Legion 1.6.3

Grid Library Manual

The Legion Group

Department of Computer Science
School of Engineering & Applied Science
University of Virginia
151 Engineer's Way, P.O. Box 400740
Charlottesville, VA 22904-4740
legion@virginia.edu
http://legion.virginia.edu/

This work partially supported by DARPA (Navy) contract # N66001-96-C-8527, DOE grant DE-FD02-96ER25290, DOE contract Sandia LD-9391, Northrup-Grumman (for the DoD HPCMOD/PET program), DOE D459000-16-3C and DARPA (GA) SC H607305A

Legion Grid library

Table of contents

1.0 Grid library

The CLegionLib.h file contains the declarations for the C Library interface to the Legion system. This library is specifically designed to allow ease of use in many languages (including Fortran, C, C++). For additional functionality, the CError.h header file contains all necessary declarations to doing some small amount of error handling.

```
/* Change Log:    */
#ifndef CLEGION_LIB_H
#define CLEGION_LIB_H

#ifdef __cplusplus
extern "C" {
#endif

#include <stdio.h>
```

Note that anytime this header file talks about "free"ing memory, it is referring to using the C-language's `free()` function to return memory to the heap, NOT the C++ language's delete operators.

```
/* LOID Parameters */
#define LOID_PARAM      0                   /* These defines give the    */
#define CONTEXT_PARAM   1                   /* two valid values for      */
                                            /* LegionIdType variables    */
                                            /* parameters.               */

typedef  int LegionIdType;                  /* This type is used in      */
                                            /* function calls to         */
                                            /* indicate whether or not   */
                                            /* the parameter is a LOID.  */
                                            /* The valid values are      */
                                            /* LOID_PARAM and            */
                                            /* CONTEXT_PARAM.            */

typedef  int CLegionBinding;                /* These typedefs are used   */
typedef  int CLegionHostReservation;        /* to represent all of the   */
typedef  int CLegionHostReservationRecord;  /* necessary C++ classes     */
typedef  int CLegionVaultReservation;       /* that would normally be    */
typedef  int CLegionVaultReservationRecord;/* used in their place. In    */
typedef  int CLegionReservation;            /* all cases these are       */
typedef  int CLegionHostObjectStatus;       /* actually pointer to LRefs */
typedef  int CLegionOPRAddress;             /* to the actual class type. */
typedef  int CUVaL_InstanceRecord;          /* type. This double level   */
typedef  int CInstancePlacementInfo;        /* of pointer indirection    */
typedef  int  CLegionCollectionData;        /* was done to allow for     */
                                            /* some small measure of     */
                                            /* reference counting.       */
```

The following section contains all of the valid Legion architectures.

```
#define LEGION_ARCH_linux            1
#define LEGION_ARCH_solaris          2
#define LEGION_ARCH_sun4             3
#define LEGION_ARCH_alpha_linux      4
#define LEGION_ARCH_alpha_DEC        5
#define LEGION_ARCH_sgi              6
#define LEGION_ARCH_rs6000           7
#define LEGION_ARCH_hp               8
#define LEGION_ARCH_t90              9
#define LEGION_ARCH_c90              10
#define LEGION_ARCH_t3e              11
#define LEGION_ARCH_x86_freebsd      12
#define LEGION_ARCH_winnt_x86        13
#define LEGION_ARCH_hppa_hpux        14
```

All programs that use the CLegionLib library contain some small amount of state which determines how the CLegionLib library interacts with certain Legion functions. The following section gives the declarations for the functions that allow the user to interface with this state.

```
void   SetTimeOutValue     (int Seconds); /* These functions set and */
void   ClearTimeOutValue   ();            /* clear the default       */
                                          /* timeout value for most  */
                                          /* Legion method           */
                                          /* invocations.            */

void   SetForceActivation  ();            /* In some small number of */
void   ClearForceActivation();            /* calls, a target Legion  */
                                          /* object may not yet be   */
                                          /* active. The force       */
                                          /* activation flag         */
                                          /* indicates whether the   */
                                          /* call should activate    */
                                          /* the object or fail.     */
```

1.1 Legion Library interface

The following section lists the functions which interface with the C++ Legion Library.

int
Legion_init();

Initializes the legion library (for a Legion object started by a Legion host, not for a command line object). This function, or one of the other init functions must be called in all Legion programs before any other Legion calls are made. This does not preclude calls to the CLegionLib which do not directly interact with Legion.

Parameters: None

Return Values:
1 on Success
0 on Failure

User Responsibility: None

int
Legion_init_with_class(*char *ClassID, int ClassIdLen*);

Initializes the Legion library (for a Legion object started as an instance of a certain Legion class, but perhaps started on the user's command line). This function, or one of the other init functions must be called in all Legion programs before any other Legion calls are made. This does not preclude calls to the CLegionLib which do not directly interact with Legion.

Parameters:
char* ClassID = The representation of the class which instantiated the given object. These are not strings so much as byte arrays.
int ClassIdLen = Length of the "byte" array given for ClassID.

Return Values:
1 on Success
0 on Failure

User Responsibility: None

int
Legion_init_command_line_class();

Initializes the legion library (for a Legion object started on the command line as a main Legion program). For most client tools, this version of create will be the one used. This function, or one of the other init functions must be called in all Legion programs before any other Legion calls are made. This does not preclude calls to the CLegionLib which do not directly interact with Legion.

Parameters: None

Return Values:
1 on Success
0 on Failure

User Responsibility: None

char*
Legion_GetLegionClassLOI ();

Obtains the string representation of the LOID of the LegionClass object.

Parameters: None

Return Values:
LOID of the LegionClass object.

Error Return:
NULL

User Responsibility:
User is responsible for "free"ing memory allocated for the LOID

CLegionBinding
Legion_GetLegionClassBinding();

Obtains the current binding for the LegionClass object.

Parameters: None

Return Values:
LegionBinding for LegionClass object

Error Return:
0

User Responsibility:
User is responsible for calling DestroyLegionBinding on the returned binding when done with it.

int
Legion_AcceptMethods();

Enables the receipt of messages in the Legion library. This function must be called prior to any other function which might make outcalls to other objects, or receive methods or return values from any other objects.

Parameters: None

Return Values:
1 on Success
0 on Failure

User Responsibility: None

int
Legion_DeleteSelf();

Indicates to the object's (program's) class that the program is finished and ready to go away. After this function is called, the user's code may run for some undetermined amount of time, but will eventually be killed if it doesn't exit.

Parameters: None

Return Values:
1 on Success
0 on Failure

User Responsibility: None

void
Legion_Sleep(*int secs*);

Causes the user's code to pause for approximately. the number of seconds requested, while still handling external Legion events (such as method receives, exception receives, etc.).

Parameters:
secs = Number of seconds to sleep for.

Return Values: None

User Responsibility: None

char*
ClassOf(*LegionIdType IdType, char *ObjName*);
Returns the LOID of the indicated object's class.

Parameters:
IdType = Either CONTEXT_PARAM or LOID_PARAM indicating what type of string name is given next.
ObjName = Either a LOID, or a context path to an object

Return Values:
LOID of the indicated object's class.

Error Return:
NULL

User Responsibility:
The user is responsible for "free"ing the LOID return value.

char*
GetMyLOID();
Returns the LOID of the running object(program).

Parameters: None

Return Values:
LOID of the current running program.

Error Return:
NULL

User Responsibility:
The user is responsible for "free"ing the LOID return value.

1.2 Exceptions

The following section describes functions which interface with the Legion Exception interface.

```
extern int CLEGION_CONTINUE_ON_EXCEPTION;      /* These variables   */
extern int CLEGION_EXIT_ON_EXCEPTION;          /* correspond with   */
extern int CLEGION_CANCEL_METHODS_ON_EXCEPTION;/* the valid values  */
extern int CLEGION_NO_MESSAGE_ON_EXCEPTION;    /* that can be used  */
                                               /* as flags when     */
                                               /* enabling an       */
                                               /* exception catcher.*/
```

void
Legion_ExceptionCatcherDefaultEnable(*int Flag*);

Enables a default Legion Exception Handler with user indicated properties.

Parameters:

Flag = A value from the valid flags listed above which indicates what the handler should do when a Legion exception is caught. Valid values include:

CLEGION_CONTINUE_ON_EXCEPTION
CLEGION_EXIT_ON_EXCEPTION
CLEGION_CANCEL_METHODS_ON_EXCEPTION
CLEGION_NO_MESSAGE_ON_EXCEPTION
(Don't do anything)

Return Values: None

User Responsibility: None

1.3 Context space

The following section provides a C interface to Legion Context Space.

int
ContextSpaceActive();

Determines whether or not a valid context space exists.

Parameters: None

Return Values:

0 if Context space is NOT active
1 if Context space is active

User Responsibility: None

ContextLookup(*char *LookupPath*);

Look up the LOID of a name in the current context (does not allow paths in the name).

Parameters:

Context name of an object to lookup in the current context.

Return Values:

LOID of the object indicated context space.

Error Return:

NULL

User Responsibility:

The user is responsible for "free"ing the LOID return value.

char*
ContextPathLookup(*char *LookupPath*);

Look up the LOID of the object named in the given "Unix-like" context path. This command will allow both absolute, and relative naming.

Parameters:

context path name of an object to lookup in context space.

Return Values:

LOID of the object indicated context space.

Error Return:

NULL

User Responsibility:

The user is responsible for "free"ing the LOID return value.

char**
ContextMultiLookup(*char *LkupPath*);

Look up the LOIDs of a number of objects named in the current context (not path specified, however). It's somewhat confusing as to why this command takes a string at all, but the string basically needs to contain "*" as its parameter.

Parameters:

Unclear, but give it "*"

Return Values:

NULL terminated array of context space names for objects in the indicated context. In other words, each element of the array is a `char*`, which is either the context name of an object, or NULL if it is the end of the list.

Error Return:
NULL

User Responsibility:
The user is responsible for "free"ing each non-NULL entry in the array, as well as the array itself.

char**
ContextPathMultiLookup(*char *LkupPath*);
Look up the LOIDs of a number of objects named in a "Unix-like" context path (absolute or relative path names are perfectly valid here). It's somewhat confusing as to what the parameter to this command means. Basically, the thing to do here is to indicate the context path you want a listing for, and append "/" and "*" to it.

Parameters:
Unclear, but basically the path you want a listing of with "`/ *`" appended to it. I.e., if you want a listing of `/home/mark`, the parameter should be "`/home/mark/ *`".

(NOTE, in the above paragraph, there are spaces between / and * that should be removed. These are there to keep the C compiler from thinking we are commenting.)

Return Values:
NULL terminated array of context space names for objects in the indicated context. In other words, each element of the array is a `char*`, which is either the context name of an object, or NULL if it is the end of the list.

Error Return:
NULL

User Responsibility:
The user is responsible for "free"ing each non-NULL entry in the array, as well as the array itself.

int
ContextAdd(*char *Name, char *Loid*);
Add an entry to the current context. This new entry will have the indicated name, and will point to the object with the given LOID. This command assumes that the name is in the current context (not a full path).

Parameters:

Name =	The name to add to the current context
Loid =	The LOID of the object to link into context space.

Return Values:
1 on Success
0 on Failure

User Responsibility: None

int
ContextRemove(*char *Name*);
Remove an entry from the current context. This entry must be in the current context. It cannot be a path name, but only a current context entry.

Parameters:
Name = The name to remove from context space.

Return Values:
1 on Success
0 on Failure

User Responsibility: None

int
ContextPathAdd(*char *Name, char *Loid*);
Add an entry to context space. This new entry will have the indicated name, and will point to the object with the given LOID. This command allows relative and absolute paths in context space to be added.

Parameters:
Name = The name to path to add to context space
LOID = The LOID of the object to link into context space.

Return Values:
1 on Success
0 on Failure

User Responsibility: None

int
ContextPathRemove(*char *Name*);
Remove a path from context space.

Parameters:
Name = The path to remove from context space.

Return Values:
1 on Success
0 on Failure

User Responsibility: None

char*
ContextSingleReverseLookup(*LegionIdType IdType,*
*char *ObjName*);

Try and obtain a context path name for a given object. This command uses the object's attributes to determine this name and only returns the first name found. Note that the attribute database may or may not actually reflect the object's actual context space names

Parameters:

IdType =	Indicates whether the ObjName gives a context path name, or a LOID.
ObjName =	Either the context name, or the LOID of the object to reverse lookup.

Return Values:
First context path name of the object looked up.

Error Return:
NULL

User Responsibility:
The user is responsible for "free"ing the name returned.

char**
ContextMultiReverseLookup(*LegionIdType, char *ObjName,*
int MaxNum);

Try and obtain a list of context path names for a givenobject. This command uses the object's attributes to determine these names and only returns the first "MaxNum" entries found. Note that the attribute database may or may not actually reflect the object's actual context space names.

Parameters:

IdType =	Indicates whether the ObjName gives a context path name, or a LOID.
ObjName =	Either the context name, or the LOID of the object to reverse lookup.
MaxNum =	Maximum number of entries to return.

Return Values:
First "MaxNum" context path names of the object. These paths are stored in a NULL terminated array of strings.

Error Return:
NULL

User Responsibility:
The user is responsible for "free"ing each name returned in the list well as the list itself.

char*
ContextPWD();
Return the context path name of the current context.

Parameters: None

Return Values:
context path name of the current context.

Error Return:
NULL

User Responsibility:
The user is responsible for "free"ing the context path name returned.

char*
legion_mkdir(*char *PathName*);
Creates a new context named by the given path.

Parameters:
Pathname of the context to create.

Return Values:
LOID of the new context that was created.

Error Return:
NULL

User Responsibility:
The user is responsible for "free"ing the context LOID.

int
legion_cd(*char *PathName*);
Change the current context to the one named in the path. NOTE that this only changes the current context for the running program, not for the shell or program that executed the program.

Parameters:
Pathname of the context to change to.

Return Values:
1 on Success
0 on Failure

User Responsibility: None

1.4 Object management

The following section contains functions which are used to manage actual Legion Objects (create, activate, destroy, etc.).

int
ObjectActive(*LegionIdType, char *Name*);

Indicates whether the given object is currently active or inert.

Parameters:

IdType =	Flag indicating whether parameter is a LOID or a context path name.
Name =	Either a context path name or a LOID.

Return Values:

1 if the object is active
0 if the object is inert

Error Return:

-1

User Responsibility: None

char*
Ping(*LegionIdType, char *Name*);

Ping a given object to see if it is responsive.

Parameters:

IdType =	Flag indicating whether parameter is a LOID or a context path name.
Name =	Either a context path name or a LOID.

Return Values:

LOID of the object pinged.

Error Return:

NULL

User Responsibility:

User is responsible for "free"ing the LOID returned.

char**
Create(*int NumObjects, LegionIdType, char* ClassName*);

Create a vector of new Legion objects by vector instantiating the given class.

Parameters:

NumObjects =	Number of object to vector instantiate.
IdType =	Flag indicating whether the ClassName parameter is a context path name or a LOID.

ClassName = LOID or context path name of the class to instantiate.

Return Values:
NULL terminated array of LOIDs indicating the objects that were instantiated. Each element of the array is a `char*` which is the LOID of one object created.

Error Return:
NULL

User Responsibility:
User is responsible for "free"ing each LOID in the array as well as the array itself.

char**
ScheduledCreate(*int NumObjects, LegionIdType CIDType, char* ClassName, LegionIdType SIDType, char *SchedName*);

Create a vector of new Legion objects by vector instantiating the given class. Used the given scheduler to schedule this vector instantiation.

Parameters:
NumObjects = Number of object to vector instantiate.
CIdType = Flag indicating whether the ClassName parameter is a context path name or a LOID.
ClassName = LOID or context path name of the class to instantiate.
SIdType = Flag indicating whether the SchedName parameter is a context path name or a LOID.
ClassName = LOID or context path name of the Scheduler to use when instantiating the class.

Return Values:
NULL terminated array of LOIDs indicating the objects that were instantiated. Each element of the array is a char* which is the LOID of one object created.

Error Return:
NULL

User Responsibility:
User is responsible for "free"ing each LOID in the array as well as the array itself.

char**
CreateRestricted(*int NumObjects,*
*LegionIdType CIDType, char *ClassName,*
*LegionIdType HIDType, char *HostName,*
*LegionIdType VIDType, char *VaultName,*
*LegionIdType IIDType, char *ImplName)*;

Create a vector of new Legion objects by vector instantiating the given class. The user also indicates any combination of restrictions which may include restrictions to hosts, vaults or implementations.

Parameters:

NumObjects =	Number of object to vector instantiate.
CIdType =	Flag indicating whether the ClassName parameter is a context path name or a LOID.
ClassName =	LOID or context path name of the class to instantiate.
HIdType =	Flag indicating whether the HostName parameter is a context path name or a LOID.
HostName =	LOID or context path name of the Host to restrict the instantiation to.
VIdType =	Flag indicating whether the HostName parameter is a context path name or a LOID.
VaultName =	LOID or context path name of the Vault to which the instantiation is restricted.
IIdType =	Flag indicating whether the HostName parameter is a context path name or a LOID.
ImplName =	LOID or context path name of the implementation to which the instantiation is restricted.

Return Values:

NULL terminated array of LOIDs indicating the objects that were instantiated. Each element of the array is a char* which is the LOID of one object created.

Error Return:

NULL

User Responsibility:

User is responsible for "free"ing each LOID in the array as well as the array itself.

char**
ScheduledCreateRestricted(*int NumObjects,*
*LegionIdType CIDType, char *ClassName,*
*LegionIdType HIDType, char *HostName,*
*LegionIdType VIDType, char *VaultName,*
*LegionIdType IIDType, char *ImplName,*
*LegionIdType SIDType, char *SchedName*);

Create a vector of new Legion objects by vector instantiating the given class. The user also indicates any combination of restrictions which may include restrictions to hosts, vaults or implementations. This command also uses the given scheduler to schedule the instantiation.

Parameters:

NumObjects =	Number of object to vector instantiate.
CIdType =	Flag indicating whether the ClassName parameter is a context path name or a LOID.
ClassName =	LOID or context path name of the class to instantiate.
HIdType =	Flag indicating whether the HostName parameter is a context path name or a LOID.
HostName =	LOID or context path name of the Host to restrict the instantiation to.
VIdType =	Flag indicating whether the HostName parameter is a context path name or a LOID.
VaultName =	LOID or context path name of the Vault to which the instantiation is restricted.
IIdType =	Flag indicating whether the HostName parameter is a context path name or a LOID.
ImplName =	LOID or context path name of the Implementation which the instantiation is restricted.
SIdType =	Flag indicating whether the SchedName parameter is a context path name or a LOID.
ImplName =	LOID or context path name of the Scheduler to use when scheduling the instantiation.

Return Values:

NULL terminated array of LOIDs indicating the objects that were instantiated. Each element of the array is a char* which is the LOID of one object created.

Error Return:

NULL

User Responsibility:

User is responsible for "free"ing each LOID in the array as well as the array itself.

char**
ComplxCreateRestricted(*int NumObjects,*
*LegionIdType CIDType, char *ClassName,*
*LegionIdType HIDType, char **HostName,*
*LegionIdType VIDType, char **VaultName,*
*LegionIdType IIDType, char **ImplName*);

Create a vector of new Legion objects by vector instantiating the given class. The user also indicates any combination of restrictions which may include restrictions to hosts, vaults or implementations.

Parameters:

NumObjects =	Number of object to vector instantiate.
CIdType =	Flag indicating whether the ClassName parameter is a context path name or a LOID.
ClassName =	LOID or context path name of the class to instantiate.
HIdType =	Flag indicating whether the HostName parameter is a NULL terminated list of context path names or a NULL terminated list of LOIDs.
HostName =	NULL terminated array of LOIDs or context path names of the Hosts to which the instantiation is limited.
VIdType =	Flag indicating whether the VaultName parameter is a NULL terminated list of context path names, or a NULL terminated list of LOIDs.
VaultName =	NULL terminated array of LOIDs or context path names of the Vaults to which the instantiation is restricted.
IIdType =	Flag indicating whether the ImplName parameter is a NULL terminated list of context path names or a NULL terminated list of LOIDs.
ImplName =	NULL terminated array of LOIDs or context path names of the Impls to which the instantiation is restricted.

Return Values:

NULL terminated array of LOIDs indicating the objects that were instantiated. Each element of the array is a char* which is the LOID of one object created.

Error Return:
NULL

User Responsibility:
User is responsible for "free"ing each LOID in the array as well as the array itself.

char**
ComplxScheduledCreateRestricted(*int NumObjects,*
*LegionIdType CIDType, char *ClassName,*
*LegionIdType HIDType, char **HostName,*
*LegionIdType VIDType, char **VaultName,*
*LegionIdType IIDType, char **ImplName,*
*LegionIdType SIDType, char *SchedName*);

Create a vector of new Legion objects by vector instantiating the given class. The user also indicates any combination of restrictions which may include restrictions to hosts, vaults or implementations. In addition, the user may specify a scheduler to use for the instantiation

Parameters:

NumObjects =	Number of object to vector instantiate.
CIdType =	Flag indicating whether the ClassName parameter is a context path name or a LOID.
ClassName =	LOID or context path name of the class to instantiate.
HIdType =	Flag indicating whether the HostName parameter is a NULL terminated list of context path names or a NULL terminated list of LOIDs.
HostName =	NULL terminated array of LOIDs or context path names of the Hosts to which the instantiation is restricted.
VIdType =	Flag indicating whether the VaultName parameter is a NULL terminated list of context path names or a NULL terminated list of LOIDs.
VaultName =	NULL terminated array of LOIDs or context path names of the Vaults to which the instantiation is restricted.
IIdType =	Flag indicating whether the ImplName parameter is a NULL terminated list of context path names, or a NULL terminated list of LOIDs.
ImplName =	NULL terminated array of LOIDs or context path names of the Impls to which the instantiation is restricted.
SIdType =	Flag indicating whether the SchedName parameter is a context path name or a LOID.

SchedName = context path name, or LOID of the scheduler to use when instantiating the class.

Return Values:
NULL terminated array of LOIDs indicating the objects that were instantiated. Each element of the array is a char* which is the LOID of one object created.

Error Return:
NULL

User Responsibility:
User is responsible for "free"ing each LOID in the array as well as the array itself.

int
Activate(*LegionIdType IType, char *ObjectName*);
Activate an inert Legion object

Parameters:
IdType = Flag indicating whether the ObjectName parameter is a context path name or a LOID.
ObjectName = LOID or context path name of the object to activate.

Return Values:
1 on Success
0 on failure

User Responsibility: None

int
ScheduledActivate(*LegionIdType OIDType, char *ObjectName, LegionIdType SIDType, char *SchedName*);
Activate an inert Legion object (using the indicated scheduler to schedule the activation)

Parameters:
OIDType = Flag indicating whether the ObjectName parameter is a context path name or a LOID.
ObjectName = LOID or context path name of the object to activate.
SIdType = Flag indicating whether the SchedName parameter is a context path name or a LOID.
SchedName = LOID or context path name of the scheduler to use when doing the activation.

Return Values:
1 on Success
0 on failure

User Responsibility: None

int
ActivateRestricted(*LegionIdType OIDType, char *ObjectName,*
*LegionIdType HIDType, char *HostName,*
*LegionIdType VIDType, char *VaultName,*
*LegionIdType IIDType, char *ImplName*);

Activate an inert Legion object, but subject to the restrictions indicated by the user.

Parameters:

OIDType =	Flag indicating whether the ObjectName parameter is a context path name or a LOID.
ObjectName =	LOID or context path name of the object to activate.
HIdType =	Flag indicating whether the HostName parameter is a context path name or a LOID.
HostName =	LOID or context path name of the Host to which the activation is restricted.
VIDType =	Flag indicating whether the VaultName parameter is a context path name or a LOID.
VaultName =	LOID or context path name of the Vault to which the activation is restricted.
IIDType =	Flag indicating whether the ImplName parameter is a context path name or a LOID.
ImplName =	LOID or context path name of the Impl to which the activation is restricted.

Return Values:
1 on Success
0 on Failure

User Responsibility: None

int
ScheduledActivateRestricted(
*LegionIdType OIDType, char *ObjectName,*
*LegionIdType HIDType, char *HostName,*
*LegionIdType VIDType, char *VaultName,*
*LegionIdType IIDType, char *ImplName,*
*LegionIdType SIDType, char *SchedName*);

Activate an inert Legion object, but subject to the restrictions indicated by the user. Also, use the indicated scheduler to schedule the activation.

Parameters:

OIDType =	Flag indicating whether the ObjectName parameter is a context path name or a LOID.
ObjectName =	LOID or context path name of the object to activate.
HIdType =	Flag indicating whether the HostName parameter is a context path name or a LOID.
HostName =	LOID or context path name of the Host to which the activation is restricted.
VIDType =	Flag indicating whether the VaultName parameter is a context path name or a LOID.
VaultName =	LOID or context path name of the Vault to which the activation is restricted.
IIDType =	Flag indicating whether the ImplName parameter is a context path name or a LOID.
ImplName =	LOID or context path name of the Impl to which the activation is restricted.
SIDType =	Flag indicating whether the SchedName parameter is a context path name or a LOID.
SchedName =	LOID or context path name of the Scheduler to use when doing the activation.

Return Values:

1 on Success
0 on failure

User Responsibility: None

int
ComplxActivateRestricted(
*LegionIdType OIDType, char *ObjectName,*
*LegionIdType HIDType, char **HostName,*
*LegionIdType VIDType, char **VaultName,*
*LegionIdType IIDType, char **ImplName*);

Activate an inert Legion object, but subject to the restrictions indicated by the user.

Parameters:

OIDType =	Flag indicating whether the ObjectName parameter is a context path name or a LOID.
ObjectName =	LOID or context path name of the object to activate.
HIdType =	Flag indicating whether the HostName parameter is a list of context path names or LOIDs.
HostName =	A NULL terminated array of LOIDs or context path names of Hosts to which the activation is restricted.
VIdType =	Flag indicating whether the VaultName parameter is a list of context path names

or LOIDs.

VaultName = A NULL terminated array of LOIDs or context path names of Vaults to which the activation is restricted.

IIdType = Flag indicating whether the ImplName parameter is a list of context path names or LOIDs.

ImpleName = A NULL terminated array of LOIDs or context path names of Implementations to which the activation is restricted.

Return Values:

1 on Success
0 on failure

User Responsibility: None

int
ComplxScheduledActivateRestricted(
*LegionIdType OIDType, char *ObjectName,*
*LegionIdType HIDType, char **HostName,*
*LegionIdType VIDType, char **VaultName,*
*LegionIdType IIDType, char **ImplName,*
*LegionIdType SIDType, char *SchedName*);

Activate an inert Legion object, but subject to the restrictions indicated by the user and using the given scheduler.

Parameters:

OIDType = Flag indicating whether the ObjectName parameter is a context path name or a LOID.

ObjectName = LOID or context path name of the object to activate.

HIdType = Flag indicating whether the HostName parameter is a list of context path names, or LOIDs.

HostName = A NULL terminated array of LOIDs or context path names of Hosts to which the activation is restricted.

VIdType = Flag indicating whether the VaultName parameter is a list of context path names or LOIDs.

VaultName = A NULL terminated array of LOIDs or context path names of Vaults to which the activation is restricted.

IIdType = Flag indicating whether the ImplName parameter is a list of context path names or LOIDs.

ImpleName = A NULL terminated array of LOIDs or context path names of Implementations to which the activation is restricted.

SIDType = A flag indicating whether the SchedName parameter is a context path name or a LOID.
SchedName = A context path name, or a LOID indicating the scheduler to use when doing the activation.

Return Values:
1 on Success
0 on failure

User Responsibility: None

int
DeleteObject(*LegionIdType IdType, char *ObjName*);
Deletes or destroys a given object by asking its class to get destroy it. This function is identical to DestroyObject().

Parameters:
IdType = Flag indicating whether parameter is a LOID or a context path name.
Name = Either the context path name or the LOID of the object to delete.

Return Values:
1 on Success
0 on Failure

User Responsibility: None

int
DestroyObject(*LegionIdType IdType, char *ObjName*);
Deletes or destroys a given object by asking its class to get destroy it. This function is identical to DeleteObject().

Parameters:
IdType = Flag indicating whether parameter is a LOID or a context path name.
Name = Either the context path name or the LOID of the object to destroy.

Return Values:
1 on Success
0 on Failure

User Responsibility: None

int
DeactivateObject(*LegionIdType IdType, char *ObjName*);

Deactivates a given object.

Parameters:

IdType =	Flag indicating whether parameter is a LOID or a context path name.
Name =	Either the context path name or the LOID of the object to deactivate.

Return Values:

1 on Success
0 on Failure

User Responsibility: None

int
ForcedDeactivateObject(*LegionIdType IdType, char *ObjName*);

Deactivates a given object. The forced in this call means that if the object is going up or going down, then an error is returned.

Parameters:

IdType =	Flag indicating whether parameter is a LOID or a context path name.
Name =	Either the context path name, or the LOID of the object to deactivate.

Return Values:

1 on Success
0 on Failure

User Responsibility: None

int
GetObjectType(*LegionIdType IdType, char *ObjName*);

Returns an integer which indicates as specifically as possible what type of object the user is talking about.

Parameters:

IdType =	Flag indicating whether parameter is a LOID or a context path.
ObjName =	Either the context path name or the LOID of the object to be described.

Return Values:

Returns an integer indicating the type of the object (from the list above).

User Responsibility: None

```
#defineLEGION_OBJ_CONTEXT              1  /* These constants are  */
#defineLEGION_OBJ_FILE                 2  /* used to indicate     */
#defineLEGION_OBJ_RUNNABLE             3  /* what type of object */
#defineLEGION_OBJ_CLASS                4  /* one is talking about. */
#defineLEGION_OBJ_HOST                 5  /* These are the values  */
#defineLEGION_OBJ_VAULT                6  /* that can be returned*/
#defineLEGION_OBJ_TTY                  7  /* by GetObjectType()   */
#defineLEGION_OBJ_IMPLEMENTATION       8
#defineLEGION_OBJ_IMPLEMENTATION_CACHE 9
#defineLEGION_OBJ_LEGION_CLASS         10
#defineLEGION_OBJ_BINDING_AGENT        11
#defineLEGION_OBJ_MPI                  12
#defineLEGION_OBJ_PVM3                 13
#defineLEGION_OBJ_UNKNOWN              14
```

int
isContext(*LegionIdType IdType, char *ObjName*);

Determines if the indicated object is a context object or not.

Parameters:

IdType =	Flag indicating whether parameter is a LOID or a context path name.
ObjName =	The context path name or the LOID of the object to test.

Return Values:

1 if the object is a context
0 if the object is not a context

User Responsibility: None

int
isFile(*LegionIdType IdType, char *ObjName*);

Determines if the indicated object is a File object or not.

Parameters:

IdType =	Flag indicating whether parameter is a LOID or a context path name.
ObjName =	The context path name or the LOID of the object to test.

Return Values:

1 if the object is a File
0 if the object is not a File

User Responsibility: None

int
isRunnable(*LegionIdType IdType, char *ObjName*);
Determines if the indicated object is a Runnable object or not.

Parameters:
IdType = Flag indicating whether parameter is a LOID or a context path name.
ObjName = Either the context path name, or the LOID of the object to test.

Return Values:
1 if the object is a Runnable
0 if the object is not a Runnable

User Responsibility: None

int
isClass(*LegionIdType IdType, char *ObjName*);
Determines if the indicated object is a Class object or not.

Parameters:
IdType = Flag indicating whether parameter is a LOID or a context path name.
ObjName = The context path name or the LOID of the object to test.

Return Values:
1 if the object is a Class
0 if the object is not a Class

User Responsibility: None

int
isHost(*LegionIdType IdType, char *ObjName*);
Determines if the indicated object is a Host object or not.

Parameters:
IdType = Flag indicating whether parameter is a LOID or a context path name.
ObjName = The context path name or the LOID of the object to test.

Return Values:
1 if the object is a Host
0 if the object is not a Host

User Responsibility: None

int
isVault(*LegionIdType IdType, char *ObjName*);
Determines if the indicated object is a Vault object or not.

Parameters:
IdType = Flag indicating whether parameter is a LOID or a context path name.
ObjName = The context path name or the LOID of the object to test.

Return Values:
1 if the object is a Vault
0 if the object is not a Vault

User Responsibility: None

int
isTTY(*LegionIdType IdType, char *ObjName*);
Determines if the indicated object is a TTY object or not.

Parameters:
IdType = Flag indicating whether parameter is a LOID or a context path name.
ObjName = The context path name or the LOID of the object to test.

Return Values:
1 if the object is a TTY
0 if the object is not a TTY

User Responsibility: None

int
isImplementation(*LegionIdType IdType, char *ObjName*);
Determines if the indicated object is a Implementation object or not.
Parameters:
IdType = Flag indicating whether parameter is a LOID or a context path name.
ObjName = The context path name or the LOID of the object to test.

Return Values:
1 if the object is a Implementation
0 if the object is not a Implementation

User Responsibility: None

int
isImplementationCache(*LegionIdType IdType, char *ObjName*);

Determines if the indicated object is a ImplementationCache object or not.

Parameters:

IdType =	Flag indicating whether parameter is a LOID or a context path name.
ObjName =	The context path name or the LOID of the object to test.

Return Values:

1 if the object is a ImplementationCache
0 if the object is not a ImplementationCache

User Responsibility: None

int
isLegionClass(*LegionIdType IdType, char *ObjName*);

Determines if the indicated object is a LegionClass object or not.

Parameters:

IdType =	Flag indicating whether parameter is a LOID or a context path name.
ObjName =	The context path name or the LOID of the object to test.

Return Values:

1 if the object is a LegionClass
0 if the object is not a LegionClass

User Responsibility: None

int
isBindingAgent(*LegionIdType IdType, char *ObjName*);

Determines if the indicated object is a BindingAgent object or not.

Parameters:

IdType =	Flag indicating whether parameter is a LOID or a context path name.
ObjName =	The context path name or the LOID of the object to test.

Return Values:

1 if the object is a BindingAgent
0 if the object is not a BindingAgent

User Responsibility: None

int
isMPI(*LegionIdType IdType, char *ObjName*);
Determines if the indicated object is a MPI object or not.

Parameters:

IdType =	Flag indicating whether parameter is a LOID or a context path name.
ObjName =	The context path name or the LOID of the object to test.

Return Values:
1 if the object is a MPI
0 if the object is not a MPI

User Responsibility: None

int
isPVM3(*LegionIdType IdType, char *ObjName*);
Determines if the indicated object is a PVM3 object or not.

Parameters:

IdType =	Flag indicating whether parameter is a LOID or a context path name.
ObjName =	The context path name or the LOID of the object to test.

Return Values:
1 if the object is a PVM3
0 if the object is not a PVM3

User Responsibility: None

1.5 Attributes

The following functions deal with the attribute database that all Legion objects support.

int
AddAttribute(*LegionIdType IdType, char *ObjName, char *Attribute*);
Add an attribute to an object's attribute database.

Parameters:

IdType =	Flag indicating whether parameter is a LOID or a context path name.
ObjName =	The context path name or the LOID of the object to which the attribute is to be added.
Attribute =	The string representation of the attribute to add to the database. This is of the form `attribute_signature(attribute_value)`.

Return Values:
1 on Success
0 on Failure

User Responsibility: None

int
AddAttributes(*LegionIdType IdType, char *ObjName, char **Attribute*);
Add a list of attributes to an object's attribute database.

Parameters:

IdType =	Flag indicating whether parameter is a LOID or a context path name.
ObjName =	The context path name or the LOID of the object to add the attribute to.
Attribute =	A NULL terminated list of the string representations of the attributes to add to the database. These are of the form `attribute_signature(attribute_value)`.

Return Values:
1 on Success
0 on Failure

User Responsibility: None

int
ReplaceAttribute(*LegionIdType IdType, char *ObjName, char *OldAttribute, char *NewAttribute*);
Replace an attribute in an object's attribute database.

Parameters:

IdType =	Flag indicating whether parameter is a LOID or a context path name.
ObjName =	The context path name or the LOID of the object to add the attribute to.
OldAttribute =	The string representation of the old attribute to replace in the database. This is of the form `attribute_signature(attribute_value)`.
NewAttribute =	The string representation of the new attribute to put into the database. This is of the form `attribute_signature(attribute_value)`.

Return Values:
1 on Success
0 on Failure

User Responsibility: None

int
ReplaceAttributeSig(*LegionIdType IdType, char *ObjName, char *OldAttributeSig, char *NewAttribute*);

Replace an attribute in an object's attribute database.

Parameters:

- IdType = Flag indicating whether parameter is a LOID or a context path name.
- ObjName = The context path name or the LOID of the object to which the attribute will be added.
- OldAttributeSig = The string representation of the old attribute's attribute signature.
- NewAttribute = The string representation of the new attribute to put into the database. This is of the form `attribute_signature(attribute_value)`.

Return Values:

- 1 on Success
- 0 on Failure

User Responsibility: None

int
ReplaceAttributes(*LegionIdType IdType, char *ObjName, char **OldAttributes, char **NewAttributes*);

Replace a list of attributes in an object's attribute database.

Parameters:

- IdType = Flag indicating whether parameter is a LOID or a context path name.
- ObjName = The context path name or the LOID of the object to add the attribute to.
- OldAttributes = A NULL terminated list of the string representations of the old attributes to replace in the database. These are of the form `attribute_signature(attribute_value)`.
- NewAttributes = A NULL terminated list of the string representations of the new attributes to put into the database. These are of the form `attribute_signature(attribute_value)`.

Return Values:

- 1 on Success
- 0 on Failure

User Responsibility: None

int
ReplaceAttributeSigs(*LegionIdType IdType, char *ObjName,*
char **OldAttributeSigs, *char **NewAttributes*);
Replace a list of attributes in an object's attribute database.

Parameters:

IdType =	Flag indicating whether parameter is a LOID or a context path name.
ObjName =	The context path name or the LOID of the object to add the attribute to.
OldAttributeSigs =	A NULL terminated list of the string representations of the old attribute signatures to replace in the database.
NewAttributes=	A NULL terminated list of the string representations of the new attributes to put into the database. These are of the form `attribute_signature(attribute_value)`.

Return Values:
1 on Success
0 on Failure

User Responsibility: None

int
RemoveAttribute(*LegionIdType IdType,*
*char *ObjName, char *Attribute*);
Remove an attribute in an object's attribute database.

Parameters:

IdType =	Flag indicating whether parameter is a LOID or a context path name.
ObjName =	The context path name or the LOID of the object to add the attribute to.
Attribute =	The string representation of the attribute to remove from the object's attribute database.

Return Values:
1 on Success
0 on Failure

User Responsibility: None

int
RemoveAttributeSig(*LegionIdType IdType,*
*char *ObjName, char *AttributeSig*);
Remove an attribute in an object's attribute database.

Parameters:

IdType =	Flag indicating whether parameter is a LOID or a context path name.
ObjName =	The context path name or the LOID of the object to add the attribute to.
Attribute =	The string representation of the signature of the attribute to remove from the object's attribute database.

Return Values:
1 on Success
0 on Failure

User Responsibility: None

int
RemoveAttributes(*LegionIdType IdType,*
*char *ObjName, char **Attributes*);
Remove a set of attributes from an object's attribute database.

Parameters:

IdType =	Flag indicating whether parameter is a LOID or a context path name.
ObjName =	The context path name or the LOID of the object to add the attribute to.
Attributes =	A NULL terminated list of the string representations of the attributes to remove from the object's attribute database.

Return Values:
1 on Success
0 on Failure

User Responsibility: None

int
RemoveAttributeSigs(*LegionIdType IdType,*
*char *ObjName, char **AttributeSigs*);
Remove a set of attributes from an object's attribute database.

Parameters:

IdType =	Flag indicating whether parameter is a LOID or a context path name.
ObjName =	The context path name or the LOID of the object to add the attribute to.

AttributeSigs = A NULL terminated list of the string representations of the signatures of attributes to remove from the object's attribute database.

Return Values:
1 on Success
0 on Failure

User Responsibility: None

char*
RetrieveAttribute(*LegionIdType IdType,*
*char *ObjName, char *Attribute*);
Retrieve an attribute from an object's attribute database.

Parameters:
IdType = Flag indicating whether parameter is a LOID or a context path name.
ObjName = The context path name or the LOID of the object to add the attribute to.
Attribute = The string representation of the attribute to retrieve from the object's attribute database.

Return Values:
The string representation of the attribute retrieved.

Error Return:
NULL

User Responsibility:
The user is responsible for "free"ing the returned attribute.

char**
RetrieveAttributes(*LegionIdType IdType,*
*char *ObjName, char **Attributes*);
Retrieve a set of attributes from an object's attribute database.

Parameters:
IdType = Flag indicating whether parameter is a LOID or a context path name.
ObjName = The context path name or the LOID of the object to add the attribute to.
Attributes = A NULL terminated list of the string representations of the attributes to retrieve from the object's attribute database.

Return Values:
A NULL terminated list of the string representations of the attributes retrieved.

Error Return:
NULL

User Responsibility:
The user is responsible for "free"ing each returned attribute in the list as well as for "free"ing the list itself.

char*
RetrieveAttributeSig(*LegionIdType IdType,*
*char *ObjName, char *AttributeSig*);
Retrieve an attribute from an object's attribute database.

Parameters:

IdType =	Flag indicating whether parameter is a LOID or a context path name.
ObjName =	Either the context path name, or the LOID of the object to add the attribute to.
AttributeSig =	The string representation of the signature of the attribute to retrieve from the object's attribute database.

Return Values:
The string representation of the attribute retrieved.

Error Return:
NULL

User Responsibility:
The user is responsible for "free"ing the returned attribute.

char**
RetrieveAttributeSigs(*LegionIdType IdType,*
*char *ObjName, char **AttributeSigs*);
Retrieve a list of attributes from an object's attribute database.

Parameters:

IdType =	Flag indicating whether parameter is a LOID or a context path name.
ObjName =	The context path name or the LOID of the object to add the attribute to.
AttributeSigs =	A NULL terminated list of the string representations of the signatures of attributes to retrieve from the object's attribute database.

Return Values:
A NULL terminated list of the string representations of the attributes retrieved.

Error Return:
NULL

User Responsibility:
The user is responsible for "free"ing each returned attribute in the list as well as for "free"ing the list itself.

char**
RetrieveAllAttributes(*LegionIdType IdType, char *ObjName*);
Retrieve all of the attributes from an object's attribute database.

Parameters:

IdType =	Flag indicating whether parameter is a LOID or a context path name.
ObjName =	The context path name or the LOID of the object to add the attribute to.

Return Values:
A NULL terminated list of the string representations of the attributes retrieved.

Error Return:
NULL

User Responsibility:
The user is responsible for "free"ing each returned attribute in the list as well as for "free"ing the list itself.

The following function deals with the interfaces to Legion objects.

char**
GetInterface(*LegionIdType, char *ObjName*);
Retrieve all of the method signatures for a Legion object.

Parameters:

IdType =	Flag indicating whether parameter is a LOID or a context path name.
ObjName =	The context path name or the LOID of the object to retrieve the interface for.

Return Values:
A NULL terminated list of the string representations of the method's which are currently part of the indicated object's interface.

Error Return:
NULL

User Responsibility:
The user is responsible for "free"ing each returned signature in the list as well as for "free"ing the list itself.

1.6 C++ type manipulation

The following section contains functions which allow for manipulation of typedefs which are normally C++ classes.

void
Show_CLegionHostReservation(
*CLegionHostReservation, FILE**);
void
Show_CLegionHostReservationRecord(
*CLegionHostReservationRecord, FILE**);
void
Show_CLegionVaultReservation(
*CLegionVaultReservation, FILE**);
void
Show_CLegionVaultReservationRecord(
*CLegionVaultReservationRecord, FILE**);
void
Show_CLegionReservation(*CLegionReservation, FILE**);
void
Show_CLegionBinding(*CLegionBinding, FILE**);
void
Show_CLegionHostObjectStatus(
*CLegionHostObjectStatus, FILE**);
void
Show_CLegionOPRAddress(*CLegionOPRAddress, FILE**);
void
Show_CUVaL_InstanceRecord(*CUVaL_InstanceRecord, FILE**);
void
Show_CInstancePlacementInfo(*CInstancePlacementInfo, FILE**);
void
Show_CLegionCollectionData(*CLegionCollectionData, FILE**);

Call the C++ show function on the given C++ class (named by Show_xxx).

Parameters:
All show functions take two parameters, the type to show and the stream to "show" the output on.

Return Values:
NONE

User Responsibility: NONE

void
DestroyHostReservation(*CLegionHostReservation*);
void
DestroyHostReservationRecord(
CLegionHostReservationRecord);
void
DestroyVaultReservation(*CLegionVaultReservation*);
void
DestroyVaultReservationRecord(
CLegionVaultReservationRecord);
void
DestroyReservation(*CLegionReservation*);
void
DestroyLegionBinding(*CLegionBinding*);
void
DestroyHostObjectStatus(*CLegionHostObjectStatus*);
void
DestroyLegionOPRAddress(*CLegionOPRAddress*);
void
DestroyUVaL_InstanceRecord(*CUVaL_InstanceRecord*);
void
DestroyInstancePlacementInfo(*CInstancePlacementInfo*);
void
DestroyLegionCollectionData(*CLegionCollectionData*);

All of the complicated C++ types that are indirectly represented here as typedefs to ints must be explicitly destroyed so that there destructors may be called. To destroy an object of type xxx, call the Destroyxxx function on its C typedef.

Parameters:

All destroy functions take one parameter, the variable of the given type to destroy.

Return Values:

NONE

User Responsibility: NONE

int
HostReservationRecord_GetStatus(
CLegionHostReservationRecord);
char*
HostReservationRecord_GetObjLoid(
CLegionHostReservationRecord);
int
VaultReservationRecord_GetStatus(
CLegionVaultReservationRecord);
char*
VaultReservationRecord_GetObjLoid(

CLegionVaultReservationRecord);
CLegionHostReservation
LegionReservation_GetHostReservation(*CLegionReservation*);
CLegionVaultReservation
LegionReservation_GetVaultReservation(
CLegionReservation);
CLegionHostReservation
CreateLegionHostReservation(*LegionIdType, char *HostName,
int identity, int flags*);
CLegionVaultReservation
CreateLegionVaultReservation(*LegionIdType,
char *VaultName,int identity*);
CLegionReservation
CreateReservation(*CLegionHostReservation,
CLegionVaultReservation*);
char*
InstancePlacementInfo_GetInstance(
CInstancePlacementInfo);
char*
InstancePlacementInfo_GetHost(*CInstancePlacementInfo*);
char*
InstancePlacementInfo_GetImpl(*CInstancePlacementInfo*);
char*
InstancePlacementInfo_GetVault(*CInstancePlacementInfo*);
CLegionReservation
InstancePlacementInfo_GetRSVN(*CInstancePlacementInfo*);
int
InstancePlacementInfo_GetArch(*CInstancePlacementInfo*);
CInstancePlacementInfo
CreateInstancePlacementInfo(*char *InstanceLOID,
LegionIdType, char *HostName,
LegionIdType, char *ImplName,
LegionIdType, char *VaultName,
CLegionReservation,int Arch*);
char*
HostObjectStatus_GetLoid(*CLegionHostObjectStatus*);
char*
HostObjectStatus_GetOwner(*CLegionHostObjectStatus*);
int
HostObjectStatus_GetActivationID(*CLegionHostObjectStatus*);
int
HostObjectStatus_GetStatus(*CLegionHostObjectStatus*);
int
HostObjectStatus_GetErrorCode(*CLegionHostObjectStatus*);
CLegionBinding
HostObjectStatus_GetBinding(*CLegionHostObjectStatus*);
char*
UVaL_InstanceRecord_GetLoid(*CUVaL_InstanceRecord*);
char*
UVaL_InstanceRecord_GetOwner(*CUVaL_InstanceRecord*);

int
UVaL_InstanceRecord_GetStatus(*CUVaL_InstanceRecord*);

char
UVaL_InstanceRecord_GetCommandLineObj(
CUVaL_InstanceRecord);
unsigned long
UVaL_InstanceRecord_GetLastTransitionTime(
CUVaL_InstanceRecord);
int
UVaL_InstanceRecord_GetActivationId(*CUVaL_InstanceRecord*);
char*
UVaL_InstanceRecord_GetHost(*CUVaL_InstanceRecord*);
char*
UVaL_InstanceRecord_GetVault(*CUVaL_InstanceRecord*);
int
UVaL_InstanceRecord_GetTransferringOpr(
CUVaL_InstanceRecord);
char*
UVaL_InstanceRecord_GetTransferringToVault(
CUVaL_InstanceRecord);
int
UVaL_InstanceRecord_GetTesting(*CUVaL_InstanceRecord*);
unsigned long
UVaL_InstanceRecord_GetStartTestTime(
CUVaL_InstanceRecord);
unsigned long
UVaL_InstanceRecord_GetXferTime(*CUVaL_InstanceRecord*);
unsigned int
UVaL_InstanceRecord_GetHostTries(*CUVaL_InstanceRecord*);
char*
LegionCollectionData_GetLOID(*CLegionCollectionData*);
long
LegionCollectionData_GetLastUpdateTime(
CLegionCollectionData);

```
*/ These resource descriptions are in the form of a/*
*/ null terminated list of UVaL_ObjAttributes     /*
*/ (in string form)                               /*
```

char**
LegionCollectionData_GetResourceDesc(
CLegionCollectionData);

All of the complicated C++ types that are indirectly represented here as typedefs to ints have internal members which are useful to access. For that purpose, a number of "accessor" functions are provided. All function names are of the form <C++ TypeName>_Get<C++ Field>.

Parameters:

All "accessor" functions take one parameter, the variable of from which the user wishes to access a member.

Return Values:

All "accessor" functions return one of three things:

- a basic integral type
- another C typedef for a C++ class
- a char* which is the LOID of the value requested

User Responsibility:

When a C typedef for a C++ class is returned, the user is responsible for Destroying that type when done. When a char* is returned, the user is responsible for "free"ing the dynamically created memory.

1.7 Host object

The following section contains functions which are used to make calls directly on host objects.

CLegionHostReservation
MakeReservationFromHost(
*LegionIdType HIdType, char *HostName,*
*LegionIdType VIdType, char *ReqVltName*);

Ask a host to make a reservation for either an instantiation, or an activation.

Parameters

HIdType =	A Flag indicating whether HostName is a context path name or a LOID.
HostName =	The name of the host with which the reservation should be made.
VIdType =	A Flag indicating whether ReqVltName is a context path name or a LOID.
ReqVltName =	The name of the vault to be paired up with this host reservation.

Return Values:

The CLegionHostReservation that represents the HostReservation just made.

Error Return:

0

User Responsibility:

User is responsible for destroying the host reservation when done.

CLegionHostReservationRecord
CheckHostReservation(*LegionIdType HIdType, char *HostName,*
CLegionHostReservation HstRes);

Check on the status of a host reservation previously made.

Parameters:

HIdType =	A Flag indicating whether HostName is a context path name, or a LOID.
HostName =	The name of the host that will check the reservation.
HstRes =	The Host reservation previously made.

Return Values:

A CLegionHostReservationRecord containing the status of the reservation.

Error Return:

0

User Responsibility:

User is responsible for destroying the host reservation record when done.

int
CancelHostReservation(*LegionIdType HIdType, char *HostName*, *CLegionHostReservation HstRes*);

Cancel a previously made host reservation.

Parameters:

HIdType =	A Flag indicating whether HostName is a context path name, or a LOID.
HostName =	The name of the host that will cancel the reservation.
HstRes =	The Host reservation previously made.

Return Values:

1 on Success
0 on Failure

User Responsibility: None

int
GetImplArchFromHost(*LegionIdType HIdType, char *HostName*);

Retrieve the implementation architecture for a given host.

Parameters:

HIdType =	A Flag indicating whether HostName is a context path name, or a LOID.
HostName =	The name of the host to get the arch from.

Return Values:

The architecture of the given host.

Error Return:

-1

User Responsibility: None

int
KillObject(*LegionIdType HIdType, char *HostName,*
CLegionBinding binding, int actId);
Kill an object running on a host.

Parameters:
HIdType = A Flag indicating whether HostName is a context path name or a LOID
HostName = The name of the host to kill the object on
binding = The binding to the object that you wish to kill.
actId = The host's activation id for the object to kill.

Return Values:
1 on Success
0 on Failure

User Responsibility: None

int
HostDeactivateObject(*LegionIdType HIdType, char *HostName,*
CLegionBinding binding, int actId);
Ask the host to deactivate an object it is running.

Parameters:
HIdType = A Flag indicating whether HostName is a context path name or a LOID.
HostName= The name of the host to deactivate the object on.
binding = The binding to the object that you wish to deactivate.
actId = The host's activation id for the object to deactivate.

Return Values:
1 on Success
0 on Failure

User Responsibility: None

CLegionHostObjectStatus*
ListObjects(*LegionIdType HIdType, char *HostName*);
Ask the host to return a list of objects that it is currently managing.

Parameters:
HIdType = A Flag indicating whether HostName is a context path name, or a LOID.

HostName = The name of the host to acquire the list from.

Return Values:
A NULL terminated list of CLegionHostObjectStatus's. Each array element represents the status of one object currently being managed by a host.

Error Return:
NULL

User Responsibility:
The user is responsible for destroying each CLegionHostObjectStatus in the list, as well as for "free"ing the list itself.

char**
GetCompatibileVaults(*LegionIdType HIdType, char *HostName*);
Ask the host to return a list of vaults that are compatible with it.

Parameters:
HIdType = A Flag indicating whether HostName is a context path name or a LOID.
HostName= The name of the host to check.

Return Values:
A NULL terminated list of LOIDs. Each LOID represents one vault that is compatible with the given host.

Error Return:
NULL

User Responsibility:
The user is responsible for "free"ing each LOID, as well as for "free"ing the list itself.

int
VaultOK(*LegionIdType HIdType, char *HostName,*
*LegionIdType VIdType, char *VaultName*);
Ask the host if a certain vault is compatible with it or not.

Parameters:
HIdType = A Flag indicating whether HostName is a context path name or a LOID.
HostName= The name of the host to check the vault with.
VIdType = A Flag indicating whether VaultName is a

context path name or a LOID.

VaultName= The name of the vault to check for compatibility with.

Return Values:

0 if the vault is not OK.

1 if the vault is OK.

Error Return:

-1

User Responsibility: None

int
GetObjectStatus(*LegionIdType HIdType, char *HostName, CLegionBinding Binding*);

Ask the host for the status of one of its objects.

Parameters:

HIdType = A Flag indicating whether HostName is a context path name, or a LOID.

HostName= The name of the host to check the vault with.

Binding = The object in question's binding.

Return Values:

The object's status.

User Responsibility: None

char*
GetImplementationCache(*LegionIdType HIdType, char *HostNam*);

Ask the host for the LOID of its ImplementationCacheObject.

Parameters:

HIdType = A Flag indicating whether HostName is a context path name or a LOID.

HostName= The name of the host to check the vault with

Return Values:

The LOID of the host object's implementation cache.

Error Return:

NULL

User Responsibility:

The user is responsible for "free"ing the memory returned by this function.

char*
ChangeObjectOwner(*LegionIdType HIdType, char *HostName,*
*LegionIdType OOIdType, char *OrigOwnerNm,*
*LegionIdType NOIdType, char *NewOwnerNm*);

Change the owner of a certain object that the host is running.

Parameters:

HIdType =	A Flag indicating whether HostName is a context path name, or a LOID.
HostName =	The name of the host to check the vault with.
OOIdType =	A Flag indicating whether OrigOwnerNm is a context path name or a LOID.
OrigOwnerNm =	The name of the Original owner of the object.
NOIdType =	A Flag indicating whether NewOwnerNm is a context path name or a LOID.
NewOwnerNm =	The name of the New owner of the object.

Return Values:

The LOID of the object's owner.

Error Return:

NULL

User Responsibility:

The user is responsible for "free"ing the memory returned by this function.

1.8 Vault object

The following section contains functions which are used to make calls directly on host objects.

CLegionVaultReservation
MakeReservationFromVault(
*LegionIdType VIdType, char *VaultName,*
*LegionIdType HIdType, char *ReqHostN*);

Ask a vault to make a reservation for either an instantiation, or an activation.

Parameters:

VIdType =	A Flag indicating whether VaultName is a context path name, or a LOID.
VaultName =	The name of the vault that will make the reservation.
HIdType =	A Flag indicating whether ReqHostName is

a context path name, or a LOID.
ReqHostName =The name of the host to be paired up with this vault reservation.

Return Values:
The CLegionVaultReservation that represents the VaultReservation just made.

Error Return:
0

User Responsibility:
User is responsible for destroying the vault reservation when done.

CLegionVaultReservationRecord
CheckVaultReservation(*LegionIdType VIdType, char *VaultName, CLegionVaultReservation VtRes*);
Check on the status of a vault reservation previously made.

Parameters:
HIdType = A Flag indicating whether VaultName is a context path name or a LOID.
VaultName = The name of the vault that will check the reservation.
VtRes = The Vault reservation previously made.

Return Values:
A CLegionVaultReservationRecord containing the status of the reservation.

Error Return:
0

User Responsibility:
User is responsible for destroying the vault reservation record when done.

int
CancelVaultReservation(*LegionIdType VIdType, char *VaultName, CLegionVaultReservation VtRes*);
Cancel a previously made vault reservation.

Parameters:
HIdType = A Flag indicating whether VaultName is a context path name or a LOID.
VaultName = The name of the vault that will cancel the reservation.
VtRes = The Vault reservation previously made.

Return Values:

1 on Success
0 on Failure

User Responsibility: None

int
HostOK(*LegionIdType VIdType, char *VaultName,*
*LegionIdType HIdType, char *HostName*);
Ask the vault if a certain host is compatible with it or not.

Parameters:
VIdType = A flag indicating whether VaultName is a context path name or a LOID.
VaultName = The name of the vault that will check the host.
HIdType = A flag indicating whether HostName is a context path name or a LOID.
HostName = The name of the host to check for compatibility.

Return Values:
0 if the host is not OK.
1 if the host is OK.

Error Return:
-1

User Responsibility: None

char**
GetCompatibleHosts(*LegionIdType VIdType, char *VaultName*);
Ask the vault to return a list of hosts that are compatible with it.

Parameters:
VIdType = A flag indicating whether VaultName is a context path name or a LOID.
VaultName= The name of the vault to check.

Return Values:
A NULL terminated list of LOIDs. Each LOID represents one host that is compatible with the given vault.

Error Return:
NULL

User Responsibility:
The user is responsible for "free"ing each LOID, as well as for "free"ing the list itself.

int
TransferOPRsAndDestroySelf(*LegionIdType VIdType,*

*char *VaultName*);

Ask the vault to transfer all of its OPRs to another vault, and then to die.

Parameters:

VIdType =	A flag indicating whether VaultName is a context path name or a LOID.
VaultName =	The name of the vault to talk to.

Return Values:

1 on Success
0 on Failure

User Responsibility: None

char*
ChangeOPROwner(*LegionIdType VIdType, char *VaultName, LegionIdType OOIdType, char *OrigOwnerNm, LegionIdType NOIdType, char *NewOwnerNm*);

Change the owner of a certain OPR that the vault is managing.

Parameters:

VIdType =	A flag indicating whether VaultName is a context path name or a LOID.
VaultName =	The name of the vault to talk with.
OOIdType =	A flag indicating whether OrigOwnerNm is a context path name or a LOID.
OrigOwnerNm =	The name of the original owner of the OPR.
NOIdType =	A flag indicating whether NewOwnerNm is a context path name or a LOID.
NewOwnerNm =	The name of the new owner of the OPR.

Return Values:

The LOID of the OPR’s owner.

Error Return:

NULL

User Responsibility:

The user is responsible for "free"ing the memory returned by this function.

1.9 Class object

The following section contains a set of functions which interface directly with Class objects.

int
AddImplementation(*LegionIdType CIdType, char *ClassName, LegionIdType IIdType, char *ImplName, int Arch*);

Adds a new implementation of a class binary to the given class.

Parameters:

CIdType =	A flag indicating whether ClassName is a LOID or a context path.
ClassName =	The name of the class to add the implementation to.
IIdType =	A flag indicating whether ImplName is a LOID or a context path.
ImplName =	The name of an implementation object to add.
Arch =	The architecture for which the implimentation is valid.

Return Values:

1 on Success
0 on Failure

User Responsibility: None

int
DeleteInstance(*LegionIdType CIdType, char *ClassName, LegionIdType IIdType, char *InstanceNm*);

Asks a class to delete one of its instances.

Parameters:

CIdType =	A flag indicating whether ClassName is a LOID or a context path.
ClassName =	The name of the class to delete the instance from.
IIdType =	A flag indicating whether InstanceNm is a LOID or a context path.
InstanceNm =	The name of an object instance to delete.

Return Values:

1 on Success
0 on Failure

User Responsibility: None

int
ActivateInstance(*LegionIdType CIdType, char *ClassName, CInstancePlacementInfo CInfo*);

Asks a class to activate one of its instances.

Parameters:

CIdType =	A flag indicating whether ClassName is a LOID or a context path.
ClassName =	The name of the class to activate the instance for.

CInfo = A CInstancePlacementInfo variable indicating which instance to activate and where.

Return Values:
1 on Success
0 on Failure

User Responsibility: None

int
DeactivateInstance(*LegionIdType CIdType, char *ClassName, LegionIdType OIdType, char *ObjName*);
Asks a class to deactivate one of its instances.

Parameters:
CIdType = A flag indicating whether ClassName is a LOID or a context path.
ClassName = The name of the class to deactivate the instance for.
OIdType = A flag indicating whether ObjName is a LOID or a context path.
ObjName = A LOID or context path indicating which object to deactivate.

Return Values:
1 on Success
0 on Failure

User Responsibility: None

int
ForcedDeactivateInstance(
*LegionIdType CIdType, char *ClassName, LegionIdType OIdType, char *ObjName*);
Asks a class to deactivate one of its instances. If the object is in the going-up state or in the going-down state, this call will fail.

Parameters:
CIdType = A flag indicating whether ClassName is a LOID or a context path.
ClassName = The name of the class whose instance is to be deactivated.
OIdType = A flag indicating whether ObjName is a LOID or a context path.
ObjName = A LOID or context path indicating which

object to deactivate.

Return Values:
1 on Success
0 on Failure

User Responsibility:
None

CUVaL_InstanceRecord*
GetInstanceList(*LegionIdType CIdType, char *ClassName*);
Asks a class to return a list of all of the instances that it is currently managing.

Parameters:
CIdType = A flag indicating whether ClassName is a LOID or a context path.
ClassName = The name of the class to retrieve the list from.

Return Values:
A NULL terminated list of CUVaL_InstanceRecords where each element in the array is the UVaL_InstanceRecord for one of the objects that the class is managing.

Error Return:
NULL

User Responsibility:
The user is responsible for destroying each CUVaL_InstanceRecord in the list, as well as for "free"ing the list itself.

1.10 Implicit parameters

The following section contains the functions which allow a program to manipulate its own method and message implicit parameters.

int
MethodImplicitParm_InsertLOID(*char *ParmName, char *Loid*);
Insert a LOID implicit parameter into the Implicit method parameters

Parameters:

ParmName = The name of the implicit parameter to which the action is being applied.
Loid = The string representation of the LOID to add.

Return Values:
0 on Failure
1 on Success

User Responsibility: None

int
MethodImplicitParm_InsertInt(*char *ParmName, int Value*);
Insert an int implicit parameter into the implicit method parameters

Parameters:
ParmName = The name of the implicit parameter to which the action is being applied.
Value = The integer value to be placed into the parameters.

Return Values:
0 on Failure
1 on Success

User Responsibility: None

int
MethodImplicitParm_InsertString(*char *ParmName, char *Str*);
Insert a new string implicit parameter into the implicit method parameter database.

Parameters:
ParmName = The name of the implicit parameter to which the action is being applied.
Str = The new string to add to the database.

Return Values:
0 on Failure
1 on Success

User Responsibility: None

int
MethodImplicitParm_Remove(*char *ParmName*);
Remove an implicit parameter from the implicit method

parameters database.

Parameters:
ParmName = The name of the implicit parameter to which the action is being applied.

Return Values:
0 on Failure
1 on Success

User Responsibility: None

char*
MethodImplicitParm_FindLOID(*char *ParmName*);
Find an implicit parameter in the implicit method parameters database (this parameter must be a LOID parameter).

Parameters:
ParmName = The name of the implicit parameter to which the action is being applied.

Return Values:
The string representation of the LOID that was found.

Error Return:
NULL

User Responsibility:
The user is responsible for "free"ing the returned LOID when done.

int
MethodImplicitParm_FindInt(*char *ParmName*);
Find an implicit parameter in the implicit method parameters database (this parm must be an int parameter).

Parameters:
ParmName = The name of the implicit parameter to which the action is being applied.

Return Values:
The integer parameter that was retrieved from the database.

Error Return:
Need to check LegionErrno for this one.

User Responsibility: None

char*
MethodImplicitParm_FindString(*char *ParmName*);

Retrieve a string parameter from the implicit method parameter database.

Parameters:

ParmName = The name of the implicit parameter to which the action is being applied.

Return Values:

The string parameter retrieved from the implicit method database.

Error Return:

NULL

User Responsibility:

The user is responsible for "free"ing the memory returned from this function call.

int
MethodImplicitParm_ReplaceLOI (*char *ParmName, char *Loid*);

Replace an implicit method parameter in the database with another (LOID) parameter.

Parameters:

ParmName = The name of the implicit parameter to which the action is being applied.

Loid = The LOID parameter to replace `<ParmName>` with.

Return Values:

0 on Failure
1 on Success

User Responsibility: None

int
MethodImplicitParm_ReplaceInt(*char *ParmName, int Value*);

Replace an implicit method parameter in the database with another (Int) parameter.

Parameters:

ParmName = The name of the implicit parameter to which the action is being applied.

Value = The value of the integer parameter to replace `<ParmName>` with.

Return Values:

0 on Failure

1 on Success

User Responsibility: None

int
MethodImplicitParm_ReplaceString(*char *ParmName, char *Str*);
Replace an implicit method parameter in the database with another (string) parameter.

Parameters:
ParmName = The name of the implicit parameter to which the action is being applied.
Str = The string parameter which the user wishes to replace <ParmName> with.
Return Values:
0 on Failure
1 on Success

User Responsibility: None

int
MessageImplicitParm_InsertLOID(*char *ParmName, char *Loid*);
Insert a LOID implicit parameter into the implicit message parameters

Parameters:
ParmName = The name of the implicit parameter to which the action is being applied.
Loid The string representation of the LOID to add

Return Values:
0 on Failure
1 on Success

User Responsibility:
None

int
MessageImplicitParm_InsertInt(*char *ParmName, int Value*);
Insert an int implicit parameter into the implicit message parameters.

Parameters:
ParmName = The name of the implicit parameter to which the action is being applied.
Value = The integer value to be placed into the parameters.

Return Values:
0 on Failure

1 on Success

User Responsibility: None

int
MessageImplicitParm_InsertString(*char *ParmName, char *Str*);
Insert a new string implicit parameter into the implicit message parameter database.

Parameters:
ParmName = The name of the implicit parameter to which the action is being applied.
Str = The new string to add to the database.

Return Values:
0 on Failure
1 on Success

User Responsibility: None

int
MessageImplicitParm_Remove(*char *ParmName*);
Remove an implicit parameter from the implicit message parameters database.

Parameters:
ParmName = The name of the implicit parameter to which the action is being applied.

Return Values:
0 on Failure
1 on Success

User Responsibility: None

char*
MessageImplicitParm_FindLOID(*char *ParmName*);
Find an implicit parameter in the implicit message parameters database (this parm must be a LOID parameter).

Parameters:
ParmName = The name of the implicit parameter to which the action is being applied.

Return Values:
The string representation of the LOID that was found.

Error Return:
NULL

User Responsibility:
The user is responsible for "free"ing the returned LOID when done.

int
MessageImplicitParm_FindInt(*char *ParmName*);
Find an implicit parameter in the implicit message parameters database (this parameter must be an int parameter).

Parameters:
ParmName = The name of the implicit parameter to which the action is being applied.

Return Values:
The integer parameter that was retrieved from the database.

Error Return:
Need to check LegionErrno for this one.

User Responsibility: None

char*
MessageImplicitParm_FindString(*char *ParmName*);
Retrieve a string parameter from the implicit message parameter database.

Parameters:
ParmName = The name of the implicit parameter to which the action is being applied.

Return Values:
The string parameter retrieved from the implicit message database.

Error Return:
NULL

User Responsibility:
The user is responsible for "free"ing the memory returned from this function call.

int
MessageImplicitParm_ReplaceLOID(*char *ParmName, char *Loid*);
Replace an implicit message parameter in the database with another (LOID) parameter.

Parameters:
ParmName = The name of the implicit parameter to which the action is being applied.

Loid = The LOID parameter to replace <ParmName> with.

Return Values:
0 on Failure
1 on Success

User Responsibility: None

int
MessageImplicitParm_ReplaceInt(*char *ParmName, int Value*);
Replace an implicit message parameter in the database with another (Int) parameter.

Parameters:
ParmName = The name of the implicit parameter to which the action is being applied.
Value = The value of the integer parameter to replace <ParmName> with.

Return Values:
0 on Failure
1 on Success

User Responsibility: None

int
MessageImplicitParm_ReplaceString(*char *ParmName, char *Str*);
Replace an implicit message parameter in the database with another (string) parameter.

Parameters:
ParmName = The name of the implicit parameter to which the action is being applied.
Str = The string parameter which the user wishes to replace <ParmName> with.

Return Values:
0 on Failure
1 on Success

User Responsibility: None

1.11 Collections

The following section contains the functions which allow a program to manipulate, update, and query Legion Collection objects.

int
JoinCollection(
LegionIdType CollectionIdType, char CollectionName,*
*LegionIdType JoinerIdType, char *JoinerName*);

Join an object into a Collection

Parameters:

CollectionIDType =	A flag indicating whether CollectionName is a LOID or a context name.
CollectionName =	Either the LOID or the context path of the Collection object in question.
JoinerIDType =	A flag indicating whether the JoinerName is a LOID or a context path.
JoinerName =	Either a LOID or a context path for the object being added to the Collection.

Return Values:
0 on Failure
1 on Success

User Responsibility: None

int
JoinCollection_wData(
LegionIdType CollectionIdType, char CollectionName,*
*LegionIdType JoinerIdType, char *JoinerName,*
*char **Attributes*);

Join an object into a Collection and give init data.

Parameters:

CollectionIDType =	A flag indicating whether CollectionName is a LOID or a context name.
CollectionName =	Either the LOID or the context path of the Collection object in question.
JoinerIDType =	A flag indicating whether the JoinerName is a LOID or a context path.
JoinerName =	Either a LOID or a context path for the object being added to the Collection.
Attributes =	NULL terminated array of object attributes.

Return Values:
0 on Failure
1 on Success

User Responsibility: None

int
LeaveCollection(*LegionIdType CollectionIdType, char* CollectionName, LegionIdType LeaverIdType, char *LeaverName*);
Indicate that an object is leaving a Collection.

Parameters:

CollectionIDType =	A flag indicating whether CollectionName is a LOID or a context name.
CollectionName =	Either the LOID or the context path of the Collection object in question.
LeaverIDType =	A flag indicating whether the LeaverName is a LOID or a context path.
LeaverName =	Either a LOID or a context path for the object leaving the Collection.

Return Values:
0 on Failure
1 on Success

User Responsibility: None

int
UpdateCollectionEntry(
LegionIdType CollectionIdType, char CollectionName, LegionIdType ConstituentIdType, char *ConstituentName*, *char **Attributes*);
Update a Collection entry's data.

Parameters:

CollectionIDType =	A flag indicating whether CollectionName is a LOID or a context name.
CollectionName =	Either the LOID or the context path of the Collection object in question.
ConstituentIDType =	A flag indicating whether the ConstituentName is a LOID or a context path.
ConstituentName =	Either a LOID or a context path for the object whose collection data is being updated.
Attributes =	NULL terminated array of object attributes.

Return Values:
0 on Failure
1 on Success

User Responsibility: None

CLegionCollectionData*
QueryCollection(*LegionIdType CollectionIdType, char* CollectionName, char *QueryString*);

Parameters:

CollectionIDType =	A flag indicating whether CollectionName is a LOID or a context name.
CollectionName =	Either the LOID or the context path of the Collection object in question.
QueryString =	A MESSIAHS-style query string

Return Values:
A NULL terminated list of CollectionData entries

Error Returns:
Returns NULL on failure

User Responsibility:
User is responsible for destroying the individual elements returned, as well as for "free"ing up the array.

```
#ifdef_cplusplus
}
#endif

#endif
```

2.0 Error messages

The CError.h file contains the declarations for the error reporting mechanism used in the C Library interface to the Legion system. Below are the identifiers and their codes.

```
#define LEGION_NO_ERROR                     0
#define LEGION_NOT_A_LOID                   1
#define LEGION_NOT_IN_CONTEXT_SPACE         2
#define LEGION_UNKNOWN_ERROR                3
#define LEGION_CONTEXT_SPACE_INVALID        4
#define LEGION_ROOT_CONTEXT_ALREADY_EXISTS  5
#define LEGION_CONTEXT_ALREADY_EXISTS       6
#define LEGION_NO_CONTEXT_CLASS             7
#define LEGION_CONTEXT_CREATE_FAILED        8
#define LEGION_UNABLE_TO_ADD_PATH           9
#define LEGION_METHOD_TIMED_OUT             10
#define LEGION_BAD_ATTRIBUTE_FORMAT         11
#define LEGION_UNABLE_TO_GET_INTERFACE      12
#define LEGION_PARM_ALREADY_EXISTS          13
#define LEGION_PARM_DOES_NOT_EXIST          14
#define LEGION_PARM_WRONG_TYPE              15
#define LEGION_PARM_NO_DATA                 16
#define LEGION_PARM_UNABLE_TO_REPLACE       17
#define LEGION_NOT_A_CLASS                  18
#define LEGION_NOT_A_HOST                   19
#define LEGION_NOT_A_VAULT                  20
#define LEGION_NOT_AN_IMPL                  21
#define LOID_NOT_EXPECTED                   22
#define LOID_EXPECTED                       23
#define LEGION_PARM_TYPE_UNKNOWN            24
#define LEGION_ OUTCALL_FAILED              25
#define LEGION_OUTCALL_PERMISSION_DENIED    26
#define LEGION_CLASS_NOT_FOUND              27
#define LEGION_BAD_ARGUMENT                 28
#define LEGION_OBJECT_TYPE                  29
#define LEGION_NO_CURRENT_CONTEXT           30
#define LEGION_OBJECT_NOT_FOUND             31
#define LEGION_INTERNAL_ERROR               32
#define LEGION_EMPTY_LOID                   33
```

Index

C

G