TrueImage

CHAPTER 6

6.1 Introduction

Over the last five years technology has become available that allows personal computer users to produce high-quality printed output using nothing more than a home computer, commercial software, and a laser printer. The phrase "desktop publishing" has been coined, describing the facility with which individuals can turn out professional-looking documents comprising both text and graphics. Previously, quality document production was a cumbersome process involving traditional mechanical printing methods, and printing jobs could only be carried out by professional printers possessing the necessary training and equipment.

A significant factor in this computer-inspired revolution has been the PostScript page description language (PDL) written by Adobe Systems Inc. PostScript is a means by which the design and contents of a page or sequence of pages specified on a computer can be rendered by a laser printer (or any other high-quality printing device, such as a Linotron).

TrueImage is a page description language developed by MicroSoft Corporation that is based on, and designed to be compatible with PostScript.

As well as PostScript compatibility TrueImage also incorporates the TrueType font technology introduced by Apple Computer Inc. as part of the Macintosh System 7 operating system, and incorporated by MicroSoft into version 3.1 of the Windows operating environment for IBM PCs. Like PostScript fonts, TrueType fonts allow printer text output at any size.

To most users, TrueImage is simply a term denoting high-quality laser-printed output; users compose documents on their computers (text, graphics, tables or any combination) and print them out on their TrueImage printer. The printer output is a faithful reproduction of their on-screen documents.

In fact, TrueImage is a computer language that can be used to describe printed output. When a user creates a document using application software, such as a word-processor or illustration program, he specifies the appear-

ance of the document on screen using the computer's mouse and keyboard. When he prints his document the application automatically converts the document to a TrueImage page description program which is then sent via cable to the printer. The TrueImage program is a sequence of commands that are executed in order. A TrueImage program file (which might be generated by first printing the document to disk, instead of to the printer) is simply a text file containing the commands, which can be viewed and edited using a text editor.
The printer contains a TrueImage interpreter, a program that executes the in-coming TrueImage program commands, constructing and printing each page of the computer-based document.

A TrueImage program is not unlike a program in C, BASIC or any other language. However, the key difference is that TrueImage programs are, in most cases, generated by an application, not by a human programmer.

The commands that make up the TrueImage language are known as operators. There are over 250 different operators offering a wide range of programming facilities.

TrueImage page description programs are typically of the following format: a short header block containing general information about the document, a prologue section in which procedures are defined (e.g. a procedure to draw a commonly-occurring shape) and set-up procedures applicable to the whole document are performed, and then sections describing each individual page separately. For example, the program sent to the printer when a ten-page file mixing text and graphics produced using a page-layout program is printed might well contain a short header with comments about the application and the file, a prologue section defining a procedure to draw squares and output elements common to every page such as a logo, followed by sections of code individually describing each page in the document.

In short, applications send data to the printer in the form of executable TrueImage programs. To most users this is transparent, however, application developers need to know the commands and structure of the TrueImage language, in order to make their software generate appropriate output. Also, in certain cases it is useful for users to be able to modify printer output by first generating a TrueImage program file (by printing the document to disk) and then editing it before sending it to the printer.

6.1.1 TrueImage output on different printers

Printers lay down an image on the page as a matrix of tiny dots. The greater the number of dots per unit area of the page, the higher the quality of the final image. Typically home or office laser printers have a resolution of (300×300) dots per square inch. Higher-quality output devices, such as Linotrons, typically have a resolution of (2400×2400) dots per inch.

TrueImage page descriptions describe output in terms of geometric shapes defined numerically in terms of coordinates, not as a matrix of dots. The printer itself converts the TrueImage code to a dot matrix, performing a process known as scan conversion. Hence TrueImage is device independent, in that the quality of the print-out (the smoothness of curves, appearance of gray scales etc.) is limited only by the printing device's own dots-per-inch resolution, not by any inherent limitation in the TrueImage language.

6.2 TrueImage print model

The following model is used to describe the way in which TrueImage output is built up by the printer. The image on a page is constructed by placing paint on the page in selected areas. The painted areas can form any shape: characters, geometric shapes, lines, shaded areas. The paint can be black, white, gray or colored. Output can be restricted (clipped) to any area within the page. When a page has been fully constructed it is printed out.

A print job may consist of any number of pages. Each page starts as completely white. TrueImage constructs and outputs each page in turn, working to completion on each individual page, before commencing the next one. The page which TrueImage is constructing at any given time is known as the current page. When the current page is complete the **showpage** operator is used to print it out.

Paint marks of any color are always opaque and obscure any previously laid down marks which they overlap. Hence the order in which elements of a page are painted onto the page determines which are wholly visible and which are wholly or partly obscured.

Paint operators paint each element onto the current page. The principle paint operators are **fill**, **stroke**, **show** and **image**. **fill** fills an area, **stroke** draws a line, **show** displays text characters and **image** renders an imported sampled image, for example a scanned-in photograph.

Most paint operators function with reference to the **current path**. A **path** is a sequence of connected and disconnected points, lines and curves that define a shape and its position on the page. Path construction operators such as **newpath**, **moveto**, **lineto**, **curveto** and **arc**, are used to build up the **current path**. These operators do not mark the page, they merely define a shape and a position on the page that the paint operators can work with. For example, **fill** fills the current path and **show** outputs text starting at the endpoint of the current path.

A **subpath** is a series of connected line segments (i.e. defined by operators other than **moveto** and **rmoveto**). A path consists of one or more subpaths. A subpath may be closed by the **closepath** operator, which joins a subpath's endpoint to its starting point.

A collection of settings known as the **graphics state** determine the way in which path construction and paint operators are interpreted and hence the appearance of printed output. Graphics state settings include parameters such as the current path, line thickness, line pattern and current font. The graphics state is described fully in a following section starting on page 199. Many operators change the graphics state when they are executed. Two operators, **gsave** and **grestore**, are provided to save and restore the current state, enabling a TrueImage program to revert to a particular known state at any time. For example, the **stroke** and **fill** operators both reset the current path to empty when they are executed. To stroke and fill a path, the following sequence of operators would be executed:

Path construction operators defining the path
gsave - to save the graphics state containing the defined path
stroke - to stroke the path
grestore - to restore the saved state and the old current path
fill - to fill the path

The current **clipping path** defines the area of the page to which output is confined. The clipping path, which is part of the graphics state, can be arbitrarily complex.

A TrueImage page description normally consists of many operator calls. The recurring pattern of operation is as follows:

Lay down a path using path construction operators.
Modify as necessary any graphics state settings, such as line-width.
Paint the path using paint operators.

6.3 Coordinate systems

TrueImage defines an ideal coordinate system, known as **user space**. All TrueImage operations are defined in terms of user space coordinates. The default user space origin is in the bottom left-hand corner of the page, and its x- and y-axis units are 1/72".

The coordinate system that the printer uses to construct its output is known as device space. User space and device space are completely independent of one another. The TrueImage interpreter automatically maps user space to device space when it executes a TrueImage page description.

The TrueImage interpreter maps user space to device space by maintaining a current transformation matrix (CTM). Multiplying user space coordinates by the CTM yields the corresponding device space coordinates. The CTM is part of the graphics state (see next section).

Transposition operators, such as the **translate**, **rotate** and **scale** operators, change the relationship between user and device space by modifying the CTM, enabling page output and individual graphic elements to be positioned. For example, the area of a page on which a laser printer can place output is normally less than the whole page; there is usually a small boundary around the outside of the page which cannot be painted. It is often useful to make the user space origin map to a corner of this imageable area. Also the desired rotation and scaling of output is subject to change, as users may wish to print landscape pages or thumbnail miniature pages.

Since the CTM is part of the graphics state, a useful programming technique is to use transposition operators in combination with **gsave** and **grestore** to transpose a single graphic element. For example, a text string may be printed in several different orientations by enclosing the **rotate** and **show** commands within successive **gsave**, **grestore** pairs. Each coordinate rotation is only current when the string is printed. All other page elements are unaffected by the rotation.

In fact, it is more convenient to think of the transposition operators as transposing user coordinate space relative to its default origin, unit size and orientation, and this is the convention we shall adopt in this chapter.

6.4 Graphics state

The TrueImage interpreter maintains a collection of settings known as the graphics state. These settings define the actual appearance of output generated when TrueImage operators are executed. Some operators change the graphics state either directly or as a side effect to their main function. For example, the **setlinewidth** operator sets the width of lines, and the **fill** operator, in addition to filling the current path, also resets the current path to empty. Graphics states can be stored and retrieved; they are stored on the graphics state stack. Stack operation is explained in the following section. The parameters that make up the graphics state are as follows. Further explanation will be found in the relevant operator and operator category descriptions.

Parameter	Value	Default (if any)	Operators directly affecting the parameter
CTM	Current transformation matrix defining the mapping from user space coordinates to device space coordinate	Matrix mapping default user space to device space	**translate** **rotate** **scale**
color	The painting color	Black	**sethbscolor** **setrgbcolor**
position	Current position in user space	Undefined	Path construction operators
path	Current path as defined by path construction operators	Empty	Path construction operators
clipping path	Path defining a boundary to which output is clipped	Imageable area of page	**clip** **eoclip**
font	Currently selected font		**setfont**
line width	The thickness of lines in user coordinate units	1	**setlinewidth**
line cap	Line end shape	Butt end	**setlinecap**
line join	Line join shape	Mitered	**setlinejoin**
halftone screen	Gray scale setting or color intensity		**setscreen**
transfer	Mapping of user gray scales to device gray scales		**settransfer**
flatness	Smoothness of curved segments		**setflat**
miter limit	Maximum length of a mitered line join	10	**setmiterlimit**
dash pattern	Pattern used for drawing lines	Solid line	**setdash**
device	Current output device		

6.5 TrueImage language features

6.5.1 Program execution

The TrueImage interpreter receives a TrueImage page description as a sequence of objects which it executes in turn. The page description is received as a stream of characters which the interpreter scans, looking for tokens (short character sequences) that define objects. Objects may be data (numbers, booleans strings and arrays) or program elements (names, operators and procedures). What execution of a particular object actually entails, depends upon the object's type. Objects are processed using a data structure known as the **operand stack**. This is described below.

6.5.2 Regular and special characters

Any printable characters in the ASCII character set may be used in TrueImage programs, plus the whitespace characters (space, tab and newline). The following special characters have particular meaning within a program: (,), <, >, [,], {, }, / and %. Their significance is explained in the following sections. Characters other than printable ASCII and whitespace characters may be used in a page description, however, their use is not recommended since the results of their use are not always predictable. Any characters in a program that do not belong to the group of special characters are referred to as regular characters.

6.5.3 Comments

Comments in a TrueImage page description are preceded by a % character. When the interpreter encounters a %, it ignores all characters up to the next newline character, after which it resumes scanning the in-coming character stream for recognizable TrueImage objects.

6.5.4 TrueImage objects

TrueImage objects may be any of the following types:

integer	dictionary
real	operator
boolean	file
array	mark
packedarray	null
string	save
name	fontID

integer - Decimal integers are represented by a string of digits, which may have a sign, e.g. 100, –75 +10. Integers may also be specified in other bases in the form *base#number*: e.g. a binary number might be specified as 2#10011, an octal number as 8#76767 or a hexadecimal number as 16#DEF1. Digits greater than 9 are represented by the letters A – F, or a – f. Non-decimal numbers cannot be signed.

real - Real numbers are represented by an optional sign followed by a string of digits, which may optionally contain a decimal point, an exponent, or both. An exponent is represented by the character E or e followed by an optional sign and one or more digits. e.g. –0.2, 38.4, –4.9, 45.7e9, 2E–5

boolean - A boolean is either *true* or *false.*

array - An array is a one-dimensional collection of objects that can be regarded as a single entity. The individual objects within the array need not be of the same type and can be of any TrueImage object type. Hence an array could contain an integer, a real and a boolean. An array appears in a TrueImage program enclosed in square brackets e.g. [24 32.6 *true*] An executable array (also known as a procedure) is a special type of array whose objects can be executed in sequence. An executable array appears in a TrueImage program enclosed in curly brackets e.g. { **add** 4 **mul** }

packedarray - A packed array is simply a more compact representation of an executable array. Packed arrays are read-only.

string - A string is stored as a list of integer character codes in the range 0 – 255. A string appears in a TrueImage program enclosed within brackets e.g. (This is a string). Within a string the \ character is used to escape special characters and non-printing characters.

\n	linefeed (newline)	\(	open bracket
\r	carriage return	\)	close bracket
\t	tab	\ddd	octal character code *ddd* - used to specify a character outside the standard character set.
\b	backspace		
\f	form feed	\newline	end of line (without the *newline* character becoming part of the string)
\\	backslash		

Alternatively a string may appear as a sequence of hexadecimal code pairs enclosed in angle brackets e.g. <6D657C>. If the final character is missing it is assumed to be 0. Whitespace characters in a hexadecimal string are ignored.

name - A name can be any string of regular (non-special) characters that cannot be interpreted as a number. Names stand for variables. Variables can be of the following types: integer, real, boolean, array, packed array, string, dictionary, file or fontID. As the interpreter encounters a name it will attempt to execute it. The meaning of execution for different types of object is described in the section entitled Execution. A name immediately preceded by a / or // is treated differently by the interpreter. This is also described in the section entitled Execution.

dictionary - A dictionary is a table of key-value pairs. The keys in a dictionary are normally names, though the string equivalent of a name may also be used. TrueImage dictionary operators allow you to create dictionaries, insert key-value pairs into dictionaries, look up values in a dictionary by key, and perform various other operations. TrueImage automatically maintains a **userdict** which normally contains the current program's name and procedure definitions, and a **systemdict**, in which the actions associated with operators are looked up. **errordict** is a dictionary listing error names and associated error-handling procedures. Dictionaries are manipulated using the dictionary stack. See the section on stacks on page 204. TrueImage fonts are also dictionaries in which the keys are character names and the values procedures for rendering the characters' shapes.

operator - An operator is one of TrueImage's built-in commands, such as **add** or **fill**. Operators are identified by name. When the interpreter encounters an operator object, it looks up the associated action and performs it. The user is free to redefine the actions associated with any TrueImage operator name.

file - A file is a readable or writable sequence of characters. TrueImage file operators can be used to create and manipulate file objects. TrueImage provides two standard files: the standard input and standard output file. The standard input file is normally the source of the page description program being executed, the standard output file is the destination for the interpreter's error and status messages.

mark - A mark object is used as a place-holder in the stack. Array and stack operators make use of the mark.

null - The interpreter uses null objects to fill uninitialized positions in composite objects such as arrays or dictionaries, when they are created.

save - A save object is a snapshot of TrueImage's memory. Save objects are used by the **save** and **restore** operators.

fontID - A fontID is a unique font identifier, inserted as a value in a font dictionary.

Arrays, strings and dictionaries are known as composite objects. When copies of these types of object are made, the copies share data with the original. When any other kind of object is copied, a separate copy of its value is made.

6.5.5 Stacks

A stack is a data structure onto which the interpreter places (or pushes) objects and from which it removes (or pops) objects. At any given time only the topmost objects on the stack can be accessed. TrueImage operators pass objects between one another using the **operand stack**. An example using simple arithmetic will serve to demonstrate the principle. Suppose that the stack contains several objects, the top two being the integer objects 14 and 23.

14
23
(A string)
14.2
[1 2 3]

If the TrueImage interpreter next encounters the operator **add**, it removes the top two items, adds them and puts their sum back on top of the stack.

37
(A string)
14.2
[1 2 3]

Now suppose that it is required to multiply the top object, 37, by the third object, the real number 14.2. The operator **mul** will multiply two numbers together, however, like **add** it can only use the top two stack elements. At this point direct stack manipulation comes in useful. The **roll** operator rotates objects on the top of the stack, in preparation for other operators to use. **roll** needs two parameters which must themselves be taken from the stack. The program sequence **3 –1 roll** first causes the interpreter to push the two parameters onto the stack.

−1
3
37
(A string)
14.2
[1 2 3]

then the **roll** operator immediately removes them,

37
(A string)
14.2
[1 2 3]

and rotates the three topmost elements into the new order shown. The values 3 and −1 instruct the **roll** operator to rotate the top three elements, bringing the third element to the top, and moving the other two down one position.,

14.2
37
(A string)
[1 2 3]

Now the two numbers occupy the top two stack positions. If the interpreter now receives a **mul** operator, the top two objects are multiplied and their product placed on the stack.

525.4
(A string)
[1 2 3]

The result, 525.4, is now available to any other operator that reads a number from the top of the operand stack. All TrueImage operator activity can be described in terms of the operand stack.

In addition to the operand stack the TrueImage interpreter maintains three other stacks: the **dictionary stack**, the **execution stack** and the **graphics state stack**.

The dictionary stack holds dictionaries that define the values associated with names and the actions performed when procedures (executable arrays) are called.

The execution stack holds the object (procedure or file) currently being executed and all partially executed procedures and files that have been put on hold while the interpreter executes a more recently encountered executable object. The topmost object is the one currently being executed. When execution of the topmost object is complete, the object is popped off the top of the stack.

The graphics state stack holds graphics states saved with the **gsave** operator. Graphics states are popped from the stack, and made current by the **grestore** operator. In keeping with the characteristic of the stack data structure, graphics states can only be restored in the reverse order to that in which they were saved.

The four stacks are completely independent from one another. The operand stack is under the control of TrueImage programs whose operators can push and pop objects freely. Some dictionary operators can be used to manipulate the dictionary stack, however, the two TrueImage-maintained dictionaries **userdict** and **systemdict** cannot be popped. The execution stack is completely controlled by the interpreter. The graphics state stack is maintained by the interpreter in response to the various graphics state, **gsave** and **grestore** operators encountered.
In this chapter references to "the stack" refer to the operand stack.

6.5.6 Syntax

The syntax of TrueImage programs is rather unusual. It differs from that of most other programming languages, the notable exception being FORTH.

The difference is that in TrueImage programs commands (operators) are preceded by their parameters (operands). Hence a typical TrueImage program fragment might be as follows:

```
2 3 add % add 2 & 3
5 mul % multiply result of 2x3 by 5
100 100 moveto % move to coordinate position (100,100)
200 200 lineto % draw a line from (100,100) to (200,200)
```

This rather strange looking order is used because of the way in which the TrueImage interpreter processes in-coming programs. On receiving a number object, the interpreter pushes it onto the stack. On receiving an operator object the interpreter executes the operator using the numbers on the top of the stack as operands (parameters). Hence, the operands always precede the operator in the programs, so that the interpreter receives them first.

6.5.7 Execution of objects

When the TrueImage interpreter receives an object (number, array, name etc.) it attempts to execute it, unless the program syntax specifies otherwise. The meaning of execution for each of the valid object types is summarized below.

integer	The number is pushed onto the stack.
real	The number is pushed onto the stack.
boolean	The boolean value (true or false) is pushed onto the stack.
array	An array enclosed in [] brackets (a data array) is pushed onto the stack. An array enclosed in { } brackets (a procedure) is pushed onto the stack if it is encountered directly by the interpreter as part of the in-coming program stream. However, if the interpreter encounters the procedure indirectly, i.e. by looking up a name or operator in a dictionary, the interpreter executes each of the objects in the array in turn.
packed array	A packed array is pushed onto the stack if it is encountered directly by the interpreter as part of the in-coming program stream. However, if the interpreter encounters the procedure indirectly, i.e. by looking up a name or operator in a dictionary, the interpreter executes each of the objects in the packed array in turn
string	A string constant enclosed in () brackets is pushed onto the stack. A string that has been made executable is pushed onto the execution stack and the interpreter scans through it, executing in turn each of the objects that it encounters.
name	The name is used as a key and is looked up in the current dictionary. The value associated with the key is executed. This value will also be an object of some kind.
dictionary	The dictionary is pushed onto the stack.
operator	The operator is executed. The actions associated with each operator are described in the Operator section of this chapter.
file	The file is pushed onto the execution stack and the interpreter scans through it, executing in turn each of the objects that it encounters.
mark	The mark is pushed onto the stack.
null	No action is performed.
save	The save is pushed onto the stack.
fontID	The fontID is pushed onto the stack.

Sometimes it is desirable to inhibit the execution of an object. For example, to associate a name with a value, the operator **def** is used. **def** takes two operands, the name and the value, which it reads from the operand stack. Suppose we want to associate the name *myvariable* with the value 5, equivalent to *myvariable = 5* in a conventional programming language. The program line

```
myvariable 5 def
```

will not work since the interpreter will attempt to execute the name *myvariable* by trying to look up an associated value. To suppress execution of an object we can precede it with a /. Any object that the interpreter encounters with a / before it is simply pushed onto the stack. Note that for some objects execution entails pushing them onto the stack in any case, hence / is never needed.

The program line

```
/myvariable 5 def
```

accomplishes the task of setting *myvariable* to 5.
There are cases where we may want the value of a name to be substituted for the name itself. Preceding a name by // achieves this. When the interpreter encounters a name preceded by // it immediately looks up the current value of the name and replaces the name with the value. This process is simply a substitution; the value is not executed. The purpose of this feature is to allow programs to force the current value of a particular object to be used in a procedure.

6.5.8 Executable and access attributes of objects

Objects may explicitly be made literal (non-executable), or executable, using the **cvlit** and **cvx** operators. Objects with the literal attribute are simply pushed onto the stack; those that are executable are looked up and executed. Objects may also be assigned an access attribute, either *unlimited, read only, execute only* or *no access.* These specify how TrueImage operators may or may not manipulate them.

6.5.9 Errors

TrueImage operators can generate errors for a number of reasons. On encountering an error the interpreter restores the stack to the state it was in when execution of the current object began, pushes the object onto the stack, looks up the error name in **errordict**, and executes the associated procedure. Default error procedures normally involve terminating the current program and writing an error message to the standard output file.

TrueImage programs may modify **errordict**, defining new error-handling procedures for given error names.

The possible errors are described in the Errors section on page 290. Each of the possible errors that an operator can generate is listed under the operator's description.

6.5.10 Virtual memory

Virtual memory is the name given to the storage area where the values of TrueImage composite objects (arrays, dictionaries and strings) are held. A pair of operators, **save** and **restore**, allow programs to save the state of the virtual memory and restore it again at a later juncture. It is good practice to encapsulate each separate page of a TrueImage page description program within a **save**, **restore** pair. This has the effects of freeing up virtual memory consumed by the pages as they are executed, and restoring the initial set of conditions established by the program's prologue section.

If you are using Legal-sized paper, less printer memory is available for use as virtual memory. With the standard memory configuration, a **VMerror** will be generated when the printer attempts to print. If you intend to use Legal-sized paper, ensure that you install an additional 2MB of RAM at least.

6.6 Fonts

Since the majority of printing work involves the production of text, TrueImage is geared to support text and font handling at all levels. The printer includes 35 built-in TrueType fonts, which are available for use at any time. These fonts are listed in Chapter 7, the Technical Supplement. Additional commercial TrueType fonts may be downloaded from the host computer. In addition to supporting TrueType fonts, TrueImage can also use PostScript type 1 fonts and type 3 (user-defined) fonts. For a general discussion of fonts and related issues, refer to chapter 3 of this manual.

Typically a TrueImage program may simply select fonts for printing, selecting a built-in typeface and weight, and sizing it as required. Procedures may be defined to select frequently-used fonts. On occasion, a different character set may be required; this can be achieved using TrueImage operators. If need be, a TrueImage program may even be used to define a font.

TrueType fonts are comprised of characters: each character is defined as a graphical shape that can be rendered on the page. A TrueType font is a dictionary that contains various information. Most importantly, the dictionary contains the names of every character in the font, and for each name, a corresponding procedure for drawing the character. It also contains another dictionary which associates character code numbers with character names.

TrueImage renders text using a collection of operators that take a string as an operand and print it on the page at the current position. A TrueImage string consists of a sequence of characters: each character represented by an integer character code in the range 0 – 255. TrueImage maps each code to a corresponding name, and then executes the procedure corresponding to that name to render the character. The correspondence between codes and character shapes can be changed by changing the vector which defines how codes correspond to character names.

Font operators prepare and select fonts for printing. A typical sequence is as follows:

```
/Arial findfont
20 scalefont setfont
100 100 moveto
(This is a text message) show
```

This is a text message

findfont puts the Arial font dictionary on the stack. **scalefont** takes the dictionary and creates a copy in which the characters are scaled by the specified factor in user units. In this case a font whose size is 20 user space units is created. Notice that size is defined in terms of user space units, not in typographic points. (The **makefont** operator can be used to scale a font by different factors in the x- and y-directions, and to rotate and translate it). **setfont** makes the font left on the stack by **scalefont** the current font. **show** then prints the string "This is a text message", using the selected font and starting from the point (100,100). The **moveto** is necessary since the current position must be known before a string can be printed. Each character in a font has a certain width. Ordinarily printing a character updates the current position by the character's width.

To associate a name with a scaled font (or any other modified copy of a font), the **definefont** operator is used. The new font may then be selected by a unique name and need not be rescaled each time.

Effects can be applied to characters, for example they may be printed in color or in a selected gray scale. The outline shapes of characters may be appended to the current path using the **charpath** operator. This allows a variety of effects, such as the use of a string as a mask: only shapes enclosed within the character shapes may appear on the page. The following sample program demonstrates the use of this effect.

```
0 setgray
/Helvetica findfont 170 scalefont setfont
newpath 50 130 moveto
(JAPAN) true charpath
2 setlinewidth
clip
stroke
.5 setgray
newpath
300 200 moveto
300 200 40 0 360 arc
fill
.5 setgray
newpath
8 setlinewidth
0 10 360 {dup 5 add 300 200 300 4 index 3 index arc 300
200 lineto} for
stroke
```

JAPAN

6.6.1 Font caching

TrueImage renders characters by converting their shapes to a bitmap that can be displayed on the printer. To avoid performing this conversion for each single occurrence of a given character in a stream of text, TrueImage stores (caches) bitmap representations of characters that it has already calculated. This allows much faster printing.

This process is entirely automatic, however, there are four operators that allow explicit control of the font cache.

There is a maximum character size (in bytes) that is permitted for cached bitmap images. Characters exceeding this size are not cached. There is also a compression size limit. Characters small enough to be cached that exceed the compression size limit are cached and compressed. Compressed characters take up less space in the cache, but take longer to render, since they must first be decompressed every time. These limits may be adjusted using the font cache operators.

The font cache does not retain color or gray-scale information. For this reason, some graphics operators, notably the **image** operator, may not be used to define the shape of a character that is to be cached.

6.6.2 Font dictionaries

Font dictionaries contain certain key-value pairs. Some are fixed, while some may be altered by TrueImage operators. The following key-value pairs are mandatory.

FontMatrix	array	matrix mapping character definition units to user space units. Built-in fonts are defined on a 1000×1000 dot grid, hence their matrix is [0.001 0 0 0.001 0 0]
FontType	integer	number indicating type. 1 for PostScript fonts, 3 for user-defined, 42 for TrueType.
FontBBox	array	four-number array specifying lower-left and upper-right character definition coordinates of font bounding box, the smallest rectangle enclosing the shapes of all characters in the font.
Encoding	array	array of 256 character names, defining character code-to-character name mapping.

Built-in fonts also contain the following entries:

FontName	name	the font's name
PaintType	integer	a code describing character appearance 0 - filled 1 - stroked 2 - outlined 3 - (setting held in character description)
Metrics	dictionary	width and side bearing (although this is normally encoded in the character description)
StrokeWidth	number	stroke width for outline fonts (**PaintType** 2)
FontInfo	dictionary	dictionary containing further information
UniqueID	integer	unique font identifier
CharStrings	dictionary	dictionary associating character names with shape description procedures. (Shape descriptions are stored in a protected format)
Private	dictionary	further protected information

When fonts are named using **definefont**, a new key, **FID**, is inserted into the dictionary and a FontID value is associated with it. When a copy of an existing font is manipulated in some way, the copy's **FID** key-value pair should be discarded.

6.6.3 Character encoding

As already mentioned, font dictionaries map character names to shapes, and the encoding vector maps character codes to names. Character names are typically the character itself 'T' or 't', or a descriptive term, such as 'ampersand' or 'four'. The encoding vector is a 256-element array that holds the names of characters in successive array elements. The array index is used to index the names, hence the order of the character names in the array determines the correspondence between integer character codes and the character names.

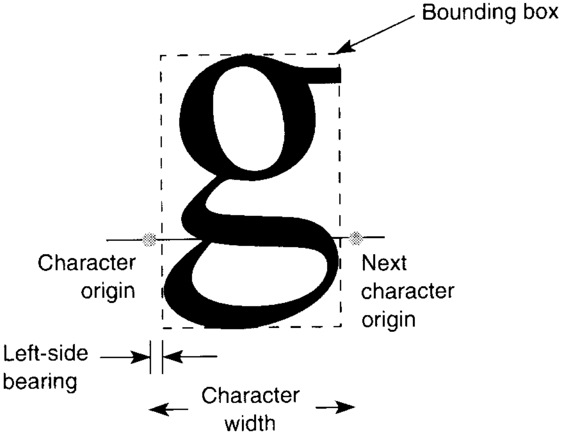
If a particular code does not have a corresponding name, that position in the array contains the name **.notdef**. Printing an undefined character produces no visible output, however, undefined characters do have a small width, causing the current position to be updated.

When a printing operator such as **show** attempts to print a character within a string (say, character code 65) it first looks up element 65 in the encoding vector to find the name of the character. Supposing the name of the character is 'A', it then looks up the procedure value associated with the name 'A' in the **CharStrings** directory of the current font dictionary, and executes it, rendering the shape onto the page.

Character encoding may be altered by modifying the encoding vector. For example, if element 65 of the vector is set to the name 'four', the character code 65 in a string would be rendered according to the procedure definition associated with the name 'four' in **CharStrings**.
Thus the mapping from character codes to character shapes may be freely altered. This allows any character set to be combined with any typeface.

6.6.4 Font metrics

Font metrics are a set of parameters defining a character's position relative to the characters either side. Within a font character shapes are defined on a grid coordinate system known as the character coordinate system.



Character rendering is referenced to the origin (0,0) of the character coordinate system. Printing operators such as **show** align the character's origin with the user space current position when printing the character.

A character's width is the distance between its origin and the point at which the next character's origin will be.

The bounding box is the smallest vertical rectangle that will enclose the character's shape. The bounding box is expressed in terms of its lower-left and upper-right hand corners, and is stored in the font directory under the key **FontBBox**.

The side bearing is the distance between the character's origin and the left edge of the bounding box. This distance may be negative.

6.6.5 Modifying fonts

Apart from simply specifying a size, the most common font manipulation that is performed by TrueImage programs is to change the encoding. This is done by making a copy of the font required, discarding the **FID** key-value pair, and inserting a new encoding vector into the copy, under the key **Encoding**.

The following example demonstrates how the EBCDIC encoding may be applied to a copy of an existing font, to create a new font. The code assumes that a dictionary **newfontdict** has already been defined, containing the EBCDIC character code-to-character name mapping. The new font is stored under the name **Times-Roman-EBCDIC**.

```
/Times-Roman findfont
dup length dict /newfontdict exch def
{ 1 index /FID ne
{newfontdict 3 1 roll put }
{pop pop}
ifelse
} forall
newfontdict /Encoding EBCDIC put
/Times-Roman-EBCDIC newfontdict definefont pop
```

Similarly a font's metrics may be altered. This is done by making a copy of the font required, discarding the **FID** key-value pair, and inserting a new dictionary into the copy, under the key **Metrics**. The new dictionary associates character names with either a new x-width only (specified as a single number), or a new left side bearing and x-width (specified either as an array of two numbers, or as an array of four numbers which specify vectors).

In the following example, this technique is used to create a new version of the Courier font, New-Courier, in which the letters "A – Z" and "a – z" have their x-widths and left-side bearings set to 900 and 50 character coordinate units respectively. (One character coordinate unit = 1/1000 of a user space unit).

```
/Courier findfont
dup length 1 add dict /newfontdict exch def
{ 1 index /FID ne
{newfontdict 3 1 roll put }
{pop pop}
ifelse
} forall
52 dict begin
[/A /B /C /D /E /F /G /H /I /J /K /L /M /N /O /P /Q /R /
S /T /U /V /X /Y /Z /a /b /c /d /e /f /g /h /i /j /k /l
/m /n /o /p /q /r /s /t /u /v /x /y /z]
{50 900 def} forall
newfontdict /Metrics currentdict put end
/New-Courier newfontdict definefont pop
```

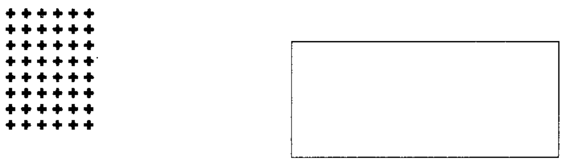
6.6.6 Creating a new font

Creating a new TrueType font is a significant undertaking. High-level applications exist to perform this function, so the need to create a font at the level of TrueImage code will rarely surface. Briefly, a user-defined font must contain the required font entries described above, must have a FontType of 3, and must also contain a procedure called **BuildChar** that constructs the characters according to the character coordinate system.

6.7 Graphic effects

6.7.1 Gray scales

On a monochrome printer, gray scales are rendered using a technique known as half-toning. This involves laying down a screen, some pattern of black and white pixels so that the result may appears as a shade of gray to the naked eye. The half-tone screen is defined in terms of an imaginary grid of rectangular cells covering the device space. Each printer pixel belongs in a particular cell, and each cell normally contains many pixels. The grid's frequency is the number of cells per inch, and the grid may be orientated at any angle to the device coordinate system. Each cell can be made to approximate to a given gray scale by having a set combination of its pixels painted black, and the rest left as white. The darker the gray scale, the more pixels are painted black.



5% gray scale using the half-tone screen shown

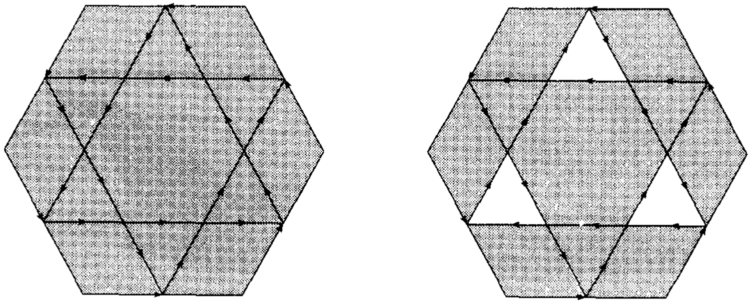
A TrueImage program may re-define the half-tone screen by defining a procedure to determine the exact pixel color combination for any requested gray scale. This can be set using the **setscreen** operator.

If gray scales specified by TrueImage are not accurately reflected on the printer, a new mapping of specified gray levels to printer gray levels may be defined using the **settransfer** function.

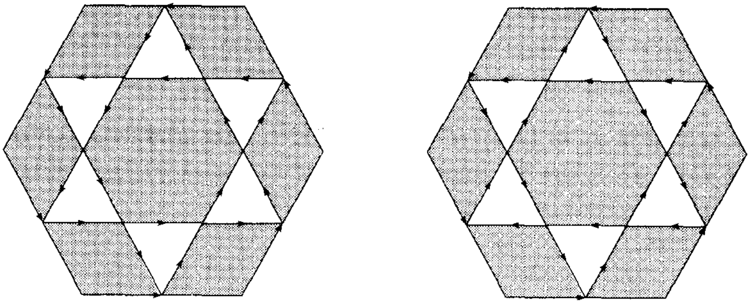
6.7.2 Filling complex paths

Complex paths that intersect themselves, or that contain subpaths that enclose other subpaths, are filled according to one of two rules: the non-zero winding rule and the even-odd rule. In either case, areas that are judged inside the path are painted, areas outside the path are left blank.

Using the zero-winding rule, a point's status is determined as follows. Imagine a straight line from the point to a point outside the path. Start with a counter at zero. Add one to the counter for each time the line is crossed by a path segment from left to right, and subtract one for each time it is crossed by a path segment from right to left. If the final result is zero, the point is outside the path, otherwise it is inside.



The even-odd rule also imagines a straight line from the point to a point outside the path. If this line is crossed an odd number of times by path segments, it is inside the path, otherwise it is outside.

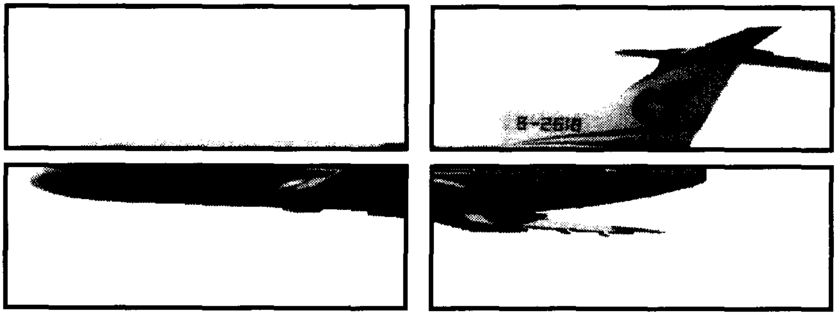


Polygons are filled in the same manner irrespective of the direction of the constituent sub-paths

fill paints paths using the zero-winding rule; **eofill** uses the even-odd rule. In some instances each operator yields the same output. In other cases they will generate different results.

6.7.3 Clipping path

The clipping path is a path that defines the area of the page in which graphic output can appear. The clipping path can be any path. This feature enables images or other graphic elements to be clipped, and also allows interesting special effects to be achieved.



6.7.4 Importing images

Sampled bitmap images, such as TIFF images, may be rendered as part of a TrueImage page description. The **image** operator performs this function. The image can be from any source; typically it may be read from a file. Image samples (pixels) may be rendered in up to 256 gray scales.

Images are read as a set of raster rows, from left to right, and from bottom to top. **image** always renders the image starting at the point (0,0), so it is usually necessary to use **translate** immediately beforehand.

6.8 Operators

6.8.1 Operator description syntax

This section contains explanation of all TrueImage operators available in the language version implemented on this printer. The formal specification of each operator shows the operator name in bold, preceded by its operands (the objects it takes from the operand stack), and followed by the objects that it places on the operand stack. A dash preceding the operator name indicates that it takes no operands; a dash following the name indicates that it returns no result. Hence this notation shows the state of the top of the stack immediately before and immediately after execution of the operator. The order in which operands are shown indicates their relative position on the stack; the rightmost operand is on top.

The names used to describe operands either indicate their object type or the parameter they represent. *any* stands for an object of any type, *num* stands for an integer or real number, *proc* represents an executable array or packed array, *matrix* is a six-number array, and *font* a font dictionary. *angle*, *height* etc. are numbers that represent the suggested parameter.

The symbol |- represents the bottom of the stack.

6.8.2 Stack operators

pop

any **pop** -

discards the top stack element.

Errors - stackunderflow

exch

any_1 any_2 **exch** any_2 any_1

exchanges the top two stack elements.

Errors - stackunderflow

dup

any **dup** any any

duplicates the top stack element.

Errors - stackoverflow, stackunderflow

copy

any_1 any_2 ... any_n n **copy** any_1 any_2 ... any_n any_1 any_2 ... any_n

duplicates the *n* stack elements any_1 to any_n.

Errors - invalidaccess, rangecheck, stackunderflow, stackoverflow, typecheck

index

any_n ... any_0 n **index** any_n ... any_0 any_n

makes a copy of element any_n (the *n*th element down from the top of the stack) and puts it on top of the stack.

Errors - rangecheck, stackunderflow, typecheck

roll

any_{n-1} ... any_0 n j **roll** $any_{(j-1) \bmod n}$ ... any_0 any_{n-1} $any_{j \bmod n}$

rotates the elements any_{n-1} ... any_0 through *j* stack positions. *n* is the number of elements rotated. Positive *j* indicates that elements shift upwards with the old topmost element(s) inserted at position. Negative *j* indicates that elements shift downwards with the former lowest element(s) brought to the top of the stack.

(1) (2) (3) (4) 3 –1 **roll** => (1) (3) (4) (2)
(1) (2) (3) (4) 4 2 **roll** => (3) (4) (1) (2)
Errors - rangecheck, stackunderflow, stackoverflow, typecheck

clear

I- any_1 ... any_n **clear** -

discards all elements from the stack.

count

I- any_1 ... any_n **count** I- any_1 ... any_n n

returns the number of items on the stack.

Errors - stackoverflow

mark

- **mark** mark

pushes a mark object onto the stack. A mark acts as place-holder. The stack may contain any number of marks.

Errors - stackoverflow

cleartomark

mark obj_1 ... obj_n **cleartomark** -

discards all objects from the stack above and including the topmost mark object.

Errors - unmatchedmark

counttomark

mark obj_1 ... obj_n **counttomark** mark obj_1 ... obj_n n

returns the number of elements on the stack above the topmost mark object.

Errors - stackoverflow, unmatchedmark

6.8.3 Maths operators

add

num_1 num_2 **add** sum

returns the sum of the two numbers on top of the stack. The result is integer if both operands are integers, and real otherwise.

Errors - stackunderflow, typecheck, undefinedresult

div

num_1 num_2 **div** quotient

returns the result of dividing num_1 by num_2. The result is always real.

Errors - stackunderflow, typecheck, undefinedresult

idiv

int_1 int_2 **idiv** quotient

returns the result of dividing int_1 by int_2. The result is always an integer.

Errors - rangecheck, stackunderflow, typecheck, undefinedresult

mod

int_1 int_2 **mod** remainder

returns the remainder left when dividing int_1 by int_2. The result is always an integer and has the same sign as int_1.

Errors - stackunderflow, typecheck, undefinedresult

mul

num_1 num_2 **mul** product

returns the product of the two numbers on top of the stack. The result is integer if both operands are integers, and real otherwise.

Errors - stackunderflow, typecheck, undefinedresult

sub

num_1 num_2 **sub** difference

returns the result of subtracting num_2 from num_1. The result is integer if both operands are integers, and real otherwise.

Errors - stackunderflow, typecheck, undefinedresult

abs

num_1 **abs** num_2

returns the absolute value of num_1.

Errors - stackunderflow, typecheck

neg

num_1 **neg** num_2

returns the result of multiplying *num_1* by −1.

Errors - stackunderflow, typecheck

ceiling

num_1 **ceiling** num_2

returns the smallest integer value not less than *num_1*. If *num_1* is a real number, *num_2* will be also.

Errors - stackunderflow, typecheck

floor

num_1 **floor** num_2

returns the largest integer value not greater than *num_1*. If *num_1* is a real number, *num_2* will be also.

Errors - stackunderflow, typecheck

round

num_1 **round** num_2

returns the closest integer value to *num_1*. If *num_1* is equidistant between two integers, the larger of the two is returned. If *num_1* is a real number, *num_2* will be also.

Errors - stackunderflow, typecheck

truncate

num_1 **truncate** num_2

returns the closest integer value obtained by removing fractional part from *num_1*. If *num_1* is a real number, *num_2* will be also.

Errors - stackunderflow, typecheck

sqrt

num **sqrt** real

returns the square root of *num*.

Errors - rangecheck, stackunderflow, typecheck

atan

num_1 num_2 **atan** angle

returns the angle, in degrees, whose tangent is *num_1/num_2*. The result is real. *num_1* and *num_2* cannot both be 0.

Errors - stackunderflow, typecheck, undefinedresult

cos

angle **cos** real

returns the cosine of *angle* in degrees.

Errors - stackunderflow, typecheck

sin

angle **sin** real

returns the sine of *angle* in degrees.

Errors - stackunderflow, typecheck

exp

num exponent **exp** real

returns the result of raising *num* to the power *exponent.* The result is a real number.

Errors - stackunderflow, typecheck, undefinedresult

ln

num **ln** real

returns the natural logarithm of *num.* The result is a real number.

Errors - stackunderflow, typecheck, undefinedresult

log

num **log** real

returns the base 10 logarithm of *num.* The result is a real number.

Errors - stackunderflow, typecheck, undefinedresult

rand

- **rand** int

returns a random integer in the range $0 - 2^{31}$.

Errors - stackoverflow

srand

int **srand -**

seeds the random number generator using int

Errors - stackunderflow, typecheck

rrand

- **rrand** int

returns an integer representing the current position in the random number sequence. This result may be used by srand to reset the random number generator to the given position in the sequence.

Errors - stackoverflow

6.8.4 Logical operators

eq

any_1 any_2 **eq** bool

compares two objects for equality, returning *true* if they are equal, *false* if they are not. Simple objects are equal if their types and values are the same. Composite objects other than strings are equal only if they share the same value: separate, but identical, values are considered unequal. Strings are equal if they are the same length and are made up of the same characters in the same order. An integer and a real number can be equal to one another, as can a name and a string.

The executable and access attributes of any_1 and any_2 need not be the same for them to be considered equal.

Errors - invalidaccess, stackunderflow

ne

any_1 any_2 **ne** bool

compares two objects for inequality, returning *false* if they are equal, *true* if they are not. Equality of objects is as described above under the **eq** operator.

Errors - invalidaccess, stackunderflow

ge

num_1 num_2 **ge** bool

returns *true* if num_1 is greater than or equal to num_2, and *false* if num_1 is less than num_2.

Errors - invalidaccess, stackunderflow, typecheck

gt

num_1 num_2 **gt** bool

returns *true* if num_1 is greater than num_2, and *false* if num_1 is less than or equal to num_2.

Errors - invalidaccess, stackunderflow, typecheck

le

num_1 num_2 **le** bool

returns *true* if num_1 is less than or equal to num_2, and *false* if num_1 is greater than num_2.

Errors - invalidaccess, stackunderflow, typecheck

lt

num_1 num_2 **lt** bool

returns *true* if num_1 is less than num_2, and *false* if num_1 is greater than or equal to num_2.

Errors - invalidaccess, stackunderflow, typecheck

and

bool bool **and** bool
int int **and** int

If the operands are boolean, **and** returns *true* if both are true and *false* otherwise. If the operands are integers, **and** converts them to binary, performs a bitwise 'and' operation, and returns the result as a decimal integer.
Errors - stackunderflow, typecheck

not

bool **not** bool
int **not** int

If the operand is boolean, **not** returns the opposite boolean value. If the operand is an integer, **not** converts it to binary, performs a bitwise 'not' operation, and returns the result as a decimal integer.
Errors - stackunderflow, typecheck

or

bool bool **or** bool
int int **or** int

If the operands are boolean, **or** returns *true* if either is true and *false* if both are false. If the operands are integers, **or** converts them to binary, performs a bitwise 'inclusive or' operation, and returns the result as a decimal integer.
Errors - stackunderflow, typecheck

xor

bool bool **xor** bool
int int **xor** int

If the operands are boolean, **xor** returns *true* if one of them only is true and *false* if both are true or both are false. If the operands are integers, **xor** converts them to binary, performs a bitwise 'exclusive or' operation, and returns the result as a decimal integer.
Errors - stackunderflow, typecheck

true

- **true** true

pushes a boolean object with value *true* onto the stack.
Errors - stackoverflow

false

- **false** false

pushes a boolean object with value *false* onto the stack.
Errors - stackoverflow

bitshift

int$_1$ shift **bitshift** int$_2$

converts *int* to binary, shifts the binary number left by *shift* bits, and returns the result as a decimal integer. Bits shifted out are lost, zeroes are shifted in from the right. A negative value of *shift* causes a right shift to be performed (which will only be arithmetically correct if the original number is positive). *int* and *shift* must both be integers.

Errors - stackunderflow, typecheck

6.8.5 Path construction operators

newpath

- newpath -

sets the current path to empty. After a **newpath** the current point is undefined. Use the **moveto** operator to set a new current point, and start the definition of a new path.

currentpoint

- currentpoint x y

returns the user coordinates of the current point, the endpoint of the current path. Since the TrueImage interpreter always immediately converts points in the current path to device space coordinates, modification to the CTM will change the (x,y) values returned by a given device space point.

Errors - nocurrentpoint, stackoverflow, undefinedresult

moveto

x y **moveto -**

sets (x,y) to be the current point, thereby starting a new subpath within the current path. **moveto** does not add any line segments to the current path. If the previous current point is not connected to any other point by a line, **moveto** causes it to be deleted from the current path.

Errors - limitcheck, stackunderflow, typecheck

rmoveto

dx dy **rmoveto -**

sets the current point relative to the previous current point. (dx, dy) specifies the coordinates of the new current point in relation to the previous one. If the current path is empty, a **nocurrentpoint** error is executed. Otherwise **rmoveto** functions in the same way as **moveto**.

Errors - nocurrentpoint, limitcheck, stackunderflow, typecheck

lineto

x y **lineto -**

adds a straight line segment to the current path from the current point to (x,y). (x,y) becomes the new current point. If the current path is empty, a **nocurrentpoint** error is executed.

Errors - nocurrentpoint, limitcheck, stackunderflow, typecheck

rlineto

dx dy **rlineto** -

adds a straight line segment to the current path from the current point, (x,y), to (x+dx,y+dy). (dx, dy) specifies the coordinates of the line endpoint in relation to the current point. (x+dx,y+dy) becomes the new current point. If the current path is empty, a **nocurrentpoint** error is executed.

Errors - nocurrentpoint, limitcheck, stackunderflow, typecheck

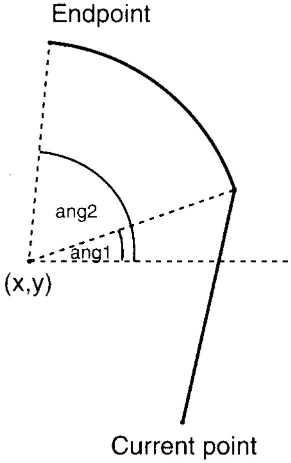
arc

x y radius ang$_1$ ang$_2$ **arc** -

adds a circular arc to the current path, optionally preceded by a straight line segment. (x,y) is the arc's center, *radius* its radius, *ang$_1$* the angle of elevation of the arc's start point and *ang$_2$* the elevation of its endpoint. Angles are counterclockwise from the user space x-axis. The endpoint becomes the new current point.

If the current path is not empty when **arc** is invoked, **arc** includes a straight line from the current point to the arc's start point. Otherwise no straight-line segment is included.

If x- and y-axis units have been scaled to different sizes, the arc will appear elliptical.



Errors - rangecheck, limitcheck, stackunderflow, typecheck

arcn

x y radius ang_1 ang_2 **arcn** -

performs the same function as **arc**, except that ang_1 and ang_2 are interpreted as clockwise from the user space x-axis.

Errors - rangecheck, limitcheck, stackunderflow, typecheck

arcto

x_1 y_1 x_2 y_2 radius **arcto** xt_1 yt_1 xt_2 yt_2

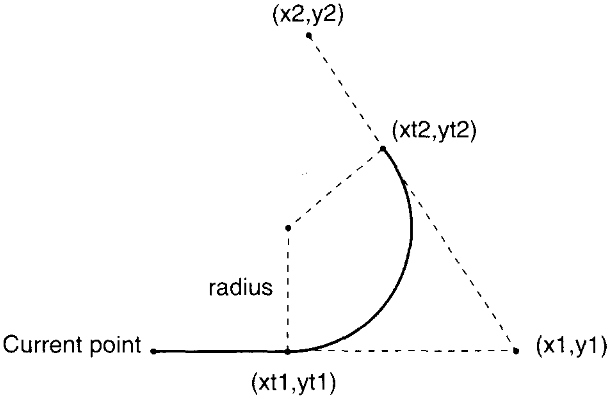
adds a circular arc to the current path, optionally preceded by a straight line segment. The arc is defined by the radius *radius* and two lines, a line from the current point to (x_1, y_1), and a line from (x_1, y_1) to (x_2, y_2). These lines are tangential to the arc.

arcto includes a straight line from the current point to the arc's start point, unless they coincide.

arcto returns the start and endpoints of the arc, (xt_1, yt_1), and (xt_2, yt_2). The arc's endpoint, (xt_2, yt_2), becomes the new current point.

If x- and y-axis units have been scaled to different sizes, the arc will appear elliptical.

If the current path is empty, a **nocurrentpoint** error is executed.

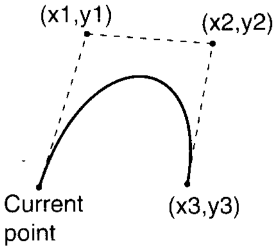


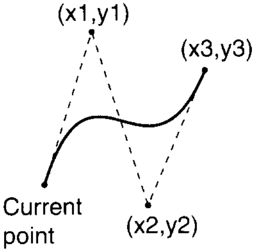
Errors - nocurrentpoint, rangecheck, limitcheck, stackunderflow, typecheck, undefinedresult

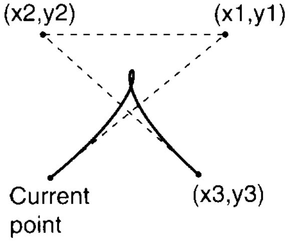
curveto

x_1 y_1 x_2 y_2 x_3 y_3 **curveto** -

adds a curve to the current path from the current point to the point (x_3,y_3). (x_3,y_3) becomes the new current point. The three parameter points and the current point define the curve geometrically. The lines from the current point to (x_1,y_1), and from (x_2,y_2) to (x_3,y_3) are tangential to the curve. The curve leaves the current point in the direction of (x_1,y_1) and approaches the point (x_3,y_3) from the direction of (x_2,y_2). (x_1,y_1) and (x_2,y_2) are control points: their positions relative to the current point and (x_3,y_3) define how steep the curve is along its length. The curve is always enclosed by the convex quadrilateral linking the four points.







If the current path is empty, a **nocurrentpoint** error is executed.

Errors - limitcheck, nocurrentpoint, stackunderflow, typecheck

rcurveto

dx_1 dy_1 dx_2 dy_2 dx_3 dy_3 **rcurveto** -

adds a curve to the current path from the current point, (x,y) to the point (x+dx_3,y+dy_3). (x+dx_3,y+dy_3) becomes the new current point. **rcurveto** functions in the same way as **curveto** except that the operand points are specified relative to the current point.

Errors - limitcheck, nocurrentpoint, stackunderflow, typecheck,
undefinedresult

closepath

- closepath -

closes the current subpath within the current path by adding a straight line from the current point to the subpath's starting point, the point moved to with the most recent **moveto** or **rmoveto** operator.
Errors - limitcheck

flattenpath

- flattenpath -

replaces the current path with an equivalent path in which all curved segments are replaced by a series of straight lines that approximate the curves. The degree of flattening is determined by the flatness parameter in the current graphics state.
Errors - limitcheck

reversepath

- reversepath -

reverses the direction and order of all segments in each subpath of the current path. The order of the subpaths within the current path remains unchanged.

strokepath

- strokepath -

calculates the path that would tightly enclose the shape of the current path, if it were stroked. The resulting path is made the current path.
Errors - limitcheck

charpath

string bool **charpath** -

calculates the path formed by the outlines of the characters in string, according to the current font's size and character definitions. **charpath** adds the resulting path to the current path. If *bool* = true, **charpath** applies **strokepath** to the character path, otherwise it does not. Setting *bool* to true makes the resulting path suitable for use with the **fill** or **clip** operators, but not with **stroke**. If *bool* = false, the path is suitable for stroking only.

Fonts designed to be stroked have a dictionary **PaintType** value set to 1; fonts designed for filling have **PaintType** 2; and those designed for outlining have **PaintType** 0.
Errors - limitcheck, nocurrentpoint, stackunderflow, typecheck

clippath

- clippath -

makes the current clipping path the current path. **clippath** can be used to find out the printer's imageable area.

pathbbox

- **pathbbox** ll_x ll_y ur_x ur_y

returns the user coordinates of the lower left- and upper right-hand corners of the current path's bounding box. The bounding box is a rectangle, with sides parallel to the user space axes, that tightly encloses the current path plus the control points of any curved segments in the path. To obtain the bounding box of the current path alone (without curve control points), first flatten the path with the **flattenpath** operator.
If the current path is empty, a **nocurrentpoint** error is executed.
Errors - nocurrentpoint, stackunderflow

pathforall

moveproc lineproc curveproc closeproc **pathforall** -

executes one of the four procedure operands on each element of the current path in turn. Path elements fall into four categories, those defined with a **moveto** or **rmoveto**, those defined with a **lineto** or **rlineto**, those defined with a **curve** or **arc** operator, and those set with **closepath**. **pathforall** uses the appropriate procedure for each segment.

For each element in turn **pathforall** executes a procedure as follows:

Element type (definition operators)	Action
moveto, rmoveto	push x,y : execute moveproc
lineto, rlineto	push x,y : execute lineproc
curved	push x_1,y_1,x_2,y_2,x_3,y_3 : execute curveproc
closepath	push x,y : execute closeproc

If **charpath** has been used to define part of the current path, an **invalidaccess** error is executed. x and y coordinates are user space coordinates which **pathforall** obtains by multiplying the device space coordinates by the inverse of the CTM. If the CTM has been modified since the path was laid down, the coordinates will be different to those that were used to define the path. Conversely, **pathforall** may be used to convert a path defined in one user coordinate system for use in another.
Errors - stackunderflow, stackoverflow, typecheck

initclip

- initclip -

sets the clipping path to the printer's default value; usually the imageable area. **framedevice** and **banddevice** can be used to set the default clipping path.

clip

- clip -

closes any open subpaths in the current path and sets the clipping path to be the intersection of the current clipping path with the current path. The inside of the current path is established according to the non-zero winding rule; the inside of the current clipping path is established according to whichever rule was in force when it was set.

clip does not perform an automatic **newpath**. Subsequently defined path elements are appended to the new path.
To restore the previous clipping path, enclose **clip** in a **gsave**, **grestore** pair.
Errors - limitcheck

eoclip

- eoclip -

performs the same function as clip, except that the inside of the current path is established according to the even-odd rule.
Errors - limitcheck

6.8.6 Painting operators

erasepage

- erasepage -

paints the entire current page (not just the clipping path) using gray level 1, which is usually white. The **settransfer** operator can be used to assign a different mapping of TrueImage gray scales to device gray scales.

fill

- fill -

fills the current path with the current color. Any open subpaths of the current path are automatically closed. **fill** uses the non-zero winding rule to determine the inside of a path. After filling the current path **fill** sets the current path to empty. To preserve the current path, encapsulate **fill** within a **gsave, grestore** pair.

Errors - limitcheck

eofill

- eofill -

fills the current path with the current color. **eofill** uses the even-odd rule to determine the inside of a path. Otherwise, it behaves identically to the **fill** operator.

Errors - limitcheck

stroke

- stroke -

paints a line tracing the current path using the current color. **stroke** renders lines according to the current graphics state settings. After stroking the current path **stroke** sets the current path to empty. To preserve the current path, encapsulate **stroke** within a **gsave, grestore** pair.

A subpath consisting of a single point, or more than one point at the same coordinates, will be stroked only if the subpath is closed and round caps are the current line cap setting. Otherwise no output is generated.

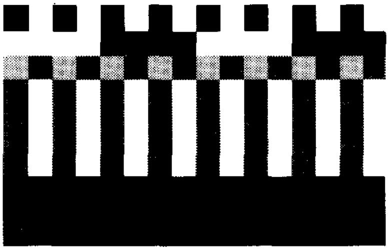
Errors - limitcheck

image

width height bps matrix proc **image** -

draws an imported image on the current page. The image is made up of samples, each sample corresponding to one pixel in the original image. The image is *width* x *height* pixels in dimension, and each pixel is represented by *bps* bits. *bps* can be 1, 2, 4 or 8. The image is rendered starting from (0, 0). *matrix* maps the image to user space. The image data is received as a stream of characters (values from 0 to 255), one row at a time. Each row consists of a whole number of characters, any trailing bits within the final character of a row are ignored. **image** executes *proc* as many times as is necessary to obtain the specified amount of data. Any extra data is discarded. For each sample a bit setting of all 1s maps to a white pixel, and all 0s to a black. Intermediate gray scales have values in between.

```
newpath
10 10 translate
18 18 scale % graphics unit is 1/4 inch square
16 10 2 [ 1 0 0 1 0 0 ]
{<00000000000000000000000077777777777777777777777777777
77799999999ff00ff0033333333>} image
% 16*10 pixels, 2 bits/pixel, 1*1 pixel/graphics unit
showpage
```



Errors - stackunderflow, typecheck

imagemask

width height polarity matrix datasrc **imagemask** -
dict **imagemask** -

performs a similar function to the **image** operator, rendering an imported image onto the current page. **imagemask** uses the source image as a mask of one-bit samples to build up an image in the current color.

Parameters may be specified as a list of objects or as a single dictionary object that contains the relevant key-value pairs.

The image is *width* x *height* pixels in dimension and is rendered starting from (0, 0).

polarity is a boolean value that determines the mask's polarity. If *polarity* = *true*, those parts of the image represented by 1 are painted, those represented by 0 are left unchanged. If *polarity* = *false*, parts represented by 0 are painted, and those represented by 1 are left unchanged. In the second form of **imagemask**, the polarity is specified by the **Decode** entry in the image dictionary. **Decode** values of [1,0] and [0,1] correspond to *true* and *false* respectively.

matrix maps the image to user space.

datasrc may be a procedure, string or readable file object. **imagemask** either executes or reads from *datasrc* as many times as is necessary to obtain the specified amount of data. The image data is received as a stream of characters (values from 0 to 255), one row at a time. Each row consists of a whole number of characters. Any trailing bits are discarded.

Any extra image data is discarded.

Errors - stackunderflow, typecheck, undefinedresult, limitcheck,
invalidaccess, ioerror

6.8.7 String operators

string

int **string** string

creates a string of length *int* and initializes all characters to the value 0. *int* may not be negative.

Errors - limitcheck, rangecheck, stackunderflow, typecheck, VMerror

length

string **length** int

returns the number of characters in the string.

Errors - invalidaccess, stackunderflow, typecheck

get

string index **get** int

returns the character in the string identified by *index*. *index* can range from 0 to $n-1$, where n is the number of characters in the string.

Errors - invalidaccess, rangecheck, stackunderflow, typecheck, undefined

put

string index int **put** -

replaces the character in the string identified by *index* with *int*. *index* can range from 0 to $n-1$, where n is the number of characters in the string.

Errors - invalidaccess, rangecheck, stackunderflow, typecheck

getinterval

string index count **getinterval** substring

creates a new string comprising a sequence of *count* characters from the original string, starting from the character in *string* identified by *index*. *index* + *count* cannot exceed the number of characters in the string. *count* must be positive.

Errors - invalidaccess, rangecheck, stackunderflow, typecheck

putinterval

string_1 index string_2 **putinterval** -

copies $string_2$ into $string_1$, replacing the sub-sequence of characters of $string_1$ beginning with the character identified by *index*.

Errors - invalidaccess, rangecheck, stackunderflow, typecheck

copy

string$_1$ string$_2$ **copy** substring

copies all characters of *string$_1$* into *string$_2$*, returning the initial substring of *string$_2$* that contains the copied characters. The executable and access attributes of *substring* are the same as those of *string$_2$*. *string$_1$* cannot be longer than *string$_2$*.

Errors - invalidaccess, rangecheck, stackunderflow, stackoverflow, typecheck

forall

string proc **forall** -

executes *proc* on each character of the string in turn. The integer representation of each character, starting with the first, is pushed onto the stack for use by *proc*. If *proc* does not use or dispose of the string's integer character representations, they will build up on the stack. If *proc* executes an **exit, forall** ceases execution. If the string is 0 characters long, *proc* is not executed.

Errors - invalidaccess, stackunderflow, typecheck

anchorsearch

string seek **anchorsearch** post match true

string seek **anchorsearch** string false

tests to see whether the string *seek* matches the start of the string *string*. If it does, **anchorsearch** returns *true*, *match*, the matching part of *string*, and *post*, the rest of *string*. If *seek* does not match, **anchorsearch** returns *false*, and the original string *string*. In order to match, *seek* must be no longer than *string*.

Errors - invalidaccess, stackunderflow, stackoverflow, typecheck

search

string seek **search** post match pre true

string seek **search** string false

tests to see whether the string *seek* matches any substring of the string *string*. If it does, **search** returns *true*, *pre*, the non-matching starting sequence of *string*, *match* (the matching part of *string*) and *post*, the rest of *string*. If *seek* does not match, **search** returns *false*, and the original string *string*. In order to match, *seek* must be no longer than *string*.

Errors - invalidaccess, stackunderflow, stackoverflow, typecheck

token

string **token** post obj true
string **token** false

scans *string*, searching for a token that represents a TrueImage object. If **token** can locate an object token within *string*, it returns *true*, the object itself, and the substring from the end of the token to the end of the string. If **token** cannot locate an object token within *string*, it returns *false*. The object can be a number, name, string, data array or executable array. The object is the same as the object that would be returned if the string were executed directly, however, the object is not executed, merely pushed onto the operand stack.

Only the first object encountered is returned. To parse the whole string, repeated use of **token** would be necessary.

token discards all characters up to the final character of the token. If the token is a name or number, the first following whitespace character is discarded as well. If the token is a string or array ending with a), >,] or }, that character (but no following characters) is discarded.

Errors - invalidaccess, ioerror, rangecheck, stackunderflow, stackoverflow, syntaxerror, typecheck, undefinedresult

eq

string_1 string_2 **eq** bool

compares two strings, or a string and a name, for equality, returning *true* if they are equal, *false* if they are not. Strings (or a sting and a name) are equal if they are the same length and are made up of the same characters in the same order.

The executable and access attributes of $\textit{string}_1$ and $\textit{string}_2$ need not be the same for them to be considered equal.

Errors - invalidaccess, stackunderflow

ne

string_1 string_2 **ne** bool

compares two strings, or a string and a name, for inequality, returning *false* if they are equal, *true* if they are not. Equality is as described above under the **eq** operator.

Errors - invalidaccess, stackunderflow

ge

string$_1$ string$_2$ **ge** bool

returns *true* if *string$_1$* is greater than or equal to *string$_2$*, and *false* if *string$_1$* is less than *string$_2$*. The two strings are compared character value by character value until a pair of values is found that differ (or until one string is exhausted). Whichever string's character in the unequal pair has the higher value (or whichever string is longer if all character pairs match) is considered the greater of the two. Strings are equal if they are the same length and are made up of the same characters in the same order.

Errors - invalidaccess, stackunderflow, typecheck

gt

string$_1$ string$_2$ **gt** bool

returns *true* if *string$_1$* is greater than *string$_2$*, and *false* if *string$_1$* is less than or equal to *string$_2$*. String ordering is as described under the **ge** operator above.

Errors - invalidaccess, stackunderflow, typecheck

le

string$_1$ string$_2$ **le** bool

returns *true* if *string$_1$* is less than or equal to *string$_2$*, and *false* if *string$_1$* is greater than *string$_2$*. String ordering is as described under the **ge** operator above.

Errors - invalidaccess, stackunderflow, typecheck

lt

string$_1$ string$_2$ **lt** bool

returns *true* if *string$_1$* is less than *string$_2$*, and *false* if *string$_1$* is greater than or equal to *string$_2$*. String ordering is as described under the **ge** operator above.

Errors - invalidaccess, stackunderflow, typecheck

6.8.8 Array operators

array

int **array** array

creates an array of length *int*, and initializes all elements to null objects.
Errors - rangecheck, stackunderflow, typecheck, VMerror

[

- **[** mark

pushes a mark object onto the stack, marking the start of a sequence of objects that will be formed into an array.
Errors - stackoverflow

]

mark obj_0 ... obj_{n-1} **]** array

creates an array comprising all the elements above the topmost mark on the stack. The object immediately above the mark is the first element of the array, and the topmost object is the last.
Errors - unmatchedmark, VMerror

length

array **length** int

returns the number of elements in the array.
Errors - invalidaccess, stackunderflow, typecheck

get

array index **get** any

returns the array element identified by *index*. *index* can range from 0 to *n*−1, where *n* is the number of elements in the array.
Errors - invalidaccess, rangecheck, stackunderflow, typecheck, undefined

put

array index any **put** -

replaces the element in *array* identified by *index* with *any*. *index* can range from 0 to *n*−1, where *n* is the number of elements in the array.
Errors - invalidaccess, rangecheck, stackunderflow, typecheck

getinterval

array index count **getinterval** subarray

creates a new array comprising a sequence of *count* elements from the original array, starting from the element in *array* identified by *index*. *index* + *count* cannot exceed the number of elements in the array. *count* must be positive.
Errors - invalidaccess, rangecheck, stackunderflow, typecheck

putinterval

array index subarray **putinterval** -

copies the elements of *subarray* into *array*, replacing the sub-sequence of elements of *array* beginning with the element identified by *index*. If elements of *subarray* are composite objects, their values are shared between *array* and *subarray*.

Errors - invalidaccess, rangecheck, stackunderflow, typecheck

aload

array **aload** element_0 ... element_{n-1} array

pushes the elements of the array onto the stack in order, followed by the array itself.

Errors - invalidaccess, stackoverflow, stackunderflow, typecheck

astore

any_0 ... any_{n-1} array **astore** array

fills the array with the *n* objects any_0 to any_{n-1}, where *n* is the array's length. any_0 becomes the first element of the array and any_{n-1} the last.

Errors - invalidaccess, stackunderflow, typecheck

copy

array_1 array_2 **copy** subarray

copies all elements of $array_1$ into $array_2$, returning the initial subarray of $array_2$ that contains the copied objects. If elements of $array_1$ are composite objects, their values are shared between $array_1$ and $array_2$. The executable and access attributes of *subarray* are the same as those of $array_2$. $array_1$ cannot be longer than $array_2$.

Errors - invalidaccess, rangecheck, stackunderflow, stackoverflow, typecheck

forall

array proc **forall** -

executes *proc* on each element of the array in turn. Each array element, starting with element 0, is pushed onto the stack for use by *proc*. If *proc* does not use or dispose of the array's objects, they will build up on the stack. If *proc* executes an **exit**, **forall** ceases execution. If *array* is empty, *proc* is not executed.

Errors - invalidaccess, stackunderflow, typecheck

6.8.9 Packed array operators

packedarray

any$_0$ ... any$_{n-1}$ n **packedarray** packedarray

creates a packed array of length *n* that has the objects *any$_0$* to *any$_{n-1}$* as its elements. The resulting object is of type *packedarraytype*, and is read-only. In all other respects a packed array behaves in the same manner as an ordinary procedure array.

Errors - rangecheck, stackunderflow, typecheck, VMerror

currentpacking

- **currentpacking** bool

returns the current array packing mode. The array packing mode can be set with the **setpacking** operator.

Errors - stackoverflow

setpacking

bool **setpacking** -

sets the array packing mode to the specified value. *true* turns array packing on; *false* turns it off. The TrueImage interpreter creates procedure arrays when it encounters TrueImage program text enclosed between '{' and '}'. If array packing is on, procedure arrays are created and stored in packed (compact) form. If array packing is off, procedure arrays are created and stored in ordinary form.

The array packing mode setting remains in effect until another **setpacking** operator is encountered, or until a **restore** command restores a previous setting.

Errors - stackunderflow, typecheck

length

packedarray **length** int

returns the number of elements in the packed array.

Errors - invalidaccess, stackunderflow, typecheck

get

packedarray index **get** any

returns the packed array element identified by *index*. *index* can range from 0 to *n*–1, where *n* is the number of elements in the array.

Errors - invalidaccess, rangecheck, stackunderflow, typecheck, undefined

getinterval

packedarray index count **getinterval** subarray

creates a new packed array comprising a sequence of *count* elements from the original packed array, starting from the element in *packedarray* identified by *index*. *index* + *count* cannot exceed the number of elements in the packed array. *count* must be positive.

Errors - invalidaccess, rangecheck, stackunderflow, typecheck

aload

packedarray **aload** $element_0$ ... $element_{n-1}$ packedarray

pushes the elements of the packed array onto the stack in order, followed by the packed array itself.

Errors - invalidaccess, stackoverflow, stackunderflow, typecheck

copy

$packedarray_1$ $array_2$ **copy** subarray

copies all elements of $packedarray_1$ into $array_2$, returning the initial subarray of $array_2$ that contains the copied objects. If elements of $packedarray_1$ are composite objects, their values are shared between $packedarray_1$ and $array_2$. The executable and access attributes of *subarray* are the same as those of $array_2$. $packedarray_1$ cannot be longer than $array_2$.

Errors - invalidaccess, rangecheck, stackunderflow, stackoverflow, typecheck

forall

packedarray proc **forall** -

executes *proc* on each element of the packed array in turn. Each packed array element, starting with element 0, is pushed onto the stack for use by *proc*. If *proc* does not use or dispose of the packed array's objects, they will build up on the stack. If *proc* executes an **exit**, **forall** ceases execution. If *packedarray* is empty, *proc* is not executed.

Errors - invalidaccess, stackunderflow, typecheck

6.8.10 Dictionary operators

dict

int **dict** dict

creates an empty dictionary with space for *int* key-value pairs.
Errors - rangecheck, stackunderflow, typecheck, VMerror

length

dict **length** int

returns the number of key-value pairs currently in the dictionary.
Errors - invalidaccess, stackunderflow, typecheck

maxlength

dict **maxlength** int

returns the maximum possible number of key-value pairs that could be held in the dictionary.
Errors - invalidaccess, stackunderflow, typecheck

begin

dict **begin** -

pushes *dict* onto the dictionary stack, making it the current dictionary, the first dictionary in which the interpreter will look up the names it encounters.
Errors - dictstackoverflow, invalidaccess, stackunderflow, typecheck

end

- **end** -

pops the current dictionary off the dictionary stack, making the one below the current dictionary. If **end** attempts to remove the bottom-most **userdict**, a **dictstackunderflow** error is executed.
Errors - dictstackunderflow

def

key value **def** -

adds the key-value pair to the current dictionary. If *key* already exists in the dictionary, the corresponding value is overwritten.
Errors - dictfull, invalidaccess, limitcheck, stackunderflow, typecheck

load

key **load** value

searches the dictionaries on the dictionary stack for *key* and returns the value corresponding to the first occurrence of *key* that it finds. **load** searches the dictionary stack starting with the topmost dictionary (the current dictionary), and works downwards. If *key* is not found, an **undefined** error is executed.

load looks up values in exactly the same way as the TrueImage interpreter, however, **load** merely returns the value, it does not try to execute it.

Errors - invalidaccess, stackunderflow, typecheck, undefined

store

key value **store** -

searches the dictionaries on the dictionary stack for *key* and associates *value* with the first occurrence of *key* that it finds. If *key* is not found, the key-value pair is added to the current dictionary. **store** searches the dictionary stack starting with the topmost dictionary (the current dictionary), and works downwards.

Errors - dictfull, invalidaccess, limitcheck, stackunderflow

get

dict key **get** any

returns the value corresponding to *key* in *dict.*

Errors - invalidaccess, rangecheck, stackunderflow, typecheck, undefined

put

dict key any **put** -

associates *any* with *key* in the dictionary. If *key* is already present in *dict*, **put** overwrites the existing value with *any.* If *key* is not present, the new key-value pair is added to *dict.* If *dict* is full, a **dictfull** error is executed.

Errors - dictfull, invalidaccess, rangecheck, stackunderflow, typecheck

known

dict key **known** bool

returns *true* if *key* is present in *dict, false* otherwise. *dict* need not be on the dictionary stack.

Errors - invalidaccess, stackunderflow, typecheck

where

key **where** dict true
key **where** false

searches the dictionaries on the dictionary stack for *key*. If it finds *key*, **where** returns *true* and the dictionary containing the first occurrence of *key*. **where** searches the dictionary stack starting with the topmost dictionary (the current dictionary), and works downwards. If *key* is not found, **where** returns *false*.

Errors - invalidaccess, stackoverflow, stackunderflow

copy

dict_1 dict_2 **copy** dict_2

copies all key-value pairs in $dict_1$ into $dict_2$, returning $dict_2$. If some values in $dict_1$ are composite objects, they are shared between $dict_1$ and $dict_2$. The executable and access attributes of $dict_2$ are the same as those of $dict_1$. $dict_2$ must initially contain no key-value pairs, and must be at least as long as $dict_1$.

Errors - invalidaccess, rangecheck, stackunderflow, stackoverflow,
typecheck

forall

dict proc **forall** -

executes *proc* on each element of the dictionary in turn. The key and the value of each key-value pair is pushed onto the stack for use by *proc*. If *proc* does not use or dispose of the dictionary's keys and values, they will build up on the stack. If *proc* executes an **exit**, **forall** ceases execution. If *dict* is empty, *proc* is not executed.

The order in which key-value pairs are processed by **forall** is unspecified. New key-value pairs generated by *proc* may or may not have *proc* executed on them.

Errors - invalidaccess, stackunderflow, typecheck

errordict

- **errordict** dict

pushes **errordict** onto the operand stack. **errordict** is the dictionary which associates the name of each error with an action.

Errors - stackoverflow

systemdict

- **systemdict** dict

pushes **systemdict** onto the operand stack. **systemdict** is the dictionary which associates the name of each TrueImage operator with its corresponding action.
Errors - stackoverflow

userdict

- **userdict** dict

pushes **userdict** onto the operand stack. **userdict** is the dictionary associating names defined by TrueImage programs with their values.
Errors - stackoverflow

currentdict

- **currentdict** dict

pushes **currentdict** onto the operand stack. **currentdict** is the dictionary on the top of the dictionary stack.
Errors - stackoverflow

countdictstack

- **countdictstack** int

returns the number of dictionaries currently on the dictionary stack.
Errors - stackoverflow

dictstack

array **dictstack** subarray

copies the names of all dictionaries on the dictionary stack into *array*, returning the initial subarray of *array* containing the dictionary names. **dictstack** writes the bottommost dictionary name into element 0 of *array*, and the topmost into element $n-1$, where n is the number of dictionaries on the dictionary stack. If *array* is too small to hold all the names, a **rangecheck** error is executed.
Errors - invalidaccess, rangecheck, stackunderflow, typecheck

6.8.11 Control operators

exec

any **exec** -

pushes the operand onto the execution stack, causing it to be executed immediately. The effects of executing an object depend on its type and access attribute, as discussed in the section Execution of objects on page 208.

Errors - stackunderflow

if

bool proc **if** -

executes proc if **bool** = true.

Errors - stackunderflow, typecheck

ifelse

bool proc_1 proc_2 **ifelse** -

executes $proc_1$ if **bool** = true, or $proc_2$ if **bool** = false.

Errors - stackunderflow, typecheck

for

start increment finish proc **for** -

executes *proc* repeatedly. **for** maintains a counter whose initial value is *start* and which is increased to *finish* in steps of *increment. proc* is executed each time the counter is incremented. The value of the counter is pushed onto the stack for use by *proc*. If *proc* does not use or dispose of it, successive values of the counter build up on the stack.

Execution ends when the counter's value exceeds *finish* (or is less than *finish*, if *increment* is negative).

Errors - stackoverflow, stackunderflow, typecheck

repeat

int proc **repeat** -

executes *proc int* times. If *proc* contains an **exit**, **repeat** will terminate when the **exit** is encountered by the interpreter.

Errors - rangecheck, stackunderflow, typecheck

loop

proc **loop** -

executes *proc* repeatedly until an **exit** or **stop** is encountered by the interpreter. If neither is encountered, execution continues until an external interrupt (an interrupt error) is received.

Errors - rangecheck, stackunderflow, typecheck

exit

- exit -

jumps out of the innermost loop, initiated by a **for**, **loop**, **repeat**, **forall**, **pathforall** or **renderbands** operator, popping the relevant operator and everything above it from the execution stack. **exit** does not change the operand or dictionary stacks.

If **exit** occurs in the context of a **run** or **stopped** operator, an **invalidexit** error is executed.

If there is no enclosing loop, **quit** is executed.

Errors - invalidexit

stop

- stop -

terminates execution of an executable object executed by a **stopped** operator, popping the **stopped** operator and everything above it from the execution stack. **stop** does not change the operand or dictionary stacks.

If there is no enclosing **stopped** context, **quit** is executed.

stopped

any **stopped** bool

executes *any*, returning *false* if *any* terminates normally, or *true* if *any* is terminated by a **stop**. Irrespective of the outcome, normal execution is then resumed.

Errors - stackunderflow

countexecstack

- countexecstack int

returns the number of objects on the execution stack.

Errors - stackoverflow

execstack

array **execstack** subarray

copies all elements on the execution stack into *array*, returning the initial subarray of *array* containing the execution stack elements. The bottom-most execution stack element is copied into array element 0, the topmost into array element (*n*–1), where *n* is the depth of the execution stack. The execution stack is not affected. If *array* is too small to hold all the elements of the execution stack, a **rangecheck** error is executed.

Errors - invalidaccess, rangecheck, stackunderflow, typecheck

quit

- quit -

terminates the current TrueImage program (if **quit** is looked up in **userdict**) or terminates the operation of the TrueImage interpreter completely (if it is looked up in **systemdict**). Normally the **userdict** definition takes precedence.

start

- start -

executed by the TrueImage interpreter on start-up, to establish the working environment.

6.8.12 *Type and attribute operators*

type

any **type** name

returns a name indicating the type of *any*.

type	name	type	name
integer`	integertype	dictionary	dicttype
real	realtype	operator	operatortype
boolean	booleantype	file	filetype
array	arraytype	mark	marktype
packed array	packedarraytype	null	nulltype
string	stringtype	save	savetype
name	nametype	fontID	fonttype

name is executable.
Errors - stackunderflow

cvlit

any **cvlit** any

makes *any* literal (non-executable).
Errors - stackunderflow

cvx

any **cvx** any

makes *any* executable.
Errors - stackunderflow

xcheck

any **xcheck** bool

returns *true* if the object is executable, *false* if it is literal.
Errors - stackunderflow

executeonly

obj **executeonly** obj

reduces the access attribute of an array, packed array, file or string object to *execute only*, and returns the modified object. Henceforth the object cannot be read or altered. The access attributes of any objects sharing the value of *obj* are not affected. **executeonly** cannot change an object's access attribute if it has been set to *none.*

Errors - invalidaccess, stackunderflow, typecheck

noaccess

obj **noaccess** obj

sets the access attribute of an array, packed array, file, dictionary or string object to *none*, and returns the modified object. Henceforth the object cannot be read, altered or executed. If *obj* is a dictionary, the access attributes of any dictionaries sharing the value of *obj* are also set to *none*. For array, packed array, file or string objects, the access attributes of any objects sharing the value of *obj* are not affected.

Errors - invalidaccess, stackunderflow, typecheck

readonly

obj **readonly** obj

reduces the access attribute of an array, packed array, file, dictionary or string object to *read only*, and returns the modified object. Henceforth the object cannot be altered. If *obj* is a dictionary, the access attributes of any dictionaries sharing the value of *obj* are also set to *read only*. For array, packed array, file or string objects, the access attributes of any objects sharing the value of *obj* are not affected. **readonly** cannot change an object's access attribute if it has been set to *execute only* or *none*.

Errors - invalidaccess, stackunderflow, typecheck

rcheck

obj **rcheck** bool

returns *true* if the array, packed array, file, dictionary or string object's access attribute allows reading of the object (i.e. the access attribute has not been set to *execute only* or *none*), and *false* otherwise.

Errors - stackunderflow, typecheck

wcheck

obj **wcheck** bool

returns *true* if the array, packed array, file, dictionary or string object's access attribute allows writing to the object (i.e. the access attribute is *unlimited*), and *false* otherwise.

Errors - stackunderflow, typecheck

cvi

obj **cvi** int

converts a number or string to the equivalent integer. If *obj* is an integer, its value is returned unchanged. If *obj* is a real number, it is converted to an integer by truncation towards 0. If *obj* is a string whose characters represent a legal TrueImage number, it is converted to the equivalent number, which, if real, is converted to an integer by truncation towards 0.

If *obj* is a string whose characters do not represent a legal number, a **typecheck** error is executed. If a real number is too large to be represented as an integer, a **rangecheck** error is executed. (**round**, **truncate**, **ceiling** and **floor** remove fractional parts without converting a number's type).

Errors - invalidaccess, rangecheck, stackunderflow, syntaxerror, typecheck, undefinedresult

cvn

string **cvn** name

converts a string operand to a name comprising the same characters as the string. If the string is executable, the name is made executable.

Errors - invalidaccess, rangecheck, stackunderflow, typecheck

cvr

obj **cvr** real

converts a number or string to the equivalent real number. If *obj* is a real number, its value is returned unchanged. If *obj* is an integer, it is converted to real. If *obj* is a string whose characters represent a legal TrueImage number, it is converted to the equivalent number, which, if integer, is converted to a real number.

If *obj* is a string whose characters do not represent a legal number, a **typecheck** error is executed.

Errors - invalidaccess, rangecheck, stackunderflow, syntaxerror, typecheck, undefinedresult

cvrs

num radix string **cvrs** substring

converts a number to an equivalent string representation in the specified radix, writes it to *string*, and returns the initial substring of *string* that holds the number representation. If *num* is a real number, it is first converted to an integer by truncation towards 0. The initial part of *string* is overwritten by **cvrs**. Digits above 9 are represented by the letters A – Z. *radix* is a positive decimal integer between 2 and 36.

If *string* is too small to hold the number's representation, a **rangecheck** error is executed.

Errors - invalidaccess, rangecheck, stackunderflow, typecheck

cvs

any string **cvs** substring

converts an object *any* to an equivalent string representation, writes it to *string*, and returns the initial substring of *string* that holds the object's string representation. The initial part of *string* is overwritten by **cvs**.

If *any* is a number, **cvs** returns a string representation of the number. If *any* is a boolean, **cvs** returns either the string *true* or *false*. If *any* is a string, **cvs** simply copies its contents into *string*. If *any* is a name or an operator, **cvs** returns the text representation of the name or operator name. If *any* is of any other type, **cvs** returns the string (*--nostringval--*).

If *string* is too small to hold the result, a **rangecheck** error is executed.

Errors - invalidaccess, rangecheck, stackunderflow, typecheck

6.8.13 Font operators

definefont

key font **definefont** font

associates the font dictionary *font* with *key* (usually a name) in **FontDirectory**. **definefont** checks that *font* contains all necessary key-value pairs, and adds a key, FID, and corresponding FontID value. The dictionary must be large enough to hold this extra key-value pair. The dictionary's access is set to *read only.*

Errors - dictfull, invalidfont, stackunderflow, typecheck

findfont

key **findfont** font

returns the font dictionary associated with *key* in **FontDirectory**.

Errors - invalidfont, stackunderflow, typecheck

makefont

$font_1$ matrix **makefont** $font_2$

returns a new font whose characters are the characters of $font_1$, transformed by *matrix*. **makefont** creates a copy of $font_1$'s dictionary and then multiplies its **FontMatrix** value by *matrix*. Printing characters with the new font yields the same results as would be achieved by multiplying the CTM by *matrix*, and then printing using $font_1$.

Errors - stackunderflow, typecheck

scalefont

$font_1$ scale **scalefont** $font_2$

returns a new font whose characters are the characters of $font_1$, scaled by a factor of *scale*. **makefont** creates a copy of $font_1$'s dictionary and then multiplies its **FontMatrix** value by *scale*. Printing characters with the new font yields the same results as would be achieved by multiplying the CTM by *scale*, and then printing using $font_1$.

Errors - invalidfont, stackunderflow, typecheck, undefined

setfont

font **setfont** -

selects the current font. *font* must be a valid font dictionary returned by **findfont, scalefont** or **makefont**.

Errors - stackunderflow, typecheck

currentfont

- **currentfont** font

returns the current graphics state's current font dictionary.

Errors - stackoverflow

show

string **show** -

prints the string on the current page, starting from the current point, and using the current font. Character spacing is determined by each individual character's width. When the string has been printed, the current point is adjusted by the sum of the widths of the characters in *string*. If no current point has been set, a **nocurrentpoint** error is executed.

Errors - invalidaccess, invalidfont, nocurrentpoint, stackunderflow,
typecheck

ashow

x y string **ashow** -

performs the same function as **show**, except that the width of each of the string's characters is modified by adding *x* to its x-width and *y* to its y-width. This allows the spacing between characters to be modified. *x* and *y* are specified in user space coordinates, not in character coordinates.

Errors - invalidaccess, invalidfont, nocurrentpoint, stackunderflow,
typecheck

widthshow

x y char string **widthshow** -

performs the same function as **show**, except that the width of each occurrence of *char* in the string is modified by adding *x* to its x-width and *y* to its y-width. This modifies the spacing between *char* and the character following. *char* is a character code value in the range 0 – 255. *x* and *y* are specified in user space coordinates, not in character coordinates.

Errors - nocurrentpoint, stackunderflow, typecheck

awidthshow

x_1 y_1 char x_2 y_2 string **awidthshow** -

combines the functions of **ashow** and **awidthshow**, modifying the width of each of *string*'s characters by adding x_2 to its x-width and y_2 to its y-width, and modifying the width of each occurrence of *char* in the string by adding x_1 to its x-width and y_1 to its y-width. This allows the spacing between characters to be modified, and the spacing between *char* and the character following to be modified independently. x_1, y_1, x_2 and y_2 are specified in user space coordinates, not in character coordinates.

Errors - invalidaccess, invalidfont, nocurrentpoint, stackunderflow,
typecheck

kshow

proc string **kshow** -

performs the same function as **show**, except that *proc* is executed in between the printing of each successive pair of characters in *string*. The two characters (the one that has just been printed and the one about to be printed) are pushed onto the stack prior to each invocation of *proc* so that *proc* may make use of them. As each character is printed, the current point is updated by the character's width.

proc may alter the graphics state.

If *proc* does not make use of or dispose of the characters, they build up on the stack.

Errors - invalidaccess, invalidfont, nocurrentpoint, stackunderflow, typecheck

stringwidth

string **stringwidth** x y

returns the change in the current point that would result if *string* were printed using **show**. *x* and *y* are specified in user space coordinates. **stringwidth** may place characters in the font cache, if it executes their descriptions.

Errors - invalidaccess, invalidfont, stackunderflow, typecheck

FontDirectory

- **FontDirectory** dict

pushes **FontDirectory** onto the operand stack. **FontDirectory** is the directory which associates keys with font directories and which contains the names of all fonts present in virtual memory. **FontDirectory** has read-only access, however, **definefont** can modify it.

Errors - stackoverflow

StandardEncoding

- **StandardEncoding** array

pushes the standard encoding vector onto the operand stack. The standard encoding vector is a 256-element array, indexed by character code, that holds the character names corresponding to each of the codes.

Errors - stackoverflow

6.8.14 Font cache operators

cachestatus

- **cachestatus** bsize bmax msize mmax csize cmax blimit

returns current consumption and maximum space available for the following: bytes of bitmap storage (*bsize* and *bmax*), font/matrix combinations (*msize* and *mmax*), and number of cached characters (*csize* and *cmax*), plus the maximum number of bits that may be used to cache a single character, *blimit*.

Errors - stackoverflow

setcachedevice

x y ll_x ll_y ur_x ur_y **setcachedevice** -

executed by a user-defined font's **BuildChar** procedure, prior to the definition and rendition of a character. **setcachedevice** requests the interpreter to place the character whose shape is rendered by the procedures which follow, in the font cache (if possible) and on the current page. The interpreter uses the information specified to decide whether to store the character in the cache, and to render it on the page.

The operands are all specified in character coordinate system units. x and y specify the characters width, ll_x, ll_y, ur_x and ur_y specify the lower-left and upper-right corners respectively of the character's bounding box.

Errors - stackunderflow, typecheck, undefined

setcharwidth

x y **setcharwidth** -

functions in the same way as **setcachedevice**, passing the interpreter the character's width, but designating that the character should not be stored in the cache. **setcharwidth** should be executed instead of **setcachedevice** when **BuildChar** is to execute **setgray**, **setrgbcolor**, **sethsbcolor**, **settransfer** or **image**.

Errors - stackunderflow, typecheck, undefined

setcachelimit

num **setcachelimit** -

sets the maximum number of bytes that may be used to cache the bitmap of a single character. Any character larger than this will not be cached; its description will be executed each time it is encountered. Characters already in the font cache are not affected.

Errors - limitcheck, rangecheck, stackunderflow, undefinedfilename

setcacheparams

mark size lower upper **setcacheparams** -

sets the cache parameters to the values specified by the integer objects above the topmost mark on the stack. All objects down to the topmost mark are popped from the stack after execution. The number of cache parameters may vary. If more than three parameters are specified, the topmost three are used and the rest are ignored. If fewer than three parameters are specified, default values are substituted.

upper is the maximum number of bytes that may be used to cache the pixel array of a single character; the same parameter may also be set by **setcachelimit**.

lower specifies a threshold size in bytes, above which characters may be stored in compressed form. If *lower* = 0, all characters will be compressed. If *lower* is greater than or equal to *upper*, compression is disabled.

size sets the new size of the font cache in bytes (equivalent to the *bmax* parameter set by **cachestatus**). If *size* is not specified, the current cache size is retained. If *size* is not within the range of permissable font cache sizes, the nearest valid size is used instead. Reducing the font cache size may cause some characters that are presently cached to be discarded.

Errors - rangecheck, typecheck, unmatchedmark

currentcacheparams

- **setcachelimit** mark size lower upper

pushes a mark object onto the stack, followed by the current cache parameter settings. The cache parameters are as described above under **setcacheparams**; the number of cache parameters may vary.

Errors - stackoverflow

6.8.15 File operators

file

string$_1$ string$_2$ **file** file

creates a file object for the file specified by *string$_1$*. The access type is specified by *string$_2$*: 'r' specifies an input (read-only) file, 'w' an output (write-only) file. The file remains available for reading or writing until either it is closed with **closefile**, an end-of-file character is read, or a **restore** is encountered whose corresponding **save** was performed before the **file** that created the file object. *%stdin* and *%stdout* are the standard input and output files.

Errors - invalidfileaccess, limitcheck, stackunderflow, typecheck,
undefinedfilename

closefile

file **closefile** -

closes a file, breaking the association between the file object and the file itself. If the file is an output file, any buffered characters are immediately transmitted before the file is closed

Errors - ioerror, stackunderflow, typecheck

read

file **read** int true
file **read** false

reads a character from an input file, returning the integer representation of the character and *true*, unless end-of-file is encountered, in which case **read** returns *false*.

If a parity or checksum error occurs, an **ioerror** is executed.

Errors - invalidaccess, ioerror, stackoverflow, stackunderflow, typecheck

write

file int **write** -

appends a character to an output file *file*. *int* is the integer representation of the character and should be in the range 0 to 255. If it is greater than 255, the value of *int* modulo 256 is used.

If the file is not a valid output file, or some other error is detected, an **ioerror** is executed.

Errors - invalidaccess, ioerror, stackunderflow, typecheck

readhexstring

file string **readhexstring** substring bool

reads pairs of hexadecimal digits from *file*, writing them into *string*, starting at the beginning of the string. Reading continues until either the string is full or an end-of-file is encountered. **readhexstring** returns the newly-written substring of *string*, plus *true* if *string* was filled, or plus *false* if an end-of-file was encountered before *string* could be filled. Characters other than 0 – 9 and A – F (or a – f) are ignored.

Errors - invalidaccess, ioerror, rangecheck, stackunderflow, typecheck

writehexstring

file string **writehexstring** -

writes the characters of *string* to *file* as hexadecimal digits, starting from the beginning of the string. **writehexstring** converts each character-code integer in *string* to a pair of hexadecimal digits (0 – 9 or a – f) and appends the digits to the file.

Errors - invalidaccess, ioerror, stackunderflow, typecheck

readstring

file string **readstring** substring bool

reads characters from *file*, writing them into *string*, starting at the beginning of the string. Reading continues until either the string is full or an end-of-file is encountered. **readstring** returns the newly-written substring of *string*, plus *true* if string was filled, or plus *false* if an end-of-file was encountered before *string* could be filled. Characters read from *file* are all regarded simply as integers in the range 0 – 255. None are regarded as control codes.

Errors - invalidaccess, ioerror, rangecheck, stackunderflow, typecheck

writestring

file string **writestring** -

writes the characters of *string* to *file*, starting from the beginning of the string. **writestring** does not append a *newline* to the file.

Errors - invalidaccess, ioerror, stackunderflow, typecheck

readline

file string **readline** substring bool

reads a line of characters terminated by a *newline* character from *file*, and writes them into *string*, starting at the beginning of the string. **readstring** returns the newly-written substring of *string*, plus *true* if a *newline* character was present, plus *false* if an end-of-file was encountered before a *newline* character was read. The *newline* is not written to the string. If *string* is filled before a *newline* is read, a **rangecheck** error is executed.

Errors - invalidaccess, ioerror, rangecheck, stackunderflow, typecheck

token

file **token** any true
file **token** false

reads characters from *file*, searching for a token that represents a TrueImage object. If **token** can read an object token from *file*, it returns the object and *true*. If **token** cannot read an object token from *file*, it returns *false*. (If **token** encounters an end-of-file without reading any non-whitespace characters, it also closes the file).

The object can be a number, name, string, data array or executable array. The object is the same as the object that would be returned if the file were executed directly, however, the object is not executed, merely pushed onto the operand stack.

Only the first object encountered is returned. To parse the whole file, repeated use of **token** would be necessary.

token discards all characters up to the final character of the token. If the token is a name or number, the first following whitespace character is discarded as well. If the token is a string or array ending with a), >,] or }, that character (but no following characters) is discarded.

Errors - invalidaccess, ioerror, rangecheck, stackunderflow, stackoverflow, syntaxerror, typecheck, undefinedresult

bytesavailable

file **bytesavailable** int

returns the number of bytes available to be read immediately from *file*. −1 is returned if end-of-file has been encountered or if the number cannot be established.

Errors - ioerror, stackunderflow, typecheck

flush

- flush -

immediately sends any buffered characters to the standard output file.

Errors - ioerror

flushfile

file **flushfile** -

If *file* is an output file, **flushfile** immediately sends any buffered characters to it. If *file* is an input file, **flushfile** reads characters from the file until it encounters an end-of-file.

Errors - ioerror, stackunderflow, typecheck

resetfile

file **resetfile** -

disposes of any buffered characters associated with *file*. If *file* is an input file, **resetfile** discards any characters that have been received from the file, but have not yet been processed. If *file* is an output file, **resetfile** discards any characters that have been written to *file*, but not yet transmitted.
Errors - stackunderflow, typecheck

status

file **status** bool

returns *true* if *file* is still available for reading or writing, *false* otherwise.
Errors - stackunderflow, typecheck

run

string **run** -

reads and executes the contents of the file specified by *string* as a TrueImage program. **run** closes the file on encountering an end-of-file or a **stop** operator. If an **exit** is encountered, an **invalidexit** error is executed.
Errors - ioerror, limitcheck, stackunderflow, typecheck, undefinedfilename

currentfile

- **currentfile** file

returns the file object from which the interpreter has most recently read program input, the top file on the execution stack.
If the last token read by the interpreter was a name or number followed by white space, characters can now be read starting from the character after the whitespace character immediately following the name or number. If the last token read stood for any other object, characters can be read starting from the character immediately after the token.
The file returned is usually the default input file.
Errors - stackoverflow

print

string **print** -

writes *string* to the standard output file, enabling text to be sent to a host computer.
Errors - stackunderflow, typecheck

=

any = -

writes a text representation of the value of a number, boolean, string, name or operator object to the standard output file, and '-nostringval-' for any other object.

Errors - stackunderflow

stack

- **stack** -

performs the same function as the = operator, but for each object on the stack.

Errors - stackoverflow

==

any == -

writes a text representation of the value of an object to the standard output file. Literal names are preceded by /. Strings, arrays and packed arrays are shown in their entirety, enclosed within (),[] and { }. Type names of unprintable types are shown (see the **type** operator on page 256), and operator names are shown as follows: *--opname--*.

Errors - stackunderflow

pstack

- **pstack** -

performs the same function as the == operator, but for each object on the stack.

Errors - stackoverflow

prompt

- **prompt** -

prompts the user for the next statement (only in an interactive environment).

echo

boolean **echo** -

If boolean = *true*, characters are echoed from the standard input file to the standard output file (in an interactive environment). If boolean =*false*, characters are not echoed.

Errors - stackunderflow, typecheck

6.8.16 Virtual memory operators

save

- **save** save

saves the state of virtual memory, returning a save object, and pushes a copy of the graphics state onto the graphics state stack.

Errors - limitcheck, stackoverflow

restore

save **restore** -

restores the saved virtual memory state described by *save* and pops the graphics state from the top of the graphics state stack. A save object may only be restored once: *save* and any more recently created save objects are discarded. If the operand, dictionary or execution stacks contain array, dictionary, file, name, save or string objects newer than the save object being restored, an **invalidrestore** error is executed.

Errors - invalidrestore, rangecheck, stackunderflow, typecheck

vmstatus

- **vmstatus** level used maximum

describes the state of TrueImage virtual memory. *level* is the current number of saved VM states, *used* the number of bytes used so far, and *maximum* the maximum number of bytes available.

Errors - stackoverflow

6.8.17 Miscellaneous operators

bind

proc **bind** proc

replaces the executable operator names in a procedure by their values. If a name is not found, or its value is not an operator, no action is taken for that name. For elements of *proc* that are procedures with unlimited access, bind performs the same process on them, and then sets their access to *read only.* **bind** is used to ensure that a procedure will execute the operator definitions it was intended to, and to make it run faster.

Errors - typecheck

null

- **null** null

pushes a null object onto the stack.

Errors - stackoverflow

usertime

- **usertime** int

returns the current value of a clock counter that counts in milliseconds.

Errors - stackoverflow

executive

- **executive** -

invokes the interactive executive, enabling the user to address the TrueImage interpreter directly using a terminal program. **executive** makes use of the **%statementedit** file to obtain commands from the user. If echo has been turned on with the **echo** operator, commands are echoed to the user's terminal as the user enters them.

Errors - undefined

version

- **version** string

returns a string detailing the version of the TrueImage language and interpreter being used.

Errors - stackoverflow

gsave

- **gsave** -

saves the current graphics state, pushing it onto the graphics state stack.

Errors - limitcheck

grestore

- **grestore** -

restores the graphics state saved with the most recent **gsave** command, popping it off the top of the graphics state stack. If no **gsave** has been executed, or if the most recent **gsave** came before a **save** whose VM state has not yet been restored, **grestore** restores the graphics state on top of the graphics state stack without popping it.

grestoreall

- **grestoreall** -

pops graphics states off the graphics state stack until it reaches either the bottommost graphic state, or a state saved by a **save**. This is then made the current graphics state, but is not popped from the stack.

initgraphics

- **initgraphics** -

sets the following graphics state settings to their default values

CTM	default for printer	line width	1 user unit
path	empty	line cap	butt caps
position	undefined	line join	mitered
clipping path	default for printer	line dash	solid
color	black	miter limit	10

setlinewidth

num **setlinewidth** -

sets the line width for the current graphics state to *num*. This determines the thickness of lines generated by **stroke**. If scaling is unequal in the x- and y-directions, a line's thickness will vary according to its orientation. A line width of 0 specifies the thinnest possible line.
Errors - stackunderflow, typecheck

currentlinewidth

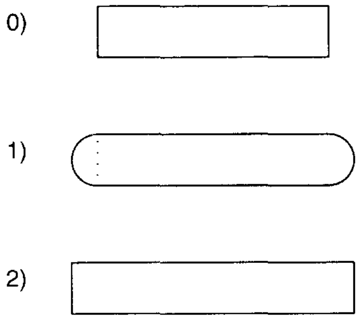
- **currentlinewidth** num

returns the current line width in the current graphics state.
Errors - stackoverflow

setlinecap

int **setlinecap** -

sets the line cap type for the current graphics state. This determines the shape of the end of open subpaths rendered by **stroke**. 0 selects butt cap (the stroke is cut off at the subpath's endpoint), 1 selects round cap (projecting semi-circular line ends), and 2 selects square cap (projecting squared line ends).



Errors - rangecheck, stackunderflow, typecheck

currentlinecap

- ***currentlinecap*** int

returns the current line cap setting in the current graphics state.

Errors - stackoverflow

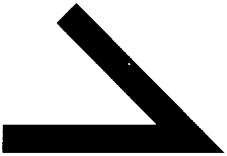
setlinejoin

int **setlinejoin** -

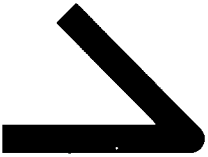
sets the line join type for the current graphics state. This determines the shape of the corners of paths rendered by **stroke**. 0 selects mitered join (the outside edges of the converging lines are extended until they meet), 1 selects round join (rounded circular line joins), and 2 selects bevel join (a straight-line angular join).

If a mitered join length would exceed the miter limit, a beveled join is used instead.

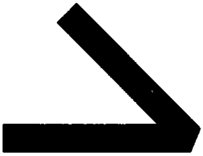
The line join type is only applied to consecutive segments of paths.



0- Miter



1 - Round



2 - Bevel

Errors - rangecheck, stackunderflow, typecheck

currentlinejoin

- **currentlinejoin** int

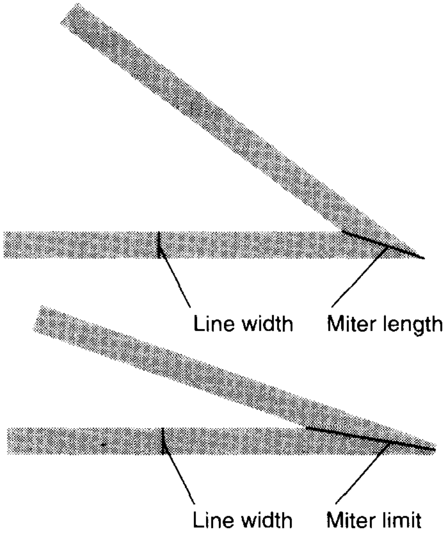
returns the current line join setting in the current graphics state.

Errors - stackoverflow

setmiterlimit

num **setmiterlimit** -

sets the miter limit for the current graphics state. Miter length is the length of the spike produced by two lines that join at an angle. Miter limit is the maximum allowed ratio of miter length to line width. If mitered line joins are selected, but the miter limit would be exceeded, a beveled join is used instead. Setting the miter limit to 1 causes all mitered joins to be beveled instead.



If the miter length exceeds the miter limit, the line join is beveled instead

Errors - rangecheck, stackunderflow, typecheck

currentmiterlimit

- **currentmiterlimit** num

returns the current miter limit in the current graphics state.
Errors - stackoverflow

setdash

array offset **setdash** -

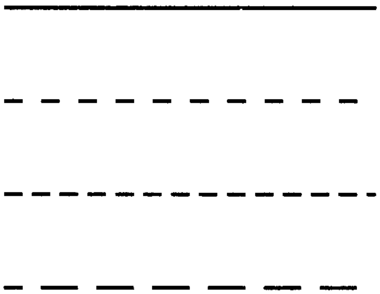
sets the current line dash pattern for the current graphics state. The dash pattern is specified by an array of numbers that specify alternating lengths of line and spacing. A single number defines a dash pattern that alternates equal lengths of line and spacing. The numbers in *array* should be non-negative and should not all be 0. If *array* is empty, lines are solid. Dash lengths are in user units.

offset specifies an initial length of the pattern to be skipped when stroking of a subpath commences.

The dash pattern is used cyclically; when **stroke** reaches the end of the pattern it starts again from the beginning.

Each subpath is stroked separately; the dash pattern restarts from the beginning (or from *offset*).

```
2 setlinewidth
[] 0 setdash
100 200 moveto 299 200 lineto
stroke
[10 10] 0 setdash
100 150 moveto 299 150 lineto
stroke
[10 5] 0 setdash
100 100 moveto 299 100 lineto
stroke
[20 10] 10 setdash
100 50 moveto 299 50 lineto
stroke
```



Errors - limitcheck, stackunderflow, typecheck

currentdash

- **currentdash** array offset

returns the current dash pattern in the current graphics state.
Errors - stackoverflow

setflat

num **setflat** -

sets the flatness setting for the current graphics state. Flatness is a measure of how smooth or jerky rendered curved line segments are. All curved lines are made up of sequences of small straight lines. The more straight lines that are used, the smoother a curve is.
For small values of *num*, higher numbers of straight lines are used, and hence curves appear smoother. However, this can consume large amounts of virtual memory.
num can range from 0.2 to 100.
Errors - stackunderflow, typecheck

currentflat

- **currentflat** num

returns the current flatness setting in the current graphics state.
Errors - stackoverflow

setgray

num **setgray** -

sets the color parameter in the current graphics state to a specified gray scale. Subsequent lines and shapes are stroked in the selected shade. *num* ranges from 0 (black) to 1 (white). Values in between represent varying shades of gray.
Errors - stackunderflow, typecheck, undefined.

currentgray

- **currentgray** num

returns the current gray value of the current color in the current graphics state. If the current color is not black, the current color's brightness component is returned.
Errors - stackoverflow

sethsbcolor

hue saturation brightness **sethsbcolor** -

sets the hue, saturation and brightness of the color parameter in the current graphics state to the specified values. Each number can range from 0 to 1. On a color device subsequent lines and shapes are stroked in the selected color.

Errors - stackunderflow, typecheck, undefined.

currenthsbcolor

- **currenthsbcolor** hue saturation brightness

returns the hue, saturation and brightness components of the current color in the current graphics state.

Errors - stackoverflow

setrgbcolor

red blue green **setrgbcolor** -

sets the red, green and blue components of the color parameter in the current graphics state to the specified values. Numbers can range from 0 to 1. On a color device subsequent lines and shapes are stroked in the selected color.

Errors - stackunderflow, typecheck, undefined.

currentrgbcolor

- **currentrgbcolor** red blue green

returns the red, blue and green components of the current color in the current graphics state.

Errors - stackoverflow

setscreen

freq angle proc **setscreen** -

sets the current half-tone screen settings in the current graphics state. *freq* specifies the number of half-tone cells per device-space inch, *angle* specifies the angle of the screen to the device space coordinate system, and *proc* is a procedure that defines the combination of white and black pixels for any gray setting.

Errors - limitcheck, rangecheck, stackunderflow, typecheck

currentscreen

- **currentscreen** freq angle proc

returns the current halftone screen settings in the current graphics state.

Errors - stackoverflow

settransfer

proc **settransfer** -

sets the current transfer function for the current graphics state. *proc* is a procedure that takes a number in the range 0 to 1 as input and returns a number in the same range. *proc* maps TrueImage gray levels set by **setgray** to printer gray levels.

Errors - stackunderflow, typecheck

currenttransfer

- **currenttransfer** proc

returns the current transfer function in the current graphics state.

Errors - stackoverflow

6.8.18 Coordinate operators

matrix

- **matrix** matrix

pushes a 6-element identity matrix [1.0 0.0 0.0 1.0 0.0 0.0] onto the stack.
Errors - stackoverflow

initmatrix

- initmatrix -

sets the CTM to the default value for the printer. The effect of this is to restore the default user space-to-device space mapping.

identmatrix

matrix **identmatrix** matrix

converts *matrix* to the identity matrix, [1.0 0.0 0.0 1.0 0.0 0.0], which maps any point to itself.
Errors - rangecheck, stackunderflow, typecheck

defaultmatrix

matrix **defaultmatrix** matrix

converts *matrix* to the printer's default transformation matrix.
Errors - rangecheck, stackunderflow, typecheck

currentmatrix

matrix **currentmatrix** matrix

converts *matrix* to the current CTM.
Errors - rangecheck, stackunderflow, typecheck

setmatrix

matrix **setmatrix** -

makes *matrix* the current CTM. Normally the CTM will be modified using the **rotate**, **translate** and **scale** operators instead.
Errors - rangecheck, stackunderflow, typecheck

translate

x y **translate** -

x y matrix **translate** matrix

If there is no *matrix* operand, **translate** modifies the CTM, repositioning the origin of the user space coordinate system at (x,y) relative to its present position. This is equivalent to multiplying the CTM by a matrix

1 0 0
0 1 0
x y 1

If there is a *matrix* operand, **translate** sets its value to that of the matrix shown above, and does not alter the CTM.

Errors - rangecheck, stackunderflow, typecheck

scale

x y **scale** -

x y matrix **scale** matrix

If there is no *matrix* operand, **scale** modifies the CTM, scaling the user space coordinate system units by *x* and *y* relative to their current size. The user space origin and rotation are not changed. This is equivalent to multiplying the CTM by a matrix

x 0 0
0 y 0
0 0 1

If there is a *matrix* operand, **scale** sets its value to that of the matrix shown above, and does not alter the CTM.

Errors - stackunderflow, typecheck

rotate

angle **rotate** -

angle matrix **rotate** matrix

If there is no *matrix* operand, **rotate** modifies the CTM, rotating the user space coordinate system counterclockwise by *angle* degrees. The user space origin and the size of its units are not changed. This is equivalent to multiplying the CTM by a matrix

cos(angle) sin(angle) 0
–sin(angle) cos(angle) 0
0 0 1

If there is a *matrix* operand, **rotate** sets its value to that of the matrix shown above, and does not alter the CTM.

Errors - stackunderflow, typecheck

concat

matrix **concat** -

modifies the CTM by multiplying it by *matrix.*

Errors - stackunderflow, typecheck

concatmatrix

matrix_1 matrix_2 matrix_3 **concatmatrix** matrix_3

sets *$matrix_3$* to the value obtained by multiplying *$matrix_2$* by *$matrix_1$*.

Errors - stackunderflow, typecheck

transform

x_1 y_1 **transform** x_2 y_2

x_1 y_1 matrix **transform** x_2 y_2

If there is no *matrix* operand, **transform** returns the current device space coordinates of the user space point (x_1,y_2) according to the current CTM. If *matrix* is supplied, **transform** transforms the point using *matrix* instead.

Errors - stackunderflow, typecheck

dtransform

dx_1 dy_1 **dtransform** dx_2 dy_2

dx_1 dy_1 matrix **dtransform** dx_2 dy_2

If there is no *matrix* operand, **dtransform** returns the device space equivalent of the user space distance vector (dx_1, dy_2), transforming it by the current CTM. If *matrix* is supplied, **dtransform** transforms the distance vector using *matrix* instead.

Errors - stackunderflow, typecheck

itransform

x_1 y_1 **itransform** x_2 y_2

x_1 y_1 matrix **itransform** x_2 y_2

If there is no *matrix* operand, **itransform** returns the current user space coordinates of the device space point (x_1,y_2), transforming it by the inverse of the current CTM. If *matrix* is supplied, **itransform** transforms the point using the inverse of *matrix* instead.

Errors - stackunderflow, typecheck, undefinedresult

idtransform

dx_1 dy_1 **idtransform** dx_2 dy_2

dx_1 dy_1 matrix **idtransform** dx_2 dy_2

If there is no *matrix* operand, **idtransform** returns the user space equivalent of the device space distance vector (dx_1, dy_2), transforming it by the inverse of the current CTM. If *matrix* is supplied, **idtransform** transforms the distance vector using the inverse of *matrix* instead.

Errors - stackunderflow, typecheck, undefinedresult

invertmatrix

matrix_1 matrix $_2$ **invertmatrix** matrix_2

sets *$matrix_2$* to the inverse of *$matrix_1$*.

Errors - stackunderflow, typecheck, undefinedresult

6.8.19 Device set-up operators

showpage

- showpage -

causes the current page to be printed out, and then performs **erasepage** and **initgraphics** to prepare the next page. **showpage** looks up the name *#copies* in the dictionary stack, and prints the number of copies specified.

copypage

- copypage -

causes one copy of the current page to be printed out. **copypage** is intended primarily for debugging use.

framedevice

matrix width height proc **framedevice** -

installs a frame buffer as raster memory for an output device. The frame buffer is 8 x *width* pixels wide and *height* pixels high. *matrix* is made the current CTM. *proc* is a procedure to be executed by **showpage** and **copypage** to transmit the contents of the frame buffer to the device.

Errors - stackunderflow, typecheck

nulldevice

- nulldevice -

makes the "null device" the current output device. Stroking and painting operators do not mark the current page. **showpage** and **copypage** have no effect.

6.8.20 LS-5TT-specific operators

Operators marked with an asterisk (*) are defined in the **statusdict** dictionary. To use these operators, precede them with the TrueImage program statement:

```
statusdict begin
```

This will enable your program to use them.

setdojamrecovery *

bool **setdojamrecovery** -

turns jam recovery on (true) or off (false). If jam recovery is on, pages that get jammed will be reprinted when the jam has been cleared; if off (the factory default), the print job is abandoned. Jam recovery may reduce throughput.
Errors - stackunderflow, typecheck

dojamrecovery *

- **dojamrecovery** bool

returns the jam recovery setting: on (true) or off (false).
Errors - stackoverflow, typecheck

setdorep *

bool **setdorep** -

turns resolution enhancement (300×600 dots per inch) on (true) or off (false). The factory default setting is off.
Errors - stackunderflow, typecheck

dorep *

- **dorep** bool

returns the resolution enhancement setting: on (true) or off (false).
Errors - stackoverflow, typecheck

settray *

traynum **settray** -

selects the tray from which to feed paper. Valid values of *traynum* are as follows:

traynum	tray
0	front tray
1	cassette
2	auto selection
3	lower cassette

Errors - stackunderflow, typecheck

papertray *

- **papertray** traynum

returns an integer whose value indicates the current tray selection. Values for *traynum* are as for **settray** above.
Errors - stackoverflow, typecheck

traysup *

traynum **traysup** bool

indicates whether a particular tray is available (true) or not (false). Values for *traynum* are as for **settray** above.
Errors - stackunderflow, typecheck

ppapersize *

traynum **ppapersize** papersize

indicates the size of paper in the specified tray. Values for *traynum* are as for **settray** above. Values for *papersize* are as follows:

papersize	size	*papersize*	size
0	Letter	5	Monarch
1	Legal	6	Com-10
2	A4	7	DL
3	Executive	8	C5
4	B5		

Errors - stackunderflow, typecheck

setpapertray *

traynum **setpapertray** -

selects the tray from which to feed paper, and sets the clipping path (imageable area) according to the size of the paper in the selected tray. Values for *traynum* are as for **settray** above.

Errors - stackunderflow, typecheck

findtray *

papersize **findtray**

seraches for a tray containing paper of size *papersize*. If one is found, the tray is selected as the current tray and the imageable area (clipping path) is set according to the paper size specified by *papersize*. Values for *papersize* are as for **ppapersize** above.

Errors - stackunderflow, typecheck

executivepage

- **executivepage** -

sets a page size of 7.25" by 10.50" and an imageable area (clipping path) of 6.72" by 10.00" centered on the page.

com10envelope

- com10envelope -
sets a page size of 4.125" by 9.50" and an imageable area (clipping path) of 3.63" by 9.00" centered on the page.

monarcenvelope

- monarcenvelope -
sets a page size of 3.875" by 7.50" and an imageable area (clipping path) of 3.41" by 7.00" centered on the page.

c5envelope

- c5envelope -
sets a page size of 6.38" by 9.01" and an imageable area (clipping path) of 5.87" by 8.51" centered on the page.

dlenvelope

- dlenvelope -
sets a page size of 110mm by 220mm and an imageable area (clipping path) of 97.54mm by 207.3mm centered on the page.

setemulation

emulation **setemulation -**
switches the printer to the selected emulation. Valid values of *emulation* are as follows

Value	Emulation
0	HP LaserJet III
5	TrueImage

Other values are ignored.

lettertray

- lettertray -
causes the printer to search for a tray containing Letter-sized paper. If one is found, page size is set to Letter, and the tray is selected as the current tray. If no tray containing Letter paper is found, a **rangecheck** error is executed.
Errors - rangecheck

legaltray

- legaltray -

causes the printer to search for a tray containing Legal-sized paper. If one is found, page size is set to Legal, and the tray is selected as the current tray. If no tray containing Legal paper is found, a **rangecheck** error is executed.
Errors - rangecheck

a4tray

- a4tray -

causes the printer to search for a tray containing A4-sized paper. If one is found, page size is set to A4, and the tray is selected as the current tray. If no tray containing A4 paper is found, a **rangecheck** error is executed.
Errors - rangecheck

executivetray

- executivetray -

causes the printer to search for a tray containing Executive-sized paper. If one is found, page size is set to executive, and the tray is selected as the current tray. If no tray containing Executive paper is found, a **rangecheck** error is executed.
Errors - rangecheck

b5tray

- b5tray -

causes the printer to search for a tray containing B5-sized paper. If one is found, page size is set to B5, and the tray is selected as the current tray. If no tray containing B5 paper is found, a **rangecheck** error is executed.
Errors - rangecheck

monarcenvelopetray

- monarcenvelopetray -

causes the printer to search for a tray containing Monarch-sized envelopes. If one is found, page size is set to monarch, and the tray is selected as the current tray. If no tray containing Monarch envelopes is found, a **rangecheck** error is executed.
Errors - rangecheck

com10envelopetray

- com10envelopetray -

causes the printer to search for a tray containing Com-10-sized envelopes. If one is found, page size is set to Com-10, and the tray is selected as the current tray. If no tray containing Com-10 envelopes is found, a **rangecheck** error is executed.

Errors - rangecheck

dlenvelopetray

- dlenvelopetray -

causes the printer to search for a tray containing DL-sized envelopes. If one is found, page size is set to DL, and the tray is selected as the current tray. If no tray containing DL envelopes is found, a **rangecheck** error is executed.

Errors - rangecheck

c5envelopetray

- c5envelopetray -

causes the printer to search for a tray containing C5-sized envelopes. If one is found, page size is set to C5, and the tray is selected as the current tray. If no tray containing C5 envelopes is found, a **rangecheck** error is executed.

Errors - rangecheck

6.8.21 Errors

dictfull
dictionary is full

dictstackoverflow
dictionary stack is full

dictstackunderflow
dictionary stack is empty

execstackoverflow
execution stack is full

handleerror
a procedure that reports information about errors

interrupt
external interrupt detected

invalidaccess
object does not have requested access attribute

invalidexit
exit is not within a loop construct

invalidfileaccess
file operand access string is not acceptable

invalidfont
invalid font name or font dictionary encountered

invalidrestore
improper **restore** attempted

ioerror
input/output error

limitcheck
some implementation-specific limit exceeded

nocurrentpoint
current coordinate point has not been set (using **moveto** or **rmoveto**)

rangecheck
operand value exceeds implementation limits

stackoverflow
Operand stack is full

stackunderflow
Operand stack is empty

syntaxerror
Syntax error in TrueImage program code

timeout
time limit exceeded

typecheck
operand is of the wrong type for the operator

undefined
name not found

undefinedfilename
file not found

undefinedresult
value is too great or too small to be represented, or result is meaningless

unmatchedmark
operator cannot find mark in operand stack

unregistered
internal error

VMerror
Virtual memory is full

MEMO

Technical supplement

CHAPTER 7

This section provides summary lists of all commands available in the PCL5, GL2 and TrueImage languages, character code tables for all symbol sets available in HP LaserJet III mode, and samples of all internal fonts.

7.1 Command summary

7.1.1 Printer Control Language (PCL) commands

Command	Function	Page
<BS>	Backspace	68
<HT>	Horizontal tab	68
<LF>	Line feed	67
<FF>	Form feed	68
<CR>	Carriage return	67
<SO>	Select secondary font	80
<SI>	Select primary font	80
<SP>	Space	67
<ESC> & a *n* C	Horizontal cursor position (columns)	69
<ESC> & a *n* H	Horizontal cursor position (decipoints)	69
<ESC> & a *n* L	Set left margin	63
<ESC> & a *n* M	Set right margin	64
<ESC> & a *n* P	Print direction	72
<ESC> & a *n* R	Vertical cursor position (rows)	70
<ESC> & a *n* V	Vertical cursor position (decipoints)	70
<ESC> & d @	Turn underlining off	88
<ESC> & d *n* D	Turn underlining on	88
<ESC> & f *n* S	Push/pop cursor position	71
<ESC> & f *n* X	Macro control	114
<ESC> & f *n* Y	Macro ID	114
<ESC> & k *n* G	Line termination	73
<ESC> & k *n* H	Horizontal motion index	62
<ESC> & ℓ *n* A	Page size	56
<ESC> & ℓ *n* C	Vertical motion index	62
<ESC> & ℓ *n* D	Set line spacing	63
<ESC> & ℓ *n* E	Top margin	65
<ESC> & ℓ *n* F	Text length	66

Command	Function	Page
<ESC> & ℓ *n* H	Paper source	53
<ESC> & ℓ *n* L	Perforation skip	66
<ESC> & ℓ *n* O	Logical page orientation	60
<ESC> & ℓ *n* P	Page length	57
<ESC> & ℓ *n* U	Left offset registration	58
<ESC> & ℓ *n* X	Select number of copies	52
<ESC> & ℓ *n* Z	Top offset registration	59
<ESC> & p *n* X *<character data>*	Transparent print data	88
<ESC> & s *n* C	End of line wrap	73
<ESC> (3 @	Set primary font to default values	80
<ESC> (*n*	Select primary font symbol set	81
<ESC> (*n* X	Select primary font by ID number	80
<ESC> (s *n* B	Select primary font stroke weight	86
<ESC> (s *n* H	Set primary font pitch	83
<ESC> (s *n* P	Select primary font spacing type	82
<ESC> (s *n* S	Select primary font style	85
<ESC> (s *n* T	Select primary font typeface	87
<ESC> (s *n* V	Set primary font point size	84
<ESC> (s *n* W *<descripter and data>*	Send character descripter and data	98
<ESC>) 3 @	Set secondary font to default values	80
<ESC>) *n*	Select secondary font symbol set	82
<ESC>) *n* X	Select secondary font by ID number	81
<ESC>) s *n* B	Select secondary font stroke weight	87
<ESC>) s *n* H	Set secondary font pitch	83
<ESC>) s *n* P	Select secondary font spacing type	82
<ESC>) s *n* S	Select secondary font style	85
<ESC>) s *n* T	Select secondary font typeface	88
<ESC>) s *n* V	Set secondary font point size	84
<ESC>) s *n* W*<descripter>*	Send font descripter	92
<ESC> * b *n* M	Set compression mode	109
<ESC> * b *n* W *<data>*	Transfer raster data	111
<ESC> * b *n* Y	Set raster y-offset	109
<ESC> * c *n* A	Set rectangle width in dots	105
<ESC> * c *n* B	Set rectangle height in dots	106
<ESC> * c *n* D	Font ID	90
<ESC> * c *n* E	Send character code	98
<ESC> * c *n* F	Font control	90
<ESC> * c *n* G	Set area fill identity	104
<ESC> * c *n* H	Set rectangle width in decipoints	106
<ESC> * c *n* P	Draw filled rectangle	106
<ESC> * c *n* V	Set rectangle height in decipoints	106
<ESC> * p *n* X	Horizontal cursor position (dots)	69

Command	Function	Page
<ESC> * p *n* Y	Vertical cursor position (dots)	70
<ESC> * r B	End raster transfer	111
<ESC> * r *n* A	Start raster transfer	111
<ESC> * r *n* F	Set raster image orientation	108
<ESC> * r *n* S	Set raster area width	108
<ESC> * r *n* T	Set raster area height	108
<ESC> * t *n* R	Set raster resolution	107
<ESC> * v *n* N	Set source transparency	103
<ESC> * v *n* O	Set pattern transparency	103
<ESC> * v *n* T	Set pattern type	105
<ESC> 9	Clear horizontal margins	64
<ESC> =	Half line feed	71
<ESC> E	Reset	52
<ESC> Y	Display functions on	74
<ESC> Z	Display functions off	74
<ESC> [C *n*	Select feeder	53
<ESC> [E *n*	Change emulation	54
<ESC> [O *n*	Select orientation	61
<ESC> [S *n*	Select paper size	55
<ESC> z	Self test	74

7.1.2 GL2 commands

Command	Function	Page
<ESC> % *n* A	Enter PCL mode	122
<ESC> % *n* B	Enter GL2 mode	122
<ESC> * c 0 T	Set picture frame ancher point	121
<ESC> * c *n* K	Specify horizontal plot size	122
<ESC> * c *n* L	Specify vertical plot size	121
<ESC> * c *n* X	Set picture frame horizontal size	121
<ESC> * c *n* Y	Set picture frame vertical size	121
AA	Draw absolute arc	146
AC	Anchor corner	157
AD	Define alternate font	175
AR	Draw relative arc	148
AT	Draw absolute three point arc	147
CF	Character fill mode	185
CI	Draw circle	145
CP	Character plot	184
DF	Default values	129
DI	Absolute direction	181
DR	Relative direction	182
DT	Define label terminator	179
DV	Define variable text path	183
EA	Edge absolute rectangle	151
EP	Edge polygon	152
ER	Edge relative rectangle	151
ES	Extra space	191
EW	Edge wedge	152
FI	Select primary font	176
FN	Select secondary font	177
FP	Fill polygon	155
FT	Fill type	158
IN	Initialize	128
IP	Input scaling points	130
IR	Input relative scaling points	131
IW	Input window	137
LA	Line attributes	160
LB	Define label	178
LO	Label origin	179
LT	Line type	162
PA	Plot absolute	140
PD	Pen down	139
PE	Polyline encoded	142
PG	Advance full page	138
PM	Polygon mode	149
PR	Plot relative	141
PU	Pen up	139

Command	Function	Page
PW	Pen width	164
RA	Fill absolute rectangle	154
RF	Raster fill definition	164
RO	Rotate coordinate system	136
RP	Replot	138
RR	Fill relative rectangle	154
RT	Draw relative three point arc	148
SA	Select alternate font	176
SB	Scalable or bitmap fonts	190
SC	Scale	132
SD	Define standard font	172
SI	Set absolute character size	186
SL	Set character slant	189
SM	Symbol mode	166
SP	Select pen	167
SR	Set relative character size	187
SS	Select standard font	176
SV	Screened vectors	167
TD	Transparent data	191
TR	Transparency mode	168
UL	User-defined line type	168
WG	Fill wedge	156
WU	Select pen width unit	170

7.1.3 TrueImage operators

Operator	Function	Page
[	Start array construction	245
]	End array construction	245
=	Write text representation of *any* to standard output file	269
==	Write syntactic representation of *any* to standard output file	269
a4tray	Look for A4 size paper tray	288
abs	Absolute value of *num1*	225
add	*num1* plus *num2*	225
aload	Push all elements of *array* on stack	246
aload	Push all elements of *packedarray* on stack	248
anchorseach	Determine if *seek* is initial substring of *string*	242
and	Logical I bitwise and	229
arc	Append counterclockwise arc	232
arcn	Append clockwise arc	233
arcto	Append tangent arc	233
array	Create array of length *int*	245
ashow	Add (*x*, *y*) to width of each character while showing *string*	261
astore	Pop elements from stack into *array*	246
atan	Arctangent of *num1/num2* in degrees	226
awidthshow	Combine effects of **ashow** and **widthshow**	261
b5tray	Look for B5 size paper tray	288
begin	Push *dict* on dictionary stack	249
bind	Replace operator names in *proc* by operators	271
bitshift	Bitwise shift to *int1* (positive is left)	230
bytesavailable	Number of bytes available to read	267
c5envelope	Establish imaging area to C5 size envelope	287
c5envelopetray	Look for C5 size envelope tray	289
cachestatus	Return font cache status and parameters	263
ceiling	Ceiling of *num1*	226
charpath	Append character outline to current path	235
clear	Discard all elements	224
cleartomark	Discard elements down through **mark**	224
clip	Clip using non-zero winding number rule	237
clippath	Set current path to clipping path	235
closefile	Close *file*	265
closepath	Connect subpath back to its starting point	235
com10envelope	Establish imaging area to COM-10 size envelope	287
com10envelopetray	Look for COM-10 size envelope tray	289
concat	Replace CTM by *matrix* × CTM	281
concatmatrix	Fill *matrix3* with *matrix1* × *matrix2*	282
copy	Duplicate top *n* elements	223
copy	Copy elements of *string1* to initial substring of *string2*	242
copy	Copy elements of *array1* to initial subarray of *array2*	246
copy	Copy elements of *packedarray1* to initial subarray of *array2*	248
copy	Copy contents of *dict1* to *dict2*	251

Operator	Function	Page
copypage	Transmit current page	283
cos	Cosine of *angle* (degrees)	227
count	Count elements on stack	224
countdictstack	Count elements on dictionary stack	252
countexecstack	Count elements on exec stack	254
counttomark	Count elements down to *mark*	224
currentcacheparams	Return current font cache parameters	264
currentdash	Return current dash pattern	277
currentdict	Push current dictionary on orerand stack	252
currentfile	Return file currently being executed	268
currentflat	Return current flatness	277
currentfont	Return current font dictionary	260
currentgray	Return current color as gray value	277
currenthsbcolor	Return current color as hue, saturation, brightness	278
currentlinecap	Return current line cap	273
currentlinejoin	Return current line join	274
currentlinewidth	Return current line width	272
currentmatrix	Fill *matrix* with CTM	280
currentmiterlimit	Return current miter limit	275
currentpacking	Return array packing mode	247
currentpoint	Return current point coordinate	231
currentrgbcolor	Return current color as red, green, blue	278
currentscreen	Return current gray halftone screen	278
currenttransfer	Return current gray transfer function	279
curveto	Append Bézier cubic sction	234
cvi	Convert to integer	258
cvlit	Make object be literal	256
cvn	Convert to name	258
cvr	Convert to real	258
cvs	Convert to string	259
cvrs	Convert to string with radix	258
cvx	Make object be executable	256
def	Associate *key* and *value* in current dictionary	249
defaultmatrix	Fill *matrix* with device default matrix	280
definefont	Register *font* as a font dictionary	260
dict	Create dictionary with capacity for *int* elements	249
dictfull	No more room in dictionary	290
dictstack	Copy dictionary stack into *array*	252
dictstackoverflow	Too many **begins**	290
dictstackunderflow	Too many **ends**	290
div	num_1 divided by num_2	225
dlenvelope	Establish imaging area to DL size envelope	287
dlenvelopetray	Look for DL size envelope tray	289
dojamrecovery	Indicate whether jam recovery is on or off	284
dorep	Indicate whether REP is on or off	284
dtransform	Transform distance (dx_1, dy_1) by CTM or *matrix*	282

Operator	Function	Page
dup	Duplicate top element	223
echo	Turn on/off echoing	269
end	Pop dictionary stack	249
eoclip	Clip using even-odd inside rule	237
eofill	Fill using even-odd rule	238
eq	Test equal	228, 243
erasepage	Paint current page white	238
errordict	Error handler dictionary	251
exch	Exchange top two elements	223
exec	Execute arbitrary object	253
executive	Invoke interactive executive	271
execstack	Copy exec stack into *array*	254
execstackoverflow	Exec nesting too deep	290
executeonly	Reduce access to execute-only	257
executivepage	Establish imaging area to executive size	286
executivetray	Look for Executive size paper tray	288
exit	Exit innermost active loop	254
exp	Raise *num* to *exponent* power	227
false	Push boolean value *false*	229
file	Open file identified by *string1* with access *string2*	265
fill	Fill current path with current color	238
findfont	Return font dictionary identified by *key*	260
findtray	Find the specific paper tray	286
flattenpath	Convert curves to sequences of straight lines	235
floor	Floor of *num1*	226
flush	Send buffered data to standard output file	267
flushfile	Send buffered data or read to EOF	267
FontDirectory	Dictionary of font dictionaries	262
for	Execute *proc* with values form *start* by steps of *increment* to *finish*	253
forall	Execute *proc* for each element of *string*	242
forall	Execute *proc* for each element of *array*	246
forall	Execute *proc* for each element of *packedarray*	248
forall	Execute *proc* for each element of *dict*	251
framedevice	Install frame buffer device	283
ge	Test greater or equal	228, 244
get	Get string element indexed by *index*	241
get	Get array element indexed by *index*	245
get	Get *packedarray* element indexed by *index*	247
get	Get value associated with *key* in *dict*	250
getinterval	Substring of *string* at *index* for *count* elements	241
getinterval	Subarray of *array* starting at *index* for *count* elements	245
getinterval	Subarray of *packedarray* starting at *index* for *count* elements	248
grestore	Pop graphics state	272
grestoreall	Pop to bottommost graphics state	272
gsave	Push graphics state	271

Operator	Function	Page
gt	Test greater than	228, 244
handleerror	Called to report error information	290
identmatrix	Fill *matrix* with identity transform	280
idiv	Integer divide	225
idtransform	Inverse transform distance (dx_1, dy_1) by CTM or *matrix*	282
if	Execute *proc* if *bool* is true	253
ifelse	Execute proc_1 if *bool* is true, proc_2 if *bool* is false	253
image	Paint monochrome sampled image	239
imagemask	Paint current color through mask	240
index	Duplicate arbitrary element	224
initclip	Set clipping path to device default	237
initgraphics	Reset graphics state parameters	272
initmatrix	Set CTM to device default	280
interrupt	External interrupt request	290
invalidaccess	Attempt to violate access attribute	290
invalidexit	**exit** not in loop	290
invalidfileaccess	Unacceptable access string	290
invalidfont	Invalid font name or dictionary	290
invalidrestore	Improper **restore**	290
invertmatrix	Fill matrix_2 with inverse of matrix_1	282
ioerror	Input/output error occurred	290
itransform	Inverse transform (x_1, y_1) by CTM or *matrix*	282
known	Test whether *key* is in *dict*	250
kshow	Execute *proc* between characters shown from *string*	262
le	Test less or equal	228, 244
legaltray	Look for Legal size paper tray	288
length	Number of elements in *string*	241
length	Number of elements in *array*	245
length	Number of elements in *packedarray*	247
length	Number of key-value pairs in *dict*	249
lettertray	Look for Letter size paper tray	287
limitcheck	Implementation limit exceeded	290
lineto	Append straight line to (x, y)	231
ln	Natural logarithm (base *e*)	227
load	Search dictionary stack for *key* and return associated *value*	250
log	Logarithm (base 10)	227
loop	Execute *proc* an indefinite number of times	253
lt	Test less than	228, 244
makefont	Transform font_1 by *matrix* to produce new font_2	260
mark	Push mark on stack	224
matrix	Create identity matrix	280
maxlength	Current capacity of *dict*	249
mod	int_1 mod int_2	225
monarcenvelope	Establish imaging area to Monarch size envelope	287
monarcenvelopetray	Look for Monarch size envelope tray	288
moveto	set current point to (x, y)	231

Operator	**Function**	**Page**
mul	*num1* times *num2*	225
ne	Test not equal	228, 243
neg	Negative of *num1*	226
newpath	Initialize current path to be empty	231
noaccess	Disallow any access	257
nocurrentpoint	Current point is undefined	290
not	Logical | bitwise not	229
null	Push *null* on operand stack	271
nulldevice	Install no-output device	283
or	Logical | bitwise inclusive or	229
packedarray	Create packed array consisting of the specified *n* elements	247
papertray	Return int indicating the current tray	285
pathbbox	Return bounding box of current path	236
pathforall	Enumerate current path	236
pop	Discard top element	223
ppapersize	Ask the paper size of the specific paper tray	286
print	Write *string* to standard output file	268
prompt	Executed when ready for interactive input	269
pstack	Print stack non-destructively using ==	269
put	Put *int* into *string* at *index*	241
put	Put *any* into *array* at *index*	245
put	Associate *key* with *value* in *dict*	250
putinterval	Replace substring of *string1* starting at *index* by *string2*	241
putinterval	Replace subarray of *array* starting at *index* by *subarray*	246
quit	Terminate interpreter	255
rand	Generate pseudo-random integer	227
rangecheck	Operand out of bounds	290
rcheck	Test read access	257
rcurveto	Relative **curveto**	234
read	Read one character from *file*	265
readhexstring	Read hex from *file* into string	266
readline	Read line from *file* into string	266
readonly	Reduce access to read-only	257
readstring	Read string from *file*	266
repeat	Execute *proc int* times	253
resetfile	Discard buffered characters	268
restore	Restore VM snapshot	270
reversepath	Reverse direction of current path	235
rlineto	Relative **lineto**	232
rmoveto	Relative **moveto**	231
roll	Roll *n* elements up *j* times	224
rotate	Rotate user space or define rotation by *angle* degrees	281
round	Round *num1* to nearest integer	226
rrand	Return random number seed	227
run	Execute contents of named file	268
save	Create VM snapshot	270

Operator	Function	Page
scale	Scale user space or define scaling by *x* and *y*	281
scalefont	Scale *font*$_1$ by *scale* to produce new *font*$_2$	260
search	Search for *seek* in string	242
setcachedevice	Declare cached character metrics	263
setcachelimit	Set maximum bytes in cached character	263
setcacheparams	Change font cache parameters	264
setcharwidth	Declare uncached character metrics	263
setdash	Set dash pattern for stroking	276
setdojamrecovery	Turn jam recovery on/off	284
setdorep	Turn REP on/off	284
setemulation	Switch the emulation	287
setflat	Set flatness tolerance	277
setfont	Set font dictionary in graphics state	260
setgray	Set color to specified gray value	277
sethsbcolor	Set color to specified hue, saturation, brightness	278
setlinecap	Set shape of line ends for stroke	273
setlinejoin	Set shape of corners for stroke	274
setlinewidth	Set line width	272
setmatrix	Replace CTM by *matrix*	280
setmiterlimit	Set miter length limit	275
setpacking	Set array packing mode	247
setpapertray	Establish which input tray and set the imaging area	286
setrgbcolor	Set color to specified red, green, blue	278
setscreen	Set gray halftone screen	278
settransfer	Set gray transfer function	279
settray	Set tray which paper will be fed	285
show	Paint characters of *string* on page	261
showpage	Transmit and reset current page	283
sin	Sine of *angle* (degrees).	227
sqrt	Square root of *num*	226
srand	Set random number seed	227
stack	Print stack non-destructively using =	269
stackoverflow	Operand stack overflow	291
stackunderflow	Operand stack underflow	291
StandardEncoding	Standard font encoding vector	262
start	Executed at interpreter startup	255
status	Return status of *file*	268
stop	Terminate **stopped** context	254
stopped	Establish context for catching **stop**	254
store	Replace topmost definition of *key*	250
string	Create string of length *int*	241
stringwidth	Width of *string* in current font	262
stroke	Draw line along current path	238
strokepath	Compute outline of stroked path	235
sub	*num*$_1$ minus *num*$_2$	225
syntaxerror	Language syntax error	291

Operator	Function	Page
systemdict	System dictionary	252
timeout	Time limit exceeded	291
token	Read token from start of *string*	243
token	Read token from *file*	267
transform	Transform (x_1, y_1) by CTM or *matrix*	282
translate	Translate user space or define translation by (x, y)	281
traysup	Check whether the specified paper tray is supplied	285
true	Push boolean value *true*	229
truncate	Remove fractional part of num_1	226
type	Return name identifying the type of *any*	256
typecheck	Operand of wrong type	291
undefined	Name not known	291
undefinedfilename	File not found	291
undefinedresult	Over/underflow or meaningless result	291
unmatchedmark	Expected mark not on stack	291
unregistered	Internal error	291
userdict	Writable dictionary in local VM	252
usertime	Return execution time in milliseconds	271
version	Interpreter version	271
VMerror	VM exhausted	291
vmstatus	Report VM status	270
wcheck	Test write access	257
where	Find dictionary in which *key* is defined	251
widthshow	Add (x, y) to width of *char* while showing *string*	261
write	Write one character to *file*	265
writehexstring	Write *string* to *file* as hex	266
writestring	Write *string* to *file*	266
xcheck	Test executable attribute	256
xor	Logical | bitwise exclusive or	229

7.2 Character set tables

ISO 60: Norwegian

	0	1	2	3	4	5	6	7
0	0	16	32	0 48	@ 64	P 80	` 96	p 112
1	1	17	! 33	1 49	A 65	Q 81	a 97	q 113
2	2	18	" 34	2 50	B 66	R 82	b 98	r 114
3	3	19	# 35	3 51	C 67	S 83	c 99	s 115
4	4	20	$ 36	4 52	D 68	T 84	d 100	t 116
5	5	21	% 37	5 53	E 69	U 85	e 101	u 117
6	6	22	& 38	6 54	F 70	V 86	f 102	v 118
7	7	23	' 39	7 55	G 71	W 87	g 103	w 119
8	<BS> 8	24	(40	8 56	H 72	X 88	h 104	x 120
9	<HT> 9	25	) 41	9 57	I 73	Y 89	i 105	y 121
A	<LF> 10	26	* 42	: 58	J 74	Z 90	j 106	z 122
B	11	<ESC> 27	+ 43	; 59	K 75	Æ 91	k 107	æ 123
C	<FF> 12	28	, 44	< 60	L 76	Ø 92	l 108	ø 124
D	<CR> 13	29	- 45	= 61	M 77	Å 93	m 109	å 125
E	<SO> 14	30	. 46	> 62	N 78	^ 94	n 110	‾ 126
F	<SI> 15	31	/ 47	? 63	O 79	_ 95	o 111	▒ 127

Roman Extension

	0	1	2	3	4	5	6	7
0	0	16	32	¯ 48	â 64	Å 80	Á 96	Þ 112
1	1	17	À 33	Ý 49	ê 65	î 81	Ã 97	þ 113
2	2	18	Â 34	ý 50	ô 66	Ø 82	ã 98	· 114
3	3	19	È 35	° 51	û 67	Æ 83	Đ 99	µ 115
4	4	20	Ê 36	Ç 52	á 68	å 84	ð 100	¶ 116
5	5	21	Ë 37	ç 53	é 69	í 85	Í 101	¾ 117
6	6	22	Î 38	Ñ 54	ó 70	ø 86	Ì 102	- 118
7	7	23	Ï 39	ñ 55	ú 71	æ 87	Ó 103	¼ 119
8	<BS> 8	24	´ 40	¡ 56	à 72	Ä 88	Ò 104	½ 120
9	<HT> 9	25	` 41	¿ 57	è 73	ì 89	Õ 105	ª 121
A	<LF> 10	26	^ 42	¤ 58	ò 74	Ö 90	õ 106	º 122
B	11	<ESC> 27	¨ 43	£ 59	ù 75	Ü 91	Š 107	« 123
C	<FF> 12	28	~ 44	¥ 60	ä 76	É 92	š 108	■ 124
D	<CR> 13	29	Ù 45	§ 61	ë 77	ï 93	Ú 109	» 125
E	<SO> 14	30	Û 46	ƒ 62	ö 78	ß 94	Ÿ 110	± 126
F	<SI> 15	31	£ 47	¢ 63	ü 79	Ô 95	ÿ 111	127

ISO 25: French

	0	1	2	3	4	5	6	7
0	0	16	32	0 48	à 64	P 80	` 96	p 112
1	1	17	! 33	1 49	A 65	Q 81	a 97	q 113
2	2	18	" 34	2 50	B 66	R 82	b 98	r 114
3	3	19	£ 35	3 51	C 67	S 83	c 99	s 115
4	4	20	$ 36	4 52	D 68	T 84	d 100	t 116
5	5	21	% 37	5 53	E 69	U 85	e 101	u 117
6	6	22	& 38	6 54	F 70	V 86	f 102	v 118
7	7	23	' 39	7 55	G 71	W 87	g 103	w 119
8	<BS> 8	24	(40	8 56	H 72	X 88	h 104	x 120
9	<HT> 9	25	) 41	9 57	I 73	Y 89	i 105	y 121
A	<LF> 10	26	* 42	: 58	J 74	Z 90	j 106	z 122
B	11	<ESC> 27	+ 43	; 59	K 75	° 91	k 107	é 123
C	<FF> 12	28	, 44	< 60	L 76	ç 92	l 108	ù 124
D	<CR> 13	29	- 45	= 61	M 77	§ 93	m 109	è 125
E	<SO> 14	30	. 46	> 62	N 78	^ 94	n 110	¨ 126
F	<SI> 15	31	/ 47	? 63	O 79	_ 95	o 111	▒ 127

HP German

	0	1	2	3	4	5	6	7
0	0	16	32	0 48	§ 64	P 80	` 96	p 112
1	1	17	! 33	1 49	A 65	Q 81	a 97	q 113
2	2	18	" 34	2 50	B 66	R 82	b 98	r 114
3	3	19	£ 35	3 51	C 67	S 83	c 99	s 115
4	4	20	$ 36	4 52	D 68	T 84	d 100	t 116
5	5	21	% 37	5 53	E 69	U 85	e 101	u 117
6	6	22	& 38	6 54	F 70	V 86	f 102	v 118
7	7	23	' 39	7 55	G 71	W 87	g 103	w 119
8	<BS> 8	24	(40	8 56	H 72	X 88	h 104	x 120
9	<HT> 9	25	) 41	9 57	I 73	Y 89	i 105	y 121
A	<LF> 10	26	* 42	: 58	J 74	Z 90	j 106	z 122
B	11	<ESC> 27	+ 43	; 59	K 75	Ä 91	k 107	ä 123
C	<FF> 12	28	, 44	< 60	L 76	Ö 92	l 108	ö 124
D	<CR> 13	29	- 45	= 61	M 77	Ü 93	m 109	ü 125
E	<SO> 14	30	. 46	> 62	N 78	^ 94	n 110	ß 126
F	<SI> 15	31	/ 47	? 63	O 79	_ 95	o 111	▒ 127

ISO 15: Italian

	0	1	2	3	4	5	6	7
0	0	16	32	0 48	§ 64	P 80	ù 96	p 112
1	1	17	! 33	1 49	A 65	Q 81	a 97	q 113
2	2	18	" 34	2 50	B 66	R 82	b 98	r 114
3	3	19	£ 35	3 51	C 67	S 83	c 99	s 115
4	4	20	$ 36	4 52	D 68	T 84	d 100	t 116
5	5	21	% 37	5 53	E 69	U 85	e 101	u 117
6	6	22	& 38	6 54	F 70	V 86	f 102	v 118
7	7	23	' 39	7 55	G 71	W 87	g 103	w 119
8	<BS> 8	24	(40	8 56	H 72	X 88	h 104	x 120
9	<HT> 9	25	) 41	9 57	I 73	Y 89	i 105	y 121
A	<LF> 10	26	* 42	: 58	J 74	Z 90	j 106	z 122
B	11	<ESC> 27	+ 43	; 59	K 75	° 91	k 107	à 123
C	<FF> 12	28	, 44	< 60	L 76	ç 92	l 108	ò 124
D	<CR> 13	29	- 45	= 61	M 77	é 93	m 109	è 125
E	<SO> 14	30	. 46	> 62	N 78	^ 94	n 110	ì 126
F	<SI> 15	31	/ 47	? 63	O 79	_ 95	o 111	▓ 127

JIS ASCII

	0	1	2	3	4	5	6	7
0	0	16	32	0 48	@ 64	P 80	` 96	p 112
1	1	17	! 33	1 49	A 65	Q 81	a 97	q 113
2	2	18	" 34	2 50	B 66	R 82	b 98	r 114
3	3	19	# 35	3 51	C 67	S 83	c 99	s 115
4	4	20	$ 36	4 52	D 68	T 84	d 100	t 116
5	5	21	% 37	5 53	E 69	U 85	e 101	u 117
6	6	22	& 38	6 54	F 70	V 86	f 102	v 118
7	7	23	' 39	7 55	G 71	W 87	g 103	w 119
8	<BS> 8	24	(40	8 56	H 72	X 88	h 104	x 120
9	<HT> 9	25	) 41	9 57	I 73	Y 89	i 105	y 121
A	<LF> 10	26	* 42	: 58	J 74	Z 90	j 106	z 122
B	11	<ESC> 27	+ 43	; 59	K 75	[91	k 107	{ 123
C	<FF> 12	28	, 44	< 60	L 76	¥ 92	l 108	| 124
D	<CR> 13	29	- 45	= 61	M 77	] 93	m 109	} 125
E	<SO> 14	30	. 46	> 62	N 78	^ 94	n 110	‾ 126
F	<SI> 15	31	/ 47	? 63	O 79	_ 95	o 111	▓ 127

ECMA-94 Latin 1

	0	1	2	3	4	5	6	7
0	0	16	32	0 48	@ 64	P 80	` 96	p 112
1	1	17	! 33	1 49	A 65	Q 81	a 97	q 113
2	2	18	" 34	2 50	B 66	R 82	b 98	r 114
3	3	19	# 35	3 51	C 67	S 83	c 99	s 115
4	4	20	$ 36	4 52	D 68	T 84	d 100	t 116
5	5	21	% 37	5 53	E 69	U 85	e 101	u 117
6	6	22	& 38	6 54	F 70	V 86	f 102	v 118
7	7	23	' 39	7 55	G 71	W 87	g 103	w 119
8	<BS> 8	24	(40	8 56	H 72	X 88	h 104	x 120
9	<HT> 9	25	) 41	9 57	I 73	Y 89	i 105	y 121
A	<LF> 10	26	* 42	: 58	J 74	Z 90	j 106	z 122
B	11	<ESC> 27	+ 43	; 59	K 75	[91	k 107	{ 123
C	<FF> 12	28	, 44	< 60	L 76	\ 92	l 108	| 124
D	<CR> 13	29	- 45	= 61	M 77	] 93	m 109	} 125
E	<SO> 14	30	. 46	> 62	N 78	^ 94	n 110	~ 126
F	<SI> 15	31	/ 47	? 63	O 79	_ 95	o 111	▓ 127

ECMA-94 Latin 1

	8	9	A	B	C	D	E	F
0	128	144	160	° 176	À 192	Ð 208	à 224	ð 240
1	129	145	¡ 161	± 177	Á 193	Ñ 209	á 225	ñ 241
2	130	146	¢ 162	² 178	Â 194	Ò 210	â 226	ò 242
3	131	147	£ 163	³ 179	Ã 195	Ó 211	ã 227	ó 243
4	132	148	¤ 164	´ 180	Ä 196	Ô 212	ä 228	ô 244
5	133	149	¥ 165	µ 181	Å 197	Õ 213	å 229	õ 245
6	134	150	¦ 166	¶ 182	Æ 198	Ö 214	æ 230	ö 246
7	135	151	§ 167	· 183	Ç 199	× 215	ç 231	÷ 247
8	136	152	¨ 168	¸ 184	È 200	Ø 216	è 232	ø 248
9	137	153	© 169	¹ 185	É 201	Ù 217	é 233	ù 249
A	138	154	ª 170	º 186	Ê 202	Ú 218	ê 234	ú 250
B	139	155	« 171	» 187	Ë 203	Û 219	ë 235	û 251
C	140	156	¬ 172	¼ 188	Ì 204	Ü 220	ì 236	ü 252
D	141	157	- 173	½ 189	Í 205	Ý 221	í 237	ý 253
E	142	158	® 174	¾ 190	Î 206	Þ 222	î 238	þ 254
F	143	159	¯ 175	¿ 191	Ï 207	ß 223	ï 239	ÿ 255

ISO 11: Swedish

	0	1	2	3	4	5	6	7
0	0	16	32	0 48	É 64	P 80	é 96	p 112
1	1	17	! 33	1 49	A 65	Q 81	a 97	q 113
2	2	18	" 34	2 50	B 66	R 82	b 98	r 114
3	3	19	# 35	3 51	C 67	S 83	c 99	s 115
4	4	20	¤ 36	4 52	D 68	T 84	d 100	t 116
5	5	21	% 37	5 53	E 69	U 85	e 101	u 117
6	6	22	& 38	6 54	F 70	V 86	f 102	v 118
7	7	23	' 39	7 55	G 71	W 87	g 103	w 119
8	<BS> 8	24	(40	8 56	H 72	X 88	h 104	x 120
9	<HT> 9	25	) 41	9 57	I 73	Y 89	i 105	y 121
A	<LF> 10	26	* 42	: 58	J 74	Z 90	j 106	z 122
B	11	<ESC> 27	+ 43	; 59	K 75	Ä 91	k 107	ä 123
C	<FF> 12	28	, 44	< 60	L 76	Ö 92	l 108	ö 124
D	<CR> 13	29	- 45	= 61	M 77	Å 93	m 109	å 125
E	<SO> 14	30	. 46	> 62	N 78	Ü 94	n 110	ü 126
F	<SI> 15	31	/ 47	? 63	O 79	_ 95	o 111	▒ 127

US-ASCII

	0	1	2	3	4	5	6	7
0	0	16	32	0 48	@ 64	P 80	` 96	p 112
1	1	17	! 33	1 49	A 65	Q 81	a 97	q 113
2	2	18	" 34	2 50	B 66	R 82	b 98	r 114
3	3	19	# 35	3 51	C 67	S 83	c 99	s 115
4	4	20	$ 36	4 52	D 68	T 84	d 100	t 116
5	5	21	% 37	5 53	E 69	U 85	e 101	u 117
6	6	22	& 38	6 54	F 70	V 86	f 102	v 118
7	7	23	' 39	7 55	G 71	W 87	g 103	w 119
8	<BS> 8	24	(40	8 56	H 72	X 88	h 104	x 120
9	<HT> 9	25	) 41	9 57	I 73	Y 89	i 105	y 121
A	<LF> 10	26	* 42	: 58	J 74	Z 90	j 106	z 122
B	11	<ESC> 27	+ 43	; 59	K 75	[91	k 107	{ 123
C	<FF> 12	28	, 44	< 60	L 76	\ 92	l 108	| 124
D	<CR> 13	29	- 45	= 61	M 77	] 93	m 109	} 125
E	<SO> 14	30	. 46	> 62	N 78	^ 94	n 110	~ 126
F	<SI> 15	31	/ 47	? 63	O 79	_ 95	o 111	▒ 127

ISO 61: Norwegian

	0	1	2	3	4	5	6	7
0	0	16	32	0 48	@ 64	P 80	` 96	p 112
1	1	17	! 33	1 49	A 65	Q 81	a 97	q 113
2	2	18	" 34	2 50	B 66	R 82	b 98	r 114
3	3	19	§ 35	3 51	C 67	S 83	c 99	s 115
4	4	20	$ 36	4 52	D 68	T 84	d 100	t 116
5	5	21	% 37	5 53	E 69	U 85	e 101	u 117
6	6	22	& 38	6 54	F 70	V 86	f 102	v 118
7	7	23	' 39	7 55	G 71	W 87	g 103	w 119
8	<BS> 8	24	(40	8 56	H 72	X 88	h 104	x 120
9	<HT> 9	25	) 41	9 57	I 73	Y 89	i 105	y 121
A	<LF> 10	26	* 42	: 58	J 74	Z 90	j 106	z 122
B	11	<ESC> 27	+ 43	; 59	K 75	Æ 91	k 107	æ 123
C	<FF> 12	28	, 44	< 60	L 76	Ø 92	l 108	ø 124
D	<CR> 13	29	- 45	= 61	M 77	Å 93	m 109	å 125
E	<SO> 14	30	. 46	> 62	N 78	^ 94	n 110	| 126
F	<SI> 15	31	/ 47	? 63	O 79	_ 95	o 111	▓ 127

ISO 4: UK

	0	1	2	3	4	5	6	7
0	0	16	32	0 48	@ 64	P 80	` 96	p 112
1	1	17	! 33	1 49	A 65	Q 81	a 97	q 113
2	2	18	" 34	2 50	B 66	R 82	b 98	r 114
3	3	19	£ 35	3 51	C 67	S 83	c 99	s 115
4	4	20	$ 36	4 52	D 68	T 84	d 100	t 116
5	5	21	% 37	5 53	E 69	U 85	e 101	u 117
6	6	22	& 38	6 54	F 70	V 86	f 102	v 118
7	7	23	' 39	7 55	G 71	W 87	g 103	w 119
8	<BS> 8	24	(40	8 56	H 72	X 88	h 104	x 120
9	<HT> 9	25	) 41	9 57	I 73	Y 89	i 105	y 121
A	<LF> 10	26	* 42	: 58	J 74	Z 90	j 106	z 122
B	11	<ESC> 27	+ 43	; 59	K 75	[91	k 107	{ 123
C	<FF> 12	28	, 44	< 60	L 76	\ 92	l 108	| 124
D	<CR> 13	29	- 45	= 61	M 77	] 93	m 109	} 125
E	<SO> 14	30	. 46	> 62	N 78	^ 94	n 110	‾ 126
F	<SI> 15	31	/ 47	? 63	O 79	_ 95	o 111	▒ 127

ISO 69: French

	0	1	2	3	4	5	6	7
0	0	16	32	0 48	à 64	P 80	µ 96	p 112
1	1	17	! 33	1 49	A 65	Q 81	a 97	q 113
2	2	18	" 34	2 50	B 66	R 82	b 98	r 114
3	3	19	£ 35	3 51	C 67	S 83	c 99	s 115
4	4	20	$ 36	4 52	D 68	T 84	d 100	t 116
5	5	21	% 37	5 53	E 69	U 85	e 101	u 117
6	6	22	& 38	6 54	F 70	V 86	f 102	v 118
7	7	23	' 39	7 55	G 71	W 87	g 103	w 119
8	<BS> 8	24	(40	8 56	H 72	X 88	h 104	x 120
9	<HT> 9	25	) 41	9 57	I 73	Y 89	i 105	y 121
A	<LF> 10	26	* 42	: 58	J 74	Z 90	j 106	z 122
B	11	<ESC> 27	+ 43	; 59	K 75	° 91	k 107	é 123
C	<FF> 12	28	, 44	< 60	L 76	ç 92	l 108	ù 124
D	<CR> 13	29	− 45	= 61	M 77	§ 93	m 109	è 125
E	<SO> 14	30	. 46	> 62	N 78	^ 94	n 110	¨ 126
F	<SI> 15	31	/ 47	? 63	O 79	_ 95	o 111	▓ 127

ISO 21: German

	0	1	2	3	4	5	6	7
0	0	16	32	0 48	§ 64	P 80	` 96	p 112
1	1	17	! 33	1 49	A 65	Q 81	a 97	q 113
2	2	18	" 34	2 50	B 66	R 82	b 98	r 114
3	3	19	# 35	3 51	C 67	S 83	c 99	s 115
4	4	20	$ 36	4 52	D 68	T 84	d 100	t 116
5	5	21	% 37	5 53	E 69	U 85	e 101	u 117
6	6	22	& 38	6 54	F 70	V 86	f 102	v 118
7	7	23	' 39	7 55	G 71	W 87	g 103	w 119
8	<BS> 8	24	(40	8 56	H 72	X 88	h 104	x 120
9	<HT> 9	25	) 41	9 57	I 73	Y 89	i 105	y 121
A	<LF> 10	26	* 42	: 58	J 74	Z 90	j 106	z 122
B	11	<ESC> 27	+ 43	; 59	K 75	Ä 91	k 107	ä 123
C	<FF> 12	28	, 44	< 60	L 76	Ö 92	l 108	ö 124
D	<CR> 13	29	- 45	= 61	M 77	Ü 93	m 109	ü 125
E	<SO> 14	30	. 46	> 62	N 78	^ 94	n 110	ß 126
F	<SI> 15	31	/ 47	? 63	O 79	_ 95	o 111	▒ 127

HP Spanish

	0	1	2	3	4	5	6	7
0	0	16	32	0 48	@ 64	P 80	` 96	p 112
1	1	17	! 33	1 49	A 65	Q 81	a 97	q 113
2	2	18	" 34	2 50	B 66	R 82	b 98	r 114
3	3	19	# 35	3 51	C 67	S 83	c 99	s 115
4	4	20	$ 36	4 52	D 68	T 84	d 100	t 116
5	5	21	% 37	5 53	E 69	U 85	e 101	u 117
6	6	22	& 38	6 54	F 70	V 86	f 102	v 118
7	7	23	' 39	7 55	G 71	W 87	g 103	w 119
8	<BS> 8	24	(40	8 56	H 72	X 88	h 104	x 120
9	<HT> 9	25	) 41	9 57	I 73	Y 89	i 105	y 121
A	<LF> 10	26	* 42	: 58	J 74	Z 90	j 106	z 122
B	11	<ESC> 27	+ 43	; 59	K 75	¡ 91	k 107	{ 123
C	<FF> 12	28	, 44	< 60	L 76	Ñ 92	l 108	ñ 124
D	<CR> 13	29	- 45	= 61	M 77	¿ 93	m 109	} 125
E	<SO> 14	30	. 46	> 62	N 78	° 94	n 110	~ 126
F	<SI> 15	31	/ 47	? 63	O 79	_ 95	o 111	▒ 127

ISO 57: Chinese

	0	1	2	3	4	5	6	7
0	0	16	32	0 48	@ 64	P 80	` 96	p 112
1	1	17	! 33	1 49	A 65	Q 81	a 97	q 113
2	2	18	" 34	2 50	B 66	R 82	b 98	r 114
3	3	19	# 35	3 51	C 67	S 83	c 99	s 115
4	4	20	¥ 36	4 52	D 68	T 84	d 100	t 116
5	5	21	% 37	5 53	E 69	U 85	e 101	u 117
6	6	22	& 38	6 54	F 70	V 86	f 102	v 118
7	7	23	' 39	7 55	G 71	W 87	g 103	w 119
8	<BS> 8	24	(40	8 56	H 72	X 88	h 104	x 120
9	<HT> 9	25	) 41	9 57	I 73	Y 89	i 105	y 121
A	<LF> 10	26	* 42	: 58	J 74	Z 90	j 106	z 122
B	11	<ESC> 27	+ 43	; 59	K 75	[91	k 107	{ 123
C	<FF> 12	28	, 44	< 60	L 76	\ 92	l 108	| 124
D	<CR> 13	29	- 45	= 61	M 77	] 93	m 109	} 125
E	<SO> 14	30	. 46	> 62	N 78	^ 94	n 110	‾ 126
F	<SI> 15	31	/ 47	? 63	O 79	_ 95	o 111	▒ 127

ISO 17: Spanish

	0	1	2	3	4	5	6	7
0	0	16	32	0 48	§ 64	P 80	` 96	p 112
1	1	17	! 33	1 49	A 65	Q 81	a 97	q 113
2	2	18	" 34	2 50	B 66	R 82	b 98	r 114
3	3	19	£ 35	3 51	C 67	S 83	c 99	s 115
4	4	20	$ 36	4 52	D 68	T 84	d 100	t 116
5	5	21	% 37	5 53	E 69	U 85	e 101	u 117
6	6	22	& 38	6 54	F 70	V 86	f 102	v 118
7	7	23	' 39	7 55	G 71	W 87	g 103	w 119
8	<BS> 8	24	(40	8 56	H 72	X 88	h 104	x 120
9	<HT> 9	25	) 41	9 57	I 73	Y 89	i 105	y 121
A	<LF> 10	26	* 42	: 58	J 74	Z 90	j 106	z 122
B	11	<ESC> 27	+ 43	; 59	K 75	¡ 91	k 107	° 123
C	<FF> 12	28	, 44	< 60	L 76	Ñ 92	l 108	ñ 124
D	<CR> 13	29	- 45	= 61	M 77	¿ 93	m 109	ç 125
E	<SO> 14	30	. 46	> 62	N 78	^ 94	n 110	~ 126
F	<SI> 15	31	/ 47	? 63	O 79	_ 95	o 111	▓ 127

ISO 2: IRV

	0	1	2	3	4	5	6	7
0	0	16	32	0 48	@ 64	P 80	` 96	p 112
1	1	17	! 33	1 49	A 65	Q 81	a 97	q 113
2	2	18	" 34	2 50	B 66	R 82	b 98	r 114
3	3	19	# 35	3 51	C 67	S 83	c 99	s 115
4	4	20	¤ 36	4 52	D 68	T 84	d 100	t 116
5	5	21	% 37	5 53	E 69	U 85	e 101	u 117
6	6	22	& 38	6 54	F 70	V 86	f 102	v 118
7	7	23	' 39	7 55	G 71	W 87	g 103	w 119
8	<BS> 8	24	(40	8 56	H 72	X 88	h 104	x 120
9	<HT> 9	25	) 41	9 57	I 73	Y 89	i 105	y 121
A	<LF> 10	26	* 42	: 58	J 74	Z 90	j 106	z 122
B	11	<ESC> 27	+ 43	; 59	K 75	[91	k 107	{ 123
C	<FF> 12	28	, 44	< 60	L 76	\ 92	l 108	| 124
D	<CR> 13	29	- 45	= 61	M 77	] 93	m 109	} 125
E	<SO> 14	30	. 46	> 62	N 78	^ 94	n 110	‾ 126
F	<SI> 15	31	/ 47	? 63	O 79	_ 95	o 111	▒ 127

ISO 10: Swedish

	0	1	2	3	4	5	6	7
0	0	16	32	0 48	@ 64	P 80	` 96	p 112
1	1	17	! 33	1 49	A 65	Q 81	a 97	q 113
2	2	18	" 34	2 50	B 66	R 82	b 98	r 114
3	3	19	# 35	3 51	C 67	S 83	c 99	s 115
4	4	20	¤ 36	4 52	D 68	T 84	d 100	t 116
5	5	21	% 37	5 53	E 69	U 85	e 101	u 117
6	6	22	& 38	6 54	F 70	V 86	f 102	v 118
7	7	23	' 39	7 55	G 71	W 87	g 103	w 119
8	<BS> 8	24	(40	8 56	H 72	X 88	h 104	x 120
9	<HT> 9	25	) 41	9 57	I 73	Y 89	i 105	y 121
A	<LF> 10	26	* 42	: 58	J 74	Z 90	j 106	z 122
B	11	<ESC> 27	+ 43	; 59	K 75	Ä 91	k 107	ä 123
C	<FF> 12	28	, 44	< 60	L 76	Ö 92	l 108	ö 124
D	<CR> 13	29	- 45	= 61	M 77	Å 93	m 109	å 125
E	<SO> 14	30	. 46	> 62	N 78	^ 94	n 110	‾ 126
F	<SI> 15	31	/ 47	? 63	O 79	_ 95	o 111	▒ 127

ISO 16: Portuguese

	0	1	2	3	4	5	6	7
0	0	16	32	0 48	§ 64	P 80	` 96	p 112
1	1	17	! 33	1 49	A 65	Q 81	a 97	q 113
2	2	18	" 34	2 50	B 66	R 82	b 98	r 114
3	3	19	# 35	3 51	C 67	S 83	c 99	s 115
4	4	20	$ 36	4 52	D 68	T 84	d 100	t 116
5	5	21	% 37	5 53	E 69	U 85	e 101	u 117
6	6	22	& 38	6 54	F 70	V 86	f 102	v 118
7	7	23	' 39	7 55	G 71	W 87	g 103	w 119
8	<BS> 8	24	(40	8 56	H 72	X 88	h 104	x 120
9	<HT> 9	25	) 41	9 57	I 73	Y 89	i 105	y 121
A	<LF> 10	26	* 42	: 58	J 74	Z 90	j 106	z 122
B	11	<ESC> 27	+ 43	; 59	K 75	Ã 91	k 107	ã 123
C	<FF> 12	28	, 44	< 60	L 76	Ç 92	l 108	ç 124
D	<CR> 13	29	− 45	= 61	M 77	Õ 93	m 109	õ 125
E	<SO> 14	30	. 46	> 62	N 78	^ 94	n 110	° 126
F	<SI> 15	31	/ 47	? 63	O 79	_ 95	o 111	▓ 127

ISO 84: Portuguese

	0	1	2	3	4	5	6	7
0	0	16	32	0 48	´ 64	P 80	` 96	p 112
1	1	17	! 33	1 49	A 65	Q 81	a 97	q 113
2	2	18	" 34	2 50	B 66	R 82	b 98	r 114
3	3	19	# 35	3 51	C 67	S 83	c 99	s 115
4	4	20	$ 36	4 52	D 68	T 84	d 100	t 116
5	5	21	% 37	5 53	E 69	U 85	e 101	u 117
6	6	22	& 38	6 54	F 70	V 86	f 102	v 118
7	7	23	' 39	7 55	G 71	W 87	g 103	w 119
8	<BS> 8	24	(40	8 56	H 72	X 88	h 104	x 120
9	<HT> 9	25	) 41	9 57	I 73	Y 89	i 105	y 121
A	<LF> 10	26	* 42	: 58	J 74	Z 90	j 106	z 122
B	11	<ESC> 27	+ 43	; 59	K 75	Ã 91	k 107	ã 123
C	<FF> 12	28	, 44	< 60	L 76	Ç 92	l 108	ç 124
D	<CR> 13	29	- 45	= 61	M 77	Õ 93	m 109	õ 125
E	<SO> 14	30	. 46	> 62	N 78	^ 94	n 110	~ 126
F	<SI> 15	31	/ 47	? 63	O 79	_ 95	o 111	▒ 127

ISO 85: Spanish

	0	1	2	3	4	5	6	7
0	0	16	32	0 48	· 64	P 80	` 96	p 112
1	1	17	! 33	1 49	A 65	Q 81	a 97	q 113
2	2	18	" 34	2 50	B 66	R 82	b 98	r 114
3	3	19	# 35	3 51	C 67	S 83	c 99	s 115
4	4	20	$ 36	4 52	D 68	T 84	d 100	t 116
5	5	21	% 37	5 53	E 69	U 85	e 101	u 117
6	6	22	& 38	6 54	F 70	V 86	f 102	v 118
7	7	23	' 39	7 55	G 71	W 87	g 103	w 119
8	<BS> 8	24	(40	8 56	H 72	X 88	h 104	x 120
9	<HT> 9	25	) 41	9 57	I 73	Y 89	i 105	y 121
A	<LF> 10	26	* 42	: 58	J 74	Z 90	j 106	z 122
B	11	<ESC> 27	+ 43	; 59	K 75	¡ 91	k 107	´ 123
C	<FF> 12	28	, 44	< 60	L 76	Ñ 92	l 108	ñ 124
D	<CR> 13	29	- 45	= 61	M 77	Ç 93	m 109	ç 125
E	<SO> 14	30	. 46	> 62	N 78	¿ 94	n 110	¨ 126
F	<SI> 15	31	/ 47	? 63	O 79	_ 95	o 111	▒ 127

Roman-8

	0	1	2	3	4	5	6	7
0	0	16	32	0 48	@ 64	P 80	` 96	p 112
1	1	17	! 33	1 49	A 65	Q 81	a 97	q 113
2	2	18	" 34	2 50	B 66	R 82	b 98	r 114
3	3	19	# 35	3 51	C 67	S 83	c 99	s 115
4	4	20	$ 36	4 52	D 68	T 84	d 100	t 116
5	5	21	% 37	5 53	E 69	U 85	e 101	u 117
6	6	22	& 38	6 54	F 70	V 86	f 102	v 118
7	7	23	' 39	7 55	G 71	W 87	g 103	w 119
8	<BS> 8	24	(40	8 56	H 72	X 88	h 104	x 120
9	<HT> 9	25	) 41	9 57	I 73	Y 89	i 105	y 121
A	<LF> 10	26	* 42	: 58	J 74	Z 90	j 106	z 122
B	11	<ESC> 27	+ 43	; 59	K 75	[91	k 107	{ 123
C	<FF> 12	28	, 44	< 60	L 76	\ 92	l 108	| 124
D	<CR> 13	29	- 45	= 61	M 77	] 93	m 109	} 125
E	<SO> 14	30	. 46	> 62	N 78	^ 94	n 110	~ 126
F	<SI> 15	31	/ 47	? 63	O 79	_ 95	o 111	▒ 127

Roman-8

	8	9	A	B	C	D	E	F
0	128	144	160	‾ 176	â 192	Å 208	Á 224	Þ 240
1	129	145	À 161	Ý 177	ê 193	î 209	Ã 225	þ 241
2	130	146	Â 162	ý 178	ô 194	Ø 210	ã 226	· 242
3	131	147	È 163	° 179	û 195	Æ 211	Đ 227	µ 243
4	132	148	Ê 164	Ç 180	á 196	å 212	ð 228	¶ 244
5	133	149	Ë 165	ç 181	é 197	í 213	Í 229	¾ 245
6	134	150	Î 166	Ñ 182	ó 198	ø 214	Ì 230	– 246
7	135	151	Ï 167	ñ 183	ú 199	æ 215	Ó 231	¼ 247
8	136	152	´ 168	¡ 184	à 200	Ä 216	Ò 232	½ 248
9	137	153	` 169	¿ 185	è 201	ì 217	Õ 233	ª 249
A	138	154	ˆ 170	¤ 186	ò 202	Ö 218	õ 234	º 250
B	139	155	¨ 171	£ 187	ù 203	Ü 219	Š 235	« 251
C	140	156	˜ 172	¥ 188	ä 204	É 220	š 236	■ 252
D	141	157	Ù 173	§ 189	ë 205	ï 221	Ú 237	» 253
E	142	158	Û 174	ƒ 190	ö 206	ß 222	Ÿ 238	± 254
F	143	159	£ 175	¢ 191	ü 207	Ô 223	ÿ 239	255

IBM-PC(US)

	0	1	2	3	4	5	6	7
0	0	0 16	32	0 48	@ 64	P 80	` 96	p 112
1	! 1	1 17	! 33	1 49	A 65	Q 81	a 97	q 113
2	" 2	2 18	" 34	2 50	B 66	R 82	b 98	r 114
3	# 3	3 19	# 35	3 51	C 67	S 83	c 99	s 115
4	$ 4	4 20	$ 36	4 52	D 68	T 84	d 100	t 116
5	% 5	5 21	% 37	5 53	E 69	U 85	e 101	u 117
6	& 6	6 22	& 38	6 54	F 70	V 86	f 102	v 118
7	' 7	7 23	' 39	7 55	G 71	W 87	g 103	w 119
8	(8	8 24	(40	8 56	H 72	X 88	h 104	x 120
9	) 9	9 25	) 41	9 57	I 73	Y 89	i 105	y 121
A	* 10	: 26	* 42	: 58	J 74	Z 90	j 106	z 122
B	+ 11	; 27	+ 43	; 59	K 75	[91	k 107	{ 123
C	, 12	< 28	, 44	< 60	L 76	\ 92	l 108	| 124
D	- 13	= 29	- 45	= 61	M 77	] 93	m 109	} 125
E	. 14	> 30	. 46	> 62	N 78	^ 94	n 110	~ 126
F	/ 15	? 31	/ 47	? 63	O 79	_ 95	o 111	⌂ 127

IBM-PC(US)

	8	9	A	B	C	D	E	F
0	Ç 128	É 144	á 160	░ 176	└ 192	╨ 208	α 224	≡ 240
1	ü 129	æ 145	í 161	▒ 177	┴ 193	╤ 209	ß 225	± 241
2	é 130	Æ 146	ó 162	▓ 178	┬ 194	╥ 210	Γ 226	≥ 242
3	â 131	ô 147	ú 163	│ 179	├ 195	╙ 211	π 227	≤ 243
4	ä 132	ö 148	ñ 164	┤ 180	─ 196	╘ 212	Σ 228	⌠ 244
5	à 133	ò 149	Ñ 165	╡ 181	┼ 197	╒ 213	σ 229	⌡ 245
6	å 134	û 150	ª 166	╢ 182	╞ 198	╓ 214	µ 230	÷ 246
7	ç 135	ù 151	º 167	╖ 183	╟ 199	╫ 215	τ 231	≈ 247
8	ê 136	ÿ 152	¿ 168	╕ 184	╚ 200	╪ 216	Φ 232	° 248
9	ë 137	Ö 153	⌐ 169	╣ 185	╔ 201	┘ 217	Θ 233	∙ 249
A	è 138	Ü 154	¬ 170	║ 186	╩ 202	┌ 218	Ω 234	· 250
B	ï 139	¢ 155	½ 171	╗ 187	╦ 203	█ 219	δ 235	√ 251
C	î 140	£ 156	¼ 172	╝ 188	╠ 204	▄ 220	∞ 236	ⁿ 252
D	ì 141	¥ 157	¡ 173	╜ 189	═ 205	▌ 221	φ 237	² 253
E	Ä 142	₧ 158	« 174	╛ 190	╬ 206	▐ 222	ε 238	■ 254
F	Å 143	ƒ 159	» 175	┐ 191	╧ 207	▀ 223	∩ 239	255

IBM-PC(Denmark/Norway)

	0	1	2	3	4	5	6	7
0	0	0 16	32	0 48	@ 64	P 80	` 96	p 112
1	! 1	1 17	! 33	1 49	A 65	Q 81	a 97	q 113
2	" 2	2 18	" 34	2 50	B 66	R 82	b 98	r 114
3	# 3	3 19	# 35	3 51	C 67	S 83	c 99	s 115
4	$ 4	4 20	$ 36	4 52	D 68	T 84	d 100	t 116
5	% 5	5 21	% 37	5 53	E 69	U 85	e 101	u 117
6	& 6	6 22	& 38	6 54	F 70	V 86	f 102	v 118
7	' 7	7 23	' 39	7 55	G 71	W 87	g 103	w 119
8	(8	8 24	(40	8 56	H 72	X 88	h 104	x 120
9	) 9	9 25	) 41	9 57	I 73	Y 89	i 105	y 121
A	* 10	: 26	* 42	: 58	J 74	Z 90	j 106	z 122
B	+ 11	; 27	+ 43	; 59	K 75	[91	k 107	{ 123
C	, 12	< 28	, 44	< 60	L 76	\ 92	l 108	| 124
D	- 13	= 29	- 45	= 61	M 77	] 93	m 109	} 125
E	. 14	> 30	. 46	> 62	N 78	^ 94	n 110	~ 126
F	/ 15	? 31	/ 47	? 63	O 79	_ 95	o 111	⌂ 127

IBM-PC(Denmark/Norway)

	8	9	A	B	C	D	E	F
0	Ç 128	É 144	á 160	░ 176	└ 192	╨ 208	α 224	≡ 240
1	ü 129	æ 145	í 161	▒ 177	┴ 193	╤ 209	ß 225	± 241
2	é 130	Æ 146	ó 162	▓ 178	┬ 194	╥ 210	Γ 226	≥ 242
3	â 131	ô 147	ú 163	│ 179	├ 195	╙ 211	π 227	≤ 243
4	ä 132	ö 148	ñ 164	┤ 180	─ 196	╘ 212	Σ 228	⌠ 244
5	à 133	ò 149	Ñ 165	╡ 181	┼ 197	╒ 213	σ 229	⌡ 245
6	å 134	û 150	õ 166	╢ 182	╞ 198	╓ 214	µ 230	÷ 246
7	ç 135	ù 151	Õ 167	╖ 183	╟ 199	╫ 215	τ 231	≈ 247
8	ê 136	ÿ 152	¿ 168	╕ 184	╚ 200	╪ 216	Φ 232	° 248
9	ë 137	Ö 153	ã 169	╣ 185	╔ 201	┘ 217	Θ 233	∙ 249
A	è 138	Ü 154	Ã 170	║ 186	╩ 202	┌ 218	Ω 234	· 250
B	ï 139	ø 155	ℓ 171	╗ 187	╦ 203	█ 219	δ 235	√ 251
C	î 140	£ 156	ŉ 172	╝ 188	╠ 204	▄ 220	∞ 236	^{n} 252
D	ì 141	Ø 157	¡ 173	╜ 189	═ 205	▌ 221	φ 237	^{2} 253
E	Ä 142	Ŀ 158	^{3} 174	╛ 190	╬ 206	▐ 222	ϵ 238	■ 254
F	Å 143	ŀ 159	¤ 175	┐ 191	╧ 207	▀ 223	∩ 239	255

PC-850

	0	1	2	3	4	5	6	7
0	0	0 16	32	0 48	@ 64	P 80	` 96	p 112
1	! 1	1 17	! 33	1 49	A 65	Q 81	a 97	q 113
2	" 2	2 18	" 34	2 50	B 66	R 82	b 98	r 114
3	# 3	3 19	# 35	3 51	C 67	S 83	c 99	s 115
4	$ 4	4 20	$ 36	4 52	D 68	T 84	d 100	t 116
5	% 5	5 21	% 37	5 53	E 69	U 85	e 101	u 117
6	& 6	6 22	& 38	6 54	F 70	V 86	f 102	v 118
7	' 7	7 23	' 39	7 55	G 71	W 87	g 103	w 119
8	(8	8 24	(40	8 56	H 72	X 88	h 104	x 120
9	) 9	9 25	) 41	9 57	I 73	Y 89	i 105	y 121
A	* 10	: 26	* 42	: 58	J 74	Z 90	j 106	z 122
B	+ 11	; 27	+ 43	; 59	K 75	[91	k 107	{ 123
C	, 12	< 28	, 44	< 60	L 76	\ 92	l 108	| 124
D	- 13	= 29	- 45	= 61	M 77	] 93	m 109	} 125
E	. 14	> 30	. 46	> 62	N 78	^ 94	n 110	~ 126
F	/ 15	? 31	/ 47	? 63	O 79	_ 95	o 111	⌂ 127

PC-850

	8	9	A	B	C	D	E	F
0	Ç 128	É 144	á 160	░ 176	└ 192	ð 208	Ó 224	– 240
1	ü 129	æ 145	í 161	▒ 177	┴ 193	Ð 209	ß 225	± 241
2	é 130	Æ 146	ó 162	▓ 178	┬ 194	Ê 210	Ô 226	= 242
3	â 131	ô 147	ú 163	│ 179	├ 195	Ë 211	Ò 227	¾ 243
4	ä 132	ö 148	ñ 164	┤ 180	─ 196	È 212	õ 228	¶ 244
5	à 133	ò 149	Ñ 165	Á 181	┼ 197	ı 213	Õ 229	§ 245
6	å 134	û 150	ª 166	Â 182	ã 198	Í 214	µ 230	÷ 246
7	ç 135	ù 151	º 167	À 183	Ã 199	Î 215	þ 231	¸ 247
8	ê 136	ÿ 152	¿ 168	© 184	╚ 200	Ï 216	Þ 232	° 248
9	ë 137	Ö 153	® 169	╣ 185	╔ 201	┘ 217	Ú 233	¨ 249
A	è 138	Ü 154	¬ 170	║ 186	╩ 202	┌ 218	Û 234	· 250
B	ï 139	ø 155	½ 171	╗ 187	╦ 203	█ 219	Ù 235	¹ 251
C	î 140	£ 156	¼ 172	╝ 188	╠ 204	▄ 220	ý 236	³ 252
D	ì 141	Ø 157	¡ 173	¢ 189	═ 205	¦ 221	Ý 237	² 253
E	Ä 142	× 158	« 174	¥ 190	╬ 206	Ì 222	¯ 238	■ 254
F	Å 143	ƒ 159	» 175	┐ 191	¤ 207	▀ 223	´ 239	255

7.3 Resident font samples

PCL5 fonts

Courier 12-point (10 cpi)	!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~▒
Courier Bold 12-point (10 cpi)	**!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~▒**
Courier Italic 12-point (10 cpi)	*!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~▒*
Courier 10-point (12 cpi)	!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~▒
Courier Bold 10-point (12 cpi)	**!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~▒**
Courier Italic 10-point (12 cpi)	*!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~▒*
Line Printer 8.5-point (16.6 cpi)	!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~▒
Univers Medium	!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_'abcdefghijklmnopqrstuvwxyz{|}~▒

Univers Medium Italic	!"#$%&'()* +,-./0123456789:;< = >?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_'abcdefghijklmnopqrstuvwxyz{|}~▒
Univers Bold	!"#$%&'()* +,-./0123456789:;< = >?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_'abcdefghijklmnopqrstuvwxyz{|}~▒
Univers Bold Italic	!"#$%&'()* +,-./0123456789:;< = >?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_'abcdefghijklmnopqrstuvwxyz{|}~▒
CG Times	!"#$%&'()*+,-./0123456789:;< = >?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_'abcdefghijklmnopqrstuvwxyz{|}~▒
CG Times Italic	!"#$%&'()*+,-./0123456789:;< = >?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_'abcdefghijklmnopqrstuvwxyz{|}~▒
CG Times Bold	!"#$%&'()*+,-./0123456789:;< = >?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_'abcdefghijklmnopqrstuvwxyz{|}~▒
CG Times Bold Italic	!"#$%&'()*+,-./0123456789:;< = >?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_'abcdefghijklmnopqrstuvwxyz{|}~▒

TrueImage fonts

Arial	!"#$%&’()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_‘abcdefghijklmnopqrstuvwxyz{|}~
Arial Bold	!"#$%&’()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_‘abcdefghijklmnopqrstuvwxyz{|}~
Arial Bold Oblique	!"#$%&’()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_‘abcdefghijklmnopqrstuvwxyz{|}~
Arial Oblique	!"#$%&’()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_‘abcdefghijklmnopqrstuvwxyz{|}~
Arial Narrow	!"#$%&’()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_‘abcdefghijklmnopqrstuvwxyz{|}~
Arial Narrow Bold	!"#$%&’()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_‘abcdefghijklmnopqrstuvwxyz{|}~
Arial Narrow Bold Oblique	!"#$%&’()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_‘abcdefghijklmnopqrstuvwxyz{|}~
Arial Narrow Oblique	!"#$%&’()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_‘abcdefghijklmnopqrstuvwxyz{|}~
Century Schoolbook Bold	!"#$%&’()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_‘abcdefghijklmnopqrstuvwxyz{|}~
Century Schoolbook Bold Italic	!"#$%&’()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_‘abcdefghijklmnopqrstuvwxyz{|}~

Century Schoolbook Italic	!"#$%&’()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_‘abcdefghijklmnopqrstuvwxyz{|}~
Century Schoolbook Roman	!"#$%&’()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_‘abcdefghijklmnopqrstuvwxyz{|}~
Courier	!"#$%&’()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_‘abcdefghijklmnopqrstuvwxyz{|}~
Courier Bold	!"#$%&’()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_‘abcdefghijklmnopqrstuvwxyz{|}~
Courier Bold Oblique	!"#$%&’()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_‘abcdefghijklmnopqrstuvwxyz{|}~
Courier Oblique	!"#$%&’()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_‘abcdefghijklmnopqrstuvwxyz{|}~
ITC Avant Garde Gothic Book	!"#$%&’()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_‘abcdefghijklmnopqrstuvwxyz{|}~
ITC Avant Garde Gothic Book Oblique	!"#$%&’()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_‘abcdefghijklmnopqrstuvwxyz{|}~
ITC Avant Garde Gothic Demi	!"#$%&’()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_‘abcdefghijklmnopqrstuvwxyz{|}~
ITC Avant Garde Gothic Demi Oblique	!"#$%&’()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_‘abcdefghijklmnopqrstuvwxyz{|}~

ITC Bookman Demi	**!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_'abcdefghijklmnopqrstuvwxyz{|}~**
ITC Bookman Demi Italic	***!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_'abcdefghijklmnopqrstuvwxyz{|}~***
ITC Bookman Light	!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_'abcdefghijklmnopqrstuvwxyz{|}~
ITC Bookman Light Italic	*!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_'abcdefghijklmnopqrstuvwxyz{|}~*
ITC Zapf Chancery Medium Italic	*!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_'abcdefghijklmnopqrstuvwxyz{|}~*
ITC Zapf Dingbats	✁✂✃✄☎✆✇✈✉☛☞✌✍✎✏✐✑✒✓✔✕✖✗✘✙✚✛✜✝✞✟✠✡✢✣✤✥✦✧★✩✪✫✬✭✮✯✰✱✲✳✴✵✶✷✸✹✺✻✼✽✾✿❀❁❂❃❄❅❆❇❈❉❊❋●❍■❏❐❑❒▲▼◆❖◗❘❙❚❛❜❝❞
Symbol	!∀#∃%&∍()∗+,−./0123456789:;<=>?≅ΑΒΧΔΕΦΓΗΙϑΚΛΜΝΟΠΘΡΣΤΥςΩΞΨΖ[∴]⊥_‾αβχδεφγηιϕκλμνοπθρστυϖωξψζ{|}~
Times New Roman	!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_'abcdefghijklmnopqrstuvwxyz{|}~
Times New Roman Bold	**!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_'abcdefghijklmnopqrstuvwxyz{|}~**
Times New Roman Bold Italic	***!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_'abcdefghijklmnopqrstuvwxyz{|}~***

Times New Roman Italic	*!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_'abcdefghijklmnopqrstuvwxyz{|}~*
Zapf Calligraphic Bold	**!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_'abcdefghijklmnopqrstuvwxyz{|}~**
Zapf Calligraphic Bold Italic	***!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_'abcdefghijklmnopqrstuvwxyz{|}~***
Zapf Calligraphic Italic	*!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_'abcdefghijklmnopqrstuvwxyz{|}~*
Zapf Calligraphic Roman	!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_'abcdefghijklmnopqrstuvwxyz{|}~

Glossary

Absolute movement Movement of the cursor relative to the coordinate system origin.

Absolute plotting Drawing using coordinates relative to the coordinate system origin.

Addressable area See logical page.

Anchor point The top left-hand corner of the PCL picture frame.

Anisotropic scaling GL2 scaling mode where x- and y-axis units can be of different sizes.

Ascender Part of a character that extends upwards above the level of most other characters, for example the top parts of a ‘k’ or ‘l’.

ASCII codes Codes (0-255) identifying alphabetic, numeric and control code characters.

Attribute A characteristic of a font or character.

Baseline An imaginary line on which characters lie. Most characters sit squarely on the baseline, however, some extend below the baseline.

Bitmap font A font comprised of characters defined as patterns of dots. Bitmap font characters cannot be scaled.

Bold Thicker type, used to make text more prominent.

Boolean A TrueImage variable type that can have two possible values - true or false.

Bounding box The smallest enclosing upright rectangle into which a character will fit.

CTM See Current transformation matrix.

Caching Storage of character bitmaps that have been calculated from character definition outlines. TrueImage performs font caching in order to avoid recalculating a character's bitmap pattern every time it prints the character.

Calling a macro Macro invocation in which any changes made to the modified print environment are temporary.

Cartridge A storage medium for fonts and macros that can be inserted into the printer's cartridge slot, permitting the use of more fonts and macros without using up printer memory.

Cartridge font A font supplied on a cartridge. Cartridge fonts are available from a number of different vendors.

Cartridge macro A macro stored on cartridge. Users can create their own macros and copy them onto cartridge.

Character cell An imaginary rectangular box surrounding a character that defines its placement relative to other characters.

Character code A number that uniquely identifies a character.

Character descriptor A block of data that describes characteristics of a downloadable font character.

Character set See Symbol set.

Clipping path The path to which page output is clipped. In TrueImage emulation mode, this may be any shape.

Column A vertical sub-division of the page defined by the HMI (horizontal motion index). The PCL cursor moves one column width across the page when a monospaced font character is printed, or when the space character of a proportionally spaced font is printed. See also HMI.

Control code An ASCII code that instructs the printer to perform a particular function, for example a carriage return.

Cross-hatching Criss-cross diagonal shading.

Current path The path that is currently being built-up by TrueImage path operators, and which may be rendered using paint operators. See also path.

Current position Position in TrueImage user space from which path definition may proceed.

Current settings The settings with which the printer is currently working, as established by control panel settings and software commands.

Current transformation matrix Matrix that translates TrueImage user space coordi nates to the coordinates used internally by the printer device space coordinates.

Current units The currently effective GL2 coordinate system units - plotter units or user units. User units are defined using the **SC** command.

Cursor position The position on the current page from which printing or cursor movement can proceed.

Decipoint A unit equal to 1/720".

Descender The lower part of a character, such as a 'y' or 'g' that extends below the baseline.

Destination image Text and graphics that have already been committed to the page. The LaserJet III print model defines the interaction between the source and destination images.

Device space In TrueImage mode, the printer's own internal coordinate system, which is usually transparent to the user.

Dictionary A table associating keys (names) with values. TrueImage uses dictionaries to store font data (character names are associated with the procedures that render them) and also to associate procedure and operator names with their actions.

Dot A unit equal to 1/300"

Downloadable font A font that can be downloaded to the printer from a host computer. Downloaded fonts reside in printer memory.

Downloading The action of transferring a font, macro or page description file from a host computer to the printer's memory.

Effective window A rectangular area on a page within which GL2 graphic output will be visible. The effective window is the intersection of the logical page, picture frame, printable area and input window.

Emulation mode A mode in which the printer imitates the functionality of another printer or class of printer.

Enable for overlay Macro invocation whereby a macro is run as the final operation before every page is printed. Overlaid macros use the settings of the macro overlay environment.

Escape sequence A sequence of character codes starting with an ESC character, which is followed by one or more other characters. PCL5 printer commands are implemented as escape sequences.

Even-odd rule A rule that defines whether a point lies inside a path or not, for the purpose of filling the path. If a line from the point to another point that lies outside the path is crossed an odd number of times by path segments, the original point lies inside the path; otherwise it lies outside. See also the zero-winding rule.

Factory default environment Printer settings made before the printer is sent out from the factory. Factory settings can be restored from the control panel.

Fill Shading applied to a shape or character.

Fixed spacing See monospacing.

Font A collection of characters with common attributes. Printer fonts may be resident in printer ROM, may be read from cartridge or downloaded from a host computer.

Font descriptor A block of data describing common characteristics of a font's characters.

Font dictionary A TrueType or PostScript font is represented as a dictionary - a table of keys and values that associates the name of each character with a procedure to render the character.

Graphics state In TrueImage mode, a collection of settings that determine the way in which path construction and painting operators are interpreted. Graphics states may be saved and restored.

Gray scale Shade of gray that ranges from 0%, white, to 100%, black (HP LaserJet III mode), or from 0, black, to 1, white (TrueImage).

Half-tone A pattern of black and white dots designed to simulate a gray scale.

Hard clip limits The area of the page on which the printer can print visible GL2 output - equivalent to the PCL printable area.

Hatching Parallel-line shading.

Height The height of a font measured from the top of the highest ascender, to the bottom of the lowest descender. PCL5 fonts are measured in typographic points(1/72"); TrueImage fonts are specified in terms of the current unit size.

HMI Horizontal motion index. The width of a single column. This is the horizontal distance the PCL5 cursor moves across the page when printing a single monospaced font character, or the space character of a proportionally-spaced font. The HMI may be set using PCL5 commands. See also Column.

Horizontal plot size The horizontal size of a GL2 graphic image that is to be imported. The specification of horizontal and vertical plot sizes allows images to be fitted exactly into the picture frame.

Initial settings A collection of printer settings consisting of all the current control panel settings. A software or control panel reset restores the initial settings, without changing the current emulation.

Input window A rectangular area, defined by the **IW** command, outside which no GL2 output can appear. The input window is sometimes referred to as the soft clip limits.

Internal font A font that is resident in the printer's ROM, such as Univers in HP LaserJet III mode or Times New Roman in TrueImage mode. Each mode has a number of these fonts, which can be selected at any time that the printer is in that mode.

Interpreter The software in the printer that executes the commands in TrueImage page description programs and any other TrueImage software.

Isotropic scaling GL2 scaling mode in which x- and y-axis units must be the same size.

Justification The alignment of text output on the page. Left justification aligns the left edge of every line; right justification aligns the right edge of every line.

Label A GL2 text string.

Landscape A page orientation that sets the long edge of the page as the top edge.

Line attribute Line end type, line join type or miter limit.

Logical page The area of the PCL physical page within which the cursor may be positioned. The logical page can be repositioned on the physical page.

Macro A sequence of PCL5 commands that the user downloads to printer memory or onto cartridge. A single command causes the macro to be run. There are three ways of running a macro: calling a macro, executing a macro and enabling a macro for overlay.

Macro execution Macro invocation in which any changes made to the modified print environment are retained after macro execution has finished.

Macro overlay environment Environment used by a macro enabled for overlay. The macro overlay environment is a combination of the user default environment and the modified print environment.

Medium Type of normal line thickness - used for body copy.

Miter length The length of the spike formed by the intersection of two lines that join at an angle. The miter length is the distance between the inside and outside corners of the line join.

Miter limit The maximum permitted ratio of miter length to line width. Line joins whose miter length would exceed the miter lirnit are clipped to a different shape.

Modified print environment Environment consisting of all current HP LaserJet III printer settings. If a macro is called or enabled for overlay, the modified print environment is saved and then restored when the macro has run.

Monospacing Font spacing type where each character occupies an equal horizontal space on a line of text. Courier fonts are monospaced fonts.

Object Element in a TrueImage program.

Operator Built-in TrueImage command.

Path A sequence of connected and disconnected points, straight lines and curves that defines a shape and its position on the page. See also subpath, current path and clipping path

Pattern A hatching pattern or gray scale that can be used to fill a shape or character.

Pattern transparency The patterned (non-white) areas of a source image can be either transparent or opaque. If transparent, the destination image will be visible through any white parts of the source image's patterned areas. If opaque, the destination image will not be visible at all through the patterned areas of the source image.

PCL Printer Control Language. PCL5 commands control the printer in HP LaserJet III mode.

Pen Imaginary pen whose movements plot or define shapes in GL2 mode. There are two pens available - white and black. A pen must be selected before any lines can be drawn.

Perforation skip A function prohibiting the printer from printing text below the bottom margin. Text flows onto the next page instead. In PCL5 mode perforation skip may be turned on or off.

Permanent font In HP LaserJet III mode, a downloaded font that is retained when a printer reset is performed.

Permanent macro In HP LaserJet III mode, a macro in printer memory that is retained when a printer reset is performed.

Physical page The medium (paper, overhead projection slide or envelope) on which output is printed.

Picture frame The area of the physical page within which GL2 output can appear. The size and position of the picture frame can be set using PCL commands.

Pitch The number of monospaced font characters in an inch of text.

Plot An image rendered by GL2 commands.

Plotter units The default GL2 coordinate system units. 1 plotter unit = 1/1016".

Point The standard unit of font height. 1 point = 1/72.27".

Point factor scaling GL2 scaling mode where x- and y-axis units are specified as multiples of plotter units. x- and y-axis units can be of different sizes.

Point size See height.

Polygon A shape comprising one or more closed sets of connected lines.

Polygon buffer An area of printer memory set aside for storing polygons. Some GL2 commands can reference the buffer explicitly, while others use it automatically.

Portrait A page orientation in which the side edges of the page are longer than the top edge.

Posture A characteristic of a font. A font can be upright or italic (oblique).

Primary font One of two font definitions that are always maintained in PCL mode.

Print model A way of considering the interaction between different graphic elements. The HP LaserJet III print model describes the interaction in terms of a source image, a pattern and a destination image.

Printable area The area of the physical page in which the printer can place output.

Print position The current cursor position.

Proportional spacing Font spacing type in which the horizontal space occupied by each different character in a line of text varies according to its design. Univers and Times fonts are proportionally-spaced.

RAM (Random Access Memory), the printer's memory. The printer uses its memory to compose each page of output before printing it, to store downloaded fonts and macros, and to store other necessary data, such as current environment settings.

Raster graphics Graphic images made up of successive lines of zeroes and ones that represent white areas and patterned areas.

Relative movement Cursor movement relative to the current cursor position.

Relative plotting Drawing using coordinates relative to the current pen position.

Reset A printer reset restores the printer's initial settings. A reset may be performed from the control panel or in software.

ROM (Read Only Memory), the printer's ROM memory contains its emulation mode software and the internal fonts. The contents of ROM cannot be altered from a host computer.

Row A horizontal sub-division of the page, defined by the VMI (vertical motion index). A line feed causes the PCL cursor to move down the page one row. See also VMI.

Sans serif A typeface normally used for headings, headlines and other text that is to be prominently displayed. Sans serif characters lack the small curly hooks (serifs) that make serif-font body text more readable.

Scalable font A font comprised of characters defined as outlines. The user may select the font in any size - the printer automatically scales the characters to the required size. Compare bitmap font.

Scaling In GL2 mode, setting the size of coordinate system units using the **SC** command, to determine the size of graphic output. Three types of scaling are available: anisotropic, isotropic and point factor.
In TrueImage mode, setting the ratio of device space units to user space units, in order to set the size of output.

Scaling points The reference points, P1 and P2, which establish the position of GL2 output. The scaling points can be positioned using the **IP** and **IR** commands.

Scan conversion The conversion of the output described in a TrueImage page description to the dot pattern that the printer applies to the page.

Secondary font One of two font definitions that are always maintained in PCL mode.

Serif A typeface normally used for body text. *Serif* typeface characters have small curly hooks (serifs) that serve to make *serif*-font body text more readable.

Soft clip limits See Input window.

Source image In the LaserJet III print model, graphic image that is superimposed onto the destination image. The current source and pattern transparency settings determine the resultant output.

Source transparency A source image can be either transparent or opaque. If transparent, the destination image will be visible through white parts of the source image. If opaque, the destination image will not be visible at all through the source image.

Stack A data structure used by TrueImage to process TrueImage code. The object placed on the stack most recently must be retrieved first. TrueImage also uses stacks to store graphics states, virtual memory states and environments.

Stick font The default GL2 font, designed for use in technical drawings. Stick font characters are comprised of thin straight lines.

Stroke weight The thickness of character strokes. The normal stroke weight is Medium. Other common weights are Bold, Black and Light.

Subpath A series of connected line segments, forming a shape. A TrueImage path is made up of one or more subpaths.

Sub-polygon A single closed set of connected lines, forming a shape. A GL2 polygon is made up of one or more sub-polygons.

Symbol set A set of printable characters. Character sets usually include the alphabet in upper- and lowercase, the digits 0-9, punctuation symbols and some additional characters. There are many specialized character sets, used for special purposes, such as printing foreign language characters.

Temporary font In HP LaserJet III mode, a downloaded font that is not retained when a printer reset is performed.

Temporary macro In HP LaserJet III mode, a macro in printer memory that is not retained when a printer reset is performed.

Text area The area of the physical page on which text can be printed.

Text direction The direction in which text is printed, relative to the physical page's orientation.

TIFF (Tagged Image File Format), a compressed raster graphics file format.

Transparency See pattern transparency and source transparency.

Typeface The design of a font's characters. Typefaces are designed so that the individual character shapes work together to produce visually pleasing, readable text.

User default environment In HP LaserJet III mode, an environment that is a combination of the factory default settings and the control panel settings. The user default environment takes effect on power-up in HP LaserJet III mode, or when HP LaserJet III mode is entered from another emulation mode. The printer can be reset to user default settings either from the control panel or in software with the **<ESC> E** command.
The user-default environment settings are equivalent to the initial settings.

User space TrueImage's coordinate system. User space coordinates referenced in TrueImage page description programs are translated to the printer's device space coordinates.

User units GL2 coordinate system units specified with the **SC** command.

Vertical plot size The vertical size of a GL2 graphic image that is to be imported. The specification of horizontal and vertical plot sizes allows images to be fitted exactly into the picture frame.

Virtual memory In TrueImage mode, an area of printer memory in which the values of TrueImage arrays, dictionaries and strings are stored. Snapshots of virtual memory may be saved and restored.

VMI Vertical motion index. The height of a single row. The horizontal distance that the PCL5 cursor moves across the page when a single monospaced font character or the space character of a proportionally-spaced font is printed. The VMI may be set using PCL5 commands. See also Row.

Zero-winding rule A rule that defines whether a point lies inside a path or not, for the purpose of filling the path. If a line from the point to another point that lies outside the path is crossed an equal number of times from left to right and from right to left by path segments, the original point lies outside the path; otherwise it lies inside. See also the even-odd winding rule.

MEMO

Index

Consumer Response

Star Micronics Co., Ltd. invites your suggestions and comments on your printer and this manual. Please address your correspondence to:

Worldwide Headquarters:
STAR MICRONICS CO., LTD.
20-10 Nakayoshida
Shizuoka, JAPAN 422-91
Attn: Product Manager

American Market:
STAR MICRONICS AMERICA, INC.
420 Lexington Avenue, Suite 2702-25
New York, NY 10170
Attn: Product Manager

European Market:
STAR MICRONICS DEUTSCHLAND GMBH
Westerbachstraße 59
P.O. Box 940330
D-6000 Frankfurt/Main 90
F.R. of Germany
Attn: Product Manager

U.K. Market:
STAR MICRONICS U.K., LTD.
Star House
Peregrine Business Park
Gomm Road, High Wycombe
Bucks. HP13 7DL, U.K.
Attn: Product Manager

French Market:
STAR MICRONICS FRANCE S.A.R.L.
25, rue Michaël Faraday
78180 Montigny-le-Bretonneux
Attn: Product Manager

Asian Market:
STAR MICRONICS ASIA LTD.
18/F Tower 2, Enterprise Square
9 Sheung Yuet Road, Kowloon Bay, HONG KONG
Attn: Product Manager

PRINTED IN JAPAN