CHAPTER 2 A DETAILED ASCEND EXAMPLE FOR BEGINNERS: THE MODELING OF A VESSEL

the purpose for this chapter

You read our propaganda about the ASCEND system in which we said it was to help technical people create hard models. We said you can tackle really large models -- 100,000 equations, compiling and solving them in minutes on a PC. We also pointed out that you can readily solve the small problems many currently solve using a spreadsheet, only once posed you can solve them inside out, upside down and backwards.

This sounded intriguing so you downloaded the system and installed it. Aside from getting the load module to transfer without error (there still are network problems), this step proved quite straight forward. You double clicked the ASCEND icon on your desktop and started it up for the first time. Four windows opened up. You panicked.

Who wouldn't?

To use this system properly requires that you learn how to use it. If you pay the price to do so - and we hope it is not a large price, then we believe you will find the tools we have provided to help you create and debug models will pay you back handsomely.

This (Chapter 2)and the next two chapters (Chapter 3 and Chapter 4) are meant to be a good first step along the path to learning how to use ASCEND. We shall lead you through the steps for creating and testing a simple model. You will also learn how to improve this model so it may be more readily shared with others. We will present our reasons for the steps we take. We shall show you all the buttons you should push as you proceed.

We strongly suggest you put time aside and go through all three of these chapters to introduce yourself to ASCEND. It should take you about two to three hours. The second chapter is particularly important if you wish to understand our approach to good modeling practices.

the problem

Step 1: *We are going to create and test an ASCEND model to compute, the mass of the metal in the sides and ends of the thin-walled cylindrical vessel shown in Figure 2-1.*

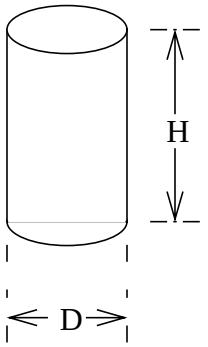

Figure 2-1 A thin-walled cylindrical vessel with flat ends

Step 2: *This model is to become a part of a library of models which others can use in the future. You must document it. You must add methods to it to make it easy for others to make it well-posed. You should probably parameterize it, and finally you must create a script which anyone can easily run that solves an example problem to illustrate its use.*

topics covered

Topics covered in this and the following two chapters are:

Chapter 2 (this chapter)

- Converting the word description to an ASCEND model.
- Loading the model into ASCEND, dealing with the error messages.
- Compiling the model.
- Browsing the model to see if it looks right
- Solving the model.
- Examining the results.
- More thoroughly testing the model.

Chapter 3

- Converting the model to a more reusable form by adding methods to it and by parameterizing it.
- Creating a script to load and execute an instance of the model.

Chapter 4

- Creating an array of models.

- Using an existing library model for plotting.
- Creating a case study using the model.

We shall introduce many of the features of the modeling language as well as the use of the interactive interface you use when compiling, debugging, solving and exploring your model. Language features include units conversion, arrays and sets.

2.1 CONVERTING THE WORD DESCRIPTION INTO AN ASCEND MODEL

an ASCEND model is a type definition

Every ASCEND model is, in fact, a type definition. To "solve a model," we make an instance of a type and solve the instance. So we shall start by creating a vessel *type* definition. We will have to create our type definition as a text file using a text editor. (Possible text editors are Word, Framemaker, Emacs, and Notepad. We shall discuss editors shortly.)

We need first to decide the parts to our model. In this case we know that we need the variables listed in Table 2-1. We readily fill in the first three columns in this table. We shall discuss the entry in the last column in a moment.

Table 2-1 Variables required for model

Symbol	**Meaning**	**Typical Units**	**ASCEND variable type**
D	vessel diameter	m, ft	`length`
H	vessel height	m, ft	`length`
wall_thickness	wall thickness	mm, in	`length`
metal_density	metal density	kg/m^3, lbm/ft^3	`mass_density`

We will be computing the masses for the metal in the side wall and in the ends for this vessel. As this is a thin-walled vessel, we shall compute the volume of metal as the area of the walls times the wall thickness. The following equations allow us to compute the required areas

$$\text{side wall area} = \pi DH \tag{2.1}$$

$$\text{single end area} = \frac{\pi D^2}{4} \tag{2.2}$$

We should be interested in the volume of the vessel, which we compute as:

$$\text{vessel volume} = \text{end area} \times H \qquad \textbf{(2.3)}$$

We add the variables in Table 2-2 to our list.

Table 2-2 Some more variables required for vessel model

Symbol	Meaning	Typical Units	ASCEND variable type
side_area	area in the side wall of the vessel	m^2, ft^2	`area`
end_area	total area in the ends of the vessel	m^2, ft^2	`area`
vessel_volume	volume of the vessel	m^3, ft^3	`volume`
metal_volume	total volume of metal in walls	m^3, ft^3	`volume`
metal_mass	total mass of the metal in the walls of the vessel	kg, lbm	`mass`

We believe that no one should create a model of any consequence without worrying about the units for expressing the variables within it. We consider that to be a commandment handed down from somewhere on high; however, we know that others do not believe as we do. Grant us our beliefs. We have created in the ASCEND system a library of variable and constant types called

atoms.a4l.

The file type ".a4l" designates it to be an "ASCEND IV library" file. Double click on this link to see the approximately 150 different types ranging from universal constants such as π (=3.14159...) and e (=2.718...) to length, mass and angles. If we have not created one that you need, you can use this library of types to see how to construct one for yourself and add it to your file of type definitions. You will find detailed instructions for how to make your own variable type library in Chapter 7, "How to Define Variables and Scaling Values in an ASCEND Model," on page 73.

type definition library for variables and constants

ASCEND considers variable and constant types to be elementary or "atomic" to the system. These type definitions can contain only attributes for variables and constants. They cannot contain equations,

for example. Thus ASCEND calls such a type definition an *atom* rather than a *model*. Figure 2-2 illustrates the definition for the type *volume*.

```
ATOM volume REFINES solver_var
   DIMENSION L^3
   DEFAULT 100.0{ft^3};
   lower_bound := 0.0{ft^3};
   upper_bound := 1e50{ft^3};
   nominal := 100.0{ft^3};
END volume;
```

Figure 2-2 A typical type definition called an atom used to define variable and constant types. Here we illustrate the type that defines volume.

The definition starts by stating that volume is a specialization of *solver_var.* The type *solver_var* refines a base type in the system known as *real* and adds several attributes to it that a nonlinear equation solver may need, such as a lower and upper bounds, a fixed flag, and so forth.

dimensions and units in ASCEND.

The type definition for volume states that volume has dimensionality of length to the power 3 (L^3) where L is one of the 10 dimensions supported by ASCEND (see “Dimensionality:” on page 161 in ASCEND Syntax document for the 10 dimensions defined within the ASCEND language).

One may express the value for a volume using any units which are consistent with the dimensionality of L^3, such as {ft^3}, {m^3}, {gal}, or even {mile^4/mm}. Setting the lower bound to 0 {ft^3} says volume must be a nonnegative number. ASCEND used the nominal value for scaling a variable of type volume when solving, here 100 ft^3.

One may change the values for the bounds, default and nominal values at any time.

We now can understand the last column in Table 2-1 and Table 2-2. For each variable or constant in the system, we have identified its type in the file *atoms.a4l*. That is, we looked in this file for the type definition that corresponded to the variable we were defining and listed that type here. This task is not as onerous as it seems. As we shall see later, we provide a tool to find for you all atom types that correspond to a particular set of units, e.g, ft^3 -- i.e., the computer will do the searching for you.

In Figure 2-3 we see the definition of one of the universal constants contained in atoms.a4l. This definition is very short; it gives the name

of the type *circle_constant*, that it refines *real_constant* and that it has the value 1 {PI} where the internal conversion needed for {PI} is defined in the file defining the built-in units in ASCEND. One can add more units if desired at any time to ASCEND by defining one or more personal units files (Chapter 9 tells you how to do this).

universal constant definition

```
UNIVERSAL CONSTANT circle_constant
    REFINES real_constant :== 1{PI};
```

Figure 2-3 The type definition for circle_constant which has the value of 1 {PI} (equals 3.1415926536)

We shall in fact find this constant useful in our program, and we can either introduce a constant with this value or simply use the value 1{PI} in our program. We shall choose to do the latter.

It is time to write our first version for the model, which we do in Figure 2-4. We first list any other files containing type definitions which this model will use; here we list “atoms.a4l” following the keyword `REQUIRE`. ASCEND is sensitive to case so pay attention to where we use and do not use capital letters. Keywords are always capitalized. Often for clarification we use capital letters in a name we use for a variable or label (e.g., we use D for diameter rather than d). Note that all ASCEND statements end with a semicolon (i.e., with ;) and not at the end of a line and that blank lines have no impact. Comments are between opening and closing parenthesis/asterisk pairs, i.e., ‘*(**’ and ‘***)*’.

the first version of the code for vessel

```
REQUIRE "atoms.a4l";
MODEL vessel;
   (* variables *)
   side_area, end_area     IS_A  area;
   vessel_vol, wall_vol    IS_A  volume;
   wall_thickness, H, D    IS_A  distance;
   H_to_D_ratio            IS_A  factor;
   metal_density           IS_A  mass_density;
   metal_mass              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;
```

Figure 2-4 First version of the type definition for *vessel.* (Available as *vesselPlain.a4c* in the ASCEND model library)

Our model definition has the following structure for it so far:

- MODEL statement
- list of variable we intend to use in the type definition
- equations
- END statement

While we have put the statements in this order, we could mix up and intermix the middle two types of statements, even going to the extreme of defining the variables after we first use them. The MODEL and END statements begin and end the type definition.

You should see little that surprises you in the syntax here. However, you may have noted that we have created a definition that says absolutely nothing about how to use the variables and equations listed. There is no solution procedure buried in this type definition. In ASCEND the idea of solving is separate from saying what we intend to solve. Also note that we have not said anything about the values for any of the variables nor what we intend to calculate and what variables we intend to treat as fixed input.

2.2 EDITING, COMPILING AND BROWSING AN ASCEND MODEL

Could we compile an instance of a vessel given this definition? If there had been some arrays in our definition for which we did not say how many items were in the arrays, we could not. However, here we could compile an instance, putting aside storage space for each of the variables and somehow capturing the equations relating them.

please do not alter the **models** subdirectory

When we compile new models, we need a place to store them. One possibility would be to put them into the **models** subdirectory of the ASCEND installation (e.g., in .../ASCEND/ascend4/models/). However, you really should leave the contents of this subdirectory untouched—always. You might think of it as being read only. We count on being able to replace it totally every time you install a new version of ASCEND. Whenever we add new model libraries or corrected versions of previously existing model libraries, we put them in this

subdirectory. ASCEND does nothing to enforce this rule while you run it, but please do not blame us if an upgrade wipes out changes you made in ascend4/models/; we warned you.

rather put your things into the **ascdata** subdirectory (you own it)

To avoid this problem, ASCEND also creates a subdirectory called **ascdata** that it will not touch when you install a new version of ASCEND. It will look in this subdirectory first when looking for a file to load when you have not given a full path name for finding that file. The install process for ASCEND will place **ascdata** into your home directory[1]. ASCEND tells you where it has placed this subdirectory when you start it. If you forget where it is, press the "About ASCEND IV" button on the Script window Help menu and look below the GNU ASCEND picture. It should say something like:

```
USER DATA DIRECTORY /usr0/ballan/ascdata
```

It is within the folder **ascdata** that you should place any ASCEND models you create. When running a script (which we shall talk about later), ASCEND first looks in this subdirectory for files, and then it looks in the **models** subdirectory. It stops looking when it finds the first available version of the file. For further details on this search, see Chapter 5.

create a text file containing the model definition

Next open an editor, such as Word, FrameMaker, emacs, pico, vi, vim, Notepad or Wordpad. Now type in or, better yet, cut and paste in the statements in Figure 2-4. Be very careful to match the use of capital and small letters. Do not worry about blanks between symbols but do not embed blanks within symbols. In other words, do not put a blank in the middle of the symbol *side_wall* but do not worry about putting zero or more blanks between *side_wall* and = in an equation.

When you are finished, be sure to save the file as a text file (e.g., on a PC as a .txt file). Call it vesselPlain.a4c. The ".a4c" stands for "ASCEND IV code." Editors such as Word and FrameMaker require you to use the *Save As* method to save and then to choose the file type to be *text*. Microsoft editors will append ".txt" to the file name. Remove the .txt ending off the file name -- do not let Microsoft bully you into thinking you should not -- and change it to ".a4c".

(This model is also available as *vesselPlain.a4c* in the ASCEND models library, but we suggest it would be better for you to go through

1. On Windows 95, you can identify a subdirectory to be your home directory by adding a line of the form "SET HOME=FullPathNameToSubdirectory" to the file c:\autoexec.bat. Add it without the quotes, replacing the right hand side with the full path name to the desired home directory - e.g., SET HOME=c:\mydocu~1\1998\. On UNIX and NT systems, your home directory is likely pretty obvious.

the exercise of creating your own version here. At the least copy the library file to your ASCEND space so you can play with your own version at this time.)

When you are done, you should have a text file called *vesselPlain.a4c* stored in your *ascdata* subdirectory. It should contain precisely the statements in Figure 2-4 with care having been taken to match capital and lower case letters as shown there.

start the ASCEND system. Move and resize the windows to make yourself comfortable.

Start the ASCEND system by double clicking on the ASCEND icon if you are on a PC or typing ascend at the command line if you are using a UNIX machine. Four windows[2] will appear, three smaller ones and one larger one that tells you about ASCEND. You can close this larger window by pressing its *dismiss* button. Move the three smaller ones around on your screen so they do not overlap or so they overlap very little. Resize them if you want to. You might start by putting the one called **Script** in the upper left, the one called **Library** in the upper right and the one called **Console** in the lower right. We shall assume you have placed them in these positions in the following so, even if that is not your favorite placement, it might be useful to use it for now.

The Script window shows the license and warranty information for ASCEND: ASCEND is protected by the GNU General Public License Version 2 and comes with absolutely no warranty.

note that each window by itself looks pretty nonthreatening

As you can see, each window by itself looks like a pretty normal window. Each has buttons across the top under which one will find different tools to run. Each also has one to three subwindows for displaying things. Each has a *Help* button that you can push at any time that you want to read all kinds of detailed things about the window. For the moment we will provide you with the “just in time” details here so you do not need to be sidetracked just yet by pushing these *Help* buttons.

hey, where did that window go? I want it back NOW!

If you ever lose a window, open the **Script** window and under the *Tools* button, select the window you wish to open. You cannot lose the **Script** window unless you shut down ASCEND. In the upper right of each window are Window 95/NT like buttons that iconify, enlarge and close the windows (underscore, box and X respectively). Picking X will remove the window from your screen. You get it back by going to the **Script** as described above or, as you will discover, by exporting something to it.

2. UNIX users of ASCEND will only see three windows appear. The xterm where you started ASCEND replaces the **Console** window.

I want to go to dinner (or I just panicked when I saw four windows). **How do I quit ASCEND?**

Picking the small X box in the upper right for the **Script** window is a first step in exiting ASCEND. Try it but hit the cancel button when you are asked to confirm your desire to leave. It always pays to know how (not just when) to quit. If you want to get the Script window out of the way, iconify it (pick the underscore button at the top right of the window). You will have to know how to recover an iconified window to retrieve it later - usually a simple single or double click on the icon does the trick.

saving window positions

If you like where you have placed the windows for ASCEND on your display, you can get ASCEND to remember all their locations by going to the **Script** window and selecting *Save all appearances* under the *View* button. A similar tool exists for each window for saving only its position.

start by loading and compiling using tools in the Library window

We shall start with the **Library** window in the upper right. This window provides you with the tools to load and compile files containing type definitions. You can also display the code for the different types you have loaded.

use the left mouse button unless we tell you otherwise (however, on you own explore using the right mouse button in any of the windows)

Let's load your file. Under the *File* button select the *Read types from file* tool. You select this tool by clicking on it using the left mouse button - i.e., the button you should have expected to use. A window will appear asking you to find the file you want to read into ASCEND. Navigate to where you stored *vesselPlain.a4c* (in the subdirectory *ascdata*) and select that file. If you have the wrong ending on the file (you left *.txt* or you forgot to put *.a4c* as the ending), tell the system to list all files and pick the one you want. The *.a4c* is used by the system to list only the files it thinks you might want to load, but ASCEND isn't fussy. It will attempt to load any file you pick.

Look in the Console window at the lower right, and, if the file loads without any errors being listed there, you should see

```
AscendIV% REQUIREing file "atoms.a4l"
REQUIREing file "system.a4l"
REQUIREing file "basemodel.a4l"
REQUIREing file "measures.a4l"
```

If this is what you see, you can skip past the next bit to where you should start to compile an instance. The next bit has some useful hints on how to debug your models. If you want some debugging experience, put a known error into your *vesselPlain.a4c* file and see what happens. This move will give you a reason to read the following section.

DO NOT ignore the diagnostics that might appear in the **Console** window

If the Console window in the lower right starts filling with several tens of lines of diagnostics, look to see if you included the REQUIRE statement at the beginning of your model file. Without that statement, ASCEND is missing all the definitions for the types of variables in your model, and it will go wild telling you so. It might also be choking on a Word document because you forgot to save it as a text file.

While loading the files containing these types, ASCEND will look very closely at the syntax and will give you all kinds of diagnostic messages in the Console window (lower right) if you have done something wrong. It will also at times spew out some warning messages if you have done something thought to be poor modeling style. You must heed the error messages as the file will not load if there are any. ASCEND will tell you if it did not load the file.

You should consider heeding the warnings if you get any. If you ignore them now, they may come back and haunt you later. However, there are times when we issue a warning but everything will work, and you will think we were not too clever. Our response: better modeling style can eliminate these warnings. (It's been our system so we get to have the last word.)

how do I jump to line 100 of a file when using some of the standard editors?

The error and warning messages will contain a line number in the file where the error has occurred. This will be the line number as counted by an editor with the first line being line 1 in the file. Editors always provide you with a means to get directly to a line number in a file. Find out how to do that or you will not be too happy with debugging a large file. For example, in emacs, type a *Ctrl-c* (type the letter *c* while holding down the *Ctrl* key) followed by the letter *g*, then a line number and a carriage return. In Word and FrameMaker on the PC, type *Ctrl-g* and follow the instructions. For FrameMaker on UNIX, find the *Go to Page* tool and open it (Esc-v p or look under *View*).

You will be in the debug mode for a new system so do not expect it to be totally obvious the first few times you make an error. We have tried to use language that should be meaningful, but we may have failed or the error may be pretty subtle and not possible for us to anticipate how to describe it in your terms. (Send us a bug report if you have any good ideas on language.)

reloading a file overwrites the previous version

You can reload any file your have corrected using the *Read types from file* tool under the *File* button. It will overwrite the previous version of the file only if the file has changed since it was last loaded (pretty clever, right -- we do not reload those big files unless you make a change even if you tell us to).

displaying the code

You can display the code you have written. Select the model vessel in the right window of the Library. Then under the Display button at the top, select the tool Code. The Display window will open displaying the code for this model.

now compile as "*v*"

Okay, you have your file loaded without getting any diagnostics. You are ready to compile. In the **Library** window, look in the left window and select the file *vesselPlain.a4c*. It contains the type definition you wish to compile. You should see the type *vessel* appear in the right window. Select *vessel*. Under the *Edit* button, select *Create simulation.* A small window opens and asks you to name the simulation. Call it "*v*" -- yes, just the letter "*v*" and select "*OK*." Short names for instances often seem to be preferable.

Look again in the **Console** window for diagnostics. If everything worked without error, you will see some statistics telling you how many models, relations and so forth you have created during the compile step.

You may see the following message in the **Console** window:

```
Found STOP statement in METHOD basemodel.a4l:239
  STOP {Error! Standard method "default_self" called but
not written in MODEL.};
  In call to METHOD default_self (depth 1) in instance v
    Line 239, File: basemodel.a4l.
```

You can safely ignore this message for now. In the next chapter, we will discuss writing methods and the meaning of this message.

and pass the instance to the **Browser**

Select *v is a vessel* in the bottom of the **Library** window. Then under the *Export* button, select *Simulation to Browser* to export *v* to the **Browser** tool set. The **Browser** window will open and contain *v.* It might be useful to enlarge this window and move it down a bit, placing it a bit to the right of the center of your computer display. (Remember you can save this positioning and sizing of the **Browser** window by going under the *View* button and picking *Save appearance*.)

examine *v* by playing with it in the **Browser**

In the left upper window of the **Browser**, you will find *v* to be the current object. Listed in the right window are all the parts of the current object. You will see the variables listed here along with an indication of their type. For example, you will find *Cylinder IS A relation* and *D IS A distance* listed, among many others. *Cylinder* is one of the equations you wrote describing the model while *D* was the diameter of the vessel.

included flags for relations

If you pick any of the parts in the right or bottom windows, it becomes the current object; its parts then show in the right window. For example, a relation has a *boolean* part (a flag that takes the value TRUE or FALSE) indicating whether or not it is to be included when ASCEND solves the equations you defined for the model.

If you wish to display the current value for this flag, pick the tool *Display Atom Values* under the *View* button. This tool toggles a switch that causes either the value or the type to show for a variable, a constant or a relation in the upper right window of the **Browser**. Try toggling it back and forth and looking at different things in the **Browser**.

Pick each of the tools under *View* and note what happens to the displaying of things in the **Browser**.

Across the bottom of the **Browser** window note the buttons you can select labeled *RV, DV* and so forth. If you have made the **Browser** window large enough, you will see to the right of these buttons the type of objects whose value you want to appear or not in the lower **Browser** window as you toggle each button. Toggle each of these buttons and see if the lower display changes. If it does not, then this type of part is not in the current object.

2.3 SOLVING AN ASCEND INSTANCE

Well, you have been patient. While there are lots of interesting tools left to explore in the **Browser**, perhaps it is time to try to solve this model. To solve *v*, make it the current object (it alone should be listed in the upper left window of the **Browser**). Then, under the *Export* button, select *to Solver.* The **Solver** window will open, along with a smaller window labeled **Eligible**. Move the **Eligible** window up a bit so it does not cover any or very little of the **Solver** window. Move the **Solver** window to the lower left and enlarge it so you can see all of its contents.

if ASCEND stops responding, hunt down one of those “nasty” windows with a “yellow lock” and close it properly

This Eligible window is one of the “nasty” ones. If it is open and you do not do something to make it happy and go away, it will stop you from doing anything else in the ASCEND system. Such windows appear with a black lock icon in a yellow field -- we shall call it a “yellow lock.” They demand you attend to them NOW. A good solution would be for such a window to stay open and on top of all the other open windows. Unfortunately we have not been able under all window managers to stop it from ducking under another window. If you ever find ASCEND unwilling to respond, iconify the other windows to get them out of the way, until you find one of these windows. On the PC you can go to the icon bar at the bottom of your screen and, by clicking

on the window, bring it to the top. Then do whatever it takes to make it happy and close properly -- such as cancel it. If you are not careful here, for example, this window will hide under the **Solver** window before you are through with it.

is our problem well posed?

The **Solver** window contains the information we need to see to explain why the **Eligible** window opened in the first place. Examine the information the **Solver** displays. It tells you that *v* has 6 relations defining it and that all are equalities and included. It has no inequalities. On the right side we see there are 10 variables and all are “free.” A free variable is one for which you want the system to compute a value. Hmm, 6 equations in 10 variables. Something is wrong here. For a well-posed problem, you want 6 equations in 6 variables (i.e., square). ASCEND reports that the system is underspecified by 4. This means you need to pick four of the variables and declare them to be fixed. You will also have to pick values for these fixed variables before you can solve for the remaining 6. For such a small problem as this one, this task is not formidable. For a model with 50,000 equations and 60,000 variables, one would quit and go home. We have exposed a need here. We certainly would like ASCEND to help us here for this small problem. But we insist that it help us in major ways to make the 50,000 equation, 60,000 variable problem possible.

picking variables we are going to fix

Okay, the small help such as needed here is why the **Eligible** window opened. Let’s return to it. It lists all the variables of those not yet fixed that are eligible to be fixed and still leave us a calculation that has a chance to solve. The very fast algorithm to find eligible variables does an analysis of the structure of the equations. It cannot guarantee the resulting problem is numerically well-posed, but picking a variable it does not list as one to fix will guarantee the problem is numerically singular. Good luck on solving it if it is. We will go for coffee rather than wait for you to succeed.

So look at the list and decide what you would like to fix for your first calculation with this model. Diameter (*v.D*) seems a good choice. Now you can see why we called the instance just plain old *v*. A longer name would get tiring here. Anyway, pick *v.D*. Immediately the list reappears with *v.D* no longer on it. (ASCEND has just repeated the eligibility analysis.)

We have three more to pick. On the list are both vessel height, *v.H*, and *v.H_to_D_ratio*. We certainly cannot pick both of these. One implies the other if we know a value for *v.D*. Pick *v.H_to_D_ratio*. Note that *v.H* is no longer eligible. Good. We would be worried if it were still there.

We see *v.metal_density*. Pick it. Strange. Metal mass and volume stayed eligible. Well, okay. If we pick metal mass, wall thickness is implied, and the same is true if we were to pick metal volume. However, it seems much more natural to pick *wall_thickness* so make that the last variable picked. The **Solver** window now says this problem is square (i.e., it has 6 equations in the same number of unknowns). Table 2-3 summarizes the four variables we have elected here to fix.

Table 2-3 Variables we have fixed

variable
D
H_to_D_ratio
metal_density
wall_thickness

ASCEND partitions the problem into smaller problems for solving

Toward the bottom right of the **Solver** window, we see there are 6 "blocks." What are blocks? ASCEND has examined the equations and, in this case, has discovered that not all the equations have to be solved simultaneously. There are 6 blocks of equations which it can solve in sequence. 6 blocks and 6 equations means that ASCEND has found a way to solve the model by solving 6 individual equations in sequence -- i.e., one at a time. That is great.

But ASCEND is going to be even smarter than this about solving in this case. If an equation is being solved by itself and if it is simple enough algebraically, ASCEND will rearrange it and solve directly for the one variable that is not yet calculated in it -- without iteration. Here all the equations are in fact that simple. This problem, with the 4 variables we selected to be fixed, can be solved entirely without iteration.

displaying the incidence matrix

Can we see what ASCEND has just discovered? It turns out we can (we would not have asked if we could not). Under the *Display* button on the **Solver**, select the *Incidence matrix* tool. A window pops open showing us the incidence of variables in the equations and display them in the order that ASCEND has found to solve them. The dark squares are incidences under the variables for which we are solving; the lighter looking X's to the right side are incidences for the variables we have fixed. Click on the incidence in the upper left corner. ASCEND immediately identifies it for us as the end_area. It identifies the equation as the one we labeled FlatEnds. We can go back to our model and find the equation ASCEND will solve first. The other variable in this equation is in the set we fixed; pick it and discover it is D, the vessel diameter. Of course we can compute the area of the ends given the diameter. The end_area is $\pi * D^2/4$.

Play with the other incidences here. See what the other equations are and the order ASCEND will use to solve them.

Okay, we return to our task of solving. We need next to supply values for the variables we have selected to be fixed. Again, the approach we are going to take is acceptable for this small problem, but we would not want to have to do what we are about to do for a large problem. Fortunately, we really have thought about these issues and have some nice approaches that work even for extremely large problem -- like 100,000 equations.

which variables are currently fixed for this problem?

Let's see. Do you remember the variables we fixed? What if you do not? Well, we go back to the **Browser**. Be sure *v* remains the current object (it alone is in the upper left window). Under the button *Find* pick the *By type* tool. A small window opens with default information in it saying it will find for us all objects contained in the current object *v* of type *solver_var* whose fixed flags are set to *TRUE*. These are precisely the attributes for the variables we have fixed. Select *OK* and a list of the four variables we fixed earlier appears.

specifying values for the fixed variables - this approach is useful for small problems

For each variable on this list, we should supply a value. Select D in the lower window of the **Browser** using the right (the right, not the left -- make *v* the current object and do it again) mouse button. A window opens in which we input a value for *D*. Put in the value **4** in the left window and **ft** in the right. Continue by putting in the values for the variables as listed in Table 2-4. These values immediately appear in the Browser window as you enter them. If you did not fully appreciate the proper handling of dimension and units before, you just got a taste of its advantages. YOU did not have to worry about specifying these

Table 2-4 Values to use for fixed variables

variable	value	units
D	4	ft
H_to_D_ratio	3	
metal_density	5000	kg/m^3
wall_thickness	5	mm

things in consistent preselected units.

You can now solve this model. Go the **Solver** window and, under the *Execute* button, pick *Solve*. You will get a message telling you the model solved. Dismiss that message and return to the **Browser** window to examine the results. You should see the following results

```
D = 1.21922 meter
H = 3.65765 meter
H_to_D_ratio = 3
end_area = 1.16748 meter^2
metal_density = 5000 kilogram/meter^3
metal_mass = 408.62 kilogram
side_area = 14.0098 meter^2
vessel_vol = 4.27025 meter^3
wall_thickness = 0.005 meter
wall_vol = 0.0817239 meter^3
```

alter the units used for displaying values

You may wish to alter the units used to display these results. For example, you enter the diameter *D* in ft. You may wish to reassure yourself the 1.21922 meter is 4 ft. Go to the **Script** window and under the *Tools* button select *Measuring units*. The **Units** window will open. Enlarge it appropriately and then place it to the top and far right of your display.

Since length is a basic dimension in ASCEND, there is only one way to change the units for displaying length: under the *Edit* button select *Set basic units*; a cascading menu will appear, select *Length*. Another cascading menu will open with all the alternate units supported in ASCEND for length. Select *ft*. The units for all length variables will switch to *ft*. Look at the values in the **Browser** window.

The left upper window of the **Units** window contains many variable types that have composite dimensions. For example, you will find volume there. Pick it and the right window fills with all the alternative units in which you can express volume.

Play with changing the units for displaying the various variables in the vessel instance *v*.

One point - the left window displaying types having composite dimensions will display only one type for each composite dimension. If the atom types you have loaded were to include volume_scale as well as volume, then only one of the two types, volume or volume_scale, will be listed here. Changing the units to express either changes the units for both.

returning to a consistent set of units

When you are done, you may wish to return to a consistent set, such as SI. Under the *View* button are different sets; pick *SI (MKS) set*.

now we can solve the model in other ways

We can now resolve our vessel instance in any number of different ways. For example we can ask what the diameter would be if we had a volume of *250 ft*^{3}. To accomplish this calculation, we need first to make

vessel_volume a variable whose value we wish to fix. When we do this the model will be overspecified. ASCEND will indicate this problem to us and offer us a list of variables - including the vessel diameter *D*, one of which we will have to "unfix." Finally we need to alter the value of *vessel_volume* to the desired value and solve. Explicit instructions to accomplish these steps are as follows.

- In the **Browser** window, make *vessel_volume* the current object (select it using the left mouse button). The right window of the **Browser** display the parts of the *vessel_volume*, among them is the *fixed* flag with a value of *FALSE*.
- (If you do not see the value for *fixed* but rather its type as a *boolean*, under the *View* button at the top, select *Display Atom Values*.)
- Pick *fixed* with the right mouse button, and, in the small window that opens, delete the value FALSE, enter the value *TRUE* and select OK.
- Now make *v* the current object by picking it in the left window of the **Browser**.
- Export *v* to the **Solver** again by selecting *to Solver* under the *Export* button. A window entitled **Overspecified** will appear listing the variables *v.D*, *v.H_to_D_ratio* and *v.vessel_volume*. Pick *v.D* and hit the *OK* button; ASCEND will reset its fixed flag to *FALSE*.
- Finally, return to the **Browser** window and select *vessel_volume* with the right mouse button. In the small window that appears type *250* in the left window, *ft^3* in the right, and hit the *OK* button.
- Under the *Execute* button in the Solver window, select Solve.

Note the **Solver** reports only 4 blocks for 6 equations. This time it has to solve some equations simultaneously. In the **Solver** window, under the *Display* button, select the *Incidence matrix* tool. You will see that the first three equations must be solved together as a single block of equations.

clearing all the *fixed* flags

For a more complicated model you may wish to start over on the process of selecting which variables are fixed. You can set the *fixed* flags for all the variables in a problem to *FALSE* all at once -- without knowing which are currently set to *TRUE*. In the **Browser** window, under the *Edit* button, select the *Run method* tool. A window will open that displays a list of default methods that are automatically attached to every model in ASCEND. One is called *ClearAll*. Pick it and hit *OK*.

All the fixed flags for the entire model will now be reset to *FALSE*. Can you think of a way to check if this is true? (Do you remember how to check which variables are currently fixed? Repeat that check and you should find no variables are on the list.)

You might now want to play by changing what you calculate and fix.

2.4 DISCUSSION

You have just completed the creation and solving of a very small model in ASCEND. In doing so, you have been exposed to some interesting issues. How can we separate the concept of the model from how we intend to solve it? How do we make a model to be well-posed -- i.e., a model involving *n* equations in *n* unknowns -- so we can solve it? How should one handle the units for the variables in a modeling system? What we have shown you here is for a small model. We still need to show you how one can make a large model well-posed, for example. You will start to understand how one can do this in the next chapter.

The next chapter is crucial for you to understand if you want to begin to understand how we approach good modeling practice. Please do continue with it. As it uses the vessel model, it would, of course, be best to continue with that chapter now.