Chapter 1. Scheme Extenstions Aa thru Fz

Topic: Ignore

Scheme is a public domain programming language, based on the LISP language, that uses an interpreter to run commands. ACIS provides extensions (written in C++) to the native Scheme language that can be used by an application to interact with ACIS through its Scheme Interpreter. The C++ source files for ACIS Scheme extensions are provided with the product. *Spatial*'s Scheme based demonstration application, Scheme ACIS Interface Driver Extension (Scheme AIDE), also uses these Scheme extensions and the Scheme Interpreter. Refer to the *3D ACIS Online Help User's Guide* for a description of the fields in the reference template.

afig:apply–transform

Scheme Extension: Animation, Transforms

Action: Applies a transform to an animation–figure.

Filename: gi/gi_scm/afig_scm.cxx

APIs: None

Syntax: (**afig:apply–transform** fig tform)

Arg Types:

fig	animation–figure
tform	transform

Returns: unspecified

Errors: None

Description: Applies a transform to an animation figure. This procedure concatenates the given transform with the current transform of the figure. For example, if you call afig:apply–transform 10 times with a transform which rotates by 5 degrees, the net effect will be that the figure will have been rotated by a total of 50 degrees. By contrast, if you call afig:set–transform 10 times with the same transform, the figure will only be rotated by 5 degrees.

Limitations: None

Example:
```
; afig:apply–transform
; Create a figure to rotate.
(define block1
    (solid:block (position 0 0 0)
    (position 10 15 20)))
;; block1
(define cyl1
    (solid:cylinder (position 4 8 12)
    (position 8 –10 –15) 5))
;; cyl1
; Apply a transform to an animation figure.
(define fig (afig:create (list block1 cyl1)))
;; fig
(afig:show fig)
;; ()
; OUTPUT Original

(define t
    (transform:rotation (position 0 0 0)
    (gvector 0 1 0) 1))
;; t
; Rotate 200 increments.
(let loop ((i 0))
    (if (< i 200) (begin
        (afig:apply–transform fig t)
        (loop (+ i 1))
    )))
;; ()
; Displays the block and cylinder actively
; rotating 200 degrees before stopping.
; OUTPUT Result
```

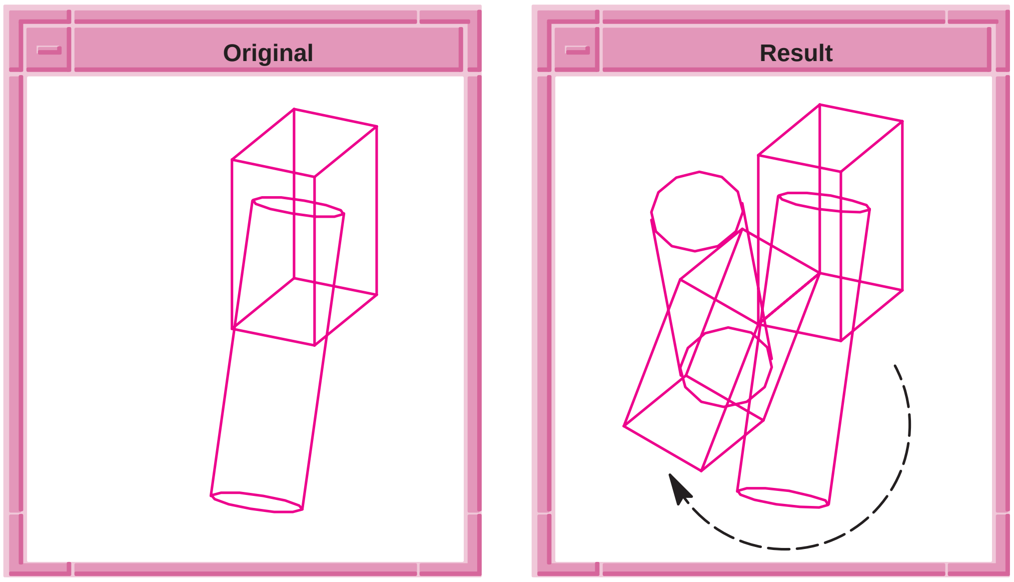

Figure 1-1. afig:apply–transform

afig:create

Scheme Extension:	Animation, Rubberbanding
Action:	Creates an animation–figure.
Filename:	gi/gi_scm/afig_scm.cxx
APIs:	None
Syntax:	`(`**`afig:create`**` entity–lits)`
Arg Types:	entity–list entity | (entity ...)
Returns:	animation–figure
Errors:	None
Description:	Creates an animation figure from a list of entities. An animation figure is a type of rubberband driver.
Limitations:	None

Example:

```
; afig:create
; Create a figure to rotate
(define block1
    (solid:block (position 0 0 0)
    (position 10 15 20)))
;; block1
(define cyl1
    (solid:cylinder (position 4 8 12)
    (position 8 –10 –15) 5))
;; cyl1
; Define the animation figure.
(define fig (afig:create (list block1 cyl1)))
;; fig
;  Highlight the animated figure.
(afig:show fig)
;; ()
; Define the rotation of the transform.
(define t
    (transform:rotation (position 0 0 0)
    (gvector 0 1 0) 1))
;; t
; OUTPUT Original

; Define the rotation as 200 degrees and apply the
;   transform to the animation figure.
(let loop ((i 0))
    (if (< i 200) (begin
        (afig:apply–transform fig t)
        (loop (+ i 1))
    )))
;; ()
; Displays the block and cylinder rotating
; 200 degrees.
; OUTPUT Result
```

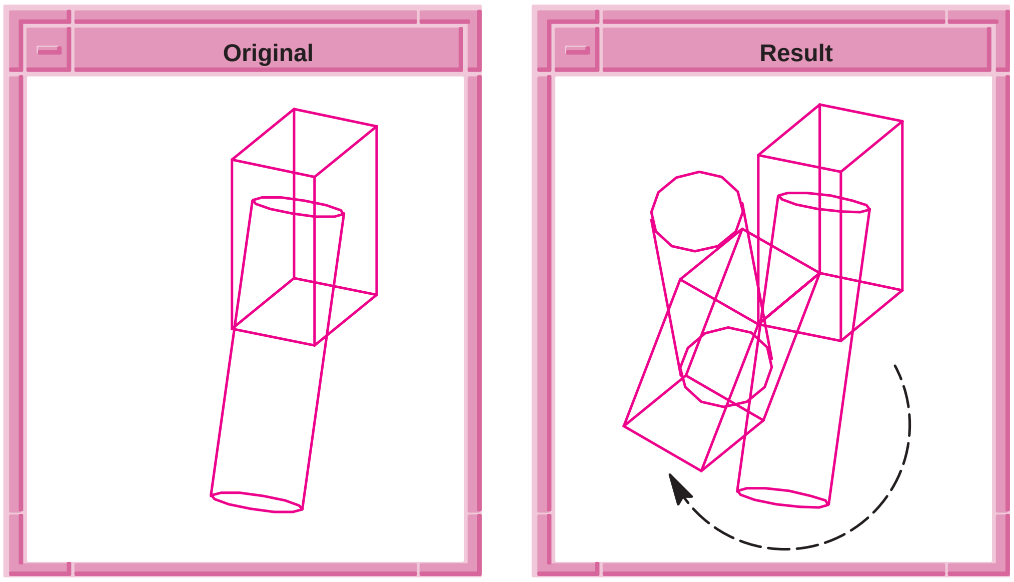

Figure 1-2. afig:create

afig:get–transform

Scheme Extension: Animation, Transforms

Action:	Gets the current transformation for an animation–figure.
Filename:	gi/gi_scm/afig_scm.cxx
APIs:	None
Syntax:	`(afig:get–transform fig)`
Arg Types:	fig animation–figure
Returns:	transform
Errors:	None
Description:	When afig:apply–transform has been applied a number of times, you can get the current composite transform using afig:get–transform.
Limitations:	None

Example:

```
; afig:get–transform
; Create a figure to rotate.
(define block1
    (solid:block (position 0 0 0)
    (position 10 15 20)))
;; block1
(define cyl1
    (solid:cylinder (position 4 8 12)
    (position 8 –10 –15) 5))
;; cyl1
; Apply a transform to an animation figure.
(define fig (afig:create (list block1 cyl1)))
;; fig
(afig:show fig)
;; ()
(define t
    (transform:rotation (position 0 0 0)
    (gvector 0 1 0) 1))
;; t
(let loop ((i 0))
    (if (< i 90) (begin
        (afig:apply–transform fig t)
        (loop (+ i 1))
    )))
;; ()
; Displays the block and cylinder
; rotating 360 degrees.
(define t2 (afig:get–transform fig))
;; t2
```

afig:set–transform

Scheme Extension: Animation, Transforms

Action: Sets the transform for an animation–figure.

Filename: gi/gi_scm/afig_scm.cxx

APIs: None

Syntax: (**afig:set–transform** fig tform)

Arg Types:

fig	animation–figure
tform	transform

Returns: animation–figure

Errors: None

Description: Sets the transform of an animation figure. The difference between this procedure and afig:apply–transform is that the transform passed to this procedure acts as an absolute transform, whereas the transform passed to afig:apply–transform is a relative transform which is concatenated with the current transform for the figure.

For example, if you call afig:apply–transform 10 times with a transform which rotates by 5 degrees, the net effect is the figure is rotated a total of 50 degrees. By contrast, if you call afig:set–transform 10 times with the same transform, the figure is only rotated a total of 5 degrees.

Limitations: None

Example:

```
; afig:set–transform
; Create a figure to rotate.
(define block1
    (solid:block (position 0 0 0)
    (position 10 15 20)))
;; block1
(define cyl1
    (solid:cylinder (position 4 8 12)
    (position 8 –10 –15) 5))
;; cyl1
; Apply a transform to an animation figure.
(define fig (afig:create (list block1 cyl1)))
;; fig
(afig:show fig)
;; ()
(define t
    (transform:rotation (position 0 0 0)
    (gvector 0 1 0) 1))
;; t
(afig:set–transform fig t)
;; ()
```

afig:show

Scheme Extension: Animation

Action: Shows or hides an animation–figure.

Filename: gi/gi_scm/afig_scm.cxx

APIs: None

Syntax:	(**afig:show** fig [off] [view=active])	
Arg Types:	fig	animation–figure
	off	boolean
	view	view
Returns:	unspecified	
Errors:	None	

Description: Displays or hides an animation figure. To hide an animation figure, call this procedure with #f for the second argument.

The optional third argument lets you specify a view to display the animation figure in. If it is not given, the figure is displayed in the active view.

Limitations: None

Example:

```
; afig:show
; Create a figure to rotate.
(define block1
    (solid:block (position 0 0 0)
    (position 10 15 20)))
;; block1
(define cyl1
    (solid:cylinder (position 4 8 12)
    (position 8 –10 –15) 5))
;; cyl1
; Apply a transform to an animation figure.
(define fig (afig:create (list block1 cyl1)))
;; fig
(afig:show fig)
;; ()
(define t
    (transform:rotation (position 0 0 0)
    (gvector 0 1 0) 1))
;; t
; Rotate 200 increments.
(let loop ((i 0))
    (if (< i 200) (begin
        (afig:apply–transform fig t)
        (loop (+ i 1))
    )))
;; ()
; Displays the 2nd block and cylinder actively
; rotating 200 degrees before stopping.
; OUTPUT Original
; Turn off afig:show, animation figure disappears.
(afig:show fig #f)
;; ()
; Animation figure is no longer displayed.
; OUTPUT Result

; Reactivate the display/show feature.
(afig:show fig)
;; ()
; Animation figure is displayed again.
```

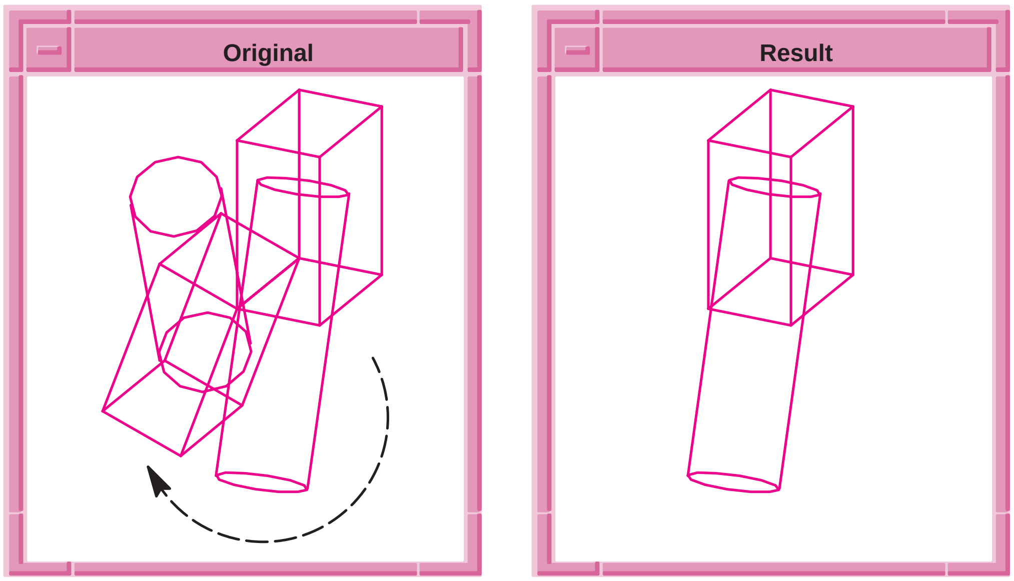

Figure 1-3. afig:show

```
; Request the color of poly1 and poly2.
(dl-item:color poly1)
;; #[color 1 0 0]
; color 1 0 0 is red.
(dl-item:color poly2)
;; #[color 0 0 1]
; color 0 0 1 is blue.
```

dl:dynamic–silhouette–display

Scheme Extension: Viewing, Silhouette and Isoparametric Curves

Action: Sets the dynamic silhouette display of solids.

Filename:	gi/gi_scm/dl_scm.cxx	
APIs:	None	
Syntax:	(**dl:dynamic–silhouette–display** on–off)	
Arg Types:	on–off	boolean
Returns:	boolean	
Errors:	None	

Description: on recomputes the silhouette edges of solids automatically when the view orientation changes. off does not recompute the silhouette edges of solids automatically when the view orientation changes. The default is on. For faster refresh speed in dynamic rotation, turn this extension off.

The value of the previous state is returned. The Scheme file rotsph.scm has a more detailed example. A display item is not part of the model and does not get saved and restored.

Limitations: None

Example:
```
; dl:dynamic–silhouette–display
; Turn on the automatic recomputation
; of display silhouettes of solids.
(dl:dynamic–silhouette–display #t)
;; #t
; Turn off the automatic recomputation
; of display silhouettes of solids.
(dl:dynamic–silhouette–display #f)
;; #t
```

dl:interleaf

Scheme Extension: Viewing, Image Output

Action:	Writes a view to a file in Interleaf format.	
Filename:	gi/gi_scm/dl_scm.cxx	
APIs:	None	
Syntax:	(**dl:interleaf** [filename=plotfile.il] [color=#f] [x–size y–size] [view=active])	
Arg Types:	filename	string
	color	boolean
	x–size	real
	y–size	real
	view	view

Returns: string

Errors: None

Description: The optional filename specifies where to send the image. If the filename is not specified, the image is sent to the default filename, plotfile.il. The optional color specifies a logical (TRUE or FALSE); the default is FALSE. The optional x–size defines the *x*–value in pixel dots per inch. The optional y–size defines the *y*–value in pixel dots per inch. The optional view defines the view that saves the image file; the default is the active view.

This extension returns the filename in a string. dl:interleaf sends the wireframe image to the output file, while render:interleaf renders the image sent to the output file. A display item is not part of the model and does not get saved and restored.

Limitations: None

Example:

```
; dl:interleaf
; Write a view to a default file in Interleaf format.
(dl:interleaf)
;; ”plotfile.il”
; Write a view to the test.il file with a specified
; x–size and y–size in Interleaf format.
(dl:interleaf ”test.il” 1500 1500)
;; ”test.il”

; Another example where something is shown.
; Define a new view.
(define view1 (view:dl))
;; view1
; Create a solid block.
(define block1
    (solid:block (position 0 0 0)
    (position 35 35 35)))
;; block1
; Create an Interleaf view 1” by 1”
(define ileaf_view
    (dl:interleaf ”block.il” 25.8 25.8 view1))
;; ileaf_view
; 1” at 300 dpi is 1200
; Another example where the size of the view
; and viewport are changed before an output image
; is made.
; Delete the view from the previous example
(view:delete view1)
;; ()
; Define a new view.
(define view2 (view:dl 100 200))
;; view2
; This queries the current view size. Then
; it resizes the viewport. Then it makes the
; object appear smaller by changing the width
; and height parameter of the view.
(let ((w (view:width view2)))
    (view:set–viewport 0 0 300 300 view2)
    (view:set–size (* w 2) (* w 2) view2))
;; #[view 109820984]
; Refresh the view to see the changes
(view:refresh view2)
;; #[view 109829897]
; Create an Interleaf view 1” by 1”
(define ileaf_view
    (dl:interleaf ”block.il” 25.8 25.8 view2))
;; ileaf_view
```

dl:metafile

Scheme Extension: Viewing, Image Output

Action: Writes a view to a file in Windows Enhanced Metafile format.

Filename: gi/gi_scm/dl_scm.cxx

APIs: None

Syntax:
```
(dl:metafile [filename] [view=active] [emf–wmf]
    [header=#f])
```

Arg Types:

filename	string
view	view
emf–wmf	boolean
header	boolean

Returns: view

Errors: None

Description: When running on the Windows NT platform, this extension writes a view in Windows Enhanced Metafile format to a file or to the clipboard and returns the specified view. dl:metafile sends the wireframe image to the output file, while render:metafile renders the image sent to the output file.

The third argument controls whether an old style metafile (wmf) is made or the new enhanced format (emf) is made. If an old style is specified (#t), the fourth argument is used to add some extra header information that is required for some Windows programs such as Word. Metafile type defaults to #t which is an enhanced metafile. Header type defaults to #f which does not add the extra header.

Limitations: This extension works on the MS Windows platform only; on other platforms the extension returns “Metafile output not supported for this platform.”

Example:

```
; dl:metafile
; create a new view.
(define view1 (view:dl))
;; view1
(view:set-title ”view1” view1)
;; #[view 1076886864]
(define view2 (view:dl))
;; view2
(view:set-title ”view2” view2)
;; #[view 1076892048]
(define body (solid:subtract
    (solid:cylinder (position 0 0 -10)
    (position 0 0 10) 10)
    (solid:sphere (position 0 0 20) 20)
    (solid:cylinder (position 10 0 -10)
    (position 10 0 10) 5)
    (solid:cylinder (position -10 0 0)
    (position 10 0 0) 3)
    (solid:cylinder (position 0 -10 0)
    (position 0 10 0) 3)))
;; body
; These are the metafile test cases.
; The following generates an enhanced metafile
(dl:metafile ”c:/tmp/dltest1.emf”)
;; #[view 1076886864]
; The following generates an enhanced metafile
; for view2
(dl:metafile ”c:/tmp/dltest2.emf” view2)
;; #[view 1076892048]
```

```
; The following generates an enhanced metafile
; as specified from arg 3
(dl:metafile ”c:/tmp/dltest3.emf” #t)
;; #[view 1076886864]
; The following generates an enhanced metafile
; for view2. arg 4 says enhanced metafile
(dl:metafile ”c:/tmp/dltest4.emf” view2 #t)
;; #[view 1076892048]
; The following generates an old metafile format
; without the extra header information
(dl:metafile ”c:/tmp/dltest5.wmf” #f)
;; #[view 1076892048]
; The following generates an old metafile format for
; view2 without the extra header information
(dl:metafile ”c:/tmp/dltest6.wmf” view2 #f)
;; #[view 1076892048]
; The following generates an old metafile format
; with the extra header information
(dl:metafile ”c:/tmp/dltest7.wmf” #f #t)
;; #[view 1076892048]
; The following generates an old metafile format
; for view2 with the extra header information
(dl:metafile ”c:/tmp/dltest8.wmf” view2 #f #t)
;; #[view 1076892048]
; The following generates an enhanced metafile that
; is sent to the clipboard (suggested format for
; clipboard transfers)
(dl:metafile)
;; #[view 1076892048]
```

dl:postscript

Scheme Extension: Viewing, Image Output

Action: Writes a view to a file in PostScript format.

Filename: gi/gi_scm/dl_scm.cxx

APIs: None

Syntax:
```
(dl:postscript [filename=plotfile.ps] [color=#f]
    [x–size y–size] [view=active])
```

Arg Types:	filename	string
	color	boolean
	x–size	real
	y–size	real
	view	view

Returns: string

Errors: None

Description: The optional filename specifies where to send the image. If filename is not specified, the image is sent to the default filename, plotfile.ps. The optional color specifies a logical (TRUE or FALSE); the default is FALSE. The optional x–size defines the *x*–value in pixel dots per inch. The optional x–size and y–size specify the dimensions in millimeters of a rectangle into which you want the printed image of the viewport to fit as tightly as possible without distortion. The optional view defines the view to save the image file; the default is the active view.

This extension returns the filename in a string. dl:postscript sends the wireframe image to the output file, while render:postscript renders the image sent to the output file. A display item is not part of the model and does not get saved and restored.

Limitations: None

Example:

```
; dl:postscript
; Write a view to the default file
; in PostScript format.
(dl:postscript)
;; ”plotfile.ps”
; Write a view to the test.ps file with a specified
; x–size and y–size in PostScript format.
(dl:postscript ”test.ps” 1500 1500)
;; ”test.ps”
```

dl:print

Scheme Extension: Viewing, Image Output

Action: Writes a view to Windows.

Filename: gi/gi_scm/dl_scm.cxx

APIs: None

Syntax:

```
(dl:print [x–resolution=300 y–resolution=300]
    [view=active])
```

Arg Types:	x–resolution	integer
	y–resolution	integer
	view	view

Returns: view

Errors: None

Description: The optional x–resolution specifies the *x*–value in pixel dots per inch; the default is 300 dots per inch. The optional y–resolution specifies the *y*–value in pixel dots per inch; the default is 300 dots per inch. The optional view specifies the view orientation at the specified x–resolution and y–resolution to be printed on a laser printer; the default is the active view.

For example, if the output laser printer has a resolution of 300 dots per inch, and the x–resolution and y–resolution values are both 300, the image output to the laser printer is one square inch. A Print box displays requesting information for the desired printer and output port or the filename if the image is to be output to a file. If the image is output to a file, a dialog box will pop up asking for the name of the file.

Change the printer setup information by selecting the Print Setup button. Modify page orientation and paper size to accommodate specific requirements. The image is sent to the specified printer and an image file is created and stored in the current directory; the default is PostScript. This extension also returns the view where the file was saved.

Note *This extension requires Windows and a Personal Computer (PC). If the PC is not attached to a printer, a Print To File window automatically displays to specify an output filename.* dl:print *sends the wireframe image to the output file, while* render:print *renders the image sent to the output file.*

Limitations: This extension works on the MS Windows or Windows NT platforms only; on other platforms the extension returns “Printing is not supported on this platform.”

Example:

```
; dl:print
; Print a view in Windows.
(define ISO (view:dl))
;; ISO
(dl:print 1500 1500 ISO)
;; #[view 1075840784]
```

env:views

Scheme Extension: Viewing, Scheme Interface

Action: Gets a list of all defined views.

Filename: gi/gi_scm/view_scm_gi.cxx

APIs: None

Syntax: **(env:views)**

Arg Types: None

Returns: (view ...)

Errors: None

Description: Refer to Action.

Limitations: None

Example:

```
; env:views
; Create a new view.
(define view (view:dl))
;; view
(define view2 (view:dl))
;; view2
view
;; #[view 1076066832]
view2
;; #[view 1076071408]
; Get a list of all views.
(define viewlist (env:views))
;; viewlist
; (#[view 1076016872] #[view 1073761464]
; #[view 1076066832] #[view 1076071408])
; This assumes the two views created in this example
; and two views created by manual.scm. One of the
; views from manual.scm is an interleaf view that
; you don’t see.
```