

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 June 1999 meeting.

Unanswered questions:

Overlap in the blockcyclic case was not clearly defined at the meeting (e.g., does it mean adjacent blocks or adjacent elements?)

'Do' is ordered sequentially. The acquire/insert extract/release calls are less well defined. Is the user allowed to acquire more than one buffer from a transfer object? If so, what if he inserts them in the wrong order? Same questions with extract and release.

Can we allow two different transfer objects on the same processor to share buffers with each other, without requiring the bufiter/buffer object hierarchy and monolithic commit of MPI/RT?

```
/*  
 * Global Data Object  
 */
```

```
/* dri_gdo_create */  
dri_gdo_create - Create a global data object
```

SYNOPSIS

```
dri_gdo_create(ndims, dimsizes[ndims], dataspec, 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: dataspec (DRI_dataspec) - data type of each element of the global data
OUT: gdo (DRI_gdo) - 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

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, dimsizes, and dataspec parameters. Implementations may place an upper limit on the ndims parameter. However, all implementations must minimally support $1 \leq ndims \leq 3$

```
/* dri_gdo_get_ndims */  
dri_gdo_get_ndims - Query a global data object to get the number of dimensions
```

SYNOPSIS

dri_gdo_get_ndims (gdo, ndims)

PARAMETERS

IN: gdo (DRI_gdo) - global data object
OUT: ndims (integer) - number of dimensions of the global data object

DESCRIPTION

A gdo query function that returns the number of dimensions in the global application data associated with the gdo

COMMUNICATION BEHAVIOR

Local

RESTRICTIONS / POLICY

/***** dri_gdo_get_dimsize *****/
dri_gdo_get_dimsize - Get size of a specified dimension of a global data object

SYNOPSIS

dri_gdo_get_dimsize(gdo, dim, dimsize)

PARAMETERS

IN: gdo (DRI_gdo) - global data object
IN: dim (integer) - dimension for which the size is requested
OUT: dimsize (integer) - size of the requested dimension

DESCRIPTION

A gdo query function that allows the user to determine the size of any dimension of the global application data associated with the gdo

COMMUNICATION BEHAVIOR

Local

RESTRICTIONS / POLICY

/***** dri_gdo_get_dimsizes *****/
dri_gdo_get_dimsizes - Get dimension sizes of a global data object

SYNOPSIS

dri_gdo_get_dimsizes(gdo, dimsizes[])

PARAMETERS

IN: gdo (DRI_gdo) - global data object
OUT: dimsizes (array of integers) - array containing sizes of each dimension

DESCRIPTION

Returns the size of global application data associated with the gdo

COMMUNICATION BEHAVIOR

Local

RESTRICTIONS / POLICY

/***** dri_gdo_get_dataspec *****/
dri_gdo_get_dataspec - Get datatype of a global data object

SYNOPSIS

dri_gdo_get_dataspec(gdo, dataspec)

PARAMETERS

IN: gdo (DRI_gdo) - global data object
OUT: dataspec (DRI_dataspec) - data type

DESCRIPTION

Returns the datatype of the global data associated with the gdo

COMMUNICATION BEHAVIOR

Local

RESTRICTIONS / POLICY

/***** dri_gdo_destroy *****/
dri_gdo_destroy - destroy a global data object

SYNOPSIS

dri_gdo_destroy(gdo)

PARAMETERS

IN: gdo (DRI_gdo) - global data object

DESCRIPTION

Destroys the global data object gdo. Frees any memory used internally by the gdo. This function does not destroy application data buffers associated with the gdo.

COMMUNICATION BEHAVIOR

Local

RESTRICTIONS / POLICY

/*****
* Process Group Object
*****/

/***** dri_group_create *****/
dri_group_create - Create an object to represent a group of processes

SYNOPSIS

dri_group_create(nprocs, procs[nprocs], group)

PARAMETERS

IN: nprocs (integer) - total number of processes in the group
IN: procs (array of integers) - array of unique process identifiers
OUT: group (DRI_group) - process group object

DESCRIPTION

Creates an object to represent a group of unique processes in the parallel processing environment. The group is a one-dimensional ordering of processes. The procs array must consist of a list of unique identifiers for each process in the group to be formed by this call. Because this is a meta-API, the intent of the group object is to be the same object used by the underlying (or co-) layer, such as MPI or MPI/RT. Also, an implication is that the procs array may need to be specified differently depending on the middleware that is used to implement the data reorganization interface. For example, in

a general-purpose workstation MPI environment, it may be sufficient for the user to specify a list of MPI ranks in the procs array. In some embedded environments (or others) that involve multiple programs in the run-time environment, more precise information (e.g., system-assigned process id) may be required in the procs array.

COMMUNICATION BEHAVIOR

Local.

RESTRICTIONS / POLICY

A process gets its group rank (logical id within the group) based on where it is referenced in the procs array. The first process listed in procs will be assigned rank 0, for example. Ranks are therefore assigned values in the range 0..(nprocs-1). In order for all processes that call this function to have a consistent understanding of process ranks, the procs array must be specified identically by all callers.

```
/****** dri_group_myrank *****/  
dri_group_myrank - Return the rank of the calling process in specified group
```

SYNOPSIS

```
dri_group_myrank(group, myrank)
```

PARAMETERS

IN: group (DRI_group) - group object
OUT: rank (integer) - rank of the calling process in the group

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

```
/****** dri_group_size *****/  
dri_group_size - Return the total number of processors in the specified group
```

SYNOPSIS

```
dri_group_size(group, size)
```

PARAMETERS

IN: group (DRI_group) - group object
OUT: size (integer) - number of processes in the specified group

DESCRIPTION

Returns the total number of processors in the specified group

COMMUNICATION BEHAVIOR

Local.

RESTRICTIONS / POLICY

```
/****** dri_group_destroy *****/  
dri_group_destroy - Destroy a group object
```

SYNOPSIS

```
dri_group_destroy(gdo)
```

PARAMETERS

IN: gdo (DRI_group) - group object

DESCRIPTION

Destroys the gdo group object. Frees memory used internally by the gdo.

COMMUNICATION BEHAVIOR

Local

RESTRICTIONS / POLICY

```
/* High-level Data Partitioning Specification Object
*****
/***** dri_distspec_create *****/
dri_distspec_create - Create a high-level data distribution specification
```

SYNOPSIS

```
dri_distspec_create(dist_type, nprocs, minsz, mod, lov, rov, blksize, distspec)
```

PARAMETERS

IN: dist_type (DRI_dist_type) - distribution policy being specified
can be one of:
DRI_DIST_INDIVISIBLE
DRI_DIST_BLOCK
DRI_DIST_BLOCKCYCLIC

IN: nprocs (integer) - number of processors dividing the data
(if 0, allows implementation to decide)
(ignored for dist_type==DRI_DIST_INDIVISIBLE)

IN: minsz (integer) - minimum number of local elements required
(user specifies 0 to indicate no preference)
(ignored when dist_type != DRI_DIST_BLOCK)

IN: mod (integer) - modulo request
(user specifies 1 to indicate no preference)
(ignored when dist_type != DRI_DIST_BLOCK)

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: distspec (DRI_distspec) - high-level data distribution object

DESCRIPTION

This function describes, at a high-level, the type of data distribution policy that will be applied to a single dimension of a global data object. Users will associate a separate DRI_dataspec object with each dimension of their global data that is being partitioned (in a subsequent call to the

dri_dist_create function). This is only a high-level specification object that is being created. It will NOT tell the user exactly which piece of global data it will receive in the actual data partitioning. The exact data partitioning information is stored in the DRI_part object that can be queried only after performing the dri_dist_create operation.

Parameter dist_type characterizes the type of partitioning requested. The DRI_dist_type datatype is generally implemented as an enumeration type, so no separate _create function is necessary for construction of this argument.

Parameter nprocs is used to specify how many processors will divide the data. If it is 0, then it allows the implementation to ultimately decide the logical process topology that will divide the data. In the event that nprocs is specified as 0, the user can find out exactly what the implementation ultimately decided by getting a DRI_part object after the completion of the dri_dist_create function.

Parameter mod is used only for block partitioning requests. This parameter specifies that the number of local elements ultimately assigned to the calling process must be a multiple of mod.

Parameters lov and rov specify overlaps (left and right, respectively). Overlap specifications allow the calling process to indicate the number of adjacent, but non-local data positions (elements) to be stored locally as copies. Generally, a left overlap refers to overlap in the direction of _lower_ global indices. 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 blksize is used in block-cyclic partitionings to define the size (in number of elements) of the blocks that get assigned to processors.

COMMUNICATION BEHAVIOR

Local

RESTRICTIONS / POLICY

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

Object-oriented language bindings should be allowed to use techniques such as overloaded constructors to distinguish between the different distribution types (INDIVISIBLE, BLOCK, BLOCKCYCLIC) and to simplify the calling syntax for users.

In the same spirit, procedural language bindings can define higher-level helper functions that eliminate some arguments (by providing reasonable defaults)

```
/* Overlap specification object
*****
/***** dri_overlap_create *****/
dri_overlap_create - Create an overlap data partitioning object
```

SYNOPSIS

```
dri_overlap_create(ovr_type, num_pos, overlap)
```

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: `overlap` (`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 the `dri_distspec_create` function as a left or right overlap argument.

NOTE: Just like the `DRI_distspec` 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 the `dri_distspec_create` function.

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, and 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.

COMMUNICATION BEHAVIOR

Local.

RESTRICTIONS / POLICY

```
/* Memory Layout Object
*****
```

```
***** dri_layout_create *****/
dri_layout_create - Create a local memory layout object
```

SYNOPSIS

```
dri_layout_create(ndims, dims_order, begin_align, repeat_align, layouth)
```

PARAMETERS

IN: ndims (integer) - number of dimensions in the local data buffer described by this layout

IN: dims_order (array of integer) - permutation array of dimensions of the local data buffer.

IN: begin_align (integer) - alignment of the beginning of the local data buffer described by this layout. Alignment value is given in bytes

IN: repeat_align (integer) - alignment of the data buffer at the start point of `_every_` position in the first dimension (the least-contiguous dimension - see discussion below). Alignment value is given in bytes

OUT: layouth - memory layout object

DESCRIPTION

Each position of the `dims_order` array references a dimension number in the range `0..(ndims-1)`. The contents of the array specify the ordering of the dimensions as they are laid out in linear memory space. The least-contiguous dimensions are listed first, with the most-contiguous dimension listed in the last position of the array.

The `begin_align` parameter specifies that the local data buffer is aligned so that the first memory position of the buffer starts at a multiple of `begin_align` number of bytes.

The `repeat_align` parameter specifies that the start position of each element in the least-contiguous dimension (specified in `dims_order[0]`) is aligned so that it starts at a multiple of `repeat_align` number of bytes.

COMMUNICATION BEHAVIOR

Local.

RESTRICTIONS / POLICY

```
/* Data Distribution Object
* - AND -
* Low-level Partitioning Object
*****
```

```
/****** dri_dist_create *****/
dri_dist_create - Create a distribution object for a specific global data
                  object over a specific process group
```

SYNOPSIS

```
dri_dist_create(gdo, group, distspecs, layout, disth)
```

PARAMETERS

```
IN:  gdo (DRI_gdo) - global data object
IN:  group (DRI_group) - process group
IN:  distspecs (array of DRI_distspec) - high-level data distribution specs
      (one array entry per gdo dimension)

IN:  layout (DRI_layout) - memory layout of local data buffers
OUT: disth (DRI_dist) - data distribution object
```

DESCRIPTION

This function aggregates all of the input objects into a single container, a DRI_dist object. It also calculates the specific data block(s) from the global data object that will be assigned to the calling process based on the high-level data partitioning object. The user will be able to query this low-level information following the execution of this call.

COMMUNICATION BEHAVIOR

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

RESTRICTIONS / POLICY

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 any communication between processes in the specified group during the execution of this function.

```
/****** dri_dist_get_mypart *****/
dri_dist_get_mypart - Get low-level partitioning information from a DRI_dist
                     object for the calling process
```

SYNOPSIS

```
dri_dist_get_mypart(disth, parth)
```

PARAMETERS

```
IN:  disth (DRI_dist) - distribution object created by dri_dist_create
OUT: parth (DRI_part) - specific low-level partitioning information
      indicating which part(s) of the global data are assigned to the
      calling process
```

DESCRIPTION

Following a dri_dist_create operation, the calling process can now find out exactly which piece(s) of the global data object have been assigned to it. During the dri_dist_create execution, a DRI_part object was created and stored internally to the DRI_dist object. This function allows the user to retrieve this low-level partitioning information as a standalone object. The DRI_part object can, in turn, be queried for this low-level information.

COMMUNICATION BEHAVIOR

Local.

RESTRICTIONS / POLICY

The user must be able to call this function immediately following a call to the `dri_dist_create` function.

```
/****** dri_dist_get_part *****/  
dri_dist_get_part - Get another low-level partitioning information for any  
                    process associated with a DRI_dist object
```

SYNOPSIS

```
dri_dist_get_part(disth, rank, parth)
```

PARAMETERS

IN: `disth` (`DRI_dist`) - data distribution object

IN: `rank` (`integer`) - integer rank of process for which the caller wants low-level partitioning information

OUT: `parth` (`DRI_part`) - low-level partitioning information object

DESCRIPTION

For the specified data distribution (which includes a specific global data object and process group over which the data gets distributed) and specific process rank, return that process's low-level partitioning information.

COMMUNICATION BEHAVIOR

Local.

RESTRICTIONS / POLICY

This function must be callable following the `dri_dist_create` call.

```
/****** dri_part_block_get_mybounds *****/  
dri_part_block_get_mybounds - Query low-level partitioning object for  
                              specific bounds assigned to the calling process  
                              (supports only block partitions)
```

SYNOPSIS

```
dri_part_block_get_mybounds(parth, ndims, bounds[ndims])
```

PARAMETERS

IN: `parth` (`DRI_part`) - low-level partitioning object

OUT: `ndims` (`integer`) - number of dimensions in local data buffer
(should equal number of dimensions in associated `gdo`)

OUT: `bounds` (array of `ndims` `bounds_t` structures)

DESCRIPTION

This function provides the caller with explicit information about which block of data was assigned to it in a previous call to the `dri_dist_create` function.

This call returns information that applies only to cases where the high-level partitioning specification (`DRI_distspec`) contained only `DRI_DIST_BLOCK` or `DRI_DIST_INDIVISIBLE`. In other words, this is a "helper" function that allows the user to avoid dealing with the "number of blocks" abstraction that is associated with more arbitrary block-cyclic partitionings.

The number of dimensions of the local data buffer are returned in parameter `ndims`. This should equal the number of dimensions in the associated global data object.

The exact bounds (in terms of global data indices) is returned in the `bounds` argument. This is an array of `bounds_t` structure elements, one per dimension, of the following form:

```
{
  left_ov (integer) - specifies the number of "left" overlap positions
  left_ix (integer) - specifies the first global index that is owned by
                      the calling process
  right_ix (integer) - specifies the last global index that is owned by
                      the calling process
  right_ov (integer) - specifies the number of "right" overlap positions
}
```

COMMUNICATION BEHAVIOR

Local.

RESTRICTIONS / POLICY

This function must be callable following the `dri_dist_create` call.

```
/****** dri_part_get_mybounds *****/
dri_part_get_mybounds - Query low-level partitioning object for
                       specific bounds assigned to the calling process
                       (universal support for both arbitrary/block-cyclic
                       and block-only high-level partitionings)
```

SYNOPSIS

```
dri_part_get_mybounds(parth, ndims, nlocal_blocks[ndims], bounds_lists[ndims])
```

PARAMETERS

IN: `parth` (`DRI_part`) - low-level partitioning object
OUT: `ndims` (integer) - # dimensions in the locally stored data block(s)
OUT: `nlocal_blocks` (array of integer) - For each dimension, specifies the number of blocks stored locally to the process associated with `parth`
OUT: `bounds_lists` (array of array of `bounds_t`) - array of "bounds" in each dimension of the dataset

DESCRIPTION

This function gives the low level "bounds" partitioning information for a process associated with the `parth` low-level partitioning descriptor. For each dimension of the dataset, an array of `bounds_t` structures is returned (one `bounds_t` element per block that is stored locally). For block-only data distributions, the `nlocal_blocks` array will contain the value 1 in each position. For arbitrary/block-cyclic data partitionings, it is possible (and likely) for the values in the `nlocal_blocks` array to differ. The values in the `nlocal_blocks` array correspond directly to the number of `bounds_t` elements that are stored in the `bounds_lists` output parameter.

The following pseudo-code demonstrates a possible use of the output parameters `ndims`, `nlocal_blocks`, and `bounds_lists`:

```
dri_part_get_mybounds(mypart, ndim, blists, nloc_blks)
```

```

for dim_ct = 0..(ndim-1)
  for blk_ct = 0..(nloc_blks[dim_ct])

    do something with information in blists[dim_ct][blk_ct]

  end // for blk_ct
end // for dim_ct

```

COMMUNICATION BEHAVIOR

Local.

RESTRICTIONS / POLICY

```

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

```

SYNOPSIS

```
dri_part_calc_local_size(parth, local_size)
```

PARAMETERS

IN: parth (DRI_part) - low-level partitioning object
 OUT: local_size (integer) - number of bytes specifying the size of data buffers that will be used in future data reorganizations

DESCRIPTION

This function tells the caller what size (in bytes) is required of application data buffers that participate in data transfers associated with the partitioning object parth. The returned local_size parameter is calculated based on the high-level partitioning parameters that were passed to the dri_dist_create call (e.g., distspec with overlap and layout specifications).

COMMUNICATION BEHAVIOR

Local.

RESTRICTIONS / POLICY

```

/***** dri_part_islocal *****/
dri_part_islocal - Determine if a data point specified in terms of global
                  indices is assigned to the calling process

```

SYNOPSIS

```
dri_part_islocal(parth, global_index, flag)
```

PARAMETERS

IN: parth (DRI_part) - low-level partitioning object
 IN: global_index (array of integer) - global coordinates of data point in question (one index per dimension of the global data object)
 OUT: flag (Boolean) - indicates whether specified data point is local to the calling process

DESCRIPTION

For a given partitioning description and global data point, determine whether the data point in question is assigned to (stored locally to) the calling

process.

COMMUNICATION BEHAVIOR

Local.

RESTRICTIONS / POLICY

The number of entries in the `global_index` array should minimally equal the number of dimensions in the global data object (`DRI_gdo`) that is associated with the `parth` partitioning descriptor.

```
/***** dri_part_get_globalindex *****/  
dri_part_get_globalindex - Get global data index from a specified local index
```

SYNOPSIS

```
dri_part_get_globalindex (parth, local_block, local_index, global_index)
```

PARAMETERS

IN: `parth` (`DRI_part`) - low-level partitioning object
IN: `local_block` (integer) - local block index
IN: `local_index` (array of integer) - local data coordinates
OUT: `global_index` (array of integer)- corresponding global data coordinates

DESCRIPTION

This function translates a locally owned data point into its corresponding coordinate position in the global data object (`DRI_gdo`) associated with the `parth` object.

COMMUNICATION BEHAVIOR

Local.

RESTRICTIONS / POLICY

The `local_index` and `global_index` array lengths are assumed to be equal to the number of dimensions in the global data object (`DRI_gdo`) associated with the `parth` object.

```
/***** dri_part_get_localindex *****/  
dri_part_get_localindex - Get local data index from a specified global index
```

SYNOPSIS

```
dri_part_get_localindex (parth, global_index, local_block, local_index)
```

PARAMETERS

IN: `parth` (`DRI_part`) - low-level partitioning object
IN: `global_index` (array of integer) - global data coordinates
OUT: `local_block` (integer) - local block index
OUT: `local_index` (array of integer) - corresponding local data coordinates

DESCRIPTION

This function translates a global data point into the corresponding locally owned data point.

COMMUNICATION BEHAVIOR

Local.

RESTRICTIONS / POLICY

The `local_index` and `global_index` array lengths are implicitly equal to the

number of dimensions in the global data object (DRI_gdo) associated with the parth object. An error status is returned when the global_index parameter does not correspond to any locally owned data.

```
/***** dri_dist_get_gdo *****/  
dri_dist_get_gdo - Get global data object associated with a DRI_dist object
```

SYNOPSIS

```
dri_get_gdo(disth, gdoh)
```

PARAMETERS

IN: disth (DRI_dist) - data distribution object
OUT: gdoh (DRI_gdo) - global data object

DESCRIPTION

Returns a global data object associated with this data distribution object.

COMMUNICATION BEHAVIOR

Local.

RESTRICTIONS / POLICY

```
/***** dri_dist_get_group *****/  
dri_dist_get_group - Get the process group associated with a DRI_dist object
```

SYNOPSIS

```
dri_dist_get_group(disth, grouph)
```

PARAMETERS

IN: disth (DRI_dist) - data distribution object
OUT: grouph (DRI_group) - process group object

DESCRIPTION

Returns the process group object that is associated with the disth object

COMMUNICATION BEHAVIOR

Local.

RESTRICTIONS / POLICY

```
/***** dri_dist_destroy *****/  
dri_dist_destroy - Destroy a data distribution object
```

SYNOPSIS

```
dri_dist_destroy(disth)
```

PARAMETERS

IN: disth (DRI_dist) - data distribution object

DESCRIPTION

Destroys the disth object. Includes the freeing of any memory used internally by the disth object, including the DRI_part object that is normally produced internally by the dri_dist_create function. Users should expect that references to DRI_part information will be invalid following a call to dri_dist_destroy on an associated distribution object.

COMMUNICATION BEHAVIOR

Local.

RESTRICTIONS / POLICY

```
/* Transfer object */
```

```
***** dri_transfer_create *****/
```

SYNOPSIS

There are three forms of this call:

```
dri_transfer_create_send(name, srcDist, nSrcBufs, srcBufs[], transfer);
dri_transfer_create_recv(name, destDist, nDestBufs, destBufs[], transfer);
dri_transfer_create_sendrecv(name, srcDist, nSrcBufs, srcBufs[], destDist,
                             nDestBufs, destBufs[], transfer);
```

PARAMETERS

IN name: (string/integer?) Identifier for the transfer
IN srcDist: (DRI_dist) distribution object on the send side
IN destDist: (DRI_dist) distribution object on the receive side
IN nSrcBufs: (integer) number of buffers associated with the transfer
on the send side
IN nDestBufs: (integer) number of buffers associated with the transfer
on the receive side
IN srcBufs: (array of pointers) send side data buffers
IN destBufs: (array of pointers) receive side data buffers
OUT transfer: (DRI_transfer) Transfer object created

DESCRIPTION

The send transfer object allows the given node to participate in a transfer as a sender. The receive transfer object has a similar (obvious) function. The sendrecv transfer object allows a node to participate both as a sender and as a receiver.

COMMUNICATION BEHAVIOR

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

RESTRICTIONS / POLICY

Buffers are assumed to be big enough to contain all the data transferred. To find the size of the buffer, use the function `dri_part_calc_local_size`.

Currently, we assume that transfers are either bi-partite or clique-based. 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 connect time.

```
***** dri_transfer_accessors *****/
```

SYNOPSIS

```
dri_transfer_get_name(transfer, name);
dri_transfer_get_srcDist(transfer, srcDist);
dri_transfer_get_nSrcBufs(transfer, nSrcBufs);
dri_transfer_get_srcBufs(transfer, srcBufs[]);

dri_transfer_get_destDist(transfer, destDist);
```

```
dri_transfer_get_nDestBufs(transfer, nDestBufs);
dri_transfer_get_destBufs(transfer, destBufs[]);
```

DESCRIPTION

These functions return the parameters passed in at the `dri_transfer_create` call. See that call for a description of each parameter.

COMMUNICATION BEHAVIOR

Local.

RESTRICTIONS / POLICY

All transfer query functions may be called on any type of Transfer object (`send`, `recv`, or `sendrecv`). The 'name' query function will return valid data for any type of transfer object. Both the source and destination query functions return valid information when used on a `sendrecv` Transfer object. Only the source set (respectively, destination set) returns valid data when used on a `send` (`recv`) Transfer object. `NULL` is returned when using the destination set (source set) of query functions on `send` (`recv`) Transfer objects.

```
/****** dri_transfer_connect *****/
```

SYNOPSIS

```
dri_transfer_connect(transfer);
```

PARAMETERS

INOUT `transfer`: (Transfer) transfer object to be connected

DESCRIPTION

Enables a given transfer for use: 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 transfer: it is collective and blocking.

RESTRICTIONS / POLICY

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

If any process involved in the transfer is both a sender and a receiver, all processes must be both senders and receivers, or an error will result here.

```
/****** dri_transfer_isconnected *****/
```

SYNOPSIS

```
dri_transfer_isconnected(transfer, flag);
```

PARAMETERS

IN `transfer`: (Transfer) transfer object
OUT `flag`: (boolean) connection indicator

DESCRIPTION

The `flag` returned is true if transfer object has been passed to the `connect` operation and has been successfully connected, false otherwise.

COMMUNICATION BEHAVIOR

Local.

RESTRICTIONS / POLICY

/***** dri_transfer_do *****/

SYNOPSIS

dri_transfer_do(transfer);

PARAMETERS

IN transfer: (Transfer) transfer object

DESCRIPTION

This is a blocking call that transfers data from sender to receiver. The appropriate buffers are used in a round-robin fashion on each side of the transfer.

COMMUNICATION BEHAVIOR

The call is blocking, that is, it returns when the local processor has finished sending or receiving data, or both, as appropriate. A 'send' buffer can be re-used after this call completes, and a 'recv' buffer will contain all received data after this call completes.

RESTRICTIONS / POLICY

/***** dri_transfer_buffer_available *****/

SYNOPSIS

dri_transfer_buffer_available(transfer, flag);

PARAMETERS

IN transfer: (Transfer) transfer object

OUT flag: (Boolean) availability indicator

DESCRIPTION

Flag is true if there is a buffer available to be acquired from the given send or sendrecv transfer object, false otherwise.

COMMUNICATION BEHAVIOR

Local.

RESTRICTIONS / POLICY

Can only be called on send and sendrecv transfer objects. Returns an error when called on a recv transfer object.

/***** dri_transfer_acquire_buffer *****/

SYNOPSIS

dri_transfer_acquire_buffer(transfer,buffer);

PARAMETERS

IN transfer: (Transfer) transfer object

OUT buffer: (pointer) Pointer to a buffer

DESCRIPTION

This call returns the user an empty buffer for use. Presumably, the user will fill the buffer with data and insert it.

COMMUNICATION BEHAVIOR

Local, blocking. To prevent blocking, test first with the `buffer_available` method, above.

RESTRICTIONS / POLICY

Can only be called on `send` or `sendrecv` transfer objects. Returns an error when called on a `recv` transfer object.

/***** dri_transfer_insert *****/

SYNOPSIS

```
dri_transfer_insert(transfer, buffer);
```

PARAMETERS

IN `transfer`: (Transfer) transfer object

IN `buffer`: (pointer) Pointer to the buffer used in the transfer

DESCRIPTION

This call requests that the data in the given buffer be transferred.

COMMUNICATION BEHAVIOR

This is a non-blocking call: if possible, it will return before the requested transfer is complete.

RESTRICTIONS / POLICY

The user must not try to access the memory in the buffer after insertion. Doing so may corrupt the transfer. The user should not alter the buffer until it is re-acquired using the `'acquire'` call above.

Can only be called on `send` or `sendrecv` transfer objects. Returns an error when called on a `recv` transfer object.

/***** dri_transfer_data_available *****/

SYNOPSIS

```
dri_transfer_data_available(transfer, flag);
```

PARAMETERS

IN `transfer`: (Transfer) transfer object

OUT `flag`: (Boolean) availability indicator

DESCRIPTION

Flag is true if there is data available to be extracted from the given transfer object, false otherwise.

COMMUNICATION BEHAVIOR

Local.

RESTRICTIONS / POLICY

Can only be called on `recv` or `sendrecv` transfer objects. Returns an error when called on a `send` transfer object.

/***** dri_transfer_extract *****/

SYNOPSIS

```
dri_transfer_extract(transfer, buffer);
```

PARAMETERS

IN `transfer`: (Transfer) transfer object

OUT `buffer`: (pointer) Pointer to the buffer used in the transfer

DESCRIPTION

This method returns the user a buffer filled with data received from the given transfer.

COMMUNICATION BEHAVIOR

This is a blocking call: to prevent blocking, test first with the `data_available` method, above.

RESTRICTIONS / POLICY

The buffer will not be re-used by the implementation until the user calls `dri_transfer_release_buffer`. If buffers are not released in a timely fashion, the implementation may run out of buffers.

Can only be called on `recv` or `sendrecv` transfer objects. Returns an error when called on a `send` transfer object.

/***** dri_transfer_release_buffer *****/

SYNOPSIS

```
dri_transfer_release_buffer(transfer, buffer);
```

PARAMETERS

IN `transfer`: (Transfer) transfer object

IN `buffer`: (pointer) Pointer to the buffer to be released

DESCRIPTION

This is a signal to the transfer object that the buffer is released to system control.

COMMUNICATION BEHAVIOR

Local.

RESTRICTIONS / POLICY

The buffer is assumed to be available to the implementation for future transfer operations. The user must not try to access the memory in the buffer after release, until the buffer is actually returned by an 'extract' operation. Doing so may corrupt the transfer.

Can only be called on `recv` or `sendrecv` transfer objects. Returns an error when called on a `send` transfer object.

/***** dri_transfer_destroy *****/

SYNOPSIS

```
dri_transfer_destroy(transfer);
```

PARAMETERS

IN `transfer`: (Transfer) transfer object

DESCRIPTION

Destroys the given transfer object.

COMMUNICATION BEHAVIOR

Local.

RESTRICTIONS / POLICY

It is an error to destroy a transfer object on a subset of the processes involved in the original connect call and then try to use any of the transfer operations (`do`, `extract`, `insert`, `acquire`, `release`...) on other

processes.