Appendix A: Running Your Programs

LogiQL programs and the workspaces they deal with are managed by a command interpreter invoked by issuing the `lb` command to the operating system's shell. At a high level, these commands interact with a server process responsible for managing one or more workspaces. This includes creating them, adding and removing blocks, installing data, and displaying results.

Each of these commands* takes the following form:

`lb` *commandName* [*options* | *arguments*]

Options control the execution of the given command, and *arguments* typically supply names of workspaces or files. Not all options are pertinent to all commands.

For example, the simple command `lb status` tells you whether the server process is running. Normally, when you run this command, you should expect to see the response: `Server is 'ON'`. If you do not see this, then you should run the command `lb services start` to get it running.

Note that the `lb` command and all of its subcommands take the optional `-h` option that responds with a description of the (sub)command and its options and arguments.

One of the first things you will need to do to run your program is to create a workspace for it. The `lb create` command is used to do this for you. For example, the command

```
lb create workspaceName
# Create a new workspace with name workspaceName.
```

* Examples in this appendix use the `bash` command language found on Linux.

creates a workspace with name *workspaceName*, which appears in italics to indicate that you can supply a name of your choice for the created workspace. The newly created workspace will be subsequently managed by the `lb` command. What this means is that the files containing your programs and data are under the control of the server, and you should not expect to see any changes in your current working directory.

There are two options to `lb create` you might want to use under certain circumstances: (1) `lb create --unique` creates a workspace with a unique name; that is, you do not have to supply the name. You might use this option if you are running a short test and do not expect to retain the results. (2) `lb create --overwrite` *workspaceName* reuses an existing name by first ensuring that the old contents of the workspace with the supplied name are deleted.

After you have created the workspace, you will want to add your program to it. This is called *installing* the program in the workspace. Assuming that your program is contained in the file named `program.logic`, you can accomplish this goal with the following `lb` command. Note that the suffix on the program name should be `".logic"`:

```
lb addblock -f program.logic workspaceName
# Add the program in file program.logic to the
# workspace named workspaceName.
```

Complementing the `lb addblock` subcommand is `lb exec`. Whereas `addblock` is used to install your intensional database (IDB) rules, `exec` is used to alter the extensional database (EDB), typically by asserting new facts. Here is an example of using `exec` to update *workspaceName* with assertions/retractions taken from the file `programData.logic`:

```
lb exec -f program.logic workspaceName
# Update the EDB in workspaceName with assertions
# and retractions from the file programData.logic.
```

For both `addblock` and `exec` you have the option of including logic on the command line. That is, instead of using the `-f` *fileName* argument, you can instead include the logic explicitly at the end of the line. If you do this, however, you should be careful to enclose the logic within apostrophes to ensure that the shell does not try to execute your code.

Of course, it does not do you much good to install a program and execute facts unless you can also query the results. There are several ways to do this with the lb command. One way is with the `query` subcommand. That is, `lb query` *workspaceName* `'`*someLogic*`'` executes the query expressed in *someLogic* against the *workspaceName* workspace.

Another way to effect this query is to include it as a *query predicate* in one of the fact files you execute. A query predicate's name begins with an underscore, and its resultant facts are reported to the user rather than being stored into the workspace. For example, if the `factData` predicate is declared as follows:

```
factData[u, c, v] = fData ->
    Unit(u), Category(c), Version(v), float(fData).
```

then the query predicate

```
_(u, c, f) <- factData[u, c, "Budget"] = f.
```

retrieves those facts from `factData` for which the `Version` is "`Budget`". If this predicate has been defined, then you can use the `--print` option to `lb exec` to see the results:

```
lb exec workspaceName -f query.logic --print
```

A third way to see the results of your computation is to use the `lb print` *workspaceName predicateName* command to print out the facts stored in a predicate. There are two arguments to `lb print`: *workspaceName* containing the name of your workspace, and *predicateName* naming the predicate you are interested in. The `lb print` command prints out all of the facts in the named predicate.

The commands described above as well as several other helpful commands are summarized in Table A.1 through Table A.4.

TABLE A.1 Commands for Creating Workspaces

Workspace Commands	
`lb create` *workspaceName*	Create a workspace with the name *workspaceName*.
`lb create --unique`	Create a workspace with a (new) unique name.
`lb create --overwrite` *workspaceName*	Create a workspace *workspaceName*. If one already exists with that name, overwrite it.

TABLE A.2 Commands for Adding Logic

Adding Logic	
`lb addblock -f` *fileName workspaceName*	Add the logic contained in file *fileName* into workspace *workspaceName*.
`lb addblock` *workspaceName* `'`*someLogic*`'`	Add the logic expressed explicitly as *someLogic* into workspace *workspaceName*.

TABLE A.3 Commands for Adding and Querying Data

Adding Data and Querying	
`lb exec -f` *fileName workspaceName*	Execute the logic in file *fileName* against workspace *workspaceName*.
`lb exec` *workspaceName* `'`*someLogic*`'`	Execute *someLogic* against workspace *workspaceName*.
`lb exec -f` *fileName workspaceName* `--print`	Execute logic in file *fileName* against workspace *workspaceName* and display the contents of any anonymous predicates.
`lb query` *workspaceName* `'`*someLogic*`'`	Execute the query expressed in *someLogic* against *workspaceName*.

TABLE A.4 Other Useful Commands

Other Useful Commands and Options	
`lb print` *workspaceName predicateName*	Display the contents of predicate *predicateName* in *workspaceName*.
`lb predinfo` *workspaceName predicateName*	Display information about predicate *predicateName* contained in *workspaceName*, including its arity, types of its arguments, and other physical and logical properties.
`lb list` *workspaceName*	List the predicates defined in *workspaceName*.
`lb addproject` *workspaceName directoryName*	Install a compiled project into the workspace named *workspaceName*. *directoryName* is the name of the directory containing your project file.
`lb delete` *workspaceName*	Delete the workspace named *workspaceName*.
`lb workspaces`	List the currently managed workspaces.
`lb version`	List the version of the runtime engine currently active.
`lb compile file` *fileName*	Compile file *fileName*.
`lb compile project` `[--out-dir` *directoryName*`]` `[--libpath` *path projectFileName*`]`	Compile the project whose project file is named *projectFileName*. If `--out-dir` is provided, then place the compiled files into directory *directoryName*. If `--libPath` is provided, then use *path* to find libraries referenced in the project's code.

lb INTERACTIVE

In addition to command-line invocation of `lb` described above, you can execute the subcommands interactively. If you merely type the command `lb` by itself, you will see a prompt displayed (`lbi >`). You can then enter any of the subcommands you need to use (without the `lb` prefix). That is, if you have a series of commands you would like to execute, running `lb` interactively in this fashion may be easier than running them from the shell. When you wish to leave interactive mode, you can type `exit` at the prompt.

A slight variant to the above interactive usage is also available to you. If you have a series of subcommands that you wish to run repeatedly, you can type them into a file (without the `lb` prefix). Then, you can run the sequence from the shell by entering `lb` *fileName*, where *fileName* is the name of the file containing them. The subcommands will be executed one after the other, and you should see the results displayed on your screen.

CAVEAT

This glimpse of `lb` should give you enough information to get started running programs. In so doing, you may be tempted to copy examples from this book. If you do so, be aware that some seemingly innocent text copied from a `.pdf` file, while looking correct, may actually use illegal characters. For example, the hyphen character ('-'), although looking identical to the hyphen you type to `lb` or include in a `.logic` file, is actually represented by different characters in the `.pdf` file and the command window or editor into which you are typing. You may see an unexpected error message from `lb` or the execution engine if it detects such a character. For example, if you copy the following line from this appendix:

```
lb exec -f file.logic ws
```

you will see the following error message printed:

```
ERROR. Invalid argument: '–f'
```

To work around this problem, you can either type the text in directly, copy it from one of the included resource files, or edit the pasted text to substitute for the hyphens before executing the command.