CHAPTER 3 PREPARING A MODEL FOR REUSE

There are four major ways to prepare a model for reuse. First, you should add comments to a model. Second, you should add methods to a model definition to pass to a future user your experience in creating an instance of this type which is well-posed. Third, you should parameterize the model type definition to alert a future user as to which parts of this model you deem to be the most likely to be shared. And fourth, you should prepare a script that a future user can run to solve a sample problem involving an instance of the model. We shall consider each of these items in turn in what follows.

3.1 ADDING COMMENTS AND NOTES

In ASCEND we can create traditional comments for a model — i.e., add text to the code that aids anyone looking at the code to understand what is there. We do this by enclosing text with the delimiters (* and *). Thus the line

```
(* This is a comment *)
```

is a comment in ASCEND. Traditional comments are only visible when we display the code using the *Display code* tool in the **Library** window or when we view the code in the text editor we used to create it.

We suggest we can do more for the modeler with the concept of **Notes**, a form of "active" comments available in ASCEND. ASCEND has tools to extract notes and display them in searchable form.

notes are active comments

In Figure 3-1 we show two types of notes the modeler can add. Longer notes are set off in block style starting with the keyword NOTES and ending with END NOTES. In this model, we declare two notes in this manner: (1) to indicate who the author is and (2) to indicate the creation date for this model. Note that the notes are directed to documenting SELF which is the model itself — i.e., the vessel model as a whole object. The object one documents can be any instance in the model — any variable, equation or part. The tools for handling notes can sort on the terms enclosed in single quotes so one could, for example, isolate the *author* notes for all the models.

```
REQUIRE "atoms.a4l";
MODEL vessel;
    NOTES
   'author'          SELF {Arthur W. Westerberg}
   'creation date'   SELF {May, 1998}
    END NOTES;
   (* variables *)
   side_area      "the area of the cylindrical side wall of the vessel",
   end_area       "the area of the flat ends of the vessel"
                IS_A area;
   vessel_vol     "the volume contained within the cylindrical vessel",
   wall_vol       "the volume of the walls for the vessel"
                IS_A volume;
   wall_thickness "the thickness of all of the vessel walls",
   H              "the vessel height (of the cylindrical side walls)",
   D              "the vessel diameter"
                IS_A distance;
   H_to_D_ratio   "the ratio of vessel height to diameter"
                IS_A factor;
   metal_density  "density of the metal from which the vessel
                    is constructed"
                IS_A mass_density;
   metal_mass     "the mass of the metal in the walls of the vessel"
                IS_A mass;
   (* equations *)
FlatEnds:            end_area = 1{PI} * D^2 / 4;
Sides:               side_area = 1{PI} * D * H;
Cylinder:            vessel_vol = end_area * H;
Metal_volume:        (side_area + 2 * end_area) * wall_thickness = wall_vol;
HD_definition:       D * H_to_D_ratio = H;
VesselMass:          metal_mass = metal_density * wall_vol;
END vessel;

ADD NOTES IN vessel;
'description' SELF {This model relates the dimensions of a
           cylindrical vessel -- e.g., diameter, height and wall thickness
           to the volume of metal in the walls.  It uses a thin wall
           assumption -- i.e., that the volume of metal is the area of
           the vessel times the wall thickness.}
'purpose' SELF {to illustrate the insertion of notes into a model}
END NOTES;
```

Figure 3-1 Vessel model with Notes added (model vesselNotes.a4c)

A user may use any term desired in the single quotes. We have not decided yet what the better set of terms should be so we do not as yet suggest any. With time we expect the terms used to settle down to just a few that are repeated for all the models in a library.

there are short notes, long notes and separate notes

There are also short notes we can attach to every variable in the model. A "one liner" in double quotes just following the variable name allows the automatic annotation of variables in reports.

The last few lines of Figure 3-1 shows adding notes we write in a separate *ADD NOTES IN* object. This object can appear before or after or in a different file from the object it describes. This style of note writing is useful as it allows another person to add notes to a model without changing the code for a model. Thus it allows several different sets of notes to exist for a single model, with the choice of which to use being up to the person maintaining the model library. Finally, it allows one to eliminate the "clutter" the documentation often adds to the code.

3.2 ADDING METHODS

We would next like to pass along our experiences in getting this model to be well-posed—i.e., we would like to tell future users which variables we decided to fix and which we decided to calculate. We would also like to provide some typical values for the variables we decided to fix. ASCEND allows us to attach any number of methods to a type definition. Methods are procedural code that we can request be run through the interface while browsing a model instance. We shall include methods as described Table 3-1 to set just the right fixed flags and variable values for an instance of our vessel model to be well-posed.

The system has defaults definitions for all these methods. You already saw that to be true if you went through the process of setting all the *fixed* flags to *FALSE* in the previous chapter. In case you did not, load and compile the *vesselPlain.a4c* model in ASCEND. Export the compiled instance to the **Browser**. Then in the **Browser**, under the *Edit* button, select *Run method*. You will see a list containing these and other methods we shall be describing shortly. Select *specify* and hit the *OK* button. Then look in the Console window. A message similar to the following will appear, with all but the first line being in red to signify you should pay attention to the message:

```
Running method specify in v
Found STOP statement in METHOD
C:\PROGRAM FILES\ASCEND\ASCEND4\models\basemodel.a4l:307
```

```
STOP {Error! Standard method "specify" called but not written in MODEL.};
```

This message is telling you that you have just run the default *specify* method. We have to hand-craft every *specify* method so the default method is not appropriate. This message is alerting us to the fact that we did not yet write a special *specify* method for this model type.

Try running the *ClearAll* method. The default *ClearAll* method is always the one you will want so it does not put out a message to alert you that it is the default. d

Table 3-1 Some of the methods we require for putting a model into an ASCEND library

method	**description**
ClearAll	a method to set all the *.fixed* flags for variables in the type to *FALSE*. This puts these flags into a known standard state — i.e., all are *FALSE*. All models inherit this method from the base model and the need to rewrite it is very, very rare.
specify	a method which assumes all the fixed flags are currently *FALSE* and which then sets a suitable set of *fixed* flags to *TRUE* to make an instance of this type of model well-posed. A well-posed model is one that is square (*n* equations in *n* unknowns) and solvable.
reset	a method which first runs the ClearAll method and then the *specify* method. We include this method because it is very convenient. We only have to run one method to make any simulation well-posed, no matter how its fixed flags are currently set. All models inherit this method from the base model, as with *ClearAll*. It should only rarely have to be rewritten for a model.
values	a method to establish typical values for the variables we have fixed in an application or test model. We may also supply values for some of the variables we will be computing to aid in solving a model instance of this type. These values reflectiveness that we have tested for a simulation of this type and found to work.

writing the *specify* and *values* methods

To write the *specify* and *values* methods for our vessel model, we note that we have successfully solved the vessel model in at least two different ways above. Thus both variations are examples of being "well-posed." We can choose which variation we shall use when creating the *specify* method for our vessel type definition. Let us choose the alternative where we fixed *vessel_volume, H_to_D_ratio, metal_density* and *wall_thicknes* and provided them with the values of *250 ft^3, 3, 5000 kg/m^3* and *5 mm* respectively to be our "standard" specification.

default methods *ClearAll* and *reset* are appropriate

As already noted, the purpose of *ClearAll* is to set all the fixed flags to *FALSE*, a well-defined state from which we can start over to set these flags as we wish them set. *Reset* simply runs *ClearAll* and then *specify* for a model. The default versions for these two methods are generally exactly what one wants so one need not write these.

Figure 3-2 illustrates our vessel model with our local versions added for *specify* and *values*. Look only at these for the moment and note that they do what we described above. We show some other methods we shall explain in a moment.

```
REQUIRE "atoms.a4l";
MODEL vessel;
    NOTES
   'author'          SELF {Arthur W. Westerberg}
   'creation date'   SELF {May, 1998}
    END NOTES;

   (* variables *)
   side_area      "the area of the cylindrical side wall of the vessel",
   end_area       "the area of the flat ends of the vessel"
                IS_A area;
   vessel_vol     "the volume contained within the cylindrical vessel",
   wall_vol       "the volume of the walls for the vessel"
                IS_A volume;
   wall_thickness "the thickness of all of the vessel walls",
   H              "the vessel height (of the cylindrical side walls)",
   D              "the vessel diameter"
                IS_A distance;
   H_to_D_ratio   "the ratio of vessel height to diameter"
                IS_A factor;
   metal_density  "density of the metal from which the vessel
                    is constructed"
                IS_A mass_density;
   metal_mass     "the mass of the metal in the walls of the vessel"
                IS_A mass;
   (* equations *)
FlatEnds:         end_area = 1{PI} * D^2 / 4;
Sides:            side_area = 1{PI} * D * H;
Cylinder:         vessel_vol = end_area * H;
Metal_volume:     (side_area + 2 * end_area) * wall_thickness = wall_vol;
HD_definition:    D * H_to_D_ratio = H;
VesselMass:       metal_mass = metal_density * wall_vol;

METHODS
METHOD specify;
```

```
    NOTES
   'purpose' SELF {to fix four variables and make the problem well-posed}
    END NOTES;
   vessel_vol.fixed        := TRUE;
   H_to_D_ratio.fixed      := TRUE;
   wall_thickness.fixed    := TRUE;
   metal_density.fixed     := TRUE;
END specify;
METHOD values;
    NOTES
   'purpose' SELF {to set the values for the fixed variables}
    END NOTES;
   H_to_D_ratio            := 2;
   vessel_vol              := 250 {ft^3};
   wall_thickness          := 5 {mm};
   metal_density           := 5000 {kg/m^3};
END values;
METHOD bound_self;
END bound_self;
METHOD scale_self;
END scale_self;
METHOD default_self;
   D                       := 1 {m};
   H                       := 1 {m};
   H_to_D_ratio            := 1;
   vessel_vol              := 1 {m^3};
   wall_thickness          := 5 {mm};
   metal_density           := 5000 {kg/m^3};
END default_self;
END vessel;

ADD NOTES IN vessel;
'description' SELF {This model relates the dimensions of a
           cylindrical vessel -- e.g., diameter, height and wall thickness
           to the volume of metal in the walls.  It uses a thin wall
           assumption -- i.e., that the volume of metal is the area of
           the vessel times the wall thickness.}
'purpose' SELF {to illustrate the insertion of notes into a model}
END NOTES;
```

Figure 3-2 Version of vessel with methods added (vesselMethods.a4c)

In Table 3-2 we describe additional methods we require before we will put a model into one of our libraries. Each of these had two versions, both of which we require. The designation *_self* is for a method to do something for all the variables and/or parts we have defined locally within the current model with an *IS_A* statement. The designation *_all* is for a method to do something for parts that are defined within an "outer" model that has an instance of this model as a part. The "outer" model is at a higher scope. It can share its parts with this model by passing them in as parameters, a topic we cover shortly in Section 3.3. Only the *_self* versions of these methods are relevant here and are in Figure 3-2.

Table 3-2 Additional methods required for model in ASCEND libraries

method	**description** (The *_self* versions of each of these methods should run the *_self* versions for the same method for all of its parts that are instances of models created with an *IS_A* statement. The *_all* version should first run the _self version of the same method and then the *_all* version for all of its parts passed in as parameters with a *WILL_BE* statement.)
default_self *default_all*	a method called automatically when any simulation is compiled to provide default values and adjust bounds for any variables which may have unsuitable defaults in their ATOM definitions. Usually the variables selected are those for which the model becomes ill-behaved if given poor initial guesses or bounds (e.g., zero).
bound_self *bound_all*	a method to update the *.upper_bound* and *.lower_bound* value for each of the variables. ASCEND solvers use these bound values to help solve the model equations.
scale_self *scale_all*	a method to update the *.nominal* value for each of the variables. ASCEND solvers will use these nominal values to rescale the variable to have a value of about one in magnitude to help solve the model equations.
check_self *check_all*	a method to check that the computations make sense. At first this method may be empty, but, with experience, one can add statements that detect answers that appear to be wrong. As ASCEND already does bounds checking, one should not check for going past bounds here. However, there could be a rule of thumb available that suggests one computed variable should be about an order of magnitude larger than another. This check could be done in this method.

adding our remaining *standard* methods to a model definition

The *bound_self* and *scale_self*, methods we have written are both empty. We anticipate no difficulties with variable scaling or bounding for this small model. Larger models can often give difficult problems in solving if the variables in them are not properly scaled and bounded; these issues must be taken very seriously for such models.

We have included the variables that define the geometry of the vessel in *defaults_self* method to avoid such things as negative initial values for *vessel_volume*. The compiler for ASCEND runs this method as soon as the model is compiled into an instance so the variables mentioned here start with their default values.

using methods when solving

Exit ASCEND and repeat all the steps above to edit, load and compile this new vessel type definition. Then proceed as follows.

- In the *Browser* window, examine the values for those variables mentioned in the *default_self* method. Note they already have their default values.
- To place the new instance *v* in a solvable state, go to the **Browser** window. Pick the command *Run method* under the *Edit* button. Select first the method *values* and hit *OK*.
- Repeat the last step but this time select the method *reset*.

In the **Browser**, examine the values for the variables listed in the method *values* in Figure 3-2. They should be set to those stated (remember you can alter the units ASCEND uses to report the values by using the tools in the **Units** window).Also examine the *fixed* flags for these variables; they should all be *TRUE* (remember that you can find which variables are fixed all at once by using the *By type* command under the *Find* button).

- Finally export *v* to the **Solver**. The **Eligible** window should NOT appear; rather that **Solver** should report the model to be *square*.
- Solve by selecting *Solve* under the *Execute* button.

The inclusion of methods has made the process of making this model much easier to get well-posed. This approach is the one that works for really large, complex models. For chemical engineering process unit models there are one or two additional tips covered in Chapter 10.

3.3 PARAMETERIZING THE VESSEL MODEL

let's compute *metal_mass* vs. *H_to_D_ratio*

Reuse generally implies creating a model which will have as a part an instance of a previously defined type. For example, let us compute *metal_mass* as a function of the *H_to_D_ratio* for a vessel for a fixed *vessel_volume*. We would like to see if there is a value for the *H_to_D_ratio* for which the *metal_mass* is minimum for a vessel with a given *vessel_volume*. We might wonder if *metal_mass* goes to infinity as this ratio goes either to zero or infinity.

3.3.1 CREATING A PARAMETERIZED VERSION OF VESSEL

parameters indicate likely object sharing

To use instances of our model as parts in another model, we can parameterize it. We use parameterization to tell a future user that the parameters are objects he or she is likely to share among many different parts of a model. We wish to create a table containing different values of *H_to_D_ratio* vs. *metal_mass*. We can accomplish this by computing simultaneously several different vessels having the same *vessel_volume*, *wall_thickness* and *metal_density.* The objects we want to see and/or share for each instance of a vessel should include, therefore: *H_to_D_ratio*, *metal_mass*, *metal_density*, *vessel_volume* and *wall_thickness*.

The code in Figure 3-3 indicates the changes we make to the model declaration statement and the statements defining the variables to parameterize our model.

```
REQUIRE "atoms.a4l";
MODEL vessel(
   vessel_vol     "the volume contained within the cylindrical vessel"
      WILL_BE volume;
   wall_thickness "the thickness of all of the vessel walls"
      WILL_BE distance;
   metal_density  "density of the metal from which the vessel is constructed"
      WILL_BE mass_density;
   H_to_D_ratio   "the ratio of vessel height to diameter"
      WILL_BE factor;
   metal_mass     "the mass of the metal in the walls of the vessel"
      WILL_BE mass;
);
    NOTES
   'author' SELF        {Arthur W. Westerberg}
   'creation date' SELF {May, 1998}
    END NOTES;

   (* variables *)
   side_area      "the area of the cylindrical side wall of the vessel",
   end_area       "the area of the flat ends of the vessel"
                IS_A area;
   wall_vol       "the volume of the walls for the vessel"
                IS_A volume;
   H              "the vessel height (of the cylindrical side walls)",
   D              "the vessel diameter"
                IS_A distance;
   (* equations *)
FlatEnds:            end_area = 1{PI} * D^2 / 4;
```

```
Sides:            side_area = 1{PI} * D * H;
Cylinder:         vessel_vol = end_area * H;
Metal_volume:     (side_area + 2 * end_area) * wall_thickness = wall_vol;
HD_definition:    D * H_to_D_ratio = H;
VesselMass:       metal_mass = metal_density * wall_vol;

METHODS
METHOD specify;
    NOTES
   'purpose' SELF {to fix four variables and make the problem well-posed}
    END NOTES;
   vessel_vol.fixed        := TRUE;
   H_to_D_ratio.fixed      := TRUE;
   wall_thickness.fixed    := TRUE;
   metal_density.fixed     := TRUE;
END specify;
METHOD values;
    NOTES
   'purpose' SELF {to set the values for the fixed variables}
    END NOTES;
   H_to_D_ratio            := 2;
   vessel_vol              := 250 {ft^3};
   wall_thickness          := 5 {mm};
   metal_density           := 5000 {kg/m^3};
END values;
METHOD bound_self;
END bound_self;
METHOD bound_all;
   RUN bound_self;
END bound_all;
METHOD scale_self;
END scale_self;
METHOD scale_all;
   RUN scale_self;
END scale_all;
METHOD default_self;
   D                       := 1 {m};
   H                       := 1 {m};
END default_self;
METHOD default_all;
   RUN default_self;
   vessel_vol              := 1 {m^3};
   wall_thickness          := 5 {mm};
   metal_density           := 5000 {kg/m^3};
   H_to_D_ratio            := 1;
END default_all;
END vessel;
ADD NOTES IN vessel;
'description' SELF {This model relates the dimensions of a
           cylindrical vessel -- e.g., diameter, height and wall thickness
           to the volume of metal in the walls.  It uses a thin wall
           assumption -- i.e., that the volume of metal is the area of
           the vessel times the wall thickness.}
'purpose' SELF {to illustrate the insertion of notes into a model}
END NOTES;
```

Figure 3-3 The parameterized version of vessel model (vesselParams.a4c)

Substitute the statements in Figure 3-3 for lines 2 through 9 in Figure 3-2. Save the result in the file *vesselParam.a4c*.

Note the use of the WILL_BE statement in the parameter list. By declaring that the type of a parameter will be compatible with the types shown, the compiler can tell immediately if a user of this model is passing the wrong type of object when defining an instance of a vessel.

3.3.2 USING THE PARAMETERIZED VESSEL MODEL

Creating a table of metal_mass values vs. H_to_D_ratio

We next need to create a type definition that will set up our table of *H_to_D_ratio* values vs. *metal_mass* so we can observe approximately where it attains a minimum value. ASCEND allows us to create arrays of instances of any type. Here we shall create an array of vessels. The type definition is shown in Figure 3-4.

```
REQUIRE "atoms.a4l";
MODEL vessel(
   vessel_vol "the volume contained within the cylindrical vessel"
      WILL_BE volume;
   wall_thickness "the thickness of all of the vessel walls"
      WILL_BE distance;
   metal_density "density of the metal from which the vessel is constructed"
      WILL_BE mass_density;
   H_to_D_ratio "the ratio of vessel height to diameter"
      WILL_BE factor;
   metal_mass "the mass of the metal in the walls of the vessel"
      WILL_BE mass;
);
    NOTES
   'author' SELF {Arthur W. Westerberg}
   'creation date' SELF {May, 1998}
    END NOTES;
   (* variables *)
   side_area "the area of the cylindrical side wall of the vessel",
   end_area "the area of the flat ends of the vessel"
         IS_A area;
   wall_vol "the volume of the walls for the vessel"
         IS_A volume;
   H     "the vessel height (of the cylindrical side walls)",
   D     "the vessel diameter"
         IS_A distance;
   (* equations *)
FlatEnds:end_area = 1{PI} * D^2 / 4;
Sides:side_area = 1{PI} * D * H;
Cylinder:vessel_vol = end_area * H;
Metal_volume:(side_area + 2 * end_area) * wall_thickness = wall_vol;
HD_definition:D * H_to_D_ratio = H;
VesselMass:metal_mass = metal_density * wall_vol;
METHODS
METHOD specify;
    NOTES
   'purpose' SELF {to fix four variables and make the problem well-posed}
    END NOTES;
   vessel_vol.fixed     := TRUE;
   H_to_D_ratio.fixed   := TRUE;
   wall_thickness.fixed := TRUE;
   metal_density.fixed  := TRUE;
END specify;
METHOD values;
    NOTES
   'purpose' SELF {to set the values for the fixed variables}
    END NOTES;
   H_to_D_ratio         := 2;
   vessel_vol           := 250 {ft^3};
   wall_thickness       := 5 {mm};
   metal_density        := 5000 {kg/m^3};
END values;
METHOD bound_self;
END bound_self;
METHOD bound_all;
   RUN bound_self;
END bound_all;
METHOD scale_self;
END scale_self;
METHOD scale_all;
   RUN scale_self;
END scale_all;
```

```
METHOD default_self;
   D                    := 1 {m};
   H                    := 1 {m};
END default_self;
METHOD default_all;
   RUN default_self;
   vessel_vol           := 1 {m^3};
   wall_thickness       := 5 {mm};
   metal_density        := 5000 {kg/m^3};
   H_to_D_ratio         := 1;
END default_all;
END vessel;

ADD NOTES IN vessel;
'description' SELF {This model relates the dimensions of a
           cylindrical vessel -- e.g., diameter, height and wall thickness
           to the volume of metal in the walls.  It uses a thin wall
           assumption -- i.e., that the volume of metal is the area of
           the vessel times the wall thickness.}
'purpose' SELF {to illustrate the insertion of notes into a model}
END NOTES;

MODEL tabulated_vessel_values;
    vessel_volume    "volume of all the tabulated vessels"
         IS_A volume;
    wall_thickness   "thickness of all the walls for all the vessels"
         IS_A distance;
    metal_density    "density of metal used for all vessels"
         IS_A mass_density;
    n_entries        "number of vessels to simulate"
         IS_A integer_constant;
    n_entries :== 20;
    H_to_D_ratio[1..n_entries]   "set of H to D ratios for which we are
                     computing metal mass"
         IS_A factor;
    metal_mass[1..n_entries]     "mass of metal in walls of vessels"
         IS_A mass;
    FOR i IN [1..n_entries] CREATE
   v[i]             "the i-th vessel model"
         IS_A  vessel(vessel_volume, wall_thickness,
   metal_density, H_to_D_ratio[i], metal_mass[i]);
    END FOR;

METHODS
METHOD default_self;
END default_self;
```

```
METHOD specify;
   RUN v[1..n_entries].specify;
END specify;
METHOD values;
    NOTES 'purpose' SELF {to set up 20 vessel models having H to D ratios
        ranging from 0.1 to 2.}
    END NOTES;
   vessel_volume             := 250 {ft^3};
   wall_thickness            := 5 {mm};
   metal_density             := 5000 {kg/m^3};
   FOR i IN [1..n_entries] DO
      H_to_D_ratio[i]        := i/10.0;
   END FOR;
END values;
METHOD scale_self;
END scale_self;
END tabulated_vessel_values;

ADD NOTES IN tabulated_vessel_values;
'description' SELF {This model sets up an array of vessels to
           compute a range of metal_mass values for different values
           of H_to_D_ratio.}
'purpose' SELF {to illustrate the use of arrays in ASCEND}
END NOTES;
```

Figure 3-4 The code for the *tabulated_vessel_values* model (vesselTabulated.a4c)

Add this model to the end of the file *vesselParam.a4c* (after the vessel model) and save the file as *vesselTabulated.a4c*. Compile an instance of *tabulated_vessel_values* (call it *tvv*), run the *values* and *specify* methods for it, and then solve it. You will discover that the tenth element of the *metal_mass* array, corresponding to an *H_to_D_ratio* of *1* has the minimum value of *510.257 kilograms*.

3.4 CREATING A SCRIPT TO DEMONSTRATE THIS MODEL

The last step to make the model reusable is to create a script that anyone can easily run. Running the model successfully will allow a user to demonstrate the use of the model and to explore an instance it by browsing it.

ASCEND allows one to create such a script using either an editor or the tools in the **Script** window.

Restart the ASCEND system. You will have three windows open plus the large one which you can close by pressing its *dismiss* button. The **Script**, the **Library** and the **Console**[1] windows remain

In the **Script** window you will see the license agreement information for ASCEND. First get a new script buffer by selecting the *New file* tool under the *File* button.

Select the tool *Record actions* under the *Edit* button to start recording the steps you are about to undertake.

- In the **Library** window, under the *Edit* button, select *Delete all types*. Hit *Delete all* on the small confirmation window that appears.
- Load the file *vesselTabulated.a4c*, the file containing the model called *tabulated_vessel_values.* Do this by selecting the *Read types from file* tool under the *File* button and browsing the file system to find it. If you have trouble finding it, be sure to set the *Files of type* window at the bottom of the file browsing window to allow all types of files to be seen.
- Select the type *tabulated_vessel_values* in the right **Library** window and compile an instance of it by selecting the *Create simulation* tool under the *Edit* button. In the small window that appears, enter the name *tvv* and hit *OK*.
- Export the instance to the **Browser** by selecting the *Simulation to Browser* tool under the *Export* button.
- Initialize the variable values by running the *values* method. Do this by selecting the *Run method* tool under the *Edit* button. Select the *values* method and hit *OK*.
- Set the fixed flags to get a well-posed problem by repeating the last step but this time select the *reset* method.
- Export the instance *tvv* to the **Solver** by selecting the *to Solver* tool under the *Export* button.
- Solve tvv by selecting the *Solve* tool under the *Execute* button in the **Solver** window.
- Return to the **Script** window and turn off the recording by selecting the *Record actions* tool under the *Edit* button.

1. UNIX users should treat the xterm where they started ASCEND as their **Console** window.

- Save the script you have just created by selecting the *Save* tool under the *File* button of the **Script** window. Name the file *vesselTabulated.a4s* (note the '*s*' ending) to indicate it is the script to run an example problem for models in the *vesselTabulated.a4c* (note the '*c*' ending) file.
- Exit ASCEND by selecting the *Exit ASCEND* tool under the *File* button on the **Script** window. The contents of the **Script** window will be similar to that in Figure 3-5 (the path to the file may differ).
- Restart ASCEND.
- Open the script you just created by selecting the *Read file* tool under the *File* button on the **Script** window. (Be sure you are allowing the system to see files with the ending *a4s* by setting the *Files of type* window at the bottom of the file browsing window.)
- Highlight all the instructions in this script and then execute the highlighted instructions by selecting the *Statements selected* tool under the *Execute* button.

You will run the same sequence of instructions you ran to create the script.

```
DELETE TYPES;
READ FILE "vesselTabulated.a4c";
COMPILE tvv OF tabulated_vessel_values;
BROWSE {tvv};
RUN {tvv.reset};
RUN {tvv.values};
SOLVE {tvv} WITH QRSlv;
```

Figure 3-5 Script to run vesselTabulated.a4c (this is the contents of the file vesselTabulated.a4s)

3.4.1 DISCUSSION

In this chapter we converted the vessel model into a form where you and others in the future will have a chance to reuse it. We did this by first adding methods to make the problem well-posed and to provide values for the fixed variables for which we readily found a solution when playing with our original model as we did in the previous chapter. We then thought of a typical use for this model and developed a parameterized version based on that use. If this model were in a library, a future user of it would most often simply have to understand the parameters to create an instance of this type of model. We next added

NOTES, a form of active comments, to the model. We suggest that notes are much more useful than comments as we can provide tools that can extract them and allow us to search them, for example, to find a model with a given functionality. Finally, we showed you how to create a script by turning on a "phone" session where ASCEND records the actions one takes when loading, compiling and solving a model. One can save and play this script in the future to see a typical use of the model.

In the next chapter, we look at how we can plot the results we created in the model vesselTabulated.a4c. We will have to reuse a model someone else has put into the library of available models. In other words, the "shoe is on the other foot," and we quickly experience the difficulties with reuse first hand. We will also learn how to run a case study from which we can extract the same information with a single vessel model run multiple times.