

Nov 21, 00 16:12

DRI\_LIS\_09292000.txt

Page 1/25

\$Id: DRI\_LIS.txt,v 1.7 2000/11/21 05:02:53 kcain Exp \$

This is a language-independent specification of a meta-API for data reorganization. This version of the specification is based on the consensus of the Data Reorganization Forum following its September 2000 meeting.

DATE OF THIS DRAFT: 09/29/2000

There are 4 sections to this document:

- 1) Administrative notes
- 2) Change summary between this version and the prior version
- 3) Current version of "critical path" interfaces in the API
- 4) Current version of other interfaces in the API

----- SECTION 1: Administrative Notes -----

These notes are mostly re-iterated from previous versions of this draft.

NOTE #1: A change has been made in the organization of this file so that the functions that are in the "critical path" to creating and initiating a data reorganization function are presented first. Other functions such as object query functions that are not in this "critical path" are presented later.

As of this date (09/29/2000), this draft contains only the critical path calls (i.e., Section 4 is empty). Since there have been many changes to these functions, we should agree on their specification before handling the low-level ("power user") and non-critical functions.

NOTE #2: A new feature is being added for the committee members - a list of changes that have been introduced in this newer version of the API and the corresponding reasons. The goal is to prevent revisiting issues that have been resolved in past working meetings.

----- SECTION 2: Change Summary -----

\*\*\*\*\* Changes from July 2000 draft to September 2000 (POST SEPTEMBER MEETING) \*\*\*\*\*  
draft:

0. Reorganization of API and associated terminology:

CORE  
Standalone / pure  
Standalone / middleware adapter

CORE Data Reorg interfaces must appear in all implementations. They are inherently part of Data Reorg and are not likely to appear in other existing standard APIs.

List of CORE objects/interfaces:

For non-CORE interfaces, there is an implicit overlap with existing middleware services. Implementers can fully implement these services

according to the Data Reorg "standalone" specification (akin to re-inventing the wheel), or they can make reasonable choices about how to leverage the existing services in so-called "middleware adapters". In this respect, the Data Reorg specification can be thought of as "META" (i.e., not precisely specified, because it could be instantiated in many ways corresponding to the many middlewares that might "co-layer" with Data Reorg).

The Data Reorg committee will define, along with the first CORE and pure standalone interfaces, an MPI middleware adapter.

List of Standalone objects/interfaces:

1. Removed dataspec parameter from DRI\_Global\_Data\_create

We are deferring data type specification until Channel creation time.

Having a data type input parameter to DRI\_Global\_Data\_create would confuse its distinction as a Data Reorg "CORE" interface. Data types can take many forms, since they are implemented in many other middlewares (e.g., MPI, MPI/RT, VSIPL, ...).

2. Added "name" parameter to DRI\_Global\_Data\_create

This string parameter was added as a way to connect off-line configuration file specifications to run-time calls to the Data Reorg library. Some architectures use a configuration file approach to specifying communication resources that will be used at run-time.

Another benefit of adding this parameter is to facilitate any future debugging or profiling features either included in DRI itself or provided by a third party as a useful complementary capability.

3. Changed DRI\_Distribution\_calc\_local\_size to  
DRI\_Distribution\_get\_local\_count

In addition to the name change, the return value has changed from number of bytes to number of data elements.

The change from returning bytes to returning elements is necessary because the data type of the data being partitioned has not yet been bound in any prior calls (see related change to DRI\_Global\_Data\_create).

4. In DRI\_Distribution\_create documentation, clarified a special case that corresponds to "broadcasting" data.

The case is when the DRI\_Partition object refers to a "whole" partitioning of a data dimension (i.e., indivisible), but the process group logical topology (group\_dims parameter) specifies more than 1 process. Here, such distributions will effectively replicate the data across all associated processes.

5. In DRI\_Bufferset\_system\_create, renamed dist parameter to disth to be consistent with other documentation in this document.

6. DRI\_Distribution\_create - wholesale modifications!!

Nov 21, 00 16:12

DRI\_LIS\_09292000.txt

Page 3/25

- 6a) Changed how layouts input array works. There are 3 cases supported:
- layouts is NULL (no layout specified at all - default operation request)
  - layouts is a mixture of user-instantiated DRI\_Layout and pre-defined DRI\_LAYOUT\_NULL objects
  - layouts consists entirely of user-instantiated DRI\_Layout objects
- See documentation for DRI\_Distribution\_create for details
- 6b) Removed group parameter, in favor of user specifying group\_size and myrank
- This was motivated by the need to remove so-called "META" arguments out of "CORE" Data Reorg functions like this
7. Moved DRI\_Dataspec input parameter into DRI\_Channel\_create\_send and DRI\_Channel\_create\_recv functions (from DRI\_Global\_Data\_create). See reasoning above in item 1
8. Changed DRI\_Bufferset\_system\_create and DRI\_Bufferset\_user\_create to take bufsize (in bytes) instead of DRI\_Distribution object parameter.
- This, of course, is because of the change described in item 1 (no dataspec is bound early on in the API - it is deferred until DRI\_Channel creation time)
9. Added built-in datatypes and DRI\_Dataspec\_get\_size
- \*\*\*\*\* Changes from March 2000 draft to July 2000 (POST JUNE DR MEETING) draft:
1. Changed capitalization conventions as decided in prior meetings, but has not yet been implemented. The new convention is:
    - DRI\_ prefix for everything (objects/types, function names, etc.)
    - DRI\_Data\_Type (capitalized first letter of object/type names)
    - DRI\_Data\_Type\_method (lowercase method/function names)
    - DRI\_Method (for standalone functions without associated objects)
  2. Changed DRI\_Distspec object to DRI\_Partition, per group's decision at the June 2000 meeting
 

An associated change is that the pre-defined object DRI\_DISTSPEC\_INDIVISIBLE has been changed to DRI\_PARTITION\_WHOLE

(2 changes - DISTSPEC to PARTITION, and INDIVISIBLE to WHOLE)
  3. Changed DRI\_Dist object to DRI\_Distribution, per group's decision at the June 2000 meeting
  4. Changed DRI\_Bufferset\_create to DRI\_Bufferset\_system\_create
  5. Modified DRI\_Bufferset\_system\_create function to take a DRI\_Distribution object parameter (as decided in June 2000 meeting)
  6. Created DRI\_Bufferset\_user\_create function to import user-allocated memory into a DRI\_Bufferset object (instead of calling DRI\_Bufferset\_system\_create

Nov 21, 00 16:12

DRI\_LIS\_09292000.txt

Page 4/25

to have the library/run-time perform the memory allocation).

7. Added DRI\_Init and DRI\_Finalize functions to the API specification
8. Specified a number of pre-defined "NULL" objects (one for each major data type in the API specification). DRI\_Init creates these NULL objects at runtime.
9. Added a new function DRI\_Partition\_whole\_create that allows the user to get a DRI\_Partition object that specifies no partitioning at all (per June meeting).
10. Changed DRI\_Group\_create parameters to require an "original\_group" from which a subset process group is to be created
11. REMOVED the following language from parts of this document, based on decisions made in June 2000 meeting.

<UNRESOLVED>

What if we are not using standard middleware, but a process set construct exists? Do we want to leverage those (non-portable) approaches to alleviate the need to do process set management in DRI? If we choose to do this, then this object and its methods become unnecessary?

</UNRESOLVED>

\*\*\*\*\* Changes from December 1999 draft to March 2000 (PRE DR MEETING) draft:

1. Marked appropriate functions as "<META>" to indicate areas where Data Reorg will likely use infrastructure from other middleware  
  
Each meta function now has a "META NOTES" section to try to organize the discussion on the meta-nature of the DRI API.
2. Inserted <UNRESOLVED> notation in this document where some remaining decisions are needed. Before final API is settled, we need to go back and remove these notes and replace with the final decisions
3. Inserted \_destroy functions for the following objects:  
DRI\_Global\_Data  
DRI\_Group  
DRI\_Overlap  
DRI\_Distspec  
DRI\_Bufferset  
DRI\_Channel.

\*\*\*\*\* Changes from September 1999 to December 1999 drafts:

1. DRI\_Group\_myrank name changed to DRI\_Group\_get\_rank()
2. DRI\_Dist\_create - added a note in RESTRICTIONS/POLICY section reiterating that this call may not involve collective communication, at the implementation's discretion. This of course means that erroneous programs could cause DRI\_Dist\_create to calculate an incorrect data partitioning.
3. DRI\_Dist\_create - Added useful default layout parameters so the user doesn't have to use DRI\_Layout\_create to create commonly-needed objects (e.g., DRI\_LAYOUT\_PACKED\_012). All possible packed layouts for 1, 2, and 3 dimensions are provided. This effectively makes DRI\_Layout\_create a non

"critical-path" function in the API.

4. DRI\_Dist\_create - A NULL pointer argument for the group\_dims parameter specifies that the user wants to have the implementation determine an appropriate logical process set topology to use in dividing up the data during the execution of this call.
5. DRI\_Dist\_create - Added a note in the DESCRIPTION section that says that a valid entry in the distspecs array parameter is DRI\_DISTSPEC\_INDIVISIBLE (this capability was accidentally removed from the API in prior edits).
6. DRI\_Dist\_create - Noted that the group\_dims array parameter corresponds directly to the dimsizes array parameter of the DRI\_Global\_Data\_create function.
7. Added a DRI\_Dist\_get\_numblocks function
8. Added a DRI\_Dist\_get\_blockinfo function to return a single structure that gives all needed information about a locally owned block of data following the partitioning process. Returns a new DRI\_Blockinfo structure type. internally, the DRI\_Blockinfo structure contains an array of DRI\_Blockdim structures. This pair of structures replaces the old DRI\_Part object and bounds\_t structure. DRI\_Part was not adding anything beyond DRI\_Dist, so we have opted directly query DRI\_Dist for the low level partitioning information. The bounds\_t structure was poorly named, and needed to be more descriptive (we now call it DRI\_Blockdim). Additional information was needed beyond what the old bounds\_t provided, so we just
9. Changed DRI\_Part\_calc\_local\_size to DRI\_Dist\_calc\_local\_size, since the DRI\_Part object has been removed and we now just query DRI\_Dist objects.
10. Added the DRI\_Bufferset object and its DRI\_Bufferset\_create function
11. DRI\_Transfer object has been renamed to DRI\_Channel

----- SECTION 3: Current API for "critical path" functions -----

```

/***** <STANDALONE/ADAPTER> DRI_Init *****/
DRI_Init - Initialize the data reorganization run-time environment

```

#### SYNOPSIS

```

DRI_Init(argvp, argcp) - C language binding
DRI_Init(???)         - other language bindings

```

#### PARAMETERS

```

INOUT: argvp (pointer to array of strings) - application command line
        arguments
INOUT: argcp (pointer to integer) - address of integer variable that
        stores the number of command line arguments contained in argvp

```

#### STANDALONE/ADAPTER NOTES

Co-layer (middleware adapter) implementations based on MPI or MPI/RT may be able to accomplish the necessary Data Reorg init actions within their respective Init functions, depending on how they are implemented.

## DESCRIPTION

Parses the application command line for implementation-specific data reorganization library options.

Synchronizes with all other data reorganization processes in the environment, and produces the DRI\_GROUP\_WORLD object that is used to represent all processes in a data reorganization based application.

If they are not already provided at compile-time, this function creates a pre-defined "null" objects:

- DRI\_GROUP\_NULL
- DRI\_GLOBAL\_DATA\_NULL
- DRI\_DATASPEC\_NULL
- DRI\_OVERLAP\_NULL
- DRI\_PARTITION\_NULL
- DRI\_DISTRIBUTION\_NULL
- DRI\_LAYOUT\_NULL
- DRI\_CHANNEL\_NULL
- DRI\_BUFFERSET\_NULL
- DRI\_BUFFER\_ID\_NULL

Also, if necessary, creates the following pre-defined objects:

- DRI\_PARTITION\_WHOLE

## COMMUNICATION BEHAVIOR

Collective. Synchronizes with all other data reorganization processes in the environment, and produces the DRI\_GROUP\_WORLD object that is used to represent all processes in a data reorganization based application.

## RESTRICTIONS / POLICY

DRI\_Init must be the first data reorganization library function called.

```

/***** DRI_Global_Data_create *****/
DRI_Global_Data_create - Create a global data object

```

## SYNOPSIS

```
DRI_Global_Data_create(ndims, dimsizes[ndims], name, gdo)
```

## PARAMETERS

- IN: ndims (integer) - number of dimensions in the global data
- IN: dimsizes (integer array) - size of each dimension of the global data
- IN: name (string) - symbolic name of data object represented by gdo
- OUT: gdo (DRI\_Global\_Data) - object that describes the global data

## DESCRIPTION

Creates a global data object to describe application data. The size information supplied by the user refers to the size of the application data `_without_` considering how the data will eventually be partitioned across a group of processes in the parallel environment.

Nov 21, 00 16:12

DRI\_LIS\_09292000.txt

Page 7/25

## COMMUNICATION BEHAVIOR

Local. All processes that will participate in a future data reorganization involving this data must create this object independently.

## RESTRICTIONS / POLICY

All processes that will participate in a data reorganization on the described data must call this function with identical ndims and dimsizes parameters. Implementations may place an upper limit on the ndims parameter. However, all implementations must minimally support  
 $1 \leq \text{ndims} \leq 3$

/\*\*\*\*\* <STANDALONE/ADAPTER> DRI\_Group\_create \*\*\*\*\*/  
 DRI\_Group\_create - Create an object to represent a group of processes

## SYNOPSIS

DRI\_Group\_create(original\_group, num\_ranks, rank\_list, new\_group)

## PARAMETERS

IN: original\_group (DRI\_Group) - group from which a subset will be taken to produce the new\_group of processes  
 IN: num\_ranks (integer) - total number of processes in the group to be created  
 IN: rank\_list (array of integer) - list of logical process ranks from the original\_group that will form the new\_group of processes  
 OUT: new\_group (DRI\_Group) - process group object

## STANDALONE/ADAPTER NOTES

This is meta at the "object level". That is, some implementations may choose to completely leverage constructs from other middleware APIs (e.g., MPI Communicators) as part of a co-layered implementation with Data Reorg. Implementations that take this approach may elect not to implement DRI\_Group objects in the standalone format shown here.

## DESCRIPTION

Creates an object to represent a group of unique processes in the parallel processing environment. Groups are one-dimensional logical orderings of processes. Each process is assigned an integer rank, numbered between zero and the total number of processes - 1. The original\_group parameter must be a valid data reorganization group. The pre-defined DRI\_Group object DRI\_GROUP\_WORLD must be used to create the first subset group of processes.

## COMMUNICATION BEHAVIOR

Local.

## RESTRICTIONS / POLICY

/\*\*\*\*\* <STANDALONE/ADAPTER> DRI\_Group\_get\_rank \*\*\*\*\*/  
 DRI\_Group\_get\_rank - Return the rank of the calling process in specified group

## SYNOPSIS

DRI\_Group\_get\_rank(group, rank)

Nov 21, 00 16:12

DRI\_LIS\_09292000.txt

Page 8/25

## PARAMETERS

IN: group (DRI\_Group) - group object  
 OUT: rank (integer) - rank of the calling process in the group

## STANDALONE/ADAPTER NOTES

See DRI\_Group\_create

## DESCRIPTION

Returns the rank (logical process id) in the given group to the caller.

## COMMUNICATION BEHAVIOR

Local

## RESTRICTIONS / POLICY

Only members of the specified group may call this function successfully

/\*\*\*\*\* <STANDALONE/ADAPTER> DRI\_Group\_get\_size \*\*\*\*\*/  
 DRI\_Group\_get\_size - Return the size of the specified group

## SYNOPSIS

DRI\_Group\_get\_size(group, size)

## PARAMETERS

IN: group (DRI\_Group) - group object  
 OUT: size (integer) - size of the specified group

## STANDALONE/ADAPTER NOTES

See notes for DRI\_Group\_create.

## DESCRIPTION

Returns the number of participating processes in the given group

## COMMUNICATION BEHAVIOR

Local

## RESTRICTIONS / POLICY

Only members of the specified group may call this function successfully

/\*\*\*\*\* DRI\_Overlap\_create \*\*\*\*\*/  
 DRI\_Overlap\_create - Create an overlap data partitioning object

## SYNOPSIS

DRI\_Overlap\_create(ovr\_type, num\_pos, overlaph)

Nov 21, 00 16:12

DRI\_LIS\_09292000.txt

Page 9/25

## PARAMETERS

IN: ovr\_type (DRI\_Overlap\_type) - overlap policy to implement at the edges of a global data object. Can be one of:

DRI\_OVERLAP\_TRUNCATE  
 DRI\_OVERLAP\_TOROIDAL  
 DRI\_OVERLAP\_PAD\_ZEROS  
 DRI\_OVERLAP\_PAD\_REPLICATED

IN: num\_pos (integer) - number of positions to overlap

OUT: overlaph (DRI\_Overlap) - overlap object

## DESCRIPTION

Creates the overlap attribute used in the data distribution high-level specification. The resulting DRI\_Overlap object is to be passed into either DRI\_Partition\_block\_create or DRI\_Partition\_blockcyclic\_create as a left or right overlap argument.

NOTE: Just like the DRI\_Partition object, the user is expected to create a DRI\_Overlap object specification for each dimension of global data (where a nonzero overlap is desired). In the event that no overlap is requested by the user, DRI\_NO\_OVERLAP can be passed as the left and right overlap arguments to one of the the DRI\_Partition\_<block|blockcyclic>create functions.

In general, overlap is the storage of extra data in a processor's local data buffer to hold data that is adjacent in the global data context, but that is assigned to another processor, based on the data partitioning. Overlap therefore refers to data that is stored on processor boundaries in the partitioning of the global data.

There are different overlap policies supported:

1) ovr\_type == DRI\_OVERLAP\_TRUNCATE

The local buffer should contain enough space to store copies of num\_pos adjacent, non-local elements. At the ends of the global data object, extra storage is not required in the local data buffer, and is truncated accordingly.

2) ovr\_type == DRI\_OVERLAP\_TOROIDAL

The local buffer should contain enough space to store copies of num\_pos adjacent, non-local elements. At the ends of the global data object, extra storage is required in the local data buffer, and will be filled with data from the num\_pos elements that start at the opposite end of the global data dimension.

3) ovr\_type == DRI\_OVERLAP\_PAD\_ZEROS

The local buffer should contain enough space to store copies of num\_pos adjacent, non-local elements. At the ends of the global data object, extra storage is required in the local data buffer, and will be filled with zeros.

4) ovr\_type == DRI\_OVERLAP\_PAD\_REPLICATED

The local buffer should contain enough space to store copies of num\_pos adjacent, non-local elements. At the ends of the global data object, extra storage is required in the local data buffer, and will be filled with a copy of the last num\_pos \_locally\_ held elements.

Nov 21, 00 16:12

DRI\_LIS\_09292000.txt

Page 10/25

## COMMUNICATION BEHAVIOR

Local.

## RESTRICTIONS / POLICY

```

/***** DRI_Partition_block_create *****/
/***** DRI_Partition_blockcyclic_create *****/
/***** DRI_Partition_whole_create *****/

```

DRI\_Partition\_block\_create - Create a block distribution specification  
DRI\_Partition\_blockcyclic\_create - Create a block cyclic distribution  
DRI\_Partition\_whole\_create - Create an indivisible (whole) distribution

## SYNOPSIS

```

DRI_Partition_block_create(minsz, mod, lov, rov, part)
DRI_Partition_blockcyclic_create(lov, rov, blksize, part)
DRI_Partition_whole_create(part)

```

## PARAMETERS

IN: minsz (integer) - minimum number of local elements required  
(user specifies 0 to indicate no preference)

IN: mod (integer) - modulo requirement  
(user specifies 1 to indicate no preference)

IN: lov (DRI\_Overlap) - left overlap (DRI\_NO\_OVERLAP specifies no overlap)

IN: rov (DRI\_Overlap) - right overlap (DRI\_NO\_OVERLAP specifies no overlap)

IN: blksize (integer) - block-cyclic partitioning block size  
(user specifies 1 for pure cyclic partition)

OUT: part (DRI\_Partition) - high-level data distribution object

## DESCRIPTION

These functions create a DRI\_Partition object that stores information about either a block, blockcyclic, or indivisible (whole) partitioning of global application data. Users must associate a separate DRI\_Partition object with each dimension of partitioned global data. The output object, part, is only a high-level specification of the requested data partitioning. It does not store exact partitioning details such as specific global data indices assigned to a particular process. Because a DRI\_Partition object is not associated with any single global data array, it can be reused for many different data partitionings. The more exact partitioning information for a particular global data array is stored in the DRI\_Distribution object that can be queried for detailed partitioning information following the DRI\_Distribution\_create operation.

Calling DRI\_Partition\_whole\_create will produce an object equivalent to the pre-defined object DRI\_PARTITION\_WHOLE. Implementations may in fact return a reference to this pre-defined object as the output of DRI\_Partition\_whole\_create.

Parameter mod specifies that the number of local elements ultimately assigned

Nov 21, 00 16:12

DRI\_LIS\_09292000.txt

Page 11/25

to the calling process must be some multiple of mod.

Parameters lov and rov specify element overlaps (left and right, respectively). These parameters do not change the mapping of global data indices to processors in the data partitioning. They allow copies of adjacent global data elements at the (left or right) boundaries of the data partitioning to be stored locally. A right overlap refers to overlap in the direction of `_higher_` global indices. Consult the section on the `DRI_Overlap` object for additional details about the overlap specification.

Parameter `blksz` is used in block-cyclic partitionings to define the size (in number of elements) of the blocks that get assigned to processors in the global data partitioning.

#### COMMUNICATION BEHAVIOR

Local

#### RESTRICTIONS / POLICY

This object may NOT be queried until the completion of a subsequent `DRI_Distribution_create` call.

#### COMMUNICATION BEHAVIOR

Local

```

/***** <STANDALONE/ADAPTER> DRI_Distribution_create *****/
DRI_Distribution_create - Create a distribution object for a
                        global data object over a group of processes

```

#### SYNOPSIS

```

DRI_Distribution_create(gdo, group_size, myrank, group_dims, parts, layout,
                        disth)

```

#### PARAMETERS

```

IN:  gdo (DRI_Global_Data) - global data object
IN:  group_size (integer) - total number of processes in group dividing data
IN:  myrank (integer) - my logical process rank within the group
IN:  group_dims (array of integer) - logical dimensions of process group
IN:  parts (array of DRI_Partition) - high-level data distribution specs
     (one array entry per gdo dimension)

IN:  layouts (DRI_Layout) - memory layout of local data buffers
OUT: disth (DRI_Distribution) - data distribution object

```

#### STANDALONE/ADAPTER NOTES

This function can be implemented in a completely standalone fashion

#### DESCRIPTION

This function aggregates all of the input objects into a single container, a `DRI_Distribution` object. It also calculates explicitly the data block(s) of the global data that will be assigned to processes (and stores that detailed information in the resulting `DRI_Distribution` object). The user will be able to query this low-level information following the execution of this call. Note

that the data partitioning performed here guarantees that each global data element is assigned to a process. It is unlikely, but possible that some processes could be assigned NO global data elements as a result of this call.

The `group_size` parameter defines the size of the group (number of processes) that will be dividing the data set.

The `myrank` parameter uniquely defines the calling process so that a unique portion of the global data set can be assigned to it by this call.

The `group_dims` array specifies a logical process set dimensionality for the process group identified by the "group" parameter. The number of elements in `group_dims` must be equal to the number of dimensions specified for the `gdo` parameter in a prior call to `DRI_Global_Data_create`. The product of all values in `group_dims` must equal the total number of processes, defined by the `group_size` parameter. `group_dims` gives the caller more explicit control over the global data partitioning process performed by `DRI_Distribution_create`.

The `group_dims` array can take one of three forms:

1. NULL (no process set dimensionality specified)  
User has no preference and the implementation can choose any values that make the product of `group_dims` equal to `group_size`
2. Mixture of zero and nonzero/positive values  
Nonzero/positive values represent a specific process set dimensionality that should be respected for the associated global data dimensions. Zero valued elements effectively represent a "don't care", and the implementation is free to select values as long as they satisfy the overall requirement that the `group_dims` product be equal to the `group_size`
3. All nonzero/positive values  
User is specifying a specific process set dimensionality that should be used in the partitioning process.

The `layouts` parameter specifies, for each dimension of the global data represented by `gdo`, how the locally stored data is to be arranged in linear memory space. The form of the `layouts` array parameter will control the following 2 characteristics of locally stored data:

- "ordering" of multi-dimensional data  
(e.g., which dimension is ordered "fastest" in linear memory)
- "striding" of local data (with or without strides between consecutive values in memory)

The `layouts` array can take one of three forms:

1. NULL array (no layout specification for any of the data dimensions)
  - Assigns natural ordering of multidimensional data in linear memory. The most contiguous dimension corresponds to the first specified dimension in the `dimsizes` input array to the earlier `DRI_Global_Data_create` call
  - the data in each dimension is stored with no strides between consecutive data values

Nov 21, 00 16:12

DRI\_LIS\_09292000.txt

Page 13/25

2. Individual elements with value DRI\_LAYOUT\_NULL
  - for layouts array elements not equal to DRI\_LAYOUT\_NULL, the order and striding of the corresponding data dimensions are defined by the properties of the supplied DRI\_Layout objects
  - for a layouts array element equal to DRI\_LAYOUT\_NULL, the order of the corresponding data dimensions is equal to its position within the layouts array. This is similar to the first case.
  - the data in the dimensions whose layouts are specified DRI\_LAYOUT\_NULL are stored with no strides between consecutive data values
3. Fully populated array of DRI\_Layout objects created earlier using the DRI\_Layout\_create call
  - both multidimensional ordering and striding are defined precisely by the supplied DRI\_Layout objects stored in the array

The parts parameter is an array of DRI\_Partition objects, one entry per dimension of data being partitioned. The entries in the array are created prior to this call by using one of the DRI\_Partition\_<block|blockcyclic>\_create functions.

#### Special case:

If one of the data dimensions has a partitioning description (parts parameter) of "whole", but the group\_dims parameter specifies more than one process "dividing" the data, then the data distribution approach is to provide copies of all global data in that dimension to each of the processes in that dimension. This can effectively be used to implement a broadcast or replication of the affected data.

(see earlier DRI\_Partition\_create\_whole, and DRI\_PARTITION\_WHOLE descriptions)

#### COMMUNICATION BEHAVIOR

At the implementation's discretion, this can be performed either as a collective operation, or as a local operation.

#### RESTRICTIONS / POLICY

This function may or may not be implemented in a collective fashion. It is therefore possible that the constituent processes that make up the group could (erroneously) supply different specifications for the following important parameters: gdo, group\_dims, parts. In that case the resulting data distribution that is computed and stored in the DRI\_Distribution output parameter may be incorrect.

The user must be able to query the low-level partitioning details that result from this call immediately following completion of this call. This is true even if the implementation does not perform this call collectively among the affected processes.

```

/***** DRI_Distribution_get_numblocks *****/
DRI_Distribution_get_numblocks - Return the number of locally stored blocks
                                from a data partitioning

```

Nov 21, 00 16:12

DRI\_LIS\_09292000.txt

Page 14/25

## SYNOPSIS

```
DRI_Distribution_get_numblocks (disth, nblocks)
```

## PARAMETERS

```
IN:  disth (DRI_Distribution) - data distribution object
OUT: nblocks (integer) - number of blocks associated with the low-level
      partitioning referred to by the disth parameter
```

## DESCRIPTION

This function returns the number of blocks assigned as part of a low-level data partitioning (described by the disth parameter that was created in an earlier DRI\_Distribution\_create call). For block data partitionings, this function will return a value of 1 in the nblocks output parameter. For block-cyclic partitionings, a value greater than 1 may be returned in the nblocks parameter.

## COMMUNICATION BEHAVIOR

Local.

## RESTRICTIONS/POLICY

```
/***** DRI_Distribution_get_blockinfo *****/
DRI_Distribution_get_blockinfo - Get detailed information about a
                                local block of data
```

## SYNOPSIS

```
DRI_Distribution_get_blockinfo (disth, block_num, blockinfo)
```

## PARAMETERS

```
IN:  disth (DRI_Distribution) - data distribution object
IN:  block_num (integer) - local block number for which information is needed
OUT: blockinfo (DRI_Blockinfo) - returned structure containing detailed
      information about the block
```

## DESCRIPTION

For a specified low-level data partitioning object (disth) and local block index (block\_num), allocates and returns a structure to the user containing the following:

```
{
  ndims (integer) - number of dimensions in the local data block described
  first_offset (integer) - offset (in elements) from the beginning of the local
                          application's memory buffer to the first "owned"
                          element of this data block. It therefore in some
                          cases does not identify the first data element in
                          the block, since the first element in storage could
                          be the result of an overlapped data partitioning.
  elem_size (integer) - number of bytes per data element in the local block.
                        This can be obtained by querying other objects
                        (DRI_Global_Data and DRI_Partition), but is provided in
                        this structure for user convenience.
  dims[ndims] (array of DRI_Blockdim structures) - detailed information
```

Nov 21, 00 16:12

DRI\_LIS\_09292000.txt

Page 15/25

```

    (on a per-dimension basis) about the range of global indices covered
    by the local block of data referred to by this DRI_Blockinfo structure
}

```

The DRI\_Blockdim structure referred to above is defined as follows:

```

{
  lov (DRI_Overlap) - left overlap in this dimension
  rov (DRI_Overlap) - right overlap in this dimension
  global_begin_ix (integer) - global index of the first "owned" data element in
  the block in this dimension
  length (integer) - number of "owned" data elements in this dimension
  stride (integer) - number of elements between consecutive data elements
  in the local data buffer in this dimension.  If this value is 1, then
  the data is densely packed, with no spacing between consecutive elements.
}

```

#### COMMUNICATION BEHAVIOR

Local.

#### RESTRICTIONS/POLICY

```

/***** DRI_Distribution_get_local_count *****/
DRI_Distribution_get_local_count - Calculate size of local buffers associated
with one side of a data reorganization

```

#### SYNOPSIS

```
DRI_Distribution_get_local_count(disth, local_size)
```

#### PARAMETERS

```

IN:  disth (DRI_Distribution) - low-level data partitioning object
OUT: local_size (integer) - number of elements in the data
    buffers associated with this data distribution

```

#### DESCRIPTION

This function tells the caller the size of data buffers (in elements) associated with the disth data distribution. The returned local\_size parameter is calculated based on a combination of

- user-specified partitioning parameters
- user-specified memory layout parameters
- and implementation-imposed local memory layout policies

The number of elements returned in the local\_size parameter specifies the size of a memory buffer large enough to hold all local blocks from a data partitioning. This is relevant for block-cyclic partitionings, in which it is possible and likely that multiple blocks of data are assigned to a single process.

#### COMMUNICATION BEHAVIOR

Local.

#### RESTRICTIONS / POLICY

Nov 21, 00 16:12

DRI\_LIS\_09292000.txt

Page 16/25

```

/***** <STANDALONE/ADAPTER> DRI_Bufferset_system_create *****/
DRI_Bufferset_system_create - create shared application/library buffers
                             for processing and data reorganization

```

## SYNOPSIS

```
DRI_Bufferset_system_create (nbufs, bufsize, bufset)
```

## PARAMETERS

```

IN:  nbufs (integer) - number of buffers of size bufsize that will make up
     the buffer set to be created by this function
IN:  bufsize (integer) - number of bytes needed for each buffer in bufferset
     buffer sizes that will be created in the set
OUT: bufset (DRI_Bufferset) - buffer set object created

```

## STANDALONE/ADAPTER NOTES:

Because of similar constructs defined in other APIs (most notably MPI/RT), buffersets can be implemented in a middleware adapter, or in a completely standalone fashion, at the implementation's discretion.

## DESCRIPTION

Creates a buffer set object that will be associated with a later data reorganization (represented by a DRI\_Channel object). After this call, the user will never directly query or manipulate the DRI\_Bufferset object created. Once the association of the buffer set is made with a channel object (in a later call do DRI\_Channel\_create), all access to the buffer set's constituent buffers will be made through that associated channel object. In that interaction, the user will work with individual DRI\_Buffer\_Id objects that are obtained with a call to DRI\_Channel\_get and returned to the channel with a call to DRI\_Channel\_put. See the documentation for the get/put functions for additional details on buffer set management.

## COMMUNICATION BEHAVIOR

Local.

## RESTRICTIONS / POLICY

```

/***** <STANDALONE/ADAPTER> DRI_Bufferset_user_create *****/
DRI_Bufferset_user_create - create shared application/library buffers
                             from user-allocated memory
                             for processing and data reorganization

```

## SYNOPSIS

```
DRI_Bufferset_user_create (nbufs, buffer_ptrs[], bufsize, bufset)
```

## PARAMETERS

```

IN:  nbufs (integer) - number of buffers that will make up
     the buffer set to be created by this function
IN:  buffer_ptrs (array of pointer) - addresses of user-allocated buffers
IN:  bufsize (integer) - number of bytes needed for each buffer in bufferset
OUT: bufset (DRI_Bufferset) - buffer set object created

```

Nov 21, 00 16:12

DRI\_LIS\_09292000.txt

Page 17/25

## STANDALONE/ADAPTER NOTES

See notes for DRI\_Bufferset\_system\_create

## DESCRIPTION

Creates a buffer set object (to be used in conjunction with an associated DRI\_Channel object) from user-allocated memory. Although the application programmer will have access to the addresses of each buffer using this approach, "safe" use of these memory areas must be negotiated by calling DRI\_Channel\_get and DRI\_Channel\_put for the associated channel object.

## COMMUNICATION BEHAVIOR

Local.

## RESTRICTIONS / POLICY

The buffers that are supplied as input parameters here must provide a sufficient amount of memory to insure correct function of the associated channel operation (data reorg) to take place. See DRI\_Channel\_create\_send and DRI\_Channel\_create\_recv for guidance on how to appropriately size the buffers.

/\*\*\*\*\* <STANDALONE/ADAPTER> DRI Built-In Datatypes \*\*\*\*\*/

Equivalence table between DRI built in datatypes, and other middleware:

DRI	MPI(C/FORTRAN)	VSIPL Scalar	ANSI C
DRI_Dataspec	MPI_Datatype	Datatype	
-----	-----	-----	-----
DRI_FLOAT	MPI_FLOAT	vsipl_scalar_f	float
DRI_DOUBLE	MPI_DOUBLE	vsipl_scalar_d	double
DRI_COMPLEX	NA / MPI_COMPLEX	vsipl_cscalar_f	NA
DRI_DOUBLE_COMPLEX	NA / MPI_DOUBLE_COMPLEX	vsipl_cscalar_d	NA
DRI_COMPLEX_SPLIT	NA / NA	vsipl_cscalar_f	NA
DRI_DOUBLE_COMPLEX_SPLIT	NA / NA	vsipl_cscalar_d	NA
DRI_INTEGER	MPI_INTEGER	vsipl_scalar_i	int
DRI_SHORT	MPI_SHORT	vsipl_scalar_si	signed short int
DRI_UNSIGNED_SHORT	MPI_UNSIGNED_SHORT	vsipl_scalar_us	unsigned short int
DRI_LONG	MPI_LONG	vsipl_scalar_li	signed long int
DRI_UNSIGNINGED_LONG	MPI_UNSIGNED_LONG	vsipl_scalar_ul	unsigned long int

/\*\*\*\*\* <STANDALONE/ADAPTER> DRI\_Dataspec\_get\_size \*\*\*\*\*/

DRI\_Dataspec\_get\_size - Get the number of bytes needed to store a data type

## SYNOPSIS

DRI\_Dataspec\_get\_size (dataspec, nbytes)

## PARAMETERS

IN dataspec (DRI\_Dataspec) data type descriptor

OUT nbytes: (integer) number of bytes needed to store data of type described by the dataspec parameter

Nov 21, 00 16:12

DRI\_LIS\_09292000.txt

Page 18/25

## STANDALONE/ADAPTER NOTES

Datatypes are commonly implemented in other APIs (see table above). This is an area where DRI implementations can take advantage of this existing functionality, which will almost always offer a broader feature set compared to the standalone DRI interfaces for data types.

## DESCRIPTION

Returns amount of memory (in bytes) needed to store a single data element of the type specified in the dataspec parameter

## COMMUNICATION BEHAVIOR

Local.

## RESTRICTIONS / POLICY

```

/***** <STANDALONE/ADAPTER> DRI_Channel_create_send *****/
/***** <STANDALONE/ADAPTER> DRI_Channel_create_recv *****/
DRI_Channel_create - create data reorganization communication channel

```

## SYNOPSIS

There are two forms of this call:

```

DRI_Channel_create_send(name, srcDist, srcBufs, channel);
DRI_Channel_create_recv(name, destDist, destBufs, channel);

```

## PARAMETERS

```

IN name: (string/integer?) Identifier for the channel
IN dataspec (DRI_Dataspec) type of data stored in associated buffers
IN srcDist: (DRI_Distribution) distribution object on the send side
IN destDist: (DRI_Distribution) distribution object on the receive side
IN srcBufs: (DRI_Bufferset) send side data buffers
IN destBufs: (DRI_Bufferset) receive side data buffers
OUT channel: (DRI_Channel) Data reorganization (channel) object created

```

## STANDALONE/ADAPTER NOTES

Channel constructs appear in other APIs (e.g., MPI/RT) and so DRI\_Channel can be implemented in a middleware adapter or in a completely standalone fashion, at the discretion of the implementation.

## DESCRIPTION

The send channel object allows the calling process to participate in a data reorganization as a sender. The receive channel object has a similar (obvious) function. To properly set up data reorganizations in which the caller is both a sender and receiver of data, both forms must be called, resulting in two DRI\_Channel objects.

## COMMUNICATION BEHAVIOR

Nov 21, 00 16:12

DRI\_LIS\_09292000.txt

Page 19/25

Local. Processes create channel objects independently and in any order.

#### RESTRICTIONS / POLICY

Buffers supplied here are assumed to be large enough to contain all the data transferred. To find the appropriate size for these buffers, use the functions `DRI_Distribution_get_local_count` (number of elements) and `DRI_Dataspec_get_size` (number of bytes/element). Alternatively, the user can have the DRI implementation allocate the associated bufferset using the `DRI_Bufferset_system_create` call.

Currently, we assume that data reorganizations are either bi-partite (pipeline) or clique-based (Single Program Multiple Data). Intermediate cases, that is, partially overlapping process groups, are disallowed. If any process is both a sender and a receiver, all processes must be both senders and receivers, or an error will result at the time of the subsequent `DRI_Channel_connect` call.

On a given "side" of a channel (send or receive), all of the participating processes must provide buffersets that contain the same number of local buffers as every other process. The number of buffers on the send side of a channel `_can_` be different than the number of buffers in the bufferset associated with the receive side of the same channel. The reason for the restriction is to enable high performance implementations. The middleware will be able to compute in advance:

- the explicit pairings of send/rcv buffers in the data reorganizations to be performed with this channel
- the precise order in which the pairings will occur (if there are multiple buffers on the send and receive sides of the channel)

```

/***** <STANDALONE/ADAPTER> DRI_Channel_connect *****/
DRI_Channel_connect(chan) - Pipeline channel connect

```

#### SYNOPSIS

```
DRI_Channel_connect(chan)
```

#### PARAMETERS

INOUT chan: (DRI\_Channel) channel object to be connected

#### STANDALONE/ADAPTER NOTES

See notes for `DRI_Channel_create`.

#### DESCRIPTION

Enables a given pipeline data reorganization: calculates which processors are Sending to and receiving from which other processors.

#### COMMUNICATION BEHAVIOR

The connect call is a synchronization point between all processors in the send and receive sides of the data reorganization identified by the chan parameter: it is collective and blocking.

Nov 21, 00 16:12

DRI\_LIS\_09292000.txt

Page 20/25

## RESTRICTIONS / POLICY

Multiple channel objects must be connected in the correct order by the involved parties or deadlock may (will probably) result.

```
/****** <STANDALONE/ADAPTER> DRI_Channel_connect_sendrecv *****/
```

## SYNOPSIS

DRI\_Channel\_connect\_sendrecv(c\_send, c\_recv) - Clique channel connect

## PARAMETERS

INOUT c\_send: (DRI\_Channel) object managing the "send side" of a data reorg

INOUT c\_recv: (DRI\_Channel) object managing the "receive side" of a data reorg

## STANDALONE/ADAPTER NOTES

See notes for DRI\_Channel\_create.

## DESCRIPTION

Enables a given clique data reorganization: calculates which processors are Sending to and receiving from which other processors.

## COMMUNICATION BEHAVIOR

The connect call is a synchronization point between all processors in the send and receive sides of the given data reorganization: it is collective and blocking.

## RESTRICTIONS / POLICY

Multiple channel objects must be connected in the correct order by the involved parties or deadlock may (will probably) result.

```
/****** <STANDALONE/ADAPTER> DRI_Channel_get *****/
```

```
/****** <STANDALONE/ADAPTER> DRI_Channel_put *****/
```

## SYNOPSIS

DRI\_Channel\_get (chan, buf) - Receive data reorg buffer / Get free buffer

DRI\_Channel\_put (chan, buf) - Send data reorg buffer / Return used buffer

## PARAMETERS

INOUT chan: (DRI\_Channel) channel object managing a data reorganization

OUT buf: (DRI\_Buffer\_Id) handle to memory buffer

## STANDALONE/ADAPTER NOTES

See notes for DRI\_Channel\_create.

## DESCRIPTION

Discussion of DRI\_Channel\_get:

Nov 21, 00 16:12

DRI\_LIS\_09292000.txt

Page 21/25

If the channel object argument refers to the "receive side" of a data reorganization, this function returns a buffer that represents the received data from a set of sending processes. If the channel object argument refers to the "send side" of a data reorganization, this function returns an available buffer to the application so that it can produce the data that will be sent in a subsequent data reorganization operation.

Discussion of DRI\_Channel\_put:

If the channel object argument refers to the "send side" of a data reorganization, this function initiates the communication using the data provided in the input buffer argument. If the channel object refers to the "receive side" of a data reorganization, then this call simply returns the buffer to the DRI library so that it can be filled up with received data in a subsequent data reorganization operation.

General discussion of put and get in context:

In pipeline data reorganizations, incoming buffers are obtained by calling DRI\_Channel\_get with a "receive side" channel object input argument. If, after processing the received buffer, the program needs to send the data "downstream" in the pipeline, the same buffer can be used as input to a DRI\_Channel\_put call, but with a separate channel object (representing the "send side" of a different data reorganization). In cases where the calling program is at the beginning or end of an application pipeline, the buffer may be returned to the buffer set by calling DRI\_Channel\_put with the same channel object parameter that was used in the earlier DRI\_Channel\_get.

For clique data parallel applications, there are two channel objects associated with the same data reorganization (one for the send side, one for the receive side). To execute clique data reorganizations, the program calls DRI\_Channel\_get with the send-side channel object as input. The returned buffer is filled and a data reorganization is initiated with a call to DRI\_Channel\_put (passing again as input the send-side channel object and the buffer id). The program then calls DRI\_Channel\_get, using the second channel object (associated with the receive side of the data reorganization).

#### COMMUNICATION BEHAVIOR

DRI\_Channel\_get is a blocking call and does not return until a full buffer of received data is available

DRI\_Channel\_put is a non-blocking call and returns immediately to the calling application, regardless of whether the associated communication has completed. The channel object will manage the availability of the buffers associated with the data reorganization, protecting the buffer from future application use (via DRI\_Channel\_get) until the communication has completed and it is safe to reuse the buffer.

#### RESTRICTIONS / POLICY

It is possible to use the same DRI\_Channel object for two different data reorganizations when using a clique data-parallel design. The receive-side channel object from the first data reorganization executed can also act as the send-side channel object for a second, distinct data reorganization. This is permissible when the data buffer sizes do not change as a result of application processing between the

Nov 21, 00 16:12

DRI\_LIS\_09292000.txt

Page 22/25

two data reorganizations.

----- SECTION 4: Current API for remaining functions -----

/\*\*\*\*\* DRI\_Global\_Data\_destroy \*\*\*\*\*/  
DRI\_Global\_Data\_destroy - destroy a global data object

#### SYNOPSIS

DRI\_Global\_Data\_destroy(global\_data)

#### PARAMETERS

INOUT: global\_data (DRI\_Global\_Data) - object that describes the global data

#### DESCRIPTION

Destroys the global data object referred to by the global\_data input parameter.

#### COMMUNICATION BEHAVIOR

Local.

#### RESTRICTIONS / POLICY

This function should only free resources associated with the global\_data object when necessary. That is, all references to the global data object must be "destroyed" via this call before the actual internal resources used by the global data object are freed and returned to the system.

/\*\*\*\*\* <STANDALONE/ADAPTER> DRI\_Group\_destroy \*\*\*\*\*/  
DRI\_Group\_destroy - Destroy an object representing a group of processes

#### SYNOPSIS

DRI\_Group\_destroy(grp)

#### PARAMETERS

INOUT: grp (DRI\_Group) - process group object

#### STANDALONE/ADAPTER NOTES

See notes for DRI\_Group\_create.

#### DESCRIPTION

Destroys the process set group object referred to by the grp input parameter.

#### COMMUNICATION BEHAVIOR

Local.

#### RESTRICTIONS / POLICY

This function should only free resources associated with the group object when necessary. That is, all references to the group object must be "destroyed" via this call before the actual internal resources

Nov 21, 00 16:12

DRI\_LIS\_09292000.txt

Page 23/25

used by the group object are freed and returned to the system.

```

/***** DRI_Overlap_destroy *****/
DRI_Overlap_destroy - Destroy an overlap data partitioning object

```

## SYNOPSIS

```
DRI_Overlap_destroy(ov)
```

## PARAMETERS

```
INOUT: ov (DRI_Overlap) - overlap object
```

## DESCRIPTION

Destroys the object referred to by the ov parameter

## COMMUNICATION BEHAVIOR

Local.

## RESTRICTIONS / POLICY

This function should only free resources associated with the overlap object when necessary. That is, all references to the overlap object must be "destroyed" via this call before the actual internal resources used by the overlap object are freed and returned to the system.

```

/***** DRI_Partition_destroy *****/
DRI_Partition_destroy - Destroy a data distribution specification object

```

## SYNOPSIS

```
DRI_Partition_destroy(part)
```

## PARAMETERS

```
INOUT: part (DRI_Partition) - high-level data distribution object
```

## DESCRIPTION

Destroys the object referred to by the part parameter. This parameter can refer to either a block or block-cyclic distribution object (created by DRI\_Partition\_block\_create or DRI\_Partition\_blockcyclic\_create, respectively).

## COMMUNICATION BEHAVIOR

Local

## RESTRICTIONS / POLICY

This function should only free resources associated with the part object when necessary. That is, all references to the part object must be "destroyed" via this call before the actual internal resources used by the part object are freed and returned to the system.

Nov 21, 00 16:12

DRI\_LIS\_09292000.txt

Page 24/25

```

/***** <STANDALONE/ADAPTER> DRI_Bufferset_destroy *****/
DRI_Bufferset_destroy - destroy shared application/library buffers
                        for processing and data reorganization

```

## SYNOPSIS

```
DRI_Bufferset_destroy (nbufs, bufsize bufset)
```

## PARAMETERS

```
INOUT: bufset (DRI_Bufferset) - buffer set object destroyed
```

## STANDALONE/ADAPTER NOTES

```
See DRI_Bufferset_create notes.
```

## DESCRIPTION

```
Destroys the object referred to by the bufset parameter.
```

## COMMUNICATION BEHAVIOR

```
Local.
```

## RESTRICTIONS / POLICY

```
This function should only free resources associated with the bufferset
object when necessary. That is, all references to the bufferset object
must be "destroyed" via this call before the actual internal resources
used by the bufferset object are freed and returned to the system.
```

```

/***** <STANDALONE/ADAPTER> DRI_Channel_destroy *****/
DRI_Channel_destroy - destroy data reorganization communication channel

```

## SYNOPSIS

```
DRI_Channel_destroy(chan);
```

## PARAMETERS

```
INOUT chan: (DRI_Channel) Data reorganization (channel) object destroyed
```

## STANDALONE/ADAPTER NOTES

```
See DRI_Channel_create notes.
```

## DESCRIPTION

```
Destroys the channel referred to by the chan parameter. Frees all internal
resources used by the channel, including temporary buffers that may
have been created during the earlier DRI_Channel_connect() call.
```

## COMMUNICATION BEHAVIOR

```
<UNRESOLVED>
```

```
It would be nice to be able to "shut down" a channel gracefully
(i.e., in a "collective" fashion). This could be difficult with
respect to process synchronization in pipeline application
architectures, where many processes participate in two data
reorganization channels. This scenario forces a specific order in
```

Nov 21, 00 16:12

DRI\_LIS\_09292000.txt

Page 25/25

which channels must be destroyed (or else deadlock could occur). Since this will apparently be pushed to the application level, a proposal would be to make DRI\_Channel\_destroy have local communication behavior, and to have applications use other middlewares for the necessary "graceful synchronization".

</UNRESOLVED

#### RESTRICTIONS / POLICY

This destroy call is different than most others because all internal resources associated with the channel can be freed without checking for other "references". Channels in the Data Reorganization API cannot be referenced more than once.

```

/***** <STANDALONE/ADAPTER> DRI_Finalize *****/
DRI_Finalize - Free resources used by the data reorganization
                run-time environment

```

#### SYNOPSIS

```
DRI_Finalize()
```

#### PARAMETERS

#### STANDALONE/ADAPTER NOTES

Co-layer (middleware adapter) implementations based on MPI or MPI/RT may be able to accomplish the necessary Data Reorg finalize actions within their respective Finalize functions, depending on how they are implemented.

#### DESCRIPTION

Frees any internal resources used by the data reorganization implementation.

#### COMMUNICATION BEHAVIOR

Local

#### RESTRICTIONS / POLICY