文部科学省次世代ＩＴ基盤構築のための研究開発
「革新的シミュレーションソフトウエアの研究開発」

RSS21 フリーソフトウエア

HEC ミドルウェア（HEC-MW）

PC クラスタ用ライブラリ型 HEC-MW

（hecmw-PC-cluster） バージョン 2.01

API リファレンス

お問い合わせ先
（公開／契約窓口） （財）生産技術研究奨励会
〒153-8505 東京都目黒区駒場４－６－１
（ソフトウェア管理元） 東京大学生産技術研究所 計算科学技術連携研究センター
〒153-8505 東京都目黒区駒場４－６－１
Fax ： 03-5452-6662
E-mail ： software@rss21.iis.u-tokyo.ac.jp

目　　次

1. I/O, 可視化

本節では、「I/O」「HEC-MW の初期化、終了処理」および「メモリ渡し可視化」に関する API について説明します。

hecmw_init

HEC-MW の初期化処理を行います。

```
subroutine hecmw_init()
```

引数

なし

説明

HEC-MW の初期化処理を行います。

これは、プログラムの開始直後に必ず呼び出さねばなりません。

MPI の初期化を行うため、この呼び出し以降、並列処理が可能となります。また、全体制御ファイルが自動的に読み込まれます。

hecmw_finalize

HEC-MW の終了処理を行います。

```
subroutine hecmw_finalize
```

引数

なし

説明

HEC-MW の終了処理を行います。
これは、プログラムの終了直前に呼び出さねばなりません。
MPI の終了処理はこの呼び出しで行われます。

hecmw_get_mesh

メッシュデータをファイルから読み込みます。

```
subroutine hecmw_get_mesh(name_ID, mesh)

character(len=HECMW_NAME_LEN) :: name_ID
type(hecmwST_local_mesh) :: mesh
```

引数

name_ID

!MESH または!MESH GROUP を特定する識別子

mesh

読み込まれたメッシュデータの格納先

説明

ファイルからメッシュデータを読み込みます。

この API は、全体制御ファイルから入力ファイルの情報を取得します。

読み込み可能なメッシュファイルの種類は以下のとおりです。

- HEC-MW 分散メッシュデータ
- HEC-MW 単一領域メッシュデータ
- GeoFEM メッシュデータ
- ABAQUS メッシュデータ

メッシュファイルの種類は全体制御ファイルで指定します。

読み込むメッシュファイルは、全体制御ファイルの!MESH で定義されており、かつ NAME が name_ID のものです。!MESH GROUP によってメッシュファイルがグループ化されている場合は、グループ内の全てのメッシュファイルを読み込みます。

読み込まれるメッシュタイプが分散メッシュデータの場合、実際に読み込むファイルのファイル名は、全体制御ファイルから取得したファイル名の末尾に「.<ランク番号>」を付加したものとなります。

hecmw_dist_free

分散メッシュデータ構造体に確保されているメモリを解放します。

```
subroutine hecmw_dist_free(mesh)

type(hecmwST_local_mesh) :: mesh
```

引数

mesh

解放する分散メッシュデータ構造体

説明

分散メッシュデータ構造体に確保されているメモリを解放します。

hecmw_put_mesh

メッシュデータをファイルに出力します。

```
subroutine hecmw_put_mesh(name_ID, mesh)

character(len=HECMW_NAME_LEN) :: name_ID
type(hecmwST_local_mesh) :: mesh
```

引数

name_ID

!MESH を特定する識別子

mesh

出力するメッシュデータ

説明

メッシュデータをファイルに出力します。

出力対象となるメッシュデータは、引数 mesh に格納されているデータで、出力ファイルの種類は分散メッシュデータとなります。

出力されるファイルは、全体制御ファイルの!MESH で定義されており、かつ NAME が name_ID のものです。

出力されるファイルのファイル名は、全体制御ファイルから取得したファイル名の末尾に「.<ランク番号>」を付加したものとなります。

hecmw_result_init

結果ファイル出力の初期化処理を行います。

```
subroutine  hecmw_result_init(nnode, nelelem, tstep, header)

integer(kind=kint) :: nnode
integer(kind=kint) :: nelem
integer(kind=kint) :: tstep
character(len=HECMW_HEADER_LEN) :: header
```

引数

nnode
: 節点数

nelem
: 要素数

tstep
: タイムステップ

header
: ヘッダ

説明

結果ファイル出力の初期化処理を行います。

これにより、結果ファイル出力のために必要な情報を取得します。ただし、header は結果ファイルにコメントとして用いられるだけで、何ら影響を与えません。

hecmw_result_add

結果ファイルに出力するデータを指定します。

```
subroutine hecmw_result_add(node_or_elem, n_dof, label, data)

integer(kind=kint) :: node_or_elem
integer(kind=kint) :: n_dof
character(len=HECMW_NAME_LEN) :: label
real(kind=kreal) :: data
```

引数

node_or_elem

　　指定する値が節点値なのか要素値なのかを示す

　　1:節点　2:要素

n_dof

　　自由度数

label

　　ラベル

data

　　結果データ

説明

結果ファイルに出力するデータを指定します。

これは、複数回呼び出すことが可能です。この呼び出しによって指定されたデータの情報は、一旦 HEC-MW の内部に蓄えられます。蓄えられたデータは、hecmw_result_write または hecmw_result_write_by_name によって出力されます。

この呼び出し以前に、hecmw_result_init によって初期化が行われていなければなりません。

hecmw_result_write

結果データをファイルへ出力します。

```
subroutine hecmw_result_write()
```

引数

なし

説明

結果データをファイルへ出力します。
出力されるデータは、hecmw_result_add で指定されたデータです。
出力されるファイルは、全体制御ファイルの!RESULT で定義されており、IO=OUT かつ全体制御ファイル内で最初に定義されているファイルです。
出力されるファイルのファイル名は、全体制御ファイルから取得したファイル名の末尾に「.<ランク番号>.<tstep>」を付加したものとなります。

hecmw_result_write_by_name

結果データをファイルへ出力します。

```
subroutine hecmw_result_write_by_name(name_ID)

character(len=HECMW_NAME_LEN) :: name_ID
```

引数

name_ID

!RESULT を特定する識別子

説明

結果データをファイルへ出力します。

出力ファイルは、全体制御ファイルで指定された!RESULT のうち、NAME が name_ID のものです。この場合、!RESULT の IO パラメータは無視されます。

出力されるファイルのファイル名は、全体制御ファイルから取得したファイル名の末尾に「.<ランク番号>.<tstep>」を付加したものとなります。

ファイル名の取得方法以外は、hecmw_result_write と同等です。

hecmw_result_write_st

結果データをファイルへ出力します。

```
subroutine hecmw_result_write_st(result_data, n_node, n_elem, tstep,
header)

type(hecmwST_result_data)::result_data
integer(kind=kint)::n_node,n_elem,tstep
character(len=HECMW_HEADER_LEN)::header
```

引数

result_data

結果データ構造体

n_node

節点数

n_elem

要素数

tstep

タイムステップ

header

結果ファイルヘッダ

説明

結果データをファイルへ出力します。

出力されるデータは、result_data で指定されたデータです。

出力されるファイルは、全体制御ファイルの!RESULT で定義されており、IO=OUT かつ全体制御ファイル内で最初に定義されているファイルです。

出力されるファイルのファイル名は、全体制御ファイルから取得したファイル名の末尾に「.<ランク番号>.<tstep>」を付加したものとなります。

hecmw_result_write_st_by_name

結果データをファイルへ出力します。

```
subroutine    hecmw_result_write_st_by_name(name_ID,    result_data,
n_node, n_elem, tstep, header)

character(len=HECMW_NAME_LEN) :: name_ID
type(hecmwST_result_data)::result_data
integer(kind=kint)::n_node,n_elem,tstep
character(len=HECMW_HEADER_LEN)::header
```

引数

name_ID

!RESULT を特定する識別子

result_data

結果データ構造体

n_node

節点数

n_elem

要素数

tstep

タイムステップ

header

結果ファイルヘッダ

説明

引数 result_data の結果データをファイルへ出力します。

出力ファイルは、全体制御ファイルで指定された!RESULT のうち、NAME が name_ID のものです。この場合、!RESULT の IO パラメータは無視されます。

出力されるファイルのファイル名は、全体制御ファイルから取得したファイル名の末尾に「.<ランク番号>.<tstep>」を付加したものとなります。

hecmw_result_finalize

結果データ出力の後処理を行います。

```
subroutine hecmw_result_finalize()
```

引数

なし

説明

hecmw_result_add で指定された結果データの登録をクリアします。

hecmw_result_read

結果ファイルから結果データを入力します。

```
subroutine hecmw_result_read(tstep, result)

integer(kind=kint) :: tstep
type(hecmwST_result_data) :: result
```

引数

tstep

タイムステップ

result

結果データ格納用構造体

説明

結果ファイル内の結果データが読み込まれ、result に格納されます。

入力されるファイルは、全体制御ファイルの!RESULT で定義されており、IO=IN かつ全体制御ファイル内で最初に定義されているファイルです。

入力されるファイルのファイル名は、全体制御ファイルから取得したファイル名の末尾に「.<ランク番号>.<tstep>」を付加したものとなります。

hecmw_result_read_by_name

結果ファイルから結果データを入力します。

```
subroutine hecmw_result_read_by_name(name_ID, tstep, result)

character(len=HECMW_NAME_LEN) :: name_ID
integer(kind=kint) :: tstep
type(hecmwST_result_data) :: result
```

引数

name_ID

!RESULT を特定する識別子

tstep

タイムステップ

result

結果データ格納用構造体

説明

結果ファイル内の結果データが読み込まれ、result に格納されます。

入力ファイルは、全体制御ファイルで指定された!RESULT のうち、NAME が name_ID のものです。この場合、!RESULT の IO パラメータは無視されます。

ファイル名の取得方法以外は、hecmw_result_read と同等です。

hecmw_restart_add_int

リスタートファイルに出力するデータを指定します。

```
subroutine hecmw_restart_add_int(data, n_data)

integer(kind=kint),dimension(:) :: data
integer(kind=kint) :: n_data
```

引数

data

リスタートデータ（整数型一次元配列）

n_data

配列要素数

説明

リスタートファイルに出力するデータを指定します。

これは、複数回呼び出すことが可能です。この呼び出しによって指定されたデータの情報は、一旦 HEC-MW の内部に蓄えられます。蓄えられたデータは、hecmw_restart_write または hecmw_restart_write_by_name によって出力されます。したがって、実際に出力が完了するまでデータを変更してはなりません。

hecmw_restart_add_real

リスタートファイルに出力するデータを指定します。

```
subroutine hecmw_restart_add_int(data, n_data)

integer(kind=kint),dimension(:) :: data
integer(kind=kint) :: n_data
```

引数

data

リスタートデータ（浮動小数点型一次元配列）

n_data

配列要素数

説明

引数の型が浮動小数点型一次元配列であること以外は、hecmw_restart_add_real と同等です。

hecmw_restart_write

リスタートデータをファイルへ出力します。

```
subroutine hecmw_restart_write()
```

引数

なし

説明

リスタートデータをファイルへ出力します。

出力されるデータは、hecmw_restart_add_int または hecmw_restart_real で指定されたデータです。

出力されるファイルは、全体制御ファイルの!RESTART で定義されており、IO=OUT または IO=INOUT、かつ全体制御ファイル内で最初に定義されているファイルです。

出力されるファイルのファイル名は、全体制御ファイルから取得したファイル名の末尾に「.<ランク番号>」を付加したものとなります。

hecmw_restart_write_by_name

リスタートデータをファイルへ出力します。

```
subroutine hecmw_restart_write_by_name(name_ID)

character(len=HECMW_NAME_LEN) :: name_ID
```

引数

name_ID

!RESTART を特定する識別子

説明

リスタートデータをファイルへ出力します。

出力ファイルは、全体制御ファイルで指定された!RESTART のうち、NAME が name_ID のものです。この場合、!RESTART の IO パラメータは無視されます。

出力されるファイルのファイル名は、全体制御ファイルから取得したファイル名の末尾に「.<ランク番号>」を付加したものとなります。

ファイル名の取得方法以外は、hecmw_restart_write と同等です。

hecmw_restart_open

リスタートファイルを入力用にオープンします。

```
hecmw_restart_open()
```

引数

なし

説明

リスタートファイルを入力用にオープンします。
オープンされるファイルは、全体制御ファイルの!RESTART で定義されており、IO=IN または IO=INOUT、かつ全体制御ファイル内で最初に定義されているファイルです。

hecmw_restart_open_by_name

リスタートファイルを入力用にオープンします。

```
subroutine hecmw_restart_open_by_name(name_ID)

character(len=HECMW_NAME_LEN) :: name_ID
```

引数

name_ID

!RESTART を特定する識別子

説明

リスタートファイルを入力用にオープンします。

出力ファイルは、全体制御ファイルで指定された!RESTART のうち、NAME が name_ID のものです。この場合、!RESTART の IO パラメータは無視されます。

hecmw_restart_read_int

リスタートデータからデータを入力します。

```
subroutine hecmw_restart_read_int(dst)

integer(kind=kint), dimension(:) :: dst
```

引数

dst

リスタートデータ格納先（整数型一次元配列）

説明

リスタートファイル内のデータの一部が読み込まれ、dst に格納されます。

この呼び出し以前に、hecmw_restart_open または hecmw_restart_open_by_name によって入力リスタートファイルがオープンされていなければなりません。

dst には、データ格納に必要な領域を事前に確保しておかなければなりません。

この呼び出しで注意すべきことは、リスタートファイルの入力と出力で整合性がとれていなければならないということです。

リスタートファイルの入力には hecmw_restart_write_int または hecmw_restart_write_real を使用しますが、その順番とそれぞれの出力で出力したデータサイズに入力時もあわせる必要があります。つまり、hecmw_restart_write_int で出力したデータは、hecmw_restart_read_int で入力せねばならず、かつその配列の要素サイズを同じにしておかなければなりません。これは、hecmw_restart_write_real, hecmw_restart_read_real についても同じです。

hecmw_restart_read_real

リスタートデータからデータを入力します。

```
subroutine hecmw_restart_read_real(dst)

real(kind=kreal),dimension(:) :: dst
```

引数

dst

リスタートデータ格納先（浮動小数点型一次元配列）

説明

引数の型が浮動小数点型一次元配列であること以外は、hecmw_restart_read_real と同等です。

hecmw_restart_close

リスタートファイルをクローズします。

```
subroutine hecmw_restart_close()
```

引数

なし

説明

オープンされているリスタートファイルをクローズします。

hecmw_visualize_init

可視化の初期化処理を行います。

```
subroutine hecmw_visualize_init()
```

引数

なし

説明

可視化の初期化処理を行います。

hecmw_visualize(mesh, result, tstep, max_step, is_force)

可視化を行います。

```
subroutine hecmw_visualize(mesh, result, tstep, max_step, is_force)

type(hecmwST_local_mesh) :: mesh
type(hecmwST_result_data) :: result
integer(kind=kint) :: tstep
integer(kind=kint) :: max_step
integer(kind=kint) :: is_force
```

引数

mesh

メッシュデータ

result

可視化用結果データ

t step

タイムステップ

max_step

最大タイムステップ

is_force

最後のステップも強制的に描くかどうかを示す。

0:描かない　1:描く

説明

可視化を行います。

hecmw_visualize_init によって事前に初期化されている必要があります。

可視化に必要な結果データには result、メッシュには mesh が使用されます。

hecmw_visualize_finalize

可視化の後処理を行います。

```
subroutine hecmw_visualize_finalize
```

引数

なし

説明

可視化の後処理を行います。

hecmw_ctrl_get_control_file

制御ファイル名を取得します。

```
subroutine hecmw_ctrl_get_control_file(name_ID, filename)

character(len=HECMW_NAME_LEN) :: name_ID
character(len=HECMW_FILENAME_LEN) :: filename
```

引数

name_ID

!CONTROL を特定する識別子

filename

制御ファイル名格納先

説明

全体制御ファイルの!CONTROL で定義した制御ファイル名を取得します。

2. 線形ソルバ

本節では「線形ソルバ」「マトリクス・ベクトル演算」および「領域間通信処理」に関するAPIについて説明します。

hecmw_solve_11

線形ソルバ（1 節点あたり 1 自由度）を呼び出します。

```
subroutine hecmw_solve_11(mesh, matrix)

type(hecmwST_local_mesh) :: mesh
type(hecmwST_matrix    ) :: matrix
```

引数

mesh

メッシュデータの格納先

matrix

マトリクスデータおよびソルバ制御情報の格納先

説明

- 線形ソルバ（1 節点あたり 1 自由度）を呼び出します。
- 係数マトリクスに関する情報，右辺ベクトル，ソルバ制御情報は全て「matrix」に格納されています。
- 「matrix」の内容については，「2.2」をご覧ください。

hecmw_solve_22

線形ソルバ（1 節点あたり 2 自由度）を呼び出します。

```
subroutine hecmw_solve_22(mesh, matrix)

type(hecmwST_local_mesh) :: mesh
type(hecmwST_matrix    ) :: matrix
```

引数

mesh

メッシュデータの格納先

matrix

マトリクスデータおよびソルバ制御情報の格納先

説明

- 線形ソルバ（1 節点あたり 2 自由度）を呼び出します。
- 係数マトリクスに関する情報，右辺ベクトル，ソルバ制御情報は全て「matrix」に格納されています。
- 「matrix」の内容については，「2.2」をご覧ください。

hecmw_solve_33

線形ソルバ（1 節点あたり 3 自由度）を呼び出します。

```
subroutine hecmw_solve_33(mesh, matrix)

type(hecmwST_local_mesh) :: mesh
type(hecmwST_matrix    ) :: matrix
```

引数

mesh

メッシュデータの格納先

matrix

マトリクスデータおよびソルバ制御情報の格納先

説明

- 線形ソルバ（1 節点あたり 3 自由度）を呼び出します。
- 係数マトリクスに関する情報，右辺ベクトル，ソルバ制御情報は全て「matrix」に格納されています。
- 「matrix」の内容については，「2.2」をご覧ください。

hecmw_solve_direct

直接法ソルバを呼び出します。

```
subroutine hecmw_solve_direct(mesh, matrix, ifmsg)

type(hecmwST_local_mesh) :: mesh
type(hecmwST_matrix    ) :: matrix
integer(kind=kint) :: ifmsg
```

引数

mesh
: メッシュデータの格納先

matrix
: マトリクスデータおよびソルバ制御情報の格納先

ifmsg
: エラー時のメッセージを出力するデバイス番号

説明

- 直接法を呼び出します。
- 係数マトリクスに関する情報，右辺ベクトル，ソルバ制御情報は全て「matrix」に格納されています。
- 「matrix」の内容については，「2.2」をご覧ください。

注

本バージョンでは並列実行時には機能いたしません.

hecmw_solve_direct_parallel

並列直接法ソルバを呼び出します。

```
subroutine hecmw_solve_direct_parallel(mesh, matrix, ifmsg)

type(hecmwST_local_mesh) :: mesh
type(hecmwST_matrix    ) :: matrix
integer(kind=kint) :: ifmsg
```

引数

mesh
: メッシュデータの格納先

matrix
: マトリクスデータおよびソルバ制御情報の格納先

ifmsg
: エラー時のメッセージを出力するデバイス番号

説明

- 並列直接法を呼び出します。
- 係数マトリクスに関する情報，右辺ベクトル，ソルバ制御情報は全て「matrix」に格納されています。
- 「matrix」の内容については，「2.2」をご覧ください。

hecmw_barrier

MPI プロセスの同期のための Barrier を設定します。

```
subroutine hecmw_barrier(mesh)

type(hecmwST_local_mesh) :: mesh
```

引数

mesh

対象とするコミュニケーショングループのメッシュデータの格納先

説明

- 内部からは，MPI_BARRIER を呼び出しています。

hecmw_allREDUCE_R

実数値または実数ベクトルの各成分に関してグローバルな演算を実施します。

```
subroutine hecmw_allREDUCE_R(mesh, VAL, m, FLAG)

type(hecmwST_local_mesh) :: mesh
integer(kind=kint) :: m, FLAG

real(kind=kreal) :: VAL
   or
real(kind=kreal), dimension(m) :: VAL
```

引数

mesh

対象とするコミュニケーショングループのメッシュデータの格納先

VAL

対象とする実数ベクトル（またはスカラー）

m

ベクトルのサイズ（スカラーの場合は 1）

FLAG

グローバル演算の識別子

hecmw_sum	総　和
hecmw_min	最小値
hecmw_max	最大値

説明

- 内部からは，MPI_ALLREDUCE を呼び出しています。

hecmw_allREDUCE_I

整数値または整数ベクトルの各成分に関してグローバルな演算を実施します。

```
subroutine hecmw_allREDUCE_I(mesh, VAL, m, FLAG)

type(hecmwST_local_mesh) :: mesh
integer(kind=kint) :: m, FLAG

integer(kind=kint) :: VAL
   or
integer(kind=kint), dimension(m) :: VAL
```

引数

mesh

対象とするコミュニケーショングループのメッシュデータの格納先

VAL

対象とする整数ベクトル（またはスカラー）

m

ベクトルのサイズ（スカラーの場合は 1）

FLAG

グローバル演算の識別子

hecmw_sum	総　和
hecmw_min	最小値
hecmw_max	最大値

説明

- 内部からは，MPI_ALLREDUCE を呼び出しています。

hecmw_bcast_R

実数値または実数ベクトル値を全プロセッサに送信します。

```
subroutine hecmw_bcast_R(mesh, VAL, m, nbase)

type(hecmwST_local_mesh) :: mesh
integer(kind=kint) :: m, nbase

real(kind=kreal) :: VAL
   or
real(kind=kreal), dimension(m) :: VAL
```

引数

mesh
: 対象とするコミュニケーショングループのメッシュデータの格納先

VAL
: 対象とする実数ベクトル（またはスカラー）

m
: ベクトルのサイズ（スカラーの場合は 1）

nbase
: 発信元の PE のプロセス ID（0 から開始）

説明

- 内部からは，MPI_BCAST を呼び出しています。

hecmw_bcast_I

整数値または整数ベクトル値を全プロセッサに送信します。

```
subroutine hecmw_bcast_I(mesh, VAL, m, nbase)

type(hecmwST_local_mesh) :: mesh
integer(kind=kint) :: m, nbase

integer(kind=kint) :: VAL
   or
integer(kind=kint) , dimension(m) :: VAL
```

引数

mesh
: 対象とするコミュニケーショングループのメッシュデータの格納先

VAL
: 対象とする整数ベクトル（またはスカラー）

m
: ベクトルのサイズ（スカラーの場合は 1）

nbase
: 発信元の PE のプロセス ID（0 から開始）

説明

- 内部からは，MPI_BCAST を呼び出しています。

hecmw_bcast_C

Character 変数値または Character 変数ベクトル値を全プロセッサに送信します。

```
subroutine hecmw_bcast_I(mesh, VAL, m, LEN, nbase)

type(hecmwST_local_mesh) :: mesh
integer(kind=kint) :: m, length, nbase

character (length=LEN) :: VAL
   or
character (length=LEN), dimension(m) :: VAL
```

引数

mesh
: 対象とするコミュニケーショングループのメッシュデータの格納先

VAL
: 対象とする Character 変数ベクトル（またはスカラー）

m
: ベクトルのサイズ（スカラーの場合は 1）

LEN
: Character 変数の文字長

nbase
: 発信元の PE のプロセス ID（0 から開始）

説明

- 内部からは，MPI_BCAST を呼び出しています。

hecmw_update_1_R

オーバーラップ領域における実数ベクトル（1 節点あたり 1 自由度）の値を，領域間通信によって更新します。

```
subroutine hecmw_update_1_R (mesh, VAL, N)

type(hecmwST_local_mesh) :: mesh
integer(kind=kint) :: N
real(kind=kreal), dimension(N) :: VAL
```

引数

mesh
: メッシュデータの格納先

VAL
: 実数ベクトル値

N
: ベクトルのサイズ

説明

hecmw_update_2_R

オーバーラップ領域における実数ベクトル（1 節点あたり 2 自由度）の値を，領域間通信によって更新します。

```
subroutine hecmw_update_2_R(mesh, VAL, N)

type(hecmwST_local_mesh) :: mesh
integer(kind=kint) :: N
real(kind=kreal), dimension(2*N) :: VAL
```

引数

mesh

メッシュデータの格納先

VAL

実数ベクトル値

N

ベクトルのサイズ

説明

hecmw_update_3_R

オーバーラップ領域における実数ベクトル（1 節点あたり 3 自由度）の値を，領域間通信によって更新します。

```
subroutine hecmw_update_3_R(mesh, VAL, N)

type(hecmwST_local_mesh) :: mesh
integer(kind=kint) :: N
real(kind=kreal), dimension(3*N) :: VAL
```

引数

mesh

メッシュデータの格納先

VAL

実数ベクトル値

N

ベクトルのサイズ

説明

hecmw_update_m_R

オーバーラップ領域における実数ベクトル（1 節点あたり m 自由度）の値を，領域間通信によって更新します。

```
subroutine hecmw_update_2_R(mesh, VAL, N, m)

type(hecmwST_local_mesh) :: mesh
integer(kind=kint) :: N, m
real(kind=kreal), dimension(2*N) :: VAL
```

引数

mesh

メッシュデータの格納先

VAL

実数ベクトル値

N

ベクトルのサイズ

m

節点あたりの自由度数

説明

hecmw_matvec_11

実数ベクトル（1 節点あたり 1 自由度）の行列ベクトル積を求めます。

```
subroutine hecmw_matvec_11 (mesh, matrix, X, Y, N)

type(hecmwST_local_mesh) :: mesh
type(hecmwST_matrix)     :: matrix
integer(kind=kint) :: N
real(kind=kreal), dimension(N) :: X, Y
```

引数

mesh
: メッシュデータの格納先

matrix
: マトリクスデータの格納先

X
: 行列を乗じるベクトル

Y
: 行列ベクトル積の結果ベクトル

N
: ベクトルのサイズ

説明

- 係数マトリクスに関する情報，は全て「matrix」に格納されています。
- 内部から「hecmw_update_1_R」を呼び出しています。

hecmw_matvec_22

実数ベクトル（1 節点あたり 2 自由度）の行列ベクトル積を求めます。

```
subroutine hecmw_matvec_22 (mesh, matrix, X, Y, N)

type(hecmwST_local_mesh) :: mesh
type(hecmwST_matrix)     :: matrix
integer(kind=kint) :: N
real(kind=kreal), dimension(2*N) :: X, Y
```

引数

mesh
: メッシュデータの格納先

matrix
: マトリクスデータの格納先

X
: 行列を乗じるベクトル

Y
: 行列ベクトル積の結果ベクトル

N
: ベクトルのサイズ

説明

- 係数マトリクスに関する情報，は全て「matrix」に格納されています。
- 内部から「hecmw_update_2_R」を呼び出しています。

hecmw_matvec_33

実数ベクトル（1 節点あたり 3 自由度）の行列ベクトル積を求めます。

```
subroutine hecmw_matvec_33 (mesh, matrix, X, Y, N)

type(hecmwST_local_mesh) :: mesh
type(hecmwST_matrix)     :: matrix
integer(kind=kint) :: N
real(kind=kreal), dimension(3*N) :: X, Y
```

引数

mesh

メッシュデータの格納先

matrix

マトリクスデータの格納先

X

行列を乗じるベクトル

Y

行列ベクトル積の結果ベクトル

N

ベクトルのサイズ

説明

- 係数マトリクスに関する情報，は全て「matrix」に格納されています。
- 内部から「hecmw_update_3_R」を呼び出しています。

hecmw_innerProduct_I

整数ベクトルの内積を求めます。

```
subroutine hecmw_innerProduct_I (mesh, vec, dof, sum)

type(hecmwST_local_mesh) :: mesh
integer(kind=kint),pointer :: vec(:)
integer(kind=kint) :: dof, sum
```

引数

mesh
: メッシュデータの格納先

vec
: ベクトルデータの格納先

dof
: ベクトルの自由度

sum
: 内積の値

説明

- ベクトルの長さは mesh に格納された情報と自由度から自動的に計算されます。
- 内部から「hecmw_allreduce_I1」を呼び出しています。

hecmw_innerProduct_R

実数ベクトルの内積を求めます。

```
subroutine hecmw_innerProduct_R (mesh, vec, dof, sum)

type(hecmwST_local_mesh) :: mesh
real(kind=kint),pointer :: vec(:)
integer(kind=kint) :: dof
real(kind=kint) :: sum
```

引数

mesh
: メッシュデータの格納先

vec
: ベクトルデータの格納先

dof
: ベクトルの自由度

sum
: 内積の値

説明

- ベクトルの長さは mesh に格納された情報と自由度から自動的に計算されます。
- 内部から「hecmw_allreduce_R1」を呼び出しています。

3. 有限要素演算機能

- 「有限要素演算機能」で使用する，サブルーチンおよび API について説明します。

hecmw_mat_con

係数行列の CRS テーブルを作ります。

```
subroutine hecmw_mat_con (mesh, matrix)

type(hecmwST_local_mesh) :: mesh
type(hecmwST_matrix)     :: matrix
```

引数

mesh

メッシュデータの格納先

matrix

マトリクスデータの格納先

説明

- 全体化された係数行列の圧縮格納用データテーブルを作ります。
- 生成されたテーブルデータは、matrix 構造体に格納されます。
- CRS フォーマットに関しては HEC ミドルウェアの線形ソルバを参照してください。
- 内部から「hecmw_mat_con0」および「hecmw_mat_con1」を呼び出しています。

hecmw_mat_ass_elem

要素行列を全体行列に足しこみます。

```
subroutine hecmw_mat_ass_elem (hecMAT, nn, nodLOCAL, matrix)

type(hecmwST_matrix)     :: hecMAT
integer(kind=kint) :: nn
integer(kind=kint) :: nodLOCAL(:)
real(kind=kreal) :: matrix(:,:)
```

引数

hecMAT

マトリクスデータ（CRS テーブル）の格納先

nn

要素の節点数

nodLOCAL

要素を構成する節点

matrix

要素行列

説明

- 要素行列を全体行列に足しこみます。
- 要素行列は、一辺が節点数×節点自由度数の正方行列（2 次元配列）として与えます。

hecmw_mat_ass_equation

全体行列に多点拘束の条件式をくみこみます。

```
subroutine hecmw_mat_ass_equation (hecMESH, hecMAT)

type(hecmwST_mesh)     :: hecMESH
type(hecmwST_matrix)     :: hecMAT
```

引数

hecMESH

メッシュデータの格納先

hecMAT

マトリクスデータ（CRS テーブル）の格納先

説明

- メッシュデータ内に定義されている多点拘束の条件式をくみこみます。
- 用いるペナルティ値は hecMAT%rarray(11) に予めセットしておきます。

hecmw_mat_ass_bc

全体行列に拘束条件をくみこみます。

```
subroutine hecmw_mat_ass_bc (hecMAT, inode, idof, RHS)

type(hecmwST_matrix)     :: hecMAT
integer(kind=kint) :: inode
integer(kind=kint) :: idof
real(kind=kreal) :: RHS
```

引数

hecMAT
: マトリクスデータ（CRS テーブル）の格納先

inode
: 拘束される節点

idof
: 拘束される自由度

RHS
: 拘束する値

説明

- 指定した節点の指定した自由度を指定した値に拘束します。

hecmw_Jacob_231

要素タイプ 231 に関するガウス積分の重み、形状関数、形状関数の微係数の演算。

```
subroutine hecmw_Jacob_231 (mesh, iElem, DET, w, N, NX, NY)

type(hecmwST_local_mesh) :: mesh
integer(kind=kint) :: iElem
real(kind=kreal) :: DET
real(kind=kreal) :: w(3), N(3,3), NX(3,3), NY(3,3)
```

引数

mesh
: メッシュデータの格納先

iElem
: 要素番号

DET
: ヤコビアンの値

w
: 各積分点における重みの値

N
: 各積分点における形状関数の値

NX
: 各積分点における形状関数の X 方向微係数の値

NY
: 各積分点における形状関数の Y 方向微係数の値

説明

- ガウス積分の次数は 2、積分点数は 3 です。
- w および N、NX、NY の第 1 引数が積分点番号、第 2 引数が要素内節点番号です。

hecmw_Jacob_241

要素タイプ 241 に関するガウス積分の重み、形状関数、形状関数の微係数の演算。

```
subroutine hecmw_Jacob_241 (mesh, iElem, DET, w, N, NX, NY)

type(hecmwST_local_mesh) :: mesh
integer(kind=kint) :: iElem
real(kind=kreal) :: DET
real(kind=kreal) :: w(4), N(4,4), NX(4,4), NY(4,4)
```

引数

mesh
: メッシュデータの格納先

iElem
: 要素番号

DET
: ヤコビアンの値

w
: 各積分点における重みの値

N
: 各積分点における形状関数の値

NX
: 各積分点における形状関数の X 方向微係数の値

NY
: 各積分点における形状関数の Y 方向微係数の値

説明

- ガウス積分の次数は 2、積分点数は 4 です。
- w および N、NX、NY の第 1 引数が積分点番号、第 2 引数が要素内節点番号です。

hecmw_Jacob_341

要素タイプ 341 に関するガウス積分の重み、形状関数、形状関数の微係数の演算。

```
subroutine hecmw_Jacob_341 (mesh, iElem, DET, w, N, NX, NY, NZ)

type(hecmwST_local_mesh) :: mesh
integer(kind=kint) :: iElem
real(kind=kreal) :: DET
real(kind=kreal) :: w(4), N(4,4), NX(4,4), NY(4,4), NZ(4,4)
```

引数

mesh

メッシュデータの格納先

iElem

要素番号

DET

ヤコビアンの値

w

各積分点における重みの値

N

各積分点における形状関数の値

NX

各積分点における形状関数の X 方向微係数の値

NY

各積分点における形状関数の Y 方向微係数の値

NZ

各積分点における形状関数の Z 方向微係数の値

説明

- ガウス積分の次数は 2、積分点数は 4 です。
- w および N、NX、NY の第 1 引数が積分点番号、第 2 引数が要素内節点番号です。

hecmw_Jacob_361

要素タイプ 361 に関するガウス積分の重み、形状関数、形状関数の微係数の演算。

```
subroutine hecmw_Jacob_361 (mesh, iElem, DET, w, N, NX, NY, NZ)

type(hecmwST_local_mesh) :: mesh
integer(kind=kint) :: iElem
real(kind=kreal) :: DET
real(kind=kreal) :: w(8), N(8,8), NX(8,8), NY(8,8), NZ(8,8)
```

引数

mesh

メッシュデータの格納先

iElem

要素番号

DET

ヤコビアンの値

w

各積分点における重みの値

N

各積分点における形状関数の値

NX

各積分点における形状関数の X 方向微係数の値

NY

各積分点における形状関数の Y 方向微係数の値

NZ

各積分点における形状関数の Z 方向微係数の値

説明

- ガウス積分の次数は 2、積分点数は 8 です。

w および N、NX、NY の第 1 引数が積分点番号、第 2 引数が要素内節点番号です。

4. 適応格子機能

「適応格子機能」で使用する，サブルーチンおよび API について説明します。「HEC-MW」でサポートしている適応格子機能を利用する場合には，以下のサブルーチンをこの順番に呼び出す必要があります。

- hecmw_adapt_init　　「適応格子機能」初期化
- hecmw_adapt_proc　　「適応格子機能」実施
- hecmw_adapt_new_mesh　　新規メッシュ構造体格納

また，辺情報を取得するために以下のサブルーチンを使用します。

- hecmw_adapt_edge_info　　辺情報取得

hecmw_adapt_init

「適応格子機能」を初期化します

```
subroutine hecmw_adapt_init(hecMESH)

type(hecmwST_local_mesh) :: hecMESH
```

引 数

hecMESH　　メッシュデータの格納先（構造体）

説 明

- 辺情報の生成，通信テーブル（辺，要素）を実施します。
- このサブルーチンを呼ぶ前に，「hecmw_get_mesh」等により，メッシュデータ構造体に分散メッシュ情報を格納しておく必要があります。

hecmw_adapt_proc

「適応格子機能」を実施します

```
subroutine hecmw_adapt_proc(hecMESH)

type(hecmwST_local_mesh) :: hecMESH
```

引 数

hecMESH　　メッシュデータの格納先（構造体）

説 明

- 分割辺設定，節点・要素・グループ情報の更新を実施します。
- 通信テーブルの再構成を実施します。
- このサブルーチンを呼び出す前に，hecMESH%adapt_iemb（辺分割の情報，分割する辺については hecMESH%adapt_iemb(edge)=1，分割しない場合には hecMESH%adapt_iemb(edge)=0）の値を入れておく必要があります。

hecmw_adapt_new_mesh

新規メッシュデータ構造体へのメッシュ情報の格納を行います

```
subroutine hecmw_adapt_new_mesh(hecMESH, hecMESHnew)

type(hecmwST_local_mesh) :: hecMESH
type(hecmwST_local_mesh) :: hecMESHnew
```

引 数

hecMESH	メッシュデータの格納先（構造体）
hecMESHnew	新規メッシュデータの格納先（構造体）

説 明

hecmw_adapt_edge_info

辺情報を取得します。

```
subroutine hecmw_adapt_edge_info (hecMESH, nod1, nod2, edge, FLAG)

type(hecmwST_local_mesh) :: hecMESH
integer :: nod1, nod2, edge, FLAG
```

引 数

hecMESH	メッシュデータの格納先（構造体）
nod1	第1点の節点番号
nod2	第2点の節点番号
edge	辺番号
FLAG	フラグ（通常は1とする）

説 明

- FLAG=0 の場合は新規辺生成，FLAG=1 の場合は既存辺番号取得，FLAG=2 の場合は辺情報クリアですが，ユーザーが利用する場合は「FLAG=1」となります。

5. 動的負荷分散機能

「動的負荷分散機能」で使用する，サブルーチンおよび API について説明します。「HEC-MW」でサポートしている動的負荷分散機能を利用する場合には，以下のサブルーチンをこの順番に呼び出す必要があります。

- hecmw_transfer_data_f2c
  負荷バランスのとれていないメッシュデータとその結果データを Fortran 領域から C 領域へ転送します。

- hecmw_dynamic_load_balancing
  メッシュデータを際領域分割することで、負荷バランスのとれたメッシュデータとその結果データを作成します。

- hecmw_transfer_data_c2f
  負荷バランスのとれたメッシュデータとその結果データを C 領域から Fortran 領域へ転送します。

hecmw_transfer_data_f2c

負荷バランスのとれていないメッシュデータとその結果データを Fortran 領域から C 領域へ転送します。

```
subroutine hecmw_transfer_data_f2c(hecMESH, adapRES)

type(hecmwST_local_mesh) :: hecMESH
type(hecmwST_result_data):: adapRES
```

引 数

hecMESH: 負荷バランスのとれていないメッシュデータ
adapRES: 負荷バランスのとれていない結果データ

hecmw_adapt_dynamic_load_balancing

メッシュデータを際領域分割することで、負荷バランスのとれたメッシュデータとその結果データを作成します。

```
subroutine hecmw_adapt_dynamic_load_balancing()
```

引 数

hecmw_transfer_data_c2f

負荷バランスのとれたメッシュデータとその結果データを C 領域から Fortran 領域へ転送します。

```
subroutine hecmw_transfer_data_c2f(hecMESH, adapRES)

type(hecmwST_local_mesh) :: hecMESH
type(hecmwST_result_data):: adapRES
```

引 数

hecMESH:　　負荷バランスのとれたメッシュデータ
adapRES:　　負荷バランスのとれた結果データ

6. 連成カップリング機能

本節では、「連成カップリング機能」に関する API について説明します。

hecmw_couple_get_mesh

メッシュデータをファイルから読み込みます。

```
subroutine hecmw_couple(name_ID, unit_ID)

character(len=HECMW_NAME_LEN) :: name_ID
character(len=HECMW_NAME_LEN) :: unit_ID
```

引数

name_ID

!MESH または!MESH GROUP を特定する識別 ID

unit_ID

!COUPLE UNIT を特定する識別 ID

説明

ファイルからメッシュデータを読み込みます。

メッシュデータの入力には，通常 hecmw_get_mesh()というサブルーチンを使用しますが，連成カップリング機能を利用する際には，本サブルーチンによってメッシュデータを入力しなくてはなりません。ここで読み込まれるメッシュファイルは，全体制御ファイルに記述された!MESH もしくは!MESH GROUP セクションのうち，NAME パラメータが name_ID であるものとなります。読み込み可能なメッシュファイルの種類などは，サブルーチン hecmw_get_mesh に準じますので，詳細は「I/O，全体制御 プログラム使用説明書」を参照してください。

取得したメッシュデータは，全体制御ファイルに記述された!COUPLE UNIT セクションのうち，NAME パラメータが unit_ID の連成ユニットに割り当てられます。

なお，name_ID を NAME パラメータにもつ!MESH もしくは!MESH GROUP セクションが全体制御ファイルで定義されていない場合，および，unit_ID を NAME パラメータにもつ!COUPLE UNIT セクションが全体制御ファイルで定義されていない場合には，エラーとなります。

hecmw_couple_init

連成情報を作成します。

```
subroutine hecmw_couple_init(boundary_id, mesh_unit1, mesh_unit2)

character(len=HECMW_NAME_LEN) :: boundary_id
type(hecmwST_local_mesh) :: mesh_unit1, mesh_unit2
```

引数

boundary_id

!COUPLE BOUNDARY を特定する識別 ID

mesh_unit1

連成ユニット 1 のメッシュデータ

mesh_unit2

連成ユニット 2 のメッシュデータ

説明

連成する 2 つの連成ユニットの境界部でのマッピング処理などを行い，連成情報を作成します。

ここでは，全体制御ファイルに記述された!COUPLE BOUNDARY セクションのうち，NAME パラメータが boundary_id であるセクションの連成情報が作成されます。連成情報の制御データとしては，該当する!COUPLE BOUNDARY セクションの COUPLE パラメータに与えられた連成情報識別 ID を NAME パラメータにもつ!COUPLE セクションのものが使用されます。連成される 2 つの連成ユニットは，この!COUPLE セクションの UNIT1 および UNIT2 パラメータに指定された連成ユニット情報識別 ID を NAME パラメータにもつ!COUPLE UNIT セクションのものとなり，UNIT1 パラメータに指定された連成ユニットが連成ユニット 1 に，UNIT2 パラメータに指定された連成ユニットが連成ユニット 2 になります。

第 2 引数と第 3 引数には，連成させる 2 つの連成ユニットに割り当てられたメッシュデータを渡しますが，第 2 引数には連成ユニット 1 のものを，第 3 引数には連成ユニット 2 のものを渡してください。なお，このサブルーチンを呼び出す MPI プロセスが連成ユニッ

ト 1 もしくは 2 のどちらか，またはどちらにも割り当てられていない場合にも，空の hecmwST_local_mesh 型の変数を引数で渡す必要があります。

boundary_ID に該当する!COUPLE BOUNDARY セクションが全体制御ファイルで定義されていない場合には、エラーとなります。

hecmw_couple_finalize

連成情報を破棄します。

```
subroutine hecmw_finalize(boundary_id)

character(len=HECMW_NAME_LEN) :: boundary_id
```

引数

boundary_id

!COUPLE BOUNDARY を特定する識別 ID

説明

サブルーチン hecmw_couple_init で作成した連成情報を破棄します。

ここ破棄される連成情報は，hecmw_couple_init の呼び出しによって作成された情報のうち，その第 1 引数に boundary_id が与えられて作成された情報になります。

boundary_id に対する連成情報が hecmw_couple_init によって作成されていない場合には，エラーとなります。

hecmw_couple_startup

連成させる値についての情報を提供します。

```
subroutine hecmw_dist_free(boundary_id, couple_value)

character(len=HECMW_NAME_LEN) :: boundary_id
type(hecmw_couple_value) :: couple_value
```

引数

boundary_id

!COUPLE BOUNDARY を特定する識別 ID

couple_value

連成値情報

説明

連成値情報の一部を提供します。

本 API は構造体 hecmw_couple_value 型の変数を作成し，boundary_id によって特定される連成情報に従って，以下の 3 つのメンバに値を代入します。

couple_value%n

couple_value%item_type

couple_value%item

連成を実行する際には，この情報を元に

couple_value%n_dof

couple_value%value

に値を代入し，hecmw_couple の引数へと渡すことになります。

本 API を呼び出す前に boundary_id を第 1 引数に与えて hecmw_couple_init を呼び出し，連成情報を作成しておく必要があります。

hecmw_couple_cleanup

連成値情報を破棄します。

```
subroutine hecmw_couple_cleanup(couple_value)

type(hecmw_couple_value) :: couple_value
```

引数

couple_value

破棄する連成値情報

説明

連成値情報の構造体のメンバに割り当てられたメモリ領域を開放し，連成値情報を破棄します。

hecmw_couple

連成を実行します。

```
subroutine  hecmw_couple(boundary_id, couple_value)

character(len=HECMW_NAME_LEN) :: boundary_id
type(hecmw_couple_value) :: couple_value
```

引数

boundary_id

!COUPLE BOUNDARY を特定する識別 ID

couple_value

連成境界値

説明

連成を実行し，連成ユニット間でのデータの補間処理を行ないます。

連成の実行に際しては，第 1 引数に boundary_id を与えられて呼び出された hecmw_couple_init が作成した連成情報が用いられます。

第 2 引数の couple_value は送受信兼用になっています。送信側の連成ユニットのメンバプロセスは couple_value の構造体のメンバ全てに値を代入した上で，本サブルーチンの引数に渡さなければなりません。一方，受信側の連成ユニットのメンバプロセスは，連成の実行によって得られたデータを couple_value に受けることになります。そのため，連成タイプが Max(M,N)タイプの場合など，一つの MPI プロセスが送信側，受信側の両方の連成ユニットのメンバプロセスとなる場合には，本サブルーチンを呼び出す際に couple_value に代入された値は全て破棄され，受信用に使用されることに注意してください。

hecmw_couple_is_member

連成のメンバプロセスであるか否かのフラグを提供します。

```
function hecmw_couple_is_member(boundary_id), result(is_member)

character(len=HECMW_NAME_LEN) :: boundary_id
integer(kind=kint) :: is_member
```

引数

boundary_id

!COUPLE BOUNDARY を特定する識別 ID

説明

連成プロセスのメンバプロセスであるか否かのフラグを提供する関数です。

本関数を呼び出した MPI プロセスが，全体制御ファイルの!COUPLE BOUNDARY セクションのうち，NAME オプションが boundary_id であるセクションのメンバプロセスである場合には「1」が、メンバプロセスでない場合には「0」が返されます。ここで，メンバプロセスとなるのは，該当する!COUPLE BOUNDARY セクションの COUPLE パラメータで特定される!COUPLE セクションの，UNIT1 および UNIT2 パラメータで指定された連成ユニットに割り当てられたプロセスとなります。

全体制御ファイルに boundary_id で特定される!COUPLE BOUNDARY セクションが無い場合には、エラーとなります。

hecmw_couple_is_unit_member

連成ユニットのメンバプロセスであるか否かのフラグを提供します。

```
function hecmw_couple_is_unit_member(boundary_id, unit_specifier),
         result(is_member)

character(len=HECMW_NAME_LEN) :: boundary_id
integer(kind=kint) :: unit_specifier
integer(kind=kint) :: is_member
```

引数

boundary_id

!COUPLE BOUNDARY を特定する識別 ID

unit_specifier

連成ユニット指定子

説明

連成ユニットのメンバプロセスであるか否かのフラグを提供する関数です。

本関数を呼び出した MPI プロセスが，全体制御ファイルの!COUPLE BOUNDARY セクションのうち，NAME オプションが boundary_id であるセクションの，unit_specifier で指定される連成ユニットのメンバプロセスである場合には「1」が、メンバプロセスでない場合には「0」が返されます。該当する!COUPLE BOUNDARY セクションの，連成ユニット 1 側のメンバプロセスを対象とする場合には unit_specifier に HECMW_COUPLE_UNIT1 を，連成ユニット 2 側のメンバプロセスを対象とする場合には unit_specifier に HECMW_COUPLE_UNIT2 を指定します。

全体制御ファイルに boundary_id で特定される!COUPLE BOUNDARY セクションが無い場合, unit_specifier に HECMW_COUPLE_UNIT1 もしくは HECMW_COUPLE_UNIT2 以外が指定された場合には、エラーとなります。

hecmw_couple_is_unit_member_u

連成ユニットのメンバプロセスであるか否かのフラグを提供します。

```
function hecmw_couple_is_unit_member_u(unit_id), result(is_member)

character(len=HECMW_NAME_LEN) :: unit_id
integer(kind=kint) :: is_member
```

引数

unit_id

!COUPLE UNIT を特定する識別 ID

説明

連成ユニットのメンバプロセスであるか否かのフラグを提供する関数です。

本関数を呼び出した MPI プロセスが，全体制御ファイルの!COUPLE UNIT セクションのうち，NAME オプションが unit_id であるセクションのメンバプロセスである場合には「1」が、メンバプロセスでない場合には「0」が返されます。

全体制御ファイルに unit_id で特定される!COUPLE UNIT セクションが無い場合には、エラーとなります。

hecmw_couple_is_root

連成のルートプロセスであるか否かのフラグを提供します。

```
function hecmw_couple_is_root(boundary_id), result(is_root)

character(len=HECMW_NAME_LEN) :: boundary_id
integer(kind=kint) :: is_root
```

引数

boundary_id

!COUPLE BOUNDARY を特定する識別 ID

説明

連成プロセスのルートプロセスであるか否かのフラグを提供する関数です。

本関数を呼び出した MPI プロセスが，全体制御ファイルの!COUPLE BOUNDARY セクションのうち，NAME オプションが boundary_id であるセクションのルートプロセスである場合には「1」が、ルートプロセスでない場合には「0」が返されます。

全体制御ファイルに boundary_id で特定される!COUPLE BOUNDARY セクションが無い場合には、エラーとなります。

hecmw_couple_is_unit_root

連成ユニットのルートプロセスであるか否かのフラグを提供します。

```
function hecmw_couple_is_unit_root(boundary_id, unit_specifier),
         result(is_root)

character(len=HECMW_NAME_LEN) :: boundary_id
integer(kind=kint) :: unit_specifier
integer(kind=kint) :: is_root
```

引数

boundary_id

!COUPLE BOUNDARY を特定する識別 ID

unit_specifier

連成ユニット指定子

説明

連成ユニットのルートプロセスであるか否かのフラグを提供する関数です。

本関数を呼び出した MPI プロセスが，全体制御ファイルの!COUPLE BOUNDARY セクションのうち，NAME オプションが boundary_id であるセクションの，unit_specifier で指定される連成ユニットのルートプロセスである場合には「1」が、ルートプロセスでない場合には「0」が返されます。該当する!COUPLE BOUNDARY セクションの連成ユニット 1 側のルートプロセスを対象とする場合には unit_specifier に HECMW_COUPLE_UNIT 1 を，連成ユニット 2 側のルートプロセスを対象とする場合には unit_specifier に HECMW_COUPLE_UNIT2 を指定します。

全体制御ファイルに boundary_id で特定される!COUPLE BOUNDARY セクションが無い場合, unit_specifier に HECMW_COUPLE_UNIT1 もしくは HECMW_COUPLE_UNIT2 以外が指定された場合には、エラーとなります。

hecmw_couple_is_unit_root_u

連成ユニットのルートプロセスであるか否かのフラグを提供します。

```
function hecmw_couple_is_unit_root_u(unit_id), result(is_root)

character(len=HECMW_NAME_LEN) :: unit_id
integer(kind=kint) :: is_root
```

引数

unit_id

!COUPLE UNIT を特定する識別 ID

説明

連成ユニットのルートプロセスであるか否かのフラグを提供する関数です。

本関数を呼び出した MPI プロセスが，全体制御ファイルの!COUPLE UNIT セクションのうち，NAME オプションが unit_id であるセクションのルートプロセスである場合には「1」が、ルートプロセスでない場合には「0」が返されます。

全体制御ファイルに unit_id で特定される!COUPLE UNIT セクションが無い場合には、エラーとなります。

hecmw_intercomm_get_size

連成ユニット間通信における MPI プロセス数を提供します。

```
function hecmw_intercomm_get_size(boundary_id), result(psize)

character(len=HECMW_NAME_LEN) :: boundary_id
integer(kind=kint) :: psize
```

引数

boundary_id

!COUPLE BOUNDARY を特定する識別 ID

説明

連成ユニット間通信における MPI プロセス数を提供する関数です。

全体制御ファイルの!COUPLE BOUNDARY セクションのうち，NAME オプションが boundary_id であるセクションに割り当てられた MPI プロセス数が返されます。

全体制御ファイルに boundary_id で特定される!COUPLE BOUNDARY セクションが無い場合には、エラーとなります。

hecmw_intracomm_get_size

連成ユニット内通信における MPI プロセス数を提供します。

```
function hecmw_intracomm_get_size(boundary_id, unit_specifier),
         result(psize)

character(len=HECMW_NAME_LEN) :: boundary_id
integer(kind=kint) :: unit_specifier
integer(kind=kint) :: psize
```

引数

boundary_id

!COUPLE BOUNDARY を特定する識別 ID

unit_specifier

連成ユニット指定子

説明

連成ユニット内通信における MPI プロセス数を提供する関数です。

全体制御ファイルの!COUPLE BOUNDARY セクションのうち，NAME オプションが boundary_id であるセクションの，unit_specifier で指定される連成ユニットに割り当てられた MPI プロセス数が返されます。該当する!COUPLE BOUNDARY セクションの連成ユニット 1 側の MPI プロセス数を取得する場合には unit_specifier に HECMW_COUPLE_UNIT1 を，連成ユニット 2 側の MPI プロセス数を取得する場合には unit_specifier に HECMW_COUPLE_UNIT2 を指定します。

全体制御ファイルに boundary_id で特定される!COUPLE BOUNDARY セクションが無い場合, unit_specifier に HECMW_COUPLE_UNIT1 もしくは HECMW_COUPLE_UNIT2 以外が指定された場合には、エラーとなります。

hecmw_intracomm_get_size_u

連成ユニット内通信における MPI プロセス数を提供します。

```
function hecmw_intracomm_get_size_u(unit_id), result(psize)

character(len=HECMW_NAME_LEN) :: unit_id
integer(kind=kint) :: psize
```

引数

unit_id

!COUPLE UNIT を特定する識別 ID

説明

連成ユニット内通信における MPI プロセス数を提供する関数です。

全体制御ファイルの!COUPLE UNIT セクションのうち，NAME オプションが unit_id であるセクションに割り当てられた MPI プロセス数が返されます。

全体制御ファイルに unit_id で特定される!COUPLE UNIT セクションが無い場合には、エラーとなります。

hecmw_intercomm_get_rank

連成ユニット間通信における MPI プロセスランク番号を提供します。

```
function hecmw_intercomm_get_rank(boundary_id), result(rank)

character(len=HECMW_NAME_LEN) :: boundary_id
integer(kind=kint) :: rank
```

引数

boundary_id

!COUPLE BOUNDARY を特定する識別 ID

説明

連成ユニット間通信における MPI プロセスランク番号を提供する関数です。

全体制御ファイルの!COUPLE BOUNDARY セクションのうち，NAME オプションが boundary_id であるセクションに割り当てられた MPI プロセスのプロセスランク番号が返されます。

全体制御ファイルに boundary_id で特定される!COUPLE BOUNDARY セクションが無い場合には、エラーとなります。

hecmw_intracomm_get_rank

連成ユニット内通信における MPI プロセスランク番号を提供します。

```
function hecmw_intracomm_get_rank(boundary_id, unit_specifier),
         result(rank)

character(len=HECMW_NAME_LEN) :: boundary_id
integer(kind=kint) :: unit_specifier
integer(kind=kint) :: rank
```

引数

boundary_id

!COUPLE BOUNDARY を特定する識別 ID

unit_specifier

連成ユニット指定子

説明

連成ユニット内通信における MPI プロセスランク番号を提供する関数です。

全体制御ファイルの!COUPLE BOUNDARY セクションのうち，NAME オプションが boundary_id であるセクションの，unit_specifier で指定される連成ユニットに割り当てられた MPI プロセスのプロセスランク番号が返されます。該当する!COUPLE BOUNDARY セクションの連成ユニット 1 側の MPI プロセスを対象とする場合には unit_specifier に HECMW_COUPLE_UNIT1 を，連成ユニット 2 側の MPI プロセスを対象とする場合には unit_specifier に HECMW_COUPLE_UNIT2 を指定します。

全体制御ファイルに boundary_id で特定される!COUPLE BOUNDARY セクションが無い場合, unit_specifier に HECMW_COUPLE_UNIT1 もしくは HECMW_COUPLE_UNIT2 以外が指定された場合には、エラーとなります。

hecmw_intracomm_get_rank_u

連成ユニット内通信における MPI プロセスランク番号を提供します。

```
function hecmw_intracomm_get_rank_u(unit_id), result(rank)

character(len=HECMW_NAME_LEN) :: unit_id
integer(kind=kint) :: rank
```

引数

unit_id

!COUPLE UNIT を特定する識別 ID

説明

連成ユニット内通信における MPI プロセスランク番号を提供する関数です。

全体制御ファイルの!COUPLE UNIT セクションのうち，NAME オプションが unit_id であるセクションに割り当てられた MPI プロセスのプロセスランク番号が返されます。

全体制御ファイルに unit_id で特定される!COUPLE UNIT セクションが無い場合には、エラーとなります。

hecmw_intercomm_get_comm

連成ユニット間通信における MPI コミュニケータを提供します。

```
function hecmw_intercomm_get_comm(boundary_id), result(comm)

character(len=HECMW_NAME_LEN) :: boundary_id
integer(kind=kint) :: comm
```

引数

boundary_id

!COUPLE BOUNDARY を特定する識別 ID

説明

連成ユニット間通信における MPI コミュニケータを提供する関数です。

全体制御ファイルの!COUPLE BOUNDARY セクションのうち，NAME オプションが boundary_id であるセクションにおける MPI コミュニケータが返されます。

全体制御ファイルに boundary_id で特定される!COUPLE BOUNDARY セクションが無い場合には、エラーとなります。

hecmw_intracomm_get_comm

連成ユニット内通信における MPI コミュニケータを提供します。

```
function hecmw_intracomm_get_comm(boundary_id, unit_specifier),
         result(comm)

character(len=HECMW_NAME_LEN) :: boundary_id
integer(kind=kint) :: unit_specifier
integer(kind=kint) :: comm
```

引数

boundary_id

!COUPLE BOUNDARY を特定する識別 ID

unit_specifier

連成ユニット指定子

説明

連成ユニット内通信における MPI コミュニケータを提供する関数です。

全体制御ファイルの!COUPLE BOUNDARY セクションのうち，NAME オプションが boundary_id であるセクションの，unit_specifier で指定される連成ユニットにおける MPI コミュニケータが返されます。該当する!COUPLE BOUNDARY セクションの連成ユニット 1 側を対象とする場合には unit_specifier に HECMW_COUPLE_UNIT1 を，連成ユニット 2 側を対象とする場合には unit_specifier に HECMW_COUPLE_UNIT2 を指定します。

全体制御ファイルに boundary_id で特定される!COUPLE BOUNDARY セクションが無い場合, unit_specifier に HECMW_COUPLE_UNIT1 もしくは HECMW_COUPLE_UNIT2 以外が指定された場合には、エラーとなります。

hecmw_intracomm_get_comm_u

連成ユニット内通信における MPI コミュニケータを提供します。

```
function hecmw_intracomm_get_comm_u(unit_id), result(comm)

character(len=HECMW_NAME_LEN) :: unit_id
integer(kind=kint) :: comm
```

引数

unit_id

!COUPLE UNIT を特定する識別 ID

説明

連成ユニット内通信における MPI コミュニケータを提供する関数です。

全体制御ファイルの!COUPLE UNIT セクションのうち，NAME オプションが unit_id であるセクションにおける MPI コミュニケータが返されます。

全体制御ファイルに unit_id で特定される!COUPLE UNIT セクションが無い場合には、エラーとなります。

hecmw_intercomm_get_group

連成ユニット間通信における MPI グループを提供します。

```
function hecmw_intercomm_get_rank(boundary_id), result(group)

character(len=HECMW_NAME_LEN) :: boundary_id
integer(kind=kint) :: group
```

引数

boundary_id

!COUPLE BOUNDARY を特定する識別 ID

説明

連成ユニット間通信における MPI グループを提供する関数です。

全体制御ファイルの!COUPLE BOUNDARY セクションのうち，NAME オプションが boundary_id であるセクションにおける MPI グループが返されます。

全体制御ファイルに boundary_id で特定される!COUPLE BOUNDARY セクションが無い場合には、エラーとなります。

hecmw_intracomm_get_group

連成ユニット内通信における MPI グループを提供します。

```
function hecmw_intracomm_get_group(boundary_id, unit_specifier),
         result(group)

character(len=HECMW_NAME_LEN) :: boundary_id
integer(kind=kint) :: unit_specifier
integer(kind=kint) :: group
```

引数

boundary_id

!COUPLE BOUNDARY を特定する識別 ID

unit_specifier

連成ユニット指定子

説明

連成ユニット内通信における MPI グループを提供する関数です。

全体制御ファイルの!COUPLE BOUNDARY セクションのうち，NAME オプションが boundary_id であるセクションの，unit_specifier で指定される連成ユニットにおける MPI グループが返されます。該当する!COUPLE BOUNDARY セクションの連成ユニット 1 側を対象とする場合には unit_specifier に HECMW_COUPLE_UNIT1 を，連成ユニット 2 側を対象とする場合には unit_specifier に HECMW_COUPLE_UNIT2 を指定します。

全体制御ファイルに boundary_id で特定される!COUPLE BOUNDARY セクションが無い場合, unit_specifier に HECMW_COUPLE_UNIT1 もしくは HECMW_COUPLE_UNIT2 以外が指定された場合には、エラーとなります。

hecmw_intracomm_get_group_u

連成ユニット内通信における MPI グループを提供します。

```
function hecmw_intracomm_get_group_u(unit_id), result(group)

character(len=HECMW_NAME_LEN) :: unit_id
integer(kind=kint) :: group
```

引数

unit_id

!COUPLE UNIT を特定する識別 ID

説明

連成ユニット内通信における MPI グループを提供する関数です。

全体制御ファイルの!COUPLE UNIT セクションのうち，NAME オプションが unit_id であるセクションにおける MPI グループが返されます。

全体制御ファイルに unit_id で特定される!COUPLE UNIT セクションが無い場合には、エラーとなります。