useful when generating selected output definitions related to surfaces.

- integer function rm_getsurfacetype (id, num, name)

  Retrieves the surface site type (such as "Hfo_w") that corresponds with the surface species name. The lists of surface species names and surface species types are the same length. RM_FindComponents must be called before RM_↩ GetSurfaceType. This method may be useful when generating selected output definitions related to surfaces.

- integer function rm_getthreadcount (id)

  Returns the number of threads, which is equal to the number of workers used to run in parallel with OPENMP. For the OPENMP version, the number of threads is set implicitly or explicitly with RM_Create. For the MPI version, the number of threads is always one for each process.

- double precision function rm_gettime (id)

Returns the current simulation time in seconds. The reaction module does not change the time value, so the returned value is equal to the default (0.0) or the last time set by RM_SetTime.

- double precision function rm_gettimeconversion (id)

  Returns a multiplier to convert time from seconds to another unit, as specified by the user. The reaction module uses seconds as the time unit. The user can set a conversion factor (RM_SetTimeConversion) and retrieve it with RM_GetTimeConversion. The reaction module only uses the conversion factor when printing the long version of cell chemistry (RM_SetPrintChemistryOn), which is rare. Default conversion factor is 1.0.

- double precision function rm_gettimestep (id)

  Returns the current simulation time step in seconds. This is the time over which kinetic reactions are integrated in a call to RM_RunCells. The reaction module does not change the time step value, so the returned value is equal to the default (0.0) or the last time step set by RM_SetTimeStep.

- integer function rm_initialphreeqc2concentrations (id, bc_conc, n_boundary, bc1, bc2, f1)

  Fills an array (bc_conc) with concentrations from solutions in the InitialPhreeqc instance. The method is used to obtain concentrations for boundary conditions. If a negative value is used for a cell in bc1, then the highest numbered solution in the InitialPhreeqc instance will be used for that cell. Concentrations may be a mixture of two solutions, bc1 and bc2, with a mixing fraction for bc1 1 of f1 and mixing fraction for bc2 of (1 - f1). A negative value for bc2 implies no mixing, and the associated value for f1 is ignored. If bc2 and f1 are omitted, no mixing is used; concentrations are derived from bc1 only.

- integer function rm_initialphreeqc2module (id, ic1, ic2, f1)

  Transfer solutions and reactants from the InitialPhreeqc instance to the reaction-module workers, possibly with mixing. In its simplest form, ic1 is used to select initial conditions, including solutions and reactants, for each cell of the model, without mixing. ic1 is dimensioned (nxyz, 7), where nxyz is the number of grid cells in the user's model (RM_GetGridCellCount). The dimension of 7 refers to solutions and reactants in the following order: (1) SOLUTIONS, (2) EQUILIBRIUM_PHASES, (3) EXCHANGE, (4) SURFACE, (5) GAS_PHASE, (6) SOLID_SOLUTIONS, and (7) KINETICS. In Fortran, ic1(100, 4) = 2, indicates that cell 99 (0 based) contains the SURFACE definition with user number 2 that has been defined in the InitialPhreeqc instance (either by RM_RunFile or RM_RunString).

  It is also possible to mix solutions and reactants to obtain the initial conditions for cells. For mixing, ic2 contains numbers for a second entity that mixes with the entity defined in ic1. F1 contains the mixing fraction for ic1, whereas (1 - f1) is the mixing fraction for ic2. In Fortran, ic1(100, 4) = 2, initial_conditions2(100, 4) = 3, f1(100, 4) = 0.25 indicates that cell 99 (0 based) contains a mixture of 0.25 SURFACE 2 and 0.75 SURFACE 3, where the surface compositions have been defined in the InitialPhreeqc instance. If the user number in ic2 is negative, no mixing occurs. If ic2 and f1 are omitted, no mixing is used, and initial conditions are derived solely from ic1.

- integer function rm_initialphreeqc2speciesconcentrations (id, bc_conc, n_boundary, bc1, bc2, f1)

  Fills an array (bc_conc) with aqueous species concentrations from solutions in the InitialPhreeqc instance. This method is intended for use with multicomponent-diffusion transport calculations, and RM_SetSpeciesSaveOn must be set to true. The method is used to obtain aqueous species concentrations for boundary conditions. If a negative value is used for a cell in bc1, then the highest numbered solution in the InitialPhreeqc instance will be used for that cell. Concentrations may be a mixture of two solutions, bc1 and bc2, with a mixing fraction for bc1 of f1 and mixing fraction for bc2 of (1 - f1). A negative value for bc2 implies no mixing, and the associated value for f1 is ignored. If bc2 and f1 are omitted, no mixing is used; concentrations are derived from bc1 only.

- integer function rm_initialphreeqccell2module (id, n_user, cell_numbers, n_cell)

  A cell numbered n_user in the InitialPhreeqc instance is selected to populate a series of cells. All reactants with the number n_user are transferred along with the solution. If MIX n_user exists, it is used for the definition of the solution. If n_user is negative, n_user is redefined to be the largest solution or MIX number in the InitialPhreeqc instance. All reactants for each cell in the list cell_numbers are removed before the cell definition is copied from the InitialPhreeqc instance to the workers.

- integer function rm_loaddatabase (id, db_name)

  Load a database for all IPhreeqc instances–workers, InitialPhreeqc, and Utility. All definitions of the reaction module are cleared (SOLUTION_SPECIES, PHASES, SOLUTIONs, etc.), and the database is read.

- integer function rm_logmessage (id, str)

  Print a message to the log file.

- integer function rm_mpiworker (id)

  MPI only. Workers (processes with RM_GetMpiMyself $>$ 0) must call RM_MpiWorker to be able to respond to messages from the root to accept data, perform calculations, and (or) return data. RM_MpiWorker contains a loop that reads a message from root, performs a task, and waits for another message from root. RM_SetConcentrations, RM_RunCells, and RM_GetConcentrations are examples of methods that send a message from root to get the workers to perform a task. The workers will respond to all methods that are designated "workers must be in the loop of RM_MpiWorker" in the MPI section of the method documentation. The workers will continue to respond to

messages from root until root calls RM_MpiWorkerBreak.

(Advanced) The list of tasks that the workers perform can be extended by using RM_SetMpiWorkerCallback. It is then possible to use the MPI processes to perform other developer-defined tasks, such as transport calculations, without exiting from the RM_MpiWorker loop. Alternatively, root calls RM_MpiWorkerBreak to allow the workers to continue past a call to RM_MpiWorker. The workers perform developer-defined calculations, and then RM_MpiWorker is called again to respond to requests from root to perform reaction-module tasks.

- integer function rm_mpiworkerbreak (id)

  MPI only. This method is called by root to force workers (processes with RM_GetMpiMyself > 0) to return from a call to RM_MpiWorker. RM_MpiWorker contains a loop that reads a message from root, performs a task, and waits for another message from root. The workers respond to all methods that are designated "workers must be in the loop of RM_MpiWorker" in the MPI section of the method documentation. The workers will continue to respond to messages from root until root calls RM_MpiWorkerBreak.

- integer function rm_openfiles (id)

  Opens the output and log files. Files are named prefix.chem.txt and prefix.log.txt based on the prefix defined by RM_SetFilePrefix.

- integer function rm_outputmessage (id, str)

  Print a message to the output file.

- integer function rm_runcells (id)

  Runs a reaction step for all of the cells in the reaction module. Normally, tranport concentrations are transferred to the reaction cells (RM_SetConcentrations) before reaction calculations are run. The length of time over which kinetic reactions are integrated is set by RM_SetTimeStep. Other properties that may need to be updated as a result of the transport calculations include porosity (RM_SetPorosity), saturation (RM_SetSaturation), temperature (RM_SetTemperature), and pressure (RM_SetPressure).

- integer function rm_runfile (id, workers, initial_phreeqc, utility, chem_name)

  Run a PHREEQC input file. The first three arguments determine which IPhreeqc instances will run the file–the workers, the InitialPhreeqc instance, and (or) the Utility instance. Input files that modify the thermodynamic database should be run by all three sets of instances. Files with SELECTED_OUTPUT definitions that will be used during the time-stepping loop need to be run by the workers. Files that contain initial conditions or boundary conditions should be run by the InitialPhreeqc instance.

- integer function rm_runstring (id, workers, initial_phreeqc, utility, input_string)

  Run a PHREEQC input string. The first three arguments determine which IPhreeqc instances will run the string–the workers, the InitialPhreeqc instance, and (or) the Utility instance. Input strings that modify the thermodynamic database should be run by all three sets of instances. Strings with SELECTED_OUTPUT definitions that will be used during the time-stepping loop need to be run by the workers. Strings that contain initial conditions or boundary conditions should be run by the InitialPhreeqc instance.

- integer function rm_screenmessage (id, str)

  Print message to the screen.

- integer function rm_setcomponenth2o (id, tf)

  Select whether to include H2O in the component list. The concentrations of H and O must be known accurately (8 to 10 significant digits) for the numerical method of PHREEQC to produce accurate pH and pe values. Because most of the H and O are in the water species, it may be more robust (require less accuracy in transport) to transport the excess H and O (the H and O not in water) and water. The default setting (true) is to include water, excess H, and excess O as components. A setting of false will include total H and total O as components. RM_SetComponentH2O must be called before RM_FindComponents.

- integer function rm_setconcentrations (id, c)

  Use the vector of concentrations (c) to set the moles of components in each reaction cell. The volume of water in a cell is the product of porosity (RM_SetPorosity), saturation (RM_SetSaturation), and reference volume (RM↩_SetRepresentativeVolume). The moles of each component are determined by the volume of water and per liter concentrations. If concentration units (RM_SetUnitsSolution) are mass fraction, the density (as specified by RM_↩SetDensity) is used to convert from mass fraction to per mass per liter.

- integer function **rm_setconcentrations1d** (id, c)

- integer function rm_setcurrentselectedoutputusernumber (id, n_user)

  Select the current selected output by user number. The user may define multiple SELECTED_OUTPUT data blocks for the workers. A user number is specified for each data block. The value of the argument n_user selects which of the SELECTED_OUTPUT definitions will be used for selected-output operations.

- integer function rm_setdensity (id, density)

Set the density for each reaction cell. These density values are used when converting from transported mass fraction concentrations (RM_SetUnitsSolution) to produce per liter concentrations during a call to RM_SetConcentrations. They are also used when converting from module concentrations to transport concentrations of mass fraction (RM_GetConcentrations), if RM_UseSolutionDensityVolume is set to false.

- integer function rm_setdumpfilename (id, dump_name)

  Set the name of the dump file. It is the name used by RM_DumpModule.

- integer function rm_seterrorhandlermode (id, mode)

  Set the action to be taken when the reaction module encounters an error. Options are 0, return to calling program with an error return code (default); 1, throw an exception, in C++, the exception can be caught, for C and Fortran, the program will exit; or 2, attempt to exit gracefully.

- integer function rm_seterroron (id, tf)

  Set the property that controls whether error messages are generated and displayed. Messages include PHREEQC "ERROR" messages, and any messages written with RM_ErrorMessage.

- integer function rm_setfileprefix (id, prefix)

  Set the prefix for the output (prefix.chem.txt) and log (prefix.log.txt) files. These files are opened by RM_OpenFiles.

- integer function rm_setgascompmoles (id, gas_moles)

  Use the array of concentrations (gas_moles) to set the moles of gas components in each reaction cell.

- integer function rm_setgasphasevolume (id, gas_volume)

  Transfer volumes of gas phases from the array given in the argument list (gas_volume) to each reaction cell. The gas-phase volume affects the pressures calculated for fixed-volume gas phases. If a gas-phase volume is defined with this method for a GAS_PHASE in a cell, the gas phase is forced to be a fixed-volume gas phase.

- integer function rm_setmpiworkercallback (id, fcn)

  MPI only. Defines a callback function that allows additional tasks to be done by the workers. The method RM_MpiWorker contains a loop, where the workers receive a message (an integer), run a function corresponding to that integer, and then wait for another message. RM_SetMpiWorkerCallback allows the developer to add another function that responds to additional integer messages by calling developer-defined functions corresponding to those integers. RM_MpiWorker calls the callback function when the message number is not one of the PhreeqcRM message numbers. Messages are unique integer numbers. PhreeqcRM uses integers in a range beginning at 0. It is suggested that developers use message numbers starting at 1000 or higher for their tasks. The callback function calls a developer-defined function specified by the message number and then returns to RM_MpiWorker to wait for another message.

  For Fortran, the functions that are called from the callback function can use USE statements to find the data necessary to perform the tasks, and the only argument to the callback function is an integer message argument. RM_SetMpiWorkerCallback must be called by each worker before RM_MpiWorker is called.

  The motivation for this method is to allow the workers to perform other tasks, for instance, parallel transport calculations, within the structure of RM_MpiWorker. The callback function can be used to allow the workers to receive data, perform transport calculations, and (or) send results, without leaving the loop of RM_MpiWorker. Alternatively, it is possible for the workers to return from RM_MpiWorker by a call to RM_MpiWorkerBreak by root. The workers could then call subroutines to receive data, calculate transport, and send data, and then resume processing PhreeqcRM messages from root with another call to RM_MpiWorker.

- integer function rm_setpartitionuzsolids (id, tf)

  Sets the property for partitioning solids between the saturated and unsaturated parts of a partially saturated cell.

- integer function rm_setporosity (id, por)

  Set the porosity for each reaction cell. The volume of water in a reaction cell is the product of the porosity, the saturation (RM_SetSaturation), and the representative volume (RM_SetRepresentativeVolume).

- integer function rm_setpressure (id, p)

  Set the pressure for each reaction cell. Pressure effects are considered only in three of the databases distributed with PhreeqcRM: phreeqc.dat, Amm.dat, and pitzer.dat.

- integer function rm_setprintchemistrymask (id, cell_mask)

  Enable or disable detailed output for each reaction cell. Printing for a cell will occur only when the printing is enabled with RM_SetPrintChemistryOn and the cell_mask value is 1.

- integer function rm_setprintchemistryon (id, workers, initial_phreeqc, utility)

  Setting to enable or disable printing detailed output from reaction calculations to the output file for a set of cells defined by RM_SetPrintChemistryMask. The detailed output prints all of the output typical of a PHREEQC reaction calculation, which includes solution descriptions and the compositions of all other reactants. The output can be

several hundred lines per cell, which can lead to a very large output file (prefix.chem.txt, RM_OpenFiles). For the worker instances, the output can be limited to a set of cells (RM_SetPrintChemistryMask) and, in general, the amount of information printed can be limited by use of options in the PRINT data block of PHREEQC (applied by using RM_RunFile or RM_RunString). Printing the detailed output for the workers is generally used only for debugging, and PhreeqcRM will run significantly faster when printing detailed output for the workers is disabled.

- integer function rm_setrebalancebycell (id, method)

  Set the load-balancing algorithm. PhreeqcRM attempts to rebalance the load of each thread or process such that each thread or process takes the same amount of time to run its part of a RM_RunCells calculation. Two algorithms are available; one uses individual times for each cell and accounts for cells that were not run because saturation was zero (default), and the other assigns an average time to all cells. The methods are similar, but limited testing indicates the default method performs better.

- integer function rm_setrebalancefraction (id, f)

  Sets the fraction of cells that are transferred among threads or processes when rebalancing. PhreeqcRM attempts to rebalance the load of each thread or process such that each thread or process takes the same amount of time to run its part of a RM_RunCells calculation. The rebalancing transfers cell calculations among threads or processes to try to achieve an optimum balance. RM_SetRebalanceFraction adjusts the calculated optimum number of cell transfers by a fraction from 0 to 1.0 to determine the actual number of cell transfers. A value of zero eliminates load rebalancing. A value less than 1.0 is suggested to slow the approach to the optimum cell distribution and avoid possible oscillations when too many cells are transferred at one iteration, requiring reverse transfers at the next iteration. Default is 0.5.

- integer function rm_setrepresentativevolume (id, rv)

  Set the representative volume of each reaction cell. By default the representative volume of each reaction cell is 1 liter. The volume of water in a reaction cell is determined by the procuct of the representative volume, the porosity (RM_SetPorosity), and the saturation (RM_SetSaturation). The numerical method of PHREEQC is more robust if the water volume for a reaction cell is within a couple orders of magnitude of 1.0. Small water volumes caused by small porosities and (or) small saturations (and (or) small representative volumes) may cause non-convergence of the numerical method. In these cases, a larger representative volume may help. Note that increasing the representative volume also increases the number of moles of the reactants in the reaction cell (minerals, surfaces, exchangers, and others), which are defined as moles per representative volume.

- integer function rm_setsaturation (id, sat)

  Set the saturation of each reaction cell. Saturation is a fraction ranging from 0 to 1. The volume of water in a cell is the product of porosity (RM_SetPorosity), saturation (RM_SetSaturation), and representative volume (RM_SetRepresentativeVolume). As a result of a reaction calculation, solution properties (density and volume) will change; the databases phreeqc.dat, Amm.dat, and pitzer.dat have the molar volume data to calculate these changes. The methods RM_GetDensity, RM_GetSolutionVolume, and RM_GetSaturation can be used to account for these changes in the succeeding transport calculation. RM_SetRepresentativeVolume should be called before initial conditions are defined for the reaction cells.

- integer function rm_setscreenon (id, tf)

  Set the property that controls whether messages are written to the screen. Messages include information about rebalancing during RM_RunCells, and any messages written with RM_ScreenMessage.

- integer function rm_setselectedoutputon (id, tf)

  Setting determines whether selected-output results are available to be retrieved with RM_GetSelectedOutput. 1 indicates that selected-output results will be accumulated during RM_RunCells and can be retrieved with RM_GetSelectedOutput; 0 indicates that selected-output results will not be accumulated during RM_RunCells.

- integer function rm_setspeciessaveon (id, save_on)

  Sets the value of the species-save property. This method enables use of PhreeqcRM with multicomponent-diffusion transport calculations. By default, concentrations of aqueous species are not saved. Setting the species-save property to 1 allows aqueous species concentrations to be retrieved with RM_GetSpeciesConcentrations, and solution compositions to be set with RM_SpeciesConcentrations2Module. RM_SetSpeciesSaveOn must be called before calls to RM_FindComponents.

- integer function rm_settemperature (id, t)

  Set the temperature for each reaction cell. If RM_SetTemperature is not called, worker solutions will have temperatures as defined by initial conditions (RM_InitialPhreeqc2Module and RM_InitialPhreeqcCell2Module).

- integer function rm_settime (id, time)

  Set current simulation time for the reaction module.

- integer function rm_settimeconversion (id, conv_factor)

  Set a factor to convert to user time units. Factor times seconds produces user time units.

- integer function rm_settimestep (id, time_step)

  Set current time step for the reaction module. This is the length of time over which kinetic reactions are integrated.

- integer function rm_setunitsexchange (id, option)

  Sets input units for exchangers. In PHREEQC input, exchangers are defined by moles of exchange sites (Mp). RM_SetUnitsExchange specifies how the number of moles of exchange sites in a reaction cell (Mc) is calculated from the input value (Mp).
- integer function rm_setunitsgasphase (id, option)

  Set input units for gas phases. In PHREEQC input, gas phases are defined by moles of component gases (Mp). RM_SetUnitsGasPhase specifies how the number of moles of component gases in a reaction cell (Mc) is calculated from the input value (Mp).
- integer function rm_setunitskinetics (id, option)

  Set input units for kinetic reactants.
- integer function rm_setunitsppassemblage (id, option)

  Set input units for pure phase assemblages (equilibrium phases). In PHREEQC input, equilibrium phases are defined by moles of each phase (Mp). RM_SetUnitsPPassemblage specifies how the number of moles of phases in a reaction cell (Mc) is calculated from the input value (Mp).
- integer function rm_setunitssolution (id, option)

  Solution concentration units used by the transport model. Options are 1, mg/L; 2 mol/L; or 3, mass fraction, kg/kgs. PHREEQC defines solutions by the number of moles of each element in the solution.

  To convert from mg/L to moles of element in the representative volume of a reaction cell, mg/L is converted to mol/L and multiplied by the solution volume, which is the product of porosity (RM_SetPorosity), saturation (RM_SetSaturation), and representative volume (RM_SetRepresentativeVolume). To convert from mol/L to moles of element in the representative volume of a reaction cell, mol/L is multiplied by the solution volume. To convert from mass fraction to moles of element in the representative volume of a reaction cell, kg/kgs is converted to mol/kgs, multiplied by density (RM_SetDensity) and multiplied by the solution volume.
- integer function rm_setunitsssassemblage (id, option)

  Set input units for solid-solution assemblages. In PHREEQC, solid solutions are defined by moles of each component (Mp). RM_SetUnitsSSassemblage specifies how the number of moles of solid-solution components in a reaction cell (Mc) is calculated from the input value (Mp).
- integer function rm_setunitssurface (id, option)

  Set input units for surfaces. In PHREEQC input, surfaces are defined by moles of surface sites (Mp). RM_SetUnitsSurface specifies how the number of moles of surface sites in a reaction cell (Mc) is calculated from the input value (Mp).
- integer function rm_speciesconcentrations2module (id, species_conc)

  Set solution concentrations in the reaction cells based on the vector of aqueous species concentrations (species_conc). This method is intended for use with multicomponent-diffusion transport calculations, and RM_SetSpeciesSaveOn must be set to true. The list of aqueous species is determined by RM_FindComponents and includes all aqueous species that can be made from the set of components. The method determines the total concentration of a component by summing the molarities of the individual species times the stoichiometric coefficient of the element in each species. Solution compositions in the reaction cells are updated with these component concentrations.
- integer function rm_usesolutiondensityvolume (id, tf)

  Determines the volume and density to use when converting from the reaction-module concentrations to transport concentrations (RM_GetConcentrations). Two options are available to convert concentration units: (1) the density and solution volume calculated by PHREEQC are used, or (2) the specified density (RM_SetDensity) and solution volume are defined by the product of saturation (RM_SetSaturation), porosity (RM_SetPorosity), and representative volume (RM_SetRepresentativeVolume). Transport models that consider density-dependent flow will probably use the PHREEQC-calculated density and solution volume (default), whereas transport models that assume constant-density flow will probably use specified values of density and solution volume. Only the following databases distributed with PhreeqcRM have molar volume information needed to accurately calculate density and solution volume: phreeqc.dat, Amm.dat, and pitzer.dat. Density is only used when converting to transport units of mass fraction.
- integer function rm_warningmessage (id, warn_str)

  Print a warning message to the screen and the log file.

3.3.1 Detailed Description

Fortran Documentation for the geochemical reaction module PhreeqcRM.

"USE PhreeqcRM" is included in Fortran source code to define the PhreeqcRM functions. For Windows, define the module by including the file RM_interface.F90 in your project. For Linux, configure, compile, and install the PhreeqcRM library and module file. You will need installed include directory (-I) added to the project) to reference the module file. You will need to link to the library to produce the executable for your code.

3.3.2 Function/Subroutine Documentation

3.3.2.1 rm_abort()

```
integer function phreeqcrm::rm_abort (
             integer, intent(in) id,
             integer, intent(in) irm_result,
             character(len=*), intent(in) err_str )
```

Abort the program. *irm_result* will be interpreted as an IRM_RESULT value and decoded; *err_str* will be printed; and the reaction module will be destroyed. If using MPI, an MPI_Abort message will be sent before the reaction module is destroyed. If the *id* is an invalid instance, RM_Abort will return a value of IRM_BADINSTANCE, otherwise the program will exit with a return code of 4.

Parameters

id	The instance id returned from RM_Create.
irm_result	Integer treated as an IRM_RESULT return code.
err_str	String to be printed as an error message.

Return values

IRM_RESULT	Program will exit before returning unless *id* is an invalid reaction module id.

See also

RM_Destroy, RM_ErrorMessage.

Fortran Example:

MPI:

Called by root or workers.

3.3.2.2 rm_closefiles()

```
integer function phreeqcrm::rm_closefiles (
             integer, intent(in) id )
```

Close the output and log files.

Parameters

id	The instance *id* returned from RM_Create.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_OpenFiles, RM_SetFilePrefix.

Fortran Example:

MPI:

Called only by root.

3.3.2.3 rm_concentrations2utility()

```
integer function phreeqcrm::rm_concentrations2utility (
             integer, intent(in) id,
             double precision, dimension(:,:), intent(in) c,
             integer, intent(in) n,
             double precision, dimension(:), intent(in) tc,
             double precision, dimension(:), intent(in) p_atm )
```

N sets of component concentrations are converted to SOLUTIONs numbered 1-*n* in the Utility IPhreeqc. The solutions can be reacted and manipulated with the methods of IPhreeqc. If solution concentration units (RM_↩ SetUnitsSolution) are per liter, one liter of solution is created in the Utility instance; if solution concentration units are mass fraction, one kilogram of solution is created in the Utility instance. The motivation for this method is the mixing of solutions in wells, where it may be necessary to calculate solution properties (pH for example) or react the mixture to form scale minerals. The code fragments below make a mixture of concentrations and then calculate the pH of the mixture.

Parameters

id	The instance *id* returned from RM_Create.
c	Array of concentrations to be made SOLUTIONs in Utility IPhreeqc, array size is (*n*, *ncomps*) where *ncomps* is the number of components (RM_GetComponentCount).
n	The number of sets of concentrations.
tc	Array of temperatures to apply to the SOLUTIONs, in degree C. Array of size *n*.
p_atm	Array of pressures to apply to the SOLUTIONs, in atm. Array of size n.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

Fortran Example:

MPI:

Called only by root.

3.3.2.4 rm_create()

```
integer function phreeqcrm::rm_create (
             integer, intent(in) nxyz,
             integer, intent(in) nthreads )
```

Creates a reaction module. If the code is compiled with the preprocessor directive USE_OPENMP, the reaction module is multithreaded. If the code is compiled with the preprocessor directive USE_MPI, the reaction module will use MPI and multiple processes. If neither preprocessor directive is used, the reaction module will be serial (unparallelized).

Parameters

nxyz	The number of grid cells in the user's model.
nthreads	(or *comm*, MPI) When using OPENMP, the argument (*nthreads*) is the number of worker threads to be used. If *nthreads* <= 0, the number of threads is set equal to the number of processors of the computer. When using MPI, the argument (*comm*) is the MPI communicator to use within the reaction module.

Return values

Id	of the PhreeqcRM instance, negative is failure (See RM_DecodeError).

See also

RM_Destroy.

Fortran Example:

MPI:

Called by root and workers.

3.3.2.5 rm_createmapping()

```
integer function phreeqcrm::rm_createmapping (
             integer, intent(in) id,
             integer, dimension(:), intent(in) grid2chem )
```

Provides a mapping from grid cells in the user's model to reaction cells in PhreeqcRM. The mapping is used to eliminate inactive cells and to use symmetry to decrease the number of cells for which chemistry must be run. The array *grid2chem* of size *nxyz* (the number of grid cells, RM_GetGridCellCount) must contain the set of all integers 0 <= *i* < *count_chemistry*, where *count_chemistry* is a number less than or equal to *nxyz*. Inactive cells are assigned a negative integer. The mapping may be many-to-one to account for symmetry. Default is a one-to-one mapping–all user grid cells are reaction cells (equivalent to *grid2chem* values of 0,1,2,3,...,*nxyz-1*).

Parameters

id	The instance *id* returned from RM_Create.
grid2chem	An array of integers: Nonnegative is a reaction cell number (0 based), negative is an inactive cell. Array of size *nxyz* (number of grid cells).

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.6 rm_decodeerror()

```
integer function phreeqcrm::rm_decodeerror (
             integer, intent(in) id,
             integer, intent(in) e )
```

If *e* is negative, this method prints an error message corresponding to IRM_RESULT *e*. If *e* is non-negative, no action is taken.

Parameters

id	The instance *id* returned from RM_Create.
e	An IRM_RESULT value returned by one of the reaction-module methods.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

IRM_RESULT definition:

Fortran Example:

MPI:

Can be called by root and (or) workers.

3.3.2.7 rm_destroy()

```
integer function phreeqcrm::rm_destroy (
             integer, intent(in) id )
```

Destroys a reaction module.

Parameters

id	The instance *id* returned from RM_Create.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_Create.

Fortran Example:

MPI:

Called by root and workers.

3.3.2.8 rm_dumpmodule()

```
integer function phreeqcrm::rm_dumpmodule (
             integer, intent(in) id,
             integer, intent(in) dump_on,
             integer, intent(in) append )
```

Writes the contents of all workers to file in _RAW formats, including SOLUTIONs and all reactants.

Parameters

id	The instance *id* returned from RM_Create.
dump_on	Signal for writing the dump file: 1 true, 0 false.
append	Signal to append to the contents of the dump file: 1 true, 0 false.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_SetDumpFileName.

Fortran Example:

MPI:

Called by root; workers must be in the loop of RM_MpiWorker.

3.3.2.9 rm_errormessage()

```
integer function phreeqcrm::rm_errormessage (
             integer, intent(in) id,
             character(len=*), intent(in) errstr )
```

Send an error message to the screen, the output file, and the log file.

Parameters

id	The instance *id* returned from RM_Create.
errstr	String to be printed.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_LogMessage,
RM_OpenFiles,
RM_OutputMessage, RM_ScreenMessage, RM_WarningMessage.

Fortran Example:

MPI:

Called by root and (or) workers; root writes to output and log files.

3.3.2.10 rm_findcomponents()

```
integer function phreeqcrm::rm_findcomponents (
            integer, intent(in) id )
```

Returns the number of items in the list of all elements in the InitialPhreeqc instance. Elements are those that have been defined in a solution or any other reactant (EQUILIBRIUM_PHASE, KINETICS, and others). The method can be called multiple times and the list that is created is cummulative. The list is the set of components that needs to be transported. By default the list includes water, excess H and excess O (the H and O not contained in water); alternatively, the list may be set to contain total H and total O (RM_SetComponentH2O), which requires transport results to be accurate to eight or nine significant digits. If multicomponent diffusion (MCD) is to be modeled, there is a capability to retrieve aqueous species concentrations (RM_GetSpeciesConcentrations) and to set new solution concentrations after MCD by using individual species concentrations (RM_SpeciesConcentrations2Module). To use these methods the save-species property needs to be turned on (RM_SetSpeciesSaveOn). If the save-species property is on, RM_FindComponents will generate a list of aqueous species (RM_GetSpeciesCount, RM_Get↩SpeciesName), their diffusion coefficients at 25 C (RM_GetSpeciesD25), their charge (RM_GetSpeciesZ).

Parameters

id	The instance *id* returned from RM_Create.

Return values

Number	of components currently in the list, or IRM_RESULT error code (see RM_DecodeError).

See also

RM_GetComponent, RM_GetSpeciesConcentrations,
RM_GetSpeciesCount, RM_GetSpeciesD25, RM_GetSpeciesLog10Gammas, RM_GetSpeciesLog10↩Molalities, RM_GetSpeciesName,
RM_GetSpeciesZ, RM_SetComponentH2O. RM_SetSpeciesSaveOn, RM_SpeciesConcentrations2Module.

The RM_FindComponents method also generates lists of reactants–equilibrium phases,

exchangers, gas components, kinetic reactants, solid solution components, and surfaces. The lists are cumulative, including all reactants that were defined in the initial phreeqc instance at any time RM_FindComponents was called. In addition, a list of phases is generated for which saturation indices may be calculated from the cumulative list of components.

See also

also RM_GetEquilibriumPhasesName, RM_GetEquilibriumPhasesCount, RM_GetExchangeName, RM_↩ GetExchangeSpeciesName, RM_GetExchangeSpeciesCount, RM_GetGasComponentsName, RM_Get↩ GasComponentsCount, RM_GetKineticReactionsName, RM_GetKineticReactionsCount, RM_GetSICount, RM_GetSIName, RM_GetSolidSolutionComponentsName, RM_GetSolidSolutionComponentsCount, RM↩ _GetSolidSolutionName, RM_GetSurfaceName, RM_GetSurfaceSpeciesName, RM_GetSurfaceSpecies↩ Count, RM_GetSurfaceType.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.11 rm_getbackwardmapping()

```
integer function phreeqcrm::rm_getbackwardmapping (
             integer, intent(in) id,
             integer, intent(in) n,
             integer, dimension(*), intent(in) list,
             integer, intent(inout) size )
```

Fills an array with the cell numbers in the user's numbering sytstem that map to a cell in the PhreeqcRM numbering system. The mapping is defined by RM_CreateMapping.

Parameters

id	The instance *id* returned from RM_Create.
n	A cell number in the PhreeqcRM numbering system (0 <= n < RM_GetChemistryCellCount).
list	Array to store the user cell numbers mapped to PhreeqcRM cell *n*.
size	Input, the allocated size of *list*; it is an error if the array is too small. Output, the number of cells mapped to cell *n*.

Return values

IRM_RESULT	error code (see RM_DecodeError).

See also

RM_CreateMapping, RM_GetChemistryCellCount, RM_GetGridCellCount.

C Example:

MPI:

Called by root and (or) workers.

3.3.2.12 rm_getchemistrycellcount()

```
integer function phreeqcrm::rm_getchemistrycellcount (
             integer, intent(in) id )
```

Returns the number of chemistry cells in the reaction module. The number of chemistry cells is defined by the set of non-negative integers in the mapping from user grid cells (RM_CreateMapping). The number of chemistry cells is less than or equal to the number of cells in the user's model.

Parameters

id	The instance *id* returned from RM_Create.

Return values

Number	of chemistry cells, or IRM_RESULT error code (see RM_DecodeError).

See also

RM_CreateMapping, RM_GetGridCellCount.

Fortran Example:

MPI:

Called by root and (or) workers.

3.3.2.13 rm_getcomponent()

```
integer function phreeqcrm::rm_getcomponent (
             integer, intent(in) id,
             integer, intent(in) num,
             character(len=*), intent(inout) comp_name )
```

Retrieves an item from the reaction-module component list that was generated by calls to RM_FindComponents.

Parameters

id	The instance *id* returned from RM_Create.
num	The number of the component to be retrieved. Fortran, 1 based.
comp_name	The string value associated with component *num*.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_FindComponents, RM_GetComponentCount.

Fortran Example:

MPI:

Called by root and (or) workers.

3.3.2.14 rm_getcomponentcount()

```
integer function phreeqcrm::rm_getcomponentcount (
             integer, intent(in) id )
```

Returns the number of components in the reaction-module component list. The component list is generated by calls to RM_FindComponents. The return value from the last call to RM_FindComponents is equal to the return value from RM_GetComponentCount.

Parameters

id	The instance *id* returned from RM_Create.

Return values

The	number of components in the reaction-module component list, negative is failure (See RM_DecodeError).

See also

RM_FindComponents, RM_GetComponent.

Fortran Example:

MPI:

Called by root.

3.3.2.15 rm_getconcentrations()

```
integer function phreeqcrm::rm_getconcentrations (
             integer, intent(in) id,
             double precision, dimension(:,:), intent(out) c )
```

Transfer solution concentrations from each reaction cell to the concentration array given in the argument list (*c*). Units of concentration for *c* are defined by RM_SetUnitsSolution. For concentration units of per liter, the solution volume is used to calculate the concentrations for *c*. For mass fraction concentration units, the solution mass is used to calculate concentrations for *c*. Two options are available for the volume and mass of solution that are used in converting to transport concentrations: (1) the volume and mass of solution are calculated by PHREEQC, or (2) the volume of solution is the product of saturation (RM_SetSaturation), porosity (RM_SetPorosity), and representative volume (RM_SetRepresentativeVolume), and the mass of solution is volume times density as defined by RM_SetDensity. RM_UseSolutionDensityVolume determines which option is used. For option 1, the databases that have partial molar volume definitions needed to accurately calculate solution volume are phreeqc.dat, Amm.dat, and pitzer.dat.

Parameters

id	The instance *id* returned from RM_Create.
c	Array to receive the concentrations. Dimension of the array is (*nxyz*, *ncomps*), where *nxyz* is the number of user grid cells and *ncomps* is the result of RM_FindComponents or RM_GetComponentCount. Values for inactive cells are set to 1e30.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_FindComponents, RM_GetComponentCount, RM_GetSaturation, RM_SetConcentrations, RM_SetDensity, RM_SetRepresentativeVolume, RM_SetSaturation, RM_SetUnitsSolution, RM_UseSolutionDensityVolume.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.16 rm_getdensity()

```
integer function phreeqcrm::rm_getdensity (
             integer, intent(in) id,
             double precision, dimension(:), intent(out) density )
```

Transfer solution densities from the reaction cells to the array given in the argument list (*density*). Densities are those calculated by the reaction module. Only the following databases distributed with PhreeqcRM have molar volume information needed to accurately calculate density: phreeqc.dat, Amm.dat, and pitzer.dat.

Parameters

id	The instance *id* returned from RM_Create.
density	Array to receive the densities. Dimension of the array is *nxyz*, where *nxyz* is the number of user grid cells (RM_GetGridCellCount). Values for inactive cells are set to 1e30.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.17 rm_getendcell()

```
integer function phreeqcrm::rm_getendcell (
             integer, intent(in) id,
             integer, dimension(:), intent(out) ec )
```

Returns an array with the ending cell numbers from the range of cell numbers assigned to each worker.

Parameters

id	The instance *id* returned from RM_Create.
ec	Array to receive the ending cell numbers. Dimension of the array is the number of threads (OpenMP) or the number of processes (MPI).

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_Create, RM_GetMpiTasks, RM_GetStartCell, RM_GetThreadCount.

Fortran Example:

MPI:

Called by root and (or) workers.

3.3.2.18 rm_getequilibriumphasescount()

```
integer function phreeqcrm::rm_getequilibriumphasescount (
             integer, intent(in) id )
```

Returns the number of equilibrium phases in the initial-phreeqc module. RM_FindComponents must be called before RM_GetEquilibriumPhasesCount. This method may be useful when generating selected output definitions related to equilibrium phases.

Parameters

id	The instance *id* returned from RM_Create.

Return values

The	number of equilibrium phases in the initial-phreeqc module.

See also

RM_FindComponents, RM_GetEquilibriumPhasesName.

Fortran Example:

MPI:

Called by root.

3.3.2.19 rm_getequilibriumphasesname()

```
integer function phreeqcrm::rm_getequilibriumphasesname (
             integer, intent(in) id,
             integer, intent(in) num,
             character(len=*), intent(inout) name )
```

Retrieves an item from the equilibrium phase list. The list includes all phases included in any EQUILIBRIUM_PHASES definitions in the initial-phreeqc module. RM_FindComponents must be called before RM_GetEquilibriumPhasesName. This method may be useful when generating selected output definitions related to equilibrium phases.

Parameters

id	The instance *id* returned from RM_Create.
num	The number of the equilibrium phase name to be retrieved. Fortran, 1 based.
name	The equilibrium phase name at number *num*.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_FindComponents, RM_GetEquilibriumPhasesCount.

Fortran Example:

MPI:

Called by root.

3.3.2.20 rm_geterrorstring()

```
integer function phreeqcrm::rm_geterrorstring (
             integer, intent(in) id,
             character(len=*), intent(out) errstr )
```

Returns a string containing error messages related to the last call to a PhreeqcRM method to the character argument (*errstr*).

Parameters

id	The instance *id* returned from RM_Create.
errstr	The error string related to the last call to a PhreeqcRM method.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.21 rm_geterrorstringlength()

```
integer function phreeqcrm::rm_geterrorstringlength (
             integer, intent(in) id )
```

Returns the length of the string that contains error messages related to the last call to a PhreeqcRM method.

Parameters

id	The instance *id* returned from RM_Create.

Return values

int	Length of the error message string.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.22 rm_getexchangename()

```
integer function phreeqcrm::rm_getexchangename (
             integer, intent(in) id,
             integer, intent(in) num,
             character(len=*), intent(inout) name )
```

Retrieves an item from the exchange name list. RM_FindComponents must be called before RM_GetExchange↩ Name. The exchange names vector is the same length as the exchange species names vector and provides the corresponding exchange site (for example, X corresponing to NaX). This method may be useful when generating selected output definitions related to exchangers.

Parameters

id	The instance *id* returned from RM_Create.
num	The number of the exchange name to be retrieved. Fortran, 1 based.
name	The exchange name associated with exchange species *num*.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_FindComponents, RM_GetExchangeSpeciesCount, RM_GetExchangeSpeciesName.

Fortran Example:

MPI:

Called by root.

3.3.2.23 rm_getexchangespeciescount()

```
integer function phreeqcrm::rm_getexchangespeciescount (
             integer, intent(in) id )
```

Returns the number of exchange species in the initial-phreeqc module. RM_FindComponents must be called before RM_GetExchangeSpeciesCount. This method may be useful when generating selected output definitions related to exchangers.

Parameters

id	The instance *id* returned from RM_Create.

Return values

The	number of exchange species in the initial-phreeqc module.

See also

RM_FindComponents, RM_GetExchangeSpeciesName, RM_GetExchangeName.

Fortran Example:

MPI:

Called by root.

3.3.2.24 rm_getexchangespeciesname()

```
integer function phreeqcrm::rm_getexchangespeciesname (
             integer, intent(in) id,
             integer, intent(in) num,
             character(len=*), intent(inout) name )
```

Retrieves an item from the exchange species list. The list of exchange species (such as "NaX") is derived from the list of components (RM_FindComponents) and the list of all exchange names (such as "X") that are included in EXCHANGE definitions in the initial-phreeqc module. RM_FindComponents must be called before RM_Get↩ ExchangeSpeciesName. This method may be useful when generating selected output definitions related to exchangers.

Parameters

id	The instance *id* returned from RM_Create.
num	The number of the exchange species to be retrieved. Fortran, 1 based.
name	The exchange species name at number *num*.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_FindComponents, RM_GetExchangeSpeciesCount, RM_GetExchangeName.

Fortran Example:

MPI:

Called by root.

3.3.2.25 rm_getfileprefix()

```
integer function phreeqcrm::rm_getfileprefix (
             integer, intent(in) id,
             character(len=*), intent(inout) prefix )
```

Returns the reaction-module file prefix to the character argument (*prefix*).

Parameters

id	The instance *id* returned from RM_Create.
prefix	Character string where the prefix is written.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_SetFilePrefix.

Fortran Example:

MPI:

Called by root and (or) workers.

3.3.2.26 rm_getgascompmoles()

```
integer function phreeqcrm::rm_getgascompmoles (
             integer, intent(in) id,
             double precision, dimension(:,:), intent(out), target gas_moles )
```

Transfer moles of gas components from each reaction cell to the array given in the argument list (*gas_moles*).

Parameters

id	The instance *id* returned from RM_Create.
gas_moles	Array to receive the moles of gas components for each cell. Dimension of the array is (*nxyz*, *ngas_comps*), where *nxyz* is the number of user grid cells and *ngas_comps* is the result of RM_GetGasComponentsCount. If a gas component is not defined for a cell, the number of moles is set to -1. Values for inactive cells are set to 1e30.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_FindComponents, RM_GetGasComponentsCount, RM_GetGasCompPressures, RM_GetGasCompPhi, RM_GetGasPhaseVolume, RM_SetGasCompMoles, RM_SetGasPhaseVolume.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.27 rm_getgascomponentscount()

```
integer function phreeqcrm::rm_getgascomponentscount (
             integer, intent(in) id )
```

Returns the number of gas phase components in the initial-phreeqc module. RM_FindComponents must be called before RM_GetGasComponentsCount. This method may be useful when generating selected output definitions related to gas phases.

Parameters

id	The instance *id* returned from RM_Create.

Return values

The	number of gas phase components in the initial-phreeqc module.

See also

RM_FindComponents, RM_GetGasComponentsName.

Fortran Example:

MPI:

Called by root.

3.3.2.28 rm_getgascomponentsname()

```
integer function phreeqcrm::rm_getgascomponentsname (
             integer, intent(in) id,
             integer, intent(in) num,
             character(len=*), intent(inout) name )
```

Retrieves an item from the gas components list. The list includes all gas components included in any GAS_PHASE definitions in the initial-phreeqc module. RM_FindComponents must be called before RM_GetGasComponents↩ Name. This method may be useful when generating selected output definitions related to gas phases.

Parameters

id	The instance *id* returned from RM_Create.
num	The number of the gas component name to be retrieved. Fortran, 1 based.
name	The gas component name at number *num*.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_FindComponents, RM_GetGasComponentsCount.

Fortran Example:

MPI:

Called by root.

3.3.2.29 rm_getgascompphi()

```
integer function phreeqcrm::rm_getgascompphi (
            integer, intent(in) id,
            double precision, dimension(:,:), intent(out), target gas_phi )
```

Transfer fugacity coefficients (phi) of gas components from each reaction cell to the array given in the argument list (*gas_phi*). Fugacity of a gas component is equal to the pressure of the component times the fugacity coefficient.

Parameters

id	The instance *id* returned from RM_Create.
gas_phi	Array to receive the fugacity coefficients of gas components for each cell. Dimension of the array is (*nxyz*, *ngas_comps*), where *nxyz* is the number of user grid cells and *ngas_comps* is the result of RM_GetGasComponentsCount. If a gas component is not defined for a cell, the fugacity coefficient is set to -1. Values for inactive cells are set to 1e30.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_FindComponents, RM_GetGasComponentsCount, RM_GetGasCompMoles, RM_GetGasCompPressures, RM_GetGasPhaseVolume, RM_SetGasCompMoles, RM_SetGasPhaseVolume.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.30 rm_getgascomppressures()

```
integer function phreeqcrm::rm_getgascomppressures (
             integer, intent(in) id,
             double precision, dimension(:,:), intent(out), target gas_p )
```

Transfer pressures of gas components from each reaction cell to the array given in the argument list (*gas_p*).

Parameters

id	The instance *id* returned from RM_Create.
gas_p	Array to receive the moles of gas components for each cell. Dimension of the array is (*nxyz*, *ngas_comps*), where *nxyz* is the number of user grid cells and *ngas_comps* is the result of RM_GetGasComponentsCount. If a gas component is not defined for a cell, the pressure is set to -1. Values for inactive cells are set to 1e30. Values for inactive cells are set to 1e30.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_FindComponents, RM_GetGasComponentsCount, RM_GetGasCompMoles, RM_GetGasCompPhi, RM_GetGasPhaseVolume, RM_SetGasCompMoles, RM_SetGasPhaseVolume.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.31 rm_getgasphasevolume()

```
integer function phreeqcrm::rm_getgasphasevolume (
             integer, intent(in) id,
             double precision, dimension(:), intent(out), target gas_volume )
```

Transfer volume of gas from each reaction cell to the vector given in the argument list (*gas_volume*).

Parameters

id	The instance *id* returned from RM_Create.
gas_volume	Array to receive the gas phase volumes. Dimension of the array must be *nxyz*, where *nxyz* is the number of user grid cells (RM_GetGridCellCount). If a gas phase is not defined for a cell, the volume is set to -1. Values for inactive cells are set to 1e30.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_FindComponents, RM_GetGasComponentsCount, RM_GetGasCompMoles, RM_GetGasCompPhi, RM_GetGasCompPressures, RM_SetGasCompMoles RM_SetGasPhaseVolume.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.32 rm_getgfw()

```
integer function phreeqcrm::rm_getgfw (
             integer, intent(in) id,
             double precision, dimension(:), intent(out) gfw )
```

Returns the gram formula weights (g/mol) for the components in the reaction-module component list.

Parameters

id	The instance id returned from RM_Create.
gfw	Array to receive the gram formula weights. Dimension of the array is *ncomps*, where *ncomps* is the number of components in the component list.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_FindComponents, RM_GetComponent, RM_GetComponentCount.

Fortran Example:

MPI:

Called by root.

3.3.2.33 rm_getgridcellcount()

```
integer function phreeqcrm::rm_getgridcellcount (
             integer, intent(in) id )
```

Returns the number of grid cells in the user's model, which is defined in the call to RM_Create. The mapping from grid cells to reaction cells is defined by RM_CreateMapping. The number of reaction cells may be less than the number of grid cells if there are inactive regions or symmetry in the model definition.

Parameters

id	The instance *id* returned from RM_Create.

Return values

Number	of grid cells in the user's model, negative is failure (See RM_DecodeError).

See also

RM_Create, RM_CreateMapping.

Fortran Example:

MPI:

Called by root and (or) workers.

3.3.2.34 rm_getiphreeqcid()

```
integer function phreeqcrm::rm_getiphreeqcid (
             integer, intent(in) id,
             integer, intent(in) i )
```

Returns an IPhreeqc id for the *ith* IPhreeqc instance in the reaction module. For the threaded version, there are *nthreads* + 2 IPhreeqc instances, where *nthreads* is defined in the constructor (RM_Create). The number of threads can be determined by RM_GetThreadCount. The first *nthreads* (0 based) instances will be the workers, the next (*nthreads*) is the InitialPhreeqc instance, and the next (*nthreads* + 1) is the Utility instance. Getting the IPhreeqc pointer for one of these instances allows the user to use any of the IPhreeqc methods on that instance. For MPI, each process has exactly three IPhreeqc instances, one worker (number 0), one InitialPhreeqc instance (number 1), and one Utility instance (number 2).

Parameters

id	The instance *id* returned from RM_Create.
i	The number of the IPhreeqc instance to be retrieved (0 based).

Return values

IPhreeqc	id for the *ith* IPhreeqc instance, negative is failure (See RM_DecodeError).

See also

RM_Create, RM_GetThreadCount. See IPhreeqc documentation for descriptions of IPhreeqc methods.

Fortran Example:

MPI:

Called by root and (or) workers.

3.3.2.35 rm_getkineticreactionscount()

```
integer function phreeqcrm::rm_getkineticreactionscount (
             integer, intent(in) id )
```

Returns the number of kinetic reactions in the initial-phreeqc module. RM_FindComponents must be called before RM_GetKineticReactionsCount. This method may be useful when generating selected output definitions related to kinetic reactions.

Parameters

id	The instance *id* returned from RM_Create.

Return values

The	number of kinetic reactions in the initial-phreeqc module.

See also

RM_FindComponents, RM_GetKineticReactionsName.

Fortran Example:

MPI:

Called by root.

3.3.2.36 rm_getkineticreactionsname()

```
integer function phreeqcrm::rm_getkineticreactionsname (
             integer, intent(in) id,
             integer, intent(in) num,
             character(len=*), intent(inout) name )
```

Retrieves an item from the kinetic reactions list. The list includes all kinetic reactions included in any KINETICS definitions in the initial-phreeqc module. RM_FindComponents must be called before RM_GetKineticReactions↩ Name. This method may be useful when generating selected output definitions related to kinetic reactions.

Parameters

id	The instance *id* returned from RM_Create.
num	The number of the kinetic reaction name to be retrieved. Fortran, 1 based.
name	The kinetic reaction name at number *num*.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_FindComponents, RM_GetKineticReactionsCount.

Fortran Example:

MPI:

Called by root.

3.3.2.37 rm_getmpimyself()

```
integer function phreeqcrm::rm_getmpimyself (
             integer, intent(in) id )
```

Returns the MPI task number. For the OPENMP version, the task number is always zero and the result of RM_↩ GetMpiTasks is one. For the MPI version, the root task number is zero, and all workers have a task number greater than zero. The number of tasks can be obtained with RM_GetMpiTasks. The number of tasks and computer hosts are determined at run time by the mpiexec command, and the number of reaction-module processes is defined by the communicator used in constructing the reaction modules (RM_Create).

Parameters

id	The instance *id* returned from RM_Create.

Return values

The	MPI task number for a process, negative is failure (See RM_DecodeError).

See also

RM_GetMpiTasks.

Fortran Example:

MPI:

Called by root and (or) workers.

3.3.2.38 rm_getmpitasks()

```
integer function phreeqcrm::rm_getmpitasks (
             integer, intent(in) id )
```

Returns the number of MPI processes (tasks) assigned to the reaction module. For the OPENMP version, the number of tasks is always one (although there may be multiple threads, RM_GetThreadCount), and the task number returned by RM_GetMpiMyself is zero. For the MPI version, the number of tasks and computer hosts are determined at run time by the mpiexec command. An MPI communicator is used in constructing reaction modules for MPI. The communicator may define a subset of the total number of MPI processes. The root task number is zero, and all workers have a task number greater than zero.

Parameters

id	The instance *id* returned from RM_Create.

Return values

The	number of MPI processes assigned to the reaction module, negative is failure (See RM_DecodeError).

See also

RM_GetMpiMyself, RM_Create.

Fortran Example:

MPI:

Called by root and (or) workers.

3.3.2.39 rm_getnthselectedoutputusernumber()

```
integer function phreeqcrm::rm_getnthselectedoutputusernumber (
             integer, intent(in) id,
             integer, intent(in) n )
```

Returns the user number for the *nth* selected-output definition. Definitions are sorted by user number. Phreeqc allows multiple selected-output definitions, each of which is assigned a nonnegative integer identifier by the user. The number of definitions can be obtained by RM_GetSelectedOutputCount. To cycle through all of the definitions, RM_GetNthSelectedOutputUserNumber can be used to identify the user number for each selected-output definition in sequence. RM_SetCurrentSelectedOutputUserNumber is then used to select that user number for selected-output processing.

Parameters

id	The instance *id* returned from RM_Create.
n	The sequence number of the selected-output definition for which the user number will be returned. Fortran, 1 based.

Return values

The	user number of the *nth* selected-output definition, negative is failure (See RM_DecodeError).

See also

RM_GetSelectedOutput, RM_GetSelectedOutputColumnCount, RM_GetSelectedOutputCount, RM_Get↩SelectedOutputHeading, RM_GetSelectedOutputRowCount, RM_SetCurrentSelectedOutputUserNumber, RM_SetSelectedOutputOn.

Fortran Example:

MPI:

Called by root.

3.3.2.40 rm_getsaturation()

```
integer function phreeqcrm::rm_getsaturation (
            integer, intent(in) id,
            double precision, dimension(:), intent(out) sat_calc )
```

Returns a vector of saturations (*sat_calc*) as calculated by the reaction module. Reactions will change the volume of solution in a cell. The transport code must decide whether to ignore or account for this change in solution volume due to reactions. Following reactions, the cell saturation is calculated as solution volume (RM_GetSolution↩Volume) divided by the product of representative volume (RM_SetRepresentativeVolume) and the porosity (RM↩_SetPorosity). The cell saturation returned by *RM_GetSaturation* may be less than or greater than the saturation set by the transport code (RM_SetSaturation), and may be greater than or less than 1.0, even in fully saturated simulations. Only the following databases distributed with PhreeqcRM have molar volume information needed to accurately calculate solution volume and saturation: phreeqc.dat, Amm.dat, and pitzer.dat.

Parameters

id	The instance *id* returned from RM_Create.
sat_calc	Vector to receive the saturations. Dimension of the array is set to *nxyz*, where *nxyz* is the number of user grid cells (RM_GetGridCellCount). Values for inactive cells are set to 1e30.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_GetSolutionVolume, RM_SetPorosity, RM_SetRepresentativeVolume, RM_SetSaturation.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.41 rm_getselectedoutput()

```
integer function phreeqcrm::rm_getselectedoutput (
             integer, intent(in) id,
             double precision, dimension(:,:), intent(out) so )
```

Populates an array with values from the current selected-output definition. RM_SetCurrentSelectedOutputUser↩ Number determines which of the selected-output definitions is used to populate the array.

Parameters

id	The instance *id* returned from RM_Create.
so	An array to contain the selected-output values. Size of the array is (*nxyz*, *col*), where *nxyz* is the number of grid cells in the user's model (RM_GetGridCellCount), and *col* is the number of columns in the selected-output definition (RM_GetSelectedOutputColumnCount).

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_GetNthSelectedOutputUserNumber, RM_GetSelectedOutputColumnCount, RM_GetSelectedOutput↩ Count, RM_GetSelectedOutputHeading, RM_GetSelectedOutputRowCount, RM_SetCurrentSelected↩ OutputUserNumber, RM_SetSelectedOutputOn.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.42 rm_getselectedoutputcolumncount()

```
integer function phreeqcrm::rm_getselectedoutputcolumncount (
             integer, intent(in) id )
```

Returns the number of columns in the current selected-output definition. RM_SetCurrentSelectedOutputUser↩ Number determines which of the selected-output definitions is used.

Parameters

id	The instance *id* returned from RM_Create.

Return values

Number	of columns in the current selected-output definition, negative is failure (See RM_DecodeError).

See also

RM_GetNthSelectedOutputUserNumber, RM_GetSelectedOutput, RM_GetSelectedOutputCount, RM_Get↩ SelectedOutputHeading, RM_GetSelectedOutputRowCount, RM_SetCurrentSelectedOutputUserNumber, RM_SetSelectedOutputOn.

Fortran Example:

MPI:

Called by root.

3.3.2.43 rm_getselectedoutputcount()

```
integer function phreeqcrm::rm_getselectedoutputcount (
             integer, intent(in) id )
```

Returns the number of selected-output definitions. RM_SetCurrentSelectedOutputUserNumber determines which of the selected-output definitions is used.

Parameters

id	The instance *id* returned from RM_Create.

Return values

Number	of selected-output definitions, negative is failure (See RM_DecodeError).

See also

RM_GetNthSelectedOutputUserNumber, RM_GetSelectedOutput, RM_GetSelectedOutputColumnCount, RM_GetSelectedOutputHeading, RM_GetSelectedOutputRowCount, RM_SetCurrentSelectedOutputUser↩ Number, RM_SetSelectedOutputOn.

Fortran Example:

MPI:

Called by root.

3.3.2.44 rm_getselectedoutputheading()

```
integer function phreeqcrm::rm_getselectedoutputheading (
             integer, intent(in) id,
             integer, intent(in) icol,
             character(len=*), intent(out) heading )
```

Returns a selected output heading. The number of headings is determined by RM_GetSelectedOutputColumn↩ Count. RM_SetCurrentSelectedOutputUserNumber determines which of the selected-output definitions is used.

Parameters

id	The instance *id* returned from RM_Create.
icol	The sequence number of the heading to be retrieved. Fortran, 1 based.
heading	A string buffer to receive the heading.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_GetNthSelectedOutputUserNumber, RM_GetSelectedOutput, RM_GetSelectedOutputColumnCount, RM_GetSelectedOutputCount, RM_GetSelectedOutputRowCount, RM_SetCurrentSelectedOutputUser↩ Number, RM_SetSelectedOutputOn.

Fortran Example:

MPI:

Called by root.

3.3.2.45 rm_getselectedoutputrowcount()

```
integer function phreeqcrm::rm_getselectedoutputrowcount (
             integer, intent(in) id )
```

Returns the number of rows in the current selected-output definition. However, the method is included only for convenience; the number of rows is always equal to the number of grid cells in the user's model, and is equal to RM_GetGridCellCount.

Parameters

id	The instance *id* returned from RM_Create.

Return values

Number	of rows in the current selected-output definition, negative is failure (See RM_DecodeError).

See also

RM_GetNthSelectedOutputUserNumber, RM_GetSelectedOutput, RM_GetSelectedOutputColumnCount, RM_GetSelectedOutputCount, RM_GetSelectedOutputHeading, RM_SetCurrentSelectedOutputUserNumber, RM_SetSelectedOutputOn.

Fortran Example:

MPI:

Called by root.

3.3.2.46 rm_getsicount()

```
integer function phreeqcrm::rm_getsicount (
             integer, intent(in) id )
```

Returns the number of phases in the initial-phreeqc module for which saturation indices can be calculated. RM_FindComponents must be called before RM_GetSICount. This method may be useful when generating selected output definitions related to saturation indices.

Parameters

id	The instance *id* returned from RM_Create.

Return values

The	number of phases in the initial-phreeqc module for which saturation indices could be calculated.

See also

RM_FindComponents, RM_GetSIName.

Fortran Example:

MPI:

Called by root.

3.3.2.47 rm_getsiname()

```
integer function phreeqcrm::rm_getsiname (
             integer, intent(in) id,
             integer, intent(in) num,
             character(len=*), intent(inout) name )
```

Retrieves an item from the list of all phases for which saturation indices can be calculated. The list includes all phases that contain only elements included in the components in the initial-phreeqc module. The list assumes that all components are present to be able to calculate the entire list of SIs; it may be that one or more components are missing in any specific cell. RM_FindComponents must be called before RM_GetSIName. This method may be useful when generating selected output definitions related to saturation indices.

Parameters

id	The instance *id* returned from RM_Create.
num	The number of the saturation-index-phase name to be retrieved. Fortran, 1 based.
name	The saturation-index-phase name at number *num*.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_FindComponents, RM_GetSICount.

Fortran Example:

MPI:

Called by root.

3.3.2.48 rm_getsolidsolutioncomponentscount()

```
integer function phreeqcrm::rm_getsolidsolutioncomponentscount (
             integer, intent(in) id )
```

Returns the number of solid solution components in the initial-phreeqc module. RM_FindComponents must be called before RM_GetSolidSolutionComponentsCount. This method may be useful when generating selected output definitions related to solid solutions.

Parameters

id	The instance *id* returned from RM_Create.

Return values

The	number of solid solution components in the initial-phreeqc module.

See also

RM_FindComponents, RM_GetSolidSolutionComponentsName, RM_GetSolidSolutionName.

Fortran Example:

MPI:

Called by root.

3.3.2.49 rm_getsolidsolutioncomponentsname()

```
integer function phreeqcrm::rm_getsolidsolutioncomponentsname (
             integer, intent(in) id,
             integer, intent(in) num,
             character(len=*), intent(inout) name )
```

Retrieves an item from the solid solution components list. The list includes all solid solution components included in any SOLID_SOLUTIONS definitions in the initial-phreeqc module. RM_FindComponents must be called before RM_GetSolidSolutionComponentsName. This method may be useful when generating selected output definitions related to solid solutions.

Parameters

id	The instance *id* returned from RM_Create.
num	The number of the solid solution components name to be retrieved. Fortran, 1 based.
name	The solid solution compnent name at number *num*.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_FindComponents, RM_GetSolidSolutionComponentsCount, RM_GetSolidSolutionName.

Fortran Example:

MPI:

Called by root.

3.3.2.50 rm_getsolidsolutionname()

```
integer function phreeqcrm::rm_getsolidsolutionname (
             integer, intent(in) id,
             integer, intent(in) num,
             character(len=*), intent(inout) name )
```

Retrieves an item from the solid solution names list. The list includes solid solution names included in SOLID_SOLUTIONS definitions in the initial-phreeqc module. The solid solution names vector is the same length as the solid solution components vector and provides the corresponding name of solid solution containing the component. RM_FindComponents must be called before RM_GetSolidSolutionName. This method may be useful when generating selected output definitions related to solid solutions.

Parameters

id	The instance *id* returned from RM_Create.
num	The number of the solid solution name to be retrieved. Fortran, 1 based.
name	The solid solution name at number *num*.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_FindComponents, RM_GetSolidSolutionComponentsCount, RM_GetSolidSolutionComponentsName.

Fortran Example:

MPI:

Called by root.

3.3.2.51 rm_getsolutionvolume()

```
integer function phreeqcrm::rm_getsolutionvolume (
             integer, intent(in) id,
             double precision, dimension(:), intent(out) vol )
```

Transfer solution volumes from the reaction cells to the array given in the argument list (*vol*). Solution volumes are those calculated by the reaction module. Only the following databases distributed with PhreeqcRM have molar volume information needed to accurately calculate solution volume: phreeqc.dat, Amm.dat, and pitzer.dat.

Parameters

id	The instance *id* returned from RM_Create.
vol	Array to receive the solution volumes. Dimension of the array is (*nxyz*), where *nxyz* is the number of user grid cells. Values for inactive cells are set to 1e30.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_GetSaturation.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.52 rm_getspeciesconcentrations()

```
integer function phreeqcrm::rm_getspeciesconcentrations (
             integer, intent(in) id,
             double precision, dimension(:,:), intent(out) species_conc )
```

Transfer concentrations of aqueous species to the array argument (*species_conc*) This method is intended for use with multicomponent-diffusion transport calculations, and RM_SetSpeciesSaveOn must be set to *true*. The list of aqueous species is determined by RM_FindComponents and includes all aqueous species that can be made from the set of components. Solution volumes used to calculate mol/L are calculated by the reaction module. Only the following databases distributed with PhreeqcRM have molar volume information needed to accurately calculate solution volume: phreeqc.dat, Amm.dat, and pitzer.dat.

Parameters

id	The instance *id* returned from RM_Create.
species_conc	Array to receive the aqueous species concentrations. Dimension of the array is (*nxyz*, *nspecies*), where *nxyz* is the number of user grid cells (RM_GetGridCellCount), and *nspecies* is the number of aqueous species (RM_GetSpeciesCount). Concentrations are moles per liter. Values for inactive cells are set to 1e30.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_FindComponents, RM_GetSpeciesCount, RM_GetSpeciesD25, RM_GetSpeciesLog10Gammas, RM↩ _GetSpeciesLog10Molalities, RM_GetSpeciesName, RM_GetSpeciesSaveOn, RM_GetSpeciesZ, RM_SetSpeciesSaveOn, RM_SpeciesConcentrations2Module.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.53 rm_getspeciescount()

```
integer function phreeqcrm::rm_getspeciescount (
             integer, intent(in) id )
```

The number of aqueous species used in the reaction module. This method is intended for use with multicomponent-diffusion transport calculations, and RM_SetSpeciesSaveOn must be set to *true*. The list of aqueous species is determined by RM_FindComponents and includes all aqueous species that can be made from the set of components.

Parameters

id	The instance *id* returned from RM_Create.

Return values

IRM_RESULT	The number of aqueous species, negative is failure (See RM_DecodeError).

See also

RM_FindComponents, RM_GetSpeciesConcentrations, RM_GetSpeciesD25, RM_GetSpeciesLog10↩ Gammas, RM_GetSpeciesLog10Molalities, RM_GetSpeciesName, RM_GetSpeciesSaveOn, RM_Get↩ SpeciesZ, RM_SetSpeciesSaveOn, RM_SpeciesConcentrations2Module.

Fortran Example:

MPI:

Called by root and (or) workers.

3.3.2.54 rm_getspeciesd25()

```
integer function phreeqcrm::rm_getspeciesd25 (
             integer, intent(in) id,
             double precision, dimension(:), intent(out) diffc )
```

Transfers diffusion coefficients at 25C to the array argument (*diffc*). This method is intended for use with multicomponent-diffusion transport calculations, and RM_SetSpeciesSaveOn must be set to *true*. Diffusion coefficients are defined in SOLUTION_SPECIES data blocks, normally in the database file. Databases distributed with the reaction module that have diffusion coefficients defined are phreeqc.dat, Amm.dat, and pitzer.dat.

Parameters

id	The instance *id* returned from RM_Create.
diffc	Array to receive the diffusion coefficients at 25 C, m^2/s. Dimension of the array is *nspecies*, where *nspecies* is is the number of aqueous species (RM_GetSpeciesCount).

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_FindComponents, RM_GetSpeciesConcentrations, RM_GetSpeciesCount, RM_GetSpeciesLog10↩ Gammas, RM_GetSpeciesLog10Molalities, RM_GetSpeciesName, RM_GetSpeciesSaveOn, RM_Get↩ SpeciesZ,
RM_SetSpeciesSaveOn, RM_SpeciesConcentrations2Module.

Fortran Example:

MPI:

Called by root and (or) workers.

3.3.2.55 rm_getspecieslog10gammas()

```
integer function phreeqcrm::rm_getspecieslog10gammas (
             integer, intent(in) id,
             double precision, dimension(:,:), intent(out) species_log10gammas )
```

Transfer log10 aqueous-species activity coefficients to the array argument (*species_log10gammas*) This method is intended for use with multicomponent-diffusion transport calculations, and RM_SetSpeciesSaveOn must be set to *true*. The list of aqueous species is determined by RM_FindComponents and includes all aqueous species that can be made from the set of components.

Parameters

id	The instance *id* returned from RM_Create.
species_log10gammas	Array to receive the aqueous species concentrations. Dimension of the array is (*nxyz*, *nspecies*), where *nxyz* is the number of user grid cells (RM_GetGridCellCount), and *nspecies* is the number of aqueous species (RM_GetSpeciesCount). Values for inactive cells are set to 1e30.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_FindComponents, RM_GetSpeciesConcentrations, RM_GetSpeciesCount, RM_GetSpeciesD25, RM_↩ GetSpeciesLog10Molalities, RM_GetSpeciesName, RM_GetSpeciesSaveOn, RM_GetSpeciesZ,
RM_SetSpeciesSaveOn, RM_SpeciesConcentrations2Module.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.56 rm_getspecieslog10molalities()

```
integer function phreeqcrm::rm_getspecieslog10molalities (
             integer, intent(in) id,
             double precision, dimension(:,:), intent(out) species_log10molalities )
```

Transfer log10 aqueous-species log10 molalities to the array argument (*species_log10molalities*) To use this method RM_SetSpeciesSaveOn must be set to *true*. The list of aqueous species is determined by RM_FindComponents and includes all aqueous species that can be made from the set of components.

Parameters

id	The instance *id* returned from RM_Create.
species_log10molalities	Array to receive the aqueous species molalities. Dimension of the array is (*nxyz*, *nspecies*), where *nxyz* is the number of user grid cells (RM_GetGridCellCount), and *nspecies* is the number of aqueous species (RM_GetSpeciesCount). Values for inactive cells are set to 1e30.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_FindComponents, RM_GetSpeciesConcentrations, RM_GetSpeciesCount, RM_GetSpeciesD25, RM_GetSpeciesLog10Gammas, RM_GetSpeciesName, RM_GetSpeciesSaveOn, RM_GetSpeciesZ, RM_SetSpeciesSaveOn, RM_SpeciesConcentrations2Module.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.57 rm_getspeciesname()

```
integer function phreeqcrm::rm_getspeciesname (
             integer, intent(in) id,
```

```
integer, intent(in) i,
character(len=*), intent(out) name )
```

Transfers the name of the *ith* aqueous species to the character argument (*name*). This method is intended for use with multicomponent-diffusion transport calculations, and RM_SetSpeciesSaveOn must be set to *true*. The list of aqueous species is determined by RM_FindComponents and includes all aqueous species that can be made from the set of components.

Parameters

id	The instance *id* returned from RM_Create.
i	Sequence number of the species in the species list. Fortran, 1 based.
name	Character array to receive the species name.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_FindComponents, RM_GetSpeciesConcentrations, RM_GetSpeciesCount, RM_GetSpeciesD25, RM_GetSpeciesLog10Gammas, RM_GetSpeciesLog10Molalities, RM_GetSpeciesSaveOn, RM_GetSpeciesZ, RM_SetSpeciesSaveOn, RM_SpeciesConcentrations2Module.

Fortran Example:

MPI:

Called by root and (or) workers.

3.3.2.58 rm_getspeciessaveon()

```
integer function phreeqcrm::rm_getspeciessaveon (
             integer, intent(in) id )
```

Returns the value of the species-save property. By default, concentrations of aqueous species are not saved. Setting the species-save property to true allows aqueous species concentrations to be retrieved with RM_GetSpeciesConcentrations, and solution compositions to be set with RM_SpeciesConcentrations2Module.

Parameters

id	The instance *id* returned from RM_Create.

Return values

IRM_RESULT	0, species are not saved; 1, species are saved; negative is failure (See RM_DecodeError).

See also

RM_FindComponents, RM_GetSpeciesConcentrations, RM_GetSpeciesCount, RM_GetSpeciesD25, RM_↩ GetSpeciesLog10Gammas, RM_GetSpeciesLog10Molalities, RM_GetSpeciesName, RM_GetSpeciesZ, R↩ M_SetSpeciesSaveOn, RM_SpeciesConcentrations2Module.

Fortran Example:

MPI:

Called by root and (or) workers.

3.3.2.59 rm_getspeciesz()

```
integer function phreeqcrm::rm_getspeciesz (
            integer, intent(in) id,
            double precision, dimension(:), intent(out) z )
```

Transfers the charge of each aqueous species to the array argument (*z*). This method is intended for use with multicomponent-diffusion transport calculations, and RM_SetSpeciesSaveOn must be set to *true*.

Parameters

id	The instance *id* returned from RM_Create.
z	Array that receives the charge for each aqueous species. Dimension of the array is *nspecies*, where *nspecies* is is the number of aqueous species (RM_GetSpeciesCount).

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_FindComponents, RM_GetSpeciesConcentrations, RM_GetSpeciesCount, RM_GetSpeciesD25, R↩ M_GetSpeciesLog10Gammas, RM_GetSpeciesLog10Molalities, RM_GetSpeciesName, RM_GetSpecies↩ SaveOn, RM_SetSpeciesSaveOn, RM_SpeciesConcentrations2Module.

Fortran Example:

MPI:

Called by root and (or) workers.

3.3.2.60 rm_getstartcell()

```
integer function phreeqcrm::rm_getstartcell (
             integer, intent(in) id,
             integer, dimension(:), intent(out) sc )
```

Returns an array with the starting cell numbers from the range of cell numbers assigned to each worker.

Parameters

id	The instance *id* returned from RM_Create.
sc	Array to receive the starting cell numbers. Dimension of the array is the number of threads (OpenMP) or the number of processes (MPI).

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_Create, RM_GetEndCell, RM_GetMpiTasks, RM_GetThreadCount.

Fortran Example:

MPI:

Called by root and (or) workers.

3.3.2.61 rm_getsurfacename()

```
integer function phreeqcrm::rm_getsurfacename (
             integer, intent(in) id,
             integer, intent(in) num,
             character(len=*), intent(inout) name )
```

Retrieves the surface name (such as "Hfo") that corresponds with the surface species name. The lists of surface species names and surface names are the same length. RM_FindComponents must be called before RM_Get↩ SurfaceName. This method may be useful when generating selected output definitions related to surfaces.

Parameters

id	The instance *id* returned from RM_Create.
num	The number of the surface name to be retrieved. Fortran, 1 based.
name	The surface name associated with surface species *num*.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_FindComponents, RM_GetSurfaceSpeciesCount, RM_GetSurfaceSpeciesName, RM_GetSurfaceType.

Fortran Example:

MPI:

Called by root.

3.3.2.62 rm_getsurfacespeciescount()

```
integer function phreeqcrm::rm_getsurfacespeciescount (
            integer, intent(in) id )
```

Returns the number of surface species (such as "Hfo_wOH") in the initial-phreeqc module. RM_FindComponents must be called before RM_GetSurfaceSpeciesCount. This method may be useful when generating selected output definitions related to surfaces.

Parameters

id	The instance *id* returned from RM_Create.

Return values

The	number of surface species in the initial-phreeqc module.

See also

RM_FindComponents, RM_GetSurfaceSpeciesName, RM_GetSurfaceType, RM_GetSurfaceName.

Fortran Example:

MPI:

Called by root.

3.3.2.63 rm_getsurfacespeciesname()

```
integer function phreeqcrm::rm_getsurfacespeciesname (
             integer, intent(in) id,
             integer, intent(in) num,
             character(len=*), intent(inout) name )
```

Retrieves an item from the surface species list. The list of surface species (for example, "Hfo_wOH") is derived from the list of components (RM_FindComponents) and the list of all surface types (such as "Hfo_w") that are included in SURFACE definitions in the initial-phreeqc module. RM_FindComponents must be called before RM_GetSurface↩ SpeciesName. This method may be useful when generating selected output definitions related to surfaces.

Parameters

id	The instance *id* returned from RM_Create.
num	The number of the surface type to be retrieved. Fortran, 1 based.
name	The surface species name at number *num*.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_FindComponents, RM_GetSurfaceSpeciesCount, RM_GetSurfaceType, RM_GetSurfaceName.

Fortran Example:

MPI:

Called by root.

3.3.2.64 rm_getsurfacetype()

```
integer function phreeqcrm::rm_getsurfacetype (
             integer, intent(in) id,
             integer, intent(in) num,
             character(len=*), intent(inout) name )
```

Retrieves the surface site type (such as "Hfo_w") that corresponds with the surface species name. The lists of surface species names and surface species types are the same length. RM_FindComponents must be called before RM_GetSurfaceType. This method may be useful when generating selected output definitions related to surfaces.

Parameters

id	The instance *id* returned from RM_Create.
num	The number of the surface type to be retrieved. Fortran, 1 based.
name	The surface type associated with surface species *num*.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_FindComponents, RM_GetSurfaceSpeciesCount, RM_GetSurfaceSpeciesName, RM_GetSurfaceName.

Fortran Example:

MPI:

Called by root.

3.3.2.65 rm_getthreadcount()

```
integer function phreeqcrm::rm_getthreadcount (
             integer, intent(in) id )
```

Returns the number of threads, which is equal to the number of workers used to run in parallel with OPENMP. For the OPENMP version, the number of threads is set implicitly or explicitly with RM_Create. For the MPI version, the number of threads is always one for each process.

Parameters

id	The instance *id* returned from RM_Create.

Return values

The	number of threads, negative is failure (See RM_DecodeError).

See also

RM_GetMpiTasks.

Fortran Example:

MPI:

Called by root and (or) workers; result is always 1.

3.3.2.66 rm_gettime()

```
double precision function phreeqcrm::rm_gettime (
             integer, intent(in) id )
```

Returns the current simulation time in seconds. The reaction module does not change the time value, so the returned value is equal to the default (0.0) or the last time set by RM_SetTime.

Parameters

id	The instance *id* returned from RM_Create.

Return values

The	current simulation time in seconds.

See also

RM_GetTimeConversion, RM_GetTimeStep, RM_SetTime, RM_SetTimeConversion, RM_SetTimeStep.

Fortran Example:

MPI:

Called by root and (or) workers.

3.3.2.67 rm_gettimeconversion()

```
double precision function phreeqcrm::rm_gettimeconversion (
             integer, intent(in) id )
```

Returns a multiplier to convert time from seconds to another unit, as specified by the user. The reaction module uses seconds as the time unit. The user can set a conversion factor (RM_SetTimeConversion) and retrieve it with RM_GetTimeConversion. The reaction module only uses the conversion factor when printing the long version of cell chemistry (RM_SetPrintChemistryOn), which is rare. Default conversion factor is 1.0.

Parameters

id	The instance *id* returned from RM_Create.

Return values

Multiplier	to convert seconds to another time unit.

See also

RM_GetTime, RM_GetTimeStep, RM_SetTime, RM_SetTimeConversion, RM_SetTimeStep.

Fortran Example:

MPI:

Called by root and (or) workers.

3.3.2.68 rm_gettimestep()

```
double precision function phreeqcrm::rm_gettimestep (
            integer, intent(in) id )
```

Returns the current simulation time step in seconds. This is the time over which kinetic reactions are integrated in a call to RM_RunCells. The reaction module does not change the time step value, so the returned value is equal to the default (0.0) or the last time step set by RM_SetTimeStep.

Parameters

id	The instance *id* returned from RM_Create.

Return values

The	current simulation time step in seconds.

See also

RM_GetTime, RM_GetTimeConversion, RM_SetTime, RM_SetTimeConversion, RM_SetTimeStep.

Fortran Example:

MPI:

Called by root and (or) workers.

3.3.2.69 rm_initialphreeqc2concentrations()

```
integer function phreeqcrm::rm_initialphreeqc2concentrations (
             integer, intent(in) id,
             double precision, dimension(:,:), intent(out) bc_conc,
             integer, intent(in) n_boundary,
             integer, dimension(:), intent(in) bc1,
             integer, dimension(:), intent(in), optional bc2,
             double precision, dimension(:), intent(in), optional f1 )
```

Fills an array (*bc_conc*) with concentrations from solutions in the InitialPhreeqc instance. The method is used to obtain concentrations for boundary conditions. If a negative value is used for a cell in *bc1*, then the highest numbered solution in the InitialPhreeqc instance will be used for that cell. Concentrations may be a mixture of two solutions, *bc1* and *bc2*, with a mixing fraction for *bc1* 1 of *f1* and mixing fraction for *bc2* of (1 - *f1*). A negative value for *bc2* implies no mixing, and the associated value for *f1* is ignored. If *bc2* and *f1* are omitted, no mixing is used; concentrations are derived from *bc1* only.

Parameters

id	The instance *id* returned from RM_Create.
bc_conc	Array of concentrations extracted from the InitialPhreeqc instance. The dimension of *bc_conc* is (*n_boundary*, *ncomp*), where *ncomp* is the number of components returned from RM_FindComponents or RM_GetComponentCount.
n_boundary	The number of boundary condition solutions that need to be filled.
bc1	Array of solution index numbers that refer to solutions in the InitialPhreeqc instance. Size is *n_boundary*.
bc2	Array of solution index numbers that that refer to solutions in the InitialPhreeqc instance and are defined to mix with *bc1*. Size is *n_boundary*. Optional in Fortran.
f1	Fraction of *bc1* that mixes with (1-*f1*) of *bc2*. Size is (n_boundary). Optional in Fortran.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_FindComponents, RM_GetComponentCount.

Fortran Example:

MPI:

Called by root.

3.3.2.70 rm_initialphreeqc2module()

```
integer function phreeqcrm::rm_initialphreeqc2module (
             integer, intent(in) id,
             integer, dimension(:,:), intent(in) ic1,
             integer, dimension(:,:), intent(in), optional ic2,
             double precision, dimension(:,:), intent(in), optional f1 )
```

Transfer solutions and reactants from the InitialPhreeqc instance to the reaction-module workers, possibly with mixing. In its simplest form, *ic1* is used to select initial conditions, including solutions and reactants, for each cell of the model, without mixing. *ic1* is dimensioned (*nxyz*, 7), where *nxyz* is the number of grid cells in the user's model (RM_GetGridCellCount). The dimension of 7 refers to solutions and reactants in the following order: (1) SOLUTIONS, (2) EQUILIBRIUM_PHASES, (3) EXCHANGE, (4) SURFACE, (5) GAS_PHASE, (6) SOLID_SOLUTIONS, and (7) KINETICS. In Fortran, ic1(100, 4) = 2, indicates that cell 99 (0 based) contains the SURFACE definition with user number 2 that has been defined in the InitialPhreeqc instance (either by RM_RunFile or RM_RunString).

It is also possible to mix solutions and reactants to obtain the initial conditions for cells. For mixing, *ic2* contains numbers for a second entity that mixes with the entity defined in *ic1*. *F1* contains the mixing fraction for *ic1*, whereas (1 - *f1*) is the mixing fraction for *ic2*. In Fortran, ic1(100, 4) = 2, initial_conditions2(100, 4) = 3, f1(100, 4) = 0.25 indicates that cell 99 (0 based) contains a mixture of 0.25 SURFACE 2 and 0.75 SURFACE 3, where the surface compositions have been defined in the InitialPhreeqc instance. If the user number in *ic2* is negative, no mixing occurs. If *ic2* and *f1* are omitted, no mixing is used, and initial conditions are derived solely from *ic1*.

Parameters

id	The instance *id* returned from RM_Create.
ic1	Array of solution and reactant index numbers that refer to definitions in the InitialPhreeqc instance. Size is (*nxyz*,7). The order of definitions is given above. Negative values are ignored, resulting in no definition of that entity for that cell.
ic2	Array of solution and reactant index numbers that refer to definitions in the InitialPhreeqc instance. Nonnegative values of *ic2* result in mixing with the entities defined in *ic1*. Negative values result in no mixing. Size is (*nxyz*,7). The order of definitions is given above. Optional in Fortran; omitting results in no mixing.
f1	Fraction of ic1 that mixes with (1-*f1*) of ic2. Size is (nxyz,7). The order of definitions is given above. Optional in Fortran; omitting results in no mixing.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_InitialPhreeqcCell2Module.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.71 rm_initialphreeqc2speciesconcentrations()

```
integer function phreeqcrm::rm_initialphreeqc2speciesconcentrations (
             integer, intent(in) id,
             double precision, dimension(:,:), intent(out) bc_conc,
             integer, intent(in) n_boundary,
             integer, dimension(:), intent(in) bc1,
             integer, dimension(:), intent(in), optional bc2,
             double precision, dimension(:), intent(in), optional f1 )
```

Fills an array (*bc_conc*) with aqueous species concentrations from solutions in the InitialPhreeqc instance. This method is intended for use with multicomponent-diffusion transport calculations, and RM_SetSpeciesSaveOn must be set to *true*. The method is used to obtain aqueous species concentrations for boundary conditions. If a negative value is used for a cell in *bc1*, then the highest numbered solution in the InitialPhreeqc instance will be used for that cell. Concentrations may be a mixture of two solutions, *bc1* and *bc2*, with a mixing fraction for *bc1* of *f1* and mixing fraction for *bc2* of (1 - *f1*). A negative value for *bc2* implies no mixing, and the associated value for *f1* is ignored. If *bc2* and *f1* are omitted, no mixing is used; concentrations are derived from *bc1* only.

Parameters

id	The instance *id* returned from RM_Create.
bc_conc	Array of aqueous concentrations extracted from the InitialPhreeqc instance. The dimension of *species_c* is (*n_boundary*, *nspecies*), where *nspecies* is the number of aqueous species returned from RM_GetSpeciesCount.
n_boundary	The number of boundary condition solutions that need to be filled.
bc1	Array of solution index numbers that refer to solutions in the InitialPhreeqc instance. Size is *n_boundary*.
bc2	Array of solution index numbers that that refer to solutions in the InitialPhreeqc instance and are defined to mix with *bc1*. Size is *n_boundary*. Optional in Fortran.
f1	Fraction of *bc1* that mixes with (1-*f1*) of *bc2*. Size is *n_boundary*. Optional in Fortran.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_FindComponents, RM_GetSpeciesCount, RM_SetSpeciesSaveOn.

Fortran Example:

MPI:

Called by root.

3.3.2.72 rm_initialphreeqccell2module()

```
integer function phreeqcrm::rm_initialphreeqccell2module (
             integer, intent(in) id,
             integer, intent(in) n_user,
             integer, dimension(:), intent(in) cell_numbers,
             integer, intent(in) n_cell )
```

A cell numbered *n_user* in the InitialPhreeqc instance is selected to populate a series of cells. All reactants with the number *n_user* are transferred along with the solution. If MIX *n_user* exists, it is used for the definition of the solution. If *n_user* is negative, *n_user* is redefined to be the largest solution or MIX number in the InitialPhreeqc instance. All reactants for each cell in the list *cell_numbers* are removed before the cell definition is copied from the InitialPhreeqc instance to the workers.

Parameters

id	The instance *id* returned from RM_Create.
n_user	Cell number refers to a solution or MIX and associated reactants in the InitialPhreeqc instance. A negative number indicates the largest solution or MIX number in the InitialPhreeqc instance will be used.
cell_numbers	A list of cell numbers in the user's grid-cell numbering system that will be populated with cell *n_user* from the InitialPhreeqc instance.
n_cell	The number of cell numbers in the *cell_numbers* list.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_InitialPhreeqc2Module.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.73 rm_loaddatabase()

```
integer function phreeqcrm::rm_loaddatabase (
             integer, intent(in) id,
             character(len=*), intent(in) db_name )
```

Load a database for all IPhreeqc instances–workers, InitialPhreeqc, and Utility. All definitions of the reaction module are cleared (SOLUTION_SPECIES, PHASES, SOLUTIONs, etc.), and the database is read.

Parameters

id	The instance *id* returned from RM_Create.
db_name	String containing the database name.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_Create.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.74 rm_logmessage()

```
integer function phreeqcrm::rm_logmessage (
             integer, intent(in) id,
             character(len=*), intent(in) str )
```

Print a message to the log file.

Parameters

id	The instance *id* returned from RM_Create.
str	String to be printed.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_ErrorMessage,
RM_OpenFiles,
RM_OutputMessage, RM_ScreenMessage, RM_WarningMessage.

Fortran Example:

MPI:

Called by root.

3.3.2.75 rm_mpiworker()

```
integer function phreeqcrm::rm_mpiworker (
             integer, intent(in) id )
```

MPI only. Workers (processes with RM_GetMpiMyself $>$ 0) must call RM_MpiWorker to be able to respond to messages from the root to accept data, perform calculations, and (or) return data. RM_MpiWorker contains a loop that reads a message from root, performs a task, and waits for another message from root. RM_SetConcentrations, RM_RunCells, and RM_GetConcentrations are examples of methods that send a message from root to get the workers to perform a task. The workers will respond to all methods that are designated "workers must be in the loop of RM_MpiWorker" in the MPI section of the method documentation. The workers will continue to respond to messages from root until root calls RM_MpiWorkerBreak.

(Advanced) The list of tasks that the workers perform can be extended by using RM_SetMpiWorkerCallback. It is then possible to use the MPI processes to perform other developer-defined tasks, such as transport calculations, without exiting from the RM_MpiWorker loop. Alternatively, root calls RM_MpiWorkerBreak to allow the workers to continue past a call to RM_MpiWorker. The workers perform developer-defined calculations, and then RM_MpiWorker is called again to respond to requests from root to perform reaction-module tasks.

Parameters

id	The instance *id* returned from RM_Create.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError). RM_MpiWorker returns a value only when RM_MpiWorkerBreak is called by root.

See also

RM_MpiWorkerBreak, RM_SetMpiWorkerCallback.

Fortran Example:

MPI:

Called by all workers.

3.3.2.76 rm_mpiworkerbreak()

```
integer function phreeqcrm::rm_mpiworkerbreak (
             integer, intent(in) id )
```

MPI only. This method is called by root to force workers (processes with RM_GetMpiMyself $>$ 0) to return from a call to RM_MpiWorker. RM_MpiWorker contains a loop that reads a message from root, performs a task, and waits for another message from root. The workers respond to all methods that are designated "workers must be in the loop of RM_MpiWorker" in the MPI section of the method documentation. The workers will continue to respond to messages from root until root calls RM_MpiWorkerBreak.

Parameters

id	The instance *id* returned from RM_Create.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_MpiWorker, RM_SetMpiWorkerCallback.

Fortran Example:

MPI:

Called by root.

3.3.2.77 rm_openfiles()

```
integer function phreeqcrm::rm_openfiles (
             integer, intent(in) id )
```

Opens the output and log files. Files are named prefix.chem.txt and prefix.log.txt based on the prefix defined by RM_SetFilePrefix.

Parameters

id	The instance *id* returned from RM_Create.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_CloseFiles, RM_ErrorMessage, RM_GetFilePrefix, RM_LogMessage, RM_OutputMessage, RM_SetFilePrefix, RM_WarningMessage.

Fortran Example:

MPI:

Called by root.

3.3.2.78 rm_outputmessage()

```
integer function phreeqcrm::rm_outputmessage (
             integer, intent(in) id,
             character(len=*), intent(in) str )
```

Print a message to the output file.

Parameters

id	The instance *id* returned from RM_Create.
str	String to be printed.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_ErrorMessage, RM_LogMessage, RM_ScreenMessage, RM_WarningMessage.

Fortran Example:

MPI:

Called by root.

3.3.2.79 rm_runcells()

```
integer function phreeqcrm::rm_runcells (
            integer, intent(in) id )
```

Runs a reaction step for all of the cells in the reaction module. Normally, tranport concentrations are transferred to the reaction cells (RM_SetConcentrations) before reaction calculations are run. The length of time over which kinetic reactions are integrated is set by RM_SetTimeStep. Other properties that may need to be updated as a result of the transport calculations include porosity (RM_SetPorosity), saturation (RM_SetSaturation), temperature (RM_SetTemperature), and pressure (RM_SetPressure).

Parameters

id	The instance *id* returned from RM_Create.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_SetConcentrations,
RM_SetPorosity, RM_SetPressure, RM_SetSaturation, RM_SetTemperature, RM_SetTimeStep.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.80 rm_runfile()

```
integer function phreeqcrm::rm_runfile (
            integer, intent(in) id,
            integer, intent(in) workers,
            integer, intent(in) initial_phreeqc,
            integer, intent(in) utility,
            character(len=*), intent(in) chem_name )
```

Run a PHREEQC input file. The first three arguments determine which IPhreeqc instances will run the file–the workers, the InitialPhreeqc instance, and (or) the Utility instance. Input files that modify the thermodynamic database should be run by all three sets of instances. Files with SELECTED_OUTPUT definitions that will be used during the time-stepping loop need to be run by the workers. Files that contain initial conditions or boundary conditions should be run by the InitialPhreeqc instance.

Parameters

id	The instance *id* returned from RM_Create.
workers	1, the workers will run the file; 0, the workers will not run the file.
initial_phreeqc	1, the InitialPhreeqc instance will run the file; 0, the InitialPhreeqc will not run the file.
utility	1, the Utility instance will run the file; 0, the Utility instance will not run the file.
chem_name	Name of the file to run.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_RunString.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.81 rm_runstring()

```
integer function phreeqcrm::rm_runstring (
             integer, intent(in) id,
             integer, intent(in) workers,
             integer, intent(in) initial_phreeqc,
             integer, intent(in) utility,
             character(len=*), intent(in) input_string )
```

Run a PHREEQC input string. The first three arguments determine which IPhreeqc instances will run the string–the workers, the InitialPhreeqc instance, and (or) the Utility instance. Input strings that modify the thermodynamic database should be run by all three sets of instances. Strings with SELECTED_OUTPUT definitions that will be used during the time-stepping loop need to be run by the workers. Strings that contain initial conditions or boundary conditions should be run by the InitialPhreeqc instance.

Parameters

id	The instance *id* returned from RM_Create.
workers	1, the workers will run the string; 0, the workers will not run the string.
initial_phreeqc	1, the InitialPhreeqc instance will run the string; 0, the InitialPhreeqc will not run the string.
utility	1, the Utility instance will run the string; 0, the Utility instance will not run the string.
input_string	String containing PHREEQC input.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_RunFile.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.82 rm_screenmessage()

```
integer function phreeqcrm::rm_screenmessage (
             integer, intent(in) id,
             character(len=*), intent(in) str )
```

Print message to the screen.

Parameters

id	The instance *id* returned from RM_Create.
str	String to be printed.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_ErrorMessage,
RM_LogMessage, RM_OutputMessage, RM_WarningMessage.

Fortran Example:

MPI:

Called by root and (or) workers.

3.3.2.83 rm_setcomponenth2o()

```
integer function phreeqcrm::rm_setcomponenth2o (
             integer, intent(in) id,
             integer, intent(in) tf )
```

Select whether to include H2O in the component list. The concentrations of H and O must be known accurately (8 to 10 significant digits) for the numerical method of PHREEQC to produce accurate pH and pe values. Because most of the H and O are in the water species, it may be more robust (require less accuracy in transport) to transport the excess H and O (the H and O not in water) and water. The default setting (*true*) is to include water, excess H, and excess O as components. A setting of *false* will include total H and total O as components. *RM_SetComponentH2O* must be called before RM_FindComponents.

Parameters

id	The instance id returned from RM_Create.
tf	0, total H and O are included in the component list; 1, excess H, excess O, and water are included in the component list.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_FindComponents.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.84 rm_setconcentrations()

```
integer function phreeqcrm::rm_setconcentrations (
             integer, intent(in) id,
             double precision, dimension(:,:), intent(in) c )
```

Use the vector of concentrations (*c*) to set the moles of components in each reaction cell. The volume of water in a cell is the product of porosity (RM_SetPorosity), saturation (RM_SetSaturation), and reference volume (RM_SetRepresentativeVolume). The moles of each component are determined by the volume of water and per liter concentrations. If concentration units (RM_SetUnitsSolution) are mass fraction, the density (as specified by RM_SetDensity) is used to convert from mass fraction to per mass per liter.

Parameters

id	The instance *id* returned from RM_Create.
c	Array of component concentrations. Size of array is (*nxyz*, *ncomps*), where *nxyz* is the number of grid cells in the user's model (RM_GetGridCellCount), and *ncomps* is the number of components as determined by RM_FindComponents or RM_GetComponentCount.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_SetDensity, RM_SetPorosity, RM_SetRepresentativeVolume, RM_SetSaturation, RM_SetUnitsSolution.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.85 rm_setcurrentselectedoutputusernumber()

```
integer function phreeqcrm::rm_setcurrentselectedoutputusernumber (
             integer, intent(in) id,
             integer, intent(in) n_user )
```

Select the current selected output by user number. The user may define multiple SELECTED_OUTPUT data blocks for the workers. A user number is specified for each data block. The value of the argument *n_user* selects which of the SELECTED_OUTPUT definitions will be used for selected-output operations.

Parameters

id	The instance *id* returned from RM_Create.
n_user	User number of the SELECTED_OUTPUT data block that is to be used.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_GetNthSelectedOutputUserNumber, RM_GetSelectedOutput, RM_GetSelectedOutputColumnCount, RM_GetSelectedOutputCount, RM_GetSelectedOutputHeading, RM_SetSelectedOutputOn, RM_Get↩ SelectedOutputRowCount.

Fortran Example:

MPI:

Called by root. ∗/

3.3.2.86 rm_setdensity()

```
integer function phreeqcrm::rm_setdensity (
             integer, intent(in) id,
             double precision, dimension(:), intent(in) density )
```

Set the density for each reaction cell. These density values are used when converting from transported mass fraction concentrations (RM_SetUnitsSolution) to produce per liter concentrations during a call to RM_SetConcentrations. They are also used when converting from module concentrations to transport concentrations of mass fraction (R↩ M_GetConcentrations), if RM_UseSolutionDensityVolume is set to *false*.

Parameters

id	The instance *id* returned from RM_Create.
density	Array of densities. Size of array is *nxyz*, where *nxyz* is the number of grid cells in the user's model (RM_GetGridCellCount).

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_GetConcentrations, RM_SetConcentrations, RM_SetUnitsSolution, RM_UseSolutionDensityVolume.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.87 rm_setdumpfilename()

```
integer function phreeqcrm::rm_setdumpfilename (
             integer, intent(in) id,
             character(len=*), intent(in) dump_name )
```

Set the name of the dump file. It is the name used by RM_DumpModule.

Parameters

id	The instance *id* returned from RM_Create.
dump_name	Name of dump file.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_DumpModule.

Fortran Example:

MPI:

Called by root.

3.3.2.88 rm_seterrorhandlermode()

```
integer function phreeqcrm::rm_seterrorhandlermode (
             integer, intent(in) id,
             integer, intent(in) mode )
```

Set the action to be taken when the reaction module encounters an error. Options are 0, return to calling program with an error return code (default); 1, throw an exception, in C++, the exception can be caught, for C and Fortran, the program will exit; or 2, attempt to exit gracefully.

Parameters

id	The instance id returned from RM_Create.
mode	Error handling mode: 0, 1, or 2.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.89 rm_seterroron()

```
integer function phreeqcrm::rm_seterroron (
             integer, intent(in) id,
             integer, intent(in) tf )
```

Set the property that controls whether error messages are generated and displayed. Messages include PHREEQC "ERROR" messages, and any messages written with RM_ErrorMessage.

Parameters

id	The instance *id* returned from RM_Create.
tf	*1*, enable error messages; *0*, disable error messages. Default is 1.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_ErrorMessage, RM_ScreenMessage.

Fortran Example:

MPI:

Called by root.

3.3.2.90 rm_setfileprefix()

```
integer function phreeqcrm::rm_setfileprefix (
             integer, intent(in) id,
             character(len=*), intent(in) prefix )
```

Set the prefix for the output (prefix.chem.txt) and log (prefix.log.txt) files. These files are opened by RM_OpenFiles.

Parameters

id	The instance *id* returned from RM_Create.
prefix	Prefix used when opening the output and log files.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_OpenFiles, RM_CloseFiles.

Fortran Example:

MPI:

Called by root.

3.3.2.91 rm_setgascompmoles()

```
integer function phreeqcrm::rm_setgascompmoles (
             integer, intent(in) id,
             double precision, dimension(:,:), intent(in) gas_moles )
```

Use the array of concentrations (*gas_moles*) to set the moles of gas components in each reaction cell.

Parameters

id	The instance *id* returned from RM_Create.
gas_moles	Array of moles of gas components. Dimensions of the vector are (nxyz, ngas_comps), where ngas_comps is the result of RM_GetGasComponentsCount, and *nxyz* is the number of user grid cells (RM_GetGridCellCount). If the number of moles is set to a negative number, the gas component will not be defined for the GAS_PHASE of the reaction cell.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_FindComponents, RM_GetGasComponentsCount, RM_GetGasCompMoles, RM_GetGasCompPressures, RM_GetGasCompPhi, RM_GetGasPhaseVolume, RM_SetGasPhaseVolume.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.92 rm_setgasphasevolume()

```
integer function phreeqcrm::rm_setgasphasevolume (
             integer, intent(in) id,
             double precision, dimension(:), intent(in) gas_volume )
```

Transfer volumes of gas phases from the array given in the argument list (*gas_volume*) to each reaction cell. The gas-phase volume affects the pressures calculated for fixed-volume gas phases. If a gas-phase volume is defined with this method for a GAS_PHASE in a cell, the gas phase is forced to be a fixed-volume gas phase.

Parameters

id	The instance *id* returned from RM_Create.
gas_volume	Array of gas-phase volumes. Dimension of the array is (nxyz), where *nxyz* is the number of user grid cells (RM_GetGridCellCount). If an element of the array is set to a negative number, the gas component will not be defined for the GAS_PHASE of the reaction cell.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_FindComponents, RM_GetGasComponentsCount, RM_GetGasCompMoles, RM_GetGasCompPressures, RM_GetGasCompPhi, RM_GetGasPhaseVolume, RM_SetGasCompMoles.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.93 rm_setmpiworkercallback()

```
integer function phreeqcrm::rm_setmpiworkercallback (
             integer, intent(in) id,
              fcn )
```

MPI only. Defines a callback function that allows additional tasks to be done by the workers. The method RM↩ _MpiWorker contains a loop, where the workers receive a message (an integer), run a function corresponding to that integer, and then wait for another message. RM_SetMpiWorkerCallback allows the developer to add another function that responds to additional integer messages by calling developer-defined functions corresponding to those integers. RM_MpiWorker calls the callback function when the message number is not one of the PhreeqcRM message numbers. Messages are unique integer numbers. PhreeqcRM uses integers in a range beginning at 0. It is suggested that developers use message numbers starting at 1000 or higher for their tasks. The callback function calls a developer-defined function specified by the message number and then returns to RM_MpiWorker to wait for another message.

For Fortran, the functions that are called from the callback function can use USE statements to find the data necessary to perform the tasks, and the only argument to the callback function is an integer message argument. *RM_SetMpiWorkerCallback* must be called by each worker before RM_MpiWorker is called.

The motivation for this method is to allow the workers to perform other tasks, for instance, parallel transport calculations, within the structure of RM_MpiWorker. The callback function can be used to allow the workers to receive data, perform transport calculations, and (or) send results, without leaving the loop of RM_MpiWorker. Alternatively, it is possible for the workers to return from RM_MpiWorker by a call to RM_MpiWorkerBreak by root. The workers could then call subroutines to receive data, calculate transport, and send data, and then resume processing PhreeqcRM messages from root with another call to RM_MpiWorker.

Parameters

id	The instance *id* returned from RM_Create.
fcn	A function that returns an integer and has an integer argument.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_MpiWorker, RM_MpiWorkerBreak.

Fortran Example:

MPI:

Called by workers, before call to RM_MpiWorker.

3.3.2.94 rm_setpartitionuzsolids()

```
integer function phreeqcrm::rm_setpartitionuzsolids (
             integer, intent(in) id,
             integer, intent(in) tf )
```

Sets the property for partitioning solids between the saturated and unsaturated parts of a partially saturated cell.

The option is intended to be used by saturated-only flow codes that allow a variable water table. The value has meaning only when saturations less than 1.0 are encountered. The partially saturated cells may have a small water-to-rock ratio that causes reactions to proceed differently relative to fully saturated cells. By setting *RM_Set↩ PartitionUZSolids* to true, the amounts of solids and gases are partioned according to the saturation. If a cell has a saturation of 0.5, then the water interacts with only half of the solids and gases; the other half is unreactive until the water table rises. As the saturation in a cell varies, solids and gases are transferred between the saturated and unsaturated (unreactive) reservoirs of the cell. Unsaturated-zone flow and transport codes will probably use the default (false), which assumes all gases and solids are reactive regardless of saturation.

Parameters

id	The instance *id* returned from RM_Create.
tf	*True*, the fraction of solids and gases available for reaction is equal to the saturation; *False* (default), all solids and gases are reactive regardless of saturation.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.95 rm_setporosity()

```
integer function phreeqcrm::rm_setporosity (
             integer, intent(in) id,
             double precision, dimension(:), intent(in) por )
```

Set the porosity for each reaction cell. The volume of water in a reaction cell is the product of the porosity, the saturation (RM_SetSaturation), and the representative volume (RM_SetRepresentativeVolume).

Parameters

id	The instance *id* returned from RM_Create.
por	Array of porosities, unitless. Default is 0.1. Size of array is *nxyz*, where *nxyz* is the number of grid cells in the user's model (RM_GetGridCellCount).

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_GetSaturation, RM_SetRepresentativeVolume, RM_SetSaturation.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.96 rm_setpressure()

```
integer function phreeqcrm::rm_setpressure (
             integer, intent(in) id,
             double precision, dimension(:), intent(in) p )
```

Set the pressure for each reaction cell. Pressure effects are considered only in three of the databases distributed with PhreeqcRM: phreeqc.dat, Amm.dat, and pitzer.dat.

Parameters

id	The instance *id* returned from RM_Create.
p	Array of pressures, in atm. Size of array is *nxyz*, where *nxyz* is the number of grid cells in the user's model (RM_GetGridCellCount).

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_SetTemperature.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.97 rm_setprintchemistrymask()

```
integer function phreeqcrm::rm_setprintchemistrymask (
             integer, intent(in) id,
             integer, dimension(:), intent(in) cell_mask )
```

Enable or disable detailed output for each reaction cell. Printing for a cell will occur only when the printing is enabled with RM_SetPrintChemistryOn and the *cell_mask* value is 1.

Parameters

id	The instance *id* returned from RM_Create.
cell_mask	Array of integers. Size of array is *nxyz*, where *nxyz* is the number of grid cells in the user's model (RM_GetGridCellCount). A value of 0 will disable printing detailed output for the cell; a value of 1 will enable printing detailed output for a cell.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_SetPrintChemistryOn.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.98 rm_setprintchemistryon()

```
integer function phreeqcrm::rm_setprintchemistryon (
             integer, intent(in) id,
             integer, intent(in) workers,
             integer, intent(in) initial_phreeqc,
             integer, intent(in) utility )
```

Setting to enable or disable printing detailed output from reaction calculations to the output file for a set of cells defined by RM_SetPrintChemistryMask. The detailed output prints all of the output typical of a PHREEQC reaction calculation, which includes solution descriptions and the compositions of all other reactants. The output can be several hundred lines per cell, which can lead to a very large output file (prefix.chem.txt, RM_OpenFiles). For the worker instances, the output can be limited to a set of cells (RM_SetPrintChemistryMask) and, in general, the amount of information printed can be limited by use of options in the PRINT data block of PHREEQC (applied by using RM_RunFile or RM_RunString). Printing the detailed output for the workers is generally used only for debugging, and PhreeqcRM will run significantly faster when printing detailed output for the workers is disabled.

Parameters

id	The instance *id* returned from RM_Create.
workers	0, disable detailed printing in the worker instances, 1, enable detailed printing in the worker instances.
initial_phreeqc	0, disable detailed printing in the InitialPhreeqc instance, 1, enable detailed printing in the InitialPhreeqc instances.
utility	0, disable detailed printing in the Utility instance, 1, enable detailed printing in the Utility instance.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_SetPrintChemistryMask.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.99 rm_setrebalancebycell()

```
integer function phreeqcrm::rm_setrebalancebycell (
             integer, intent(in) id,
             integer, intent(in) method )
```

Set the load-balancing algorithm. PhreeqcRM attempts to rebalance the load of each thread or process such that each thread or process takes the same amount of time to run its part of a RM_RunCells calculation. Two algorithms are available; one uses individual times for each cell and accounts for cells that were not run because saturation was zero (default), and the other assigns an average time to all cells. The methods are similar, but limited testing indicates the default method performs better.

Parameters

id	The instance *id* returned from RM_Create.
method	0, indicates average times are used in rebalancing; 1 indicates individual cell times are used in rebalancing (default).

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_SetRebalanceFraction.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.100 rm_setrebalancefraction()

```
integer function phreeqcrm::rm_setrebalancefraction (
             integer, intent(in) id,
             double precision, intent(in) f )
```

Sets the fraction of cells that are transferred among threads or processes when rebalancing. PhreeqcRM attempts to rebalance the load of each thread or process such that each thread or process takes the same amount of time to run its part of a RM_RunCells calculation. The rebalancing transfers cell calculations among threads or processes to try to achieve an optimum balance. *RM_SetRebalanceFraction* adjusts the calculated optimum number of cell transfers by a fraction from 0 to 1.0 to determine the actual number of cell transfers. A value of zero eliminates load rebalancing. A value less than 1.0 is suggested to slow the approach to the optimum cell distribution and avoid possible oscillations when too many cells are transferred at one iteration, requiring reverse transfers at the next iteration. Default is 0.5.

Parameters

id	The instance *id* returned from RM_Create.
f	Fraction from 0.0 to 1.0.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_SetRebalanceByCell.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.101 rm_setrepresentativevolume()

```
integer function phreeqcrm::rm_setrepresentativevolume (
             integer, intent(in) id,
             double precision, dimension(:), intent(in) rv )
```

Set the representative volume of each reaction cell. By default the representative volume of each reaction cell is 1 liter. The volume of water in a reaction cell is determined by the procuct of the representative volume, the porosity (RM_SetPorosity), and the saturation (RM_SetSaturation). The numerical method of PHREEQC is more robust if the water volume for a reaction cell is within a couple orders of magnitude of 1.0. Small water volumes caused by small porosities and (or) small saturations (and (or) small representative volumes) may cause non-convergence of the numerical method. In these cases, a larger representative volume may help. Note that increasing the representative volume also increases the number of moles of the reactants in the reaction cell (minerals, surfaces, exchangers, and others), which are defined as moles per representative volume.

Parameters

id	The instance *id* returned from RM_Create.
rv	Vector of representative volumes, in liters. Default is 1.0 liter. Size of array is *nxyz*, where *nxyz* is the number of grid cells in the user's model (RM_GetGridCellCount).

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_SetPorosity, RM_SetSaturation.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.102 rm_setsaturation()

```
integer function phreeqcrm::rm_setsaturation (
             integer, intent(in) id,
             double precision, dimension(:), intent(in) sat )
```

Set the saturation of each reaction cell. Saturation is a fraction ranging from 0 to 1. The volume of water in a cell is the product of porosity (RM_SetPorosity), saturation (*RM_SetSaturation*), and representative volume (RM_SetRepresentativeVolume). As a result of a reaction calculation, solution properties (density and volume) will change; the databases phreeqc.dat, Amm.dat, and pitzer.dat have the molar volume data to calculate these changes. The methods RM_GetDensity, RM_GetSolutionVolume, and RM_GetSaturation can be used to account for these changes in the succeeding transport calculation. *RM_SetRepresentativeVolume* should be called before initial conditions are defined for the reaction cells.

Parameters

id	The instance *id* returned from RM_Create.
sat	Array of saturations, unitless. Size of array is *nxyz*, where *nxyz* is the number of grid cells in the user's model (RM_GetGridCellCount).

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_GetDensity, RM_GetSaturation, RM_GetSolutionVolume, RM_SetPorosity, RM_SetRepresentativeVolume.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.103 rm_setscreenon()

```
integer function phreeqcrm::rm_setscreenon (
             integer, intent(in) id,
             integer, intent(in) tf )
```

Set the property that controls whether messages are written to the screen. Messages include information about rebalancing during RM_RunCells, and any messages written with RM_ScreenMessage.

Parameters

id	The instance *id* returned from RM_Create.
tf	*1*, enable screen messages; *0*, disable screen messages. Default is 1.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_RunCells, RM_ScreenMessage.

Fortran Example:

MPI:

Called by root.

3.3.2.104 rm_setselectedoutputon()

```
integer function phreeqcrm::rm_setselectedoutputon (
             integer, intent(in) id,
             integer, intent(in) tf )
```

Setting determines whether selected-output results are available to be retrieved with RM_GetSelectedOutput. *1* indicates that selected-output results will be accumulated during RM_RunCells and can be retrieved with RM_↩ GetSelectedOutput; *0* indicates that selected-output results will not be accumulated during RM_RunCells.

Parameters

id	The instance *id* returned from RM_Create.
tf	0, disable selected output; 1, enable selected output.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_GetSelectedOutput, RM_SetPrintChemistryOn.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.105 rm_setspeciessaveon()

```
integer function phreeqcrm::rm_setspeciessaveon (
             integer, intent(in) id,
             integer, intent(in) save_on )
```

Sets the value of the species-save property. This method enables use of PhreeqcRM with multicomponent-diffusion transport calculations. By default, concentrations of aqueous species are not saved. Setting the species-save property to 1 allows aqueous species concentrations to be retrieved with RM_GetSpeciesConcentrations, and solution compositions to be set with RM_SpeciesConcentrations2Module. RM_SetSpeciesSaveOn must be called before calls to RM_FindComponents.

Parameters

id	The instance *id* returned from RM_Create.
save_on	0, indicates species concentrations are not saved; 1, indicates species concentrations are saved.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_FindComponents, RM_GetSpeciesConcentrations, RM_GetSpeciesCount, RM_GetSpeciesD25, RM_GetSpeciesLog10Gammas, RM_GetSpeciesLog10Molalities, RM_GetSpeciesName, RM_GetSpeciesSaveOn, RM_GetSpeciesZ, RM_SpeciesConcentrations2Module.

Fortran Example:

MPI:

Called by root and (or) workers.

3.3.2.106 rm_settemperature()

```
integer function phreeqcrm::rm_settemperature (
             integer, intent(in) id,
             double precision, dimension(:), intent(in) t )
```

Set the temperature for each reaction cell. If *RM_SetTemperature* is not called, worker solutions will have temperatures as defined by initial conditions (RM_InitialPhreeqc2Module and RM_InitialPhreeqcCell2Module).

Parameters

id	The instance *id* returned from RM_Create.
t	Array of temperatures, in degrees C. Size of array is *nxyz*, where *nxyz* is the number of grid cells in the user's model (RM_GetGridCellCount).

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_InitialPhreeqc2Module, RM_InitialPhreeqcCell2Module, RM_SetPressure.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.107 rm_settime()

```
integer function phreeqcrm::rm_settime (
             integer, intent(in) id,
             double precision, intent(in) time )
```

Set current simulation time for the reaction module.

Parameters

id	The instance *id* returned from RM_Create.
time	Current simulation time, in seconds.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_SetTimeConversion,
RM_SetTimeStep.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.108 rm_settimeconversion()

```
integer function phreeqcrm::rm_settimeconversion (
             integer, intent(in) id,
             double precision, intent(in) conv_factor )
```

Set a factor to convert to user time units. Factor times seconds produces user time units.

Parameters

id	The instance *id* returned from RM_Create.
conv_factor	Factor to convert seconds to user time units.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_SetTime, RM_SetTimeStep.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.109 rm_settimestep()

```
integer function phreeqcrm::rm_settimestep (
             integer, intent(in) id,
             double precision, intent(in) time_step )
```

Set current time step for the reaction module. This is the length of time over which kinetic reactions are integrated.

Parameters

id	The instance *id* returned from RM_Create.
time_step	Current time step, in seconds.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_SetTime, RM_SetTimeConversion.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.110 rm_setunitsexchange()

```
integer function phreeqcrm::rm_setunitsexchange (
             integer, intent(in) id,
             integer, intent(in) option )
```

Sets input units for exchangers. In PHREEQC input, exchangers are defined by moles of exchange sites (*Mp*). *RM_SetUnitsExchange* specifies how the number of moles of exchange sites in a reaction cell (*Mc*) is calculated from the input value (*Mp*).

Options are 0, *Mp* is mol/L of RV (default), *Mc* = *Mp*∗*RV*, where RV is the representative volume (RM_Set↩ RepresentativeVolume); 1, *Mp* is mol/L of water in the RV, *Mc* = *Mp*∗*P*∗*RV*, where *P* is porosity (RM_SetPorosity); or 2, *Mp* is mol/L of rock in the RV, *Mc* = *Mp*∗(1-P)∗RV.

If a single EXCHANGE definition is used for cells with different initial porosity, the three options scale quite differently. For option 0, the number of moles of exchangers will be the same regardless of porosity. For option 1, the number of moles of exchangers will be vary directly with porosity and inversely with rock volume. For option 2, the number of moles of exchangers will vary directly with rock volume and inversely with porosity.

Parameters

id	The instance *id* returned from RM_Create.
option	Units option for exchangers: 0, 1, or 2.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_InitialPhreeqc2Module, RM_InitialPhreeqcCell2Module, RM_SetPorosity, RM_SetRepresentativeVolume.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.111 rm_setunitsgasphase()

```
integer function phreeqcrm::rm_setunitsgasphase (
             integer, intent(in) id,
             integer, intent(in) option )
```

Set input units for gas phases. In PHREEQC input, gas phases are defined by moles of component gases (*Mp*). *RM_SetUnitsGasPhase* specifies how the number of moles of component gases in a reaction cell (*Mc*) is calculated from the input value (*Mp*).

Options are 0, *Mp* is mol/L of RV (default), $Mc = Mp*RV$, where RV is the representative volume (RM_SetRepresentativeVolume); 1, *Mp* is mol/L of water in the RV, $Mc = Mp*P*RV$, where *P* is porosity (RM_SetPorosity); or 2, *Mp* is mol/L of rock in the RV, $Mc = Mp*(1-P)*$RV.

If a single GAS_PHASE definition is used for cells with different initial porosity, the three options scale quite differently. For option 0, the number of moles of a gas component will be the same regardless of porosity. For option 1, the number of moles of a gas component will be vary directly with porosity and inversely with rock volume. For option 2, the number of moles of a gas component will vary directly with rock volume and inversely with porosity.

Parameters

id	The instance *id* returned from RM_Create.
option	Units option for gas phases: 0, 1, or 2.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_InitialPhreeqc2Module, RM_InitialPhreeqcCell2Module, RM_SetPorosity, RM_SetRepresentativeVolume.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.112 rm_setunitskinetics()

```
integer function phreeqcrm::rm_setunitskinetics (
             integer, intent(in) id,
             integer, intent(in) option )
```

Set input units for kinetic reactants.

In PHREEQC input, kinetics are defined by moles of kinetic reactants (*Mp*). *RM_SetUnitsKinetics* specifies how the number of moles of kinetic reactants in a reaction cell (*Mc*) is calculated from the input value (*Mp*).

Options are 0, *Mp* is mol/L of RV (default), $Mc = Mp*RV$, where RV is the representative volume (RM_SetRepresentativeVolume); 1, *Mp* is mol/L of water in the RV, $Mc = Mp*P*RV$, where *P* is porosity (RM_SetPorosity); or 2, *Mp* is mol/L of rock in the RV, $Mc = Mp*(1-P)*$RV.

If a single KINETICS definition is used for cells with different initial porosity, the three options scale quite differently. For option 0, the number of moles of kinetic reactants will be the same regardless of porosity. For option 1, the number of moles of kinetic reactants will be vary directly with porosity and inversely with rock volume. For option 2, the number of moles of kinetic reactants will vary directly with rock volume and inversely with porosity.

Note that the volume of water in a cell in the reaction module is equal to the product of porosity (RM_SetPorosity), the saturation (RM_SetSaturation), and representative volume (RM_SetRepresentativeVolume), which is usually less than 1 liter. It is important to write the RATES definitions for homogeneous (aqueous) kinetic reactions to account for the current volume of water, often by calculating the rate of reaction per liter of water and multiplying by the volume of water (Basic function SOLN_VOL).

Rates that depend on surface area of solids, are not dependent on the volume of water. However, it is important to get the correct surface area for the kinetic reaction. To scale the surface area with the number of moles, the specific area (m^2 per mole of reactant) can be defined as a parameter (KINETICS; -parm), which is multiplied by the number of moles of reactant (Basic function M) in RATES to obtain the surface area.

Parameters

id	The instance *id* returned from RM_Create.
option	Units option for kinetic reactants: 0, 1, or 2.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_InitialPhreeqc2Module, RM_InitialPhreeqcCell2Module, RM_SetPorosity, RM_SetRepresentativeVolume, RM_SetSaturation.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.113 rm_setunitsppassemblage()

```
integer function phreeqcrm::rm_setunitsppassemblage (
             integer, intent(in) id,
             integer, intent(in) option )
```

Set input units for pure phase assemblages (equilibrium phases). In PHREEQC input, equilibrium phases are defined by moles of each phase (*Mp*). *RM_SetUnitsPPassemblage* specifies how the number of moles of phases in a reaction cell (*Mc*) is calculated from the input value (*Mp*).

Options are 0, *Mp* is mol/L of RV (default), $Mc = Mp * RV$, where RV is the representative volume (RM_SetRepresentativeVolume); 1, *Mp* is mol/L of water in the RV, $Mc = Mp * P * RV$, where *P* is porosity (RM_SetPorosity); or 2, *Mp* is mol/L of rock in the RV, $Mc = Mp * (1-P) * RV$.

If a single EQUILIBRIUM_PHASES definition is used for cells with different initial porosity, the three options scale quite differently. For option 0, the number of moles of a mineral will be the same regardless of porosity. For option 1, the number of moles of a mineral will be vary directly with porosity and inversely with rock volume. For option 2, the number of moles of a mineral will vary directly with rock volume and inversely with porosity.

Parameters

id	The instance *id* returned from RM_Create.
option	Units option for equilibrium phases: 0, 1, or 2.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_InitialPhreeqc2Module, RM_InitialPhreeqcCell2Module, RM_SetPorosity, RM_SetRepresentativeVolume.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.114 rm_setunitssolution()

```
integer function phreeqcrm::rm_setunitssolution (
             integer, intent(in) id,
             integer, intent(in) option )
```

Solution concentration units used by the transport model. Options are 1, mg/L; 2 mol/L; or 3, mass fraction, kg/kgs. PHREEQC defines solutions by the number of moles of each element in the solution.

To convert from mg/L to moles of element in the representative volume of a reaction cell, mg/L is converted to mol/L and multiplied by the solution volume, which is the product of porosity (RM_SetPorosity), saturation (RM_SetSaturation), and representative volume (RM_SetRepresentativeVolume). To convert from mol/L to moles of element in the representative volume of a reaction cell, mol/L is multiplied by the solution volume. To convert from mass fraction to moles of element in the representative volume of a reaction cell, kg/kgs is converted to mol/kgs, multiplied by density (RM_SetDensity) and multiplied by the solution volume.

To convert from moles of element in the representative volume of a reaction cell to mg/L, the number of moles of an element is divided by the solution volume resulting in mol/L, and then converted to mg/L. To convert from moles of element in a cell to mol/L, the number of moles of an element is divided by the solution volume resulting in mol/L. To convert from moles of element in a cell to mass fraction, the number of moles of an element is converted to kg and divided by the total mass of the solution. Two options are available for the volume and mass of solution that are used in converting to transport concentrations: (1) the volume and mass of solution are calculated by PHREEQC, or (2) the volume of solution is the product of porosity (RM_SetPorosity), saturation (RM_SetSaturation), and representative volume (RM_SetRepresentativeVolume), and the mass of solution is volume times density as defined by RM_SetDensity. Which option is used is determined by RM_UseSolutionDensityVolume.

Parameters

id	The instance *id* returned from RM_Create.
option	Units option for solutions: 1, 2, or 3, default is 1, mg/L.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_SetDensity, RM_SetPorosity, RM_SetRepresentativeVolume, RM_SetSaturation, RM_UseSolutionDensityVolume.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.115 rm_setunitsssassemblage()

```
integer function phreeqcrm::rm_setunitsssassemblage (
             integer, intent(in) id,
             integer, intent(in) option )
```

Set input units for solid-solution assemblages. In PHREEQC, solid solutions are defined by moles of each component (*Mp*). *RM_SetUnitsSSassemblage* specifies how the number of moles of solid-solution components in a reaction cell (*Mc*) is calculated from the input value (*Mp*).

Options are 0, *Mp* is mol/L of RV (default), $Mc = Mp*RV$, where RV is the representative volume (RM_SetRepresentativeVolume); 1, *Mp* is mol/L of water in the RV, $Mc = Mp*P*RV$, where *P* is porosity (RM_SetPorosity); or 2, *Mp* is mol/L of rock in the RV, $Mc = Mp*(1-P)*$RV.

Parameters

id	The instance *id* returned from RM_Create.
option	Units option for solid solutions: 0, 1, or 2.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_InitialPhreeqc2Module, RM_InitialPhreeqcCell2Module, RM_SetPorosity, RM_SetRepresentativeVolume.

If a single SOLID_SOLUTION definition is used for cells with different initial porosity, the three options scale quite differently. For option 0, the number of moles of a solid-solution component will be the same regardless of porosity. For option 1, the number of moles of a solid-solution component will be vary directly with porosity and inversely with rock volume. For option 2, the number of moles of a solid-solution component will vary directly with rock volume and inversely with porosity.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.116 rm_setunitssurface()

```
integer function phreeqcrm::rm_setunitssurface (
             integer, intent(in) id,
             integer, intent(in) option )
```

Set input units for surfaces. In PHREEQC input, surfaces are defined by moles of surface sites (*Mp*). *RM_Set↩ UnitsSurface* specifies how the number of moles of surface sites in a reaction cell (*Mc*) is calculated from the input value (*Mp*).

Options are 0, *Mp* is mol/L of RV (default), *Mc* = *Mp*∗*RV*, where RV is the representative volume (RM_Set↩ RepresentativeVolume); 1, *Mp* is mol/L of water in the RV, *Mc* = *Mp*∗*P*∗*RV*, where *P* is porosity (RM_SetPorosity); or 2, *Mp* is mol/L of rock in the RV, *Mc* = *Mp*∗(1-*P*)∗RV.

If a single SURFACE definition is used for cells with different initial porosity, the three options scale quite differently. For option 0, the number of moles of surface sites will be the same regardless of porosity. For option 1, the number of moles of surface sites will be vary directly with porosity and inversely with rock volume. For option 2, the number of moles of surface sites will vary directly with rock volume and inversely with porosity.

Parameters

id	The instance *id* returned from RM_Create.
option	Units option for surfaces: 0, 1, or 2.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_InitialPhreeqc2Module, RM_InitialPhreeqcCell2Module, RM_SetPorosity, RM_SetRepresentative↩ Volume.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.117 rm_speciesconcentrations2module()

```
integer function phreeqcrm::rm_speciesconcentrations2module (
             integer, intent(in) id,
             double precision, dimension(:,:), intent(in) species_conc )
```

Set solution concentrations in the reaction cells based on the vector of aqueous species concentrations (*species_conc*). This method is intended for use with multicomponent-diffusion transport calculations, and RM_SetSpeciesSaveOn must be set to *true*. The list of aqueous species is determined by RM_FindComponents and includes all aqueous species that can be made from the set of components. The method determines the total concentration of a component by summing the molarities of the individual species times the stoichiometric coefficient of the element in each species. Solution compositions in the reaction cells are updated with these component concentrations.

Parameters

id	The instance *id* returned from RM_Create.
species_conc	Array of aqueous species concentrations. Dimension of the array is (*nxyz*, *nspecies*), where *nxyz* is the number of user grid cells (RM_GetGridCellCount), and *nspecies* is the number of aqueous species (RM_GetSpeciesCount). Concentrations are moles per liter.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_FindComponents, RM_GetSpeciesConcentrations, RM_GetSpeciesCount, RM_GetSpeciesD25, RM_GetSpeciesLog10Gammas, RM_GetSpeciesLog10Molalities, RM_GetSpeciesName, RM_GetSpeciesSaveOn, RM_GetSpeciesZ, RM_SetSpeciesSaveOn.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.118 rm_usesolutiondensityvolume()

```
integer function phreeqcrm::rm_usesolutiondensityvolume (
             integer, intent(in) id,
             integer, intent(in) tf )
```

Determines the volume and density to use when converting from the reaction-module concentrations to transport concentrations (RM_GetConcentrations). Two options are available to convert concentration units: (1) the density and solution volume calculated by PHREEQC are used, or (2) the specified density (RM_SetDensity) and solution volume are defined by the product of saturation (RM_SetSaturation), porosity (RM_SetPorosity), and representative volume (RM_SetRepresentativeVolume). Transport models that consider density-dependent flow will probably use the PHREEQC-calculated density and solution volume (default), whereas transport models that assume constant-density flow will probably use specified values of density and solution volume. Only the following databases distributed with PhreeqcRM have molar volume information needed to accurately calculate density and solution volume: phreeqc.dat, Amm.dat, and pitzer.dat. Density is only used when converting to transport units of mass fraction.

Parameters

id	The instance *id* returned from RM_Create.
tf	*True* indicates that the solution density and volume as calculated by PHREEQC will be used to calculate concentrations. *False* indicates that the solution density set by RM_SetDensity and the volume determined by the product of RM_SetSaturation, RM_SetPorosity, and RM_SetRepresentativeVolume, will be used to calculate concentrations retrieved by RM_GetConcentrations.

See also

RM_GetConcentrations, RM_SetDensity, RM_SetPorosity, RM_SetRepresentativeVolume, RM_Set↩ Saturation.

Fortran Example:

MPI:

Called by root, workers must be in the loop of RM_MpiWorker.

3.3.2.119 rm_warningmessage()

```
integer function phreeqcrm::rm_warningmessage (
             integer, intent(in) id,
             character(len=*), intent(in) warn_str )
```

Print a warning message to the screen and the log file.

Parameters

id	The instance *id* returned from RM_Create.
warn_str	String to be printed.

Return values

IRM_RESULT	0 is success, negative is failure (See RM_DecodeError).

See also

RM_ErrorMessage,
RM_LogMessage,
RM_OpenFiles, RM_OutputMessage, RM_ScreenMessage.

Fortran Example:

MPI:

Called by root and (or) workers; only root writes to the log file.

3.4 xgb_interface Module Reference

Interface to call XGBoost library C API from fortran.

Data Types

- interface fortran_XGBoosterCreate
- interface fortran_XGBoosterFree
- interface fortran_XGBoosterLoadModel
- interface fortran_XGBoosterPredict
- interface fortran_XGBoosterSaveModel
- interface fortran_XGBoosterSetParam
- interface fortran_XGDMatrixCreateFromMat

3.4.1 Detailed Description

Interface to call XGBoost library C API from fortran.

Author

Vinicius L S Silva

Chapter 4

Data Type Documentation

4.1 multi_data_types::allocate_multi_dev_shape_funs Interface Reference

Public Member Functions

- subroutine **allocate_multi_dev_shape_funs1** (funs, DevFuns, nx_all_FE_size)
- subroutine **allocate_multi_dev_shape_funs2** (Mdims, GIdims, DevFuns, nx_all_FE_size)
- subroutine **allocate_multi_dev_shape_funs3** (cvfenlx_all, ufenlx_all, DevFuns, nx_all_FE_size)

The documentation for this interface was generated from the following file:

- /home/psalinas/Documents/workspace/MultiFluids_Dev/ICFERST/src/multi_data_types.F90

4.2 multi_data_types::allocate_multi_field Interface Reference

Public Member Functions

- subroutine allocate_multi_field1 (state, Mdims, field_name, mfield)
- subroutine **allocate_multi_field2** (Mdims, mfield, nonods_in, field_name)

4.2.1 Member Function/Subroutine Documentation

4.2.1.1 allocate_multi_field1()

```
subroutine multi_data_types::allocate_multi_field::allocate_multi_field1 (
            type( state_type ), intent(in) state,
            type( multi_dimensions ), intent(in) Mdims,
            character( len = * ), intent(in) field_name,
            type( multi_field ), intent(inout) mfield )
```

Parameters

in	*state*	*********UNTESTED*********

The documentation for this interface was generated from the following file:

- /home/psalinas/Documents/workspace/MultiFluids_Dev/ICFERST/src/multi_data_types.F90

4.3 multi_tools::bad_elements Type Reference

: Type to keep an eye on the quality of the elements @DEPRECATED

Public Attributes

- integer **bad_ele**
- real, dimension(:,:), allocatable **angle**
- real **perp_height**
- real **perp**
- real **height**
- real **from**
- real, dimension(assuming an isosceles triangle) **base**
- real, dimension(:,:), allocatable **rotmatrix**
- real, dimension(:,:), allocatable **the**
- real, dimension(:,:), allocatable **rotation**
- real, dimension(:,:), allocatable **matrix**
- real, dimension(:,:), allocatable **to**
- real, dimension(:,:), allocatable **bad**
- real, dimension(:,:), allocatable **element**
- real, dimension(:,:), allocatable **in**
- real, dimension(:,:), allocatable **direction**
- real, dimension(:,:), allocatable **normal**
- real, dimension(:,:), allocatable **big**
- real **length**
- real **of**
- real **side**
- real **opposite**
- real **large**

4.3.1 Detailed Description

: Type to keep an eye on the quality of the elements @DEPRECATED

The documentation for this type was generated from the following file:

- /home/psalinas/Documents/workspace/MultiFluids_Dev/ICFERST/src/multi_tools.F90

4.4 multi_magma::coupling_term_coef Type Reference

The copling_term follows the Darcy permeability law C=1/a/d^2∗mu∗phi^(2-b) for lower meltfraction and hindled settling C=1/d^2∗mu∗phi^(-5)∗(1-phi). Here coefficients a, grain size d, coefficients b and the cutting between the two demains needs to be set.

Public Attributes

- real **a**
- real **b**
- real **grain_size**
- real **cut_low**
- real **the**
- real **range**
- real **below**
- real **which**
- real **coupling**
- real **terms**
- real **follow**
- real **darcy**
- real **permeability**
- real **set**
- real **this**
- real **to**
- real **make**
- real **entire**
- real **domain**
- real **like**
- real **cut_high**
- real **above**
- real **hindered**
- real **settling**

4.4.1 Detailed Description

The copling_term follows the Darcy permeability law C=1/a/d^2∗mu∗phi^(2-b) for lower meltfraction and hindled settling C=1/d^2∗mu∗phi^(-5)∗(1-phi). Here coefficients a, grain size d, coefficients b and the cutting between the two demains needs to be set.

The documentation for this type was generated from the following file:

- /home/psalinas/Documents/workspace/MultiFluids_Dev/ICFERST/src/multi_magma.F90

4.5 shape_functions_linear_quadratic::detnlxr Interface Reference

: Calculates the derivatives of the shape functions

Public Member Functions

- **detnlxr1**
- **detnlxr2**
- **detnlxr3**

4.5.1 Detailed Description

: Calculates the derivatives of the shape functions

The documentation for this interface was generated from the following file:

- /home/psalinas/Documents/workspace/MultiFluids_Dev/ICFERST/src/multi_shape_fct_ND.F90

4.6 shape_functions_linear_quadratic::detnlxr_invjac Interface Reference

: Computes the derivatives of the shape functions and the inverse of the Jacobian

Public Member Functions

- **detnlxr_invjac1**
- **detnlxr_invjac2**
- **detnlxr_invjac3**

4.6.1 Detailed Description

: Computes the derivatives of the shape functions and the inverse of the Jacobian

The documentation for this interface was generated from the following file:

- /home/psalinas/Documents/workspace/MultiFluids_Dev/ICFERST/src/multi_shape_fct_ND.F90

4.7 shape_functions_prototype::detnlxr_plus_u Interface Reference

Public Member Functions

- subroutine **detnlxr_plus_u1** (ELE, X_ALL, XONDGL, weight, cvshape, cvshapelx, ushapelx, DevFuns)
- subroutine **detnlxr_plus_u2** (ELE, X, Y, Z, XONDGL, TOTELE, NONODS, X_NLOC, CV_NLOC, NGI, N, NLX, NLY, NLZ, WEIGHT, DETWEI, RA, VOLUME, D1, D3, DCYL, NX_ALL, U_NLOC, UNLX, UNLY, UNLZ, UNX_ALL)

The documentation for this interface was generated from the following file:

- /home/psalinas/Documents/workspace/MultiFluids_Dev/ICFERST/src/multi_shape_fct.F90

4.8 cv_advection::dg_derivs_all Interface Reference

Public Member Functions

- subroutine **dg_derivs_all1** (FEMT, FEMTOLD, DTX_ELE, DTOLDX_ELE, NDIM, NPHASE, NCOMP, T↩ OTELE, CV_NDGLN, XCV_NDGLN, X_NLOC, X_NDGLN, CV_NGI, CV_NLOC, CVWEIGHT, N, NLX, NLY, NLZ, X_N, X_NLX, X_NLY, X_NLZ, X_NONODS, X, Y, Z, NFACE, FACE_ELE, CV_SLOCLIST, X_SLO↩ CLIST, CV_SNLOC, X_SNLOC, WIC_T_BC, SUF_T_BC, SBCVNGI, SBCVFEN, SBWEIGH, X_SBCVFEN, X_SBCVFENSLX, X_SBCVFENSLY, get_gradU, state, P0Mesh)
- subroutine **dg_derivs_all2** (FEMT, FEMTOLD, DTX_ELE, DTOLDX_ELE, NDIM, NPHASE, TOTELE, C↩ V_NDGLN, XCV_NDGLN, X_NLOC, X_NDGLN, CV_NGI, CV_NLOC, CVWEIGHT, N, NLX, NLY, NLZ, X_N, X_NLX, X_NLY, X_NLZ, X_NONODS, X, Y, Z, NFACE, FACE_ELE, CV_SLOCLIST, X_SLOCLIST, CV_S↩ NLOC, X_SNLOC, WIC_T_BC, SUF_T_BC, SBCVNGI, SBCVFEN, SBWEIGH, X_SBCVFEN, X_SBCVF↩ ENSLX, X_SBCVFENSLY, P0Mesh)

The documentation for this interface was generated from the following file:

- /home/psalinas/Documents/workspace/MultiFluids_Dev/ICFERST/src/cv-adv-dif.F90

4.9 setbasicfortrancallbackf::fcn Interface Reference

Public Member Functions

- real(kind=c_double) function **fcn** (x1, x2, str, l)

The documentation for this interface was generated from the following file:

- /home/psalinas/Documents/workspace/MultiFluids_Dev/ICFERST/src/IPhreeqc_interface.F90

4.10 xgb_interface::fortran_XGBoosterCreate Interface Reference

Public Member Functions

- integer(c_int) function **fortran_xgboostercreate** (dmats, len, out)

The documentation for this interface was generated from the following file:

- /home/psalinas/Documents/workspace/MultiFluids_Dev/ICFERST/src/xgb_interface.F90

4.11 xgb_interface::fortran_XGBoosterFree Interface Reference

Public Member Functions

- integer(c_int) function **fortran_xgboosterfree** (handle)

The documentation for this interface was generated from the following file:

- /home/psalinas/Documents/workspace/MultiFluids_Dev/ICFERST/src/xgb_interface.F90

4.12 xgb_interface::fortran_XGBoosterLoadModel Interface Reference

Public Member Functions

- integer(c_int) function **fortran_xgboosterloadmodel** (handle, fname)

The documentation for this interface was generated from the following file:

- /home/psalinas/Documents/workspace/MultiFluids_Dev/ICFERST/src/xgb_interface.F90

4.13 xgb_interface::fortran_XGBoosterPredict Interface Reference

Public Member Functions

- integer(c_int) function **fortran_xgboosterpredict** (handle, dmat, option_mask, ntree_limit, training, out_len, out_result)

The documentation for this interface was generated from the following file:

- /home/psalinas/Documents/workspace/MultiFluids_Dev/ICFERST/src/xgb_interface.F90

4.14 xgb_interface::fortran_XGBoosterSaveModel Interface Reference

Public Member Functions

- integer(c_int) function **fortran_xgboostersavemodel** (handle, fname)

The documentation for this interface was generated from the following file:

- /home/psalinas/Documents/workspace/MultiFluids_Dev/ICFERST/src/xgb_interface.F90

4.15 xgb_interface::fortran_XGBoosterSetParam Interface Reference

Public Member Functions

- integer(c_int) function **fortran_xgboostersetparam** (handle, name, value)

The documentation for this interface was generated from the following file:

- /home/psalinas/Documents/workspace/MultiFluids_Dev/ICFERST/src/xgb_interface.F90

4.16 xgb_interface::fortran_XGDMatrixCreateFromMat Interface Reference

Public Member Functions

- integer(c_int) function **fortran_xgdmatrixcreatefrommat** (data, nrow, ncol, missing, out)

The documentation for this interface was generated from the following file:

- /home/psalinas/Documents/workspace/MultiFluids_Dev/ICFERST/src/xgb_interface.F90

4.17 multi_magma::magma_phase_diagram Type Reference

Public Attributes

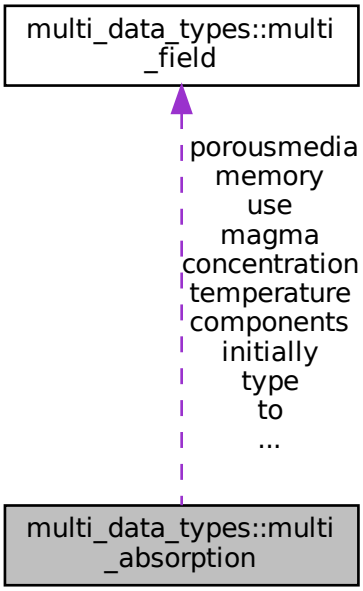
- real **a1**
- real **phase**
- real **behaviour**
- real **parameters**
- real **b1**
- real **c1**
- real **a2**
- real **b2**
- real **c2**
- real **ae**
- real **eutectic**
- real **point**
- real **ts**
- real **solidus**
- real **uniform**
- real **si**
- real **the**
- real **liquidus**
- real **at**
- real **lf**
- real **latent**
- real **heat**

The documentation for this type was generated from the following file:

- /home/psalinas/Documents/workspace/MultiFluids_Dev/ICFERST/src/multi_magma.F90

4.18 multi_data_types::multi_absorption Type Reference

Collaboration diagram for multi_data_types::multi_absorption:



Public Attributes

- type(multi_field) **porousmedia** = Memory_type = 2 -> Fully Anisotropic tensors.
- type(multi_field) components

  Memory_type = 6-> Isotropic Symmetric tensors.
- type(multi_field) **temperature**
- type(multi_field) **concentration**
- type(multi_field) **velocity**
- type(multi_field) magma

  Magma absorption.
- type(multi_field) **initially**
- type(multi_field) **to**
- type(multi_field) **use**
- type(multi_field) **memory**
- type(multi_field) **type** = 3 Isotropic coupled

The documentation for this type was generated from the following file:

- /home/psalinas/Documents/workspace/MultiFluids_Dev/ICFERST/src/multi_data_types.F90

4.19 multi_data_types::multi_dev_shape_funs Type Reference

Public Attributes

- real **volume**
- real, dimension(:, :, :), pointer **of**
- real, dimension(:, :, :), pointer **the**
- real **local**
- real **element**
- real, dimension(:), pointer **detwei** => null()
- real, dimension(:), pointer **determinant**
- real, dimension(:), pointer **times**
- real, dimension(i.e:conversor from reference element to local element), pointer **weigth**
- real, dimension(:), pointer **ra** => null()
- real, dimension(:, :, :), pointer **cvfenx_all** => null()
- real, dimension(:, :, :), pointer **space**
- real, dimension(:, :, :), pointer **derivatives**
- real, dimension(cv), pointer **pressure**
- real, dimension(:, :, :), pointer **shape**
- real, dimension(:, :, :), pointer **functions**
- real, dimension(:, :, :), pointer **ufenx_all** => null()
- real, dimension(fe), pointer **velocity**
- real, dimension(:, :, :), pointer **nx_all** => null()
- real, dimension(:, :, :), pointer **a**
- real, dimension(:, :, :), pointer **generic**
- real, dimension(:, :, :), pointer **field**
- real, dimension(:, :, :), pointer **inv_jac** => null()
- real, dimension(:, :, :), pointer **inverse**
- real, dimension(:, :, :), pointer **jacobian**
- real, dimension(:, :, :), pointer **matrix**

The documentation for this type was generated from the following file:

- /home/psalinas/Documents/workspace/MultiFluids_Dev/ICFERST/src/multi_data_types.F90

4.20 multi_data_types::multi_dimensions Type Reference

Public Attributes

- integer **ndim**
- integer **number**
- integer **of**
- integer **dimensions**
- integer **cv_nloc**
- integer **local**
- integer **control**
- integer **volumes**
- integer **u_nloc**
- integer **velocity**
- integer **nodes**
- integer **cv_snloc**

- integer **on**
- integer **the**
- integer **surface**
- integer **u_snloc**
- integer **nstate**
- integer **states**
- integer **in**
- integer **state**
- integer **ncomp**
- integer **components**
- integer **xu_nloc**
- integer **continuous**
- integer **mesh**
- integer **x_nloc**
- integer **x_snloc**
- integer **x_nloc_p1**
- integer **x_nonods_p1**
- integer **p_nloc**
- integer **pressure**
- integer **p_snloc**
- integer **mat_nloc**
- integer **material**
- integer **totele**
- integer **total**
- integer **elements**
- integer **stotel**
- integer **cv_nonods**
- integer **p_nonods**
- integer **mat_nonods**
- integer **sub**
- integer **u_nonods**
- integer **xu_nonods**
- integer **x_nonods**
- integer **ph_nloc**
- integer **ph_nonods**
- integer **nphase**
- integer **phases**
- integer **npres**
- integer **n_in_pres**
- real **init_time**
- real **initial**
- real **time**

The documentation for this type was generated from the following file:

- /home/psalinas/Documents/workspace/MultiFluids_Dev/ICFERST/src/multi_data_types.F90

4.21 multi_data_types::multi_discretization_opts Type Reference

Public Attributes

- integer **cv_ele_type**
- integer **p_ele_type**
- integer **u_ele_type**
- integer **mat_ele_type**
- integer **u_sele_type**
- integer **cv_sele_type**
- integer **t_disopt**
- integer **v_disopt**
- real **t_beta**
- real **v_beta**
- real **t_theta**
- real **v_theta**
- real **u_theta**
- real **u_beta**
- real **compcoeff**
- real **compoptval**
- integer **t_dg_vel_int_opt**
- integer **u_dg_vel_int_opt**
- integer **v_dg_vel_int_opt**
- integer **w_dg_vel_int_opt**
- integer **in_ele_upwind**
- integer **dg_ele_upwind**
- integer **nits_flux_lim_t**
- integer **nits_flux_lim_volfra**
- integer **nits_flux_lim_comp**
- integer **nits_flux_lim_c**
- logical **volfra_use_theta_flux**
- logical **volfra_get_theta_flux**
- logical **comp_use_theta_flux**
- logical **comp_get_theta_flux**
- logical **t_use_theta_flux**
- logical **t_get_theta_flux**
- logical **scale_momentum_by_volume_fraction**
- logical **compopt**

The documentation for this type was generated from the following file:

- /home/psalinas/Documents/workspace/MultiFluids_Dev/ICFERST/src/multi_data_types.F90

4.22 multi_data_types::multi_field Type Reference

Public Attributes

- real, dimension(:, :, :, :), pointer **val** => null()
- logical **have_field** = .false.
- logical **do**
- logical **we**
- logical **need**
- logical **this**
- logical **field**
- logical **for**
- logical **simulation**
- logical **is_constant** = .false.
- logical, dimension(.true.) **if**
- logical **nonods** = 1 for what follows - DELETE THIS MAYBE ???
- integer **memory_type** = -1
- integer **isotropic**
- integer, dimension(1, 1, nphase, nonods) **tensor**
- integer **this**
- integer **is**
- integer **unrolled**
- integer, dimension(ndim, ndim, nphase, nonods) **as**
- integer ndim1 = -1

  1 Isotropic - (1, 1, nphase, nonods) - diagonal 2 Anisotropic - (ndim, ndim, nphase, nonods) 3 Isotropic coupled - (1, nphase, nphase, nonods) 4 Anisotropic coupled (aka Full Metal Jacket) - (1, ndim x nphase, ndim x nphase, nonods) 6 Isotropic coupled - (1, ndim, nphase, nonods) This is for porous media. We assume isotropic properties like permiability to be diagonal
- integer **ndim2** = -1
- integer **ndim3** = -1
- integer **dimensions**
- integer **of**
- integer **field**

The documentation for this type was generated from the following file:

- /home/psalinas/Documents/workspace/MultiFluids_Dev/ICFERST/src/multi_data_types.F90

4.23 multi_data_types::multi_gi_dimensions Type Reference

Public Attributes

- integer **cv_ngi**
- integer **number**
- integer **of**
- integer **gauss**
- integer **integer**
- integer **points**
- integer **scvngi**
- integer **in**
- integer **the**

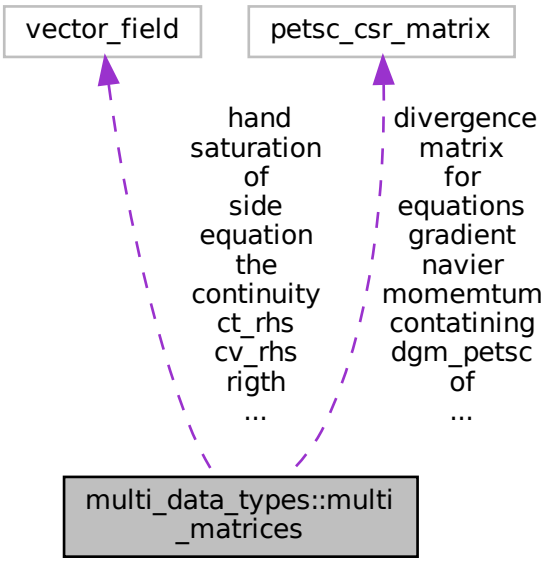
- integer **surface**
- integer **a**
- integer **control**
- integer **volume**
- integer **sbcvngi**
- integer **boundary**
- integer **nface**
- integer **faces**
- integer **per**
- integer **element**

The documentation for this type was generated from the following file:

- /home/psalinas/Documents/workspace/MultiFluids_Dev/ICFERST/src/multi_data_types.F90

4.24 multi_data_types::multi_matrices Type Reference

Collaboration diagram for multi_data_types::multi_matrices:



Public Attributes

- real, dimension(:, :, :), pointer **c** => null()
- real, dimension(:, :, :), pointer **gradient**
- real, dimension(storable), pointer **matrix**
- real, dimension(:, :, :), pointer **using**
- real, dimension(:, :, :), pointer **a**

- real, dimension(assemb_force_cty), pointer **fe**
- real, dimension(storable), pointer **discretization**
- real, dimension(:, :, :), pointer **c_cv** => null()
- real, dimension(cv_assemb), pointer **cv**
- real, dimension(:, :, :), pointer **u_rhs** => null()
- real, dimension(:, :, :), pointer **rigth**
- real, dimension(:, :, :), pointer **hand**
- real, dimension(:, :, :), pointer **side**
- real, dimension(:), pointer **of**
- real, dimension(:), pointer **the**
- real, dimension(:, :, :), pointer **momentum**
- real, dimension(:, :, :), pointer **equation**
- real, dimension(:, :, :), pointer **ct** => null()
- real, dimension(:, :, :), pointer **divergence**
- type(vector_field) **ct_rhs**
- type(vector_field), dimension(:, :, :), pointer **rigth**
- type(vector_field), dimension(:, :, :), pointer **hand**
- type(vector_field), dimension(:, :, :), pointer **side**
- type(vector_field), dimension(:), pointer **of**
- type(vector_field), dimension(:), pointer **the**
- type(vector_field) **continuity**
- type(vector_field), dimension(:, :, :), pointer **equation**
- type(petsc_csr_matrix) **petsc_acv**
- type(petsc_csr_matrix), dimension(storable), pointer **matrix**
- type(petsc_csr_matrix) **containing**
- type(petsc_csr_matrix), dimension(:), pointer **the**
- type(petsc_csr_matrix) **terms**
- type(petsc_csr_matrix), dimension(:), pointer **of**
- type(petsc_csr_matrix) **transport**
- type(petsc_csr_matrix) **equations**
- type(vector_field) **cv_rhs**
- type(vector_field) **saturation**
- real, dimension(:, :, :), pointer **pivit_mat** => null()
- real, dimension(:, :, :), pointer **mass**
- integer, dimension(:), pointer **icolor** => null()
- integer, dimension(:), pointer **array**
- integer, dimension(:), pointer **used**
- integer, dimension(:), pointer **to**
- integer, dimension(:), pointer **accelerate**
- integer, dimension(:), pointer **the**
- integer, dimension(:), pointer **creation**
- integer, dimension(:), pointer **of**
- integer, dimension(:), pointer **cmc**
- integer, dimension(:), pointer **in**
- integer, dimension(:), pointer **color_get_cmc_pha_fast**
- integer **ncolor**
- integer **number**
- integer **colors**
- type(petsc_csr_matrix) **dgm_petsc**
- type(petsc_csr_matrix) **contatining**
- type(petsc_csr_matrix) **momemtum**
- type(petsc_csr_matrix) **for**
- type(petsc_csr_matrix) **navier**
- type(petsc_csr_matrix) **stokes**
- type(petsc_csr_matrix), dimension(:, :, :), pointer **equation**

- type(petsc_csr_matrix) **c_petsc**
- type(petsc_csr_matrix) **petsc**
- type(petsc_csr_matrix) **version**
- type(petsc_csr_matrix), dimension(:, :, :), pointer **gradient**
- type(petsc_csr_matrix) **ct_petsc**
- type(petsc_csr_matrix), dimension(:, :, :), pointer **divergence**
- type(petsc_csr_matrix) **pivit_petsc**
- logical **no_matrix_store**
- logical **flag**
- logical, dimension(:), pointer **to**
- logical **whether**
- logical **calculate**
- logical **and**
- logical **use**
- logical **dgm_petsc**
- logical **or**
- logical, dimension(:, :, :), pointer **c**
- logical **cv_pressure**
- logical, dimension(:), pointer **the**
- logical **pressure**
- logical, dimension(:, :, :), pointer **using**
- logical, dimension(assemb_force_cty), pointer **fe**
- logical, dimension(cv_assemb), pointer **cv**
- logical **stored** = .false.
- logical **be**
- logical **true**
- logical **when**
- logical **storable**
- logical **matrices**
- logical **have**
- logical **been**
- logical **compact_pivit_mat** = .false.
- logical **know**
- logical, dimension(:, :, :), pointer **a**
- logical **compacted**
- logical, dimension(:, :, :), pointer **mass**
- logical, dimension(storable), pointer **matrix**
- logical **not**
- integer, dimension(:), pointer **limiters_elematpsi** => null()
- integer, dimension(:), pointer **stores**
- integer, dimension(:), pointer **locations**
- integer, dimension(:), pointer **by**
- integer, dimension(:), pointer **limiters**
- real, dimension(:), pointer **limiters_elematwei** => null()
- real, dimension(:), pointer **stores**
- real, dimension(:), pointer **weights**
- real, dimension(:), pointer **used**
- real, dimension(:), pointer **by**
- real, dimension(:), pointer **limiters**
- integer, dimension(:,:,:), pointer **wic_flip_p_vel_bcs** => null()
- integer, dimension(:, :), pointer **face_ele** => null()

The documentation for this type was generated from the following file:

- /home/psalinas/Documents/workspace/MultiFluids_Dev/ICFERST/src/multi_data_types.F90

4.25 multi_data_types::multi_ndgln Type Reference

Public Attributes

- integer, dimension(:), pointer **cv** => null()
- integer, dimension(:), pointer **control**
- integer, dimension(:), pointer **volume**
- integer, dimension(:), pointer **local**
- integer, dimension(:), pointer **to**
- integer, dimension(:), pointer **global**
- integer, dimension(:), pointer **numbering**
- integer, dimension(:), pointer **u** => null()
- integer, dimension(:), pointer **velocity**
- integer, dimension(:), pointer **p** => null()
- integer, dimension(:), pointer **pressure**
- integer, dimension(:), pointer **x** => null()
- integer, dimension(:), pointer **continuous**
- integer, dimension(:), pointer **mesh**
- integer, dimension(:), pointer **x_p1** => null()
- integer, dimension(:), pointer **p1**
- integer, dimension(:), pointer **xu** => null()
- integer, dimension(:), pointer **mat** => null()
- integer, dimension(:), pointer **discontinuous**
- integer, dimension(:), pointer **suf_cv** => null()
- integer, dimension(:), pointer **surface**
- integer, dimension(:), pointer **numering**
- integer, dimension(:), pointer **suf_p** => null()
- integer, dimension(:), pointer **suf_u** => null()

The documentation for this type was generated from the following file:

- /home/psalinas/Documents/workspace/MultiFluids_Dev/ICFERST/src/multi_data_types.F90

4.26 multi_data_types::multi_outfluxes Type Reference

strong BCs for P0DG for wells, only necessary if gamma=0 at the BC

Public Attributes

- logical calculate_flux

  Contains variables to analyse the flux across the BCs that the user is interested.
- logical **true**
- logical **if**
- logical **all**
- logical, dimension(:,:), allocatable **the**
- logical **process**
- logical **related**
- logical, dimension(:,:), allocatable **with**
- logical **this**
- logical **has**

- logical **to**
- logical **start**
- logical **or**
- logical **not**
- integer, dimension(:), allocatable **outlet_id**
- integer, dimension(:), allocatable **ids**
- integer, dimension(:,:), allocatable **the**
- integer, dimension(:), allocatable **user**
- integer, dimension(:), allocatable **wants**
- real **porevolume**
- real **for**
- real **outfluxes**
- real **csv**
- real **to**
- real **calculate**
- real, dimension(:,:), allocatable **the**
- real **pore**
- real **volume**
- real **injected**
- real, dimension(mdims%nphase, size(outlet_id)), allocatable **totout**
- real, dimension(fields, mdims%nphase, size(outlet_id)), allocatable **avgout**
- real, dimension(:, :), allocatable **area_outlet**
- real, dimension(:,:), allocatable **mdins%nphase**
- real, dimension(outlet_id), allocatable **size**
- real, dimension(:,:), allocatable **intflux**
- character(len=field_name_len), dimension(:,:), allocatable **field_names**
- character(len=field_name_len), dimension(:,:), allocatable **mdins%nphase**
- character(len=field_name_len), dimension(:,:), allocatable **nfields**
- character(len=field_name_len), dimension(:,:), allocatable **store**
- character(len=field_name_len), dimension(:,:), allocatable **with**
- character(len=field_name_len), dimension(:,:), allocatable **the**
- character(len=field_name_len), dimension(:,:), allocatable **same**
- character(len=field_name_len), dimension(:,:), allocatable **ordering**
- character(len=field_name_len), dimension(:,:), allocatable **field**
- character(len=field_name_len), dimension(:,:), allocatable **names**

4.26.1 Detailed Description

strong BCs for P0DG for wells, only necessary if gamma=0 at the BC

The documentation for this type was generated from the following file:

- /home/psalinas/Documents/workspace/MultiFluids_Dev/ICFERST/src/multi_data_types.F90

4.27 multi_data_types::multi_pipe_package Type Reference

Public Attributes

- real, dimension(:, :, :), pointer gamma_pres_abs => null()

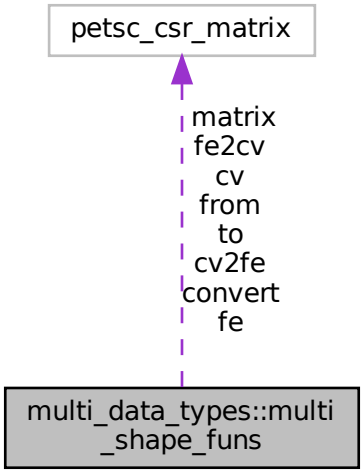
  Contains all the information required to model pipes.
- real, dimension(:), pointer **mass_pipe** => null()
- real, dimension(:), pointer **mass_cvfem2pipe** => null()
- real, dimension(:), pointer **mass_pipe2cvfem** => null()
- real, dimension(:), pointer **mass_cvfem2pipe_true** => null()
- logical, dimension(:), pointer **impose_strongbcs** => null()
- logical, dimension(:), pointer **this**
- logical, dimension(:), pointer **flag**
- logical, dimension(:), pointer **is**
- logical, dimension(:), pointer **used**
- logical, dimension(:), pointer **to**
- logical, dimension(:), pointer **trigger**
- logical, dimension(:), pointer **the**
- logical, dimension(:), pointer **imposition**
- logical, dimension(:), pointer **of**

The documentation for this type was generated from the following file:

- /home/psalinas/Documents/workspace/MultiFluids_Dev/ICFERST/src/multi_data_types.F90

4.28 multi_data_types::multi_shape_funs Type Reference

Collaboration diagram for multi_data_types::multi_shape_funs:



Public Attributes

- real, dimension(:, :), pointer **cvn** => null()
- real, dimension(:, :), pointer **control**
- real, dimension(:, :), pointer **volume**
- real, dimension(:, :), pointer **shape**
- real, dimension(:, :), pointer **function**
- real, dimension(:), pointer **cvweight** => null()
- real, dimension(:), pointer **weigth**
- real, dimension(:, :), pointer **of**
- real, dimension(:, :), pointer **the**
- real, dimension(:, :), pointer **cvfen** => null()
- real, dimension(:, :), pointer **finite**
- real, dimension(:, :), pointer **element**
- real, dimension(:, :, :), pointer **cvfenlx_all** => null()
- real, dimension(cv_nloc ∗scvngi), pointer **dimension**
- real, dimension(:, :), pointer **ufen** => null()
- real, dimension(:, :,:), pointer **ufenlx_all** => null()
- integer, dimension(:, :), pointer **cv_neiloc** => null()
- integer, dimension(cv_nloc ∗scvngi), pointer **dimension**
- logical, dimension(:, :), pointer **cv_on_face** => null()
- logical, dimension(:, :), pointer **cvfem_on_face** => null()
- logical, dimension(cv_nloc ∗scvngi), pointer **dimension**
- real, dimension(:, :), pointer **scvfen** => null()
- real, dimension(:, :), pointer **scvfenslx** => null()
- real, dimension(:, :), pointer **scvfensly** => null()
- real, dimension(:), pointer **scvfeweigh** => null()
- real, dimension(:, :,:), pointer **scvfenlx_all** => null()
- real, dimension(:, :), pointer **sufen** => null()
- real, dimension(:, :), pointer **sufenslx** => null()
- real, dimension(:, :), pointer **sufensly** => null()
- real, dimension(:, :, :), pointer **sufenlx_all** => null()
- logical, dimension(:, :), pointer **u_on_face** => null()
- logical, dimension(:, :), pointer **ufem_on_face** => null()
- real, dimension(:, :), pointer **sbcvn** => null()
- real, dimension(:, :), pointer **sbcvfen** => null()
- real, dimension(:, :), pointer **sbcvfenslx** => null()
- real, dimension(:, :), pointer **sbcvfensly** => null()
- real, dimension(:), pointer **sbcvfeweigh** => null()
- real, dimension(:, :, :), pointer **sbcvfenlx_all** => null()
- real, dimension(:, :), pointer **sbufen** => null()
- real, dimension(:, :), pointer **sbufenslx** => null()
- real, dimension(:, :), pointer **sbufensly** => null()
- real, dimension(:, :,:), pointer **sbufenlx_all** => null()
- integer, dimension(:, :), pointer **cv_sloclist** => null()
- integer, dimension(:, :), pointer **u_sloclist** => null()
- integer, dimension(:), pointer **findgpts** => null()
- integer, dimension(:), pointer **colgpts** => null()
- integer **ncolgpts**
- type(petsc_csr_matrix) **cv2fe**
- type(petsc_csr_matrix) **matrix**
- type(petsc_csr_matrix) **to**
- type(petsc_csr_matrix) **convert**
- type(petsc_csr_matrix) **from**
- type(petsc_csr_matrix) **cv**

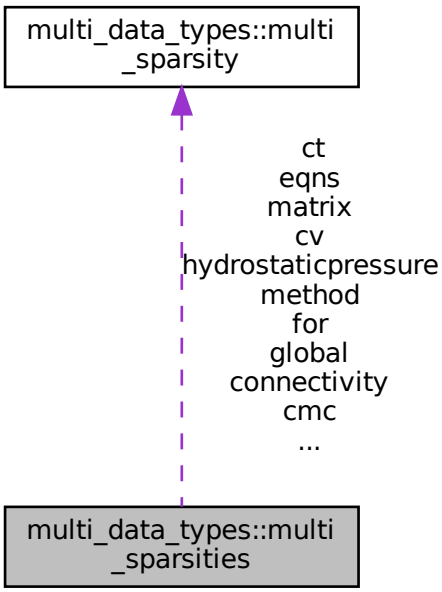
- type(petsc_csr_matrix) **fe**
- type(petsc_csr_matrix) **fe2cv**

The documentation for this type was generated from the following file:

- /home/psalinas/Documents/workspace/MultiFluids_Dev/ICFERST/src/multi_data_types.F90

4.29 multi_data_types::multi_sparsities Type Reference

Collaboration diagram for multi_data_types::multi_sparsities:



Public Attributes

- type(multi_sparsity) **acv**
- type(multi_sparsity) **cv**
- type(multi_sparsity) **multi**
- type(multi_sparsity) **phase**
- type(multi_sparsity), dimension(e.g. vol frac, temp) **eqns**
- type(multi_sparsity) **small_acv**
- type(multi_sparsity) **local**
- type(multi_sparsity) **ele**
- type(multi_sparsity) **element**
- type(multi_sparsity) **connectivity**
- type(multi_sparsity) **dgm_pha**
- type(multi_sparsity) **force**

- type(multi_sparsity) **balance**
- type(multi_sparsity) **sparsity**
- type(multi_sparsity) **ct**
- type(multi_sparsity) **global**
- type(multi_sparsity) **continuity**
- type(multi_sparsity) **eqn**
- type(multi_sparsity) **c**
- type(multi_sparsity) **operating**
- type(multi_sparsity) **on**
- type(multi_sparsity) **pressure**
- type(multi_sparsity) **in**
- type(multi_sparsity) **cmc**
- type(multi_sparsity) **matrix**
- type(multi_sparsity) **for**
- type(multi_sparsity) **projection**
- type(multi_sparsity) **method**
- type(multi_sparsity) **m**
- type(multi_sparsity) **fem**
- type(multi_sparsity) **hydrostaticpressure**

The documentation for this type was generated from the following file:

- /home/psalinas/Documents/workspace/MultiFluids_Dev/ICFERST/src/multi_data_types.F90

4.30 multi_data_types::multi_sparsity Type Reference

Public Attributes

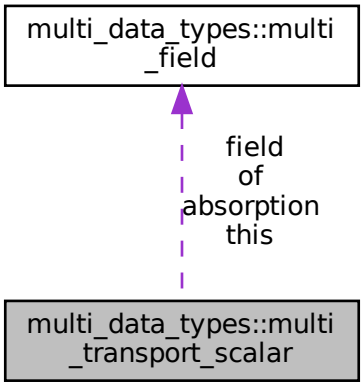
- integer **ncol**
- integer, dimension(:), pointer **fin** => null()
- integer, dimension(:), pointer **col** => null()
- integer, dimension(:), pointer **mid** => null()

The documentation for this type was generated from the following file:

- /home/psalinas/Documents/workspace/MultiFluids_Dev/ICFERST/src/multi_data_types.F90

4.31 multi_data_types::multi_transport_scalar Type Reference

Collaboration diagram for multi_data_types::multi_transport_scalar:



Public Attributes

- character(len=field_name_len) **name**
- character(len=field_name_len) **to**
- character(len=field_name_len) **extract**
- character(len=field_name_len) **from**
- character(len=field_name_len) **state**
- character(len=option_path_len) **path**
- character(len=option_path_len) **from**
- character(len=option_path_len) **diamond**
- logical **coupled_field**
- logical **is**
- logical **the**
- logical **field**
- logical **coupled**
- logical **between**
- logical **phases**
- type(multi_field) **absorption**
- type(multi_field) **of**
- type(multi_field) **this**
- type(multi_field) **field**

The documentation for this type was generated from the following file:

- /home/psalinas/Documents/workspace/MultiFluids_Dev/ICFERST/src/multi_data_types.F90

4.32 cv_advection::pack_loc_all Interface Reference

Public Member Functions

- subroutine **pack_loc_all1** (LOC_F, field1, oldfield1, field2, oldfield2, field3, oldfield3, IGOT_T_PACK, use↩ _volume_frac_T2, nfield)
- subroutine **pack_loc_all2** (LOC_F, field1, oldfield1, field2, oldfield2, field3, oldfield3, IGOT_T_PACK, use↩ _volume_frac_T2, start_phase, final_phase, nodi)
- subroutine **pack_loc_all3** (LOC_F, field1, oldfield1, field2, oldfield2, field3, oldfield3, IGOT_T_PACK, use↩ _volume_frac_T2, start_phase, final_phase, nodi)

The documentation for this interface was generated from the following file:

- /home/psalinas/Documents/workspace/MultiFluids_Dev/ICFERST/src/cv-adv-dif.F90

4.33 multi_data_types::pipe_coords Type Reference

Public Attributes

- integer **ele**
- integer, dimension(:), allocatable **npipes**
- integer **element**
- integer **containing**
- integer, dimension(:), allocatable **pipes**
- integer **per**
- logical, dimension(:), allocatable **pipe_index**
- logical, dimension(:), allocatable **nodes**
- logical, dimension(:), allocatable **with**
- logical, dimension(:), allocatable **pipes**
- integer, dimension(:), allocatable **pipe_corner_nds1**
- integer, dimension(:), allocatable **size**
- integer, dimension(:), allocatable **pipe_corner_nds2**

The documentation for this type was generated from the following file:

- /home/psalinas/Documents/workspace/MultiFluids_Dev/ICFERST/src/multi_data_types.F90

4.34 multi_data_types::porous_adv_coefs Type Reference

Public Attributes

- real, dimension(:, :, :, :), pointer **adv_coef** => null()
- real, dimension(:, :, :), pointer **sigmas**
- real, dimension(:, :, :), pointer **at**
- real, dimension(:, :, :), pointer **the**
- real, dimension(:, :, :), pointer **boundary**
- real, dimension(:, :, :), pointer **to**
- real, dimension(:, :, :), pointer **calculate**

- real, dimension(:, :, :), pointer **fluxes**
- real, dimension(:, :, :, :), pointer **inv_adv_coef** => null()
- real, dimension(:, :, :, :), pointer **inverse**
- real, dimension(:, :, :), pointer **of**
- real, dimension(:, :, :, :), pointer **adv_coef_grad** => null()
- real, dimension(:, :, :), pointer **gradient**
- real, dimension(:, :, :), pointer **inv_permeability** => null()

The documentation for this type was generated from the following file:

- /home/psalinas/Documents/workspace/MultiFluids_Dev/ICFERST/src/multi_data_types.F90

4.35 SetBasicFortranCallbackF Interface Reference

Public Member Functions

- integer(kind=c_int) function **setbasicfortrancallbackf** (id, fcn)

Public Attributes

- integer, intent(in) **id**

The documentation for this interface was generated from the following file:

- /home/psalinas/Documents/workspace/MultiFluids_Dev/ICFERST/src/IPhreeqc_interface.F90

4.36 shape_functions_ndim::xprod Interface Reference

Public Member Functions

- subroutine **xprod1** (AX, AY, AZ, BX, BY, BZ, CX, CY, CZ)
- subroutine **xprod2** (A, B, C)

The documentation for this interface was generated from the following file:

- /home/psalinas/Documents/workspace/MultiFluids_Dev/ICFERST/src/multi_shape_fct_ND.F90

Index