Chapter 6.
Using Scheme AIDE

Topic: *Scheme AIDE Application

Scheme ACIS Interface Driver Extension (Scheme AIDE) is a Scheme based ACIS demonstration application. It serves many purposes for ACIS developers:

- It provides a means of exercising ACIS functionality without writing or compiling a stand-alone C++ application. This helps developers learn and prototype functionality.
- The C++ source code is provided for the Scheme extensions used by Scheme AIDE. This can be used for insight into how to implement specific ACIS functionality in a developer's C++ based application.
- It can be used as an example of a Scheme based ACIS application, or a starting point for creating a new Scheme based ACIS application.
- It provides a means for customers to report problems against ACIS to *Spatial.*

Scheme AIDE is a command-line program that accepts Scheme commands at a *command prompt* and displays the results in a separate *view window.* The commands may be either native Scheme commands or ACIS Scheme command extensions.

This chapter describes how to run Scheme AIDE, and provides some simple examples.

Refer to the *Scheme Support Component Manual* for information on the Scheme language, Scheme procedures, and extending Scheme, and to the *Scheme AIDE Main Program Component Manual* for information on using Scheme AIDE as an example application. Refer to the reference templates for descriptions of the ACIS Scheme extensions and data types.

File and Directory Names

Topic: Scheme AIDE Application

This section describes the conventions used in naming files and directories associated with Scheme AIDE and the conventions used in presenting those names in the documentation.

Executable File

Topic: Scheme AIDE Application

The exact name of Scheme AIDE's executable file depends on your platform, and possibly the components you have installed. For documentation purposes, the "generic" filename acis3dt is used when referring to the name of the executable file if the context is not specific to a platform or installation.

Scheme Procedure Files

Topic: Scheme AIDE Application

Files named <*filename*>.scm are *Scheme procedure files*. These are text files containing one or more Scheme procedures that can be loaded into Scheme AIDE (or any another Scheme based ACIS application) and executed using the load command.

Pathnames

Topic: Scheme AIDE Application

This chapter includes instructions and examples that specify *pathnames*, which are the names of directories and files. The exact name of a directory or file may depend on the platform, the installation (e.g., components installed), or the ACIS release number. The following conventions are used to present these names in a form that is not specific to a particular installation, platform, or release:

<*install_dir*> . . Refers to your root ACIS installation directory. Substitute the actual name of your actual root installation directory for the string <*install_dir*> in pathnames.

$A3DT . . . . . Is an environment variable that points to your root ACIS installation directory, <*install_dir*>. This environment variable should be defined during installation. If it is not defined, substitute your root ACIS installation directory name for the string $A3DT in pathnames.

$ARCH . . . . . Is an environment variable that points to your platform directory (e.g., NT, hp700, sgi, sun4_cxx4, solaris_cxx4, ultrix, osf1, or aix4). This environment variable should be defined during installation. If it is not defined, substitute your platform name for the string $ARCH in pathnames.

/ . . . . . Pathnames are shown using the UNIX format unless the context is specific to the Windows or Macintosh environments. The UNIX format includes a forward slash delimiter after directory names (Windows uses a backslash).

Entering Scheme Commands

Topic: *Scheme AIDE Application

Scheme commands may be entered directly on the Scheme AIDE command line at the command prompt or loaded from a Scheme procedure file (<*filename*>.scm). Scheme commands are entered enclosed within parentheses. Any arguments for the command are also enclosed within these parentheses.

Note *The* name *of the Scheme command does* not *include the parentheses.*

A semicolon (;) at the beginning of the command line indicates the start of a comment. Comments are often used within Scheme procedure files.

Refer to the *Scheme Support Component Manual* for more information on Scheme commands (including syntax, expressions, conventions, etc.).

Starting Scheme AIDE

Topic: *Scheme AIDE Application

To start the Scheme AIDE demonstration application, you load and run the executable file. How you do this depends on your platform and environment, as described in following sections.

When you start the application, a command window in which you enter Scheme commands is automatically created. The command prompt appears in the command window as:

```
acis>
```

You create a view window by entering the view:dl Scheme extension at the prompt:

```
acis>(view:dl)
```

To terminate the application, enter the exit command at the prompt:

```
acis>(exit)
```

UNIX Platform

Topic: *Scheme AIDE Application

To start the application, type the name of the executable file at the UNIX command prompt. The executable filename is $A3DT/bin/$ARCH/acis3dt, where $A3DT is an environment variable that is defined as the name of your root installation directory and $ARCH is an environment variable that is defined as the name of your platform directory (e.g., hp700, sgi, solaris, etc.). Refer to the *Installation Guide* for specific platform names.

Windows Platform

Topic: *Scheme AIDE Application

To start the application on a Windows platform, locate the executable file in your installed directory and double click on the filename. The executable filename is $A3DT\bin\$ARCH\acis3dt.exe, where $A3DT is an environment variable that is defined as the name of your root installation directory and $ARCH is an environment variable that is defined as the name of your platform directory (NT).

Macintosh PowerPC Platform

Topic: *Scheme AIDE Application

The executable filename is :<*install_dir*>:Spatial:lib#.#:ACIS3DT, where <*install_dir*> is your root installation directory and #.# is the current release number. To start the application, double click on the icon labeled ACIS3DT in the lib#.# folder that is within the Spatial folder.

Initialization File

Topic: *Scheme AIDE Application

When the Scheme AIDE application starts, it automatically looks for an initialization file of Scheme procedures called acisinit.scm. Scheme AIDE searches your current working directory first, then your home directory, then the directories specified by the environment variable LOADPATH. If Scheme AIDE finds this file, it loads the file and does not search the remaining directories. This file can be used to execute commands to initialize your session, such as setting pathnames or loading commonly used Scheme procedures into memory.

An example acisinit.scm file is in your installation set. This file is located in the directory <*install_dir*>/scm/examples (the examples directory), along with other sample Scheme procedure files. To use this sample initialization file, copy it to your home directory, or the working directory from which you will run Scheme AIDE. Since this sample file loads other Scheme procedure files from the examples directory, you should modify your copy of acisinit.scm to start with a command that sets your loadpath to include the examples directory.

The following command (for a UNIX platform) illustrates this:

```
(set! load-path (list "<install_dir>/scm/examples"))
```

You must substitute <*install_dir*> in the above example with the name of your root installation directory.

Note *If you use this sample initialization file, a view window will be created automatically when you start* Scheme AIDE.

Startup Options

Topic: *Scheme AIDE Application

You may add options following the name of the executable file when you start the Scheme AIDE application. Arguments to Scheme procedures may also be passed in with the startup command. The general syntax for starting the program with startup options or procedure arguments is:

```
acis3dt [option [option data]] ... -- [argument] [argument] ...
```

where the options are:

–d ..... Disables all windows. No command window is created, and no views are created and displayed. Input is from stdin and output is to stdout. Example:

```
acis3dt -d
```

–e ..... Exit after loading the file specified with the –l filename option. Example:

```
acis3dt -l views.scm -e
```

–g ..... Enables garbage collection debugging. Garbage collection maintains free vs. used memory. Example:

```
acis3dt -g
```

–h heapsize ..... Defines the heap size in KBytes. Reduce the heap size to use less memory. Example:

```
acis3dt -h 4000
```

–height number ..... Specifies height for the command window (NT only). Example:

```
acis3dt.exe -height 20
```

–i ..... Enables case-insensitive symbols. Normally Scheme symbols are case-sensitive. Example:

```
acis3dt -i
```

–icon ..... Causes the command window to start iconified (NT only). Example:

```
acis3dt.exe -icon
```

–l filename ..... Loads a Scheme procedure file. Example:

```
acis3dt -l views.scm
```

–noecho ..... Does not print to the command window. (NT only). Example:

```
acis3dt.exe -noecho
```

–out filename ..... Writes all the text (commands and responses) that are displayed in the command window to the specified file. Example:

```
acis3dt –out testout.txt
```

–p loadpath . . . . . Initializes the loadpath, which is a list of directories. The specified loadpath is where Scheme looks for files when the load or autoload commands are used. loadpath must be a single argument consisting of a list of pathnames. Pathnames are separated by a : (colon). The default pathname is .;../../scm/examples Example:

```
acis3dt –p ~/scm:~/myexamples
```

–q . . . . . Does not load the acisinit.scm initialization file. Example:

```
acis3dt –q
```

–s . . . . . Creates the acis3dt command window but does not use it. Information is read from stdin and stdout. Views are created and displayed. Example:

```
acis3dt –s
```

–v . . . . . Enables verbose mode (print-linker commands). Example:

```
acis3dt –v
```

–width number . . . . . Specifies width for the command window (NT only). Example:

```
acis3dt.exe –width 60
```

–x number . . . . . Specifies *x*-position for the command window (NT only). Example:

```
acis3dt.exe –x 0
```

–y number . . . . . Specifies *y*-position for the command window (NT only). Example:

```
acis3dt.exe –y 0
```

? . . . . . Displays a usage message. Example:

```
acis3dt ?
```

The – – ends the list of options and begins arguments to Scheme procedures. For example:

```
acis3dt –l views.scm –– arg1 arg2
```

When the Scheme Interpreter has been initialized, (command–line–args) returns a list of these command-line arguments for use within a Scheme procedure. For example:

```
(display (command-line-args)) ((view:dl))
```

Scheme Extension Syntax

Topic: *Scheme AIDE Application

The syntax to each Scheme extension is documented within their respective component books. However, the syntax to a Scheme extension can be obtained directly from the Scheme AIDE application if its name is known. This is accomplished using the syntax.scm Scheme procedure file located in the directory <*install_dir*>/scm/examples (the examples directory).

Append the following line to the acisinit.scm file if it doesn't have it already.

```
(load 'syntax "syntax.scm")
```

Start Scheme AIDE. When the syntax for a given command is needed, type:

```
(syntax "<extension_name>")
```

```
(syntax "solid:block")
```

This will return the syntax and argument type portions of the Scheme extension reference template.

Interaction

Topic: Scheme AIDE Application

This section describes how to interact with Scheme AIDE. Common interaction features, special keyboard characters, and mouse button functions are described, followed by some that are platform-specific (i.e., only for the UNIX version or the Windows version).

Common Interaction

Topic: *Scheme AIDE Application

When the command window appears, the user can create views and issue any Scheme command or procedure. The command window is used to input Scheme commands and display messages. Graphics are not displayed in the command window.

The command window has several built-in features that aid in entering Scheme commands:

Prompt . . . . . The acis> prompt indicates that the program is ready for typed input.

Caret . . . . . The ^ (caret) symbol is a text cursor that indicates the current typing position.

Caret bounce . . . . . As each closing parenthesis ")" of a Scheme command is entered, the caret "bounces" momentarily back to the matching open parentheses "(" to aid in parentheses matching.

Copy and paste . . . . . Use the mouse to highlight a block of text (copy) and drop it at the prompt (paste).

Interpreter response . . . . . The Elk Scheme interpreter often responds to a procedure by returning the external representation (name) of an object. For example, when the user defines the block myblock using the command solid:block, the interpreter responds on the next line with the name of the object, myblock:

```
acis> (define myblock (lambda ()
   (solid:block (position 0 0 0)
   (position 10 10 10))))
myblock
acis>
```

Parentheses nesting level . . . A Scheme procedure may be entered on one or more lines. The interpreter continues building the procedure until all open parentheses have been matched with close parentheses. As it builds the procedure, a number next to the prompt indicates the number of close parentheses still required (i.e., the nesting level). For example, the definition of block myblock using command solid:block could be spread over several input lines:

```
acis> (define myblock
acis(1)> (lambda ()
acis(2)> (solid:block
acis(3)> (position 0 0 0)
acis(3)> (position 10 10 10)
acis(3)> )
acis(2)> )
acis(1)> )
myblock
acis>
```

Object type . . . . . If the name of some Scheme object is entered without surrounding parentheses, the Scheme Interpreter responds with the external representation (type) of that object. For example:

```
acis> list
#[primitive list]
```

A list object is a Scheme primitive (native to the Scheme language).

```
acis> solid:block
#[primitive solid:block]
```

A solid:block is a Scheme primitive (an extension to the Scheme language).

```
acis> myblock
#[compound myblock]
```

A myblock object is a compound object made up of other Scheme objects and is not a primitive.

Error messages . . . . . If a Scheme procedure fails, the Scheme Interpreter returns an error message. For example, if a command is mistyped the interpreter indicates that the command is unbound (i.e., it has no associated value):

```
acis> (solid:block (position 0 0 0)
   (position 10 10 10))
top-level: unbound variable: solid:block
```

The following special keyboard characters and mouse buttons are used:

<Left Arrow> . . . . . Backs up to new edit position in input buffer.

<Right Arrow> . . . . . Advances to new edit position in input buffer.

<Up Arrow> . . . . . Walks back through command history.

<Down Arrow> . . . . . Walks forward through command history.

<PgUp> . . . . . Scrolls window up.

<PgDn> . . . . . Scrolls window down.

<Escape> . . . . . Clears the line from the input buffer.

<Delete> . . . . . Deletes the character ahead of the cursor from the input buffer.

<Backspace> . . . . . Deletes the character behind the cursor from the input buffer.

<Insert> . . . . . Toggles insert mode for the input buffer.

<Home> . . . . . Goes to the beginning of line in the input buffer.

<End> . . . . . Goes to the end of line in the input buffer.

<Ctrl + A> . . . . . Goes to the beginning of line in the input buffer.

<Ctrl + E> . . . . . Goes to the end of line in the input buffer.

< \ > + < r > . . . . . Returns with a window scroll, if necessary.

< \ > + < n > . . . . . Provides a new line with a window scroll, if necessary.

< \ > + < b > . . . . . Backspaces one character.

< \ > + < t > . . . . . Tabs to the next multiple of eight.

<Left Mouse> . . . . . Selects text. Hold down, drag to select text, and release. This block of text becomes the primary text selection.

UNIX Platform Interaction

Topic: *Scheme AIDE Application

These built-in command window features are specific to the version for UNIX platforms:

Copy and paste . . . . . Use the mouse to highlight a block of text (copy) and drop it at the prompt (paste).

These special keyboard characters and mouse buttons are specific to the version for UNIX:

<Ctrl + C> . . . . . Aborts the current running command.

<Ctrl + X> . . . . . Interrupts rendering (in a viewport).

<Middle Mouse> . . . . . Clicks to paste selected text. Text is pasted into the command window or into any other window that supports the X Windows selection mechanism.

<Shift> + <Left Mouse> . . . . Changes the end of the selected area.

Windows Platform Interaction

Topic: *Scheme AIDE Application

These built-in command window features are specific to the version for Windows platforms:

Copy and paste . . . . . Use the mouse to highlight a block of text, <Ctrl + Insert> to copy the text, and <Shift + Insert> to paste the text at a new location.

These special keyboard characters are specific to the version for Windows:

<Ctrl + Insert> . . . . . Copies text.

<Shift + Insert> . . . . . Pastes text.

<Shift + Delete> . . . . . Cuts text.

<Ctrl + Delete> . . . . . Deletes text.

<Alt + Backspace> . . . . . Undoes an operation.

Simple Scheme Examples

Topic: Scheme AIDE Application

The following are simple examples using the Scheme AIDE application. Type the command lines, exactly as they appear in these examples, following your Scheme AIDE command prompt. The lines in the examples beginning with semicolons (;) are comment lines that you do not need to type. Lines beginning with two semicolons (;;) indicate responses from the application and are not to be typed. Some examples build on previous ones.

Command Input File

Topic: *Scheme AIDE Application

Scheme AIDE can execute commands from a Scheme procedure input file, as illustrated with the initialization file acisinit.scm. A Scheme procedure file can contain commands that create objects, perform actions, set options, etc. An input file is a text file containing Scheme commands and, optionally, comments. Refer to the *Scheme Support Component Manual* for more information on creating Scheme procedure files.

The Scheme command load is used to load and execute an input Scheme procedure file. The input file must exist in your working directory (or in your load path), or you must include the path when you specify the filename. To run Example 6-2, use a text editor to create a text file in your working directory named test.scm containing the text in Example 6-1. Figure 6-1 shows the result of loading this file.

```
; Sample input file test.scm
(define b1 (solid:block (position –20 –20 0) (position 20 20 5)))
;; b1
(define s1 (solid:sphere (position 0 0 0) 15))
;; s1
(define new (bool:intersect b1 s1))
;; new
```

Example 6-1. Sample Command Input File

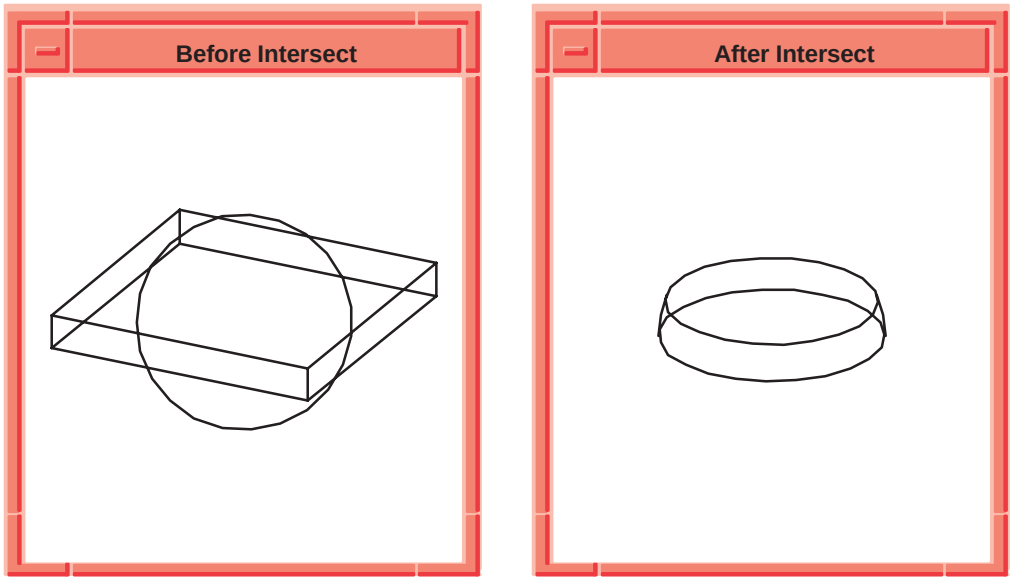


Figure 6-1. Command Input File

```
; Delete all entities from the view
(part:clear)
;; #t
; Load the Scheme input file test.scm
(load ”test.scm”)
;; ()
```

Example 6-2. Executing the Command Input File

Unite a Block and a Cylinder

Topic: Scheme AIDE Application

Example 6-3 creates a block and a cylinder, then unites them. Figure 6-2 shows the model before and after the entities are united.

```
; Create a solid block
(define b1 (solid:block (position –20 –20 –20)
    (position 20 20 20)))
;; b1
; Create a cylinder
(define c1 (solid:cylinder (position 20 0 –20)
    (position 20 0 20) 20)))
;; c1
; Unite the two bodies into a new body
(define u1 (bool:unite b1 c1))
;; u1
```

Example 6-3. Unite Block and Cylinder

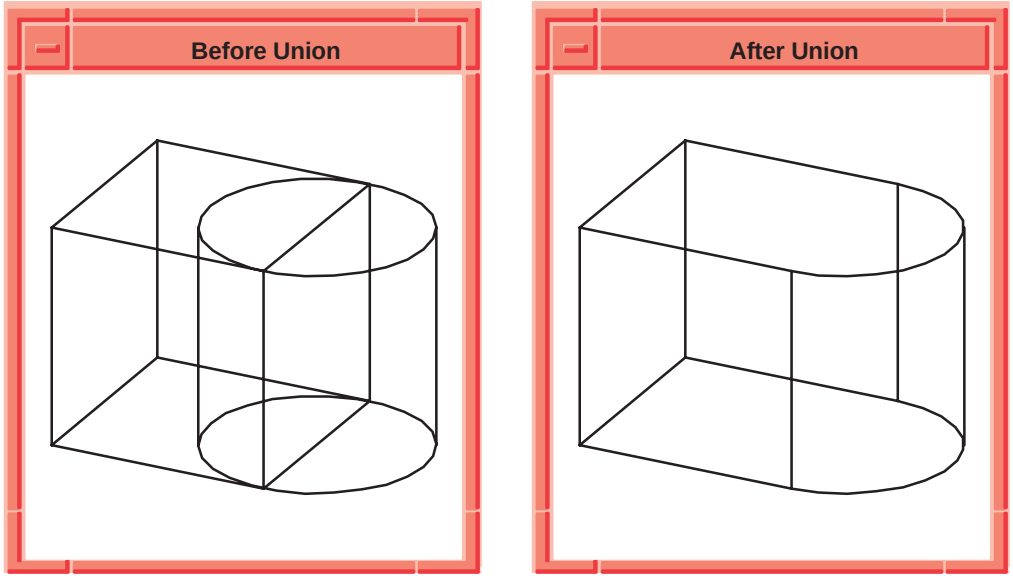


Figure 6-2. Unite Block and Cylinder

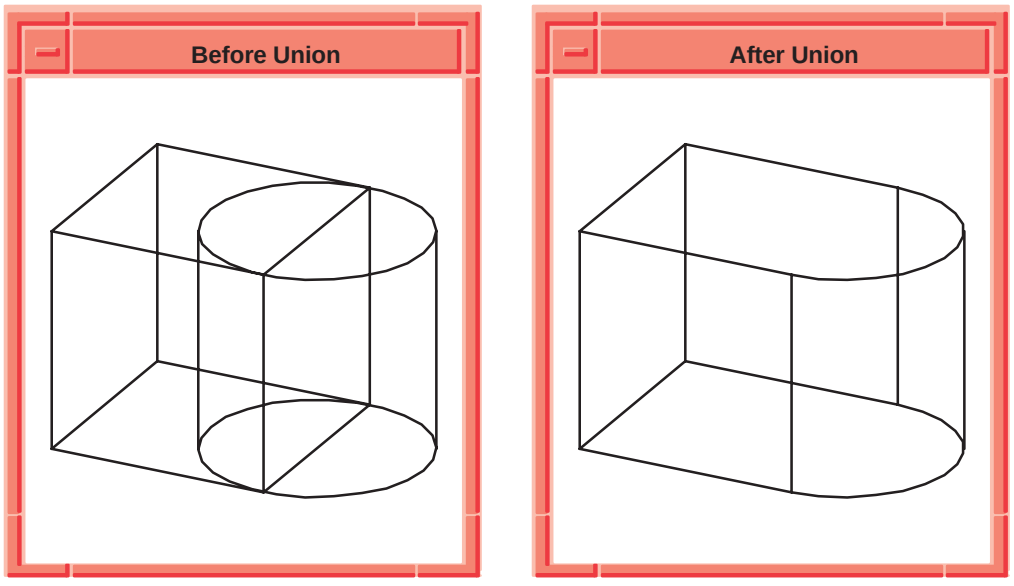
Rotate Body

Topic: Scheme AIDE Application

Example 6-4 rotates the body u1 (after the union) from Example 6-3 so you can view it from a different angle. A transform is first defined, then applied to the body. Figure 6-3 shows the rotated body.

```
; (part:clear)
; Create a solid block
(define b1 (solid:block (position –20 –20 –20)
    (position 20 20 20)))
;; b1
; Create a cylinder
(define c1 (solid:cylinder (position 20 0 –20)
    (position 20 0 20) 20)))
;; c1
; OUTPUT Before Union
; Unite the two bodies into a new body
(define u1 (bool:unite b1 c1))
;; u1
; Define a transform, t1, to rotate an object about the z–axis by
; 90 degrees
(define t1 (transform:rotation (position 0 0 0 )
    (gvector 0 0 1) 90))
;; t1
; OUTPUT After Union
; Apply the transform t1 to the body u1
(entity:transform u1 t1)
;; #[entity 2 1]
; OUTPUT Rotate/Transform
```

Example 6-4. Rotate Body



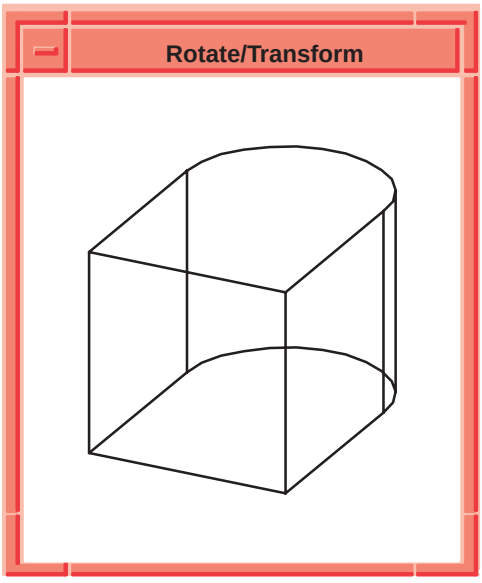


Figure 6-3. Rotate Body

Save and Load

Topic: Scheme AIDE Application

ACIS models can be saved to a part save file for later retrieval. They can be saved in text mode (file extension .sat) or binary mode (file extension .sab). Refer to the *Kernel Component Manual* for information about save files.

Example 6-5 saves the body `u1` created in Example 6-4 into a part save file in text format and then retrieves it from that file as a body with a new name.

```
; Save the model in text mode to file test.sat
(part:save ”test” #t)
;; #t
; Delete all the entities
(part:clear)
;; #t
; Nothing in view window now
; Retrieve, or load, the model from the text mode file test.sat
(part:load ”test”)
;; (#[entity 4 1])
```

Example 6-5. Save and Load