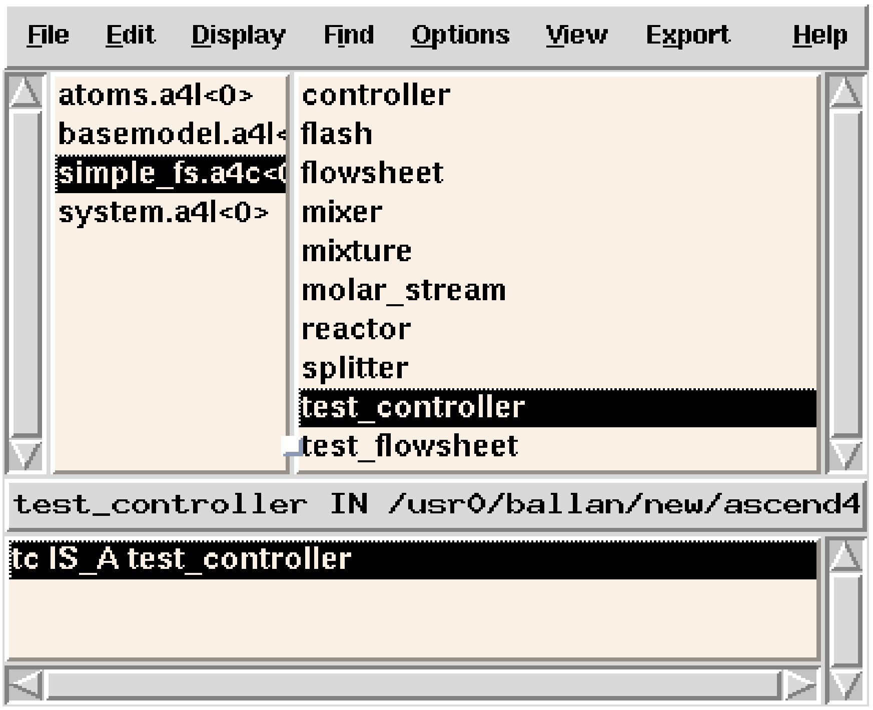
CHAPTER 4 LIBRARY

The Library window (Figure 4-1) in ASCEND allows the user to read *types* into the ASCEND system from files, *compile* types into instances, and delete types.

Types are the templates used to create simulations. They come in two flavors: ATOM, which has a value associated with the instance name when it is instantiated, and MODEL, which has no value. ATOMs are the variables and constants in ASCEND; MODELs are the complex structures one can build in ASCEND. ATOMs, further, come in vanilla



atoms.a4l<0>
basemodel.a4l<
simple_fs.a4c<(
system.a4l<0>

controller
flash
flowsheet
mixer
mixture
molar_stream
reactor
splitter
test_controller
test_flowsheet

test_controller IN /usr0/ballan/new/ascend4

tc IS_A test_controller

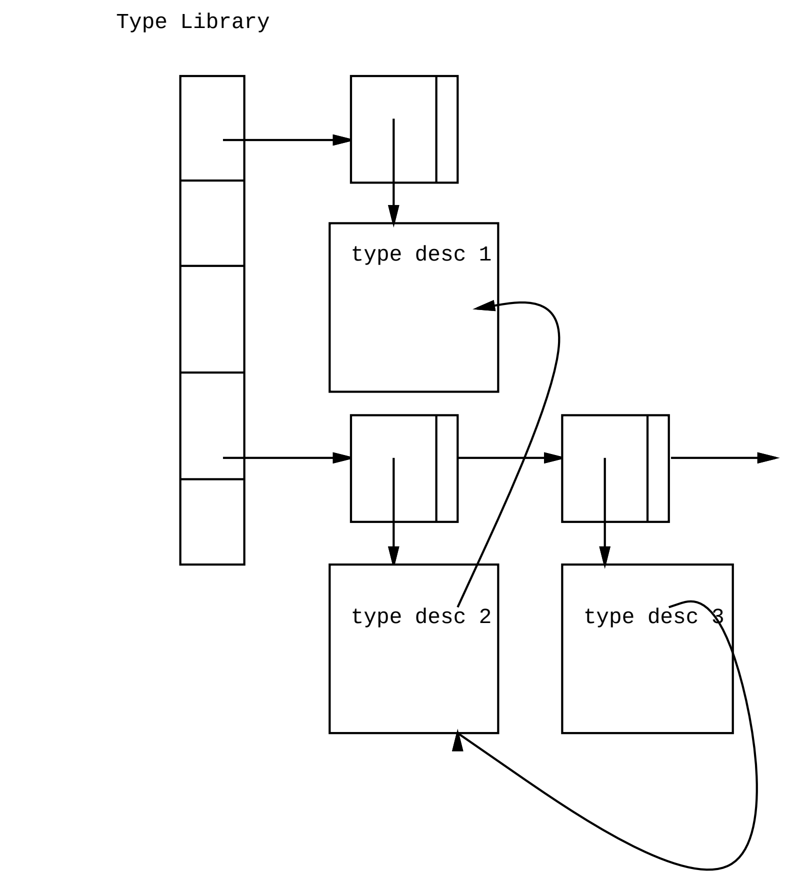
Figure 4-1 ASCEND Library Window.

and UNIVERSAL flavors. Universal atoms have a single compiled instance which is global to all simulations created.

Both ATOMS and MODELS are defined in source files. By convention, source files are named with the endings `.a4c` (ASCEND IV code) and `.a4l` (ASCEND IV library). You are free to use any other ending, but you will find the ASCEND file type filters you use when browsing for files will be ineffectual.

In the ASCEND Library window, source files appears in the upper left box. On the other hand, the types defined in the highlighted source file appear in the upper right box. A double-button2 in either box will compile the highlighted type definition. It doesn't reselect. The upper left box should perhaps have double-button2 bound to reread the selected source module. The ASCEND fundamental type such as integer, real, etc., are not shown in the library window, since their definition is performed internally, not by using a specific source file. The lower box of the ASCEND Library window contains the name of the simulations that have been compiled and can be run.

The data structure used to store type definitions is sketched in Figure 4-2.



```
Notes: type desc3 has a refinement ptr to type desc 2
       type desc2 has a refinement ptr to type desc 1

       The problem is when type desc 2 is being redefined
       by reloading a new module.
```

Figure 4-2 Data structure used to store type definitions.

4.1 MENU BAR

The menu bar on the Library window has eight entries: File, Edit, Display, Find, Options, View, Export and Help.

4.1.1 THE FILE MENU

Read types from file

This loads type definitions into the system. The file selection dialog is used to select a source file.

The names of types are unique within the system. A new definition of a type overwrites the old definition of a type in all cases. If the new definition and the old definition were read from files of the same name, this overwrite will be done silently. If the new definition comes from a different file, the overwrite will be done noisily.

This is incorrect, but perhaps is as it ought to be. *Existing types which refined or had parts that were of the old type definition will now refine or have parts which are of the new type. e.g. If you reread system.asc (and hence solver_var) everybody in the interface library who pointed at the old solver_var type will now point at the new solver_var type*.

Instances already compiled using the definitions that have been overwritten will continue to point at a copy of the old definition the system has squirreled away somewhere. These squirreled away copies will not necessarily be the same as what is in the interface type library if you have reread a file with a newer type definition. This may cause refinement of the old instance to fail. In general if you redefine a type, you will probably want to reinstantiate things that depend on that type.

`Close window` It closes the ASCEND's Library window. To reopen, use the Tools menu in the Script window or use the SCRIPT tool in the Toolbar window.

`Exit ASCEND` Exit the ASCEND system. You will be asked to confirm that you wish to do this.

4.1.2 THE EDIT MENU

`Create simulation` Create (or instantiate) a simulation based on a type definition. Anytime that the compile button is selected, the compile dialog window shown in Figure 4-3 will ask for the name which will be used to identify the simulation. All simulation created can be seen and in the lower box of the ASCEND Library window. This box can contain any number of simulations.

`Suggest methods` Pick any type in the right window and apply this tool to write suggested methods for that type. See the Howto book, Chapter 2, for a list of the methods we suggest one should write for models. These suggested methods are prototypes which you can cut and paste into your favorite text editor. You should be carefully edit them before adding them to the model type definition.

`Delete Simulation` This tool works to remove previously compiled simulations listed in the bottom subwindow of the Library window. Select a simulation to delete and use this tool to eliminate it from ASCEND. ASCEND will clear the model from the Browser and Solver. Items placed into the Probe window are not deleted.

`Delete all types` Destroys all simulations and deletes all types. This option has no effect in the fundamental definitions. Deleting all types clears the Browser and Solver windows, but not the Probe window.

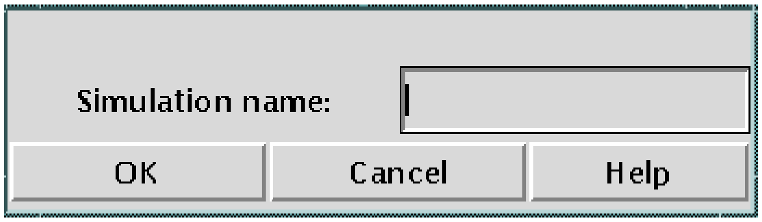


Figure 4-3 The Create Simulation Dialog

4.1.3 THE DISPLAY MENU

Most of the options in the Display Menu will be enabled only if a type definition has been selected; this is because the tasks performed in the menu are implicitly associated with a type definition.

`Code` Displays the source code of the selected type in the ASCEND Display Window.

`Ancestry` Allows the use of the Type Refinement Hierarchy Window. See Section 4.2 on page 46 documenting this window.

`Refinement hierarchy` Displays the refinement hierarchy of the selected type in the ASCEND Display window.

`External functions` Display in the Display window any external function defined from a loaded package library.

`Hide type` The Browser will not display any type definition which you select to be hidden with this tool. You may select to hide the type or the type and all its refinements. For example, doing the latter with `solver_var` will hide all variables in a compiled instance of the model.

`UnHide type` Reverses the action of “hiding” a type. Select the type in the right window and select unhide if that tool is lit. The Browser will immediately begin to display this previously hidden type. The default for all the type definitions (except fundamentals) is to be “unhidden”.

Both Hide Type and UnHide Type have two selections as a submenu. The user can ask for the un/hiding of only the **selected type**, or for the un/hiding of the selected **type and its refinements**.

Hide/Show Fundamentals

This special option is given because fundamental types do not appear as definitions in the ascend libraries, but we still may want to able/enable such types for browsing purposes. When this button is selected, the window shown in Figure 4-4 will be used to perform the desired hiding or unhiding of any of the fundamental types.

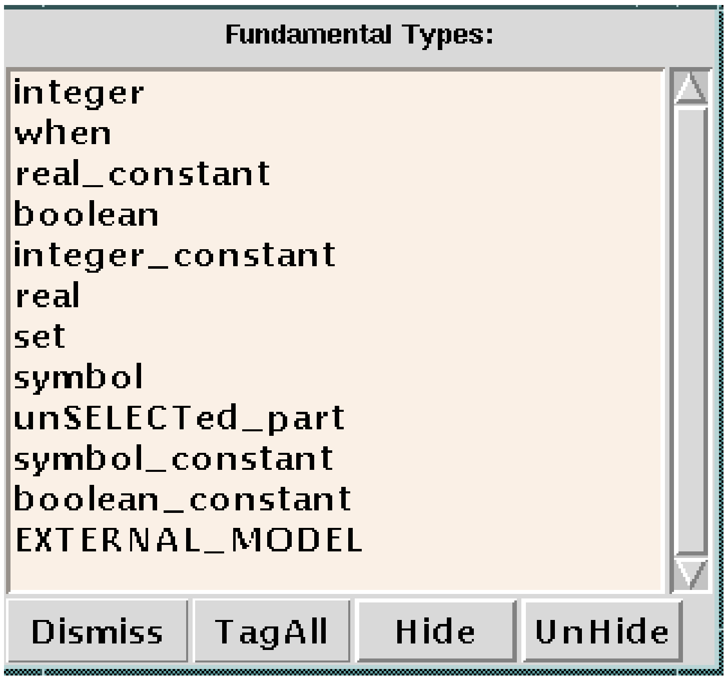


Figure 4-4 Select the fundamental type to Hide or Unhide.

4.1.4 THE FIND MENU

ATOM by units

This extremely handy tool allows you to find all the ATOMS that are currently loaded into the library whose dimensionality conforms to a user specified set of units. For example, if you have loaded the library atoms.a4l into the Library, then you can use this tool to find all atom definitions that could be expressed in ft^3 or in kJ/mol. To use, select the tool. In the window that opens type in the units and select OK. You do have to know the units ASCEND will recognize. Open the Units window and under the Display menu select Show all units for a complete list.

Type by name

Finds a type by its name. The type will become the current type highlighted in the Library right and left upper windows.

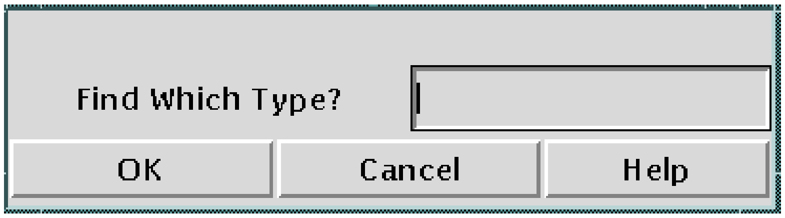


Figure 4-5 The Library's Find Type dialog.

Type by fuzzy name Finds all type names currently loaded in the Library window that match a word (provided by the user) in any fuzzy way. For instance, the name *column* would list the following: demo_column, mw_demo_column, plot_column, etc. if these were currently loaded in the Library. The fuzzy name is defined in a dialog window similar to that used in the Find Type by name option.

Pending statements There are three selections under the Pending Statements submenu, these are **To Display**, **To Console**, and **To File**. Pendings in a simulation are relations that have not yet been fully processed by ASCEND's compiler. It is the modeler's job to correct the pending relations in order to arrive at a fully functional simulation. Corrections may be made by either creating a model which refines the current model or by editing ASCEND code and starting over. This option gives the user access to information about the type and location of the pending statements. Often pending statements arise from a common cause such as a incorrectly qualiified or misspelled name for a set.

To Display By selecting the **To Display** option, all of the simulation pendings are displayed in the *Display* window.

To Console By selecting the **To Console** option, all of the simulation pendings are displayed in the Console window (in UNIX, the Console is the window from which you started ASCEND IV).

To File By selecting the **To File** option, the *File select box* is opened and the user is asked to enter the name of the file in which to save the model pendings.

4.1.5 THE OPTIONS MENU

The titles for most of these tools more or less describes their purpose. We will not describe them With these options, you can turn on or off messages ASCEND will generate while compiling. Turning off warning and error messages will, of course, mean that you will not be told about problems your model may have that we were able to detect.

Since ASCEND will still compile in spite of warning messages (which generally reflect your model does not conform to what we believe to be good modeling practice), you may wish to suppress them. See the Howto Book, Chapter 2 for a discussion of good modeling practice.

We describe only those tools that do not turn on and off compiler messages.

`Generate C binary` If you have a C compiler installed and ASCEND knows about it, then you may elect to have ASCEND compile C code to evaluate equation residuals in ASCEND. The compiler finds and will, when this option is selected and possible, compile only a piece of code for each unique equation type, of which there are very few in any model. The evaluation of residuals will be much faster using compiled C code.

`Simplify compiled equations` This option is on by default. ASCEND will reduce terms in equations such as a product of constants whose values it knows to a single resultant constant when you select this option. Whole terms in equations may disappear if ASCEND finds them multiplied by the constant zero.

`Save options` Save the current setting for all the options selected using items in this menu. ASCEND put the saved information into a text file *library_opt.a4o* and which it then saves in the *ascdata* subdirectory of your “home” directory.

4.1.6 THE VIEW MENU

`Font` Opens the window that lets you reset the fonts for this window. You can select the type of font, the style (bold, etc.) and the size for the font.

`Open automatically` Toggles a switch which, if set, will cause this window to open whenever anything is placed into it.

`Save appearance` Saves the current settings for this window for font settings and window size and placement on your computer screen. These become the default settings for opening this window in the future. These settings are saved in a *.a4o* text file for this window which the sytem stores in the subdirectory *ascdata* in your “home” directory.

4.1.7 THE EXPORT MENU

There are three selections under this submenu, these are **Simulation to Browser**, **Simulation to Solver**, and **Simulation to Probe**.

`Simulation to Browser` By selecting the **Simulation to Browser** option, the simulation highlighted in the lower box of the Library window is loaded into the *Browser*. From the *Browser*, the model can be explored in more detail.

`Simulation to Solver` By selecting the **Simulation to Solver** option, the simulation highlighted in the lower box of the Library window is loaded into the *Solver*. (Note that exporting to the solver causes a degrees of freedom analysis to be carried out.)

`Simulation to Probe` By selecting the **Simulation to Probe** option, all of the variables of the simulation highlighted in the lower box of the Library window are loaded into the *Probe*. This is not recommended as there are usually more variables in a model than the user would wish to view at one time. However, if the user does wish to look at all of the variables and their current values, the **Simulation to Probe** option can be useful.

4.1.8 THE HELP MENU

`On LIBRARY` Brings up a text description of where to look for help on this window (i.e., it points to the pdf version of this document on the WWW.) You may, of course, look into the section mentioned in any local (but perhaps outdated) copy of the documentation.

4.2 TYPE REFINEMENT HIERARCHY WINDOW

The type tree is a directed acyclic graph (DAG) based on the type hierarchy currently defined in the interface Library. Selection of the tool Display Ancestry (mentioned above when describing tools under the Display menu) for any selected type gives the entire refinement hierarchy for that type, by enabling the use of the window shown in Figure 4-6.

The current focus in the hierarchy is indicated by a rectangle around the type name and the Current type.

The buttons on the left in the type window operate on the currently selected type:

‘Atoms’ shows the types of ATOMic parts in the selected type definition. It also shows the incremental code for the type. You can select from the part types list to look at a different hierarchy.

‘Code’ shows the internally stored code of the selected type. The expressions, both algebraic and logical, are in reverse Polish notation.

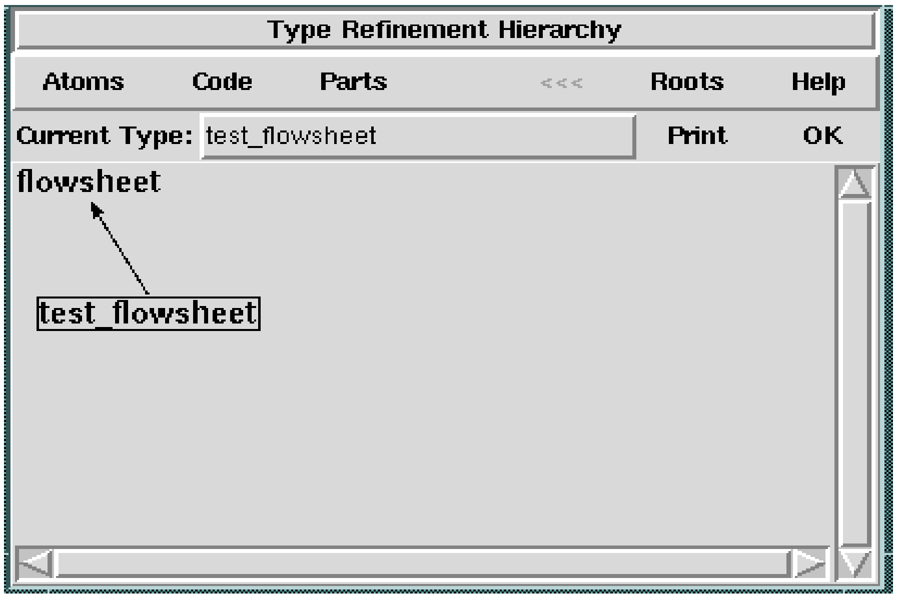


Figure 4-6 The Type Refinement Window.

This is different from the way the code of the Library Display Code button shows it. Comparison of the two is sometimes a useful debugging tool.

‘Parts’ (Figure 4-7)shows the types of MODEL parts in the selected

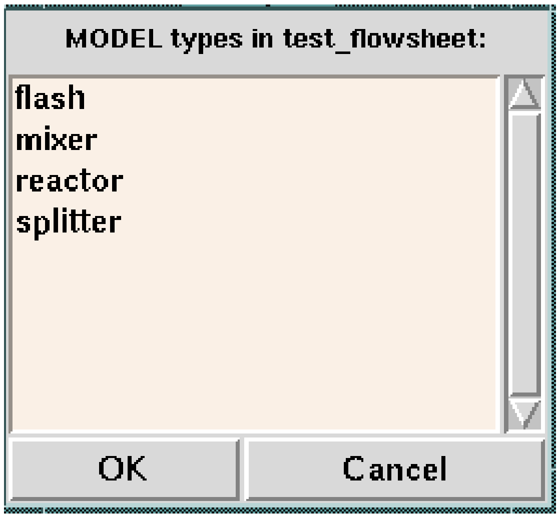


Figure4-7 The Parts window displays the parts.

type definition. It also shows the incremental code for the type.

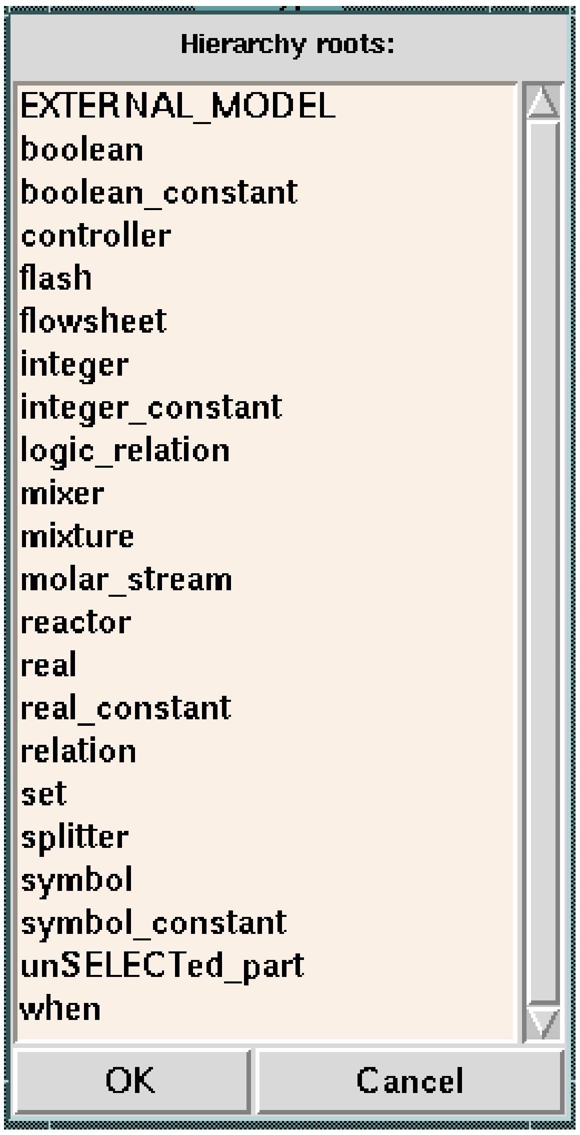
The ‘<<<‘ (or backtrack) button backs up to the previously displayed type hierarchy, if there is one.

‘Roots’ (Figure 4-8) shows the existing root types, that is, the existing types which are not refinements of anything.

While ASCEND is building the graph, you may see a spew in the window from which ASCEND was started about orphaned types. This means there are types in the Library which are refinements of older types which are no longer in the Library.

While ASCEND is getting the Atom or Model parts list for a type, part types names which are undefined will be spewed.

When an older type is replaced in the Library by a new one of the same name, the old one is squirreled away where types that refined it can still see it. The only way to get current types to look at the new definition without touching the source files for the current types is to delete all types and reread the entire Library.



EXTERNAL_MODEL
boolean
boolean_constant
controller
flash
flowsheet
integer
integer_constant
logic_relation
mixer
mixture
molar_stream
reactor
real
real_constant
relation
set
splitter
symbol
symbol_constant
unSELECTed_part
when

Figure 4-8 The Hierarchy Roots Window.