# Specification of ASPIC

by

Philip Hazel

Second edition
June 1994



Computer Laboratory
New Museums Site
Pembroke Street
Cambridge CB2 3QG
United Kingdom

*Telephone:* +44 1223 334600

# 1. Introduction

ASPIC is a program which converts a textual description of a line drawing into instructions for a typesetting program. It has been developed for use with the SGCAL typesetter, but in principle can be used with other systems by changing its configuration file, which is know as its 'steering' file.

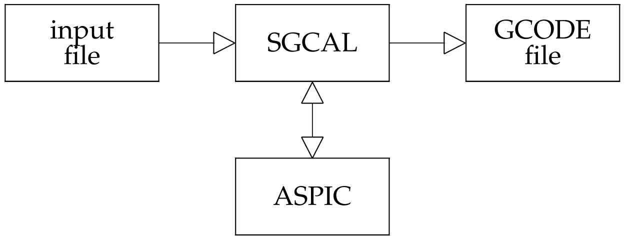
The format of the steering file is described in chapter xxx. Its location can be specified explicitly by a system variable, but if this variable is unset, ASPIC looks for a file called **AspicSteer** in the SGCAL library.

This method of drawing pictures is the same as that of the PIC system, described by Kernighan in *Software – Practice and Experience*, **12**, pp 1–21 (1982), though the details of the ASPIC commands are quite different.

The SGCAL program has a facility for calling a secondary program to process embedded input and return it as SGCAL input. This is normally used to call ASPIC. Therefore, the syntax of the command to run ASPIC is simply

`aspic` *input-file output-file*

It is not, however, normally run directly in this way, but is usually run via SGCAL's **call** directive. As a simple example, consider the following picture:




This was created by the following SGCAL input lines:



```
.display rm
.call aspic -sgcal -nv
centre ~~sys.linelength;
magnify 0.8;
hlinelength 36;
box "input" "file"; arrow;
A: box "SGCAL"; arrow;
box "GCODE" "file";
arrow both down from A; box "ASPIC";
.endcall
.endd
```

The lines between **call** and **endcall** are written by SGCAL to a temporary file; ASPIC is then called to process this file. It writes its output to a second temporary file which is then processed by SGCAL. The output from ASPIC consists of SGCAL instructions to draw lines and curves, and to position text at given positions.

The size of the picture is computed by ASPIC and passed back to SGCAL so that it can leave enough vertical space for it. It can be requested, as in this example, to centre the picture in a given line width. If an SGCAL indent is in force when ASPIC is called, it will be applied to the picture.

The lines between **call** and **endcall** are subject to SGCAL's normal processing for insertions, and so can be passed data from SGCAL variables. In the example above, the SGCAL system variable **sys.linelength** is used to pass the current line length.

Also, any SGCAL macros that are encountered are expanded. This can be useful for repeated sections of input, though ASPIC has its own macro facility as well, but it is particularly useful for encapsulating directives at the start and end of a picture. For example, if a document is to be processed for both PostScript and plain ASCII output, pictures must be omitted in the second case. The conditional directives to achieve this can be bundled up inside SGCAL macros, as follows:

```
.macro aspic
.display rm
.if ~~sys.fancy
.call aspic -sgcal -nv
centre ~~sys.linelength;
.endm

.macro endspic
.endcall
.else
<<picture omitted>>
.fi
.endd
.endm
```

and then the description of each picture can simply be enclosed between **aspic** and **endspic** directives. Any ASPIC commands that are common to all of them can be included within the macros, as the **centre** command has been above.

ASPIC should be called from an SGCAL enviroment where line filling is disabled; it will not work properly otherwise. To prevent SGCAL from splitting the picture at the bottom of a page, it should also be called from within a contiguous section. The most straightforward way of achieving both these requirements is always to call ASPIC from within an SGCAL **display**.

ASPIC assumes that an SGCAL standard style is in force, as it generates output using the standard SGCAL flags for moving the current point, positioning text, and drawing rules and curves. The recognition of flags must not be disabled when SGCAL is processing ASPIC output. In particular, if the picture is inside an SGCAL **display**, the **asis** parameter must not be set.

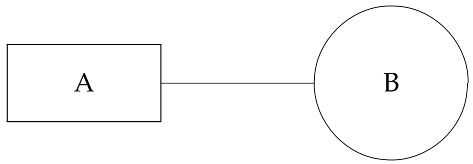


# 2. Simple examples

This chapter uses some simple examples to introduce the various facilities which are available in ASPIC. Subsequent chapters contain reference material which explains things in more detail.

Consider the following picture:

```
box "A"; line; circle "B";
```




Each ASPIC command is terminated by a semicolon, so there are three commands in this example.

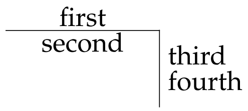
The **box** command causes a box to be drawn of a standard size, containing the text at its centre. The default standard size is 72 by 36 points (that is, 1" × 0.5"; there are commands to change this. There is a **magnify** command which affects the sizes of all shapes. In this document, all the pictures are in fact reduced by a factor of 0.8.

The **line** command draws a straight line; as nothing else is specified, the line is drawn in the current direction of motion, which defaults to the right. Its length is the standard horizontal line length, which defaults to 72 points (1").

The **circle** command draws a circle; as nothing was specified as to how it should join onto its predecessor, the 'obvious' joining position is chosen. The circle is drawn at a standard size. ASPIC provides for ellipses as well as circles.

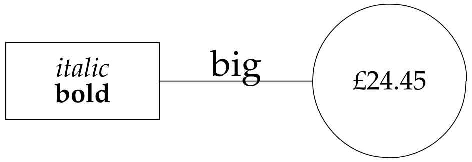
Each ASPIC command which causes a shape to be drawn may have any number of text strings associated with it. In the above example, the box and the circle each have one associated string. For closed shapes such as these, the strings are centred in the shape. For horizontal lines, the strings are centred above and below the line, while for other kinds of line they are positioned near the centre. For example,

```
line "first" "second"; line down "third" "fourth";
```



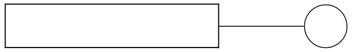

The strings are not processed by ASPIC, but are passed back in the output to be processed by SGCAL. Therefore, SGCAL flags are recognised within the strings, and in particular, the font-changing flags can be used:

```
box "$it{italic}" "$bf{bold}";
line "$chead{big}";
circle "($)24.45";
```




The commands introduced above may be used with options which change the size of the shape which is drawn. For example,

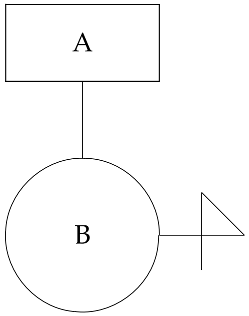
```
box width 100 depth 20; line right 40;
circle radius 10;
```



The units of length used in ASPIC commands are printer's points. There are 72 points to an inch. The current direction of motion can be changed by the commands **up**, **down**, **left**, and **right**. In addition, an individual line may be drawn in any direction and of any length by means of appropriate options.

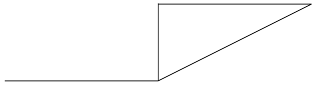
```
down; box "A"; line; circle "B";
line right 40; line left 20 up 20; line;
```





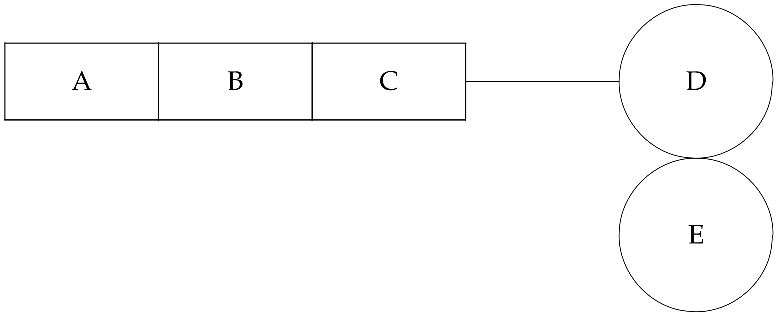

The length values for lines are interpreted as Cartesian coordinates rather than the actual length of the line drawn. There are separate standard values for the horizontal and vertical lengths, which are 72 and 36 points, respectively, by default.

```
line; line up right; line left; line down;
```



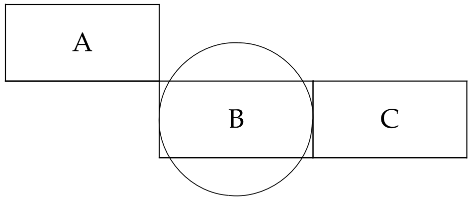
If a sequence of closed shapes occurs, they are joined together according to the current direction of motion, but a closed shape following a line joins according to the direction of the line:

```
box "A"; box "B"; box "C"; line;
down; circle "D"; circle "E";
```



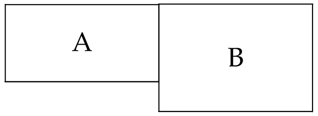

There is a **join** option for specifying where a closed shape joins its predecessor:

```
box "A"; box join top left "B";
circle join centre; box join left "C";
```



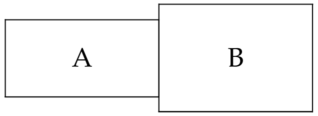

**Join** specifies a point on the new shape that is to be joined to a point on the previous shape. Thus, in the above example, the top left-hand corner of the second box is the point which is joined to the first box. The joining point on the previous shape is the complementary position by default, but it can be also specified explicitly. For example:

```
box "A";
box depth 50 join top left to top right "B";
```




There are nine possible joining points – the four corners, the midpoints of the four edges, and the centre point. The midpoints of the edges are identified by the unqualified names of the edges. Thus, if the only joining information is an edge name, two boxes are joined with the midpoints of the edges aligned:

```
box "A";
box depth 50 join left "B";
```



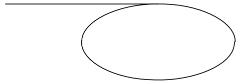

Joins can refer to items other than immediate predecessors, by using *labels*. For example:

```
BOXA: box "A";
BOXB: box "B";
      circle radius 10 join centre to centre of BOXA;
      box width 20 depth 20 join centre
        to centre of BOXB;
```




**Join** may also be used for specifying how a closed shape joins onto a line:

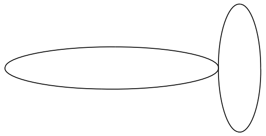
```
line; ellipse join top;
```



When a circle or an ellipse is being joined, the 'corner' joining points refer to points on the circumference, not the corners of the bounding box.

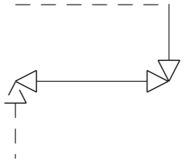
The size of an ellipse is specified as a width and a depth, which determine the horizontal and vertical axes. For example:

```
ellipse width 100 depth 20;
ellipse width 20 depth 60;
```



Lines may be drawn dashed, and, by using the **arrow** command, with arrowheads on one or both ends.

```
line dashed; arrow down; arrow left both;
arrow down back dashed;
```



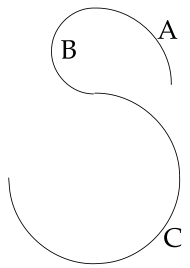
By default, **arrow** requests an arrowhead on the end of the line. The option **both** gives arrowheads on both ends, while **back** gives a backward-pointing arrowhead only.

Circular arcs are the other form of non-closed shape which ASPIC supports. By default an arc is drawn in an anti-clockwise direction, for 90 degrees, and at a standard radius (default 36). If an arc follows a line or another arc, it starts in the same direction. If the very first shape is an arc, its initial direction is upwards:

```
arc "A"; line left; arc "B";
```

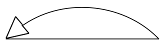

The angle subtended, the radius, and the direction of drawing can all be specified:

```
arc "A"; arc angle 180 radius 20 "B";
arc clockwise angle 270 radius 40 "C";
```



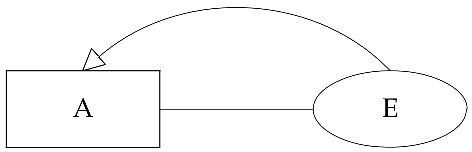

Arcs can also be drawn from and to particular points, and, by using the **arcarrow** command, with arrowheads:

```
line; arcarrow from end to start;
```



In this example, the positions 'end' and 'start' were taken to refer to the last-drawn shape. In order to refer to other shapes, ASPIC allows them to be labelled:

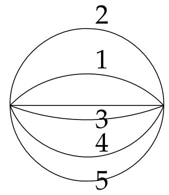
```
BOXA: box "A"; line; ellipse "E";
arcarrow from top to top of BOXA;
```





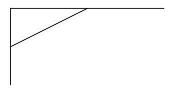

When an arc of this type is drawn, *either* the radius *or* the angle subtended at the centre of the arc *or* the 'depth' of the arc *or* a point through which the arc is to pass may be specified, but only one of these. If none is given, an angle of 90 degrees is used.

```
AA: line;
arc from end to start "1";
arc from end of AA to start of AA angle 180 "2";
arc clockwise from end of AA
  to start of AA radius 100 "3";
BB: arc to start depth 24 "4";
arc clockwise to start via middle of BB
  plus (10, -10) "5";
```



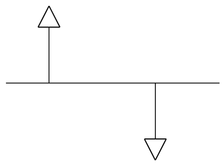

The starting and ending points of straight lines can also be specified explicitly; if both are specified, then that determines the length of the line.

```
AA: line up; BB: line right;
line from middle of AA to middle of BB;
```



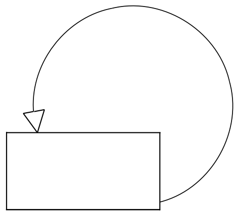
Positions on a line or arc may be specified as **start**, **middle**, or **end**; the word **centre** is used for the centre of a closed shape, or for the centre of an arc. More precise positioning can be achieved by specifying a fraction of the way along the line from a named position:

```
AA: line right 100;
arrow up from 0.2 start of AA;
arrow down from 0.3 end of AA;
```



The upward arrow starts at a point that is 0.2 of the way along the line from the start, while the downward arrow starts at a point that is 0.3 of the way along the line from the end. This feature also applies to the edges of boxes:

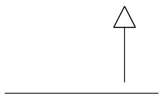
```
AA: box;
arcarrow from 0.1 right of AA
to 0.2 top of AA angle 270;
```



Positions along the top and bottom are measured from the left; positions on the sides are measured from the bottom. If these options are used with a circle or an ellipse, the positions used are those of the circumscribing box.

Any position can be further modified by an explicit distance, specified as a vector enclosed in brackets:

```
line; arrow up from middle plus (20,5);
```

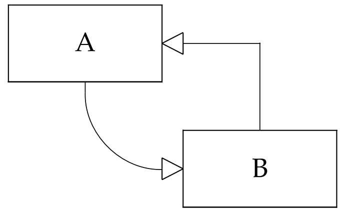


The arrow starts at a position 20 points to the right of the middle of the line, and 5 points above it.

There is one further positioning feature which is useful for horizontal or vertical lines. It allows the end of such a line to be aligned with a given point, which is often the easiest way to describe certain kinds of drawing:



```
BOXA: box "A";
line down 5; arc; arrow right 10; box "B";
line up align centre of BOXA;
arrow to right of BOXA;
```

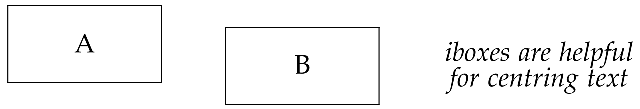



The **align** option is used in place of **to**; it defines a position, but only one of its coordinates is used as a coordinate of the end of the line. In the example above, a vertical line was specified, and so only the vertical coordinate of ‘centre of BOXA’ was used.

ASPIC pictures are best specified as a sequence of shapes whose positions are inter-related. This makes the pictures easy to adjust as they are being created, and also easy to change subsequently.

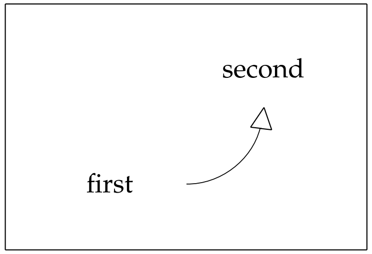
Many pictures contain shapes which are not connected to other shapes in the picture. ASPIC does allow absolute positioning for shapes, but it is often more useful to position these shapes in relation to the others. This can be done using *invisible* lines, boxes, and arcs.

```
box "A"; iline right 30 down 10; box "B";
ibox width 150 "$it{iboxes are helpful}"
"$it{for centring text}";
```




ASPIC can be requested to draw a frame round any picture, by means of the **boundingbox** command. This is followed by one dimension, which specifies the space to be left between the bounding box of the picture and the frame:

```
boundingbox 10;
ibox "first"; arcarrow; ibox "second";
```




Note that in this case the bounding box of the picture is determined by the invisible boxes which contain the text.

By default, ASPIC positions the picture such that the left-hand edge of its bounding box (or frame, if there is one) is at the left-hand side of the page, taking into account any SGCAL indent that may be in force.

The picture can be positioned in the middle of the page by including the **centre** command, followed by a value which specifies the current line length. This has been done for all the examples in this document.

This chapter has introduced many, but not all, of the features of ASPIC. The remaining chapters specify the form of the input more rigorously, and list each command together with its options.



# 3. General operation of ASPIC

ASPIC processes its input file in order. The commands contain instructions for moving about on a plane and causing shapes to be drawn and text to be printed. The units of length used by ASPIC are printer's *points*, of which there are 72 to an inch.

ASPIC distinguishes between closed and open shapes. The closed shapes are boxes, circles, and ellipses, while the open shapes are lines and circular arcs. There are default sizes for everything, and text strings may be associated with each shape. Unless explicitly positioned, each shape is placed adjacent to its predecessor, according to certain rules.

There is the concept of a *current position* on the plane, at which the next shape will be drawn by default. The default current position is at the origin of the plane's coordinate system. This point is not necessarily at the bottom left of the final picture. ASPIC computes a bounding box for the entire picture, and arranges that the bottom left of the bounding box is positioned at the bottom of the picture's space on the output page.

It is possible to specify absolute positions on the drawing plane, but it is better to describe a picture in terms of relative positions between the shapes that comprise it, because such a description is much easier to change. For many pictures, it is not necessary to worry about absolute coordinates or the position of the origin.

There is also the concept of a *current direction* which is used when a number of shapes are drawn in sequence, without any instructions as to their relative positions.

The default current direction is to the right, so the sequence '**box**; **arrow**; **box**;' places the three items in a horizontal row. There are commands to set the current direction, and, for the shape-drawing commands, options to override it.

Only very simple pictures can be drawn as a series of shapes in which each shape is positioned relative to its predecessor. ASPIC allows shapes to be labelled so that branches in the sequence of shapes may be constructed, and cross-references between different parts of the picture may be expressed.

There are many parameters for controlling the size and style of the shapes that are drawn; all of them have defaults, and most of these can be altered.

# 4. Input syntax

ASPIC input consists of a sequence of commands, each of which must be terminated by a semicolon. Newlines and other white space may appear between the components of a command in the usual way. If a sharp character (#) is encountered when a command is expected, the remainder of the input line is ignored. This provides a facility for including comments in ASPIC input.

A command consists of four components:

- Commands which notionally cause drawing or printing to take place may start with one or more labels, each terminated by a colon. A label consists of a sequence of letters and digits, starting with a letter. Upper case letters are commonly used in labels as it makes them stand out. Other commands may not be labelled.
- All commands contain a command name.
- Many commands allow optional options specifications to follow the command name. Each option consists of a keyword, possibly followed by a value. They may appear in any order.
- Following the options, on commands which notionally cause drawing or printing to take place, there may be any number of strings, enclosed in double quotes. The double-quote character itself may be included by doubling. The strings specify text which is to be printed at an appropriate position in the shape which is to be drawn (details of the positioning are given with the commands for each shape below). Strings may not extend over line boundaries.

The case of letters is significant in all components of ASPIC commands. The following example contains all four components:

| *label* | *command* | *options* | *strings* |
|---|---|---|---|
| `A:` | `box` | `dashed width 100` | `"first" "second";` |

The strings are not processed by ASPIC at all, but are handed back to the typesetting program verbatim. Therefore, in the case of SGCAL, they may contain any flags which SGCAL recognizes to effect, for example, changes of font. It is also necessary to escape characters which are normally escaped in SGCAL input. For example

```
circle "$it{italic @@}"
```

causes the text ‘italic @’ to be printed in italics inside a circle.



# 5. ASPIC macros

To simplify repetitious input, ASPIC contains a simple macro facility. A macro is defined by the command **macro**, which is followed by a name and a macro body. The body consists *either* of all the following text up to the first non-quoted semicolon, *or*, if the first character after the name is an opening brace, all the text up to the next non-quoted closing brace, which must be followed by a semicolon. For example,

```
macro bigbox box width 100 depth 100;
macro box2 { box; box; };
```

Macros are called simply by using their names as command names. Macro names must not be the same as the names of inbuilt commands.

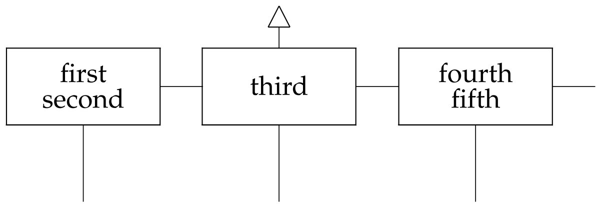
Macros can be called with arguments, which are treated as character strings. White space is used to delimit macro arguments, unless they are enclosed in either single or double quotes. If double quotes are used, they are retained when the contents of an argument are substituted into the macro body.

Macro arguments are referenced in the macro body by items of the form `&1`, `&2`, etc. If the character `&` is required for another purpose in a macro body, it must be doubled. If the string `&$` appears in a macro body, it is replaced by a sequence number which is incremented for each macro called. This can be used to generate unique labels for shapes which are drawn as a result of macro calls. Here is an example of a macro which draws a box containing given text, with two lines attached to it:

```
macro item {
  B&$: box &1 &2; line down;
  line right 20 from right of B&$;
  };
```

This macro can be used to generate an array of boxes as shown below. Because ASPIC allows multiple labels on shapes, these compound items can themselves be labelled, as in this example:

```
item "first" "second";
MID: item "third";
item "fourth" "fifth";
arrow up 20 from MID;
```




Because the macro was not defined with double quotes surrounding the argument references, double quotes had to be used when calling it in order to supply strings to the **box** command. If the quotes had been present in the definition, they could not be have been used in the calls, but single quotes could have been used if the arguments contained spaces.

If an argument which has not been supplied is referenced, nothing is substituted; thus the second call of **item** above expands into a call to **box** with only a single string argument. If too many arguments are supplied, the surplus ones are simply left in the input following the substituted text.

It is not necessary to include a semicolon before a terminating brace when defining a macro. If the semicolon is present, it is included in the replacement text when the macro is called. Sometimes it is useful to be able to set up a macro which generates part of a command such that additional options can be added on each call. This can be done by omitting the terminating semicolon. For example,

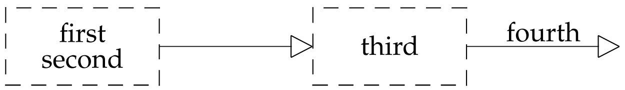
```
macro smallbox { box width 20 depth 20 };
```

When this macro is called, any text following the macro name is simply added to the command. For example,

```
smallbox dashed "text for smallbox";
```

When such additional text is required, and also not all the arguments of a macro are to be supplied, the vertical bar character can be used to mark the end of the arguments. For example,

```
macro dashbox { box dashed &1 &2; arrow };
dashbox "first" "second";
dashbox "third" | "fourth";
```






In the first call, the two strings are taken as arguments of the macro; in the second call, the second string is added onto the end of the replacement text, and therefore goes with the **arrow** command.

# 6. Types of value

Numbers may always be specified with an optional decimal point and fractional part. Negative numbers are preceded by a minus sign. Numbers are held to three decimal places. In the descriptions of the commands which follow, the following types of value are specified:

***angle***
A positive or negative number, specifying an angle in degrees.

***boxpoint***
One of the phrases *top, bottom, left, right, centre, bottom left, bottom right, top left*, or *top right*. The first four refer to the midpoints of the respective sides.

***direction***
One of the words *up, down, left,* or *right*.

***greylevel***
A number in the range 0.0 to 1.0, where 0.0 is black and 1.0 is white.

***label***
A label which identifies an existing shape, that is, one whose definition falls earlier in the input file.

***length***
A positive or negative number, specifying a length in points.

***position***
A <*position*> identifies a point in the plane. It is *either* an absolute position, specified as a pair of x-y coordinates, separated by a comma and enclosed in round brackets, *or* a relative position as follows:

<*fraction*> <*point*> of <*label*> plus <*vector*>

where all but <*point*> are optional.

If the referenced object is a box, circle, or ellipse, then <*point*> is a <*boxpoint*>. The 'corner' points of a circle or ellipse are the intersections of the shape with the diagonals of the bounding box.

If the referenced object is an arc or a line, then <*point*> is one of the words *start, end*, or *middle*. In addition, for an arc, *centre* may be specified.



<*fraction*> is either a single number or two numbers separated by a slash. It specifies a position part-way along a straight line. If it is present, there are some additional constraints on <*point*>:

- If the referenced object is closed, then <*point*> must be one of *top, bottom, left*, or *right*, and the straight line to which <*fraction*> refers is the appropriate side of the bounding box of the object. The fraction is measured from the left of horizontal lines, and from the bottom of vertical lines.
- If the referenced object is not closed, then <*point*> must be one of *start* or *end* – the fraction is then taken from that end of the line.

The presence of <*fraction*> changes the meaning of *top, bottom, left*, or *right*. With no <*fraction*>, these words refer to the midpoints of the respective sides of the bounding box; when <*fraction*> is present, they refer to the sides themselves.

The final optional component of a <*position*> is <*vector*>, which is two numbers separated by a comma and enclosed in round brackets. It specifies a Cartesian adjustment to the position defined by the remainder of the <*position*>.

Here are some examples of <*position*> specifications:

```
(45,67)
top
top plus (10,0)
centre of A
bottom right of B plus (0,-5)
1/3 top of C
middle
end of line1
0.25 start of line3 plus (0,7)
```

***text***
Text is any number of items of the following form: a string enclosed in double quotes (with doubling for any double quotes within the string), followed optionally by a <*vector*> and one of */l, /r*, or */c*. If both are present, the <*vector*> must come first. For example,

```
"the quick brown fox"
"the quick" "brown fox"
"move this up"(0,20)
"justify right"/r
"move left and centre"(-20,0)/c
```

The presence of a <*vector*> causes the position at which the string is printed to be modified by the value of the <*vector*>. The other options specify left, right, or centre justification respectively. The default justification depends on the shape with which the string is associated, and is documented below.



# 7. Drawing shapes

All the commands which notionally cause shapes to be drawn and/or text to be printed are described in this chapter. Each command is summarised by listing its options and the type of value that must follow each option keyword, where relevant. A vertical bar is used to separate alternative kinds of value.

The **thickness** and **greyness** options are common to all these commands. The former specifies the thickness of the lines used to draw the shape, and the latter the greyness of the lines – a value of 0.0 is black and 1.0 is white. If these options are not present, default values are used.

Some combinations of options are mutually exclusive, and these are noted in the description of each command below.

When options are omitted, default values are used. There are separate sets of defaults for boxes, circles, ellipses, and lines. Many of these defaults can be changed by commands which are described in chapter xxx (*Changing environment parameters*) below.

## 7.1 arc

| | |
|---|---|
| **angle** | *<angle>* |
| **clockwise** | |
| **dashed** | |
| **depth** | *<length>* |
| **from** | *<position>* \| *<label>* |
| **greyness** | *<greylevel>* |
| **radius** | *<length>* |
| **thickness** | *<length>* |
| **to** | *<position>* |
| **via** | *<position>* |
| **<*direction*>** | |
| **<*text*>** | |

A circular arc is drawn in an anti-clockwise direction from its starting point to its ending point, unless the **clockwise** option is present. The position and size of an arc may be specified in one of four different ways:

(1) If neither **from** nor **to** is specified, the arc is positioned according to the previously drawn shape. If this was a line or an arc, the new arc continues in the same direction unless an explicit <*direction*> is given.

Following a closed shape (box, circle, or ellipse), if an explicit direction is not given, the current direction is used. The direction determines the position on the closed shape from which the arc starts (‘up’ starts from the middle of the top of a box, and so on).

The user may supply either or both of a radius and an angle. If no radius is supplied, the default arc radius is used; if no angle is supplied, an arc of 90 degrees is drawn. The **depth** and **via** options may not be given.

(2) If **from** is supplied without **to**, both a radius and an angle may be supplied. If no radius is supplied, the default arc radius is used; if no angle is supplied, an arc of 90 degrees is drawn. The initial direction of the arc is given by <*direction*>, defaulting to the current direction. The **depth** and **via** options may not be given.

(3) If **to** is given without **from**, a starting point is determined from the previous shape, and then the action is as described in the following paragraph. If the previous shape is a line or arc, its end point is used; otherwise the direction (explicit or implicit) is used to decide on which side of the bounding box of the closed shape to place the starting point, but not for any other purpose. The midpoint of the appropriate side is used.

(4) If both **from** and **to** are given (or if **to** is given and **from** is determined from the previous shape as just described), then there are four ways in which the size of the arc can be specified:

(a) The **radius** option can be used to give an explicit radius; this must not be less than half the distance between the end points.

(b) The **angle** option can be used to specify the angle subtended at the centre of the arc.

(c) The **depth** option can be used to specify the distance between the midpoint of the line joining the end points and the midpoint of the arc. If the **depth** option specifies a distance that is more than half the distance between the end points then an arc of more than 180 degrees is drawn.

(d) The **via** option can be used to specify a third point through which the arc is to pass. This point must not be be on the line joining the end points, and it must also be on the appropriate side of that line. If it is not suitable, an error message is output.

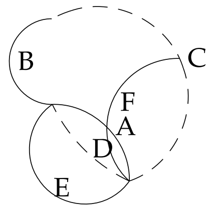
If none of these options is given, an arc that subtends 90 degrees at its centre is drawn. If more than one of these options is given, an error message is generated, and all but one are ignored.



If the **from** option specifies the label of a closed shape without further qualification, the actual starting point on that shape is determined by the initial direction of the arc.

Texts are printed approximately at the midpoint of the arc, and are left-justified by default. The following example illustrates various types of arc:

```
A: arc "A";
B: arc clockwise radius 20 angle 180 "B";
C: arc dashed from start of A to end of B
    angle 190 "C";
arc clockwise dashed from start of A
  to end of A radius 75 "D";
arc to start depth 30 "E";
arc clockwise to middle of C via middle of A "F";
```



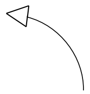

### 7.2 arcarrow

| | |
|---|---|
| **angle** | <*angle*> |
| **back** | |
| **both** | |
| **clockwise** | |
| **dashed** | |
| **depth** | <*length*> |
| **from** | <*position*> \| <*label*> |
| **greyness** | <*greylevel*> |
| **radius** | <*length*> |
| **thickness** | <*length*> |
| **to** | <*position*> |
| **via** | <*position*> |
| **<*direction*>** | |
| **<*text*>** | |

The options for **arcarrow** are exactly as for **arc**, with the addition of **both** (which specifies a double-headed arrow) and **back** (which specifies a backwards-pointing arrow only). If neither is given, an arrowhead is drawn at the end of the arc.

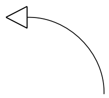
Arrowheads are drawn within the length of the arc so if, for example, a 90-degree arc is drawn from absolute angle zero, the arrow head is not horizontal:

```
arcarrow;
```



A horizontal arrowhead can be drawn by adding it on afterwards:

```
arc; arrow left 10;
```



However, this does extend beyond the end of the arc.

### 7.3 arrow

| | |
|---|---|
| **align** | *<position>* |
| **back** | |
| **both** | |
| **dashed** | |
| **down** | *<length>* |
| **from** | *<position>* \| *<label>* |
| **greyness** | *<greylevel>* |
| **left** | *<length>* |
| **right** | *<length>* |
| **thickness** | *<length>* |
| **to** | *<position>* |
| **up** | *<length>* |
| **<*text*>** | |

The **arrow** command has exactly the same options as **line** (see section xxx below), but with the addition of **both** (specifying a double-headed arrow) and **back** (specifying a backwards-pointing arrow). If neither is given, an arrowhead is drawn at the end of the line.



### 7.4 box

| | |
|---|---|
| **at** | *<position>* |
| **dashed** | |
| **depth** | *<length>* |
| **greyness** | *<greylevel>* |
| **join** | *<boxpoint>* to *<position>* \| *<label>* |
| **thickness** | *<length>* |
| **width** | *<length>* |
| **<*text*>** | |

This command causes a rectangular closed box to be drawn. The **width** and **depth** options specify the horizontal and vertical dimensions of the box. Defaults are used if either of them is omitted.

The **at** option specifies the position of the centre of the box; if not given, the centre point is computed by reference to the previous shape. In the absence of a **join** option, if the previous shape was a line or arc, the midpoint of an appropriate side of the box is joined onto its end; if it was a closed shape, the side which is abutted depends on the current direction. If there was no previous shape, the centre is placed at the origin of the coordinate system.

The **join** option specifies how the box is to be joined to a previous shape, and is mutually exclusive with **at**.

The option takes three different forms:

(1) If <*position*> is given, then the given point on the box is placed at the given position. For example:

```
box join bottom right to centre of C;
box join top left to start plus (10,15);
```

As with all <*positions*>, if ‘of <*label*>’ is omitted, the preceding shape is implied.

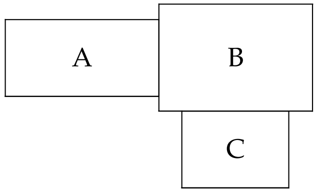
(2) If no <*position*> is given, and the previous shape (or the named shape if <*label*> is present) is a closed shape, then the given point is joined to the complementary point of the referenced shape. For example:

```
box join top;
box join top left to A;
```

(3) If no <*position*> is given, and the previous shape (or the named shape if <*label*> is present) is not a closed shape, then the given point is joined to its end.

Note that if boxes of different dimensions are joined by naming their edges, the middle points of the edges are made coincident:

```
box "A"; box depth 50 "B";
box width 50 join top "C";
```




Any text items are centred at the centre of the box. Because ASPIC does not process the text in any way, it cannot tell whether the text will actually fit.

### 7.5 circle

| | |
|---|---|
| **at** | *<position>* |
| **dashed** | |
| **greyness** | *<greylevel>* |
| **join** | *<boxpoint>* to *<position>* \| *<label>* |
| **radius** | *<length>* |
| **thickness** | *<length>* |
| **<*text*>** | |

The **at** option specifies the position of the centre of the circle; it is mutually exclusive with the **join** option, which specifies how the circle is to be joined to the previous shape, exactly as for boxes (see above). Text items are centred at the centre of the circle.

### 7.6 ellipse

| | |
|---|---|
| **at** | *<position>* |
| **dashed** | |
| **greyness** | *<greylevel>* |
| **depth** | *<length>* |
| **join** | *<boxpoint>* to *<position>* \| *<label>* |
| **thickness** | *<length>* |
| **width** | *<length>* |
| **<*text*>** | |

The options for **ellipse** are the same as for **circle**, except that **radius** is replaced by **width** and **depth**, which specify the lengths of the horizontal and vertical axes. That is, they specify the size of the bounding box.



### 7.7 iarc

*<as arc>*

The **iarc** command defines an invisible arc. Its options are the same as for the **arc** command. Although the arc is not actually drawn, any text supplied is printed.

### 7.8 ibox

*<as box>*

The **ibox** command defines an invisible box. Its options are the same as for the **box** command. Although the box is not actually drawn, any text supplied is printed.

### 7.9 icircle

*<as circle>*

The **icircle** command defines an invisible circle. Its options are the same as for the **circle** command. Although the circle is not actually drawn, any text supplied is printed.

### 7.10 iline

*<as line>*

The **iline** command defines an invisible line. Its options are the same as for the **line** command. Although the line is not actually drawn, any text supplied is printed.

### 7.11 line

| | |
|---|---|
| **align** | *<position>* |
| **dashed** | |
| **down** | *<length>* |
| **from** | *<position>* \| *<label>* |
| **greyness** | *<greylevel>* |
| **left** | *<length>* |
| **right** | *<length>* |
| **thickness** | *<length>* |
| **to** | *<position>* |
| **up** | *<length>* |
| **<*text*>** | |

This command draws a straight line. The start is given by the **from** option; if it specifies a label only, the actual starting point on the referenced shape is its end point if it is a line or arc, or is determined by the direction of the line otherwise. For example, a line to the right from a box starts at the midpoint of the right-hand edge.

The end point of the line can be specified in three different ways:

(1) A pair of horizontal and vertical distances (that is, relative Cartesian coordinates) can be given. If a direction is given without a length, the current standard length for that direction is used. Thus, the following are all valid:

```
line;
line right;
line up right;
line down 20 left;
line down 40 right 60;
```

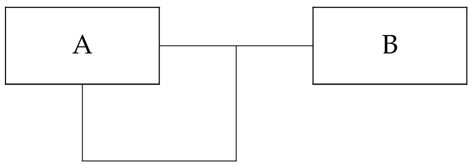
The first of these draws a line in the current direction (up, down, left, or right).

(2) The **to** option can be used to define the endpoint by reference to the previous or some other shape. For example:

```
box;
line from bottom right to 0.25 top;
```

(3) If the line is horizontal or vertical, the **align** option can be used. (It is ignored if the line is not horizontal or vertical.) In each case, only the vertical or horizontal coordinate of the <*position*> specified by **align** is used, as appropriate. For example:

```
A: box "A";
L: line right;
B: box "B";
   line down from bottom of A;
   line right align middle of L;
   line up align right of A;
```




As always, a reference to an unqualified box edge implies the midpoint of that edge.

For a horizontal line, text is centred at the centre of the line. If there is more than one string, they are positioned both above and below the line.



For non-horizontal lines, text is left-justified at the centre point of the line.

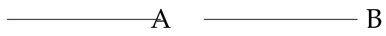
### 7.12 text

**at** <*position*>
**<*text*>**

This command provides a means of printing text without drawing a graphic shape. The text is centre-justified by default. If no position is given, the text is positioned with reference to the previously drawn shape. If it was a closed shape, its centre is used; if it was a line, its middle point is used; if it was an arc, the centre of the arc is used.

String options can be used to make fine adjustments to the position of the text. For example, if text is printed to the right of a horizontal line, it should normally be specified as left-justified.

```
line; text at end "A";
iline right 20;
line; text at end "B"/l;
```




In the first of these two examples, the default centre-justification has caused the text to print on top of the line.

# 8. Overall configuration

This chapter describes those commands which affect the overall appearance of the picture, and which normally appear at the start of the ASPIC input.

### 8.1 boundingbox

This command requests that ASPIC draw a frame round the picture. It must be followed by a length, specifying margin width between the actual bounding box of the picture, and the frame. For example,

```
boundingbox 20;
```

If a value of zero is given, the frame that is drawn is the actual bounding box. Because ASPIC does not process text items itself, it has to guess a bounding box for them, and so under some circumstances the computed bounding box for a picture may not be strictly accurate.

### 8.2 centre

This command must be followed by a length, specifying the page width within which centring is to happen. For example,

```
centre 450;
```

When ASPIC is called from within SGCAL, the length is normally specified as `~~sys.linelength`, which causes SGCAL to fill in the current linelength before handing the input to ASPIC. However, if an indent is set in SGCAL when ASPIC is called, it will apply to SGCAL's positioning of the picture, so must be taken into account if centring is required. A command such as

```
centre ~~sys.linelength - ~~sys.indent
```

is needed to achieve centring between the indent and the line length.

### 8.3 magnify

This command specifies overall magnification of the picture. It must be followed by a single number. For example

```
magnify 0.8;
magnify 1.5;
```

The magnification may be changed in the middle of an ASPIC input sequence; the new value applies to those shapes which follow.

Magnification does *not* apply to text. If smaller or larger text is required, suitable SGCAL fonts must be set up and used.



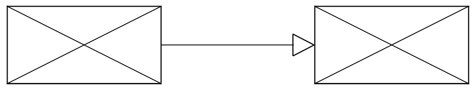
# 9. Altering the 'last' object

Occasionally it is useful to be able to change which shape is considered to be the previous shape when the next shape is drawn. The **goto** command, which is followed by a label, is used to do this. It can be particularly useful inside macros. Consider a macro which draws a box with diagonals:

```
macro diag {
A&$: box;
line from top right to bottom left;
line from top left of A&$ to bottom right of A&$;
goto A&$; };
```

When this macro is called, because of the final **goto**, any subsequent shapes will be positioned with reference to the box (which is the bounding box for this compound shape). For example:

```
diag; arrow; diag;
```



# 10. Changing the current direction

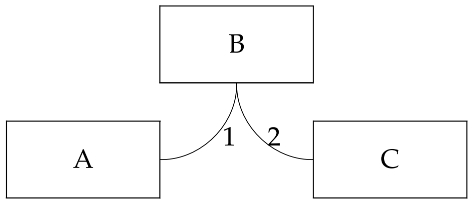
The current direction defaults to 'right'. It can be changed by the commands **up**, **down**, **left**, and **right**, which have no arguments.

The current direction is used in the following circumstances:

- When drawing a line, if no direction is specified;
- When one closed shape follows another, to determine their relative positions if no **join** option is given;
- When an arc follows a closed shape, without specification of its end points.

Note that the current direction is *not* used when a closed shape follows a line or arc; the position of the closed shape in this case is determined by the direction of the end of the line or arc. For example,

```
right; box "A"; arc "1"; box "B";
down; arc "2"; box "C";
```




The initial direction of the arcs in this example is determined by the current direction, but the boxes that follow them are positioned with reference to the ending direction of the arc. (This can be changed by using the **join** option of the **box** command.)



# 11. Saving and restoring the environment

The Aspic *environment* consists of a number of parameters which control the way shapes are drawn. Many of them can be over-ridden for a single shape by the use of options on the drawing command. The parameters are listed in the following table, together with their initial values:

| | | |
|---|---|---|
| arc radius | 36.0 | |
| arrowhead length | 10.0 | |
| arrowhead width | 10.0 | |
| box dash parameters | 7.0 | 5.0 |
| box depth | 36.0 | |
| box edge greyness | 0.0 | |
| box edge thickness | 0.5 | |
| box width | 72.0 | |
| circle dash parameters | 7.0 | 5.0 |
| circle edge greyness | 0.0 | |
| circle edge thickness | 0.4 | |
| circle radius | 36.0 | |
| ellipse dash parameters | 7.0 | 5.0 |
| ellipse depth | 36.0 | |
| ellipse edge greyness | 0.0 | |
| ellipse edge thickness | 0.4 | |
| ellipse width | 72.0 | |
| line dash parameters | 7.0 | 5.0 |
| line greyness | 0.0 | |
| line horizontal length | 72.0 | |
| line thickness | 0.4 | |
| line vertical length | 36.0 | |
| text line depth | 12.0 | |
| text font depth | 6.0 | |
| current direction | right | |
| magnification | 1.0 | |

The **magnify** command is described in section 8.3, and commands to change the current direction are described in chapter 10. Commands to change the remaining values are described in the next chapter.

It is often useful to be able to save the current state of the environment and restore it later. The **push** and **pop** commands are provided to do this. **Push** puts a copy of the current environment onto a stack, while **pop** restores the environment from the top item on the stack.

# 12. Changing environment parameters

In addition to the commands for changing the current direction and the **magnify** command, which are specified above in chapter 10 and section 8.3, respectively, the following commands are provided for changing the values of environmental parameters. All of them take a single numerical argument except for those that set dashed line parameters, which take two, specifying the length of dashes and the length of gaps, respectively.

| | |
|---|---|
| `arcradius` | default radius for arcs |
| `arrowlength` | length of arrowheads |
| `arrowwidth` | width of arrowheads |
| `boxdash` | dash parameters for boxes |
| `boxdepth` | default depth of boxes |
| `boxgreyness` | greyness of box edges |
| `boxthickness` | thickness of box edges |
| `boxwidth` | default width of boxes |
| `circledash` | dash parameters for circles |
| `circlegreyness` | greyness of circle edges |
| `circleradius` | default radius of circles |
| `circlethickness` | thickness of circle edges |
| `ellipsedash` | dash parameters for ellipses |
| `ellipsedepth` | default depth of ellipses |
| `ellipsegreyness` | greyness of ellipse edges |
| `ellipsethickness` | thickness of ellipse edges |
| `ellipsewidth` | default width of ellipses |
| `fontdepth` | line spacing for text |
| `hlinelength` | default horizontal length for lines |
| `linedash` | dash parameters for lines |
| `linegreyness` | greyness of lines |
| `linethickness` | thickness of lines |
| `textdepth` | height of text characters |
| `vlinelength` | default vertical length for lines |

Each kind of shape has its own set of parameters for controlling the default thickness and greyness of the lines used to draw it, and the appearance of dashed lines. Setting the dash parameters does not of itself cause dashed lines to be drawn; the **dashed** option must be given with the drawing command.

Line thickness is specified in points; line greyness is specified on a scale from 0.0 to 1.0, with 0.0 being black and 1.0 being white. For example,

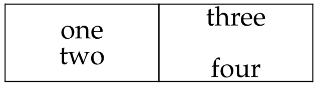
```
linethickness 1; linegreyness 0.5;
boxthickness 4; boxgreyness 0.8;
line; box; line;
```





The **textdepth** parameter controls the vertical separation of multiple text items. For example,

```
box "one" "two"; textdepth 24; box "three" "four";
```




The **fontdepth** parameter should be set to the approximate height of letters in the font being printed. It is used when positioning text vertically, for example, when centring a single line of text within a box.

# 13. List of commands

This is a complete list of all ASPIC commands.

| | |
|---|---|
| **arc** | draw a circular arc |
| **arcarrow** | draw a circular arc with arrowhead(s) |
| **arcradius** | set default arc radius |
| **arrow** | draw a straight line with arrowhead(s) |
| **arrowlength** | set length of arrowheads |
| **arrowwidth** | set width of arrowheads |
| **boundingbox** | enclose picture in frame |
| **box** | draw a box |
| **boxdash** | set dash parameters for boxes |
| **boxdepth** | set default depth for boxes |
| **boxgreyness** | set default greyness for boxes |
| **boxthickness** | set default line thickness for boxes |
| **boxwidth** | set default width for boxes |
| **centre** | centre the picture horizontally |
| **circle** | draw a circle |
| **circledash** | set dash parameters for circles |
| **circlegreyness** | set greyness for circles |
| **circleradius** | set default radius for circles |
| **circlethickness** | set thickness of lines for circles |
| **down** | set current direction |
| **ellipse** | draw an ellipse |
| **ellipsedash** | set dash parameters for ellipses |
| **ellipsedepth** | set depth of ellipses |
| **ellipsegreyness** | set greyness for ellipses |
| **ellipsethickness** | set line thickness for ellipses |
| **ellipsewidth** | set width of ellipses |
| **fontdepth** | set height of letters |
| **goto** | set named shape as previous |
| **hlinelength** | set horizontal line length |
| **iarc** | draw an invisible arc |
| **ibox** | draw an invisible box |
| **icircle** | draw an invisible circle |
| **iline** | draw an invisible line |
| **left** | set current direction |
| **line** | draw a line |
| **linedash** | set dash parameters for lines |
| **linegreyness** | set greyness for lines |
| **linethickness** | set thickness of lines |
| **magnify** | magnify or reduce the picture |
| **macro** | define an ASPIC macro |
| **pop** | restore environment from the stack |
| **push** | push environment onto the stack |
| **right** | set current direction |
| **text** | print text at given position |
| **textdepth** | set separation of text items |
| **up** | set current direction |
| **vlinelength** | set vertical line length |

# 14. Format of the steering file

The location of ASPIC's steering file can be specified explicitly by a system variable, but if this variable is unset, ASPIC looks for a file called **AspicSteer** in the SGCAL library. Under Unix and MS-DOS, the system variable **ASPIC_STEERFILE** is used. Under RISC OS, **Aspic$Steerfile** is used.

The steering file consists of a number of lines, each consisting of a keyword followed by one or more strings in quotes. The sequence `\n` in a string is interpreted as newline character. Empty lines are ignored. Each line describes the text which ASPIC should output in a particular circumstance.

- **head**: This is followed by one string, which is output at the start of an ASPIC picture. There may be any number of **head** lines.
- **tail**: This is followed by one string, which is output at the end of an ASPIC picture. There may be any number of **tail** lines.
- **space**: This is followed by two strings. Aspic outputs them surrounding a number which is the depth of the bounding box of the picture. It then outputs the picture, assuming a coordinate origin at the bottom left-hand corner of the space created.
- **up, down, left, right**: Each of these is followed by two strings. They are output surrounding a length to generate a movement of the current point.
- **line**: This is followed by three strings. They are output surrounding the horizontal and vertical lengths when a straight line is being drawn.
- **linewidth**: This is followed by two strings. They are output surrounding the width required for subsequent lines.
- **linegrey**: This is followed by two strings. They are output surrounding the grey level value required for subsequent lines.
- **linedash**: This is followed by three strings. They are output surrounding the mark and gap parameters for drawing dashed lines.
- **linesolid**: This is followed by one string. It is output to reset to solid (undashed) lines.
- **bezier**: This is followed by six strings. They are output surrounding the parameters for drawing a Bezier curve (which requires six arguments).



- **ctext**: This is followed by two strings. They are output surrounding any text that is to be centred at the current point.
- **ltext**: This is followed by two strings. They are output surrounding any text that is to be left-justified at the current point.
- **rtext**: This is followed by two strings. They are output surrounding any text that is to be right-justified at the current point.
- **home**: This is followed by one string. It is output in order to return the current point to the origin.

The steering file used to generate SGCAL input is as follows:

```
head ".newline\n"
head ".push\n"
head ".linedepth 0\n"
head ".parindent 0\n"
head ".newpar\n"

tail ".newline\n"
tail ".pop\n"
tail ".space 1ld\n"

space ".space " "\n"

up    "$U"  ""
down  "$D"  ""
left  "$s-" ""
right "$s"  ""

line   "$sr"  ","  ""
hline  "$hr"  ""
vline  "$vr"  ""

linewidth ".rulewidth "   "\n"
linegrey  ".rulegrey "    "\n"
linedash  ".ruledash "    " "   "\n"
linesolid ".ruledash 0\n"

bezier "$bc"  ","  ","  ","  ","  ","  ""
ctext  "$C"   "\n"
ltext  " "    "\n"
rtext  "$E"   "\n"

home   "\n"
```

* * *