imPRESS MANUAL

Preliminary Edition

IMAGEN CORPORATION

May 1981

IMPRESS/V0/810625

CONTENTS

1. INTRODUCTION.

This manual presents the imPRESS Layout Description Language for the IMPRINT-10 Printing System.

The imPRESS language is a powerful tool for text and graphics layout and is interpreted directly by the imPRESS processor, a module of the IMPRINT-10 system.

The following features of imPRESS should be mentioned:

- imPRESS provides a unified mechanism to deal with graphics and text. Both kinds of primitives can be mixed and share the same common description environment.
- Character shapes are loaded individually on a demand basis. The concept of font is used only as a convenient way to group and refer to characters, but only those characters required to print a page need be loaded.
- The size of shapes is not restricted, thus large figures (like logos or digitized signatures) do not require to be partitioned into smaller elements.
- The number of different primitive shapes that may be printed on a page is only limited by the memory available in the IMPRINT-10 (which is ample and may be easily expanded).
- Shapes are automatically rotated by the imPRESS processor.
- The layout may be specified in any order. There is no need to presort the layout information.
- The set of text primitives in imPRESS allows for extremely compact representations for even fully typeset text.

2. OVERVIEW OF THE imPRESS LANGUAGE

The imPRESS language permits the definition of documents to be printed on any raster printing device, in particular the IMPRINT-10.

Documents are defined as sets of **pages**, which in turn consist of sequences of printing instructions.

The shapes that may be printed are either

predefined shapes called **glyphs**,

basic graphic primitives that describe elementary surfaces, like a rectangle, a straight line, etc.

composite graphic constructs, called **boxes**, previously defined using imPRESS.

Glyphs maybe resident in the printing system or may be downloaded from the host computer. In general imPRESS assumes the existence of a **glyph table**, where the predefined shapes (characters, logos, arbitrary bit maps) are kept. Commands are provided to manage the space available for the glyph table.

The impression of a glyph occurs at a point within the page called the **current position**. This point is changed ("advanced") by the printing of the glyph, or it may be arbitrarily altered by suitable commands. The language does not assume any presorting of the descriptions and thus the current position may change in any direction.

A set of line oriented commands are provided to facilitate the generation of textual documents. Though they do not add new descriptive capabilities the use of these commands result in extremely compact representations.

The imPRESS language allows ninety degree rotations of the printing direction, thus permiting "portrait" and "landscape" modes to intermix in the same page.

When presented with invalid descriptions the imPRESS processor will take a default action (like printing a warning mark on the page) and raise an error condition that will later be reported through the standard system error channel.

There are two standard representations of imPRESS descriptions. The first one, **textual representation**, is intended for human consumption, to allow the user to read the imPRESS description of a document. The second one, **compact representation**, is an efficient encoding of imPRESS descriptions, intended mainly for computer storage and transmission. The translation between textual and compact representations is straightforward.

3. BASIC CONCEPTS.

The imPRESS environment.

The imPRESS language describes documents as a set of **boxes** that will be printed on a sequence of pages.

A box is the description unit. Once it has been described it may be positioned anywhere in the output, and in any of the allowed rotations (please see below). The box may also be replicated as needed.

The commands **define box** and **set box** are intended for the definition and printing of a box respectively.

The description of a box refers to a certain **environment**. Also the description process may alter the environment using appropriate commands. The environment essentially defines:

the coordinate system with respect to which the different graphic elements are positioned,

a roving pointer onto the box (the **current position**) and

a set of global parameters that are implicitly referred to by various commands, thus yielding more concise representations.

Two commands, **save environment** and **restore environment**, facilitate manipulating the environment within different levels of a description.

Symbols, characters, fonts.

We will call **symbol** any particular shape that has associated with it a well known meaning. For instance the letter "a", or the greek letter "γ" or the symbol "‡".

In order to print a symbol its shape must be further specified, in particular its typographic style and its dimensions. So the customary procedure has been to group certain symbols represented in some common style in a **font**, and refer to each one of them by an associated **character index**. Thus within an imPRESS description a character is referenced by stating the font where it belongs and the corresponding character index.

A font must be "introduced" to the imPRESS processor before any of its characters may be used. This is accomplished by a **font declaration** command that associates with it a **font index**, which is valid throughout the printing job.

Font and character residence.

The imPRESS processor requires each individual character that has been referenced in a page to be accessible at the time the page is printed. Notice that there is no requirement at the font level, only those characters that are used must be present.

Some fonts are stored within the IMPRINT-10 controller (for instance in PROM, or in local mass storage) or are accessible through a well defined protocol (as in the case of a local

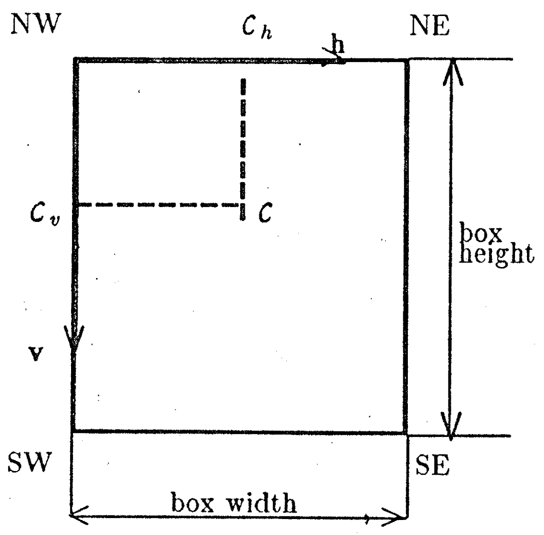

Illustration 1:

The Coordinate System.

network), these fonts are called **resident**. On the other hand **external** fonts reside on the host that submitted the printing job.

Resident fonts have a **font name** associated with them. For the usual fonts that name identifies the typographical style and the nominal size of the font.

In general, when a character that belongs in a resident font is mentioned, the imPRESS processor will take care of accessing it. Otherwise the printing job must contain **a character definition** command. (The latter is the mechanism provided in imPRESS to "download" a character from the host to the IMPRINT-10.)

Current position, coordinates, dimensions.

Commands in imPRESS refer to **positions** within the box being described. Positions are pairs of **coordinates: horizontal** and **vertical**. We will denote coordinates with the subcripts h (for horizontal) and v (for vertical). (Please see illustration 1.)

Coordinates are referred to a system that may have its origin in *any of the corners* of the box (the specific corner depends on the **rotation** as we shall see later).

A box has four corners: NW, NE, SW and SE. The **box width** is the distance between NW and NE (as well as between SW and SE). Similarly the **box height** is the distance between NW and SW (or NE and SE).

Both coordinates and dimensions are signed quantities and are measured in **pixels** both in the horizontal and vertical directions.

The **current position** is a distinguished point within the page: it specifies where the next figure is going to be printed and is the basis for most position computations in imPRESS. Also the printing actions modify the current position in the "expected" way, thus yielding very compact representations. (For instance text may be specified with almost no extra

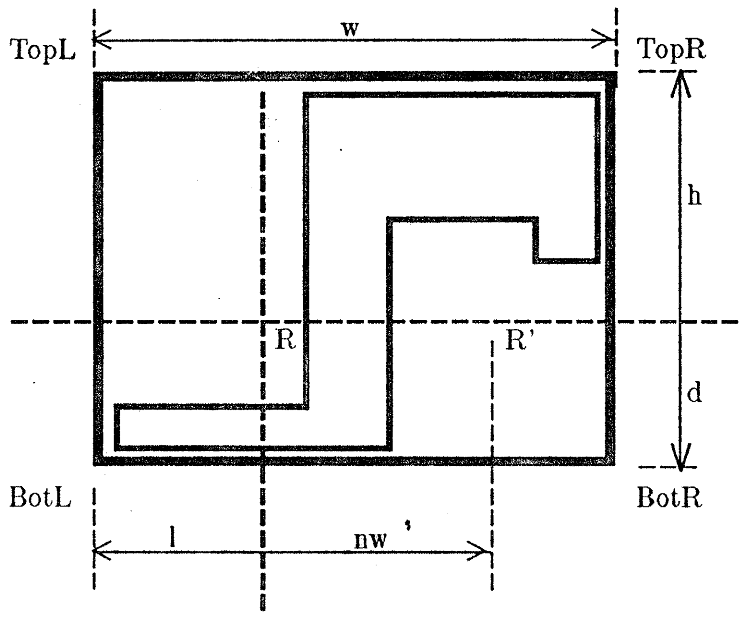

Illustration 2:

Figure Dimensions.

positioning commands.) The symbol C will stand for the current position, and C_h and C_v will refer to its horizontal and vertical coordinates.

Primitive figures.

The imPRESS language recognizes basically three different classes of figures that may be imprinted on a page:

Glyphs: a shape represented by means of a bit map. This category includes characters, logos, digitized signatures, etc.

Graphic primitives: each primitive represents a family of surfaces (lines, polygons, etc.). In each case, a set of parameters allows to choose the individual surface to be printed.

Boxes: previously defined composite figures.

Figures have a set of **dimensions** and points (please see illustration 2) associated with them:

A **reference point** (R) used to position the glyph.

A **width** (w) of the mask.

A **height** (h): vertical distance from the mask topline to the reference point.

A **depth** (d): vertical distance from the mask bottom line to the reference point.

A **line offset** (l) that relates the reference point to the current position when the glyph is printed.

Nominal height(nh) and **nominal width** (nw) define the change in the current position after the glyph is printed.

The **corner points** $TopL$, $TopR$, $BotL$ and $BotR$ delimit the glyph mask.

Figure printing.

Figures are overlaid (a logical "or" operation) onto the page. No restrictions apply about superimposition, so that many figures may overlay the same pixel.

When a figure is printed, its reference point is made to coincide with the current position. After printing the current position is altered (according to the nominal dimensions of the figure) if the **advance mode** in the environment is set to "advance".

The overlay process may be described as follows:

1. A figure is set so that its reference point R coincides with the current position C.
2. The figure mask is then overlaid ("ored in") onto the page.
3. If the advance mode is "advance" then the current position is altered by adding the glyph nominal dimensions to it:

$$C_h \leftarrow C_h + nw$$

$$C_v \leftarrow C_v + nh$$

Notice that as a result of the above rules it is very simple to describe different ways of printing. For instance *fixed and proportional spacing* are trivially represented if the nominal widths of characters within fonts are properly chosen, or *right to left printing* is automatic using negative nominal widths.

Changing layout orientation

So far nothing has been said about the actual orientation of the coordinate system with respect to the box.

Throughout a box description, the coordinate system should be seen as having its horizontal axis going from left to right and the vertical axis going from top to bottom. What this means is that in the case of latin alphabets, characters should be printed along increasing values of the horizontal axis, and a new baseline should be at a larger value of the vertical coordinate.

Figures are always set oriented along with the coordinate system (that is, the TopL corner point is set at the smallest h and v coordinates). Any required rotation of the figure is performed by the imPRESS processor.

At the beginning of the description the coordinate origin coincides with the NW corner of the box. In order to change the printing orientation imPRESS provides the command **rotate**. This command essentially affects the *position of the the box* with respect to the coordinate system, while leaving the current position C *stationary with respect to the box.*

The rotation itself can be relative (clokwise or anticlockwise) or absolute within the box (specified by selecting the corner for the coordinate system).

Illustration 3 shows the effect of various rotations:

(1) With the box in its initial state, the symbol "|" is set at the current position.

(2) After that a rotate clockwise command is issued. The current position now coincides with the (rotated) "|". A set position command changes the current position and a "||" is set.

(3) Another rotation, another change of position and a "|||" symbol.

(4) Same as before, but a "||||" is set.

Also if the box had not been rotated before the above example, the initial coordinate system would be at NW, and the first rotate would have been equivalent to a "rotate to NE" command, the second one to a "rotate to SE", etc.

The imPRESS processor effects the necessary computation of absolute coordinates within the page and the glyph mask rotations that may be needed for the actual printing task. Descriptions are always done in a "horizontal" fashion, and the imPRESS processor takes care of details, so that for instance advances along a baseline occur in a "natural" way, regardless of the box orientation.

Representation of glyphs.

A glyph is defined by its dimensions and its mask. Glyph masks are represented in two alternative ways: **encoded** or as a **raster map**.

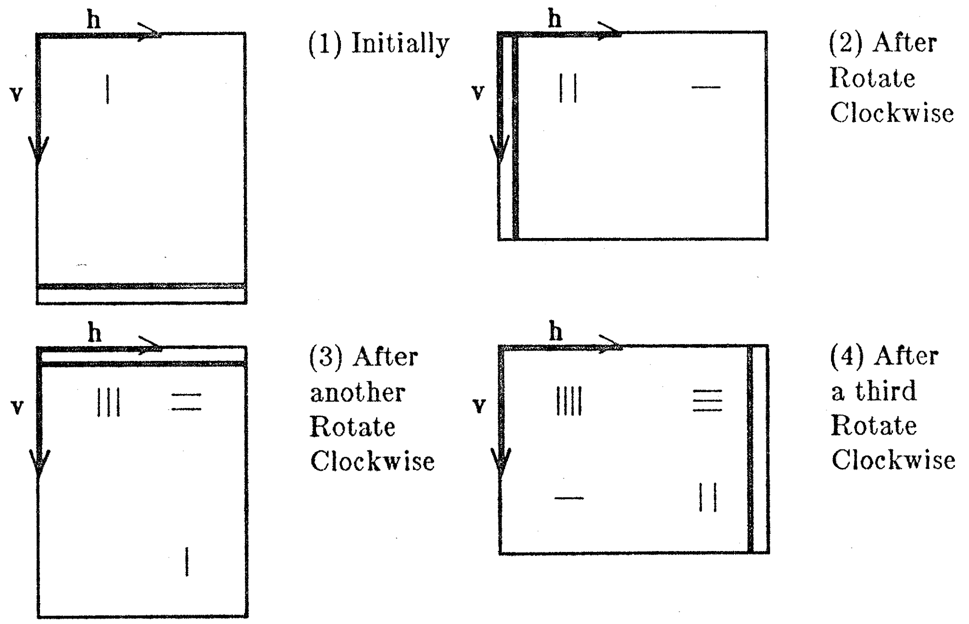
h
v
(1) Initially
(2) After Rotate Clockwise
(3) After another Rotate Clockwise
(4) After a third Rotate Clockwise

Illustration 3: Rotations and Box Orientation.

Encoded masks represent dot patterns in a highly compact form that the imPRESS processor expands into a raster when the glyph is referenced. This is the standard representation of the fonts provided by IMAGEN .

User defined glyphs are represented as raster maps.

Boxes and Pages.

As we have seen imPRESS creates descriptions of boxes. These descriptions are position and rotation independent, in the sense that a box so described may be set anywhere in a page (or another box, for that matter) in any of four possible orientations.

In order to store a box description a **define box** command is provided. This command associates a name to a box for further reference.

Another command, **set box**, replicates an already defined box. The replication simply substitutes the box definition for the set box command.

At definition time the environment is assumed to start with the coordinate system at the NW corner of the box. No assumptions are made about the rest of the environment (current font, line parameters, etc). On the other hand a box is set with its NW corner coinciding with the current position, and it is oriented so that its width and height lay in the increasing directions of h and v respectively. In this manner, the basic contents of a box may be provided at definition time with final details (position, orientation, etc.) provided when the box is set.

Finally imPRESS provides a mechanism to actually define the pages to be printed. The command **page** is essentially similar to a box definition but instead of storing it submits the definition of the page contents to the IMPRINT-10 Printing Module.

4. DESCRIPTION OF imPRESS.

An imPRESS description is a sequence of **commands** and **glyph** and **graphic primitive** references. Commands specify changes to the environment while the references indicate the printing of a given glyph or graphic element.

Representation issues.

There are two alternative ways to represent a given imPRESS description: **textual** and **compact encoding**. Both representations are exactly equivalent as far as their descriptive power, but they differ in their intended use. The first one is intended as a human readable, easily portable encoding, while the second one is better suited for storage and host to IMPRINT-10 communication purposes.

The IMPRINT-10 processor provides an utility (imPRESSDump) that prints out the textual representation of a compact description.

Textual representation.

Constructs representing commands and references are made up from ASCII strings delimited either by **spacer sequences** or the two special characters **command escape** and **command delimiter**.

A contiguous sequence of any of the characters "tab", "carriage return", "line feed" or "space" is a **spacer sequence**. Spacer sequences delimit parameters within commands, and denote the **spacing** command elsewhere. Please notice that the length and actual choice of characters within the spacer are irrelevant.

Commands are prefixed by the command escape and delimited either by the command delimiter or a spacer. These special characters are user-selectable and thruout this description we will use the symbol "@" to denote the command and ";" for the command delimiter. A command may have one of the following two formats:

```
    command →@<command prefix>
    command →@<command prefix> <parameters> ;
```

that is, the command delimiter is used only when the command has parameters.

Numeric parameters may be interpreted as either decimal or hexadecimal values, the base chosen globally by the command **Set Base**.

Character indexes are represented by ascii characters and/or the special construct "@<number>". For instance the string

```
abA@3 4x
```

represents the sequence of character indexes 97 (for "a"), 98 ("b"), 65 ("A"), 3 (the construct "@3") and 120 ("x").

Spacers are ignored after commands, but they denote interword spacing between glyph references. For instance the sequence

```
@f3 Hello there,@f1   Joe
```

denotes a font change (to font number 3), then the word "Hello", then interword spacing, then "there," another change of font (to font 1) and the word "Joe" (notice that no interword spacing will appear between the "," and "Joe" since the spacer after "**@f1**" is ignored).

Compact encoding.

The compact encoding represents an imPRESS description as a sequence of eight bit bytes.

Commands consist of a **command prefix** and parameters, if any, follow in predefined order. A command prefix is a byte between 128 and 255.

Numeric parameters are represented as binary integers, packed in 8 or 16 bit sequences (a **byte** or a **doublebyte** respectively). When a signed quantity is specified, the integers are taken in two's complement representation.

Thruout this manual the notations $[\text{number}]_{byte}$ and $[\text{number}]_{2bytes}$ denote the 8 and 16 bit representations of the quantity number, respectively. A doublebyte sequence encodes the most significant byte first. The notation $[\text{string}]_{of bytes}^{sequence}$ denotes a sequence of bytes.

Character indexes are denoted by bytes between 0 and 127.

The Environment.

The **environment** consists of the following variables:

C_h, C_v: current position.
boxwidth, boxheight: box dimensions.
rotation: selects one of the four coordinate systems associated with the box.
Curfont: current font.
margin, bskip and *spacing*: for line oriented commands.
advancemode: selects automatic advance of the current position by the nominal dimensions.

At the beginning of an imPRESS description the environment is initialized in the following manner:

$$C_h \leftarrow 0$$
$$C_v \leftarrow 0$$
$$boxwidth, boxheight \leftarrow \text{page dimensions}$$
$$rotation \leftarrow \text{NW}$$
$$Curfont \leftarrow \text{fontid 0, (corresponds to the standard resident ASCII font)}$$
$$margin, bskip, spacing \leftarrow \text{suitable values for font 0}$$
$$advancemode \leftarrow \text{advance}$$

Whenever a **Set Box** or **Page** command is executed, a new environment is created as follows:

$$C_h \leftarrow 0$$
$$C_v \leftarrow 0$$
$$boxwidth, boxheight \leftarrow \text{box or page dimensions}$$
$$rotation \leftarrow \text{NW}$$

with the coordinate system changed as explained in the subsection *Boxes and Pages*. The net result is that the current position and orientation remain unchanged, but they are referred now to the new box coordinate system.

The environment is saved and restored from an **environment stack** by the following commands.

Command	**Save Environment**
Effect	Saves the current value of the environment variables on the environment stack. The values of the variables is not affected.
Textual Encoding	`@push`
Compact Encoding	$[128]_{byte}$

Command	**Restore Environment**
Effect	Pops the value of the environment variables from the environment stack. If the stack is empty, the variables are set to the initial values presented above.
Textual Encoding	`@pop`
Compact Encoding	$[129]_{byte}$

Example: The sequence

```
@f4 @p =200 =100;
@push
@f2 @h 200; This string is set in font 2
@pop
This one in font 4
```

sets the environment to font 4 and position <200,100>, then saves it and proceeds to modify it to font 2 and position <400,100> where it prints the string `This string is set in font 2` using font 2. Finally the environment is restored to its first value so that the string `This one in font 4` is printed in font 4 at position <200,100>. ▮

Character Reference.

In order to reference a character the appropriate font must be selected as current font, then the corresponding character index will set the character.

Command	**Set Font <fontid>**
Effect	Sets *Curfont* to *fontid*. ($0 \leq fontid \leq 255$)
Textual Encoding	`@f<fontid>` (No space between `@f` and *fontid*)
Compact Encoding	$[130+n]_{byte}$ for $0 \leq n \leq 31$ $[162]_{byte}\,[n]_{byte}$ for $32 \leq n \leq 255$.

The *fontid* must correspond to a declared font, otherwise the **Undefined Font** error condition is raised.

Notice that the compact encoding reserves 32 one byte commands for the most frequent font settings.

Command	**Reference Character <index>**
Effect	Prints the character *index* within the current font. Advances the current position according to *advancemode*.
Textual Encoding	Any ASCII character (excluding spacer characters and command prefix), or `@<numeric representation of` *index*`>`
Compact Encoding	$[index]_{byte}$ for $0 \leq index \leq 127$.

When using textual encoding, although non-printable ASCII characters will make the intended reference, it is advisable to substitute the construct `@<index>` to improve readability.

Font Declaration.

A font declaration associates a set of **properties** with a font. These properties refer to the font residence (resident or external), the character size (normal or big font) and the mask encoding (raster map or encoded masks).

Properties are encoded as strings of ASCII characters, with the following meaning:

s (small font): the maximum dimension of any of it characters is less than 127 (absolute value).

b (big font): At least one dimension of one of it characters is greater than 127 (absolute value).

e (encoded masks): glyph masks are encoded.

r (raster map masks): glyph masks are represented as raster maps.

Command	**Font <fontid> <properties> <fname>**
Effect	Associates the font identifier *fontid* with the font name *fname* and the property list *properties*. If *fname* is null then the font is assumed to be external, else it is associated with the resident font identified by *fname*. Font identifiers range $0 \leq fontid \leq 255$.
Textual Encoding	`@font <fontid> <properties> <fname> ;`
Compact Encoding	$[163]_{byte}$ $[\text{fontid}]_{byte}$ $[\text{properties}]^{sequence}_{of bytes}$ $[\text{“,”}]_{byte}$ $[\text{fname}]^{sequence}_{of bytes}$ $[\text{“;”}]_{byte}$

The font name identifies a resident font. In general the font name is the concatenation of an abbreviation of the typographic style of the font and its nominal size.

Example: The commands

```
    @font 1 se courier/10;
    @font 7 sr;
    @font 4 be logos;
```

declare font 1 to be a resident font (size small, glyph masks encoded) named **courier/10** (most probably a version of the *courier* typeface, in 10 point size). Font 7 is external, small and its masks are represented as raster maps. Finally font 4 is a big font, resident and encoded, named **logos** (a font containing digitized logos?). ∎

For more details about fonts please refer to "IMAGEN FONT MANUAL" (IMAGEN publication FONT/MAN)

Changing position.

A change quantity is either an increment or an absolute coordinate. Both affect the current position in the obvious way.

Both coordinates and dimensions are measured in pixels and are signed quantities.

In textual representation operands representing changes to the current position are denoted as follows:

absolute changes $\rightarrow =$ <number>

relative changes $\rightarrow$ <number>

In compact representation, a change of magnitude c is represented as:

$$\textbf{absolute change}(c) = 2.c$$
$$\textbf{relative change}(c) = 2.c + 1$$

that is, the magnitude is shifted left one bit, and a least significant is added to identify whether the change is relative or absolute.

Command	Set Position <horizontal change> <vertical change>
Effect	Alters the current position according to the changes.
Textual Encoding	`@p <horizontal change> <vertical change> ;`
Compact Encoding	$[164]_{byte}$ $[[\text{horizontal change}]]_{2bytes}$ $[[\text{vertical change}]]_{2bytes}$

Command	Set Horizontal Position <horizontal change>
Effect	Alters C_h according to *horizontal change*
Textual Encoding	`@h <horizontal change> ;`
Compact Encoding	If the change is a small relative change: $\lvert increment \rvert < 127$ then $[165]_{byte}$ $[\text{increment}]_{byte}$ otherwise: $[166]_{byte}$ $[[\text{horizontal change}]]_{2bytes}$

Command	Set Vertical Position <vertical change>
Effect	Alters C_v according to *vertical change*
Textual Encoding	`@v <vertical change> ;`
Compact Encoding	If the change is a small relative change: $\lvert increment \rvert < 127$ then $[167]_{byte}$ $[\text{increment}]_{byte}$ otherwise: $[168]_{byte}$ $[[\text{vertical change}]]_{2bytes}$

Example: The following commented sequence shows the changes in the current position:

```
@p =100 =150;        C =< 100,200 >
@p =19 20;           C =<  19,170 >
@h 91;               C =< 110,170 >
@v =2500;            C =< 110,2500 >
@v -1000;            C =< 110,1500 >
```

∎

Advance policy.

The following two commands switch the value of *advancemode.*

Command	Advance
Effect	Sets the advance policy to advance the nominal dimensions of the figures to be printed. *advancemode* ← advance
Textual Encoding	@adv
Compact Encoding	$[169]_{byte}$

Command	No Advance
Effect	The current position is not altered by the printing of figures. *advancemode* ← noadvance
Textual Encoding	@noadv
Compact Encoding	$[170]_{byte}$

Line Oriented Commands.

For typical printing jobs these commands are intended to act as shorthand for usual sequences of instructions.

The line parameters are:

margin: starting horizontal coordinate for a line.

spacing: usual distance between words.

baselineskip: usual distance between baselines.

(See illustration 4).

Two commands set the line parameters:

Command	Set Spacing <spacesize>
Effect	Sets *spacing* ← *spacesize*
Textual Encoding	`@sp <spacesize> ;`
Compact Encoding	$[171]_{byte}$ $[\text{spacesize}]_{byte}$

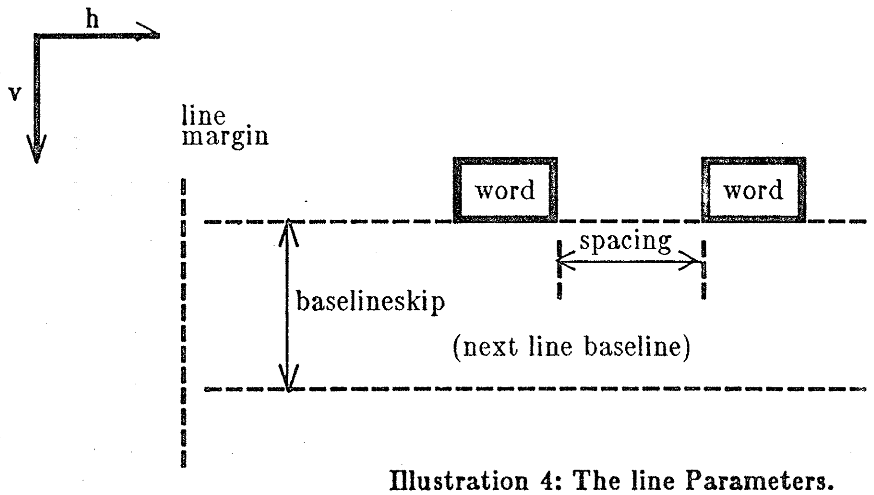

Illustration 4: The line Parameters.

Command	**Set Line Parameters <newbskip> <newmargin>**
Effect	Sets *bskip* ← *newbskip* and *margin* ← *newmargin*.
Textual Encoding	`@line <newbskip> <newmargin> ;`
Compact Encoding	$[172]_{byte}$ $[\text{newbskip}]_{byte}$ $[\text{newmargin}]_{2bytes}$

Two commands change the current position according to the line parameters:

Command	**Space**
Effect	Advance in the horizontal direction by *spacing*: $C_h \leftarrow C_h + spacing.$
Textual Encoding	A spacer sequence, or `@<spacer sequence>`
Compact Encoding	$[173]_{byte}$

Command	Newline
Effect	Increment the vertical coordinate by *bskip*: $C_v \leftarrow C_v + bskip$, and set the horizontal one to *margin*: $C_h \leftarrow margin$.
Textual Encoding	`@n`
Compact Encoding	$[174]_{byte}$

Example: The following sequence

```
@line 30 20; @sp 11;
@p =40 =200;
(1) The first line of the paragraph@n
(2) the second line@n
@v 5;
(3) This line is 5 units lower that the usual baseline skip.
(4) here @f3 the font changes@f2@ and changes again
```

shows a typical paragraph set using the line oriented commands. The baseline distance is 30 pixels except between lines (2) and (3) where it has been incremented by 5. The margin is set at $h = 20$. Line (4) shows an instance of the `@<spacer>` command used to force spacing after the `@f2` command. ▮

Rules.

A rule is a black rectangle of height h and width w and a **vertical offset** *voff*. A rule has a nominal width equal to *width*.

Command	Rule <height> <width> <voff>
Effect	Prints a rule of the given dimensions.
Textual Encoding	`@ru <height> <width> <voff>;`
Compact Encoding	$[175]_{byte}$ $[\text{height}]_{byte}$ $[\text{width}]_{byte}$ $[\text{voff}]_{byte}$ for small rules, or $[176]_{byte}$ $[\text{height}]_{2bytes}$ $[\text{width}]_{2bytes}$ $[\text{voff}]_{byte}$ for bigger rules.

Illustration 5 shows the positioning of different rules. Notice that the vertical offset allows for easy under/overlining of figures.

Rotation.

Assume that a box of height *ht* and width *wd* is set at the current position. This setting implicitly defines its four corners (and the four associated coordinate systems) as follows:

$$\begin{aligned}
\mathbf{NW} &= < C_h, C_v > \\
\mathbf{NE} &= < C_h + wd, C_v > \\
\mathbf{SE} &= < C_h + wd, C_v + ht > \\
\mathbf{SW} &= < C_h, C_v + ht >
\end{aligned}$$

The rotation command chooses the coordinate system in either of two ways:

absolute rotation: select NW, NE, SE or SW.

relative rotation: move the box in a clockwise or anticlockwise ninety degree rotation. (In clockwise sequence NE follows NW, SE follows NE, etc.)

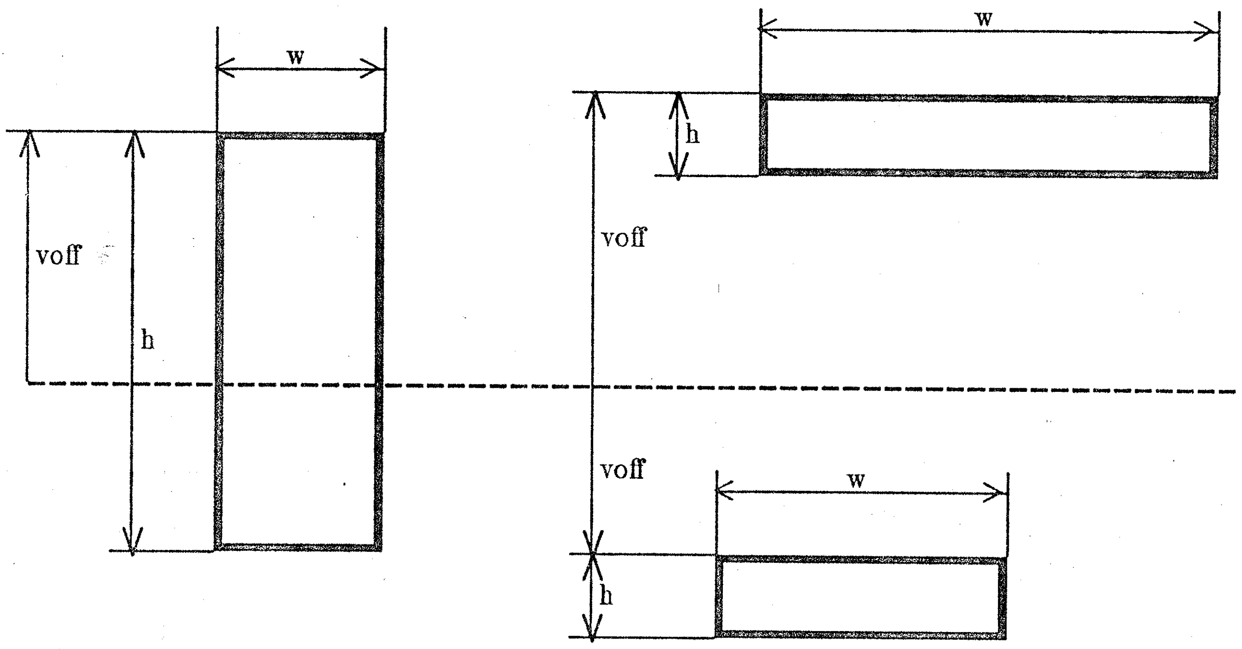

Illustration 5: Positioning of rules.

Command	Rotate <newrotation>
Effect	Chooses one of the four coordinate systems associated with the current box: $rotation \leftarrow$ newrotation $C_h, C_v \leftarrow$ transformed coordinate values
Textual Encoding	`@rot <newr> ;` where `<newr>` is "+" (clockwise), "-" (anticlockwise), or `NW`, `NE`, `SE` or `SW`.
Compact Encoding	$[177]_{byte}$ $[newr]_{byte}$ with $newr = 0, 1$ for clockwise, anticlockwise rotations $newr = 2, 3, 4, 5$ for rotate NW, NE, SE, SW.

The current position remains stationary with respect to the box, and thus its*coordinates values* change when a rotation occurs. For instance if the current position under rotation NW is $C = < h, v >$, within a box of dimensions wd and ht the following table shows the changes in C when the box is rotated:

$C_{NE} = < v, wd - h >$, (rotation to NE)
$C_{SW} = < ht - v, h >$, (rotation to SW)
$C_{SE} = < wd - h, ht - v >$, (rotation to SE)

Glyph downloading.

The **Glyph** command defines a new glyph that is added to the glyph table. The command parameters are:

glyph id: font *fontid*, character index *char*.

glyph dimensions: width *w*, height *h*, depth *d* and nominal width *nw* (all of them in pixels)

mask: this parameter varies depending on the glyph encoding. Please refer to the IMAGEN FONT MANUAL for more details.

Command	Glyph <glyphid> <dimensions> <mask>
Effect	Adds the glyph **glyphid** to the Glyph Table.
Textual Encoding	`@gly <fontid> <char> <w> <h> <d> <nw> <mask> ;`
Compact Encoding	$[178]_{byte}$ $[fontid]_{byte}$ $[char]_{byte}$ $[w]_{byte}$ $[h]_{byte}$ $[d]_{byte}$ $[nw]_{byte}$ $[mask]^{sequence}_{of bytes}$ for small fonts. $[178]_{byte}$ $[fontid]_{byte}$ $[char]_{byte}$ $[w]_{2bytes}$ $[h]_{2bytes}$ $[d]_{2bytes}$ $[nw]_{2bytes}$ $[mask]^{sequence}_{of bytes}$ for big fonts.

Glyph Memory Control.

The Glyph Memory Area contains glyphs that have been referenced and/or downloaded from the host. Documents that require a large number of different glyphs must include control information for the imPRESS processor to delete unused glyphs in order to make room for incoming ones.

The following commands control the memory allocation in the glyph table.

Command	**Delete glyph <char>**
Effect	Deletes the glyph corresponding to the character index char (in the current font) from the glyph table.
Textual Encoding	`@del <char> ;`
Compact Encoding	$[179]_{byte}$ $[\text{char}]_{byte}$

Command	**Forget font <fontid>**
Effect	Deletes all the glyphs for the font, and releases the *fontid* to be used in another font declaration.
Textual Encoding	`@dfnt <fontid> ;`
Compact Encoding	$[180]_{byte}$ $[\text{fontid}]_{byte}$

Box Definition and Replication.

A box has a **box name**, **box dimensions** (height and width), and a **box body** (the sequence of imPRESS commands that describe the box).

A box is defined by issuing a **Define Box** command, then the box body and finally an **End Definition** command.

Box definitions are kept thruout the whole imPRESS description. Box definitions are not allowed to nest.

Command	**Define Box <name> <dimensions>**
Effect	Begin the definition of the box <name>
Textual Encoding	`@Box <name> <width> <height> ;`
Compact Encoding	$[181]_{byte}$ $[\text{name}]_{of bytes}^{sequence}$ $[\text{“,”}]_{byte}$ $[\text{width}]_{2bytes}$ $[\text{height}]_{2bytes}$

Command	End Box <name>
Effect	The box <name> is defined. (The name must match the previous **Define Box** command.)
Textual Encoding	`@end <name> ;`
Compact Encoding	$[182]_{byte}$ $[\text{name}]^{sequence}_{of bytes}$ $[\text{";"}]_{byte}$

When a box is replicated with a **Set Box** command the following process takes place:
(i) A new environment is created:

$$\begin{aligned} C_h &\leftarrow 0 \\ C_v &\leftarrow 0 \\ boxwidth, boxheight &\leftarrow \text{box dimensions} \\ rotation &\leftarrow \text{NW} \\ Curfont &\leftarrow \text{previous curfont} \\ margin, bskip, spacing &\leftarrow \text{previous values} \\ advancemode &\leftarrow \text{previous value} \end{aligned}$$

(ii) The commands in the box body are executed.
(iii) Once the commands have been executed the initial environment is restored and the current position is modified according to *advancemode*.

Command	Set Box <name>
Effect	The box name is set at the current position.
Textual Encoding	`@set <name> ;`
Compact Encoding	$[183]_{byte}$ $[\text{name}]^{sequence}_{of bytes}$ $[\text{";"}]_{byte}$

Pages

The command **Pages** is essentially a box definition (it must be matched by an **End Definition** command).

The imPRESS processor processes the body of commands associated with the page and submits the result to the IMPRINT-10 Printing Subsystem, which produces the output

The processing of a page consists essentially of:

- Final positioning of individual glyphs and graphic primitives on the page.

- Checking for glyph availability: all the glyphs needed to print the page must be available (else an **Undefined Glyph** is raised).
- Rotating the glyphs according to their final orientation.

Command	**Page <pageid> <dimensions>**
Effect	A page description is processed and submitted to the IMPRINT-10 Printing Subsystem.
Textual Encoding	`@Page <name> <width> <height> ;`
Compact Encoding	$[184]_{byte}$ $[name]^{sequence}_{of bytes}$ $[\text{“,”}]_{byte}$ $[width]_{2bytes}$ $[height]_{2bytes}$

Textual Encoding Only Commands.

These commands set parameters that control the translation of textual descriptions, in particular the **number representation base** and the **command escape** and **command delimiter** characters.

Command	**Set Base <newbase>**
Effect	Sets the numeric representation base.
Textual Encoding	`@base <newbase> ;` If <newbase>="d" then numeric strings will be interpreted as decimal, otherwise they are assumed to be hexadecimal.
Compact Encoding	not applicable.

Command	**Set Escapes <esc> <delimiter>**
Effect	Selects the pair of characters for command delimiting.
Textual Encoding	`@esc <escchar> <delimiter> ;`
Compact Encoding	not applicable.

Example: The sequence

```
@base d;
@p =10 =20;
@esc [];
[base h]
[p =FF =10]
```

first declares base 10, and thus the current position is set to <10,20>. Then it changes the command delimiters to "[" and "]". The final position command sets the current position to <255,16>. ■

Appendix A: ERROR CONDITIONS.

A given imPRESS representation might cause certain error conditions. In general the imPRESS processor takes a standard default action and proceeds.

Glyph Memory Overflow:

Cause: the glyph table does not have enough room for the glyph mask.

Action: the glyph is ignored and any further reference to it will raise the same error condition.

Position Overflow:

Cause: a glyph or a graphic primitive is positioned out of the page boundaries.

Action: the glyph or graphic primitive is not printed (though the current position is advanced accordingly).

Undefined Glyph:

Cause: a glyph that has not been defined is referenced.

Action: a special "missing mark" glyph is substituted for the missing glyph.

Undefined Font:

Cause: a font that has not been declared is referenced.

Action: the font reference is ignored.

Environment Stack Overflow:

Cause: the number of **@push** commands exceeds the number of **@pop** commands by more than *MXENVIRON* (an imPRESS processor parameter)

Action: the command is ignored (no save occurs).