Legion 1.8

Basic User 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

Before you start

Getting started

Context space

Before you start

1.0 Introduction

1.1 About this manual

This manual is an introduction to Legion 1.8. It gives instructions for logging in to an active Legion system and a high-level introduction to Legion context space. Users who need to install or start a Legion system should consult the System Administrator Manual. If you need more help please contact us (page 109).

There are four Legion manuals, each aimed at a specific type of user, that can be consulted for more information. The others are:

- **System Administrator Manual**: information and documentation for administrators of Legion systems, including installing and running Legion, configuring security features, resource management, and managing a Legion system.
- **Developer Manual**: information and documentation for programmers working in Legion, and includes information on languages, libraries, core objects, and implementing new Legion objects.
- **Reference Manual**: detailed information about specific elements of the Legion system.

There are also man pages for all Legion commands included with the system files. To view a man page, type the following on the command line:

```
$ man <command name>
```

There are on-line tutorials on the Legion web site (http://legion.virginia.edu).

This manual assumes that you are working on a previously installed, compiled, and running system.[1] Before going any further, be sure that the system is properly installed and running. Check with your system administrator if you are unsure.

1 It is actually not always necessary to have a Legion system running in order to use Legion: some Legion hosts can run as client hosts in a larger Legion system. A client host does not have the full set of Legion system binaries, but a set of binary executable files that can contact other Legion hosts in its Legion system. Please e-mail us at <legion-help@virginia.edu> for more information.

1.2 Style conventions

The manuals at times refer to path names in Unix directory space and in Legion context space. The following style conventions are used:

- Unix, DOS, and local path names appear in a serif typeface.
- Functions, method names, parameters, flags, command-line utilities (such as `rm`, `cp`, and `legion_ls`), and context path names appear in `fixed typeface`.

1.3 About Legion

Developed at the University of Virginia, Legion is a grid operating system. While fully supporting existing codes written in MPI and PVM, Legion provides features and services that allow users to take advantage of much larger, more complex resource pools. A user can easily run a computation on a supercomputer at a national center while dynamically visualizing the results on a local machine, or schedule and run a large parameter space study on several workstation farms simultaneously. For example, computational scientists can use cycles wherever they are, allowing bigger jobs to run in shorter times through higher degrees of parallelization.

Key capabilities include the following:

- *Legion eliminates the need to move and install binaries manually on multiple platforms.* After Legion schedules a set of tasks over multiple remote machines, it automatically transfers the appropriate binaries to each host. A single job can run on multiple heterogeneous architectures simultaneously; Legion will ensure that the right binaries go to each, and that it only schedules onto architectures for which it has binaries.
- *Legion provides a virtual file system that spans all the machines in a Legion system.* Input and output files can be seen by all the parts of a computation, even when the computation is split over multiple machines that don't share a common file system. Different users can also use the virtual file system to collaborate, sharing data files and even accessing the same running computations.
- *Legion's object-based architecture dramatically simplifies building add-on tools* for tasks such as visualization, application steering, load monitoring, and job migration.
- *Legion provides optional privacy and integrity of communications for applications distributed over public networks.* Multiple users in a Legion system are protected from one another.

These features also make Legion attractive to administrators looking for ways to increase and simplify the use of shared high-performance machines. The Legion implementation emphasizes extensibility, and multiple policies for resource use can be embedded in a single Legion system that spans multiple resources or even administrative domains.

Getting started

2.0 Setting up and logging in

You must set up the proper environment variables before starting. You may also need to log in to a Legion system as a Legion user.

2.1 Preparing your Legion environment

Depending on how your system is set up, you may need to set up your access to your Legion system. This will probably involve running a script such as this:

```
$ . ~legion/setup.sh
```

or

```
$ source ~legion/setup.csh
```

The exact syntax will depend on what kind of shell you are using and where your Legion files are installed. Your system may have different requirements: ask your system administrator.

2.2 Logging in

If your system administrator has enabled Legion's security features, you must have a user id and log in to Legion before you can start working. Ask your system administrator to create one for you if necessary.

Your system's log in procedure may differ from what is described here, so check with your system administrator for specific instructions. The default system requires all users to have user ids and passwords. This lets Legion keep track of your objects and user privileges. It also prevents malicious users from interfering with your objects or gaining illicit access to your system.

When you log in to a Legion system you are identified by an *AuthenticationObject*, a special object that contains your password, initial implicit parameters (the Legion equivalent of a Unix "environment"), and other information. AuthenticationObjects are created when you create a user id. A set of Legion commands can be used to retrieve or change this information (see section 2.8 on page 62 in the Legion Reference Manual for these commands). When an authenticated user runs a Legion process a certificate confirming his or her identify is passed along to verify that this person has permission

to run the process. This certificate is created and signed by your AuthenticationObject, so you will have to get a new user id if your AuthenticationObject is destroyed.

2.2.1 Logging in as a user

You login with the `legion_login` command. You will be prompted for your password. *You must use the full path name for your user id* (e.g., `/users/<user id>`).

To login, run `legion_login`:

```
$ legion_login /users/bob
Password: xxxx
$
```

Or

```
$ legion_login
Legion login: /users/bob
Password: xxxx
$
```

[2]

Your current working context is automatically set to your home context (`/home/<user id>`). Use `legion_cd` to move to another part of context space (see section 5.1.4).

The `legion_login` command verifies your identity and your security privileges and puts a credentials file in your local /tmp directory. This file is a user read-only file. It is used by command-line utilities to verify your identity. You get a separate credentials file in each shell in which you run `legion_login`. You own objects you create while logged in. No one else (except the system administrator) can use them unless you specifically give them permission (see "About object permissions" on page 11). Any processes that you start after logging in will be accompanied by a copy of your AuthenticationObject's certificate.

2.2.2 Changing your password

While you are logged in, you can use `legion_passwd` to change your password. You must use the whole user id path. You will be prompted for your old and new passwords.

```
$ legion_passwd /users/nemo
New Legion password: xxxx
Retype new password: xxxx
Password changed.
$
```

2 If you wish, you can include your password on the command line. E.g.,

```
$ legion_login /users/bob -p bobspassword
```

This is not a secure method of logging in, though, and we don't recommend it.

2.2.3 About object permissions

If your system administrator has enabled Legion security, the objects that you create while logged in cannot be used by any other users. If you wish to share your objects you need to give other users any necessary read, write, and execute permissions.[3] You can use `legion_change_permissions` to change an object's permissions. The syntax is:

```
legion_change_permissions [+-rwx] [-v]
   <group/user context path>
   <target context path>
   [-debug] [-help]
```

Use the `r`, `w`, or `x` flags to add (`+`) or remove (`-`) read, write, and execute permissions on objects. For example, if you wanted to allow `bob` to read your object `foo`, you would enter:

```
$ legion_change_permissions +r /users/bob foo
```

See page 64 in the Reference Manual for more information.

2.2.4 Checking your log in status

If you want to check whether or not you are logged in or verify your current user id, use `legion_whoami`. If you are logged in, it will return your current user id:

```
$ legion_whoami
/users/nemo
$
```

If you are not logged in the command will return `<Unknown>`.

2.2.5 Logging out

To log out, run `legion_logout`. You don't need your user id path.

```
$ legion_logout
```

This will remove your credentials file. Remember that you are not in a subshell, so if you type `exit` you will close your current shell.

3 This command can only be used with common Legion object types: context, file, class, tty, implementation, host, and vault objects.

2.2.6 Using Legion in a Kerberos environment

If your site requires you to authenticate via Kerberos in order to log on or otherwise interact with any of the machines at the site (irrespective of Legion), you need to incorporate your Kerberos credentials into the Legion environment. If Legion needs to create a process or a file on your behalf on a remote machine, your Kerberos credentials must be available to authenticate you to the remote machine. To do this you will need to create a Legion proxy object that holds a copy of your Kerberos credentials. Legion can then automatically contact this proxy object whenever it needs your Kerberos credentials for a remote machine.

If you don't know whether or not you're running in a Kerberos environment, you're probably not. Check with your system administrator to confirm this.

Kerberos support in Legion is not currently fully documented; a small collection of sites that use Kerberos are working with the Legion developers to define and improve Kerberos support in Legion. In the near future, we will include detailed instructions regarding the creation and use of the Kerberos proxy objects. If you require Kerberos support in Legion, contact us at <legion-help@virginia.edu>.

2.3 Changing your profile

You can use `legion_configure_profile` to edit information about your user profile, security settings, and fault tolerance settings. It's a menu-driven command, so just enter it on the command line:

```
$ legion_configure_profile
```

and choose your options. Hit the `<enter>` key to return to the previous menu level. This command carries out three functions:

1. Edit your AuthenticationObject to include contact information
2. Change your security preferences for the message layer
3. Edit your fault tolerance settings for SKCC objects

The first option lets you provide your e-mail address, name, and company. This can be useful if you are using Legion on-line. It also let us contact you more easily.

The second option lets you set the security mode for the Legion message layer. There are three settings: private, protected, and off. We ***strongly*** recommend that you read the discussion of the message layer and these security settings before using this option (page 30 in the System Administrator Manual). Any changes you make will be applied to all messages passed by your objects.

The third option lets you edit the settings for your SKCC classes and backup vaults. This will override any settings you have previously made. It uses the following commands to edit your settings:

legion_skcc_set_class_vaults
legion_skcc_set_defaults
legion_set_backup_vaults
legion_class_vault_list

Please see the man pages or Reference Manual for information on these commands. Please see page 32 for more information about SKCC classes and backup vaults.

Context space

3.0 An introduction to context space

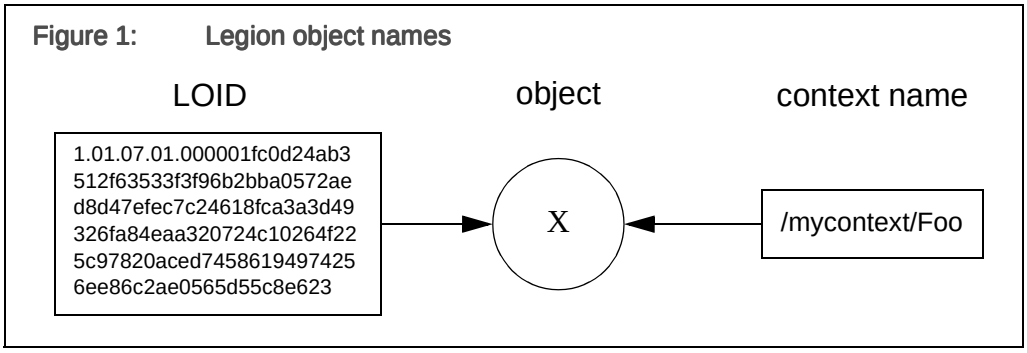
Legion is an object-based system organized by classes and metaclasses (classes of classes). All Legion classes are objects. Legion objects are organized by their classes and the classes are in turn organized into a hierarchical structures of *metaclasses*. The top metaclass is the `LegionClass`, which is responsible for every object in the system.

Every element of Legion is represented by a Legion object. You work in Legion by controlling the desired resource's representative object. This means that you must have an name for the object that you wish to use. An object can have two kinds of names: a system-level LOID or a user-assigned context name (Figure 1).

All Legion objects are automatically assigned system-level names called *Legion Object Identifiers* (LOIDs). You can use LOIDs from the command line, but they are, at best, impractical and awkward (see "About the LOID," pg. 17).

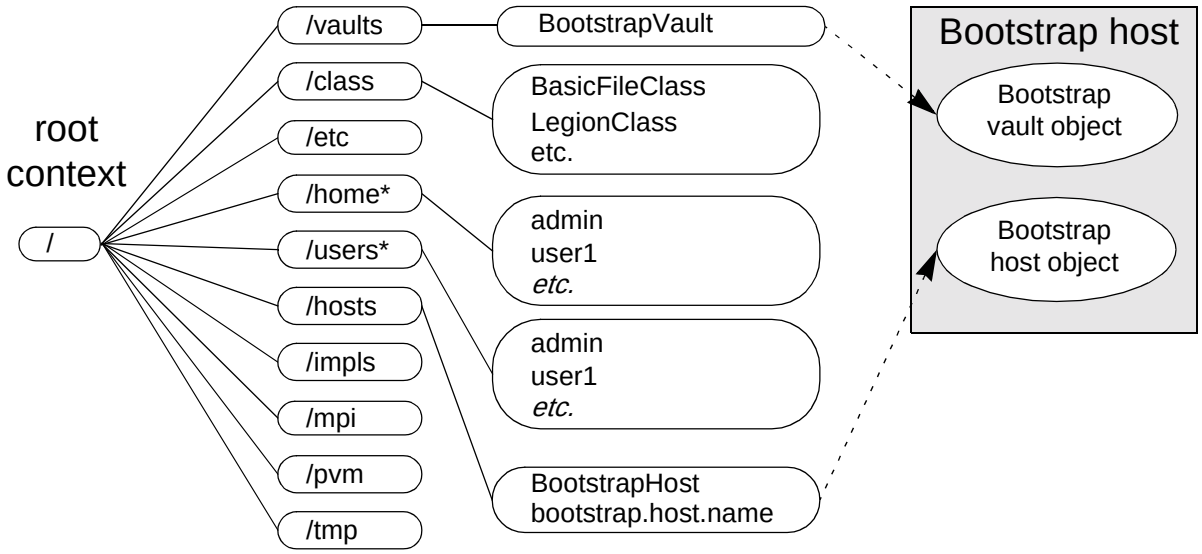
Or, an object can have a *context name*. A context name is a string name that you assign to a Legion object. You can assign multiple names to a single object. An object can be assigned different names by different users. Context names are organized into a hierarchy of *contexts* in Legion's *context space*. Contexts are organized in sets of subcontexts, just as Unix directories are organized in sets of subdirectories.

Figure 1: Legion object names



Objects are not automatically given context names: new Legion systems provide a starter context space with names for existing objects. When you create new objects, you can assign them context names. You can run programs, use far-flung resources, and take advantage of Legion's other features through context space. Figure 2, below, shows a typical new Legion system's context space. You will probably be given a portion of your system's particular context space as your personal space.

Figure 2: Contents of a typical new Legion system context space



* If your system administrator has enabled Legion security, the `/home` and `/users` contexts will have objects representing each user in the system. The `admin` is the system administrator in a secure Legion system. The `admin` holds certain privileges on Legion objects. If you log in with `legion_login` you will automatically start out in your `/home/<user id>` context, If Legion security is not enabled and no users have been created, these contexts will be empty. Please see page 30 in the System Administrator Manual for more information about Legion security.

A typical new system includes a context hierarchy and context names for commonly used objects. This provides context paths for key objects, such as classes and implementations. These objects are in the `/hosts`, `/vaults`, `/impl`, and `/class` contexts. I.e., class objects are in the `/class` context and implementation objects are in the `/impl` context.

Context space is not related to physical space or object types, since context names are assigned and organized according to users' needs, not object's physical location or description. Objects in the same context do not have be able to communicate with each other or even know of each other's existence. They do not have to be the same type of objects. A context can hold multiple subcontexts, just as Unix directories hold sets of subdirectories.

Context path names use Unix-style slash ("/") by default, and the examples here follow that style (e.g., the full context path name for the bootstrap host in the system shown in Figure 2 is `/hosts/BootstrapHost`).[4]

You can see a working context space in the npacinet browser at <http://sirius.cs.virginia.edu/browser> as a guest user or, if you have

4 You can change to the DOS-style backslash ("\") via the primitive context manipulation routines provided by the Legion library.

a npacinet account, under your account name (npacinet is our testbed: see <http://legion.virginia.edu/npacinet.html>).

Section 4.0, starting on page 17, provides more detail on Legion naming and context space and section 5.0, starting on page 20, discusses command-line tools for working in context space.

4.0 Context space

4.1 Legion object names

As discussed in section 3.0 (page 14), all Legion objects are automatically assigned a unique system-level identifying name called a Legion Object Identifier (LOID). The LOID enables Legion to manipulate Legion objects scattered on different machines. But a LOID consists of several lines of numbers (shown below), making it difficult to figure out which LOID belongs to which object. From the user's perspective it is much easier to give an object a context (string) name, such as "george" or "myHost". This method of allowing objects a system-level and a user-level name allows LOIDs to hold system-level information, such as public keys, without polluting the user-level name-space.

Moreover, each user's context space is independent of everyone else's context space. So, multiple users can assign an object a different context name to suit his or her own convenience. This is discussed further in section 4.3.

4.2 About the LOID

A LOID is a variable length binary structure, typically more than 128 bytes, which contains a type field followed by several variable size fields. A LOID may be appended with any number of other fields, each of whose size and meaning are dependent on the LOID type. LOIDs are represented as strings, using standard "dotted hex" notation. A typical LOID looks like this:

```
1.01.07.01.000001fc0d24ab3512f63533f3f96b2bba
0572aed8d47efec7c24618fca3a3d49326fa84eaa3207
24c10264f225c97820aced74586194974256ee86c2ae0
565d55c8e623
```

The groups of numbers, separated by period, show the object's characteristics. First, the type field, **1**, indicates the LOID's type. Next, the first variable field, **01**, indicates the domain in which the object was created. The second variable field, **07**, indicates which class that the object belongs to. The third variable field, **01**, indicates the object's instance number. The fourth variable field (the last field), **000001fc0d24ab...**, is the object's RSA public key. This notation is obviously cumbersome, so we expect that users will prefer context space.

4.3 Organizing context space

All contexts in Legion are organized into a Unix-like hierarchy, with a root context that contains a set of other contexts. This structure provides each object with an absolute path name (i.e., `/mycontext/myobject`). Just as a Unix user organizes files in a series of directories, a Legion user organizes objects in a series of contexts. Each user has a personal context that can be used like a home directory. It will probably be something like `/home/<your Legion id>`. Ask your system administrator.

Command-line tools let you move between contexts, just as Unix commands let Unix users move between directories. Legion provides a library interface and utility commands such as `legion_ls` and `legion_rm`. The library interface provides a Unix-like path interface for expressing navigation through multiple contexts. Paths are delimited by the "/" character. For example, the path `/home/fred/foo` would map to `/home`, then `/fred`, then to `foo`.

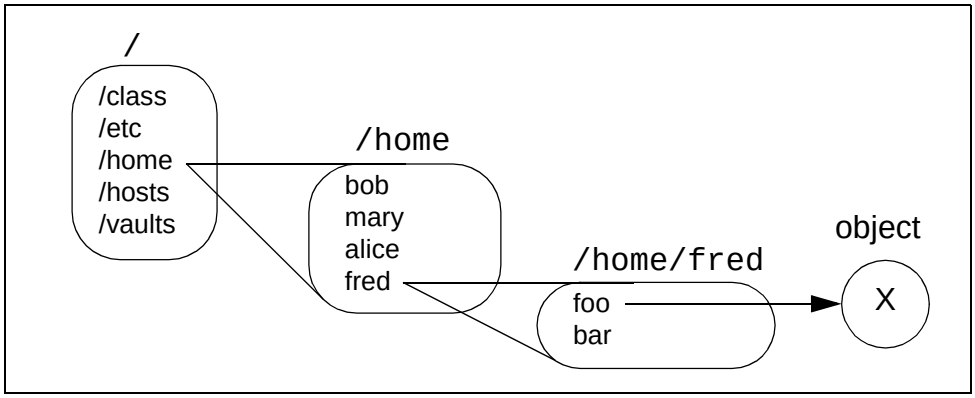


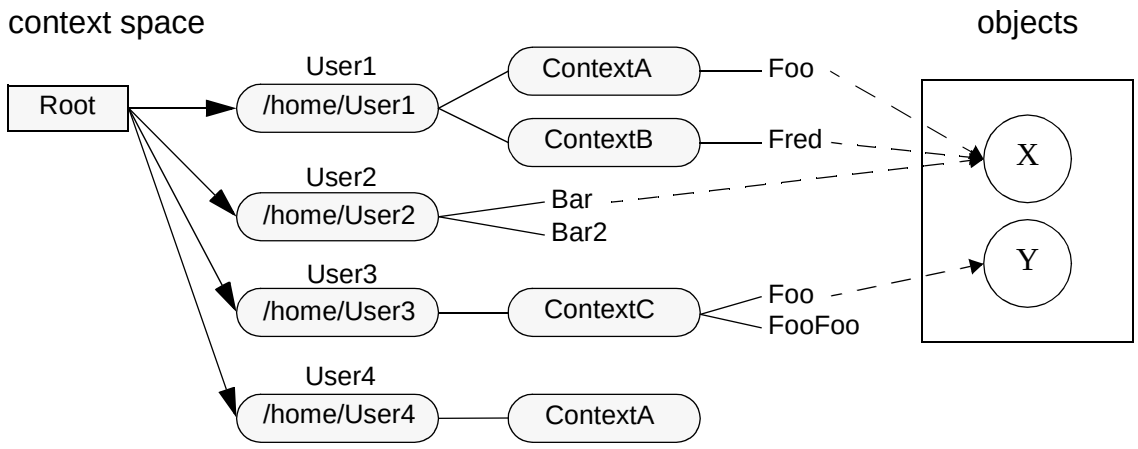
Figure 3: Context path mapping

The name `foo` points to an object somewhere in Legion: in Figure 3 it points to object X. Context space does not reflect or influence an object's physical location. Object X may be in the same building as Fred or 500 miles away. When Fred assigned it the context path name `/home/fred/foo`, the object was not moved or copied. Objects in the same context path may actually be located on different machines in different countries.

A sample context space is shown in Figure 4, below. Notice that each user has his own context space. Each context is a separate name space, so a name can be reused. In Figure 4 User1 and User4 both have subcontexts called `ContextA`.

A name can be used by more than one user to refer to the same or different Legion objects. In Figure 4, `/home/User1/ContextA` and `/home/User3/ContextC/ContextC` hold the name `Foo` but each points to different objects. Conversely, `/home/User1/ContextA/Foo`, `/home/User1/ContextB/Fred`, and `/home/User2/Bar` all point to object X.

Figure 4: Context names referring to objects



This example shows four different context paths pointing to two different objects. Three of the paths point to object X: `home/User1/ContextA/Foo` and `home/User1/ContextB/Fred` are in User1's context space, and `/home/User2/Bar` is in User2's context space. The fourth path, `/home/User3/ContextC/Foo`, points to object Y.

You can share your context space with other Legion users, just as Unix users can work in one another's directories. You can also protect individual objects or groups of objects against unauthorized use by a specific user/object or a group.

Beyond the user-level grouping of individual context spaces, you can link object groups in graph- or directory-based structure. Just as a Unix directory tree or collection of hypertext documents provides a logical means to organize information, context space provides a structure for finding and organizing Legion objects. For example, to find a host object running on an IBM SP2, you might logically start with the `/hosts` context, then look in `/ibm`, then `/sp2`, where a list of host objects running on SP2s could be found. This is like moving through a set of directories to get to a /hosts/ibm/sp2 directory.

Objects contain in their state a root context (implicitly named `/`) that provides each object with a base for absolute path names. The root context's LOID is stored in a variable that can altered by the object at runtime. Objects also contain in their state the current context's LOID, which provides the basis for relative context paths.

5.0 Working in context space

Legion has an assortment of context space command-line utility tools, and this section introduces some of the more useful ones for context space. You'll notice that many resemble Unix commands in both name and purpose, which should make Unix users feel more comfortable. Your local file path will not change as you move through context space. Full path syntax can be used as well as relative path syntax in all Legion commands

There are other sources of information for Legion commands:

- An alphabetical list of commands (section A-1.0, page 78).
- A subject list of commands (section A-1.0, page 93).
- The Reference Manual has usage and descriptions of all command-line tools.
- All Legion packages have man pages for their commands. Man page usage is:[5]

  $ man <command tool>
- The Legion web site has on-line tutorials and a quick guide to all commands (<http://legion.virginia.edu/documentation.html>)
- Finally, most commands have a `-help` flag which prints the command's full usage:

  $ <command tool> -help

  If this doesn't work, try just entering the command with no flags.

Please note that the sample outputs in this manual may not match your own. Your system administrator may have customized your system's context space or security considerations may limit your privileges in certain contexts. If you think there's something wrong, please consult your system administrator.

5 Ask your system administrator if you are unsure about using man pages on your system. You can also go to <http://legion.virginia.edu/legion_man.html> to see on-line versions of the Legion man pages.

5.1 Viewing contexts

5.1.1 Read a context's contents

Just as the Unix `ls` command lists files in a directory, the `legion_ls` command shows all objects contained in a context. You will see something like this:

```
$ legion_ls
Bar
ClassFoo
Foo
SomeObject
$
```

A context can hold classes, runnable objects, text objects, subcontexts, and any other Legion objects. A new Legion system contains a default context space with context names for existing objects. Please see page 14 for information about the default contents of context space.

Each name listed in a context refers to an object. Use `-l` to see if the object is a context, a class, or a file object.[6] The `-l` flag prints object type and description. Objects of unknown type are described as `(object)` and faulty objects are described as `(not available)`.

```
$ legion_ls -la
Bar             (context)
ClassFoo        (class)
Foo             43 bytes
SomeObject      (object)
$
```

You can see the contents of a subcontext by using it as a parameter:

```
$ legion_ls /Bar
Object1
Object2
$
```

You can use an object's path as an argument:

```
$ legion_ls -l /Bar/Object1
/bar/Object1  (object)
$
```

6 There are several flags for this command, not all of which will be discussed here, Please see page 32 in the Reference Manual for a complete list.

You can also use wildcards to look for a naming pattern. For example, to look for context names containing “foo” you would enter:

```
$ legion_ls \*foo\*
```

Please note that you must escape the “*” character.

5.1.2 Look up LOIDs

If you know an object context name and need to know its LOID, you can using `legion_ls -L`. This flag lists LOIDs associated with the specified object(s). The output in both cases will be a “dotted hex” LOID.[7] You can see the LOIDs of all objects in a context:

```
$ legion_ls -L
Fooclass 1.35fe7c3d.07..000001fc0a7838d45ed
Foofile 1.01.66000000.02000000.00000...
Fooobject 1.35fe7c3d.66000000.01000000.00
$
```

Or look up an individual object’s LOID by using its context name as a parameter.

```
$ legion_ls -L /Bar/Object1
1.01.07.01000000.000001fc0a8e60138a9...
$
```

5.1.3 Create a new context

You can create new contexts with the `legion_context_create` command. It is similar to the Unix `mkdir` command. The new context will be placed in your current context if you do not provide a full path.

```
$ legion_context_create /tmp
```

5.1.4 Change your current context

The `legion_cd` command will change your current working context (it is similar to the Unix `cd` and `chdir` commands). It sets an environment variable in the shell in which it has been invoked.

The example below sets `/tmp` as the current working context.

```
$ legion_cd /tmp
```

7 In the interests of space and readability, in this and some of the following examples the entire LOID is not reproduced.

You can check which context you are in with `legion_pwd` (below). Note that if you logged in with `legion_login` you are automatically moved to your `/home/<user id>` context.

5.1.5 Check your current context

Use `legion_pwd` (similar to Unix pwd) to check your current working context.

```
$ legion_pwd
/tmp
$
```

5.2 Naming objects

5.2.1 Assign a context name to a LOID

The `legion_context_add` command maps a new context name to a LOID. It can be used to name a previously unnamed object or to assign an extra name to a previously named object. Either way, you need to use the object's full LOID as a parameter. In the example below, the name `neo` is assigned to a LOID.

```
$ legion_context_add 1.3933cb3f.07.01000000.000001fc0a
94f51928f25f57690d7e47fc3f93ff1dc8f5616f1e606def8bb44
b0538a45ffc07f9e1ae612f455e5dbd6cb6bade7105eeebf2c7f0
6179b6bbb8826cfdf90f neo
```

5.2.2 Give an object extra context names

You can give an object extra context names with `legion_ln` (similar to the Unix `ln`). The object must already have a context name (if it doesn't, use `legion_context_add`, above, to give it one).

You can place the new name anywhere in context space (anywhere you have permission to use), regardless of where the object's existing name(s) are placed. In the example below the object's existing context name is `/vaults/BootstrapVault`. We will give it an additional name, `Foo`. The new name goes in the current context:

```
$ legion_ln /vaults/BootstrapVault Foo

$ legion_ls
Foo
$
```

The two names both point to the same object and thus to the same LOID:

```
$ legion_ls -L Foo
1.01.07.01000000.000001fc0a1213a4cab754ec019
$

$ legion_ls -L /vaults/BootstrapVault
1.01.07.01000000.000001fc0a1213a4cab754ec019
$
```

Either name can be used to call on the object.

5.2.3 Change an object's context path

An object can be given a different context path with the `legion_mv` command (similar to the Unix `mv` command). For example, to change an object's path from `Foo` to `Bar` you would enter:

```
$ legion_mv Foo Bar
```

The object has the same LOID and physical location as before. The name `Foo` is automatically removed from the current context. However, any other use of the name `Foo` (whether with this or another object) will be unchanged.

5.2.4 List context names

There are two commands that list an object's context names, `legion_list_names` and `legion_ls`. Both produce a list of all context names pointing to your object. If you only know its LOID, use `legion_list_names -l`:

```
$ legion_list_names -l 1.360830b6.07.0100000...
/hosts/BootstrapVault
$
```

You may know a context name for an object but want to know if it has any other names. You can use the same command to see what (if any) other names the object has. The example below shows that `BootstrapHost` has two context names:

```
$ legion_list_names /hosts/BootstrapHost
/hosts/BootstrapHost
/hosts/my.host.DNS.name
$
```

Or, you can use `legion_ls` with the `-A` flag. However, you must use an existing context name as an argument:

```
$ legion_ls -A /hosts/BootstrapHost
/hosts/BootstrapHost
    /hosts/BootstrapHost
    /hosts/my.host.DNS.name
$
```

These commands will list all context names for the object, including those assigned by other users. Please note, though, that you may not have "read" privileges for viewing other users' contexts and you may get a security error message if the command cannot print all of the object's context paths.

5.2.5 Remove names and objects

Remove (i.e., destroy) context names and objects with `legion_rm`. Note, though, that *if there are no other names assigned to the object, it will be destroyed.* If there are other names assigned to the object, `legion_rm` will just remove the specified name(s). (Use `legion_list_names` or `legion_ls -A` to check for extra names.)

For example, if an object has the names `Foo` and `Foo2` and you remove `Foo` the object remains. However, if you then remove `Foo2` the object will be destroyed.

You can use `-v` to run `legion_rm` in a verbose setting, which will show you whether objects or just context paths are being destroyed:

```
$ legion_rm -v foo
Removing context name "foo"
$

$ legion_rm -v foo2
Removing context name "foo2"
Destroying object "foo2" (1.3759173a.690...)
$
```

You can use wildcards to remove groups of names or obects. I.e., to remove all context names starting with "Foo" you would enter:

```
$ legion_rm Foo\*
```

Please note that you must escape the "*" character. You cannot remove names or objects that you do not own.

5.2.6 Remove an entire context

To remove an entire context, use `legion_rm -r`. This recursively removes the context and all of its contents.

5.2.7 Use the same name in different contexts

As described on page 23, you can use the same context name in different context paths. For example, suppose that you give object `Foo` the extra name `Bar`:

```
$ legion_ln Foo Bar
```

The new name goes in the current context but it can still be used in any other context. The example below uses it in the `tmp` context for object `Foo2`:

```
$ legion_ln Foo2 /tmp/Bar
```

That means that the name `Bar` is used in your current context and in `/tmp` to point to two different objects. The two paths could also point to the same object:

```
$ legion_ln Foo Bar
$ legion_ln Foo /tmp/Bar
```

5.3 Moving between local files and contexts

5.3.1 Copying

Use `legion_cp` to copy the contents of a local file to a Legion BasicFileObject and vice versa. It will also copy one BasicFileObject to another. This command only works with BasicFileObjects. Note that if you copy material to an existing BasicFileObject the new material will write over the old contents.

5.3.1.1 Copy a local file to Legion

Use `-localsrc` to copy a local file into Legion. The example below copies the Unix file UnixTextFile.txt from your local directory into Legion.

```
$ legion_cp -localsrc UnixTextFile.txt newFileObject
```

Note that this will copy the Unix file's contents into to a new Legion BasicFileObject.

5.3.1.2 Copy from Legion to a local file

Use `-localdest` to copy a Legion file object to a local file. The example below copies `Foo` to UnixTextFile.txt.

```
$ legion_cp -localdest Foo UnixTextFile.txt
```

Please note that this will overwrite any previously existing files called UnixTextFile.txt.

5.3.1.3 Copy from Legion to another Legion file object

The `legion_cp` command will also copy the contents of a BasicFileObject to a new file object. Here it copies the contents of `Foo` to BasicFileObject `Foo_copy`.

```
$ legion_cp Foo Foo_copy
```

5.3.1.4 Using wildcards when copying

You can use wildcards in the source parameter to copy a group of files into Legion context space. The example below copies all local files that start with “Foo” into `/home/myContext`.

```
$ legion_cp -localsrc Foo\* /home/myContext
```

Please note that you must escape the “*” character with a “\”. You cannot use wildcards in your destination path: the new Legion objects have the same names as their local originals. So, if `Foo\*` yields `Foo1` and `Foo2`, the duplicate Legion file objects will be `Foo1` and `Foo2`.

5.3.2 Look at a file object’s contents

You can view the contents of a Legion file object with `legion_cat` (similar to Unix `cat`). The contents are printed to standard output.

```
$ legion_cat FileObject
Here are the contents of FileObject.
$
```

This only works with text object (BasicFileObjects). If you try to run it with another kind of object (such as a class object) you’ll get an error.

5.3.3 Import a local Unix tree

You can copy the contents of a local file tree from your local file space into context space with `legion_cp`. You'll need to use `-r` to copy in recursive mode and `-localsrc` to copy from your local directory. The usage is:

`legion_cp`
`-r -localsrc <Unix directory> <new context path>`

If you do not provide a full context path for the directory (e.g., `new_stuff` instead of `/home/me/new_stuff`), the new context will be placed in your current context. It will copy the structure of the directory, including creating any necessary subcontexts. The Legion copies will have the same names as your local originals.

For example, say you want to copy your local directory wrk into Legion. The wrk tree contains two subdirectories, /foo and /bar. To copy the directory in to the current context you would enter:

```
$ legion_cp -r -localsrc wrk tmp
```

Legion will create a new context in the current context, name it `tmp`, and copy the contents of wrk. It also creates two new contexts in `tmp`, called `foo` and `bar`. Legion automatically gives all copied objects the same name as the original Unix files and subdirectories.

5.3.4 File sharing: temporarily link files to Legion

You can temporarily link your local files with context space with `legion_export_dir` temporarily links a local directory tree to context space. A new context will be created to match the local directory tree, and new context names will match the local directory's file names. For example:

```
$ legion_export_dir local_directory_Foo \
  /home/myContext/Bar
```

Legion creates a new context called `/Bar`, which is linked to local_directory_Foo. Any changes that you make to `/Bar`, such as changing or removing context names, will be reflected in local_directory_Foo. To see changes you make to the local directory while the command is running, run `legion_export_dir_rehash` with the temporary context name as a parameter. I.e.:

```
$ legion_export_dir_rehash /home/myContext/Bar
```

The command executes only for as long as you wish: you can pause the command so that the new context is temporarily unavailable. Once you resume the command, the context is available. To pause

the command, press `^-C`. To resume using the exported files, re-execute the command, specifying the same local directory path and context path. If you try to use your exported files and directories while the command is paused you will get errors. The context space will not be automatically removed when the command is paused or stopped so you should also delete the temporary context with `legion_rm -r` when you are finished. I.e.:

```
$ legion_rm -r /home/myContext/Bar
```

6.0 Host and vault objects

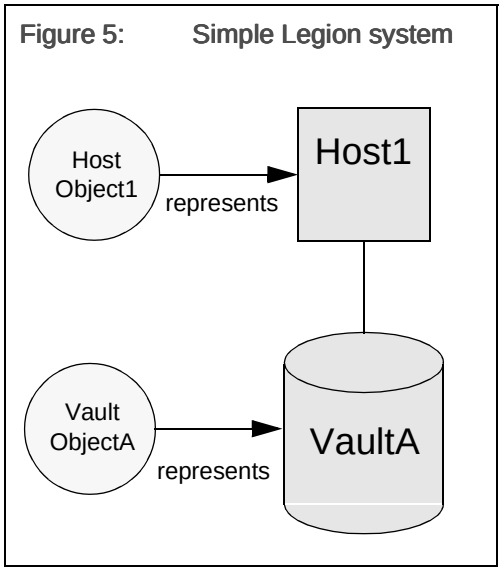
A new Legion system contains a single host object and vault object, called the *bootstrap host* and *bootstrap vault*. In order to expand the resources available to your system, however, you can add host objects and vault objects to your system.

6.1 Host/vault versus host/vault object

There is a difference between a host and a host object and between a vault and a vault object. Hosts and vaults refer to physical machines while host objects and vaults objects are Legion objects that manage hosts and vaults.

A *host* is a machine (workstation, PC, etc.) that contains at least one processor and can contain active Legion objects. A *vault* stores inactive Legion objects: it is a persistent storage space that manages the persistent storage space of inert (i.e., inactive) Legion objects and is the virtual representation of a persistent storage space on a processor. A vault can be in a file system, database system, tape drive, CD-rom, etc. A single machine can have multiple vaults.

Figure 5: Simple Legion system



A *host object* represents and can run a physical host or collection of hosts in the Legion system. The host object guards all or a portion of a host's resources, and can activate or deactivate other Legion objects. A host can have more than one host object. If there are multiple host objects, they each maintain a portion of the host's resources.

A *vault object* represents and runs the vault. Like the host object, vault objects guard a vault's resources. They do not, however, activate or deactivate the objects that they manage.

Figure 5 shows this division of labor. HostObject1 represents Host1 and VaultObjectA represents VaultA.

6.2 About the bootstrap host/vault

A new Legion system contains one host, the *bootstrap host*, and one vault, the *bootstrap vault*, as well as a *bootstrap host object* and a *bootstrap vault object* to manage them. Legion automatically assigns context paths to all new objects during the start-up procedure, and the bootstrap host and vault objects are given paths in the `/hosts` context:

```
$ legion_ls /hosts
BootstrapHost
your.bootstrap.host.name
$

$ legion_ls /vaults
BootstrapVault
$
```

While there is one context name for the vault, `BootstrapVault`, there are two for the host, `BootstrapHost` and the host's DNS name (i.e., `your.bootstrap.host.name`). You may see other host or vault objects listed in the hosts and vaults contexts, especially if your system administrator has customized your context space.

6.3 Creating objects on new hosts

You can create new objects on a new host with `legion_create_object`. The full syntax of this command is:

```
legion_create_object
  {[-c] <class context path> | -l <class LOID>}
  <context path for new object>
  [-h <host on which to place new object>]
  [-v <vault on which to place new object>]
  [-H <context path of preferred host class>]
  [-V <context path of preferred vault class>]
  [-Ch <context containing preferred hosts>]
  [-Cv <context containing preferred vaults>]
  [-d <recorder context path> <debug session name>]
  [-debug] [-help]
```

Note that you must include the context path or LOID of the class which will parent the new object: e.g., if you wish to create an instance of BasicFileClass you must include BasicFileClass's context path name (`/class/BasicFileClass` in the example below) or LOID. You can create the new object on a specific host or vault object, a host or vault object of a specified class,[8] or one of the host or vault objects in a specified context.

8 You can use `legion_create_class`, page 38 in the Reference Manual, to create your own host classes to organize the types of host objects you wish to use.

The example below creates an instance of the `BasicFileClass` on `aNewHost` and assigns it the name `file`.

```
$ legion_create_object /class/BasicFileClass file \
  -h /hosts/aNewHost
1.01.66000000.01000000.000001fc0b0eec4e02...
$
```

The command's output is the new object's LOID. The newly created object is created on `aNewHost` and given the context name `file` in the current context.

6.4 Look up an object' s host

To find out where an object is actually located, run `legion_get_host`. The output of this command is the object's host object's LOID.

```
$ legion_get_host file
1.01.07.3eb53908.000001fc0d9b155044fb5...
$
```

You can then use `legion_ls -L` to get the host object' s context path.

6.5 Instance placement on hosts and vaults

There is a set of Legion commands to control placement of a class's instances or of a specific instance: `legion_class_host_list` and `legion_class_vault_list` let you change the list of hosts and vaults upon which a given class can place its instances, and `legion_instance_host_list` lets you control where a given object can be placed. There are also commands to display lists of a specified object's acceptable hosts, vaults, and host-vault pairings. These commands are likely to be most useful for resource scheduling and management. Please see the Reference Manual, man pages, or on-line tutorials (<http://legion.virginia.edu/documentation.html>) for more information about these commands.

6.6 Backup vaults

Legion versions 1.7 and forward support backup vaults for certain types of objects. Backup vaults are used to replicate an object's persistent state in case the object's primary vault crashes or is unavailable. When the object deactivates, its state is copied to all of its backup vaults. When it is reactivated, its class will first check the

object's primary vault for its persistent state. If the primary vault is unavailable, the class will look in one of the backup vaults.

Please note that the backup vaults may not have a current copy of the object's state. If the object was previously deactivated, the backup vaults will be current. But if the object was running when the primary vault crashed, the backup vaults' copy will be out-of-date. You should only use backup vaults for objects whose state does not change or who can tolerate recovery from with out-of-date state.

At present, backup vaults can be used for BasicFileClass, ContextClass, ImplementationObject, and UserAuthenticationObject objects.[9] These classes all belong to the SKCC_MetaClass. SKCC stands for Simple K-Copy Class, meaning that the class's state is copied *k* times when an instance is deactivated.

6.6.1 WORM objects

Backup vaults are especially useful for objects whose state never changes, such as Write-Once Read-Many (WORM) objects. An object can be marked as a WORM object with the `legion_set_worm` command. A WORM object's state will be copied to its backup vaults only once. The `legion_unset_worm` command indicates that a WORM object is no longer using WORM semantics and that its state now needs to be updated each time the object deactivates. Please see page 12 in the Reference Manual for more information on these commands.

6.6.2 Assigning and synchronizing backup vaults

There are four commands related to backup vaults:

```
legion_skcc_set_class_vaults
legion_skcc_set_defaults
legion_set_backup_vaults
legion_synch_vaults
```

The first two set defaults for SKCC classes. The third adds and deletes vaults from the object's list of vaults and the fourth synchronizes the copy of the object's state in the backup and primary vaults. Please see the Reference Manual for more information on these commands.

9 These are the default SKCC objects. Your system administrator may have changed this list.

6.6.3 Using replication

Any instance of one of the classes mentioned above can use backup vaults. You can use a current existing object or create a new one:

```
$ legion_cp -localsrc localfile Foo
```

You can then use `legion_set_backup_vaults` to set the instance's backup vaults:

```
$ legion_set_backup_vaults /home/myContext/Foo \
  -a /vaults/VaultA -a /vaults/VaultB
```

When `Foo` deactivates, its state will be replicated to `VaultA` and `VaultB`. At any point you can synchronize its backup vaults' copy of its state with `legion_synch_vaults`:

```
$ legion_synch_vaults /home/mycontext/Foo
```

Alternatively, if you are an SKCC class's owner you can use `legion_skcc_set_class_vaults` to make a list of backup vaults for the class's instances. The example below adds `VaultA` and `VaultB` to `ClassFoo`'s list of backup vaults.

```
$ legion_skcc_set_class_vaults /home/myContext/ClassFoo \
-a /vaults/VaultA -a /vaults/VaultB
```

If you use this command to set up a list of backup vaults for a class, you can use `legion_skcc_set_defaults` to instruct Legion to use back up the class's instances onto a certain number of those backup vaults. For example:

```
$ legion_skcc_set_defaults -c /home/myContext/ClassFoo 3
```

Legion will now by default replicate all `ClassFoo` instances onto three of `ClassFoo`'s backup vaults.

Running applications

7.0 Running a Legion application

The mechanics of running an application in Legion will, of course, vary widely from one application to another. In general, however, you will want to compile, register, and run individual applications with commands written specifically for that purpose. For example, MPI applications use `legion_mpi_register` and `legion_mpi_run`, and PVM applications use `legion_pvm_register`. Some commands may require you to select a group of hosts that the application should use before running the application, others may just ask how many hosts should be used.

You may want to arrange parts of your context space or to organize certain resources around the needs of specific applications (a special context for host objects that can be used for a specific MPI application, for instance).

Once you're ready to start an application, you can view its output with a *tty object* (a special Legion object that monitors and directs output from a shell's processes: see page 75). The tty object can send your output to a file or a window.

Information about Legion's MPI and PVM libraries starts on page 48 and page 52, respectively. See page 23 in the Legion Developer Manual for information about Legion's Basic Fortran Support (BFS).

8.0 Executing programs remotely

Remote execution allows you to run remotely executed programs from the Legion command line, taking advantage of Legion's distributed resources. Your workload can be spread out on several processors, improving job turn-around time and performance. You can execute multiple instances of a single program or execute several different programs in parallel.

A remote program is an executable process that can be started with a command-line utility. Note that a remote program is not part of the Legion system, but is an independent program that may or may not be linked to Legion. Remote programs might be shell scripts, compilers, existing executables, etc. The term “remote” here means that the program is not or may not be executed on the local host.

8.1 Linked and independent programs

Legion permits two kinds of programs to be run remotely: Legion-linked and independent. A Legion-linked program is linked to the Legion libraries. An independent program is not linked to the Legion libraries. For example, a shell script or a program written and compiled on non-Legion systems are independent programs.

8.2 Registering independent programs

Before either a linked or independent remote program can be run from Legion it must be *registered*. The `legion_register_runnable` command registers Legion-linked programs. The `legion_register_program` command registers independent programs. Both of these commands require parameters containing a context path name, the program's binary path name, and the binary’s architecture. You can register a program multiple times to run on different architectures.

The `legion_run` command runs registered programs.

8.2.1 Independent programs

Independent remote programs are registered with `legion_register_program`. This includes information about the program's architecture (e.g., linux, sgi, solaris, etc.), its binary path name, and a program class. Once the program has been registered, it can be executed with `legion_run`.

Usage is:

```
legion_register_program
   <program class>
   <Unix executable path>
   <legion arch> [-debug] [-help]
```

The <program class> parameter is a context path name for a class that will manage the program's Legion objects. The path name can be whatever you find most convenient. You might want to use the program's name so that it will be easier to remember (e.g., if I'm registering the program MyProgram I will use MyProgram as the <program class> parameter). The context path name can be either new or a previously created path name (if you are registering multiple versions of the same program). It will refer to a (new or previously created) Legion object that permits an independent remote program to execute.

If multiple programs are registered with the same program class and architecture, the most recently registered version will be used when the program is run on that particular architecture. *Be sure to reregister the program if you recompile it*. Legion makes a copy of your executable for context space only when you register it and will not update its copy when you update the original.

An example of this command might look like this:

```
$ legion_register_program myProgram /bin/myProgram linux
Program class "myProgram" does not exist.
Creating class "myProgram".
Registering implementation for class "myProgram"
$
```

This output shows Legion creating the program class myProgram, as the command requested. If this class had been previously created, the output would look like this:

```
$ legion_register_program myProgram /bin/myProgram linux
Registering implementation for class "myProgram"
$
```

8.2.2 Legion-linked programs

The legion_register_runnable command registers programs that are linked to the Legion libraries, and exports a runnable object interface. The command creates an interface between the Legion system and the program. Like legion_register_program, this command requires the remote program's program class, executable binary path, and the binary's architecture as parameters.

Its usage is:

```
legion_register_runnable
   <program class> <executable path>
   <legion arch> [debug] [-help]
```

The <program class> parameter is a context path name for the Legion objects that will handle a particular program. You can choose a context path that suits your organizational scheme. If you reregister a program and use the same program class name, Legion will use the most recently registration information when running the program. Once the program has been registered, you can run it with legion_run. You must reregister the program if you recompile it (Legion copies your executable to context space when you register it, so the most recently registered copy will be run).

```
$ legion_register_runnable myProgram /bin/myProgram linux
Program class "myProgram" does not exist.
Creating class "myProgram".
Registering implementation for class "myProgram"
$
```

This output shows Legion creating the program class myProgram, as the command requested. If this class had been previously created, Legion will simply register the new information:

```
$ legion_register_runnable myProgram /bin/myProgram linux
Registering implementation for class "myProgram"
$
```

8.3 Running a program remotely

Use legion_run to start a single instance of a program on a remote host. If you are running a serial program with many input files and/or multiple executions you may prefer to use the legion_run_multi command (see page 104 in the Reference Manual). The program must be registered with either legion_register_program (for independent programs not linked to the Legion libraries) or legion_register_runnable (for Legion-linked programs) before you run it.

There are a number of optional parameters associated with legion_run. To try to keep things simple, we will not discuss all of them here: please see page 100 for the complete syntax and explanation of all options.

The legion_probe_run command is a useful aid when remote programs. It lets you pass input files to the remote host after the program has started running, pick up output files, check the job's status, and clean up the remote host after the job has finished.

8.3.1 Choosing a remote host

Legion will choose a host with an appropriate architecture if you do not specify one with the `-h` flag. The host must be part of your system (i.e., have a host object and be listed in the `/hosts` context: if necessary, see “Adding a new host“ on page 52 in the System Administrator Manual).

8.3.2 Command-line arguments

If your program requires command-line arguments, you must include them as the final parameters to `legion_run`. They can not be put in an option file.

8.3.3 Getting input files to the remote host

Your program may expect to find certain files in its local directory at run time, so when you execute it on a remote host you must pass copies of any expected input files to that host. If you know the name of the remote host, you can pass the files by hand, but otherwise you'll find it much easier to tell Legion the name of the files that need to be copied. You can do this before or after the program starts.

8.3.3.1 Before the program starts

You can use the `-in` and `-IN` flags to move files onto the remote host before the program starts. The `-in` flag gets files from context space and the `-IN` file gets them from the local host. Or, use `legion_probe_run` to move files to the remote host after the program starts.

From context space

The `-in` flag tells Legion to copy the contents of a context file object into a new file in the remote program's current working directory before the program executes. The new file's name will be the same as the object's context name. E.g., if you use

```
-in /home/me/myContext/inputFoo
```

Legion will copy the contents of `/home/me/mycontext/input-Foo` to a new Unix file in the job's current working directory on the remote host and will call the new file `inputFoo`.

From Unix space

The `-IN` flag tells Legion to copy the contents of a local file into a new file in the remote program's current working directory before the program executes. The new file will have the same name as the local file.

8.3.3.2 After the program has started

Use `legion_probe_run` to move input files to the remote host after the program has started running (see section 8.3.8). If you wish to use this method, please note that you must have started the program with a probe file (see section 8.3.6).

You can use the `-in` or `-IN` flags to pass input files from context space or local file space to the remote job's current working directory. These flags are identical to `legion_run`'s `-in` and`-IN` flags, described above.

8.3.4 Getting output files from the remote host

Once the program has finished, you need to retrieve the output files. If you know the name of the remote host and you are running in nonblocking mode you can get the files by hand (see section 8.3.7). Otherwise, you will need tell Legion the names of the output files that you wish to retrieve and whether you wish to put them on your local host or in context space. You can do this before or after the program starts.

8.3.4.1 Before the program starts

You can use `legion_run`'s `-out` and `-OUT` flags to specify which files you want copied from the remote host. The `-out` flag copies files into context space and the `-OUT` file copies them onto your local host.

If the program has terminated because of a crash, Legion may not be able to copy all of the results.

Into context space

The `-out` flag tells Legion to copy the contents of a specified output file from the program's current working directory (on the remote host) into a new Legion file object in your context space once the program is terminated. The new file object will have the same name as the specified output file. E.g., if you tell Legion to copy `outputFoo`:

```
-output outputFoo
```

Legion will look for a file called `outputFoo` in the remote directory and copy its contents into a new file object in your context space called `outputFoo`.

Onto your local host

The `-OUT` flag tells Legion to copy the remote output file into a new file on your local host. The new file will have the same name as remote output file.

8.3.4.2 After the program has started

You can use the `-out` and `-OUT` flags with `legion_probe_run` to specify which files you want copied from the remote host after the program has started running. If you wish to use this method, please

note that you must have started the program with a probe file (see section 8.3.8).

The `-out` flag copies files into context space. The `-OUT` file copies them onto your local host. These flags are identical to `legion_run`'s `-out` and `-OUT` flags.

8.3.5 Option file

If you find that you are trying to keep track of too many options when you start `legion_run`, you may wish to use an option file. This is a text file that holds a list of all of your flags and optional settings for that run. The file can be delimited with tabs, spaces, or new lines and can contain any of the `legion_run` flags except for the program class name and any command-line arguments: those must appear on the command line, along with the `-f` flag and the option file name. Please note that you can only use `legion_run` flags in this file.

8.3.6 Creating and using a probe file

A probe file is a Unix file on your local machine. The `legion_probe_run` command uses it to contact a remote job. To create one, simply use the `-p` flag when you start `legion_run`.

If you run in blocking mode, `legion_run` automatically will remove a job's probe file when the job is finished. If you run in nonblocking mode, `legion_probe_run`'s `-kill` option will delete it as part of its cleaning-up operation. Otherwise, the file will remain in your local file space.The probe file is good for only one remote job, so if you reuse the name Legion will simply write over the previous file if it still exists.

8.3.7 Blocking vs. nonblocking

You can run the `legion_run` command in *blocking* or *nonblocking* mode.

The default mode is blocking. This means that `legion_run` will continue to run on your command line until the remote job has finished. All output files will be collected from the remote host, the remote directory that was used to run the job will be cleaned up, and the command will exit.

Blocking

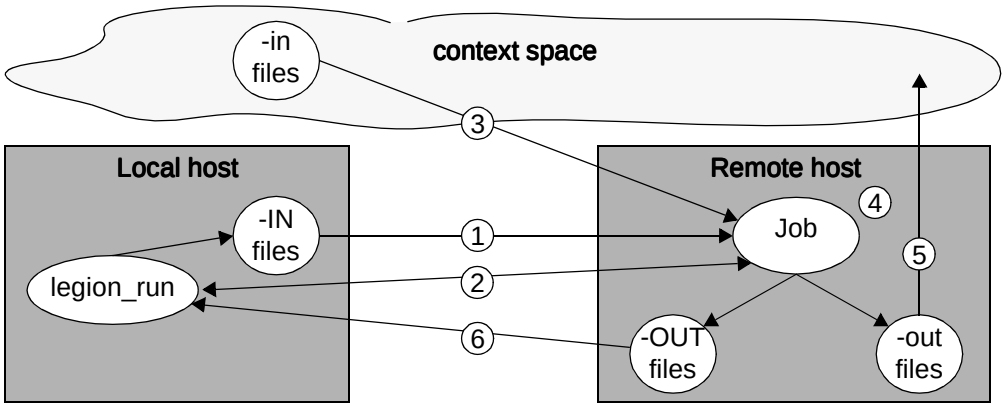
The basic steps in running a blocking remote job are shown below in Figure 6.

In this example, a user starts a remote job on his local host in blocking mode. The following events occur:

1. The `legion_run` tool sends a copy of local input files (marked with the `-IN` flag) to the remote host.

2. The `legion_run` asks the remote host to start the job. It then blocks and waits for the job to finish.
3. The remote job gets a copy of context input files (marked with the `-in` flag).
4. The job starts running.
5. The remote job finishes and creates its output files. Any files that were marked with the `-out` flag are copied into context space.
6. The remote job tells `legion_run` that it is finished. `legion_run` copies any files marked with the `-OUT` flag into the user's local file space and cleans up the remote job's working directory.

Figure 6: A blocking remote run



You can use ^C to kill the job prematurely.

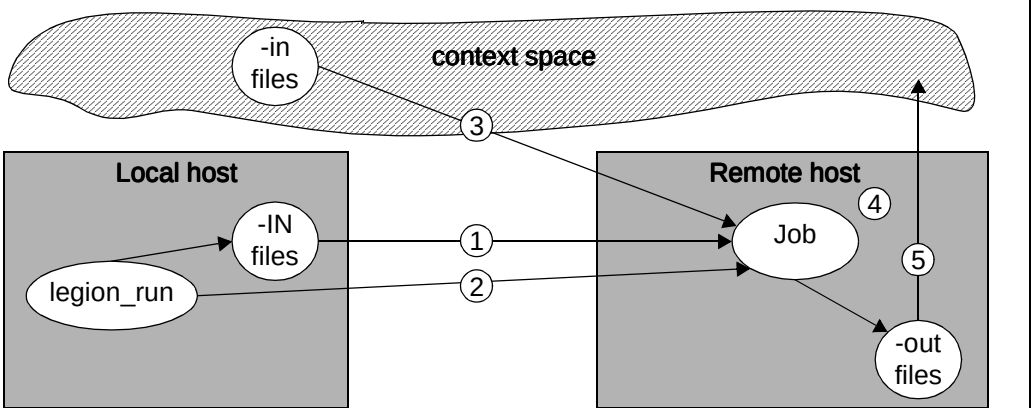
Nonblocking

If you start `legion_run` with the `-nonblock` flag, on the other hand, the command will start the program on the remote host, wait ten seconds, verify that the program can start, and then die. When the remote host has finished executing the program, the job will copy any files named with the `-out` flag into context space but ignore any files named with the `-OUT` flag. The job's working directory will remain on the remote host. The basic steps are shown below in Figure 7.

In this example, a user starts a remote job on his local host using the `-nonblock` flag. The following events occur:

1. The `legion_run` tool sends a copy of local input files (marked with the `-IN` flag) to the remote host.
2. The `legion_run` asks the remote host to start the job. It waits ten second, verifies that the job can start and exits.
3. The remote job gets a copy of context input files (marked with the `-in` flag).
4. The job starts running.

Figure 7: A nonblocking remote run



5. The remote job finishes and creates its output files. Any files that were marked with the `-out` flag are copied into context space.

Since this is nonblocking mode, the remote job's working directory does not get cleaned up. The remote host will hold on to the job's working directory for six hours. During this period, the user can remove any remaining output files by hand. If he started a probe file, he can remove copy output files to his local host with `legion_probe_run`'s `-OUT` flag.[10] If the user does not take clean up the remote host in that period, the entire directory will be tarred and moved into the user's context scratch space (see section 8.3.9).

8.3.8 About **`legion_probe_run`**

The `legion_probe_run` command checks a remote job that was started with `legion_run`. You must know the name of the job's probe file (see above). You can use this command to pass input files, pick up output files, find what host is running your program, check the job's status, see what files are in the job's current working directory, kill the job, and clean up the remote host when the job is finished. An example is shown in Figure 8, below.

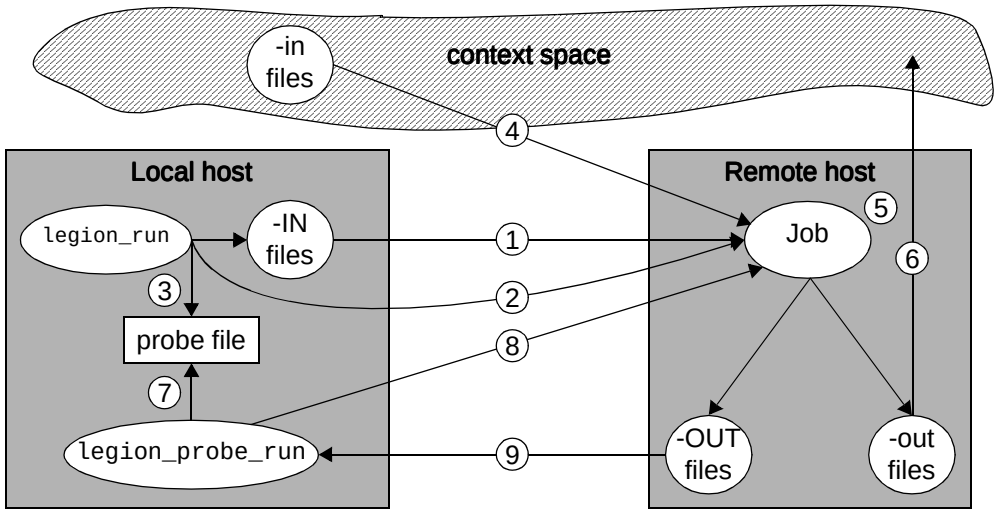
In this example, a user starts a job using `-nonblock` to run it in nonblocking mode and `-p` to create a probe file. The following events occur:

1. The `legion_run` tool sends a copy of local input files (marked with the `-IN` flag) to the remote host
2. The `legion_run` asks the remote host to start the job.
3. It also creates and stores a probe file on the local host. It then waits ten second, verifies that the job can start and exits.

10 If you run `legion_probe_run` on a terminated job, the clock will restart and give you another six hours.

4. The remote job gets a copy of context input files (marked with the `-in` flag).
5. The job starts running.
6. The remote job finishes and creates its output files. Any files that were marked with the `-out` flag are copied into context space.
7. The user runs `legion_probe_run` with the `-statjob` flag to check on the job. The tool looks for the probe file in its local execution environment.
8. The `legion_probe_run` tool contacts the remote job and sees that it is finished.
9. The user runs `legion_probe_run` with `-OUT` flag, telling Legion which files he wants copied onto his local host. (In this scenario it makes no difference whether or not the user ran `legion_run` with `-OUT` flags: if `legion_run` is set to nonblocking it will not stick around to move files back to the local host.)

Figure 8: A nonblocking remote run with `legion_probe_run`



The user can then run `legion_probe_run` with the `-kill` flag. This destroys the remote job's working directory and the probe file. Please note that if you use this flag before the job is finished you will terminate the job and lose all data.

Please see page 96 in the Reference Manual for more information about this command.

8.3.9 Context scratch space

Legion uses context space as backup storage (scratch space) for any remote jobs that finish and are not cleaned up by `legion_run` or `legion_probe_run`. If the job finishes and is not picked up or checked for six hours, the remote host will tar, compress, and move the job's working directory into your context scratch space. The default scratch space is `/tmp` but you can set it to anywhere in your context space.[11] There are several ways to do this.

1. Set the LEGION_SCRATCH_CONTEXT variable from the command line:

```
$ LEGION_SCRATCH_CONTEXT=/home/myContext/scratch/
```

2. Use `-setscratch` or `-D` when you run `legion_run`.
3. Use `-setscratch` when you run `legion_probe_run`.

If you set it multiple times, the most recent setting will be used.

8.3.10 Retrieving files from context scratch space

If your remote job's working directory was sent to context scratch space, you'll need to copy it to your local host then uncompress and untar it in order to use it. First, run `legion_cp`:

```
$ legion_cp -localdest <tar file> <local_file_name.tar.Z>
```

Be sure to give the local file name a *tar.Z suffix. You then need to run `uncompress` and `tar`:[12]

```
$ uncompress <local_file_name.tar.Z>
$ tar xvf <local_file_name.tar>
```

8.4 Example

Suppose that you have a program called Doris that you wish to run on a remote linux host. The first step is to determine whether it is linked to the Legion libraries (Legion-linked) or not (independent). In this case, we'll suppose that Doris is not linked to the Legion libraries.

11 You may need a different scratch space if security is enabled. You may not have permission to work in the `/tmp` context or your system administrator may have removed the context altogether.

12 These are common Unix tools that should be available on all Unix platforms. If you do not have access to these tools, please contact us at legion-help@virginia.edu.

1. The first step is to register the program with Legion. Since it's independent, use `legion_register_program`. To keep things simple, call the program class `doris`.

```
$ legion_register_program doris /bin/Doris linux
Program class "doris" does not exist.
Creating class "doris".
legion_update_attributes: Added 1 attributes(s) to object
legion_update_attributes: Added 1 attributes(s) to object
Registering implementation for class "doris"
$
```

2. If you haven't already set a tty object for your shell, do so now.
3. You can now run the program. You have two options: to run it as in blocking mode (the default setting) or in nonblocking mode. Doris only takes a few minutes to run, so there's no reason not to use the default setting. If it took a long time to run and you wanted to free up the command line, you might choose to run in nonblocking mode. You should, though, make a probe file called dorisProbe to check on Doris's progress.

   Furthermore, Doris expects two input files, input1 and input2, and produces two output files, output1 and output2, so you need to pass their names to Legion when you start `legion_run`.

```
$ legion_run -IN input1 -IN input2 -OUT output1 \
  -OUT output2 -p dorisProbe doris
```

4. You didn't ask for a specific host, but you can get the remote host's name and the remote job's working directory with `legion_probe_run`.

```
$ legion_probe_run -hostname -pwd -p dorisProbe
gander.cs.virginia.edu
/myDir/OPR/BootstrapVaultOPR/ReservedOPR-1.80
$
```

   Note that you can use `legion_run`'s `-h` and `-d` flags to specify a remote host and directory when you start the program.
5. Once the program finishes, `legion_run` copies the `-OUT` files to the local host, deletes the probe file and cleans up the remote working directory. If you had run this in nonblocking mode you would have to pick up the `-OUT` files and then clean up, either with `legion_probe_run` or by hand.

8.5 Converting a C/C++ program

Any C or C++ program can be made into a Legion runnable object using the following steps:

1. The program should export a C-linkable `legion_main` function in place of its `main` function.
2. The program should be linked to the Legion libraries. Add the following to your link line:

```
-lLegionRun -lLegion1 -lLegion2
```

9.0 PVM

PVM (Parallel Virtual Machine) is a software package designed to link up a heterogeneous collection of Unix and NT machines into a network that functions as a virtual single parallel computer. For more information, see the web site (<http://www.epm.ornl.gov/pvm/>).

9.1 Core PVM interface

The current Legion HPC package supports a core PVM interface [3] [15], which includes functions for data marshaling, message passing, and task creation and control. Legion supports legacy PVM codes, and provides an enhanced PVM environment using Legion features such as security and placement services. For example, PVM applications can be run with encrypted message passing on Legion's PVM implementation. If you're not sure whether or not your system includes PVM support, ask you system administrator.

A link-in replacement PVM library uses the primitive services provided by Legion, to support the PVM interface. For example, PVM tasks map directly to Legion objects: thus `pvm_spawn()` uses `Legion.CreateObject()`, and PVM task ids are mapped to Legion LOIDs. PVM Buffers are implemented by using Legion Buffers (the fundamental data container in the Legion runtime library): `pvm_pmkint()` thus uses `LegionBuffer::put_int()`. Similarly, PVM message-passing maps to Legion method invocation: the PVM-Legion library exports a new method, `LegionPVMSend()`. Sending a message to a task maps to calling the `LegionPVMSend()` method on the object corresponding to that task. A `LegionBuffer` containing the message becomes the parameter to the method.

9.2 Tids & LOIDs

Whereas PVM tasks are identified by integer task identifiers (Tids), Legion objects are identified by LOIDs, which come in varying sizes but are generally much larger than an int. The Legion-PVM library must therefore provide a mapping between Tids and LOIDs. Legion's contexts are used to manage this mapping, and the Legion library caches the mappings that have already been looked up. Since Tid-to-LOID mappings are invariant, caching can be performed without any consistency management costs.

9.3 Task classes

Whereas PVM spawn requires a “task name” parameter to identify the type of task to spawn, Legion uses class LOIDs to specify an object's type. The Legion-PVM library utilizes a single class object for each type of task: therefore, Legion-PVM tasks are instances of Legion classes. Mapping between task names (i.e., as specified to `pvm_spawn()`) and class LOIDs is managed in Legion context space.

Legion classes have associated “implementation objects,” which provide their instances with binary implementations. Because of this, PVM-Legion does not spawn tasks stored in a “well-known” directory location, as standard PVM does. A mechanism is provided for “registering” task implementations with the appropriate classes. Once executables are registered, the user does not need to explicitly manage the copying of binaries between hosts on disjoint file systems, since Legion automatically manages this, unlike standard PVM.

9.3.1 Legion-PVM library

The Legion-PVM library is in $LEGION_HPC/lib/linux/g++.

9.4 Compilation

Task object code files can be compiled as before. Link against libLegionPVM.3, and basic Legion libraries (the final linkage must be performed using a C++ compiler, or using C++ appropriate flags for `ld`, since the Legion library has C++ link-time requirements). A sample Legion PVM makefile is shown in Figure 9.

Figure 9: Sample Legion PVM makefile

```
CC    = gcc
LD    = g++
INCDIR  = $(LEGION_HPC)/include/MessagePassingLibraries/PVM3
CFLAGS  = -I$(INCDIR)

example: example.o
    legion_link -pvm example.o -o example
    legion_pvm_register example $LEGION_ARCH
example.o: example.c
    $(CC) $(CFLAGS) -c example.c -o $@
```

9.5 Registering compiled tasks

This step can be skipped for tasks that will be started only from the command line (tasks that will not be “spawned”).

Run `legion_pvm_register`:

```
legion_pvm_register <class name>
   <binary path> <platform type>
   [-debug] [-help]
```

This creates PVM-specific contexts, a class object for the task class, registers the name in Legion context space, and creates an implementation object for the class.

You can now run the PVM application. If necessary, you can examine Legion context space to get information about the PVM-Legion state with `legion_ls`.

```
legion_ls /pvm
```

or

```
legion_ls /pvm/tasks
```

The first lists the Tids of running PVM tasks (use the `-L` flag to include LOIDs), the second lists registered task classes. You can also use Legion class utilities to examine the class state (e.g., `legion_list_instances`).

9.6 Examples

There are several examples of PVM programs in the HPC package. They are in the $LEGION_HPC/src/Examples/PVM directory.

This section uses a sample program called hello_other to demonstrate the process of registering and running a PVM program. To compile this code, enter:

```
$ make $LEGION/bin/$LEGION_ARCH/hello_other
```

To register your program with the Legion/PVM system, use `legion_pvm_register`. The first time this is run, Legion will create `/pvm` and `/pvm/tasks` directories. Your output will look something like this:

```
$ legion_pvm_register hello_other \
 $LEGION/bin/$LEGION_ARCH/hello_other $LEGION_ARCH
“/pvm” does not exist - creating it
“/pvm/tasks” does not exist - creating it
Task class “/pvm/tasks/hello_other” does not exist -
creating it
$
```

The `hello_other` class is now registered in context space and can be seen with `legion_ls`.

```
$ legion_ls -l /pvm/tasks
hello_other          (class)
$
```

You can now run a simple example, with the new class. This example creates a new task and returns its and its host's PVM task ids in a simple message. Note that the command to run the example does not have a `legion_` prefix.

```
$ hello
i'm t34c8e665
from t64d7b9e5: hello, world from t64d7b9e5 (parent
t34c8e665) on xxx.xxx.xxx.xxx
$
```

Other programs can be registered and run using the same procedure.

9.7 Running PVM code with the fewest changes

Please see the Legion web site's tutorials section (see <http://legion.virginia.edu/documentation.html>) for an on-line tutorial called "Running a PVM code in Legion with the fewest changes." The tutorial shows how to adapt a sample PVM program to run in Legion. Please note, however, that while this "fewest changes" approach allows you to run the program and to transparently read and write files remotely, it does not support heterogeneous conversions of data. If you run this program on several machines which have different integer formats, such as Intel PCs (little-endian) and IBM RS6000s (big-endian), using unformatted I/O may produce surprising results. If you want to use such a heterogeneous system, you will have to either use formatted I/O (all files are text) or use the "typed binary" I/O calls instead of Fortran READ and WRITE statements. These "typed binary" I/O calls are discussed in section 6.2.4 on page 52 ("Buffered I/O library, low impact interface") in the Developer Manual.

10.0 MPI

MPI (Message Passing Interface) is a standard for writing parallel programs in message-passing environments. For more information please see the MPI web site, <http://www-unix.mcs.anl.gov/mpi/>.

The current Legion HPC package supports a core MPI interface, which includes messages passing, data marshaling and heterogeneous conversion. Legion supports legacy (native) MPI codes and provides an enhanced MPI environment using Legion features such as security and placement services. A link-in replacement MPI library uses the primitive services provided by Legion to support the MPI interface. MPI processes map directly to Legion objects. If you're not sure whether or not your system includes MPI support, ask you system administrator.

The remainder of this section is divided into two sections: legion MPI (section 10.1, below) and native MPI (section 10.2 on page 66). Legion MPI programs have been adapted to run in Legion, are linked to Legion libraries, and can only be run on machines that have the Legion binaries installed. Native MPI programs do not need to fulfill any of these requirements (although they may: a Legion MPI program can be run as a native MPI program but not vice versa).

10.1 Legion MPI

10.1.1 Task classes

MPI implementations generally require that MPI executables reside in a given place on disk. Legion's implementation objects serve a similar role. We have provided a tool, `legion_mpi_register`, to register executables of different architectures for use with Legion MPI.

10.1.2 Legion MPI libraries

The Legion-MPI libraries are in $LEGION_HPC/lib/linux/g++.

10.1.3 Compilation

MPI code may be compiled as before. Link it against libLegionMPI and the basic Legion libraries. The final linkage must be performed using a C++ compiler or by passing C++ appropriate flags to `ld`. A sample Legion MPI makefile is shown in Figure 10 (below).

Figure 10: Sample Legion MPI makefile

```
C         = gcc
CC        = g++
F77       = g77
OPT       = -g
INC_DIR   = $LEGION_HPC/include/MessagePassingLibraries/MPI
FFLAGS    = -I$(INC_DIR) -ffixed-line-length-none

mpitest: mpitest.o
    legion_link -Fortran -mpi mpitest.o -o mpitest

mpitest.o: mpitest.f
    $(F77) -c mpitest.f -I$(INC_DIR) $(FFLAGS)

mpitest_c: mpitest_c.o
    legion_link -mpi mpitest_c.o -o mpitest_c

mpitest_c: mpitest_c.c
    $(C) $(CFLAGS) mpitest_c.c -o mpitest_c $(LDFLAGS) $(LIBS)
```

10.1.4 Register compiled tasks

Run `legion_mpi_register`. Usage of this tool is:

`legion_mpi_register <class name> <binary path> <platform type> [-help]`

The example below registers /myMPIprograms/vdelay as using a Linux architecture.

```
$ legion_mpi_register vdelay /myMPIprograms/vdelay linux
```

If necessary Legion will create the MPI-specific contexts (`/mpi`, `/mpi/programs`, and `/mpi/instances`), a class object for this program, an implementation for this program, and registers the name in Legion context space. In a secure system (a Legion system with security turned on), these contexts will be placed in `/home/<user id>`. If the user is logged in as `admin` or the net is running without security then the MPI contexts will be placed in the `/` (root) context.

The `legion_mpi_register` command can be executed more than once if you have compiled your program on different architectures.

10.1.5 Running the MPI application

MPI programs are started with `legion_mpi_run`:

```
legion_mpi_run
  {-f <options file> [<flags>]} |
    {-n <num processors> [<flags>] <program class>
      [<args>]}
```

There are number of parameters and flags for this command. See page 85 in the Reference Manual for more information.

To control object scheduling, use a specification file. This can be a local file or a Legion file object and contains a list of acceptable hosts and how many objects can be run on each host. You can use `legion_make_schedule` to produce a file or write it by hand. To write a specification file by hand, list one host (by context path) and one optional integer (indicating the number of objects the host can create – default is 1) per line. A host can be listed multiple times in one file, in which case the integer values accumulate. E.g.:

```
/hosts/BootstrapHost       5
/hosts/slowMachine1
/hosts/bigPBSqueue         100
/hosts/slowMachine2        1
```

Use `-hf` or `-HF` to use a specification file.

You can start multiple MPI applications with an option file. All of the applications must have been previously registered (with `legion_mpi_register`) and must use a common MPI_COMM_WORLD. As the example below shows, an option file has one application per line, including any necessary arguments as they would appear on the command line. Each line can also contain any of the `legion_mpi_run` flags (except `-f`).

```
-n 2 /mpi/programs/mpitest
-n 3 /mpi/programs/mpitest_c
```

This would start a run of five instances (two of `mpitest` and three of `mpitest_c`). All five instances would use a single MPI_COMM_WORLD.

You must include `-f` when you run `legion_mpi_run` in order to use a option file. Note that if you use an option file, the `-n` flag, program name, and any arguments will be ignored. Any other `<flags>` will be treated as defaults and applied to all processes executed by this command, unless otherwise specified in the option file.

If you don't use an option file, you must provide an MPI program name on the command line, along with any flags and arguments. For example:

```
$ legion_mpi_run -n 2 /mpi/programs/vdelay
```

Note that the application name here is the full context space path for the class created by `legion_mpi_register` in section 10.1.4.

As the program is running you can view your application's instances in `/mpi/instances/<program_name>`:

```
$ legion_ls /mpi/instances/vdelay
```

In a secure system the instances will be in `/home/<user_name>/mpi/instances/<program_name>`. If you run multiple versions of an application simultaneously, you can use `-p` to specify a different context to hold your instance LOIDs.

To view all of your instances, use `legion_list_instances`:

```
$ legion_list_instances /mpi/programs/<program_name>
```

To destroy them, use `legion_destroy_instances` command. Be aware, though that this command will destroy *all* of the class's instances.

You can use `legion_mpi_probe` to check jobs you've started on remote hosts. You can check on jobs, move input and output files between your local and execution hosts. Please see page 84 in the Reference Manual for more information.

MPI programs cannot be deactivated.

10.1.6 Example

There is a sample MPI program included in a new Legion system, written in C (mpitest_c.c) and Fortran (mpitest.f) in the HPC package MPI examples directory ($LEGION_HPC/src/Examples/MPI). You must register whichever version you wish to use with the `legion_mpi_register` command.

The sample output below uses `mpitest_c` and shows Legion creating three new contexts (`mpi`, `programs`, and `instances`, the last two being subcontexts of `mpi`), creating and tagging a task class, and registering the implementation.

```
$ legion_mpi_register mpitest_c \
   $LEGION/bin/$LEGION_ARCH/mpitest_c linux
"/mpi" does not exist - creating it
"/mpi/programs" does not exist - creating it
"/mpi/instances" does not exist - creating it
Task class "/mpi/programs/mpitest_c" does not exist -
      creating it
"/mpi/instances/mpitest_c" does not exist - creating it
```

```
Set the class tag to: mpitest_c
Registering implementation of mpitest_c
$
```

In order to view the output of the program, you will need to create and set a tty object if you have not already done so (see page 75).

```
$ legion_tty /context_path/mytty
```

You can now run the program. Since you are not using an option file, you must specify how many processes the program should use to run the program with the `-n` flag. If you do not specify which hosts the program should run its processes on, Legion will arbitrarily choose the hosts from the host objects that are in your system. Here it runs three processes on three different hosts.

```
$ legion_mpi_run -n 3 /mpi/programs/mpitest_c
Hello from node 0 of 3 hostname Host2
Hello from node 1 of 3 hostname Host3
Node 0 has a secret, and it's 42
Node 1 thinks the secret is 42
Hello from node 2 of 3 hostname BootstrapHost
Node 2 thinks the secret is 42
Node 0 thinks the secret is 42
Node 0 exits barrier and exits.
Node 1 exits barrier and exits.
Node 2 exits barrier and exits.
$
```

Another time it might run the three processes on two hosts.

```
$ legion_mpi_run -n 3 /mpi/programs/mpitest_c
Hello from node 1 of 3 hostname BootstrapHost
Hello from node 0 of 3 hostname Host3
Node 0 has a secret, and it's 42
Node 1 thinks the secret is 42
Hello from node 2 of 3 hostname BootstrapHost
Node 2 thinks the secret is 42
Node 0 thinks the secret is 42
Node 0 exits barrier and exits.
Node 1 exits barrier and exits.
Node 2 exits barrier and exits.
$
```

10.1.7 Input and output files

If your program requires input files and/or generates output files, you may need to move these files between your local host and one or more remote hosts. There are two ways to accomplish this when you start the job: through flags in an option file or on the command line and

through subroutines in your code. After the job has started, you can use `legion_mpi_probe` (see page 84 in the Reference Manual).

10.1.7.1 Input and output flags

Like `legion_run`, `legion_run_mpi` uses `-in`/`-out` and `-IN`/`-OUT` flags to copy input and output files between your local host or context space and a remote host. The `-in`/`-IN` flags copy a specified input file to a job's remote host before execution. The `-out`/`-OUT` flags copy a specified output file from the remote host back to your local host or context space once the program has finished. You can repeat these flags multiple times in order to organize multiple files.

Note that the default setting will distribute files in these parameters to node 0 only. Use the `-a` flag if you need to get these files to other nodes. Be aware, though, this flag changes the naming pattern of your output files. The object's MPI object identification will be tacked on to the end of the file name. For example:

```
$ legion_mpi_run -n 2 -a -OUT done \
  /mpi/programs/mpiFoo
```

When `mpiFoo` has finished, it will have produced output files `done` on two nodes. Legion copies both of the files to your local directory and names them something like `done.1183374393-0.0` and `done.1183374393-0.1`.

You can also use wildcards. The following characters are wild when used with `-in`/`-out` and `-IN`/`-OUT`:

*****	match 0 or more characters
?	match any one character
[-]	match any character listed between the brackets (use to specify a range of characters)
****	treat the character as a literal

So, you could use * to identify any file that begins with "done" as an input file for `mpiFoo`:

```
$ legion_mpi_run -n 2 -IN done.* \
 /mpi/programs/mpiFoo
```

Similarly, if you wanted to use `done.1`, `done.2`, `done.3` ... `done.9` as your input files you could use square brackets to identify them as a group:

```
$ legion_mpi_run -n 2 -IN done.[0-9] \
  /mpi/programs/mpiFoo
```

Wildcards can only be used with file names, not with directories. Please see page 85 in the Reference Manual for more information.

10.1.7.2 Subroutines

You may prefer to edit your program, so as to transparently access other files for input or output. Legion has three subroutines that allow your program to read and write files in Legion context space: LIOF_LEGION_TO_TEMPFILE, LIOF_CREATE_TEMPFILE, and LIOF_TEMPFILE_TO_LEGION.

The first, LIOF_LEGION_TO_TEMPFILE, lets you copy a file from Legion context space into a local file. For example,

```
call LIOF_LEGION_TO_TEMPFILE ('input', INPUT, ierr)
open (10, file = INPUT, status = 'old')
```

will copy a file with the Legion filename `input` into a local file and store the name of that local file in the variable `INPUT`. The second line opens the local copy of the file.

The other two subroutines can be used to create and write to a local file. For example:

```
call LIOF_CREATE_TEMPFILE (OUTPUT, IERR)
open (11, file = OUTPUT, status = 'new')
call LIOF_TEMPFILE_TO_LEGION (OUTPUT, 'output', IERR)
```

The first line creates a local file and stores the file's name in the variable `OUTPUT`. The second line opens the local copy of the file. The third line copies the local file `OUTPUT` into a Legion file with the name `output`.

While this approach allows you to run MPI programs and transparently read and write files remotely, it does have one limitation: it does not support heterogeneous conversions of data. If you run this program on several machines which have different formats for an integer, such as Intel PC's (little-endian) and IBM RS6000s (big-endian), using unformatted I/O may produce surprising results. If you want to use such a heterogeneous system, you will have to either use formatted I/O (all files are text) or use the "typed binary" I/O calls instead of Fortran `READ` and `WRITE` statements. These "typed binary" I/O calls are discussed in section 6.2.4, "Buffered I/O library, low impact interface" in the Legion Developer Manual.

10.1.8 Scheduling MPI processes

As noted above, Legion may not choose the same number of hosts objects as processes specified in the `-n` parameter. If you specify three processes, Legion will arbitrarily choose to run them on one, two, or three hosts.

We will run `mpitest_c` on four hosts, which we will assume are already part of the system. If you wish to specify which hosts are used to run your processor, use the `-h` flag. It tells Legion to look in a given

context for a set of potential hosts for running your program. You'll need to create this context by hand.

To do it by hand, you must create the new context then link the desired hosts to names in the new context. For example, if you want to create an `mpi-hosts` context, run `legion_context_create`:

```
$ legion_context_create mpi-hosts
```

Then create context names for the hosts that you wish to use for this program. Use `legion_ln` to map the new names to the host objects' existing LOIDs.

```
$ legion_ln /hosts/Host1 /mpi-hosts/mpitest_Host1
$ legion_ln /hosts/Host2 /mpi-hosts/mpitest_Host2
$ legion_ln /hosts/Host3 /mpi-hosts/mpitest_Host3
$ legion_ln /hosts/Host4 /mpi-hosts/mpitest_Host4
```

The program can now be run with `legion_mpi_run`, using `-h` to specify that Legion should look in `/mpi-hosts` for hosts and `-n` to specify that they be spread over four nodes. Note that the nodes are placed on the hosts in order.

```
$ legion_mpi_run -h /mpi-hosts -n 4 /mpi/programs/mpitest
Hello from node 0 of 4 hostname Host1.xxx.xxx
Hello from node 1 of 4 hostname Host2.xxx.xxx
Node 0 has a secret, and it's 42
Hello from node 2 of 4 hostname Host3.xxx.xxx
Node 1 thinks the secret is 42
Node 2 thinks the secret is 42
Hello from node 3 of 4 hostname Host4.xxx.xxx
Node 3 thinks the secret is 42
Node 0 thinks the secret is 42
Node 0 exits barrier and exits.
Node 1 exits barrier and exits.
Node 2 exits barrier and exits.
Node 3 exits barrier and exits.
$
```

The number of processes and number of host object do not have to match: if you wanted to run only two processes on the hosts named in `mpi-hosts` you could use `-n 2`.

```
$ legion_mpi_run -h /mpi-hosts -n 2
/mpi/programs/mpitest
Hello from node 0 of 2 hostname Host1.xxx.xxx
Node 0 has a secret, and it's 42
Hello from node 1 of 2 hostname Host2.xxx.xxx
Node 1 thinks the secret is 42
Node 0 thinks the secret is 42
Node 0 exits barrier and exits.
Node 1 exits barrier and exits.
$
```

The program runs on the first two hosts listed in the `mpi-hosts` context.[13]

10.1.9 Debugging support

A utility program, `legion_mpi_debug`, allows the user to examine the state of MPI objects and print out their message queues. This is a handy way to debug deadlock. Usage is:

```
legion_mpi_debug [-q]
   {[-c] <program instance context>}
   [-help]
```

The context that is holding your program's instances goes in the `<program instance context>` parameter. If you did not specified otherwise when you started the program, this will normally be `/mpi/instances/<program_name>` or (in a secure system) `/home/<user_name>/mpi/instances/<program_name>`. The command will return a list of all of the program's instances and what each one is doing.

The `-q` flag will list the contents of the queues. The output will be something like this:

```
$ legion_mpi_debug -q /mpi/instances/vdelay
Process 0 state: spinning in MPI_Recv source 1 tag 0 comm?
Process 0 queue:

(queue empty)

Process 1 state: spinning in MPI_Recv source 0 tag 0 comm?
Process 1 queue:

MPI_message source 0 tag 0 len 8
$
```

Do not be alarmed that process `1` hasn't picked up the matching message in its queue: it will be picked up after the command has finished running.

There are a few limitations in `legion_mpi_debug`: If an MPI object doesn't respond, it will hang, and it won't go on to query additional objects. An MPI object can respond only when it enters the MPI library; if it is in an endless computational loop, it will never reply.

Output from a Legion MPI object to standard output or Fortran unit 6 is automatically directed to a tty object (see page 75). Note, though, that this output will be mixed with other output in your tty object so you may wish to segregate MPI output. Legion MPI supports a special set of library routines to print on the Legion tty, which is created using the `legion_tty` command. This will cause the MPI object's output to be printed in segregated blocks.

13 If you use `-h` in this way, be sure to double-check the order in which the host objects are listed in your context.

`LMPI_Console_Output` (*string*)
Buffers a string to be output later.

`LMPI_Console_Output_Flush` ()
Flushes all buffered strings to the tty.

`LMPI_Console_Output_And_Flush` (*string*)
Buffers and flushes in one call.

The Fortran version of these routines adds a carriage return at the end of each string, and omits trailing spaces. The C version does not.

10.1.10 Checkpointing support

As you increase the number of objects or the length of time for which an MPI application runs, you increase the probability that the application will crash due to a host or network failure.

To tolerate such failures, we provide a checkpointing library for SPMD-style (Single Program Multiple Data) applications.[14] SPMD-style applications are characterized by a regular communication pattern. Typically, each task is responsible for a region of data and periodically exchanges boundary information with a neighboring task.

We exploit the regularity of SPMD applications to provide an easy-to-use checkpointing library. Users are responsible for (1) deciding when to take a checkpoint and (2) writing functions to save and restore checkpoint state. The library provides checkpoint management support, i.e. fault detection, checkpoint storage management, and recovery of applications. Furthermore, users are responsible for taking checkpoints at a consistent point in the program. In the general case, this requirement would overwhelm most programmers. However, in SPMD applications, there are natural places to take checkpoints, i.e. at the top or bottom of the main loop [2].

10.1.10.1 Example

To use the checkpoint library, users are responsible for the following tasks: (1) recovery of checkpoint data, (2) saving the checkpoint data, and (3) deciding how often to checkpoint.

The following example consists of tasks that perform some work and exchange boundary information at each iteration.

Example *Sample uses of the checkpoint library*

```
//
// Save state. State consists of the iteration count.
//
do_checkpoint(int iteration) {
    // pack state into a buffer
        MPI_Pack(&iteration, 1, MPI_INT, user_buffer,
            1000, &position, MPI_COMM_WORLD);
```

14 Support for generic communication patterns will be provided in a future release of Legion.

```
    // save buffer into checkpoint
       MPI_FT_Save(user_buffer, 20);

    // done with the checkpoint
       MPI_FT_Save_Done();
}
//
// State consists of the iteration count
//
do_recovery(int rank, int &start_iteration) {
    // Restore buffer from checkpoint
       size = MPI_FT_Restore(user_buffer, 1000);

    // Extract state from buffer
       MPI_Unpack(user_buffer, 64, &position,
          &start_iteration, 1,
       MPI_INT, MPI_COMM_WORLD);
}

//
// C pseudo-code
// Simple SPMD example
//
// mpi_stencil_demo.c
//
main(int argc, char **argv) {
    // declaration of variables omitted...

    // MPI Initialization
    MPI_Init (&argc, &argv);
    MPI_Comm_rank (MPI_COMM_WORLD, &nodenum);
    MPI_Comm_size (MPI_COMM_WORLD, &nprocs);

    // Determine if we are in recovery mode
    recovery = MPI_FT_Init(nodenum, start_iteration);
    if (recovery)
       do_recovery(nodenum, start_iteration);
    else
       start_iteration = 0;

    for (iteration=start_iteration;
        iteration < NUM_ITERATIONS; ++iteration) {

    exchange_boundary();

    // ...do some work here...

    // take a checkpoint every 10th iteration
    if (iteration%10==0)
       do_checkpoint(iteration);
    }
}
```

The first function called is `MPI_FT_Init()`. `MPI_FT_init()` initializes the checkpointing library and returns TRUE if the object is in

recovery mode. If in recovery mode, we restore the last consistent checkpoint. Otherwise, we proceed normally and periodically take a checkpoint.

In `do_checkpoint()` we save state by packing variables into a buffer using `MPI_Pack()` and then calling `MPI_FT_Save()`. To save more than one data item, we can pack several items into a buffer (by repeatedly calling `MPI_Pack`) and then call `MPI_FT_Save()`. When the state to be saved is very large, we can break it down into smaller chunks and call `MPI_FT_Save()` for each chunk. When all data items for the checkpoints have been saved, we call `MPI_FT_Save_Done()`.

In `do_recovery()` we recover the state. Note that the sequences of `MPI_Unpack()` and `MPI_FT_Restore()` must be mirror images of the sequences of `MPI_Pack()` and `MPI_FT_Save()` in `do_checkpoint()`.

10.1.10.2 API (C & Fortran)

C API	Fortran API	Description
int MPI_FT_Init(int rank)	MPI_FT_INIT(RANK, RECOVERY) INTEGER RANK, RECOVERY	Initializes the checkpointing library. Returns TRUE if in recovery mode
int MPI_FT_ON()	MPI_FT_ON(FT_IS_ON) INTEGER FT_IS_ON	Returns TRUE if this application is running with checkpointing turned on.
void MPI_FT_SAVE(char *buffer, int size)	MPI_FT_SAVE(BUFFER, SIZE,IERR) <type> BUFFER(*) INTEGER SIZE, RET_SIZE	Saves checkpoint onto Checkpoint Server.
void MPI_FT_SAVE_DONE()	MPI_FT_SAVE_DONE(IERR) INTEGER IERR	Done with this checkpoint.
int MPI_FT_RESTORE(char *buffer, int size)	MPI_FT_RESTORE(BUFFER,SIZE, RET_SIZE) <type> BUFFER(*) INTEGER SIZE, RET_SIZE	Restore data from current checkpoint.

10.1.10.3 Running the above example

First, run the `legion_ft_initialize` script. You should see the following output:

```
$ legion_ft_initialize
legion_ft_initialize: Could not find a CheckpointStorage object ...
Attempting to register one
Stateful class "/CheckpointStorage" does not exist - creating it.
Registering implementation for class "/CheckpointStorage"
legion_ft_initialize: success initializing FT SPMD Checkpointing
support
$
```

Note that you only need to run this command once. Compile the application and register the program with Legion:

```
$ cd $LEGION_HPC/src/Examples/MPI/SPMD-Checkpointing
$ make reg_stencil
```

Create an instance of the CheckpointStorage server and place it in context space:

```
$ legion_create_object /CheckpointStorage /home/<user_name>/Foo
```

There is a set of special `legion_mpi_run` flags for running in a fault tolerant mode.

`-ft`	Turn on fault tolerance.
`-s` <checkpoint server>	Specifies the checkpoint server to use.
`-R` <checkpoint server>	Recovery mode.
`-g` <ping interval>	Ping interval. Default is 90 seconds.
`-r` <reconfiguration interval>	When failure is detected (an object has not responded in the last *x* seconds) restart the application from the last consistent checkpoint. Default value is 360 seconds.

These flags are used in specific combinations.

- You must use `-ft` in **all** cases
- You must use **either** `-s` or `-R`
- The `-g` and `-r` flags are optional

Using these rules, run the application and specify a `ping` interval and reconfiguration time:

```
$ legion_mpi_run -n 2 -ft -s /home/<user_name>/Foo -g 200 \
  -r 500 mpi/programs/mpi_stencil_demo
```

10.1.10.4 Recovering from failure

If an object fails (does not respond in `<reconfigurationInterval>` seconds) the application automatically restarts itself.

10.1.10.5 Restarting application

Under catastrophic circumstances (such as prolonged network outages), the recovery protocol described above may not work. In this case users can restart (i.e., rerun) the entire application with the `-R` flag.[15] Continuing the above example:

15 The restart mechanism can also be used to migrate applications by specifying a different set of hosts with the `-h` flag.

```
$ legion_mpi_run -n 2 -ft -R /home/<user_name>/Foo -g 200 \
    -r 500 mpi/programs/mpi_stencil_demo
```

The application will restart from the last consistent checkpoint. Note that the number of processes and checkpoint server name must match that of the original run.

Once the application has successfully completed you should delete the Checkpoint Server by typing:

```
$ legion_rm -f /home/username/ss
```

If you would like to test the restart mechanism, you can simulate failure then restart the application. Kill the application by typing `^-C`.

10.1.10.6 Compiling/makefile

To compile C applications with the checkpointing library, link your application with the -lLegionMPIFT -lLegionFT libraries. For Fortran applications, link with the -lLegionMPIFTF library also.

10.1.10.7 Limitations

We use `pings` and timeout values to determine whether an object is alive or dead. If the timeout values are set too low, we may declare an object dead falsely. We recommend setting the `ping` interval and reconfiguration time to conservatively high values.

We do not handle file recovery. When running applications that write data to files, users are responsible for recovering the file pointer.

10.1.11 Functions supported

Please note that data packed/unpacked using the `MPI_Pack` and `MPI_Unpack` functions will not be properly converted in heterogeneous networks.

All MPI 1.1 functions are currently supported in Legion MPI. Some 2.0 functions are supported: you can e-mail <legion-help@virginia.edu> for more information.

10.1.12 Running a Legion MPI code with the fewest changes

Please see the Legion web site for an on-line tutorial called “Running an MPI code in Legion with the fewest changes.” The tutorial shows how to adapt a sample MPI program to run in Legion. Please note, however, that while this “fewest changes” approach allows you to run the program and to transparently read and write files remotely, it does not support heterogeneous conversions of data. If you run this program on several machines which have different integer formats, such as Intel

PCs (little-endian) and IBM RS6000s (big-endian), using unformatted I/O will produce surprising results. If you want to use such a heterogeneous system, you will have to either use formatted I/O (all files are text) or use the “typed binary” I/O calls instead of Fortran READ and WRITE statements. These “typed binary” I/O calls are discussed in “Buffered I/O library, low impact interface,“ section 6.2.4 in the Legion Developer Manual.

10.2 Native MPI

Native MPI code is code written for an MPI implementation. Legion supports running native MPI programs without any changes. You only need the binary and a host to run it on. You can, if you wish, adapt your program to make Legion calls.

You will need a Legion host object with native MPI properties set to run these programs. Please see section 15.0 in the System Administrator manual for more information.

10.2.1 Task classes

Native MPI implementations generally require that MPI executables reside in a given place on disk. We have provided a tool, `legion_native_mpi_register`, to register executables of different architectures for use with native MPI.

10.2.2 Compilation

Native MPI code may be compiled independently of Legion, unless your code makes Legion calls (see section 10.2.5 on page 68). In that case, you must link your program to the Legion libraries. A sample makefile for this situation is in Figure 11.

Figure 11: Sample makefile for native MPI code that makes Legion calls

```
CC   =    mpiCC
MPI_INC = /usr/local/mpich/include

mpimixed:    mpimixed.c
  $(CC) -g -I$(MPI_INC) -I$(LEGION)/include -D$(LEGION_ARCH) -DGNU \
    $(LEGION)/lib/$(LEGION_ARCH)/$(CC)/libLegion1.$(LEGION_LIBRARY_VERSION).so \
    $(LEGION)/lib/$(LEGION_ARCH)/$(CC)/libLegion2.$(LEGION_LIBRARY_VERSION).so \
    $(LEGION)/lib/$(LEGION_ARCH)/$(CC)/libLegion1.$(LEGION_LIBRARY_VERSION).so \
    $(LEGION)/lib/$(LEGION_ARCH)/$(CC)/libBasicFiles.so $< -o $@
```

10.2.3 Register compiled tasks

Run `legion_native_mpi_register`. Usage of this tool is:

```
legion_native_mpi_register <class name>
   <binary path> <architecture>
   [-help]
```

The example below registers a Charmm binary at /myMPIprograms/charmm as using a Linux architecture.

```
$ legion_native_ mpi_register charmm \
  /myMPIprograms/charmm linux
```

You can run register a program multiple times, perhaps with different architectures or different platforms. If you have not registered this program before, this will create a context in the current context path (in this case, the context will be called `charmm`) and registers the name in Legion context space.

10.2.4 Running a native MPI application

MPI programs are started via `legion_native_mpi_run`. Usage is:

```
legion_native_mpi_run
   [-v] [-a <architecture>]
   [-h <host context path>]
   [-IN <local input file>]
   [-OUT <local result file>]
   [-in <Legion input file>]
   [-out <Legion result file>]
   [-n <nodes>] [-t <minutes>]
   [-legion] [-help] [-debug]
   <program context path>
   [<arg 1> <arg 2> ... <argn>]
```

The following parameters are used for this command:

-h <host context path>	Specify the host on which the program will run. The default setting is the system's default placement, which is to pick a compatible host and try to run the object. If the host fails, the system will try another compatible host.
-v	Verbose option.
-a <architecture>	Specify the architecture on which to run.
-IN <local input file>	Specify an input file that should be copied to the remote run from the local file system.

-OUT <local result file>	Specify a result file that should be copied back from the remote run to the local file system.
-in <Legion input file>	Specify an input file that should be copied to the remote run from the Legion context space.
-out <Legion result file>	Specify a result file that should be copied out from the remote run into Legion context space.
-n <nodes>	Specify the number of nodes for the remote MPI run.
-t <minutes>	Specify the amount of time requested for the remote MPI run. This option is only meaningful if the host selected to run the remote program enforces time limits for jobs.
-legion	Indicate whether the application makes Legion calls (see section 10.2.5, below).
<arg1> <arg2> ... <argn>	Arguments to be passed to the remote MPI program.

You can examine the running objects of your application using

```
$ legion_ls program_name
```

This context will have an entry for each object in this MPI application.

To kill the program and its implementations, run:

```
$ legion_rm program_name
```

10.2.5 Making Legion calls from native MPI programs

The `-legion` flag indicates that your MPI code makes Legion calls (e.g., `BasicFile` calls). If you do not use this flag and your code attempts to make Legion calls, your program may not run correctly. If your program makes Legion calls you must:

- link your program with the Legion libraries (see section 10.2.2 on page 66)
- specify the `-legion` flag when you run the code

10.2.6 Example

We have provided two sample native MPI programs, available in $LEGION_HPC/src/Examples/MPI/. The first, nativempi.c, produces exactly the same output as the mpitest_c.c program discussed in section 10.1.6 on page 55. The second, mixedmpi.c, is the same code but with Legion calls.

Please note two important adaptations that were made to mixedmpi.c in order to access Legion files. There are two new include files:

```
#include "legion/Legion_libBasicFiles.h"
#include "legion/LegionNativeMPIUtils.h"
```

and a new function call:

```
LegionNativeMPI_init(&argc, &argv);
```

These lines must be added to a native MPI code that makes any kind of Legion call.

10.2.7 Scheduling native MPI processes

Legion does not schedule native MPI processes. When a program is run with `legion_native_mpi_run`, the local `mpirun` utility on the destination host schedules the processes.

11.0 Replaying & debugging applications

You may need to debug or analyze a Legion application in order to improve performance, find errors, increase fault tolerance, etc. There are two command-line tools that allow you to record and replay events associated with objects started by a particular application. Use `legion_record` to record the application's events in a local file and `legion_replay` to replay those events, so that you can analyze and debug the application's objects.

The `legion_record` utility executes your application in record mode so that it can later be played back with `legion_replay`. All objects started by the application will record all relevant event activity in a local file. This utility also monitors the application and reports if it dies.

The `legion_reply` utility is used in tandem with `legion_record` to debug a particular application. It starts a debugging session for an object started in the application.

Before running these utilities, however, you may want to identify which class objects will be instantiated by the application and assign them a `common_name` attribute, to make it easier to identify their instances. I.e., if you want to debug application `Foo`, which instantiates classes `Bar` and `FooFoo`, you should run:

```
$ legion_update_attributes /class/Bar -a \
  "common_name('Bar')"
$ legion_update_attributes /class/FooFoo -a \
  "common_name('FooFoo')"
```

11.1 Sample record and replay

Here is the output from a record run with an application called AppClient. There are two objects involved: the AppClient object asks for the two numbers and passes them on to the TargetObject, which then divides the two numbers and prints the result. The TargetObject class has already been assigned a `common_name` attribute of `TargetObject`.

The `legion_record` command starts the application and puts the results in a file called `DebugInfo`. The application hits an error in the third calculation.

```
$ legion_record -uf DebugInfo AppClient
Initting Legion.
About to try and create the target object.
Enter two numbers to divide: 20 5
The answer is 4
```

```
Do you want to calc. some more (0 = no, 1 = yes)? 1
Enter two numbers to divide: 16 4
The answer is 4
Do you want to calc. some more (0 = no, 1 = yes)? 1
Enter two numbers to divide: 9 0
(Recorder detected an object death. Cleaning up.)
$
```

We can then use `legion_replay` to see exactly what happened. First, we run it with `-list` to see a summary of the session. The output shows the debug session name, ending status, and identification, and the two objects' session numbers and final status.

```
$ legion_replay -uf DebugInfo -list
Debug Session Name: 679982030
Session Status: An Object Died
_____________________________
Session Number:     135511016
Object Identification: 1.376a2f24.06.4cb9001e.0000...
Session Status: Closed

Session Number:     134735824
Object Identification: TargetObject
Session Status: Object down/died
$
```

The first object, the AppClient, is identified by its LOID but the second, the TargetObject, is identified by its `common_name` attribute, `TargetObject`.

We then run `legion_replay` again, this time to debug the TargetObject. Since no debugger is specified, the GNU gdb is used.[16] The output shows that the TargetObject fails when asked to divide by zero.

```
$ legion_replay -uf DebugInfo -local 134735824
(gdb) run
Starting program: /home/localtmp/mmm2a/OPR/Cached-
TargetObject-Binary-1.16
IN LegionPersistentBufferDir::inflate_universal
About to create buffer MayI
About to create buffer InvocationStore
About to create buffer LegionLibraryState
Warning: no tty object found in legion_tty_init

Program received signal SIGFPE, Arithmetic exception.
0x804cc96 in DoDivide__FGt4LRef1Z14LegionWorkUnit
(wu={value = 0xbfffecfc,
   baseValue = 0x8051494, flags = -88 '('})
  at
/home/localtmp/mmm2a/Legion/src/ServiceObjects/
Debugger/TargetObject.c:196
```

16 GNU gdb 4.17.0.4 with Linux/x86 hardware watchpoint and FPU support. Copyright 1998 Free Software Foundation, Inc. This GDB was configured as "i386-redhat-linux".

```
196       Result = Parm1 / Parm2;
(gdb) p Parm2
$1 = 0
(gdb)
```

Appendices

A-1.0 Sample makefile

The sample makefile in Figure 12, below, is for both legacy and Legion-aware programs that will be executed with either `legion_run` or `legion_run_multi`. The sample started with a legacy C++ code, gnomad.c, and modified it to use the Legion I/O libraries. This allows the application to manipulate context space directly rather than copying a lot of files around. A copy of the original program was kept in the file gnomad_orig.c.

The makefile has targets for five platforms: `intel`, `sgi`, `solaris`, `ibm`, and `alpha_linux`. The `make` targets are slightly different for each machine because the c++/g++ compiler sits in different places and may not be found by the default search path. Therefore, we hard-wired each target to known paths. If these paths don't work for you, log on to a compilation box and get the path directly.

Each make target uses `legion_make` to start a make on the desired platform. The `make` target then compiles and registers both the pure legacy code and the Legion-aware code. After each code is compiled, the resulting binary is registered in Legion space in the desired directory (here in `/home/grimshaw`).

Figure 12: Sample makefile for legacy and Legion-aware programs

```
CC    = g++

include $(LEGION)/src/macros/$(LEGION_ARCH).macros

include $(LEGION)/src/macros/$(LEGION_ARCH).$(CC).macros

CFLAGS = -I$(LEGION)/include -L$(LEGION)/lib/$(LEGION_ARCH)/g++
LIB = -lBasicFiles -lLegionClientStubs $(LIB_LEGION_LFLAGS)

local:
    g++ -O $(CFLAGS) gnomad.c -o gnomad $(LIB)
    g++ -O $(CFLAGS) gnomad_orig.c -o gnomad_orig $(LIB)

intel: gnomad.c
    legion_make -a linux usr_path
    touch intel

all_platforms: solaris ibm intel alpha_linux sgi
    make intel alpha_linux solaris sgi

gnu_path:
    /gnu/bin/g++ -O $(CFLAGS) gnomad.c -o gnomad $(LIB)
```

```
        legion_register_program /home/grimshaw/gnomad ./gnomad $(LEGION_ARCH)
        /gnu/bin/g++ -O $(CFLAGS) gnomad_orig.c -o gnomad_orig $(LIB)
        legion_register_program /home/grimshaw/gnomad_orig ./gnomad_orig $(LEGION_ARCH)

usr_local_path:
    /usr/local/bin/g++ -O $(CFLAGS) gnomad.c -o gnomad $(LIB)
    legion_register_program /home/grimshaw/gnomad ./gnomad $(LEGION_ARCH)
    /usr/local/bin/g++ -O $(CFLAGS) gnomad_orig.c -o gnomad_orig $(LIB)
    legion_register_program /home/grimshaw/gnomad_orig ./gnomad_orig $(LEGION_ARCH)

usr_path:
    /usr/bin/g++ -O $(CFLAGS) gnomad.c -o gnomad $(LIB)
    legion_register_program /home/grimshaw/gnomad ./gnomad $(LEGION_ARCH)
    /usr/bin/g++ -O $(CFLAGS) gnomad_orig.c -o gnomad_orig $(LIB)
    legion_register_program /home/grimshaw/gnomad_orig ./gnomad_orig $(LEGION_ARCH)

solaris:    gnomad.c
    legion_make -a solaris -e /gnu/bin/make gnu_path
    touch solaris

ibm:  gnomad.c
    legion_make -a rs6000 -e /gnu/bin/make gnu_path
    touch ibm

sgi:  gnomad.c
    legion_make -h /hosts/bluebox.cs.virginia.edu -e /gnu/bin/make gnu_path
    touch sgi

alpha_linux:  gnomad.c
    legion_make -a alpha_linux usr_local_path
    touch alpha_linux

register:
    legion_register_program /home/grimshaw/gnomad ./gnomad $(LEGION_ARCH)

clean:
    rm -f gnomad solaris ibm sgi intel alpha_linux

run:
    legion_run -IN ribo.x -IN ribo.d -IN params -OUT tmaplow -OUT tmapfinal gnomad ribo
```

A-1.0 About Legion tty objects

A Legion tty object directs output from Legion processes towards a specific shell. This means that you can see the output from a remotely executed program on your workstation, direct output from a locally executed program into a file, or perhaps view the output from several simultaneously executed programs in a single shell. Generally, you will want one tty object per shell, so that each shell's output will appear in that shell, regardless of where its processes are run.

There are three steps involved in using Legion tty objects:

1. ***Create*** an instance of the ttyObjectClass.
2. ***Set*** a tty object for the shell. This means that all output resulting from processes run in the shell will be directed towards the set tty object. Only one tty object can be set per shell.
3. ***Watch*** the output that arrives at a tty object. This involves creating a special tty watch object, which runs a `tty_watch` process. This process constantly watches its assigned tty object and sends a copy of any input the tty object receives to standard output in its shell. Note that a shell does not have to watch its set tty object, and that multiple shells can watch the same tty object.

Note that when you exit from a shell the `tty_watch` process will not automatically shut down. That is, it will continue to send information to the now defunct shell. This will bog down the tty process and may result in error messages. We therefore suggest that before you exit a shell you stop the `tty_watch` process.

There are two different ways to create and use tty objects in Legion, a simple approach, which will probably be acceptable for most users, and a complex approach.

A-1.1 Simple tty management

There are two commands for the simple approach: `legion_tty` and `legion_tty_off`. The former creates, sets, and watches a tty object and the latter stops the `tty_watch` process, so that the output from the shell will no longer be directed towards the shell. Note that this approach sets and watches a tty object in a single shell. If you wish to set a tty object in one shell and watch it in another you should use the complex approach (section A-1.2 on page 76).

Usage for `legion_tty` is:

`legion_tty <context path>`

You may want to create a specific context to hold your tty objects. The example below starts an object in the `/tty_objects` context.

```
$ legion_tty /tty_objects/mytty
```

If `mytty` does not already exist, a new tty object will be created and assigned the context path name `/tty_objects/mytty`. The output from all processes run in your current shell, whether on your local system or a foreign system, will now be directed to `mytty`, and a `tty_watch` process will start. This process will watch its assigned tty object and pass a copy of the tty object's information to the shell's standard output (i.e., to the shell's command line). Note that a shell can run only one `tty_watch` process at a time.

To stop the `tty_watch` process, run the `legion_tty_off` tool. This will end whatever `tty_watch` process is currently running in the current shell, although it will not destroy the tty object being watched.

A-1.2 Complex tty management

The complex procedure requires three different commands to create, set, and watch the tty object.

First, if you do not already have a tty object you must use `legion_create_object` to create a tty object. Then use `legion_set_tty` to set a tty object for your shell. Remember that you can only have one tty object set per shell. If a tty object has already been set, `legion_set_tty` will just switch to the new one. Finally, use `legion_tty_watch` to start watching your current (set) tty object. The whole procedure looks something like this:

```
$ legion_create_object /class/ttyObjectClass \
  /context_path/mytty
$ legion_set_tty /context_path/mytty
$ legion_tty_watch &
```

The `legion_tty_watch` command can be run with or without a tty object's name: multiple `tty_watches` can be run, and can be started at different times. They can be piped to a file, or run in a different window, as desired. To stop the `tty_watch` procedure, send a SIGINIT, using either "kill -INT" or the Ctrl-C key combination.

If you wish to create and set a tty object in one shell and watch it in another (see shell A's output in shell B), you should run `legion_tty_watch` in shell B.

E.g., in shell A, run:

```
$ legion_create_object /class/ttyObjectClass \
  /tty_objects/mytty
$ legion_set_tty /tty_objects/mytty
```

In shell B, run:

```
$ legion_tty_watch /tty_objects/mytty &
```

To direct the current tty object's output to a file object instead of standard output, use the `legion_tty_redirect` command. The example below redirects the current tty's object output to file object `Foo`. If no file object of that name exists, Legion will create a new one.

```
$ legion_tty_redirect Foo
```

Use `legion_tty_unredirect` to stop this process.

A-1.0 Alphabetical list of Legion commands

Full documentation of these commands can be found in the Reference Manual, in section 2.0 on page 7. There are also man pages for all Legion commands included in the system release package, and on-line tutorials on the Legion web site (<http://legion.virginia.edu>).

```
legion_activate_instances
   {[-c] <class context path> | -l <class LOID>}
   [-debug] [-help]

legion_activate_object
   {[-c] <object context path> | -l <object LOID>}
   [-debug] [-help]

legion_add_acl
   [[-c] <object context path> | -l <object LOID>] {-s | <filename>}
   [-debug] [-help]

legion_add_host_account
   {[-c] <host object context path> | -l <host object LOID>}
   {[-f <mapping file name>] | [<Unix user id>
     [-l <owner LOID> | -c <owner context path>]]}
   [-debug] [-help]

legion_add_implicit_params
   [[-c] <AuthenticationObject path> | -l <AuthenticationObject LOID>]
   {-s | <filename>}
   [-debug] [-help]

legion_add_class_mapping
   <metaclass LOID> <class LOID>
   [-debug] [-help]

legion_add_sub_collection
   {-c <collection path> | -l <collection LOID>}
   {-c <member path> | -l <member LOID>} [-q <query>]

legion_allow_activation
   [-entire_class] [-debug] [-help]
   {[-c] <context path> | -l <LOID>}

legion_bfs
   <file> [-o <out file>] [-BackEnd] [-help]
```

```
legion_cat
   <context path1> <context path2> ... <context pathN>
   [-debug] [-help]

legion_cd
   <context path>
   [-debug] [-help]

legion_change_acl
   {[-c] <object context path> | -l <object LOID>}
   <function pattern> <modifications>

legion_change_owner
   [-v] [-r] {[-c] <object context path> | -l <object LOID>}
   {[-c] <target owner context path> | -l <target owner LOID>}
   [-debug] [-help]

legion_change_permissions
   [+-rwx] [-v] <group/user context path> <target context path>
   [-debug] [-help]

legion_class_host_list
   [[-c] <class context path> | -l <class LOID>]
   [{-a | -d | -t} <host1> <host2> ... <hostn>] [-p]
   [-debug] [-help]

legion_class_vault_list
   [[-c] <class context path> | -l <class LOID>}
   [{-a | -d | -t} <vault1> <vault2> ... <vaultn>] [-p]
   [-debug] [-help]

legion_classof
   {[-c] <object context path> | -l <object LOID>}
   [-debug] [-help]

legion_combine_domains
   [-v] <list of domain cookie files>
   [-debug] [-help]

legion_config_scheduler
   {[-c] <Scheduler path> | -l <Scheduler LOID>}
   [-get_enactor] [-get_collection]
   [-set_enactor {[-c] <Enactor path> | -l <Enactor LOID>}]
   [-set_collection {[-c] <Collection path> | -l <Collection LOID>}]
   [-debug] [-help]

legion_configure_profile
```

```
legion_context_add
   <object LOID> <context name>
   [-debug] [-help]

legion_context_create
   <context path>
   [-debug] [-help]

legion_copy_class
   [-v] [-h]
   {[-c <template context> | -l] <template LOID>}
   <new class path> [-a <attribute>]*
   {-mc [-c <metaclass context> | -l <metaclass LOID>]}

legion_cp
   [-r] [-v] [-m] [-p] [-localsrc][-localdest]
   [-V <vault context path>]
   <source path> <destination path>
   [-debug] [-help]

legion_create_class
   [[-c] <context path>] [-sc <scheduler context path>]
   [-sl <scheduler LOID>] [-debug] [-help]

legion_create_implementation
   <binary path name> <architecture>
   {[-c] <class context path> | -l <class LOID>}
   [[-c] <object context path>] [-nc] [-v] [-a <attribute>]
   [-debug] [-help]

legion_create_object
   {[-c] <class context path> | -l <class LOID>}
   <new object context path>
   [-h <host name on which to place new object>]
   [-v <vault on which to place new object>]
   [-H <context path of preferred host class>]
   [-V <context path of preferred vault class>]
   [-Ch <context containing list of preferred hosts>]
   [-Cv <context containing list of preferred vaults>]
   [-d <recorder context path> <debug session name>]
   [-debug] [-help]

legion_create_object_r
   {[-c] <class context> | -l <class LOID>} <new object context>
   <host name> <host architecture> <$LEGION> <$LEGION_OPR>
   <$LEGION_OPA> <binary path> [<user id>]
   [-debug] [-help]

legion_create_stat_tree
   <base context path> [-debug] [-help]
```

```
legion_create_user
   <user id> [-debug] [-help]

legion_create_user_object
   {[-c] <class context path> | -l <class LOID>}
   [-h <host for new object> | -v <vault for new object>]
   [-f <implicit parameter file>] [-z <password>] <user name>
   [-debug] [-help]

legion_deactivate_instances
   [-stay_down] [-debug] [-help]
   {[-c] <class context name> | -l <class LOID>}

legion_deactivate_object
   [stay_down] [-debug] [-help]
   {[-c] <object context name> | -l <object LOID>}

legion_destroy_instances
   {[-c] <class context path> | -l <class LOID>}
   [-debug] [-help]

legion_destroy_host
   [-v] {[-c] <host context path> | -l <host LOID>}
   [-debug] [-help]

legion_destroy_object
   {[-c] <object context path> | -l <object LOID>}
   [-debug] [-help]

legion_destroy_vault
   {[-c] <vault context path> | -l <vault LOID>}
   [-debug] [-help]

legion_direct_output
   {[-c] <object path> | -l <object LOID>}
   {[-c] <tty context path> | -l <tty LOID>}
   [-debug] [-help]

legion_export_dir
   [-v] [-help] [-noserver] [-autorehash]
   <local directory path> <target context path>

legion_export_dir_quit
   <target context path>

legion_export_dir_rehash
   <target context path>
```

```
legion_exports_interface
   {[-c] <context path> | -l <LOID>}
   {-w <well-known class type> | -f <function signature>}+
   [-debug] [-help]

legion_ft_initialize

legion_FTPd
   [-help] [-v] [-p <port number>] [-a] [-m <max connections>]
   [-t <transfer block size>] [-o <max outstanding invocations>] [-d]
   [-w] [-e] [-rt <RMI timeout>] [-dt <detail timeout>]

legion_generate_domain_cookie
   [-o <cookie output filename>
   [-debug] [-help]

legion_get_accounting_data
   [-v] [-help] [-debug]
   {<pull-source1> <pull-source2> ... <pull-sourcen>}
   {-d <local target directory>}
   [-ot <seconds>] [-it <seconds>]

legion_get_acl
   [[-c] <context path> | -l <LOID>]
   [-debug] [-help]

legion_get_host
   {[-c] <object context path> | -l <object LOID>}
   [-debug] [-help]

legion_get_implicit_params
   [[-c] <context path> | -l <LOID>]
   [-debug] [-help]

legion_get_interface
   {[-c] <context path> | -l <class LOID>}
   [-debug] [-help]

legion_get_vault
   {[-c] <object context path> | -l <object LOID>}
   [-debug] [-help]

legion_host_stats
   {[-c] <host context path> | -l <host LOID>}
   [-debug] [-help]

legion_host_vault_list
   {[-c] <host context path> | -l <host LOID>]
   [{-a | -d | -t} <vault1> <vault2> ... <vaultn>] [-p]
   [-debug] [-help]
```

```
legion_init_arch
   [-debug] [-help]

legion_init_Apps

legion_init_Extra

legion_init_HPC

legion_init_security

legion_initialize

legion_instance_host_list
   {[-c] <context path> | -l <LOID>}
   [{-a | -d | -t} <host1> <host2> ... <hostn>] [-p]
   [-debug] [-help]

legion_instance_vault_list
   {[-c] <context path> | -l <LOID>}
   [{-a | -d | -t} <vault1> <vault2> ... <vaultn>] [-p]
   [-debug] [-help]

legion_join_collection
   {[-c] <Collection path> | -l <Collection LOID>}
   {[-c] <member path> | -l <member LOID>}
   [-debug] [-help]

legion_leave_collection
   {[-c] <Collection path> | -l <Collection LOID>}
   {[-c] <member path> | -l <member LOID>}
   [-debug] [-help]

legion_link
   [-CC <compiler>] [-Fortran] [-FC <compiler>] [-pvm] [-mpi]
   [-L<library path>] [-l<library>]
   [-v] [-o <output file>]
   [-bfs] <object file list>
   [-debug] [-help]

legion_list_attributes
   {[-c] <object context path> | -l <object LOID>}
   [-L] [<attribute name>]
   [-debug] [-help]

legion_list_domains
   [-debug] [-help]
```

```
legion_list_host_accounts
   {[-c] <host object context path> | -l <host object LOID>}
   <user id>
   [-debug] [-help]

legion_list_implementations
   [-v] {[-c] <class context path> | -l <class LOID>}
   [-debug] [-help]

legion_list_instances
   {[-c] <context path> | -l <class LOID>}
   [-debug] [-help]

legion_list_invocations
   {[-c] <object context path> | -l <object LOID>}
   [-debug] [-help]

legion_list_names
   {[-c] <object context path> | -l <object LOID>}
   [-debug] [-help]

legion_list_objects
   {[-c] <host context path> | -l <host LOID>}
   [-debug] [-help]

legion_list_oprs
   {[-c] <vault context path> | -l <vault LOID>}
   [-debug] [-help]

legion_list_sub_collections
   {-l <collection LOID> | -c <collection path>}

legion_ln
   <context path> <new alias>
   [-debug] [-help]

legion_login
   [<user id path> | -l <user id LOID>] [-p <password>]
   [-debug] [-help]

legion_logout
   [-debug] [-help]

legion_ls
   [-laLRAdqv] [<context path>]
   [-debug] [-help]
```

```
legion_make
   [-v] [-a <architecture>] [-h <host context path>]
   [-e <make command>] [-OUT <remote file>]
   [<arg1> <arg2> ... <argn>]
   [-debug] [-help]

legion_make_multi
   [-v] [-a <architecture>] [-e <make command>]
   [<arg1> <arg2> ... <argn>]
   [-debug] [-help]

legion_make_schedule
   {-c <class context path> | -l <class LOID>}
   [-n <number of nodes>] [-f <specification file>] [-q <query>]
   [-help]

legion_make_setup_script
   [-o <script basename>] [-OPR <OPR dir name>]
   [-L <$LEGION dir name>]
   [-debug] [-help]

legion_manage_job
   [-help] [-k] [-cp <priority>]
   [-fr] [-nc] [-q <context name>] <ticket number>

legion_manage_queue
   [-help] [-purge]
   [-maxrs <slots>] [-q <context name>]

legion_mkdir
   <context path>
   [-debug] [-help]

legion_mplc
   [<flags>]

legion_modify_parameters
   [-debug] [-help] <argument>

legion_mplc_reg_impl
   <class name> <binary path>
   <stateless | stateful | sequential> <arch>
   [-help]

legion_mpi_debug
   [-q] {[-c] <instance context path>}
   [-help]
```

```
legion_mpi_probe
   [-help] [-debug] [-v[erbose]]
   [-all] [-list]
   [-in <context path name>] [-out <context path name>]
   [-IN <local file name>] [-OUT <local file name>]
   [-showscratch] [-stat <remote file name>]
   <pid context name>

legion_mpi_register
   <class name> <binary local path name> <platform type>
   [-help]

legion_mpi_run
   {-f <options file> [<flags>]} |
    {-n <number of processors> [<flags>] <program class>
      [<arg1> <arg2> ... <argn>]}

legion_mv
   <context path> <new context path>
   [-debug] [-help]

legion_native_mpi_config_host
   [<wrapper>]
   [-debug] [-help]

legion_native_mpi_init
   [<architecture>]
   [-debug] [-help]

legion_native_mpi_register
   <class name> <binary path> <architecture>
   [-help]

legion_native_mpi_run
   [-v] [-a <architecture>] [-h <host context path>]
   [-IN <local input file>] [-OUT <local result file>]
   [-in <Legion input file>] [-out <Legion result file>]
   [-n <nodes>] [-t <minutes>] [-legion]
   [-help] [-debug]
   <program context path> [<arg 1> <arg 2> ... <arg n>]

legion_nq
   [-help] [-w] [-v]
   [-a <arch> ] [-h <host context path>]
   [-in <context name>] [-out <context name>]
   [-IN <local file name> ] [-OUT <local file name>]
   [-f <options file> ] [-novrun] [-t <minutes>]
   [-n <nodes>] [-r <minutes>] [-p <priority>] [-d]
   <program class> [ <arg1> <arg2> ... <argn>]
```

```
legion_object_info
   {[-c] <object context path> | -l <object LOID>} [-v]
   [-debug] [-help]

legion_output_state
   {[-c] <object context path> | -l <object LOID>} | StartupState
   [-debug] [-help]

legion_passwd
   {<user id context> | -l <user id LOID>}
   [-debug] [-help]

legion_ping
   {[-c] <object context path> | -l <object LOID>}
   [-timeout <seconds>] [-debug] [-help]

legion_print_config
   [-debug] [-help]

legion_print_domain_cookie
   [-i <cookie input file>] [-debug] [-help]

legion_probe_run
   [-help] [-debug]
   [-pwd] [-hostname] [-list] [-stat <remote file name] [-statjob]
   [-in <context path name>] [-out <context path name>]
   [-IN <local file name>] [-OUT <local file name>]
   [-setscratch <context path>] [-showscratch] [-kill]
   {-p[robe] <local file name> | -l <probe LOID>}

legion_pvm_register
   <class path name> <binary local path name> <platform type>
   [-help]

legion_pwd
   [-debug] [-help]

legion_query_collection
   [-v] {[-c] <Collection path> | -l <Collection LOID>} <query>
   [-debug] [-help]

legion_record
   {-uf <local storage file name> | [-c] <recorder context path>}
   [-name <debug session name>] [-t <interval>]
   <client application> [<arg1> <arg2> ... <argN>]
   [-debug] [-help]

legion_refresh_local_cache
```

```
legion_remove_host_account
   [-l <host object LOID> | [-c] <host object context path>]
   [-debug] [-help]

legion_replay
   {-uf <local storage file name> |
      [-c] <recorder context path> -name <debug session name>}
   {-list | [-cmd <debugger name>] [-local] <session number>}
   [-debug] [-help]

legion_register_program
   <program class> <executable path> <legion arch>
   [-debug] [-help]

legion_register_runnable
   <program class> <executable path> <legion arch>
   [-debug] [-help]

legion_remove_sub_collection
   {-l <collection LOID> | -c <collection path>}
   {-l <member LOID> | -c <member path>} [-q <query>]

legion_rm
   [-r] [-f] [-v] <context path list>
   [-debug] [-help]

legion_run
   [-help] [-debug] [-v[erbose]] [-w] [-a <arch>] [-h <host>]
   [-block] [-non[-]block] [-d[ir] <dir>] [-D <var=value>]
   [-in <contextfile>] [-out <contextfile>]
   [-IN <localfile>] [-OUT <localfile>]
   [-stdin <localfile>] [-stdout <localfile>] [-stderr <localfile>]
   [-novrun] [-p[robe] <localfile>] [-setscratch <context>]
   [-meta <option=value>]
   [-f <options file>] <program class> [<arg1> ... <argn>]

legion_run_multi
   [-help] [-debug] [-v[erbose]] [-z[ero]] [-r[estart]]
   [-e[xec] command] [-x <exceptfile>] [-a[rch] <architecture>]
   [-d[ir] <dir>] [-D <var=value>]
   {-n <number of processors> | -s[chedule] <schedfile>}
   -f[ile] <specfile> [-t[ime] <timefile>] [--] <program class>
   [<arg1> <arg2> ... <argn>]

legion_set_acl
   {[-c] <object context path> | -l <object LOID>}
   [-s | filename]
   [-debug] [-help]
```

```
legion_set_backup_vaults
   {[-c] <instance context path> | -l <instance LOID>}
   [-nodeac[tivate]]
   {[-a | -d] [-c <vault1 context path> | -l <vault1 LOID>]
        [-a | -d] [-c <vault2 context path> | -l <vault2 LOID>] ...
        [-a | -d] [-c <vaultn context path> | -l <vaultn LOID>]}
    | {-Cv <vault context path> -n <total number of vaults>}

legion_set_binding_agent
   [-unset] [-make_default] [-make_default_only]
   {-l <object context path> | -c <object LOID>}

legion_set_default_placement
   {[-c] <class context path> | -l <class LOID>}
   {[-c] <host context path> | -l <host LOID>}
   {[-c] <vault context path> | -l <vault LOID>}
   [-debug] [-help]

legion_set_host
   {[-c] <instance context name> | -l <instance LOID>}
   {[-c] <host context path> | -l <host LOID>}
   [-debug] [-help]

legion_set_implicit_params
   [[-c] <object context path> | -l <object LOID>]
   [-s | filename]
   [-debug] [-help]

legion_set_message_security
   [-c <AuthenticationObject context path> |
     -l <AuthenticationObject LOID>]
   {Off | Protected | Private} [-help]

legion_set_scheduler
   {[-c] <class context path> | -l <class LOID>}
   {[-c] <Scheduler context name> | -l <Scheduler LOID>}
   [-debug] [-help]

legion_set_scheduler_policy
   {[-c] <class context path> | -l <class LOID>} <policy>
   [-debug] [-help]

legion_set_tty
   <tty context path>
   [-debug] [-help]

legion_set_vault
   {[-c] <instance context path> | -l <instance LOID>}
   {[-c] <vault context path> | -l <vault LOID>} [-permanent]
   [-debug] [-help]
```

```
legion_set_varch
   {[-c] <host context path> | -l <host LOID>} <arch>
   [-debug] [-help]

legion_set_vrun
   {[-c] <host context path> | -l <host LOID>} <path>
   [-debug] [-help]

legion_set_worm
   [[-c] <object context path> | -l <object LOID>]

legion_setup_state
   [-i] [-debug] [-help]

legion_skcc_set_class_vaults
   <SKCC class context path>
   {[-a | -d] [-c <vault1 context path> | -l <vault1 LOID>]
      [-a | -d] [-c <vault2 context path> | -l <vault2 LOID>] ...
        [-a | -d] [-c <vaultn context path> | -l <vaultn LOID>]}

legion_skcc_set_defaults
   {[-l <SKCC class LOID> | -c <SKCC class context path>}
   <number of default vaults>

legion_show_acl
   {[-c] <object context path> | -l <object LOID>}
   [<function pattern>] [-showLoids]

legion_shutdown
   [-local] [-f] [-i]
   [-debug] [-help]

legion_shutdown_class
   {[-c] <context path> | -l <class LOID>}
   [-debug] [-help]

legion_starthost
   [-L <$LEGION>] [-O <$LEGION_OPR>] [-A <$LEGION_ARCH>]
   [-B <path>] [-N <context name>] [-U <user id>] [-C <host class>]
   <new host name> [<compatible vault list>]
   [-debug] [-help]

legion_startup
   [-local]
   [-debug] [-help]

legion_startvault
   [-L <$LEGION>] [-O <$LEGION_OPR>] [-A <$LEGION_ARCH>]
   [-N <context name>] [-U <user id>] [-C <vault class>]
   <host name> [<compatible host list>] [-debug] [-help]
```

```
legion_stateless_add_workers
   <class name> <worker name1> <worker name2> ... <worker nameN>
   [-debug] [-help]

legion_stateless_configure
   <stateless class context path>
   [-n <number of replicas>] [-Ch <host conte1xt path>]
   [-w <max number work requests>] [-FlUsH]
   [-debug] [-help]

legion_stateless_remove_workers
   <class name>
   <worker name1> <worker name2> ... <worker nameN>
   [-debug] [-help]

legion_synch_vaults
   {[-c] <instance context path> | -l <instance LOID>}
   [-nodeac[tivate]]

legion_tty
   <tty context path>
   [-debug] [-help]

legion_tty_off
   [-debug] [-help]

legion_tty_redirect
   <object context path>
   [-debug] [-help]

legion_tty_unredirect
   <object context path>
   [-debug] [-help]

legion_tty_watch
   [[-c] <tty context path> | -l <tty LOID>]
   [-debug] [-help]

legion_2drm
   <file name>

legion_unset_worm
   {[-c] <instance context path> | -l <instance LOID>}

legion_update_accounting_db
   [-v] <local configuration file path>
   <local logfile directory>
```

```
legion_update_attributes
   {[-c] <object context path> | -l <object LOID>}
   [-a <new attribute>] [-d <attribute>]
   [-r <old attribute> <new attribute>]
   [-debug] [-help]

legion_update_stat_tree
   [-v] [-help] [-t <repeat delay seconds>]
   [-site <top site context path>]
   [-collection <collection context path>]

legion_vault_host_list
   {[-c] <vault context path> | -l <vault LOID>}
   [{-a | -d | -t} <host1> <host2> ... <hostn>] [-p]
   [-debug] [-help]

legion_version
   [-debug] [-help]

legion_wellknown_class
   <well-known class name>
   [-debug] [-help]

legion_whereis
   {[-c] <object context path> | -l <object LOID>}
   [-debug] [-help]

legion_whoami
   [-debug] [-help]
```

A-1.0 Subject listing of Legion commands

Full documentation of these commands can be found in the Reference Manual, in section 2.0 on page 7, and on the Legion web site at <http://legion.virginia.edu/documentation.html>. There are also man pages for all Legion commands included in the system release package.

A-1.1 Calls on objects

```
legion_configure_profile

legion_exports_interface
   {[-c] <context path> | -l <LOID>}
   {-w <well-known class type> | -f <function signature>}+
   [-debug] [-help]

legion_FTPd
   [-help] [-v] [-p <port number>] [-a] [-m <max connections>]
   [-t <transfer block size>] [-o <max outstanding invocations>] [-d]
   [-w] [-e] [-rt <RMI timeout>] [-dt <detail timeout>]

legion_get_interface
   {[-c] <context path> | -l <class LOID>}
   [-debug] [-help]

legion_ping
   {[-c] <object context path> | -l <object LOID>}
   [-timeout <seconds>]
   [-debug] [-help]

legion_list_attributes
   {[-c] <object context path> | -l <object LOID>}
   [-L] [<attribute name>]
   [-debug] [-help]

legion_list_invocations
   {[-c] <object context path> | -l <object LOID>}
   [-debug] [-help]

legion_object_info
   {[-c] <object context path> | -l <object LOID>} [-v]
   [-debug] [-help]

legion_set_worm
   [[-c] <object context path> | -l <object LOID>]
```

```
legion_skcc_set_class_vaults
   <SKCC class context path>
   {[-a | -d] [-c <vault1 context path> | -l <vault1 LOID>]
      [-a | -d] [-c <vault2 context path> | -l <vault2 LOID>] ...
        [-a | -d] [-c <vaultn context path> | -l <vaultn LOID>]}

legion_skcc_set_defaults
   {[-l <SKCC class LOID> | -c <SKCC class context path>}
   <number of default vaults>

legion_update_attributes
   {[-c] <object context path> | -l <object LOID>}
   [-a <new attribute>] [-d <attribute>]
   [-r <old attribute> <new attribute>]
   [-debug] [-help]

legion_unset_worm
   {[-c] <instance context path> | -l <instance LOID>}
```

A-1.2 Calls on class objects

```
legion_activate_object
   {[-c] <object context path> | -l <object LOID>}
   [-debug] [-help]

legion_copy_class
   [-v] [-h]
   {[-c <template context> | -l] <template LOID>}
   <new class path> [-a <attribute>]*
   {-mc [-c <metaclass context> | -l <metaclass LOID>]}

legion_create_object
   {[-c] <class context path> | -l <class LOID>}
   <new object context path>
   [-h <host name on which to place new object>]
   [-v <vault on which to place new object>]
   [-H <context path of preferred host class>]
   [-V <context path of preferred vault class>]
   [-Ch <context containing list of preferred hosts>]
   [-Cv <context containing list of preferred vaults>]
   [-d <recorder context path> <debug session name>]
   [-debug] [-help]

legion_create_object_r
   {[-c] <class context> | -l <class LOID>} <new object context>
   <host name> <host architecture> <$LEGION> <$LEGION_OPR>
   <$LEGION_OPA> <binary path> [<user id>]
   [-debug] [-help]
```

```
legion_deactivate_instances
   [-stay_down] [-debug] [-help]
   {[-c] <class context name> | -l <class LOID>}

legion_deactivate_object
   [stay_down] [-debug] [-help]
   {[-c] <object context name> | -l <object LOID>}

legion_destroy_instances
   {[-c] <class context path> | -l <class LOID>}
   [-debug] [-help]

legion_destroy_object
   {[-c] <object context path> | -l <object LOID>}
   [-debug] [-help]

legion_get_host
   {[-c] <object context path> | -l <object LOID>}
   [-debug] [-help]

legion_get_vault
   {[-c] <object context path> | -l <object LOID>}
   [-debug] [-help]

legion_list_implementations
   [-v] {[-c] <class context path> | -l <class LOID>}
   [-debug] [-help]

legion_list_instances
   {[-c] <context path> | -l <class LOID>}
   [-debug] [-help]

legion_refresh_local_cache

legion_set_backup_vaults
   {[-c] <instance context path> | -l <instance LOID>}
   [-nodeac[tivate]]
   {[-a | -d] [-c <vault1 context path> | -l <vault1 LOID>]
        [-a | -d] [-c <vault2 context path> | -l <vault2 LOID>] ...
        [-a | -d] [-c <vaultn context path> | -l <vaultn LOID>]}
    | {-Cv <vault context path> -n <total number of vaults>}

legion_set_binding_agent
   [-unset] [-make_default] [-make_default_only]
   {-l <object context path> | -c <object LOID>}

legion_set_host
   {[-c] <instance context name> | -l <instance LOID>}
   {[-c] <host context path> | -l <host LOID>}
   [-debug] [-help]
```

```
legion_set_vault
   {[-c] <instance context path> | -l <instance LOID>}
   {[-c] <vault context path> | -l <vault LOID>} [-permanent]
   [-debug] [-help]

legion_synch_vaults
   {[-c] <instance context path> | -l <instance LOID>}
   [-nodeac[tivate]]
```

A-1.3 Calls on LegionClass

```
legion_add_class_mapping
   <metaclass LOID> <class LOID>
   [-debug] [-help]

legion_combine_domains
   [-v] <list of domain cookie files>
   [-debug] [-help]

legion_create_implementation
   <binary path name> <architecture>
   {[-c] <context path> | -l <class LOID>}
   [-debug] [-help]

legion_generate_domain_cookie
   [-o <cookie output filename>
   [-debug] [-help]

legion_init_arch
   [-debug] [-help]

legion_list_domains
   [-debug] [-help]

legion_print_domain_cookie
   [-i <cookie input file>
   [-debug] [-help]
```

A-1.4 Calls on file and context objects

```
legion_activate_instances
   {[-c] <class context path> | -l <class LOID>}
   [-debug] [-help]

legion_allow_activation
   [-entire_class] [-debug] [-help]
   {[-c] <context path> | -l <LOID>}
```

```
legion_cat
   <context path>
   [-debug] [-help]

legion_cd
   <context path>
   [-debug] [-help]

legion_context_add
   <object LOID> <context name>
   [-debug] [-help]

legion_context_create
   <context path>
   [-debug] [-help]

legion_cp
   [-r] [-v] [-m] [-p] [-localsrc] [-localdest]
   [-V <vault context path>]
   <source path> <destination path>
   [-debug] [-help]

legion_direct_output
   {[-c] <object path> | -l <object LOID>}
   {[-c] <tty context path> | -l <tty LOID>}
   [-debug] [-help]

legion_export_dir
   [-v] [-help] [-noserver] [-autorehash]
   <local directory path> <target context path>

legion_export_dir_quit
   <target context path>

legion_export_dir_rehash
   <target context path>

legion_get_host
   {[-c] <object context path> | -l <object LOID>}
   [-debug] [-help]

legion_get_vault
   {[-c] <object context path> | -l <object LOID>}
   [-debug] [-help]

legion_list_names
   {[-c] <object context path> | -l <object LOID>}
   [-debug] [-help]
```

```
legion_ln
   <context path> <new alias>
   [-debug] [-help]

legion_ls
   [-laLRAdqv] [<context path>]
   [-debug] [-help]

legion_mkdir
   <context path>
   [-debug] [-help]

legion_mv
   <context path> <new context path>
   [-debug] [-help]

legion_pwd
   [-debug] [-help]

legion_rm
   [-r] [-f] [-v] <context path list>
   [-debug] [-help]

legion_set_tty
   <tty context path>
   [-debug] [-help]

legion_tty
   <tty context path>
   [-debug] [-help]

legion_tty_off
   [-debug] [-help]

legion_tty_redirect
   <object context path>
   [-debug] [-help]

legion_tty_unredirect
   <object context path>
   [-debug] [-help]

legion_tty_watch
   [[-c] <tty context path> | -l <tty LOID>]
   [-debug] [-help]

legion_2drm
   <file name>
```

A-1.5 Start-up and shutdown functions

```
legion_add_host_account
   {[-c] <host object context path> | -l <host object LOID>}
   {[-f <mapping file name>] |
    [<Unix user id> [-c <owner context path> | -l <owner LOID>]]}
   [-debug] [-help]

legion_create_class
   [[-c] <context path>] [-sc <scheduler context path>]
   [-sl <scheduler LOID>]
   [-debug] [-help]

legion_destroy_host
   [-v] {[-c] <host context path> | -l <host LOID>}
   [-debug] [-help]

legion_destroy_vault
   {[-c] <vault context path> | -l <vault LOID>}
   [-debug] [-help]

legion_initialize

legion_list_host_accounts
   {[-c] <host object context path> | -l <host object LOID>}
   <user id>
   [-debug] [-help]

legion_make_setup_script
   [-o <script basename>] [-OPR <OPR dir name>]
   [-L <$LEGION dir name>]
   [-debug] [-help]

legion_print_config
   [-debug] [-help]

legion_remove_host_account
   [[-c] <host object context path> | -l <host object LOID>]
   [-debug] [-help]

legion_setup_state
   [-i]
   [-debug] [-help]

legion_shutdown
   [-local] [-f] [-i]
   [-debug] [-help]
```

```
legion_shutdown_class
   {[-c] <context path> | -l <class LOID>}
   [-debug] [-help]

legion_starthost
   [-L <$LEGION>] [-O <$LEGION_OPR>] [-A <$LEGION_ARCH>]
   [-B <path>] [-N <context name>] [-U <user id>] [-C <host class>]
   <new host name> [<compatible vault list>]
   [-debug] [-help]

legion_startup
   [-local]
   [-debug] [-help]

legion_startvault
   [-L <$LEGION>] [-O <$LEGION_OPR>] [-A <$LEGION_ARCH>]
   [-N <context name>] [-U <user id>] [-C <vault class>]
   <host name> [<compatible host list>]
   [-debug] [-help]
```

A-1.6 Scheduling support

```
legion_add_sub_collection
   {-c <collection path> | -l <collection LOID>}
   {-c <member path> | -l <member LOID>}
   [-q <query>]

legion_class_host_list
   [[-c] <class context path> | -l <class LOID>]
   [{-a | -d | -t} <host1> <host2> ... <hostn>] [-p]
   [-debug] [-help]

legion_class_vault_list
   [[-c] <class context name> | -l <class LOID>}
   [{-a | -d | -t} <vault1> <vault2> ... <vaultn>] [-p]
   [-debug] [-help]

legion_config_scheduler
   {[-c] <Scheduler context name> | -l <Scheduler LOID>}
   [-get_enactor] [-get_collection]
   [-set_enactor {[-c] <Enactor path> | -l <Enactor LOID>}]
   [-set_collection {[-c] <Collection path> | -l <Collection LOID>}]
   [-debug] [-help]

legion_host_vault_list
   {[-c] <host context path> | -l <host LOID>]
   [{-a | -d | -t} <vault1> <vault2> ... <vaultn>] [-p]
   [-debug] [-help]
```

```
legion_instance_host_list
   {[-c] <instance context path> | -l <instance LOID>}
   [{-a | -d | -t} <host1> <host2> ... <hostn>] [-p]
   [-debug] [-help]

legion_instance_vault_list
   {[-c] <instance context path> | -l <instance LOID>}
   [{-a | -d | -t} <vault1> <vault2> ... <vaultn>] [-p]
   [-debug] [-help]

legion_join_collection
   {[-c] <Collection path> | -l <Collection LOID>}
   {[-c] <member path> | -l <member LOID>}
   [-debug] [-help]

legion_leave_collection
   {[-c] <Collection path> | -l <Collection LOID>}
   {[-c] <member path> | -l <member LOID>}
   [-debug] [-help]

legion_list_oprs
   {[-c] <vault context path> | -l <vault LOID>}
   [-debug] [-help]

legion_list_sub_collections
   {-c <collection path> | -l <collection LOID>}

legion_make_schedule
   {-c <class context path> | -l <class LOID>}
   [-n <number of nodes>] [-f <specification file>] [-q <query>]
   [-help]

legion_query_collection
   [-v] {[-c] <Collection path> | -l <Collection LOID>} <query>
   [-debug] [-help]

legion_remove_sub_collection
   {-c <collection path> | -l <collection LOID>}
   {-c <member path> | -l <member LOID>} [-q <query>]

legion_set_default_placement
   {[-c] <class context name> | -l <class LOID>}
   {[-c] <host context name> | -l <host LOID>}
   {[-c] <vault context name> | -l <vault LOID>}
   [-debug] [-help]

legion_set_scheduler
   {[-c] <class context path> | -l <class LOID>}
   {[-c] <Scheduler context name> | -l <Scheduler LOID>}
   [-debug] [-help]
```

```
legion_set_scheduler_policy
   {[-c] <class context path> | -l <class LOID>} <policy>
   [-debug] [-help]

legion_set_varch
   {[-c] <host context path> | -l <host LOID>} <arch>
   [-debug] [-help]

legion_set_vrun
   {[-c] <host context path> | -l <host LOID>} <path>
   [-debug] [-help]

legion_vault_host_list
   {[-c] <vault context path> | -l <vault LOID>}
   [{-a | -d | -t} <host1> <host2> ... <hostn>] [-p]
   [-debug] [-help]
```

A-1.7 General functions about the state of the system

```
legion_classof
   {[-c] <object context path> | -l <object LOID>}
   [-debug] [-help]

legion_create_stat_tree
   <base context path>
   [-debug] [-help]

legion_get_accounting_data
   [-v] [-help] [-debug]
   {<pull-source1> <pull-source2> ... <pull-sourcen>}
   {-d <local target directory>}
   [-ot <seconds>] [-it <seconds>]

legion_host_stats
   {[-c] <host context path> | -l <host LOID>}
   [-debug] [-help]

legion_list_objects
   {[-c] <host context path> | -l <host LOID>}
   [-debug] [-help]

legion_update_accounting_db
   [-v] <local configuration file path> <local logfile directory>

legion_update_stat_tree
   [-v] [-help] [-t <repeat delay seconds>]
   [-site <top site context path>]
   [-collection <collection context path>]
```

```
legion_version
   [-debug] [-help]

legion_wellknown_class
   <well-known class name>
   [-debug] [-help]

legion_whereis
   {[-c] <object context path> | -l <object LOID>}
   [-debug] [-help]

legion_whoami
   [-debug] [-help]
```

A-1.8 Security

```
legion_add_acl
   [[-c] <object context path> | -l <object LOID>]
   {-s | <filename>}
   [-debug] [-help]

legion_add_implicit_params
   [[-c] <AuthenticationObject path> | -l <AuthenticationObject LOID>]
   {-s | <filename>}
   [-debug] [-help]

legion_change_acl
   {[-c] <object context path> | -l <object LOID>}
   <function pattern> <modifications>

legion_change_owner
   [-v] [-r] {[-c] <object context path> | -l <object LOID>}
   {[-c] <target owner context path> | -l <target owner LOID>}
   [-debug] [-help]

legion_change_permissions
   [+-rwx] [-v] <group/user context path> <target context path>
   [-debug] [-help]

legion_create_user
   <user id>
   [-debug] [-help]

legion_create_user_object
   {[-c] <class context path> | -l <class LOID>}
   [-h <host for new object> | -v <vault for new object>]
   [-f <implicit parameter file>] [-z <password>] <user name>
   [-debug] [-help]
```

```
legion_get_acl
   [[-c] <object context path> | -l <object LOID>]
   [-debug] [-help]

legion_get_implicit_params
   [[-c] <object context path> | -l <object LOID>]
   [-debug] [-help]

legion_init_Apps

legion_init_Extra

legion_init_HPC

legion_init_security

legion_login
   [<user id path> | -l <user id LOID>] [-p <password>]
   [-debug] [-help]

legion_logout
   [-debug] [-help]

legion_modify_parameters
   [-debug] [-help] <argument>

legion_passwd
   {<user id context> | -l <user id LOID>}
   [-debug] [-help]

legion_set_acl
   {[-c] <object context path> | -l <object LOID>}
   [-s | <filename>]
   [-debug] [-help]

legion_set_implicit_params
   [[-c] <object context path> | -l <object LOID>]
   [-s | <filename>]
   [-debug] [-help]

legion_set_message_security
   [-c <AuthenticationObject context path> |
     -l <AuthenticationObject LOID>]
   {Off | Protected | Private} [-help]

legion_show_acl
   {[-c] <object context path> | -l <object LOID>}
   [<function pattern>] [-showLoids]
```

A-1.9 Application development

```
legion_bfs
   <file> [-o <out file>] [-BackEnd]
   [-help]

legion_make
   [-v] [-a <architecture>] [-h <host context path>]
   [-e <make command>] [-OUT <remote file>]
   [<arg1> <arg2> ... <argn>]
   [-debug] [-help]

legion_make_multi
   [-v] [-a <architecture>][-e <make command>]
   [<arg1> <arg2> ... <argn>]
   [-debug] [-help]

legion_mplc
   [<flags>]

legion_mplc_reg_impl
   <class name> <binary path>
   <stateless | stateful | sequential> <arch>
   [-help]

legion_output_state
   {[-c] <object context path> | -l <object LOID>} | StartupState
   [-debug] [-help]

legion_record
   {-uf <local storage file name> | [-c] <recorder context path>}
   [-name <debug session name>] [-t <interval>]
   <client application> [<arg1> <arg2> ... <argN>]
   [-debug] [-help]

legion_replay
   {-uf <local storage file name> |
    [-c] <recorder context path> -name <debug session name>}
   {-list | [-cmd <debugger name>] [-local] <session number>}
   [-debug] [-help]

legion_stateless_add_workers
   <class name> <worker name1> <worker name2> ... <worker nameN>
   [-debug] [-help]

legion_stateless_configure
   <stateless class context path> [-n <number of replicas>]
   [-Ch <host context path>] [-w <max number work requests>] [-FlUsH]
   [-debug] [-help]
```

```
legion_stateless_remove_workers
   <class name> <worker name1> <worker name2> ... <worker nameN>
   [-debug] [-help]
```

A-1.10 Program support

```
legion_ft_initialize

legion_link
   [-CC <compiler>] [-Fortran]
   [-FC <compiler>] [-pvm] [-mpi]
   [-L<library path>] [-l<library>] [-v]
   [-o <output file>] [-bfs] <object file list>
   [-debug] [-help]

legion_manage_job
   [-help] [-k] [-cp <priority>]
   [-fr] [-nc] [-q <context name>] <ticket number>

legion_manage_queue
   [-help] [-purge]
   [-maxrs <slots>] [-q <context name>]

legion_mpi_debug
   [-q] {[-c] <instance context path>}
   [-help]

legion_mpi_probe
   [-help] [-debug] [-v[erbose]] [-all] [-list]
   [-in <context path name>] [-out <context path name>]
   [-IN <local file name>] [-OUT <local file name>]
   [-showscratch] [-stat <remote file name>]
   <pid context name>

legion_mpi_register
   <class name> <binary local path name>
   <platform type>
   [-help]

legion_mpi_run
   {-f <options file> [<flags>]} |
    {-n <number of processors> [<flags>] <program class>
      [<arg1> <arg2> ... <argn>]}

legion_native_mpi_config_host
   [<wrapper>]
   [-debug] [-help]
```

```
legion_native_mpi_init
   [<architecture>]
   [-debug] [-help]

legion_native_mpi_register
   <class name> <binary path> <architecture>
   [-help]

legion_native_mpi_run
   [-v] [-a <architecture>] [-h <host context path>]
   [-IN <local input file>] [-OUT <local result file>]
   [-in <Legion input file>] [-out <Legion result file>]
   [-n <nodes>] [-t <minutes>] [-legion]
   [-debug] [-help]
   <program context path> [<arg 1> <arg 2> ... <arg n>]

legion_nq
   [-help] [-w] [-v]
   [-a <arch> ] [-h <host context path>]
   [-in <context name>] [-out <context name>]
   [-IN <local file name> ] [-OUT <local file name>]
   [-f <options file> ] [-novrun] [-t <minutes>]
   [-n <nodes>] [-r <minutes>] [-p <priority>] [-d]
   <program class> [ <arg1> <arg2> ... <argn>]

legion_probe_run
   [-help] [-debug]
   [-pwd] [-hostname] [-list] [-stat <remote file name] [-statjob]
   [-in <context path name>] [-out <context path name>]
   [-IN <local file name>] [-OUT <local file name>]
   [-setscratch <context path>] [-showscratch] [-kill]
   {-p[robe] <local file name> | -l <probe LOID>}

legion_pvm_register
   <class path name> <binary local path name> <platform type>
   [-help]

legion_register_program
   <program class> <executable path> <legion arch>
   [-debug] [-help]

legion_register_runnable
   <program class> <executable path> <legion arch>
   [-debug] [-help]
```

```
legion_run
   [-help] [-debug] [-v[erbose]] [-w] [-a <arch>] [-h <host>]
   [-block] [-non[-]block] [-d[ir] <dir>] [-D <var=value>]
   [-in <contextfile>] [-out <contextfile>]
   [-IN <localfile>] [-OUT <localfile>]
   [-stdin <localfile>] [-stdout <localfile>] [-stderr <localfile>]
   [-novrun] [-p[robe] <localfile>] [-setscratch <context>]
   [-meta <option=value>]
   [-f <options file>] <program class> [<arg1> ... <argn>]

legion_run_multi
   [-help] [-debug] [-v[erbose]] [-z[ero]] [-r[estart]]
   [-e[xec] command] [-x <exceptfile>] [-a[rch] <architecture>]
   [-d[ir] <dir>] [-D <var=value>]
   {-n <number of processors> | -s[chedule] <schedfile>}
   -f[ile] <specfile> [-t[ime] <timefile>] [--] <program class>
   [<arg1> <arg2> ... <argn>]
```

Getting help

Please contact us at one of the addresses listed below if you have trouble with the system. Please be sure to include any relevant information about the state of your system before and during the problem.

We would greatly appreciate hearing from you whenever you find errors or bugs in Legion, so that we can avoid similar problems in future releases.

E-mail

For bug reports, help, and general Legion information, please send an e-mail message to <legion-help@virginia.edu>.

Contact Information

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
http://legion.virginia.edu/
fax: 434-982-2214

On-Line Help

A variety of technical notes, reports, and on-line tutorials are available on the Legion web site at <http://legion.virginia.edu/>.

References

Several of the Legion-related papers listed in this bibliography are available on the Legion web site, at <http://legion.virginia.edu/>.

[1] Ferrari, A.J., M.J. Lewis, C.L. Viles, A. Nguyen-Tuong, and A.S. Grimshaw. “Implementation of the Legion Runtime Library,” University of Virginia CS Technical Report CS-96-16: Nov 1996.

[2] G. A. Geist, J. A. Kohl, P. M. Papadopoulos, “CUMULVS: Providing Fault-Tolerance, Visualization and Steering of Parallel Applications," *International Journal of High Performance Computing Applications*, 11(3): 224-236, Aug 1997.

[3] Geist, A., A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V.S. Sunderam. *PVM: Parallel Virtual Machine*, Cambridge, MA: The MIT Press, 1994.

[4] Grimshaw, A.S., and W.A. Wulf. “The Legion Vision of a Worldwide Virtual Computer” *Communications of the ACM*, 40(1): 39-45, 1997.

[5] Grimshaw, A.S., and W.A. Wulf. “Legion – A View from 50,000 Feet” *Proceedings of the Fifth IEEE Symposium on High Performance Distributed Computing*, Aug 6-9, 1996, Syracuse, NY: IEEE Computer Society Press.

[6] Grimshaw, A.S. “Object-Oriented Parallel Processing with Mentat.” Postscript version available on the Legion web site. URL http://legion.virginia.edu/papers.html

[7] Grimshaw, A.S., A. Nguyen-Tuong, and W.A. Wulf. “Campus-Wide Computing: Results Using Legion at the University of Virginia” University of Virginia CS Technical Report CS-95-19: Mar 1995.

[8] Grimshaw, A.S., W.A. Wulf, J.C. French, A.C. Weaver, and P.F. Reynolds Jr. “Legion: The Next Logical Step Toward a Nationwide Virtual Computer,” University of Virginia CS Technical Report CS-94-21: June 1994.

[9] Grimshaw, A.S., J.B. Weissman, E.A. West, and E. Loyot. “Metasystems: An Approach Combining Parallel Processing and Heterogeneous Distributed Computing Systems” *Journal of Parallel and Distributed Computing*: 21(3): 257-270, June 1994.

[10] Grimshaw, A.S., W.A. Wulf, J.C. French, A.C. Weaver, P.F. Reynolds Jr., “A Synopsis of the Legion Project,” University of Virginia CS Technical Report CS-94-20: June 1994.

[11] Grimshaw, A.S. “Easy to Use Object-Oriented Parallel Programming with Mentat” *IEEE Computer*: May 1993.

[12] J.F. Karpovich. “Support for Object Placement in Wide Area Heterogeneous Distributed Systems,” University of Virginia CS Technical Report CS-96-03: Jan 1996.

[13] Lewis, M.J., and A.S. Grimshaw. “Using Dynamic Configurability to Support Object-Oriented Languages and Systems in Legion,” University of Virginia CS Technical Report CS-96-19: Dec 1996.

[14] Lewis, M.J., and A.S. Grimshaw. “The Core Legion Object Model” *Proceedings of the Fifth IEEE High Performance Distributed Computing*, August 6-9, 1996, Syracuse, NY: IEEE Computer Society Press, 551-561.

[15] Sunderam, V.S. “PVM: A framework for parallel distributed computing” *Concurrency: Practice and Experience*, 2(4): 315-339, Dec 1990.

[16] Viles, C.L., M.J. Lewis, and A.J. Ferrari, A. Nguyen-Tuong, and A.S. Grimshaw. “Enabling Flexibility in the Legion Run-Time Library,” *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications* (PDPTA’97), June 30 - July 2, 1997, Las Vegas, NV: 265-74.

[17] Wulf, W.A., C. Wang, and D. Kiezle. “A New Model of Security for Distributed Systems,” University of Virginia CS Technical Report CS-95-34, Aug 1995.

Index

A

B

C

R

S

T

U

V

W