CHAPTER 5 MANAGING MODEL DEFINITIONS, LIBRARIES, AND PROJECTS

Most complex models are built from parts in one or more libraries. In this chapter we show typical examples of how to make sure your model gets the libraries it needs. We then explain in more general terms the ASCEND mechanism which makes this work and how you can use it to manage multiple modeling projects simultaneously.

5.1 USING REQUIRE AND PROVIDE

5.1.1 REQUIREING SYSTEM.A4L

Suppose you are in a great hurry and want to create a simple model and solve it without concern for good style, dimensional consistency, or any of the other hobgoblins we preach about elsewhere. You will write equations using only *generic_real* variables as defined in system.a4l. The equations in this example do not necessarily have a solution. In your ascdata (see howto1) directory you create an application model definition file “myfile.a4c” which looks like:

```
REQUIRE “system.a4l”;
MODEL quick_n_dirty;
   x = y^2;
   y = x + 2*z;
   z = cos(x+y);
   x,y,z IS_A generic_real;
(* homework problem 3, due May 21. *)
END quick_n_dirty;
```

The very first line ‘REQUIRE “system.a4l”; tells ASCEND to find and load a file named “system.a4l” if it has not already been loaded or provided in some other way. This REQUIRE statement must come before the MODEL which uses the *generic_real* ATOM that system.a4l defines.

The REQUIRE statements in a file should all come at the beginning of the file before any other text, including comments. This makes it very

easy for other users or automated tools to determine which files, if any, your models require.

On the ASCEND command line (in the Console window or xterm) or in the Script window, you can then enter and execute the statement

```
READ FILE “myfile.a4c”;
```

to cause system.a4l and then myfile.a4c to be loaded.

5.1.2 CHAINING REQUIRED FILES

Notice when you read myfile.a4c that ASCEND prints messages about the files being loaded. You will see that a file “basemodel.a4l” is also loaded. In system.a4l you will find at the beginning the statements

```
REQUIRE “basemodel.a4l”;
PROVIDE “system.a4l”;
```

The basemodel library is loaded in turn because of the REQUIRE statement in system.a4l. We will come back to what the PROVIDE statement does in a moment. This chaining can be many files deep. To see a more complicated example, enter

```
READ FILE column.a4l;
```

and watch the long list of files that gets loaded. If you examine the first few lines of each file in the output list, you will see that each file REQUIRES only the next lower level of libraries. This style minimizes redundant loading messages and makes it easy to substitute equivalent libraries in the nested lower levels without editing too many higher level libraries. The term “equivalent libraries” is defined better in the later section on PROVIDE.

5.1.3 BETTER APPLICATION MODELING PRACTICE

never require system.a4l in an application model.

It is generally a bad idea to create a model using only generic_real variables. The normal practice is to use correct units in equations and to use dimensional variables. In the following file we see that this is done by requiring “atoms.a4l” instead of “system.a4l” and by using correct units on the coefficients in the equations.

```
REQUIRE “atoms.a4l”; MODEL quick_n_clean;
   x = y^2/1{PI*radian};
   y = x + 2{PI*radian}*z;
```

```
   z = cos(x+y);
   x, y IS_A angle;
   z IS_A dimensionless;
(* homework problem 3, due May 21. *)
END quick_n_clean;
```

5.1.4 SUBSTITUTE LIBRARIES AND PROVIDE

ASCEND keeps a list of the already loaded files, as we hinted at in Section 5.1.1. A library file should contain a PROVIDE statement, as system.a4l does, telling what library it supplies. Normally the PROVIDE statement just repeats the file name, but this is not always the case. For example, see the first few lines of the file ivpsystem.a4l which include the statement

```
PROVIDE "system.a4l";
```

indicating that ivpsystem.a4l is intended to be equivalent to file system.a4l while also supplying new features. When ivpsystem.a4l is loaded both "system.a4l" and "ivpsystem.a4l" get added to the list of already loaded files. For one explanation of when this behavior might be desirable, see Section 12.1. Another use for this behavior is when creating and testing a second library to eventually replace the first one.

When a second library provides compatible but extended definitions similar to a first library, the second can be substituted for the first one. The second library will obviously have a different file name, but there is no need to load the first library if we already have the second one loaded. ivpsystem.a4l is a second library substitutable for the first library system.a4l. Note that the reverse is not true: system.a4l does not

```
PROVIDE "ivpsystem.a4l";
```

so system is not a valid substitute for ivpsystem.

5.1.5 REQUIRE AND COMBINING MODELING PACKAGES

Model libraries frequently come in interrelated groups. For example, the models referred to in Ben Allan's thesis are published electronically as a package models/ben/ in ASCEND IV release 0.9. To use Ben's distillation libraries, which require rather less memory than the current set of more flexible models, your application model should have the statement

```
REQUIRE "ben/bencolumn.a4l";
```

at the beginning.

Combining models from different packages may be tricky if the package authors have not documented them well. Since all packages are open source code which you can copy into your ascdata directory and modify to suit your needs, the process of combining libraries usually amounts to changing the names of the conflicting model definitions in your copy.

Do NOT use \ instead of / in the package name given to a REQUIRE statement even if you are forced to use Microsoft Windows.

5.2 HOW REQUIRE FINDS THE FILES IT LOADS

The file loading mechanism of REQUIRE makes it simple to manage several independent sets of models in simultaneous development. We must explain this mechanism or the model management may seem somewhat confusing. When a REQUIRE statement is processed, ASCEND checks in a number of locations for a file with that name: ascdata, the current directory, and the ascend4/models directory. We will describe how you can extend this list later. ASCEND also looks for model packages in each of these same locations.

5.2.1 ASCDATA

If your ascdata directory exists and is readable, ASCEND looks there first for required files. Thus you can copy one of our standard libraries from the directory ascend4/models to your ascdata directory and modify it as you like. Your modification will be loaded instead of our standard library. See Section 2.2 for how to find your ascdata directory.

5.2.2 THE CURRENT DIRECTORY

The current directory is what you get if you type ‘pwd’ at the ASCEND Console or xterm prompt. Under Microsoft Windows, the current directory is usually some useless location. Under UNIX, the current directory is usually the directory from which you started ASCEND.

5.2.3 ASCEND4/MODELS/

The standard (CMU) models and packages distributed with ASCEND are found in the ascend4/models/ subdirectory where ASCEND is installed. This directory sits next to the directory ascend4/bin/ where the ascend4 or ascend4.exe executable is located.

5.2.4 MULTIPLE MODELING PROJECTS

If you dislike navigating multi-level directories while working on a single modeling project, you can separate projects by keeping all files related to your current project in one directory and changing to that directory before starting ASCEND. If you have files that are required in all your projects, keep those files in your ascdata directory. Under Windows, cd to the directory containing the current project from the Console window after starting ASCEND.

5.2.5 EXAMPLE: FINDING "BEN/BENCOLUMN.A4L"

Suppose an application model requires bencolumn.a4l from package ben as shown in Section 5.1.5. Normally ASCEND will execute this statement by searching for:

```
~/ascdata/ben/bencolumn.a4l
./ben/bencolumn.a4l
$ASCENDDIST/ascend4/models/ben/bencolumn.a4l
```

Assuming we started ASCEND from directory /usr1/ballan/projects/test1 under UNIX, the full names of these might be

```
/usr0/ballan/ascdata/ben/bencolumn.a4l
/usr1/ballan/projects/test1/ben/bencolumn.a4l
/usr/local/lib/ascend4/models/ben/bencolumn.a4l
```

Assuming we started ASCEND from some shortcut on a Windows desktop, the full names of these locations might be

```
C:\winnt\profiles\ballan\ascdata\ben\bencolumn.a4l
C:\Program Files\netscape\ben\bencolumn.a4l
C:\ASCEND\ascend4\models\ben\bencolumn.a4l
```

The first of these three which actually exists on your disk will be the file that is loaded.

5.2.6 HOW REQUIRE HANDLES FILE AND DEFINITION CONFLICTS

Normally you simply delete all types before loading a new or revised set of ASCEND models and thus you avoid most conflicts. When you are working with a large simulation and several smaller ones, you may not want to delete all the types, however. We decided to make

REQUIRE handle this situation and the almost inevitable redundant REQUIRE statements that occur in the following reasonable way.

When a file is REQUIREd, ASCEND first checks the list of loaded and provided files for a name that matches. If the name is found, then that file is checked to see if it has changed since it was loaded. If the file has changed, then any definition that was changed is loaded in the ASCEND Library and the new definition is used in building any subsequently compiled simulations. Old simulations remain undisturbed and are not updated to use the new definitions since there may be conflicts that cannot be automatically resolved.

5.2.7 EXTENDING THE LIST OF SEARCHED DIRECTORIES

ASCEND uses the environment variable ASCENDLIBRARY as the list of directory paths to search for required files. Normally you do not set this environment variable, and ASCEND works as described above.

To see or change the value of ASCENDLIBRARY that ASCEND is using, examine ASCENDLIBRARY in the System utilities window available from the Script Tools menu. Changes made to environment variables in the utilities window are NOT saved. If you are clever enough to set environment variables before running ASCEND, you can make it look anywhere you want to put your model files. Consult your operating system guru for information on setting environment variables if you do not already know how.