Reference Manual

ODYSSEUS/COSMOS

Version 3.0

Manual Release 1

Aug. 2016

Advanced Information Technology Research Center (AITrc)

KAIST

Contents

1. System Management

1.1. LRDS_Init

Syntax

Four LRDS_Init( )

Parameters

IN/OUT	Name	Type	Description
None			

Description

Initializes the COSMOS storage system.

Return value

eNOERROR : COSMOS has started normally.

< eNOERROR : Error code.

Example

```
#include "cosmos_r.h"

Four                e;

e = LRDS_Init();
if(e < eNOERROR) /* error handling */
……
e = LRDS_Final();
if(e < eNOERROR) /* error handling */
……
```

1.2. LRDS_Final

Syntax

Four LRDS_Final()

Parameters

IN/OUT	Name	Type	Description
None			

Description

Terminates the COSMOS storage system.

Return value

eNOERROR : COSMOS is terminated normally.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four                e;
```

```
e = LRDS_Init();
if(e < eNOERROR) /* error handling */
……
e = LRDS_Final();
if(e < eNOERROR) /* error handling */
……
```

1.3. LRDS_AllocHandle

Syntax

Four LRDS_AllocHandle(Four* handle)

Parameters

IN/OUT	Name	Type	Description
OUT	handle	Four*	Identifier for managing threads

Description

Assign a handle, which is used for identifying threads in a process. Most COSMOS APIs except LRDS_Init() and LRDS_Final() get a handle as the first argument. However, the coarse granularity locking no-thread version does not get the handle as an argument since it trivially has one unique handle.

Return value

eNOERROR : The handle is successfully assigned.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four               e;
Four                      handle;

……
e = LRDS_AllocHandle(&handle);
if(e < eNOERROR) /* error handling */
……
e = LRDS_FreeHandle(handle);
if(e < eNOERROR) /* error handling */
……
```

1.4. LRDS_FreeHandle

Syntax

Four LRDS_FreeHandle(Four handle)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads

Description

Returns the handle which is used to identify a thread in a process.

Return value

eNOERROR : The handle has been successfully returned.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four              e;
Four                    handle;

……
e = LRDS_AllocHandle(&handle);
if(e < eNOERROR) /* error handling */
……
e = LRDS_FreeHandle(handle);
if(e < eNOERROR) /* error handling */
……
```

1.5. LRDS_SetCfgParam

Syntax

Four LRDS_SetCfgParam(Four handle, char* name, char* value)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	name	char*	Name of the setting parameter
IN	value	char*	Value of the setting parameter

Description

Sets the values of setting parameters. There are LOG_VOLUME_DEVICE_LIST, COHERENCY_VOLUME_DEVICE, USE_DEADLOCK_AVOIDANCE, USE_BULKFLUSH for the setting parameters.

Return value

eNOERROR : The value of the setting parameter has been successfully assigned.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four              e;
Four                    handle;

……
e = LRDS_SetCfgParam(handle, “LOG_VOLUME_DEVICE_LIST”,
                     “/cosmos/log.vol”);
if(e < eNOERROR) /* error handling */
……
```

1.6. LRDS_GetCfgParam

Syntax

char* LRDS_GetCfgParam(Four handle, char* name)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	name	char*	Name of setting parameter

Description

Reads the values of the setting parameters. There are LOG_VOLUME_DEVICE_LIST, COHERENCY_VOLUME_DEVICE, USE_DEADLOCK_AVOIDANCE, USE_BULKFLUSH for the setting parameters.

Return value

value : A value of the setting parameter appointed by a name.

Example

```
#include "cosmos_r.h"

Four                    handle;
char*             value;

……
value = LOM_GetCfgParam(handle, “LOG_VOLUME_DEVICE_LIST”);
printf(“%s\n”, value);
……
```

1.7. LRDS_InitLocalDS
1.8. LRDS_InitSharedDS
1.9. LRDS_FinalLocalDS
1.10. LRDS_GetCfgParam

2. Volume Management

2.1. LRDS_Mount

Syntax

Four LRDS_Mount(Four handle, Four numDevices, char **devNames, Four *volId)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	numDevices	Four	Number of devices which configure the volume.
IN	devNames	char**	Array of device names.
OUT	volId	Four*	ID of the volume mounted

Description

Mounts the given volume so that the storage system can use it. Since a volume could consist of multiple devices, the number and names of devices configuring the volume are passed with an array as an input. The device name means the name in the UNIX file system. If the volume is successfully mounted, the volume's identifier is returned.

Return value

eNOERROR : The volume has been successfully mounted.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            e;
char            deviceNameStrings[2][256];
char**          deviceNames;
Four            volId;
……
strcpy(devNameStrings[0], "/device1-name")
strcpy(devNameStrings[1], "/device2-name")

devNames[0] = devNameStrings[0];
devNames[1] = devNameStrings[1];
e = LRDS_Mount(handle, 2, devNames, &volId);
if(e < eNOERROR) /* error handling */
……
e = LRDS_Dismount(handle, volId);
if(e < eNOERROR) /* error handling */
……
```

2.2. LRDS_Dismount

Syntax

Four LRDS_Dismount(Four handle, Four volId)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	volId	Four	ID of the database volume

Description

Dismounts the mounted volume. The volume identifier, which has been returned by mounting, specifies the volume to be dismounted.

Return value

eNOERROR : The volume has been successfully dismounted.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            e;
char            deviceNameStrings[2][256];
char**          deviceNames;
Four            volId;
……
strcpy(devNameStrings[0], "/device1-name")
strcpy(devNameStrings[1], "/device2-name")

devNames[0] = devNameStrings[0];
devNames[1] = devNameStrings[1];
e = LRDS_Mount(handle, 2, devNames, &volId);
if(e < eNOERROR) /* error handling */
……
e = LRDS_Dismount(handle, volId);
if(e < eNOERROR) /* error handling */
……
```

2.3. LRDS_FormatDataVolume

Syntax

Four LRDS_FormatDataVolume(Four handle, Four numDevices, char **devNames, char *title, Four volId, Four extSize, Four *numPagesInDevice, Four segmentSize)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	numDevices	Four	Number of devices in the volume
IN	devNames	char**	Names of devices in the volume
IN	title	char*	Name of the volume

IN	volId	Four	ID of the database volume
IN	extSize	Four	Size of extent (the type is Two in the coarse granularity locking version)
IN	numPagesInDevice	Four*	Number of pages in each device
IN	segmentSize	Four	Size of the segment

Description

Formats the data volume with the given devices. A volume consists of multiple devices.

Return value

eNOERROR : The data volume has been successfully formatted.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            e;
char            deviceNameStrings[2][256];
char**          deviceNames;
Four                  nPages[2];
Four            volId;
……
strcpy(devNameStrings[0], "/device1-name")
strcpy(devNameStrings[1], "/device2-name")

devNames[0] = devNameStrings[0];
devNames[1] = devNameStrings[1];
nPages[0] = 3200;
nPages[1] = 4800;
e = LRDS_FormatDataVolume(handle, 2, devNames, “test_volume”, 1005, 16,
nPages, 800);
if(e < eNOERROR) /* error handling */
……
```

2.4. LRDS_FormatLogVolume

Syntax

Four LRDS_FormatLogVolume(Four handle, Four numDevices, char **devNames, char *title, Four volId, Four extSize, Four *numPagesInDevice)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	numDevices	Four	Number of devices in the volume
IN	devNames	char**	Names of devices in the volume
IN	title	char*	Name of the volume

IN	volId	Four	Identifier of the log volume
IN	extSize	Four	Size of extent (the type is Two in the coarse granularity locking version)
IN	numPagesInDevice	Four*	Number of pages in each device

Description

Formats the log volume with the given devices. A log is used for recording the database's operation, and if the database system is terminated abnormally due to an external cause etc., it plays a role of restoring the database's content to its original condition. The log volume must be created if you want to use the roll back operation of transactions or the damage recovery function. If the log volume does not exist or is not specified, the database damage due to the transaction roll back or program's abnormal termination cannot be restored.

Return value

eNOERROR : The log volume has been successfully formatted.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            e;
char            deviceNameStrings[2][256];
char**          deviceNames;
Four                  nPages[2];
Four            volId;
……
strcpy(devNameStrings[0], "/device1-name")
strcpy(devNameStrings[1], "/device2-name")

devNames[0] = devNameStrings[0];
devNames[1] = devNameStrings[1];
nPages[0] = 800;
nPages[1] = 400;
e = LRDS_FormatLogVolume(handle, 2, devNames, “test_volume”, 1005, 16,
nPages);
if(e < eNOERROR) /* error handling */
……
```

2.5. LRDS_FormatTempDataVolume

Syntax

Four LRDS_FormatTempDataVolume(Four handle, Four numDevices, char **devNames, char *title, Four volId, Four extSize, Four *numPagesInDevice, Four segmentSize)

Parameters

IN/OUT	Name	Type	Description

IN	handle	Four	Identifier for managing threads
IN	numDevices	Four	Number of devices in the volume
IN	devNames	char**	Names of devices in the volume
IN	title	char*	Name of the volume
IN	volId	Four	Identifier of the log volume
IN	extSize	Four	Size of extent (the type is Two in the coarse granularity locking version)
IN	numPagesInDevice	Four*	Number of pages in each device
IN	segmentSize	Four	Size of the segment

Description

Formats the temporary data volume with the given devices. The temporary data volume is used to store temporary data when processing operations such as sort.

Return value

eNOERROR : The temporary data volume has been successfully formatted.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            e;
char            deviceNameStrings[2][256];
char**          deviceNames;
Four                  nPages[2];
Four            volId;
……
strcpy(devNameStrings[0], "/device1-name")
strcpy(devNameStrings[1], "/device2-name")

devNames[0] = devNameStrings[0];
devNames[1] = devNameStrings[1];
nPages[0] = 800;
nPages[1] = 400;
e = LRDS_FormatTempDataVolume(handle, 2, devNames, “test_volume”, 1005,
16, nPages, 200);
if(e < eNOERROR) /* error handling */
……
```

2.6. LRDS_FormatCoherencyVolume

Syntax

Four LRDS_FormatCoherencyVolume(Four handle, char *devName, char *title, Four volId)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	devName	char*	Names of devices in the volume
IN	title	char*	Name of volume
IN	volId	Four	Identifier of the log volume

Description

Formats the coherency volume with the given devices. The coherency volume is used for maintaining buffer coherency among processes in the multiple server environment, and is required only in the coarse-granularity locking version which does not use a shared memory.

Return value

eNOERROR : The coherency volume has been successfully formatted.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            e;
Four            volId;
……
e = LRDS_FormatCoherencyVolume(handle, "/devName", “test_volume”,
1005);
if(e < eNOERROR) /* error handling */
……
```

2.7. LRDS_ExpandDataVolume

Syntax

Four LRDS_ExpandDataVolume(Four handle, Four volId, Four numAddDevices, char **addDevNames, Four *numPagesInAddDevice)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	volId	Four	Identifier of the log volume
IN	numAddDevices	Four	Number of devices to be added to the volume
IN	addDevNames	char**	Names of devices to be added to the volume
IN	numPagesInAddDevice	Four*	Number of pages in each device

Description

Adds the devices to the given volume to expand the size of the volume. At

this time, the volume should have been mounted.

Return value

eNOERROR : The temporary data volume has been successfully formatted.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            e;
char            deviceNameStrings[2][256];
char            addDeviceNameStrings[2][256];
char*           devNames[2];
Four                  nPages[2];
Four            volId;
……
strcpy(devNameStrings[0], "/device1-name")
strcpy(devNameStrings[1], "/device2-name")

devNames[0] = devNameStrings[0];
devNames[1] = devNameStrings[1];

e = LRDS_Mount(handle, 2, devNames, &volId);
if(e < eNOERROR) /* error handling */

strcpy(addDevNameStrings[0], "/add_device1-name")
strcpy(addDevNameStrings[1], "/add_device2-name")

devNames[0] = addDevNameStrings[0];
devNames[1] = addDevNameStrings[1];

nPages[0] = 800;
nPages[1] = 400;

e = LRDS_ExpandDataVolume(handle, volId, 2, devNames, nPages);
if(e < eNOERROR) /* error handling */

e = LRDS_Dismount(handle, volId);
if(e < eNOERROR) /* error handling */
……
```

3. Transaction Management

3.1. LRDS_BeginTransaction

Syntax

Four LRDS_BeginTransaction(Four handle, XactID *xactId, ConcurrencyLevel ccLevel)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
OUT	xactId	XactID*	Transaction identifier
IN	ccLevel	ConcurrencyLevel	Concurrency control level to be used by the transaction

Description

Initializes a new transaction and declares the start of transaction. An identifier is assigned to identify the created transaction, and is returned by xactId.

The cclevel is a concurrency level to be used by the given transaction. When several transactions are concurrently executed, the concurrency level determines how to process them. The cclevel is a ConcurrencyLevel type, which is defined as follows. typedef enum { X_BROWSE_BROWSE, X_CS_BROWSE, X_CS_CS, X_RR_BROWSE, X_RR_CS, X_RR_RR } ConcurrencyLevel;

The current version of ODYSSEUS/COSMOS uses two concurrency levels such as X_BROWSE_BROWSE and X_RR_RR.

The X_BROWSE_BROWSE is a level to use no read lock and long write lock, which is used in the transactions which mostly do read. The transaction executed by X_BROWSE_BROWSE can execute the read operation for the given volume (data) even though other transactions process the write operation, and when other transaction does not process the write operation, it can process the write operation.

The X_RR_RR is long read lock and long_write_lock, which is used in the transactions which mostly do write. When another transaction executes the write operation, the X_RR_RR cannot execute the read operation for the given volume (data), and when another transaction does not process the read operation at the X_RR_RR level, it can process the write operation. And when another transaction does not process the write operation, it can process the write operation.

Return value

eNOERROR : The transaction has been successfully started.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"
```

```
Four            handle;
Four            e;
XactID          xactID;
……
e = LRDS_BeginTransaction(handle, &xactID, X_RR_RR);
if(e < eNOERROR) /* error handling */
……
e = LRDS_CommitTransaction(handle, &xactID);
if(e < eNOERROR) /* error handling */
……
```

3.2. LRDS_CommitTransaction

Syntax

Four LRDS_CommitTransaction(Four handle, XactID *xactId)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	xactId	XactID*	Transaction identifier

Description

Completes the given transaction. When the transaction is completed, the database-related operations executed by transactions are actually reflected to the database.

Return value

eNOERROR : The transaction has been successfully completed.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            e;
XactID          xactID;
……
e = LRDS_BeginTransaction(handle, &xactID, X_RR_RR);
if(e < eNOERROR) /* error handling */
……
e = LRDS_CommitTransaction(handle, &xactID);
if(e < eNOERROR) /* error handling */
……
```

3.3. LRDS_AbortTransaction

Syntax

Four LRDS_AbortTransaction(Four handle, XactID *xactId)

Parameters

IN/OUT	Name	Type	Description

IN	handle	Four	Identifier for managing threads
IN	xactId	XactID*	Transaction identifier

Description

Aborts the given transaction. When a transaction is aborted, all the database-related operations executed by transactions are cancelled, and the database state returns to the state before the transaction started.

Return value

eNOERROR : The transaction has been successfully aborted.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            e;
XactID          xactID;
……
e = LRDS_BeginTransaction(handle, &xactID, X_RR_RR);
if(e < eNOERROR) /* error handling */
……
e = LRDS_AbortTransaction(handle, &xactID);
if(e < eNOERROR) /* error handling */
……
```

3.4. LRDS_SetSavepoint

Syntax

Four LRDS_SetSavepoint(Four handle, SavepointID* spID)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
OUT	spID	SavepointID*	Savepoint identifier

Description

An API existing only in the fine granularity locking version, which sets a savepoint for the given transaction. You can abort the database-related operations executed after the savepoint by calling LRDS_RollbackSavepoint().

Return value

eNOERROR : The savepoint has been successfully set.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
```

```
Four            e;
SavepointID           spID;
……
e = LRDS_SetSavepoint(handle, &spID);
if(e < eNOERROR) /* error handling */
……
e = LRDS_RollbackSavepoint(handle, spID);
if(e < eNOERROR) /* error handling */
……
```

3.5. LRDS_RollbackSavepoint

Syntax

Four LRDS_RollbackSavepoint(Four handle, SavepointID spID)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	spID	SavepointID	Savepoint identifier

Description

An API existing only in the fine granularity locking version, which roll backs all the database-related operations executed after the time when the savepoint was set, and the database state returns to the state before the savepoint was set.

Return value

eNOERROR : Transactions after the savepoint have been successfully aborted.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            e;
SavepointID           spID;
……
e = LRDS_SetSavepoint(handle, &spID);
if(e < eNOERROR) /* error handling */
……
e = LRDS_RollbackSavepoint(handle, spID);
if(e < eNOERROR) /* error handling */
……
```

4. Relation and Index Management

4.1. LRDS_CreateRelation

Syntax

Four LRDS_CreateRelation(Four handle, Four volId, char* relName, LRDS_IndexDesc* idesc, Four nCols, ColInfo* cinfo, Boolean tmpRelationFlag)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	volId	Four	Volume identifier
IN	relName	char*	Name of relation to be created
IN	idesc	LRDS_IndexDesc*	Descriptor for the clustering index
IN	nCols	Four	Number of columns (the type is Two in the coarse granularity locking version)
IN	cinfo	ColInfo*	Information of columns
IN	tmpRelationFlag	Boolean	Indicates whether it is a temporary relation or not

Description

Makes a new relation. A relation consists of multiple columns, and can have a clustering index. Each column has a column number starting from 0, and this column number can be used to access the desired column. In order to create a new relation, you should specify the number of columns in the relation, the column type for each column, and the maximum length of the column value for each column. If you want to specify a clustering index, you should specify information(i.e., the column numbers corresponding the key of the index) for the clustering index. You can specify multiple column numbers since the index supports a composite key.

Return value

eNOERROR : The relation has been successfully created.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            e;
Four            volId;
LRDS_IndexDesc  idesc;
ColInfo         cinfo[2];
……
idesc.indexType = SM_INDEXTYPE_BTREE;
idesc.btree.flag = KEYFLAG_CLUSTERING;
idesc.btree.nColumns = 1;
```

```
idesc.btree.columns[0].colNo = 0;
idesc.btree.columns[0].flag = KEYINFO_COL_DESC;

cinfo[0].complexType = SM_COMPLEXTYPE_BASIC;
cinfo[0].type = SM_INT;

cinfo[1].complexType = SM_COMPLEXTYPE_BASIC;
cinfo[1].type = SM_STRING;
cinfo[1].length = 10;

e = LRDS_CreateRelation(handle, volId, "new_relation", &idesc, 2, cinfo,
SM_FALSE);
if(e < eNOERROR) /* error handling */
……
```

4.2. LRDS_DestroyRelation

Syntax

Four LRDS_DestroyRelation(Four handle, Four volId, char* relName)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	volId	Four	Volume identifier
IN	relName	char*	Name of relation to be deleted

Description

Deletes the given relation from the database. The relation to be deleted is specified with the relation name and the identifier for the volume in which the relation is located. Before deleting a relation, it checks whether the relation is opened or not, and if the relation is opened, the relation is not deleted and the function returns. If the relation is not opened (i.e., the file is not in use), it deletes the files (both data file and index file) for the given relation from the storage system. Then, the tuples for the deleted relation are deleted in the catalog table.

Return value

eNOERROR : The relation has been successfully deleted.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            e;
Four            volId;
……
e = LRDS_DestroyRelation(handle, volId, "new_relation");
if(e < eNOERROR) /* error handling */
……
```

4.3. LRDS_AddIndex

Syntax

Four LRDS_AddIndex(Four handle, Four volId, char* relName, LRDS_IndexDesc* idesc, IndexID* iid)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	volId	Four	Volume identifier
IN	relName	char*	Name of the relation
IN	idesc	LRDS_IndexDesc*	Descriptor for the index
OUT	iid	IndexID*	Index identifier

Description

Adds a new index to the relation. In order to define a new index, information about a key used for the index(i.e., numbers of the columns corresponding to the key) should be delivered as a parameter. You can specify multiple column numbers as a key since the storage system supports a multi-key (a key composed of multiple columns).

Return value

eNOERROR : The index has been successfully created.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            e;
Four            volId;
LRDS_IndexDesc  idesc;
IndexID        iid;
……
idesc.indexType = SM_INDEXTYPE_BTREE;
idesc.btree.flag = KEYFLAG_CLEAR;
idesc.btree.nColumns = 2;

idesc.btree.columns[0].colNo = 4;
idesc.btree.columns[0].flag = KEYINFO_COL_DESC;

idesc.btree.columns[1].colNo = 0;
idesc.btree.columns[1].flag = KEYINFO_COL_ASC;

e = LRDS_AddIndex(handle, volId, “new_relation”, &idesc, &iid);
if(e < eNOERROR) /* error handling */
……
```

4.4. LRDS_DropIndex

Syntax

Four LRDS_DropIndex(Four handle, Four volId, char* relName, IndexID* iid)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	volId	Four	Volume identifier
IN	relName	char*	Name of relation
IN	iid	IndexID*	Index identifier

Description

Removes an index for the given relation. The index identifier specifies the index to be removed.

Return value

eNOERROR : The index has been successfully removed.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            e;
Four            volId;
Four                 orn;
lrds_RelTableEntry  *relTableEntry;
……

orn = LRDS_OpenRelation(handle, volId, “relation”);
if(orn < eNOERROR) /* error handling */

relTableEntry = LRDS_GET_RELTABLE_ENTRY(handle, orn);

e = LRDS_DropIndex(handle, volId, “relation”,
&(LRDS_GET_IDXINFO_FROM_RELTABLE_ENTRY(relTableEntry))[0].iid);
if(e < eNOERROR) /* error handling */
……
```

4.5. LRDS_AddColumn

Syntax

Four LRDS_AddColumn(Four handle, Four volId, char* relName, ColInfo* cinfo)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	volId	Four	Volume identifier

IN	relName	char*	Name of relation
IN	cinfo	ColInfo*	Information about the column to be added

Description

Adds a column to the relation.

Return value

eNOERROR : The column has been successfully added to the relation.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            e;
Four            volId;
ColInfo         cinfo;
……
cinfo.complexType = SM_COMPLEXTYPE_BASIC;
cinfo.type = SM_FLOAT;

e = LRDS_AddColumn(handle, volId, “relation”, &cinfo);
if(e < eNOERROR) /* error handling */
……
```

4.6. LRDS_OpenRelation

Syntax

Four LRDS_OpenRelation(Four handle, Four volId, char* relName)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	volId	Four	Volume identifier
IN	relName	char*	Name of relation

Description

Opens the given relation. Opening a relation means a task to register information about the relation to the open relation table.

Return value

Open relation number: The relation has been successfully opened.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            e;
```

```
Four                    volId;
Four            orn;
……

orn = LRDS_OpenRelation(handle, volId, “relation”);
if(orn < eNOERROR) /* error handling */
……
e = LRDS_CloseRelation(handle, orn);
if(e < eNOERROR) /* error handling */
……
```

4.7. LRDS_CloseRelation

Syntax

Four LRDS_CloseRelation(Four handle, Four orn)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	orn	Four	Open relation number

Description

Closes the relation indicated by the given open relation number.

Return value

eNOERROR : The relation has been successfully closed.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            e;
Four                  volId;
Four            orn;
……

orn = LRDS_OpenRelation(handle, volId, “relation”);
if(orn < eNOERROR) /* error handling */
……
e = LRDS_CloseRelation(handle, orn);
if(e < eNOERROR) /* error handling */
……
```

4.8. LRDS_CloseAllRelations

Syntax

Four LRDS_CloseAllRelations(Four handle)

Parameters

IN/OUT	Name	Type	Description

IN	handle	Four	Identifier for managing threads

Description

Closes all the relations opened by the current thread.

Return value

eNOERROR : All the relations have been successfully closed.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            e;
……

e = LRDS_CloseAllRelations(handle);
if(e < eNOERROR) /* error handling */
……
```

4.9. LRDS_SortRelation

Syntax

Four LRDS_SortRelation(Four handle, Four volId, Four tmpVolId, char* inRelName, KeyInfo* kinfo, Boolean newRelFlag, char* outRelName, Boolean tmpRelFlag, LockParameter* lockup)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	volId	Four	Identifier of the volume where the relation to be sorted is located
IN	tmpVolId	Four	Temporary volume to be used for sort
IN	inRelName	char*	Name of the relation to be sorted
IN	kinfo	KeyInfo*	Information about the sort key
IN	newRelFlag	Boolean	Flag whether or not to store the sorted result to the new relation
IN	outRelName	char*	Name of the relation to store the sorted result
IN	tmpRelFlag	Boolean	Flag whether the outRelName is a temporary relation or not
IN	lockup	LockParameter*	Information about the lock requested

Description

Sorts the given relation.

Return value

eNOERROR : The relation has been successfully sorted.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            e;
Four            volId;
Four                  tmpVolId;
KeyInfo        kinfo;
LockParameter   lockup;
……

kinfo.flag = KEYFLAG_CLEAR;
kinfo.nColumns = 1;
kinfo.columns[0].colNo = 3;
kinfo.columns[0].flag = KEYINFO_COL_DESC;

lockup.mode = L_X;
lockup.duration = L_COMMIT;

e = LRDS_SortRelation(handle, volId, tmpVolId, “in_relation”, &kinfo,
SM_TRUE, “out_relation”, SM_FALSE, &lockup);
if(e < eNOERROR) /* error handling */
……
```

4.10. LRDS_GetFileIdOfRelation

5. Scan Management

5.1. LRDS_OpenSeqScan

Syntax

Four LRDS_OpenSeqScan(Four handle, Four orn, Four scanDirection, Four nBools, BoolExp bool[], LockParameter* lockup)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	orn	Four	Open relation number
IN	scanDirection	Four	Scan direction (FORWARD or BACKWARD)
IN	nBools	Four	Number of boolean expressions
IN	bool[]	BoolExp	List of boolean expressions
IN	lockup	LockParameter*	Information about the requested lock

Description

Opens a sequential scan for the given relation. This scan is used to access tuples in order of being physically stored in the data file, without using an index. The parameter scanDirection is used to specify the direction of scan. If the value of scanDirection is FORWARD, it accesses in order of being stored, and if its value is BACKWARD, it accesses tuples in reverse order of being stored. A Boolean expression can be used to access not all tuples but only desired ones when accessing tuples. If a boolean expression is used, only the tuples, in which certain column value satisfies the boolean expression, are seen to users. A boolean expression has a form in which multiple conditions are combined by AND. The lockup parameter is used for hierarchical lock.

Return value

Scan identifier : The sequential scan has been successfully opened.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            e;
Four                  scanId;
Four                  orn;
BoolExp       bool[2];
LockParameter   lockup;
……

bool[0].op = SM_LE;
bool[0].colNo = 1;
bool[0].data.i = 10;
```

```
bool[1].op = SM_GE;
bool[1].colNo = 1;
bool[1].data.i = 20;

lockup.mode = L_X;
lockup.duration = L_COMMIT;

scanId = LRDS_OpenSeqScan(handle, orn, FORWARD, 2, bool, &lockup);
if(scanId < eNOERROR) /* error handling */
……
e = LRDS_CloseScan(handle, scanId);
if(e < eNOERROR) /* error handling */
……
```

5.2. LRDS_OpenIndexScan

Syntax

Four LRDS_OpenIndexScan(Four handle, Four orn, IndexID *iid, BoundCond* startBound, BoundCond* stopBound, Four nBools, BoolExp bool[], LockParameter* lockup)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	orn	Four	Open relation number
IN	iid	IndexID*	Identifier of the index used in the scan
IN	startBound	BoundCond*	Start boundary of range scan (NULL value is possible)
IN	stopBound	BoundCond*	Stop boundary of range scan (NULL value is possible)
IN	nBools	Four	Number of boolean expressions
IN	bool[]	BoolExp	List of boolean expressions
IN	lockup	LockParameter*	Information about the lock requested

Description

Opens an index scan for the given relation. An index scan accesses tuples in order of the size of the given index's key value. At this time, it is possible to access only the tuples being existed in the desired range by specifying the start and stop boundaries of the scanning range (range scan). If the start boundary is smaller than the stop boundary, it accesses tuples in the ascending order of key value, on the contrary if the start boundary is larger than the stop boundary, it accesses tuples in the descending order of key value. The start or stop boundary can have a NULL value, and in this case, there is no boundary, so it accesses up to the tuple having the smallest or largest key value. Index scan, like the sequential scan, can access only the tuples satisfying a boolean expression by giving the boolean expression.

The lockup parameter is used for hierarchical lock.

Return value

Scan identifier : The index scan has been successfully opened.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            e;
Four                 volId;
Four                 scanId;
Four                 orn;
BoundCond     startBound;
BoolExp       bool[1];
LockParameter   lockup;
Four                 keyValue;
lrds_RelTableEntry  *relTableEntry;
……
orn = LRDS_OpenRelation(handle, volId, “relation”);
if(orn < eNOERROR) /* error handling */

relTableEntry = LRDS_GET_RELTABLE_ENTRY(handle, orn);
……
startBound.op = SM_LT;
keyValue = 10;
startBound.key.len = sizeof(Four);
bcopy(&keyValue, &(startBound.key.val[0]), sizeof(Four));



bool[0].op = SM_GT;
bool[0].colNo = 1;
bool[0].data.i = 20;

lockup.mode = L_X;
lockup.duration = L_COMMIT;

scanId = LRDS_OpenSeqScan(handle, orn,
   &(LRDS_GET_IDXINFO_FROM_RELTABLE_ENTRY(relTableEntry))[0].iid,
   &startBound, NULL, 1, bool, &lockup);
if(scanId < eNOERROR) /* error handling */
……
e = LRDS_CloseScan(handle, scanId);
if(e < eNOERROR) /* error handling */
……
e = LRDS_CloseRelation(handle, orn);
if(e < eNOERROR) /* error handling */
……
```

5.3. LRDS_MLGF_OpenIndexScan

Syntax

Four LRDS_MLGF_OpenIndexScan(Four handle, Four orn, IndexID *iid, MLGF_HashValue lowerBounds[], MLGF_HashValue upperBounds[], Four nBools, BoolExp bool[], LockParameter* lockup)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	orn	Four	Open relation number
IN	iid	IndexID*	Index identifier used for scan
IN	lowerBounds[]	MLGF_HashValue	Minimum value for each key of the area where objects are to be found
IN	upperBounds[]	MLGF_HashValue	Maximum value for each key of the area where objects are to be found
IN	nBools	Four	Number of boolean expressions
IN	bool[]	BoolExp	List of boolean expressions
IN	lockup	LockParameter*	Information about the lock requested

Description

Opens the MLGF index scan for the given relations. Index scan, like the sequential scan, can access only the tuples which satisfy a boolean expression by giving the boolean expression. The lockup parameter is used for hierarchical lock.

Return value

Scan identifier : The index scan has been successfully opened.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            e;
Four                 volId;
Four                 scanId;
Four                 orn;
MLGF_HashValue  lowerBounds[2];
MLGF_HashValue  upperBounds[2];
BoolExp       bool[1];
LockParameter   lockup;
Four                 keyValue;
lrds_RelTableEntry  *relTableEntry;
……
orn = LRDS_OpenRelation(handle, volId, “relation”);
if(orn < eNOERROR) /* error handling */
```

```
relTableEntry = LRDS_GET_RELTABLE_ENTRY(handle, orn);
……
lowerBounds[0] = 1;
lowerBounds[1] = 1;

upperBounds[0] = 10;
upperBounds[1] = 10;

bool[0].op = SM_GT;
bool[0].colNo = 1;
bool[0].data.i = 20;

lockup.mode = L_X;
lockup.duration = L_COMMIT;

scanId = LRDS_OpenSeqScan(handle, orn,
   &(LRDS_GET_IDXINFO_FROM_RELTABLE_ENTRY(relTableEntry))[0].iid,
   lowerBounds, upperBounds, 1, bool, &lockup);
if(scanId < eNOERROR) /* error handling */
……
e = LRDS_CloseScan(handle, scanId);
if(e < eNOERROR) /* error handling */
……
e = LRDS_CloseRelation(handle, orn);
if(e < eNOERROR) /* error handling */
……
```

5.4. LRDS_MLGF_SearchNearTuple

Syntax

Four LRDS_MLGF_SearchNearTuple(Four handle, Four orn, IndexID *iid, MLGF_HashValue kval[], TupleID *tid, LockParameter* lockup)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	orn	Four	Open relation number
IN	iid	IndexID*	Index identifier used for scan
IN	kval[]	MLGF_HashValue	Find objects close to this key value
OUT	tid	TupleID*	Tuple identifier
IN	lockup	LockParameter*	Information about the lock requested

Description

Fins objects close to the given key in the MLGF index.

Return value

eNOERROR : It has been found an object close to the given key value.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            e;
Four                 volId;
Four                 scanId;
Four                 orn;
MLGF_HashValue  kval[2];
TupleID       tid;
LockParameter   lockup;
lrds_RelTableEntry  *relTableEntry;
……
orn = LRDS_OpenRelation(handle, volId, “relation”);
if(orn < eNOERROR) /* error handling */

relTableEntry = LRDS_GET_RELTABLE_ENTRY(handle, orn);
……
kval[0] = 5;
kval[1] = 1;

lockup.mode = L_X;
lockup.duration = L_COMMIT;

e = LRDS_MLGF_SearchNearTuple(handle, orn,
   &(LRDS_GET_IDXINFO_FROM_RELTABLE_ENTRY(relTableEntry))[0].iid,
   kval, &tid, &lockup);
if(e < eNOERROR) /* error handling */
……
e = LRDS_CloseRelation(handle, orn);
if(e < eNOERROR) /* error handling */
……
```

5.5. LRDS_CloseScan

Syntax

Four LRDS_CloseScan(Four handle, Four scanId)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	scanId	Four	Scan identifier

Description

Closes the scan. The scan to be closed is specified by a scan identifier.

Return value

eNOERROR : The scan has been successfully closed.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            e;
Four                 scanId;
Four                 orn;
BoolExp       bool[2];
LockParameter   lockup;
……

bool[0].op = SM_LE;
bool[0].colNo = 1;
bool[0].data.i = 10;

bool[1].op = SM_GE;
bool[1].colNo = 1;
bool[1].data.i = 20;

lockup.mode = L_X;
lockup.duration = L_COMMIT;

scanId = LRDS_OpenSeqScan(handle, orn, FORWARD, 2, bool, &lockup);
if(scanId < eNOERROR) /* error handling */
……
e = LRDS_CloseScan(handle, scanId);
if(e < eNOERROR) /* error handling */
……
```

5.6. LRDS_CloseAllScans

Syntax

Four LRDS_CloseAllScans(Four handle)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads

Description

Closes all the scans opened by the current thread.

Return value

eNOERROR : All the scans have been successfully closed.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            e;
……

e = LRDS_CloseAllScans(handle);
```

```
if(e < eNOERROR) /* error handling */
……
```

5.7. LRDS_NextTuple

Syntax

Four LRDS_NextTuple(Four handle, Four scanId, TupleID *tid, LRDS_Cursor** cursor)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	scanId	Four	Scan identifier
OUT	tid	TupleID*	Identifier of the next tuple
OUT	cursor	LRDS_Cursor**	Cursor of the scan

Description

Moves the scan cursor to the tuple to be scanned next and returns the tuple identifier of a new tuple. Users can deliver the returned tuple identifier directly to the parameters of LRDS_FetchTuple(), LRDS_UpdateTuple() and LRDS_DestroyTuple() functions to do the desired operation for the returned tuple.

Return value

eNOERROR : The scan cursor has been successfully moved.

EOS : The scan cursor points to the last tuple.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            e;
Four                  scanId;
Four                  orn;
BoolExp       bool[2];
LockParameter   lockup;
TupleID       tid;
LRDS_Cursor*    cursor;
……

bool[0].op = SM_LE;
bool[0].colNo = 1;
bool[0].data.i = 10;

bool[1].op = SM_GE;
bool[1].colNo = 1;
bool[1].data.i = 20;

lockup.mode = L_X;
```

```
lockup.duration = L_COMMIT;

scanId = LRDS_OpenSeqScan(handle, orn, FORWARD, 2, bool, &lockup);
if(scanId < eNOERROR) /* error handling */
……
e = LRDS_NextTuple(handle, scanId, &tid, &cursor);
if(e < eNOERROR) /* error handling */
……
e = LRDS_CloseScan(handle, scanId);
if(e < eNOERROR) /* error handling */
……
```

6. Tuple Management

6.1. LRDS_CreateTuple

Syntax

```
Four LRDS_CreateTuple(Four handle, Four ornOrScanId, Boolean useScanFlag,
Four nCols, ColListStruct *clist, TupleID *tid)
```

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	ornOrScanId	Four	Open relation number or scan identifier
IN	useScanFlag	Boolean	Flag to indicate whether or not to use the scan
IN	nCols	Four	Number of columns where data is to be stored when creating tuples (the type is Two in the coarse granularity locking version)
IN	clist	ColListStruct*	Initial column values of the tuple
OUT	tid	TupleID*	Tuple identifier of the created tuple

Description

Inserts a new tuple into the relation where scan is open. The column value of a new tuple is passed through the parameter clist. The number of entries in clist is passed through the parameter nCols. The column can have a NULL value, so there is no need for the content about all columns to be included in the clist. In addition, not only the SM_VARSTRING type columns, which allow variable length, but also SM_STRING type columns can store data partially. However, for these SM_STRING type columns, a space is prepared beforehand in case these values are fully given. It means simply data is not recorded at a time but can be divided several times to record. It should be noted that the LRDS does not have a memory of whether data is filled fully or partially, so users should memorize it.

Return value

eNOERROR : The tuple has been successfully created.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            e;
Four                 volId;
Four                 orn;
TupleID       tid;
ColListStruct   clist[2];
```

```
char            data[10]="abcdefghij"
Four                  value;
……
orn = LRDS_OpenRelation(handle, volId, "relation");
if(orn < eNOERROR) /* error handling */
……
clist[0].colNo = 0;
clist[0].nullFlag = SM_FALSE;
clist[0].start = ALL_VALUE;
clist[0].dataLength = sizeof(Four);
value = 5;
memcpy(&(clist[0].data), &value, sizeof(Four));

clist[1].colNo = 3;
clist[1].nullFlag = SM_FALSE;
clist[1].start = ALL_VALUE;
clist[1].dataLength = strlen(data);
clist[1].data.ptr = data;

e = LRDS_CreateTuple(handle, orn, SM_FALSE, 2, clist, &tid);
if(e < eNOERROR) /* error handling */
……
e = LRDS_CloseRelation(handle, orn);
if(e < eNOERROR) /* error handling */
……
```

6.2. LRDS_DestroyTuple

Syntax

Four LRDS_DestroyTuple(Four handle, Four ornOrScanId, Boolean useScanFlag, TupleID* tid)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	ornOrScanId	Four	Open relation number or scan identifier
IN	useScanFlag	Boolean	Flag to indicate whether or not to use the scan
IN	tid	TupleID*	Tuple identifier to be deleted

Description

Deletes a tuple in the relation. The tuple to be deleted is specified through the parameter tid. If the value of tid is NULL, the tuple pointed currently by the scan is deleted. When deleting a tuple, it deletes not only the tuples but also index entries corresponding to the of the deleted tuples for all indexs defined in the relation.

Return value

eNOERROR : The tuple has been successfully deleted

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            e;
Four                 scanId;
Four                 orn;
BoolExp       bool[1];
LockParameter   lockup;
……

bool[0].op = SM_EQ;
bool[0].colNo = 1;
bool[0].data.i = 10;

lockup.mode = L_X;
lockup.duration = L_COMMIT;

scanId = LRDS_OpenSeqScan(handle, orn, FORWARD, 1, bool, &lockup);
if(scanId < eNOERROR) /* error handling */
……
e = LRDS_DestroyTuple(handle, scanId, SM_TRUE, NULL);
if(e < eNOERROR) /* error handling */
……
e = LRDS_CloseScan(handle, scanId);
if(e < eNOERROR) /* error handling */
……
```

6.3. LRDS_UpdateTuple

Syntax

Four LRDS_UpdateTuple(Four handle, Four ornOrScanId, Boolean useScanFlag, TupleID *tid, Four nCols, ColListStruct *clist)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	ornOrScanId	Four	Open relation number or scan identifier
IN	useScanFlag	Boolean	Flag to indicate whether or not to use the scan
IN	tid	TupleID*	Tuple identifier to be updated
IN	nCols	Four	Number of the columns to be updated (the type is Two in the coarse granularity locking version)
IN	clist	ColListStruct*	Information about the column to be updated

Description

Updates several column values of the current or given tuple. If the value of parameter tid is NULL, the current tuple is updated, and if the value of parameter tid is not NULL, the given tuple is updated. Information about numbers of the columns to be updated and the update is passed through the parameter clist. The number of entries existed in the parameter clist is specified by the parameter nCols. Four kinds of information are needed for each column's update. First is the start position of the data to be updated, second is the amount of the existing data to be updated, third is the amount of new data to replace the existing data, and fourth is the content of new data. If the amount of the existing data is smaller than the amount of new data, it means that the existing data's content is updated and at the same time more data is inserted by the difference of two values. If the amount of the existing data is larger than the amount of new data, the amount of data is decreased by the difference of the values. Since only the SM_VARSTRING has a variable column value length, the column, in which two values can be different, is only the column declared as SM_VARSTRING type. For convenience, a few special values can be used as a parameter. If the start field value of the clist is ALL_VALUE, it means that all the existing data of the column is to be updated. And if the value of start is END, it means that the existing data is not updated but new data is appended. If the length field value of the clist is REMAINDER, it means that it is updated from the position specified by the start to the end of the column value.

Return value

eNOERROR : The tuple has been successfully updated.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"


Four                  scanId;
Four                  orn;
BoolExp        bool[1];
LockParameter   lockup;
TupleID        tid;
LRDS_Cursor*    cursor;
ColListStruct   clist[1];
char            data[10]="abcdefghij"
……

bool[0].op = SM_EQ;
bool[0].colNo = 1;
bool[0].data.i = 10;

lockup.mode = L_X;
lockup.duration = L_COMMIT;

scanId = LRDS_OpenSeqScan(handle, orn, FORWARD, 1, bool, &lockup);
if(scanId < eNOERROR) /* error handling */
```

```
……

clist[0].colNo = 3;
clist[0].nullFlag = SM_FALSE;
clist[0].start = ALL_VALUE;
clist[0].dataLength = strlen(data);
clist[0].data.ptr = data;

e = LRDS_UpdateTuple(handle, scanId, SM_TRUE, NULL, 1, clist);
if(e < eNOERROR) /* error handling */

……
e = LRDS_CloseScan(handle, scanId);
if(e < eNOERROR) /* error handling */

……
```

6.4. LRDS_FetchTuple

Syntax

Four LRDS_FetchTuple(Four handle, Four ornOrScanId, Boolean useScanFlag, TupleID *tid, Four nCols, ColListStruct clist[])

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	ornOrScanId	Four	Open relation number or scan identifier
IN	useScanFlag	Boolean	Flag to indicate whether or not to use the scan
IN	tid	TupleID*	Tuple identifier of the tuple to be fetched
IN	nCols	Four	Number of the columns to be fetched (the type is Two in the coarse granularity locking version)
INOUT	clist[]	ColListStruct	Information about the column to be fetched

Description

Fetches and returned values of the given columns from the current or given tuple. The columns to be fetched are delivered through the parameter clist, and values of the fetched data are also returned through the same parameter. The number of the columns existed in the parameter clist is delivered through the parameter nCols. Users should specify the range (start/end positions) to fetch for each column. Here, the position is a relative value to each column. The start position of the data to be fetched is recorded in the start field of the clist, and the amount of the data to be fetched is recorded in the dataLength field. If the start value is ALL_VALUE, all column values are fetched. And if the dataLength value is REMAINDER, data is fetched from the given start to the end of the column and returned it. If the type of the column

to be fetched is SM_STRING or SM_VARSTRING, users should secure a space for column values and pass its pointer through the clist. This function assumes that a space is secured as much as dataLength.

Return value

eNOERROR : The tuple has been successfully fetched.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"


Four                  scanId;
Four                  orn;
BoolExp        bool[1];
LockParameter   lockup;
TupleID        tid;
ColListStruct   clist[1];
char            data[100];
……

bool[0].op = SM_EQ;
bool[0].colNo = 1;
bool[0].data.i = 10;

lockup.mode = L_X;
lockup.duration = L_COMMIT;

scanId = LRDS_OpenSeqScan(handle, orn, FORWARD, 1, bool, &lockup);
if(scanId < eNOERROR) /* error handling */
……

clist[0].colNo = 3;
clist[0].nullFlag = SM_FALSE;
clist[0].start = ALL_VALUE;
clist[0].dataLength = strlen(data);
clist[0].data.ptr = data;

e = LRDS_FetchTuple(handle, scanId, SM_TRUE, NULL, 1, clist);
if(e < eNOERROR) /* error handling */

……
e = LRDS_CloseScan(handle, scanId);
if(e < eNOERROR) /* error handling */

……
```

6.5. LRDS_FetchColLength

Syntax

Four LRDS_FetchColLength(Four handle, Four ornOrScanId, Boolean useScanFlag, TupleID *tid, Four nCols, ColLengthInfoListStruct lengthInfoList[])

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	ornOrScanId	Four	Open relation number or scan identifier
IN	useScanFlag	Boolean	Flag to indicate whether or not to use the scan
IN	tid	TupleID*	Tuple identifier
IN	nCols	Four	Number of columns which length is wanted to know (the type is Two in the coarse granularity locking version)
INOUT	lengthInfoList[ ]	ColLengthInfoListStruct	Buffer for fetching the column's length information

Description

Gets the column's length.

Return value

eNOERROR : The column's length has been successfully obtained.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"


Four                    scanId;
Four                    orn;
BoolExp         bool[1];
LockParameter    lockup;
TupleID         tid;
ColLengthInfoListStruct lengthInfo [1]
……

bool[0].op = SM_EQ;
bool[0].colNo = 1;
bool[0].data.i = 10;

lockup.mode = L_X;
lockup.duration = L_COMMIT;

scanId = LRDS_OpenSeqScan(handle, orn, FORWARD, 1, bool, &lockup);
if(scanId < eNOERROR) /* error handling */
……

lengthInfo[0].colNo = 3;

e = LRDS_FetchColLength(handle, scanId, SM_TRUE, NULL, 1, lengthInfo);
if(e < eNOERROR) /* error handling */
```

```
……
e = LRDS_CloseScan(handle, scanId);
if(e < eNOERROR) /* error handling */

……
```

7. Counter Management

7.1. LRDS_CreateCounter

Syntax

Four LRDS_CreateCounter(Four handle, Four volId, char *cntrName, Four initialValue, CounterID *cntrId)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	volId	Four	Volume identifier
IN	cntrName	char*	Name of the counter to be created
IN	initialValue	Four	Initial value of the counter to be created
OUT	cntrId	CounterID*	Identifier of the created counter

Description

Creates a counter with the given name.

Return value

eNOERROR : The counter has been successfully created.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            e;
Four            volId;
CounterID     cntrId;
……

e = LRDS_CreateCounter(handle, volId, “testCounter”, 0, &cntrId);
if(e < eNOERROR) /* error handling */
……
```

7.2. LRDS_DestroyCounter

Syntax

Four LRDS_DestroyCounter(Four handle, Four volId, char *cntrName)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	volId	Four	Volume identifier
IN	cntrName	char*	Name of the counter to be created

Description

Deletes a counter with the given name.

Return value

eNOERROR : The counter has been successfully deleted.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            e;
Four            volId;
……

e = LRDS_DestroyCounter(handle, volId, “testCounter”);
if(e < eNOERROR) /* error handling */
……
```

7.3. LRDS_GetCounterId

Syntax

Four LRDS_GetCounterId(Four handle, Four volId, char *cntrName, CounterID *cntrId)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	volId	Four	Volume identifier
IN	cntrName	char*	Name of counter
OUT	cntrId	CounterID*	Counter identifier

Description

Gets the identifier of a counter with the given name.

Return value

eNOERROR : The counter identifier has been successfully obtained.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            e;
Four            volId;
CounterID     cntrId;
……

e = LRDS_GetCounterId(handle, volId, “testCounter”, &cntrId);
if(e < eNOERROR) /* error handling */
……
```

7.4. LRDS_SetCounter

Syntax

Four LRDS_SetCounter(Four handle, Four volId, CounterID *cntrId, Four value)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	volId	Four	Volume identifier
IN	cntrId	CounterID*	Identifier of counter
IN	value	Four	Value of the counter to be set

Description

Sets the value of counter to a new value.

Return value

eNOERROR : The value of counter has been successfully set.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            e;
Four            volId;
CounterID     cntrId;
……

e = LRDS_GetCounterId(handle, volId, “testCounter”, &cntrId);
if(e < eNOERROR) /* error handling */

e = LRDS_SetCounter(handle, volId, &cntrId, 4);
if(e < eNOERROR) /* error handling */

……
```

7.5. LRDS_ReadCounter

Syntax

Four LRDS_ReadCounter(Four handle, Four volId, CounterID *cntrId, Four* value)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	volId	Four	Volume identifier
IN	cntrId	CounterID*	Counter identifier

OUT	value	Four*	Value of counter

Description

Reads the value of counter.

Return value

eNOERROR : The value of counter has been successfully read.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            e;
Four            volId;
CounterID     cntrId;
Four                  value;
……

e = LRDS_GetCounterId(handle, volId, “testCounter”, &cntrId);
if(e < eNOERROR) /* error handling */

e = LRDS_ReadCounter(handle, volId, &cntrId, &value);
if(e < eNOERROR) /* error handling */

……
```

7.6. LRDS_GetCounterValues

Syntax

Four LRDS_GetCounterValues(Four handle, Four volId, CounterID *cntrId, Four nValues, Four *startValue)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	volId	Four	Volume identifier
IN	cntrId	CounterID*	Count identifier
IN	nValues	Four	Increases the counter value by nValues
OUT	startValue	Four*	Counter value before increasing

Description

Reads the value of counter and increases by nValues.

Return value

eNOERROR : The value of counter has been read and increased successfully.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            e;
Four            volId;
CounterID     cntrId;
Four                  startValue;
……

e = LRDS_GetCounterId(handle, volId, “testCounter”, &cntrId);
if(e < eNOERROR) /* error handling */

e = LRDS_GetCounterValues(handle, volId, &cntrId, 2, &startValue);
if(e < eNOERROR) /* error handling */

……
```

8. I/O Count Information

8.1. LRDS_ResetNumberOfDiskIO

Syntax

Four LRDS_ResetNumberOfDiskIO(Four handle)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads

Description

Initialize the variable for counting the number of disk I/O.

Return value

eNOERROR : The variable for counting the number of disk I/O has been successfully initialized.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            e;
Four            read;
Four            write;
……
e = LRDS_ResetNumberOfDiskIO(handle);
if(e < eNOERROR) /* error handling */
……
e = LRDS_GetNumberOfDiskIO(handle, &read, &write);
if(e < eNOERROR) /* error handling */
……
```

8.2. LRDS_GetNumberOfDiskIO

Syntax

Four LRDS_GetNumberOfDiskIO(Four handle, Four* read, Four* write)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
OUT	read	Four*	Number of disk reads
OUT	write	Four*	Number of disk writes

Description

Read the frequency of disk I/O.

Return value

eNOERROR : The number of disk I/O has been successfully read.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            e;
Four            read;
Four            write;
……
e = LRDS_ResetNumberOfDiskIO(handle);
if(e < eNOERROR) /* error handling */
……
e = LRDS_GetNumberOfDiskIO(handle, &read, &write);
if(e < eNOERROR) /* error handling */
……
```

9. BulkLoad

9.1. LRDS_InitRelationBulkLoad

Syntax

Four LRDS_InitRelationBulkLoad(Four handle, Four volId, Four tmpVolId, char* inRelName, Boolean isNeedSort, Boolean indexBlkLdFlag, Two pff, Two eff, LockParameter* lockup)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	volId	Four	Identifier of the data volume which stores the relation
IN	tmpVolId	Four	Identifier of the temporary volume which stores the sort stream created during bulkload
IN	inRelName	char*	Relation name
IN	isNeedSort	Boolean	Flag to specify whether or not to sort and bulkload for the clustering index key
IN	indexBlkLdFlag	Boolean	Flag to specify whether or not to use the bulkload routine when creating the index
IN	pff	Two	Page filling factor
IN	eff	Two	Extent filling factor
IN	lockup	LockParameter*	Concurrency control parameter

Description

Prepares (initializes) for the relation bulkload.

Return value

Bulkload identifier: The relation bulkload has been successfully initialized.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            e;
Four            volId;
Four                  tmpVolId;
Four                  blkLdId
LockParameter   lockup;
……

lockup.mode = L_X;
```

```
lockup.duration = L_COMMIT;

blkLdId = LRDS_InitRelationBulkLoad(handle, volId, tmpVolId, “test”,
SM_FALSE, SM_FALSE, 100, 100, &lockup);
if(e < blkLdId) /* error handling */
……
e = LRDS_FinalRelationBulkLoad (handle, blkLdId);
if(e < eNOERROR) /* error handling */
……
```

9.2. LRDS_FinalRelationBulkLoad

Syntax

Four LRDS_FinalRelationBulkLoad(Four handle, Four blkLdId)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	blkLdId	Four	Bulkload identifier

Description

Finalizes (terminates) the relation bulkload.

Return value

eNOERROR : The relation bulkload has been successfully terminated.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            e;
Four            volId;
Four                  tmpVolId;
Four                  blkLdId
LockParameter   lockup;
……

lockup.mode = L_X;
lockup.duration = L_COMMIT;

blkLdId = LRDS_InitRelationBulkLoad(handle, volId, tmpVolId, “test”,
SM_FALSE, SM_FALSE, 100, 100, &lockup);
if(e < blkLdId) /* error handling */
……
e = LRDS_FinalRelationBulkLoad (handle, blkLdId);
if(e < eNOERROR) /* error handling */
……
```

9.3. LRDS_NextRelationBulkLoad

Syntax

Four LRDS_NextRelationBulkLoad(Four handle, Four blkLdId, Four nCols, ColListStruct* clist, Boolean endOfTuple, TupleID* tid)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	blkLdId	Four	Bulkload identifier
IN	nCols	Four	Number of the columns to be inserted (the type is Two in the coarse granularity locking version)
IN	clist	ColListStruct*	Information about the columns to be inserted
IN	endOfTuple	Boolean	Flag to specify whether the inserted columns are the last column composing the tuple
OUT	tid	TupleID*	Tuple identifier of the created tuple

Description

Inserts the columns composing the tuple into the relation using the relation bulkload.

Return value

eNOERROR : The tuple has been successfully inserted.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            e;
Four                 volId;
Four                 tmpVolId;
Four                 blkLdId
LockParameter   lockup;
TupleID       tid;
ColListStruct   clist[2];
char            data[10]=”abcdefghij”
Four                 value;
……

lockup.mode = L_X;
lockup.duration = L_COMMIT;

blkLdId = LRDS_InitRelationBulkLoad(handle, volId, tmpVolId, “test”,
SM_FALSE, SM_FALSE, 100, 100, &lockup);
if(e < blkLdId) /* error handling */
……
```

```
clist[0].colNo = 0;
clist[0].nullFlag = SM_FALSE;
clist[0].start = ALL_VALUE;
clist[0].dataLength = sizeof(Four);
value = 5;
memcpy(&(clist[0].data), &value, sizeof(Four));

clist[1].colNo = 1;
clist[1].nullFlag = SM_FALSE;
clist[1].start = ALL_VALUE;
clist[1].dataLength = strlen(data);
clist[1].data.ptr = data;

……
e = LRDS_NextRelationBulkLoad(handle, blkLdId, 2, clist, SM_TRUE,
&tid);
if(e < eNOERROR) /* error handling */

……
e = LRDS_FinalRelationBulkLoad(handle, blkLdId);
if(e < eNOERROR) /* error handling */
……
```

10. Ordered Set

10.1. LRDS_OrderedSet_Create

Syntax

Four LRDS_OrderedSet_Create(Four handle, Four ornOrScanId, Boolean useScanFlag, TupleID* tid, Four colNo, LockParameter* lockupPtr)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	ornOrScanId	Four	Open relation number or scan identifier
IN	useScanFlag	Boolean	Flag to indicate whether or not to use the scan
IN	tid	TupleID*	Tuple identifier
IN	colNo	Four	Number of the column for which the ordered set is created (the type is Two in the coarse granularity locking version)
IN	lockupPtr	LockParameter*	Concurrency control parameter

Description

Creates an empty ordered set in the given column. The relation and tuple that has the given column are also given as input. If the tuple identifier is NULL, a set is created in the current tuple of the scan. If the set is already created in the column, an error is returned.

Return value

eNOERROR : The set has been successfully created.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            scanId;
LockParameter   lockup;
……

lockup.mode = L_X;
lockup.duration = L_COMMIT;

e = LRDS_OrderedSet_Create(handle, scanId, SM_TRUE, NULL, 3, &lockup);
if(e < eNOERROR) /* error handling */
……
```

10.2. LRDS_OrderedSet_Destroy

Syntax

Four LRDS_OrderedSet_Destroy(Four handle, Four ornOrScanId, Boolean useScanFlag, TupleID* tid, Four colNo, LockParameter* lockupPtr)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	ornOrScanId	Four	Open relation number or scan identifier
IN	useScanFlag	Boolean	Flag to indicate whether or not to use the scan
IN	tid	TupleID*	Tuple identifier
IN	colNo	Four	Number of the column for which the ordered set is created (the type is Two in the coarse granularity locking version)
IN	lockupPtr	LockParameter*	Concurrency control parameter

Description

Deletes the ordered set stored in the given column. The relation and tuple that has the given column are also given as input. If the set is deleted, the column has NULL value.

Return value

eNOERROR : The set has been successfully deleted.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            scanId;
LockParameter   lockup;
……

lockup.mode = L_X;
lockup.duration = L_COMMIT;

e = LRDS_OrderedSet_Destroy(handle, scanId, SM_TRUE, NULL, 3, &lockup);
if(e < eNOERROR) /* error handling */
……
```

10.3. LRDS_OrderedSet_CreateNestedIndex

Syntax

Four LRDS_OrderedSet_CreateNestedIndex(Four handle, Four ornFour colNo)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads

IN	orn	Four	Open relation number
IN	colNo	Four	Number of the column for which the ordered set is created (the type is Two in the coarse granularity locking version)

Description

When a relation and an ordered set column belonging to the relation are given, it sequentially scans the tuples, and if a set large enough to have a sub-index is found, converts them into a set with a sub-index. In addition, it automatically creates a sub-index if the size of a set becomes large enough to have a sub-index until the LRDS_OrderedSet_DestroyNestedIndex() function is called. The size large enough to have a sub-index means the size that the # disk accesses with a sub-index becomes smaller than the # disk accesses without it in the operation to find elements having the given key value.

Return value

eNOERROR : The set has been configured to have a sub-index.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            orn;
……
e = LRDS_OrderedSet_CreateNestedIndex(handle, orn, 3);
if(e < eNOERROR) /* error handling */
……
```

10.4. LRDS_OrderedSet_DestroyNestedIndex

Syntax

Four LRDS_OrderedSet_DestroyNestedIndex(Four handle, Four ornFour colNo)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	orn	Four	Open relation number
IN	colNo	Four	Number of the column having the ordered set (the type is Two in the coarse granularity locking version)

Description

When a relation and an ordered set column belonging to the relation are given, it sequentially scans the tuples, and if it finds a set with a sub-index, converts it into a set without a sub-index. Before the LRDS_OrderedSet_CreateNestedIndex() function is called, it does not create

a sub-index even if the size of each set becomes large enough.

Return value

eNOERROR : The set has been configured not to have a sub-index.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            orn;
……
e = LRDS_OrderedSet_DestroyNestedIndex(handle, orn, 3);
if(e < eNOERROR) /* error handling */
……
```

10.5. LRDS_OrderedSet_AppendSortedElements

Syntax

Four LRDS_OrderedSet_AppendSortedElements(Four handle, Four ornOrScanId, Boolean useScanFlag, TupleID* tid, Four colNo, Four nElements, Four elementsBufSize, char *elementsBuf, LockParameter* lockupPtr)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	ornOrScanId	Four	Open relation number or scan identifier
IN	useScanFlag	Boolean	Flag to indicate whether or not to use the scan
IN	tid	TupleID*	Tuple identifier
IN	colNo	Four	Number of the column having the ordered set (the type is Two in the coarse granularity locking version)
IN	nElements	Four	Number of elements to be inserted
IN	elementsBufSize	Four	Size of buffer having elements to be inserted
IN	elementsBuf	char*	Buffer having the elements to be inserted
IN	lockupPtr	LockParameter*	Concurrency control parameter

Description

Inserts elements to the end of the given set. Its performance is better than inserting elements one by one since it inserts several elements at a time. The elements to be inserted are given through the buffer called elementsBuf, which has an array form in which (data length, data) pairs are consecutively arranged. The elements to be inserted should be sorted in ascending order of the key value, and more than or equal to the key value of the element located

at the end of the current set.

Return value

eNOERROR : The elements have been successfully inserted into the set.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            scanId;
LockParameter   lockup;
char                 elementBuf[100];
OrderedSet_ElementLength elementLength;
char            data1[] = “abcd”;
char            data2[] = “efg”;
Four                 offset;
……

lockup.mode = L_X;
lockup.duration = L_COMMIT;

offset = 0;

elementLength = strlen(data1);
memcpy(&elementBuf[offset], &elementLength,
                            sizeof(OrderedSet_ElementLength));
offset += sizeof(OrderedSet_ElementLength);
memcpy(&elementBuf[offset], data1, elementLength);
offset += elementLength;

elementLength = strlen(data2);
memcpy(&elementBuf[offset], &elementLength,
                            sizeof(OrderedSet_ElementLength));
offset += sizeof(OrderedSet_ElementLength);
memcpy(&elementBuf[offset], data2, elementLength);
offset += elementLength;

e = LRDS_OrderedSet_AppendSortedElements(handle, scanId, SM_TRUE,
NULL, 3, 2, offset, elementBuf, &lockup);
if(e < eNOERROR) /* error handling */
……
```

10.6. LRDS_OrderedSet_InsertElement

Syntax

Four LRDS_OrderedSet_InsertElement(Four handle, Four ornOrScanId, Boolean useScanFlag, TupleID* tid, Four colNo, char *element, LockParameter* lockupPtr)

Parameters

IN/OUT	Name	Type	Description

IN	handle	Four	Identifier for managing threads
IN	ornOrScanId	Four	Open relation number or scan identifier
IN	useScanFlag	Boolean	Flag to indicate whether or not to use the scan
IN	tid	TupleID*	Tuple identifier
IN	colNo	Four	Number of the column having the ordered set (the type is Two in the coarse granularity locking version)
IN	element	char*	Buffer having the elements to be inserted
IN	lockupPtr	LockParameter*	Concurrency control parameter

Description

Inserts an element into an appropriate position of the given set according to the key value. This is a function provided for handling the case that should unavoidably insert an element which key value is bigger than the key value of the element located at the end of the set after elements were inserted into the set. If using this function, the performance becomes poorer comparing to using the LRDS_OrderedSet_AppendSortedElements() function. The elements to be inserted are given through the buffer called element, and it has an array form in which (data length, data) pairs are located consecutively.

Return value

eNOERROR : Elements have been successfully inserted into the set.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            scanId;
LockParameter   lockup;
char                  element[100];
OrderedSet_ElementLength elementLength;
char            data[] = “abcd”;
Four                  offset;
……

lockup.mode = L_X;
lockup.duration = L_COMMIT;

offset = 0;

elementLength = strlen(data);
memcpy(&element[offset], &elementLength,
                              sizeof(OrderedSet_ElementLength));
offset += sizeof(OrderedSet_ElementLength);
memcpy(&element[offset], data, elementLength);
offset += elementLength;
```

```
e = LRDS_OrderedSet_InsertElement(handle, scanId, SM_TRUE, NULL, 3,
element, &lockup);
if(e < eNOERROR) /* error handling */
……
```

10.7. LRDS_OrderedSet_DeleteElement

Syntax

Four LRDS_OrderedSet_DeleteElement(Four handle, Four ornOrScanId, Boolean useScanFlag, TupleID* tid, Four colNo, KeyValue *kval, LockParameter* lockupPtr)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	ornOrScanId	Four	Open relation number or scan identifier
IN	useScanFlag	Boolean	Flag to indicate whether or not to use the scan
IN	tid	TupleID*	Tuple identifier
IN	colNo	Four	Number of the column having the ordered set (the type is Two in the coarse granularity locking version)
IN	kval	KeyValue *	Key value of the element to be deleted
IN	lockupPtr	LockParameter*	Concurrency control parameter

Description

Deletes an element from the given set. The element to be deleted is given through kval, which is a key value of the element.

Return value

eNOERROR : The element has been successfully deleted from the set.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            scanId;
LockParameter   lockup;
KeyValue        kval;
Four                 key;
……

lockup.mode = L_X;
lockup.duration = L_COMMIT;

key = 10;
```

```
kval.len = sizeof(Four);
memcpy(&(kval.val[0]), &key, sizeof(Four));

e = LRDS_OrderedSet_DeleteElement(handle, scanId, SM_TRUE, NULL, 3,
&kval, &lockup);
if(e < eNOERROR) /* error handling */
……
```

10.8. LRDS_OrderedSet_DeleteElements

Syntax

Four LRDS_OrderedSet_DeleteElements(Four handle, Four ornOrScanId, Boolean useScanFlag, TupleID* tid, Four colNo, Four nElementsToDelete, KeyValue *kval, LockParameter* lockupPtr)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	ornOrScanId	Four	Open relation number or scan identifier
IN	useScanFlag	Boolean	Flag to indicate whether or not to use the scan
IN	tid	TupleID*	Tuple identifier
IN	colNo	Four	Number of the column having the ordered set (the type is Two in the coarse granularity locking version)
IN	nElementsToDelete	Four	Number of elements to be deleted
IN	kval	KeyValue *	Key value of elements to be deleted (array)
IN	lockupPtr	LockParameter*	Concurrency control parameter

Description

Deletes elements from the given set. The elements to be deleted are given through kval, which is an array of key values of elements.

Return value

eNOERROR : Elements have been successfully deleted from the set.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            scanId;
LockParameter   lockup;
KeyValue        kval[2];
Four                 key;
……
```

```
lockup.mode = L_X;
lockup.duration = L_COMMIT;

key = 10;
kval[0].len = sizeof(Four);
memcpy(&(kval[0].val[0]), &key, sizeof(Four));

key = 12;
kval[1].len = sizeof(Four);
memcpy(&(kval[1].val[0]), &key, sizeof(Four));

e = LRDS_OrderedSet_DeleteElements(handle, scanId, SM_TRUE, NULL, 3,
2, kval, &lockup);
if(e < eNOERROR) /* error handling */
……
```

10.9. LRDS_OrderedSet_UpdateElement

Syntax

Four LRDS_OrderedSet_UpdateElement(Four handle, Four ornOrScanId, Boolean useScanFlag, TupleID* tid, Four colNo, KeyValue *kval, Four updateStart, Four updateLength, Four updateDataLength, void* updateData, LockParameter* lockupPtr)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	ornOrScanId	Four	Open relation number or scan identifier
IN	useScanFlag	Boolean	Flag to indicate whether or not to use the scan
IN	tid	TupleID*	Tuple identifier
IN	colNo	Four	Number of the column having the ordered set (the type is Two in the coarse granularity locking version)
IN	kval	KeyValue *	Key value of the elements to be modified
IN	updateStart	Four	Start offset of the portion to be modified in the element
IN	updateLength	Four	Length of the portion to be modified
IN	updateDataLength	Four	Length of the data to replace the portion to be modified
IN	updateData	void*	Data to replace the portion to be modified
IN	lockupPtr	LockParameter*	Concurrency control parameter

Description

Modifies elements in the given set. The element to be modified is given through kval, which is the key value of element.

Return value

eNOERROR : Elements have been successfully deleted from the set.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            scanId;
LockParameter   lockup;
KeyValue        kval;
Four                 key;
char                 data[]="abc";
……

lockup.mode = L_X;
lockup.duration = L_COMMIT;

key = 10;
kval.len = sizeof(Four);
memcpy(&(kval.val[0]), &key, sizeof(Four));

e = LRDS_OrderedSet_UpdateElement(handle, scanId, SM_TRUE, NULL, 3,
&kval, 3, 4, strlen(data), (void*)data, &lockup);
if(e < eNOERROR) /* error handling */
……
```

10.10. LRDS_OrderedSet_Scan_Open

Syntax

Four LRDS_OrderedSet_Scan_Open(Four handle, Four ornOrScanId, Boolean useScanFlag, TupleID* tid, Four colNo, LockParameter* lockupPtr)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	ornOrScanId	Four	Open relation number or scan identifier
IN	useScanFlag	Boolean	Flag to indicate whether or not to use the scan
IN	tid	TupleID*	Tuple identifier
IN	colNo	Four	Number of the column having the ordered set (the type is Two in the coarse granularity locking version)
IN	lockupPtr	LockParameter*	Concurrency control parameter

Description

Opens a scan for the given set. Relations, tuples, and columns are given as input to specify the set to be scanned. For a certain set, scan is implemented such that there is only one scan at a point in time since, if there are multiple scans, there is an overhead of switching among scans everytime the target set for scan is changed.

Return value

Set scan identifier: The scan for the set has been successfully opened.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            scanId;
Four                  setScanId;
LockParameter   lockup;
……

lockup.mode = L_X;
lockup.duration = L_COMMIT;

setScanId = LRDS_OrderedSet_Scan_Open(handle, scanId, SM_TRUE, NULL,
3, FORWARD, &lockup);
if(setScanId < eNOERROR) /* error handling */
……

e = LRDS_OrderedSet_Scan_Close(handle, setScanId);
if(e < eNOERROR) /* error handling */
……
```

10.11. LRDS_OrderedSet_Scan_Close

Syntax

Four LRDS_OrderedSet_Scan_Close(Four handle, Four setScanId)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	setScanId	Four	Scan identifier for the set

Description

Closes the scan for the given set.

Return value

eNOERROR : The scan for the set has been successfully closed.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"
```

```
Four            handle;
Four            scanId;
Four                  setScanId;
LockParameter   lockup;
……

lockup.mode = L_X;
lockup.duration = L_COMMIT;

setScanId = LRDS_OrderedSet_Scan_Open(handle, scanId, SM_TRUE, NULL,
3, FORWARD, &lockup);
if(setScanId < eNOERROR) /* error handling */
……

e = LRDS_OrderedSet_Scan_Close(handle, setScanId);
if(e < eNOERROR) /* error handling */
……
```

10.12. LRDS_OrderedSet_Scan_NextElements

Syntax

Four LRDS_OrderedSet_Scan_NextElements(Four handle, Four setScanId, Four bufSize, char *elementBuf)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	setScanId	Four	Scan identifier for the set
IN	bufSize	Four	Size of the buffer used to return the fetched elements
OUT	elementBuf	char*	Buffer to return the fetched elements

Description

Fetches elements as many as the given buffer size from the current position in the given scan. The position of scan is increased by the fetched elements. The fetched elements are returned through the buffer, which has an array form in which (data length, data) pairs are consecutively arranged. The number of the elements actually fetched is returned as the return value of the function.

Return value

The number of the fetched elements: Elements have been successfully fetched.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            scanId;
```

```
Four                    setScanId;
LockParameter   lockup;
char elementSizeBuffer[256];
char elementBuffer[1024];
Four nElementsRead;
……

lockup.mode = L_X;
lockup.duration = L_COMMIT;

setScanId = LRDS_OrderedSet_Scan_Open(handle, scanId, SM_TRUE, NULL,
3, FORWARD, &lockup);
if(setScanId < eNOERROR) /* error handling */
……

nElementsRead = LRDS_OrderedSet_Scan_NextElements(handle,
           setScanId, 256, elementSizeBuffer, 1024, elementBuffer);
if(nElementsRead < eNOERROR) /* error handling */
……

e = LRDS_OrderedSet_Scan_Close(handle, setScanId);
if(e < eNOERROR) /* error handling */
……
```

10.13. LRDS_OrderedSet_Scan_SkipElementsUntilGivenKeyValue

Syntax

Four LRDS_OrderedSet_Scan_SkipElementsUntilGivenKeyValue(Four handle, Four setScanId, Four keyLength, char* keyValue)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	setScanId	Four	Scan identifier for the set
IN	keyLength	Four	Length of the key value (the type is Two in the coarse granularity locking version)
IN	keyValue	char*	Key value

Description

Moves the scan cursor to read the elements having a larger value than the given key.

Return value

eNOERROR : The scan cursor has been successfully moved.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
```



```
Four            scanId;
Four                  setScanId;
LockParameter   lockup;
Four                  keyValue;
……

lockup.mode = L_X;
lockup.duration = L_COMMIT;

setScanId = LRDS_OrderedSet_Scan_Open(handle, scanId, SM_TRUE, NULL,
3, FORWARD, &lockup);
if(setScanId < eNOERROR) /* error handling */
……

keyValue = 10;
e = LRDS_OrderedSet_Scan_SkipElementsUntilGivenKeyValue(handle,
                      setScanId, sizeof(Four), (char*) &keyValue);
if(e < eNOERROR) /* error handling */
……

e = LRDS_OrderedSet_Scan_Close(handle, setScanId);
if(e < eNOERROR) /* error handling */
……
```

10.14. LRDS_OrderedSet_GetTotalLengthOfElements

Syntax

Four LRDS_OrderedSet_GetTotalLengthOfElements(Four handle, Four ornOrScanId, Boolean useScanFlag, TupleID* tid, Four colNo, Four* totalLength, LockParameter* lockupPtr)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	ornOrScanId	Four	Open relation number or scan identifier
IN	useScanFlag	Boolean	Flag to indicate whether or not to use the scan
IN	tid	TupleID*	Tuple identifier
IN	colNo	Four	Number of the column having the ordered set (the type is Two in the coarse granularity locking version)
OUT	totalLength	Four*	Total length of elements
IN	lockupPtr	LockParameter*	Concurrency control parameter

Description

Gets total length of the elements stored in the ordered set.

Return value

eNOERROR : The total length of elements has been successfully obtained.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            scanId;
Four                  totalLength;
LockParameter   lockup;
……

lockup.mode = L_X;
lockup.duration = L_COMMIT;

e = LRDS_OrderedSet_GetTotalLengthOfElements(handle, scanId, SM_TRUE,
NULL, 3, &totalLength, &lockup);
if(e < eNOERROR) /* error handling */
……
```

10.15. LRDS_OrderedSet_GetN_Elements

Syntax

Four LRDS_OrderedSet_GetN_Elements(Four handle, Four ornOrScanId, Boolean useScanFlag, TupleID* tid, Four colNo, Four* nElements, LockParameter* lockupPtr)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	ornOrScanId	Four	Open relation number or scan identifier
IN	useScanFlag	Boolean	Flag to indicate whether or not to use the scan
IN	tid	TupleID*	Tuple identifier
IN	colNo	Four	Number of the column having the ordered set (the type is Two in the coarse granularity locking version)
OUT	nElements	Four*	Number of elements
IN	lockupPtr	LockParameter*	Concurrency control parameter

Description

Gets the number of elements stored in the ordered set.

Return value

eNOERROR : The number of elements has been successfully obtained.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            scanId;
Four                  nElements;
LockParameter   lockup;
……

lockup.mode = L_X;
lockup.duration = L_COMMIT;

e = LRDS_OrderedSet_GetN_Elements(handle, scanId, SM_TRUE, NULL, 3,
&nElements, &lockup);
if(e < eNOERROR) /* error handling */
……
```

10.16. LRDS_OrderedSet_IsMember

Syntax

Four LRDS_OrderedSet_IsMember(Four handle, Four ornOrScanId, Boolean useScanFlag, TupleID* tid, Four colNo, KeyValue *kval, Four bufSize, char *elementBuf, LockParameter* lockupPtr)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	ornOrScanId	Four	Open relation number or scan identifier
IN	useScanFlag	Boolean	Flat to indicate whether or not to use the scan
IN	tid	TupleID*	Tuple identifier
IN	colNo	Four	Number of the column having the ordered set (the type is Two in the coarse granularity locking version)
IN	kval	KeyValue*	Key value of the element to be found
IN	bufSize	Four	Size of the buffer to return a value of the element to be found
OUT	elementBuf	char*	Buffer to return a value of the element to be found
IN	lockupPtr	LockParameter*	Concurrency control parameter

Description

Checks whether an element with the given key value belongs to the ordered set. If there is an element with the key value, it returns a (data length, data) pair via the buffer. If the (data length, data) pair is larger than the given buffer size, it fetches only as many bytes as the smaller number

of the buffer size and the ORDEREDSET_ELEMENT_FETCH_CHUNK_SIZE.

Return value

1 : There are elements.

0 : There is no element.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            scanId;
Four                 key;
KeyValue       kval;
Char                 buf[256];
LockParameter   lockup;
……

lockup.mode = L_X;
lockup.duration = L_COMMIT;

key = 10;
kval.len = sizeof(Four);
memcpy(&(kval.val[0]), &key, sizeof(Four));

e = LRDS_OrderedSet_IsMember(handle, scanId, SM_TRUE, NULL, 3, &kval,
256, buf, &lockup);
if(e < eNOERROR) /* error handling */
……
```

10.17. LRDS_OrderedSet_HasNestedIndex

Syntax

Four LRDS_OrderedSet_HasNestedIndex(Four handle, Four ornOrScanId, Boolean useScanFlag, TupleID* tid, Four colNo, LockParameter* lockupPtr)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	ornOrScanId	Four	Open relation number or scan identifier
IN	useScanFlag	Boolean	Flag to indicate whether or not to use the scan
IN	tid	TupleID*	Tuple identifier
IN	colNo	Four	Number of the column having the ordered set (the type is Two in the coarse granularity locking version)
IN	lockupPtr	LockParameter*	Concurrency control parameter

Description

Checks whether there is a sub-index in the ordered set.

Return value

SM_TRUE : There is a sub-index.

SM_FALSE : There is no sub-index.

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            scanId;
Four            isSubIndexExist;
LockParameter   lockup;
……

lockup.mode = L_X;
lockup.duration = L_COMMIT;

isSubIndexExist  =  LRDS_OrderedSet_HasNestedIndex(handle,  scanId,
SM_TRUE, NULL, 3, &lockup);
if(isSubIndexExist < eNOERROR) /* error handling */
……
```

10.18. LRDS_OrderedSet_IsNull

Syntax

Four LRDS_OrderedSet_IsNull(Four handle, Four ornOrScanId, Boolean useScanFlag, TupleID* tid, Four colNo)

Parameters

IN/OUT	Name	Type	Description
IN	handle	Four	Identifier for managing threads
IN	ornOrScanId	Four	Open relation number or scan identifier
IN	useScanFlag	Boolean	Flag to indicate whether or not to use the scan
IN	tid	TupleID*	Tuple identifier
IN	colNo	Four	Number of the column having the ordered set (the type is Two in the coarse granularity locking version)

Description

Determines whether the ordered set is NULL or not.

Return value

SM_TRUE : The set is NULL

SM_FALSE : The set is not NULL

< eNOERROR : Error code

Example

```
#include "cosmos_r.h"

Four            handle;
Four            scanId;
Four            isNull;
……

isNull = LRDS_OrderedSet_IsNull(handle, scanId, SM_TRUE, NULL, 3);
if(isNull < eNOERROR) /* error handling */
……
```

10.19. LRDS_OrderedSet_SpecifyKeyOfElement

10.20. LRDS_OrderedSet_SpecifyVolNo

10.21. LRDS_OrderedSet_GetVolNo

11. Text

11.1. LRDS_Text_AddKeywords

11.2. LRDS_Text_DeleteKeywords

11.3. LRDS_Text_GetIndexID

12. SET

13. CollectionSet

14. CollectionBag

15. CollectionList

16. Error

16.1. LRDS_Err