

LS-5 EX
LS-5 TT

APPLICATIONS MANUAL

Trademark acknowledgements

Macintosh, System 7, TrueType: Apple Computer Inc.

PostScript, Font Downloader: Adobe Systems Inc.

PageMaker: Aldus Corporation

HP, LaserJet, LaserJet III, PCL, PCL5: Hewlett-Packard Company

MicroSoft, MS-DOS, Windows, TrueImage: MicroSoft Corporation

IBM, IBM-PC: International Business Machines Corporation

Palatino, Helvetica, Univers: Linotype AG and/or its subsidiaries

Times, Linotron: Allied Corporation

Intellifont, CG Times: Agfa Corporation

ITC Avant Garde, ITC Bookman, ITC Zapf Chancery, ITC Zapf Dingbats: International Typeface Corporation

Times New Roman: Monotype Corporation plc.

- All other brand names are trademarks or registered trademarks of their respective holders.
- All rights reserved.
- Reproduction of any part of this manual in any form without the express permission of Star Micronics is forbidden.

© 1992 Star Micronics Co., Ltd.

TABLE OF CONTENTS

1. Introduction to the printer	1
1.1 The printing process	1
1.2 Emulations	2
1.3 Fonts	2
1.4 User needs	3
1.5 Binary, decimal and hexadecimal numbers	4
1.6 General advice	5
2. Controlling the printer	7
2.1 On-line	7
2.2 Program mode	8
2.3 The control panel	9
2.4 Printer settings	11
2.5 Setting parameters	12
2.6 Feeder select	20
2.7 Mode	20
2.8 Selecting display language	20
2.9 Errors and status messages	21
2.10 PCL	21
2.11 TrueImage	21
2.12 Hex dump mode	22
2.13 Superset commands	22
2.14 Fonts	22
3. Fonts	23
3.1 Introduction	23
3.2 Font attributes	27
3.3 Printer fonts	30
3.4 Font sources	33
3.5 Font selection	34
3.6 Special symbols and characters	35
3.7 How applications use fonts	36
3.8 Conclusion	37

- 4. Printer Control Language** 39
 - 4.1 Introduction..... 39
 - 4.2 Printer control language commands 40
 - 4.3 Command format 42
 - 4.4 The buffer..... 43
 - 4.5 The imaginary cursor 43
 - 4.6 The page..... 44
 - 4.7 The PCL coordinate system..... 47
 - 4.8 The printing environment 48
 - 4.9 General printer control commands 52
 - 4.10 Fonts..... 75
 - 4.11 Graphics..... 101
 - 4.12 Macros 112

- 5. Vector graphics** 117
 - 5.1 GL2 concepts 117
 - 5.2 Managing GL2 mode from PCL mode..... 119
 - 5.3 GL2 syntax..... 123
 - 5.4 Programming with GL2..... 125
 - 5.5 GL2 graphics commands 127

- 6. TrueImage** 193
 - 6.1 Introduction..... 193
 - 6.2 TrueImage print model 196
 - 6.3 Coordinate systems 198
 - 6.4 Graphics state..... 199
 - 6.5 TrueImage language features..... 200
 - 6.6 Fonts..... 211
 - 6.7 Graphic effects 220
 - 6.8 Operators..... 223

- 7. Technical supplement** 293
 - 7.1 Command summary..... 293
 - 7.2 Character set tables 305
 - 7.3 Resident font samples..... 335

- Glossary** 341

- Index** 355

Introduction to the printer

CHAPTER 1

This chapter is a general introduction to the printer, describing the capabilities that it offers, previewing the various topics that are covered more fully in the remaining chapters, and providing some general information that may be useful and of interest to the reader.

1.1 The printing process

Data transferred to the printer is composed into pages by the printer's internal software. When a complete page has been prepared in memory and is ready for printing, it can be output.

The physical process of transferring a page of text and graphics from memory onto paper is carried out by the printer's engine. The engine has a laser beam that it can point, via a series of lenses and mirrors, onto the surface of a rotating drum. Initially the entire surface of the drum has a positive electrical charge. The laser beam scans back and forth across the drum, hitting the drum's surface at selected points under the control of the engine. Wherever the laser beam hits the drum, that point on the drum is set to a neutral charge. The laser builds up the page's image on the drum as an array of neutrally-charged points.

As the drum rotates past the toner compartment, particles of black toner are attracted to the electrically neutral spots on the drum. As the drum continues to rotate it meets the paper. The paper has been negatively charged by a thin wire, known as a corona wire. The paper and drum are pressed together, attracted by their opposite electrical charge. High temperature, and pressure applied by a roller mechanism, combine to fuse the toner to the paper thus transferring the image from the drum onto the paper. The finished page is then ejected.

This is a well-established process that produces consistent, high-quality output, and which requires minimal maintenance.

1.2 Emulations

The LS-5EX printer model is supplied as standard with Hewlett Packard's Printer Control Language, PCL, in combination with the GL2 graphics language. TrueImage, MicroSoft's PostScript-compatible page description language, is available on an optional board which can be installed in the LS-5EX.

The LS-5TT is supplied as standard with both PCL and GL2, and TrueImage emulations.

PCL and PostScript are the two major worldwide small-computer system printing standards; hence this printer offers a comprehensive solution to home and office printing needs.

PCL/GL2 is the de facto standard for IBM PC word-processing and CAD applications.

PostScript has revolutionized high-quality document and illustration production, spawning an entire industry in the shape of desktop publishing. Although closely associated with Apple Macintosh computers, PostScript output can also be generated by IBM PC-based DOS and Windows applications, and those on a variety of other platforms.

TrueImage is an exact clone of PostScript, allowing any PostScript document to be output as if on a PostScript printer. TrueImage's inherent font technology, TrueType, was pioneered by Apple as part of their System 7 operating system software, and has also been incorporated into Windows 3.1. TrueImage will also support any PostScript font.

Star's implementations of PCL and TrueImage contain a number of enhancements, that bring features such as paper tray-handling, paper-size selection and emulation-switching under software control.

1.3 Fonts

As already mentioned, both emulations include a number of built-in fonts. Further fonts may be obtained from commercial vendors in a variety of forms: on disk, CD-ROM and cartridge (PCL only) and made available for printing. Font cartridges simply plug in to the font cartridge slot. Disk or CD-ROM based fonts should first be copied to the host computer's hard disk and then downloaded to the printer's memory. Often applications download the fonts they use automatically. However, failing this, utilities for the express purpose of font-downloading also exist, and are often distributed with commercial fonts.

Should you wish to create your own PCL or TrueType fonts, font creation and modification is possible directly within PCL and TrueImage. However, font creation applications are available commercially and represent a more practical, simpler alternative.

1.4 User needs

Potential users of the printer range in a broad spectrum from normal users, who simply wish to print their application documents, through more sophisticated users, who also print from their applications but who sometimes need to be able to exert a closer degree of control over the printing process, to application developers, who develop software to drive the printer directly.

The first and second categories of user are well served by the immense range of software applications that may be used with this printer, including word-processors, spreadsheets, desktop publishing programs and illustration, drawing and computer-aided design packages. Any software that will generate PostScript or print to any model in the HP LaserJet series will work with this model.

Normal users will have little need of this manual, as all the operational information they require will be contained in the Operations Manual accompanying this printer, and the reference manuals that accompany their application packages. This application manual may be of interest to these users, however, in demonstrating the correspondence between the internal methods of page control/page description and the high-level commands and option settings available to them in their applications.

The middle category of users (those who sometimes need to program their own utilities, hand-craft graphic output, create custom fonts and other special effects, or modify existing printable files) will find this manual a useful source of reference in explaining the mechanisms of the emulation languages and the details of their commands. This category of users may include desk-top publishers, font designers, system support staff, and any other people whose specific goals entail a certain amount of programming.

The third category of users (those creating full-blooded applications) will find this manual a comprehensive reference source for PCL, GL2 and TrueImage, which should enable them to generate output in a suitable form and to create programs that drive the printer successfully.

1.5 Binary, decimal and hexadecimal numbers

When counting, people almost always use the decimal number system (base 10). In the decimal system the digits 0 – 9 are used to form numbers in which each digit's significance depends on its position in the number; by convention each digit multiplies a value ten times greater than the digit to its right. Hence the number 4523 is interpreted as:

$$(4 \times 10 \times 10 \times 10) + (5 \times 10 \times 10) + (2 \times 10) + (3 \times 1).$$

4523 is simply the universally recognised form of the number.

Two more number systems that are of great importance in the world of computers are the binary (base 2) and hexadecimal (base 16). All computers represent information internally in the form of binary numbers. In this system the digits 0 and 1 are used, and each digit in a number multiplies a value twice that of the digit to its right. Hence the number 10110 is interpreted as $(1 \times 2 \times 2 \times 2 \times 2) + (0 \times 2 \times 2 \times 2) + (1 \times 2 \times 2) + (1 \times 2) + (0 \times 1)$.

In the hexadecimal system the digits 0 – 9 and A – F (or a – f) are used. A – F represent the base ten values 10 – 15. Each digit in a number multiplies a value sixteen times that of the digit to its right. Hence the number 9F3E equals $(9 \times 16 \times 16 \times 16) + (15 \times 16 \times 16) + (3 \times 16) + (14 \times 1)$, which equals 40766 in base 10.

Binary numbers can easily be converted to hexadecimal numbers, and vice versa. To convert a binary number 1011110101101110 into hexadecimal, first split it up into blocks of four binary digits (bits), 1 0111 1101 0110 1110, and then convert each block to its hexadecimal equivalent, in this case 17D6E. To convert a hexadecimal number to binary, simply convert each hexadecimal digit to its binary equivalent, and then string together the resulting binary values; hence A82 is made up of 1010 (A), 1000 (8) and 0010 (2). Thus the binary equivalent of A82 is 101010000010.

Since binary and hexadecimal numbers are so easily interchanged, hexadecimal notation is a good medium for bridging the the gap between the numerical requirements of humans, who want to use numbers that do not contain long strings of digits, and computers, which can only handle two states internally: zero and one. Hence hexadecimal numbers are often used to specify character codes, and are also used when the printer outputs the raw data that it receives (in hex dump mode)

The following table compares the three number systems.

Decimal	Binary	Hexadecimal	Decimal	Binary	Hexadecimal
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	A
3	0011	3	11	1011	B
4	0100	4	12	1100	C
5	0101	5	13	1101	D
6	0110	6	14	1110	E
7	0111	7	15	1111	F

1.6 General advice

Personal computer technology is a fast-evolving, ever-changing field in which new software products, capabilities and standards are announced almost daily. To keep abreast of new possibilities, try to refer frequently to industry magazines that cover Macintosh, IBM PC, desktop publishing and related topics. These contain informative articles, latest product announcements and many useful hints for solving problems, resolving incompatibilities and generally getting the most out of your system. Similarly, on-line bulletin boards are also a good source of relevant information, advice and encouragement.

MEMO

Controlling the printer

CHAPTER 2

The printer is controlled in two ways: either by software running on a host computer, or by use of the printer's control panel. Use of the control panel is covered fully in the Operations Manual that accompanies the printer. A short overview is given here. Software control of the printer is covered fully in Chapters 4, 5 and 6 of this manual and is briefly touched on in this chapter.

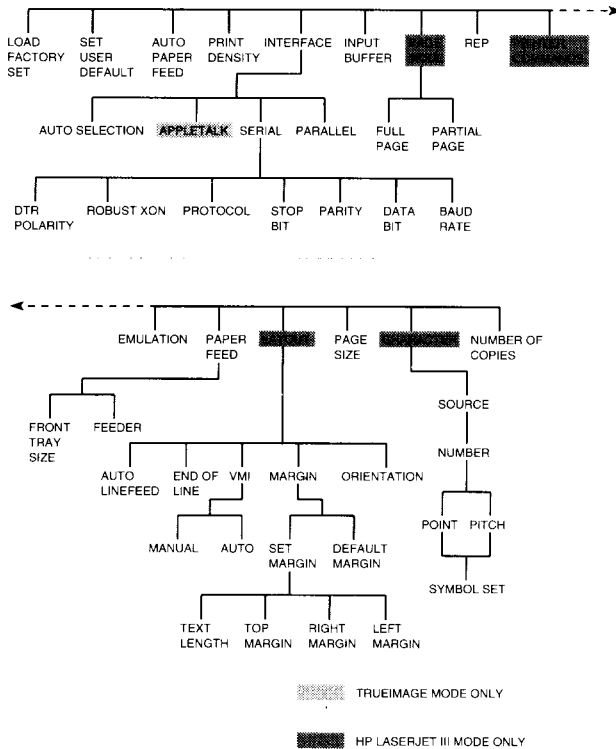
2.1 On-line

The printer is on-line when it is in a state capable of receiving commands from a host computer and transforming them into printed output. On-line should be the printer's normal operational state. Changes can only be made to printer settings via the control panel when the printer is off-line; this avoids any conflict that might arise if the printer were able to receive host data and control panel settings simultaneously.

A control panel button is used to set the printer on- and off-line, and a status LED indicates the current state.

2.2 Program mode

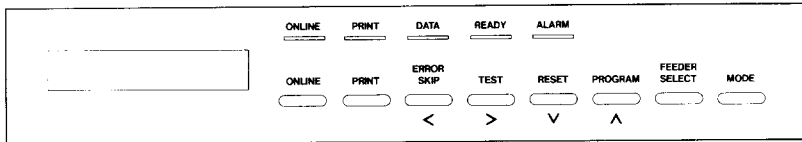
When the printer is off-line, you can set and alter various parameters, such as the current emulation, font selection, spacing, interface etc., to suit your needs. In the main these settings are made in program mode. In program mode you can step through menus of parameters and the range of their possible values, view their current settings and change them as appropriate. The settings are grouped and organised hierarchically, as a 'tree'. You can step through available parameter options, go down to the level of sub-options, or go up a menu level using the control panel arrow buttons.



To enter program mode, press the **(PROGRAM)** button. This enables the **(ERROR SKIP)**, **(TEST)**, **(RESET)** and **(PROGRAM)** buttons as arrow buttons. The Program mode functions of these buttons is explained below.

2.3 The control panel

The control panel consists of an LCD screen, five status LED lights and eight buttons. The LEDs provide information on the progress of printing jobs, and the buttons are used to set the printer on- or off-line and to make printer settings.



2.3.1 LEDs

The LEDs' significance is as follows:

On-line - lit when the printer is on-line, and not lit when it is off-line.

Print - lit when a page is in the process of being fed through the machine, and unlit otherwise.

Data - lit or blinking when print data is being processed in the printer. If the printer is powered off or reset while the data light is on (or blinking), data will be lost.


Ready - lit when the printer is ready to receive data, irrespective of whether the printer is on-line. The ready light blinks when the printer is warming up.

Alarm - lit when an error has occurred.

2.3.2 Buttons

The effects of pressing the control panel buttons are as follows:

ON LINE

sets the printer on-line and disables all other buttons (except, under certain circumstances, the  button).

PRINT

causes any page data currently held in the printer to be printed out.

ERROR SKIP / <

- (i) if an error has been detected, checks again to see if the error condition has been cleared, and if it has, restores the printer to a working state
- (ii) in program mode pressing this button causes the next parameter or parameter value (depending on the current level) to be displayed on the LCD display.

TEST / >

- (i) prints a test sheet, or font list according to the current emulation setting
- (ii) in program mode pressing this button causes the previous parameter or parameter value (depending on the current level) to be displayed on the LCD screen.

RESET / v

- (i) holding down this button resets the printer to “Initial setting” values for the current emulation. The term “Initial settings” is described in the following section, “Printer settings”.
- (ii) in program mode, if a value is shown on the LCD display, pressing the button selects the currently-displayed value as the setting for the current parameter. If a parameter is displayed, pressing the button moves down a level in the menu tree, either displaying the first in the next level of options, or the first of the available values for the current parameter.

PROGRAM / ^

- (i) enters program mode
- (ii) in program mode pressing this button moves back up the menu tree, to the next highest parameter group. If the current level is the highest level, pressing the button exits program mode.

FEEDER SELECT

selects the paper feed.

MODE

selects one of two sets of user default printer settings (mode 1 or mode 2).

2.4 Printer settings

At any given time, the printer's parameter settings, such as current emulation, font selection, spacing, and interface, define how the printer will respond to and interpret data and instructions it receives from a host computer. There are several useful ways to store collections of settings and to revert to them when necessary.

There are essentially four distinct collections of printer settings.

1) **Factory settings.** This is the group of settings programmed into the printer at the factory. The factory default settings may be restored at any time, and cannot be altered. There are two available versions of the factory default settings: US and EC. Restoring the printer's factory settings does not affect the current emulation setting

2) **User settings.** This is the group of settings which take effect when the printer is switched on or when a hard reset is performed (using the **MODE** button). There are two versions of the user default settings: mode 1 and mode 2. The parameter values that comprise each mode are set and stored in program mode. When the printer is first sent out from the factory, both mode 1 and mode 2 are the same as the EC factory default settings.

3) **Initial settings.** This is a single collection of printer and emulation settings, consisting of all the currently effective control panel settings. Initial setting parameter values are denoted on the LCD screen by an @ symbol.

On power-up, or after a hard reset, the initial settings take on the values of the mode 1 default settings. Subsequent settings made using the control panel become initial settings.

A soft reset (either made using the **RESET** button, or received as a software command from the host) causes all the current emulation's parameter settings to be reset to their initial setting values. Hence any that have been changed by software commands from the host computer are changed back. A soft reset does not change the current emulation. However, a hard reset always restores the mode 1 user default emulation setting.

4) **Current settings.** These are the settings with which the printer is currently working, that is, a combination of the initial settings and settings made by software commands in the current emulation. Virtually all parameters that can be set from the control panel can also be set in software. Hence, settings such as current font selection or margin size, may have been determined by either method.

2.5 Setting parameters

The following settings are available in program mode. To enter program mode first make sure that the printer is off-line, then press the **PROGRAM** button. “Number of copies” appears on the display. Use the **</>** buttons to scroll through the list of available parameters, and the **∇** button to select a parameter to be set. The parameters available depend on the current emulation.

2.5.1 Number of copies

(HP LaserJet III emulation and TrueImage)

The first level of options allows you to select between “One” and “Multiple” copies. Use the **</>** buttons to display the options in turn, and the **∇** button to select the option you require. To print a single copy, select “One”; to select a number of copies select “Multiple”, then use the **</>** buttons to display the number of copies you require, and confirm your selection with the **∇** button.

A single copy is the factory default setting.

2.5.2 Character

(HP LaserJet III emulation)

This feature allows you to set the current font. The behaviour of the control panel **∇** button is slightly different in this option: pressing **∇** makes a value selection and puts up a new menu. In other option menus it does one of these but not both.

First specify the font you require by source (Resident font, Cartridge font or downloaded “Soft” font); these are denoted by R, C and S respectively. Use the **</>** buttons to select the source, then confirm your selection with the **∇** button. A new menu appears listing available fonts in the selected source by number. Use the **</>** buttons to step through the available fonts until you reach the number of the font wish to select, then press the **∇** button to select it.

If the selected font is a bitmap font, a menu of available symbol sets is now shown. Use the **</>** buttons to select a symbol set, then confirm your selection with the **∇** button.







If you specified a scalable font, the font number menu is succeeded either by a list of available point sizes (for proportionally-spaced fonts) or by a list of available pitch settings (for monospaced fonts). Use the **</>** buttons to select a point size or pitch setting, then confirm your selection with the **∇** button. A list of available symbol sets is now displayed. Select one as described above.

If a soft font is selected, the character setting is not stored when a “Set user default” setting is made.

2.5.3 Page size

The following page sizes are supported.




Paper	Envelope
Letter	Monarch
Legal	COM-10
A4	International DL
B5	International C5
Executive	




Use the / buttons to display the options in turn, and the  button to select a page size. If you select A4 size in HP LaserJet III emulation mode, a further option menu appears, offering a choice of "Right End" widths of either 77 or 80. This selects a printable area width of 7.7" or 8.0". Use the / buttons to select a width, then confirm your selection with the  button.




The factory default setting page size is Letter for US, and A4 with a printable area width of 7.7" for EC.

2.5.4 Layout

(HP LaserJet III emulation)

The layout menu offers five different options which you can modify: Page orientation, Margin settings, VMI (the height of a line of text), End of line (text wrap), and Auto line feed. Use the / buttons to scroll through the list of available options, and the  button to select the one you wish to modify.

The available orientation options are Portrait (the factory default) and Landscape. Use the / buttons to display the one you wish to select, and the  button to confirm your selection.

The Margin menu allows you to select between "Default margin" and "Set margin". Select an option by displaying it with the / buttons, and pressing . The default margins are determined by the current font selection, orientation and emulation settings.

If you select "Set margin", a further menu appears, detailing the margin parameters that can be set: left margin, right margin, top margin and text length. Use the / buttons to step through the parameters to the one you want to set, and press the button. The parameter's current value is displayed. Step through the available values using the / buttons and make your setting using the button. The range of values available depends on the current orientation, VMI and character pitch.

The available End of line options are "Auto-wrap off" (the factory default) and "Auto-wrap on". Use the / buttons to display your choice, and the button to select it.

The VMI menu allows either "Auto selection" or "Manual selection". Select either by displaying it with the / buttons, and pressing . If you selected manual selection, the current VMI setting (in 1/48") is displayed. Use the / buttons to step through the available settings until you reach the value you want, then press the button to set the VMI to your chosen value. The factory default setting is a manual VMI of 8; equivalent to 6 lines of text per inch.

If Auto VMI is selected, the Text length in the Set margin menu setting determines the VMI. However, the current manual VMI value is retained for future use, in case manual VMI is reselected.

The Auto line feed options are "CR=CR" (the factory default) and "CR=CR+LF". Use the / buttons to select one, and the button to confirm your choice.

2.5.5 Paper feed

This menu allows you to select the paper feed source, and to designate the size of paper fed from the front tray. First use the / buttons to display either "FEEDER" and "FRONT TRAY SIZE", and the button to select the option you wish to set. The available feeder options are:

Cassette only
Auto Selection
Cassette
Lower cassette
Front tray
Manual

The available front tray sizes are:

Letter	Monarch
Legal	COM-10
A4	International DL
B5	International C5
Executive	

In each case use the / buttons to scroll through the list of available options, and the button to make your choice.

If the optional lower cassette unit is installed, selecting "Cassette only" as the feeder option displays a further sub-menu, with the options "Substitute" and "Normal". If "Substitute" is selected, subsequent **<ESC>H** and **<ESC>H** commands from a host computer will select the lower cassette; if "Normal" is selected, **<ESC>H** and **<ESC>H** will select the standard cassette. Use the / buttons to display the option you wish to select, and the button to confirm your choice.

The factory default feeder setting is Cassette only. The factory default front tray size is A4, however, loading the factory settings will not alter the current tray size setting.

If the selected feeder option is “Cassette only”, the Front tray size menu is not shown and the host command <ESC>&/1H cannot be used to select the Front tray.




If “Auto selection” is the feeder option, host commands specifying a paper size will cause commands that designate a paper source to be disregarded. The paper feed source can also be selected using the Feeder select control panel button.

Only physically available options are shown on the LCD display menu.

2.5.6 Emulation

On the LS-5EX the available options are HP LaserJet III mode and Hex dump mode (for debugging). If a TrueImage board has been installed, TrueImage will also be available.




On the LS-5TT the available options are HP LaserJet III mode, TrueImage mode and Hex dump mode.

Use the /  buttons to scroll through the list of available options, and the  button to select the emulation you require. Performing a factory reset does not change the selected emulation. The newly selected emulation is marked by an @ symbol.

The factory default emulation is HP LaserJet III on the LS-5EX, and TrueImage on the LS-5TT.




2.5.7 Printer commands

(HP LaserJet III emulation)

The available options are PCL+GL2 (the factory default) and GL2 only. Selecting GL2 only causes PCL commands to be disregarded. Use the /  buttons to display the options in turn, and the  button to select the option you require.

2.5.8 REP

(HP LaserJet III emulation and TrueImage)

REP (Resolution Enhancement Procedure) enables 300×600 dots per inch printing, and can be turned off or on. Use the /  buttons to display the options in turn, and the  button to select the option you require.

With the standard memory configuration (1MB on the LS-5EX, 2MB on the LS-5TT), REP is unavailable if Page mode has already been set to Full page.

If extra RAM has been installed, Full page and REP may both be selected.

2.5.9 Page mode

(HP LaserJet III emulation and TrueImage)

The available options are “Partial page” and “Full page”. “Full page” mode offers a further choice between Letter or A4 page size (the factory default) and Legal page size. Use the / buttons to display the page mode you require, and then press the button. If you have selected “Full page” mode, use / to choose a page size and confirm your selection with the button.

With the standard memory configuration (1MB on the LS-5EX, 2MB on the LS-5TT), Page mode will be unavailable if REP has already been selected.

If extra RAM has been installed, Full page and REP may both be selected.

Subsequently selecting Hex Dump mode will not alter the page mode setting.

2.5.10 Input buffer

This setting determines the size of the buffer used to store in-coming data. The buffer can be set to 1k (the factory default) or 128k. Use the / buttons to display the options in turn, and the button to select the option you require.

2.5.11 Interface

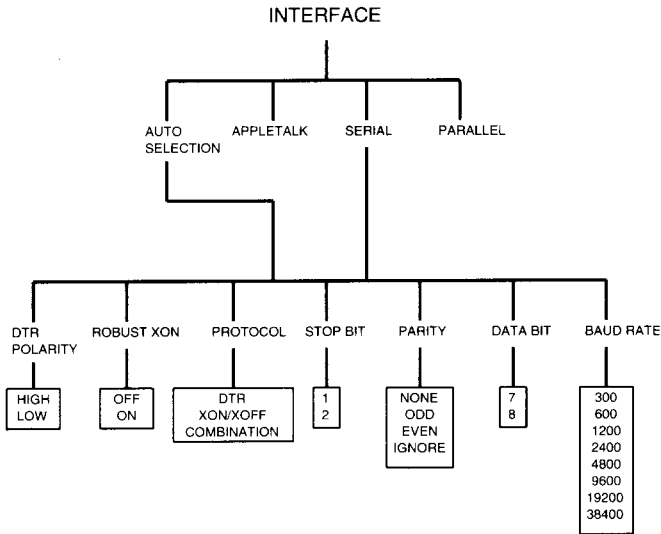
This option allows you to set up the interface between the host computer and the printer.

The LS-5EX and LS-5TT are both equipped with a Centronics parallel interface, an RS-232 serial interface and an AppleTalk interface. However, the LS-5EX's AppleTalk port is only enabled if a TrueImage board has been installed. The parallel and serial interfaces may both be connected at the same time, however, only one can be active at once.

The interface menu offers four options: Parallel, Serial, AppleTalk (LS-5TT or LS-5EX with TrueImage board only) and Auto. Use the / buttons to scroll through the list of available options, and the button to select one.

If you select Serial or Auto, a further menu appears, listing the serial data transfer parameters that must be set: baud rate, data bit, parity, stop bit, protocol, robust-XON and DTR polarity. Use the / buttons to step through the parameters to the one you want to set, and press the button.

The parameter's current setting is shown. Step through the available settings using the / buttons and select a value using the button. The range of available settings is shown below.



Parallel is the factory default interface setting.

2.5.12 *Print density*

This setting controls the relative lightness or darkness of printed output. The following print density settings are available:

Dark
Semi dark
Medium
Semi light
Light




Use the / buttons to display the options in turn, and the button to select the setting you require.

The factory default setting is medium.





2.5.13 Auto paper feed

This feature allows you to set an interval at which paper will automatically be fed into the printer. The available settings are:




Off
30 seconds
60 seconds
180 seconds

Use the / buttons to display the setting you require, and the  button to confirm your selection. The factory default setting is “Off”.

2.5.14 Set user default

This menu allows you to assign the present initial settings to one of the two user default sets: mode 1 or mode 2. Initial settings are indicated by an @ next to the value on the LCD display. On power-up or after a hard reset, the printer always adopts the parameter values of mode 1. The user default settings of mode 1 or mode 2 can be made the current settings at any time, using the  button. Use the / buttons to display “Mode 1” or “Mode 2” (whichever you wish to set) and the  button to assign the initial settings to the chosen user default mode.

2.5.15 Load factory set

This option allows you to restore all parameter settings to their factory default values. Either the EC or US factory default set may be selected. The current and initial settings are set to those of the selected factory default set. Use the / buttons to display “US Set” or “EC Set” (whichever you wish to use) and the  button to restore the factory settings. The current emulation setting is not affected.

2.6 Feeder select

The **FEEDER SELECT** button provides an alternative method of selecting the paper source and Front tray paper size. The options available are as described in the program mode section under the Paper feed parameter.

To select the paper source using the **FEEDER SELECT** button, first make sure that the printer is off-line, press the **FEEDER SELECT** button repeatedly until the chosen paper source is selected, and then press the **ONLINE** button to confirm your selection and put the printer back on-line.

To select the front tray paper size hold down the **FEEDER SELECT** button for 2 seconds until the display shows the current front tray paper size. To select a new setting, press the **FEEDER SELECT** button repeatedly until the chosen paper size is selected, and then press the **ONLINE**, **TEST**, **RESET** or **PROGRAM** button to confirm your selection.


2.7 Mode

You can select the mode 1 or mode 2 user default settings as follows. Make sure that the printer is off-line, then press the **MODE** button. The three available options are: No change, Mode 1 and Mode 2. Press **MODE** until the option you require is displayed, then press the **ONLINE**, **TEST** or **RESET** button. If you select No change, no action is taken. If you select Mode 1 or Mode 2, the settings stored as the mode 1 or 2 user default set are copied to the initial and current settings.

2.8 Selecting display language

To select the language to be used on the LCD display, switch on the printer while holding down the **RESET** button. Keep the **RESET** button held down until the message “Select language” appears on the LCD display. Use the **ERROR SKIP** and **TEST** buttons to step through the selections to the language of your choice, and the **RESET** button to confirm your selection. Finally press the **ONLINE** button to save the new setting and put the printer back on-line. If you press **ONLINE** without having pressed **RESET**, the original display language is retained. The languages available are: English, French, German, Italian and Spanish. The factory default setting is English.

2.9 Errors and status messages

Error messages and status messages are displayed on the LCD screen. Some error conditions can be cleared by pressing the  button. In other cases, some form of intervention will be necessary, e.g. a paper jam will have to be cleared by hand. A full list of errors and alarms is given in the Operations Manual.

2.10 PCL

The PCL and GL2 languages control the printer when it is in HP LaserJet III emulation mode. These languages provide a wide range of commands, including commands that can set most of the parameters available on the control panel. The printer's current settings are generally a combination of its initial settings and settings made by software. The Printer commands parameter in program mode allows you to limit the printer to interpreting GL2 commands only. The PCL and GL2 command languages are described in chapters 4 and 5 of this manual.

2.11 TrueImage

TrueImage is a page description language based on, and compatible with, Adobe System Inc's PostScript. The printer's TrueImage interpreter is capable of generating output from both TrueImage and PostScript page description programs.

The TrueImage language consists of a comprehensive range of operators that can describe the appearance of text and graphic material on the printed page. The language also contains operators that can make the most of the parameter settings available from the control panel. Star have added a number of extensions to TrueImage to enable paper size selection and tray selection. Hence programmers can enable applications to manage TrueImage output with a high degree of flexibility. TrueImage is described in Chapter 6 of this manual.

2.12 Hex dump mode

Hex dump mode is a special printer mode in which all data received by the printer is simply printed as a sequence of hexadecimal numbers. The printer does not attempt to interpret the in-coming data as emulation language commands or as graphics or character data. For debugging purposes, Hex dump mode can be a useful option, as it enables the user to examine the raw data generated by an application program or page description program. Hex dump mode is available as an emulation setting within program mode.

2.13 Superset commands

Four escape sequence commands, `<ESC>[Cn` (Select feeder), `<ESC>[En` (Change emulation mode), `<ESC>[On` (Select orientation), and `<ESC>[Sn` (Select paper size), are recognised in any emulation, HP LaserJet III, TrueImage and Hex dump. This allows any of these four functions to be performed by software at any time (provided that the printer is on-line).

2.14 Fonts

Fonts are described in detail in chapter 3. Fonts are available from several sources: resident fonts that are already installed in the printer, cartridge fonts that plug into the printer's cartridge slot (for HP LaserJet III mode only), and downloadable (soft) fonts. Soft fonts are sent to the printer from a host computer. The transfer process is known as downloading. This can be performed using a downloader application (such as Font Downloader on the Macintosh). Nowadays, however, many applications programs that use text (such as DTP packages) automatically download fonts as necessary.

Soft fonts are normally purchased, either on floppy disk or on CD ROM, transferred to the host computer's hard disk, and then downloaded to the printer. However, soft fonts may also be created by the user on a computer, either using a commercial application designed for that purpose, or by using the appropriate features in PCL or TrueImage. This latter method, however, is liable to be time-consuming and may not yield satisfactory results. In some cases, however, it may be appropriate, for example, if a small number of otherwise unavailable special symbols is needed.

Fonts

3.1 Introduction

Most printing work involves the production of text. The most basic unit of text is the single character. To facilitate text-handling characters are grouped into fonts, in which all characters have a consistent appearance. A knowledge of fonts and the basic principles of typography is a useful asset, and will also help you understand the way in which the printer handles text.

3.1.1 *Definition of a font*

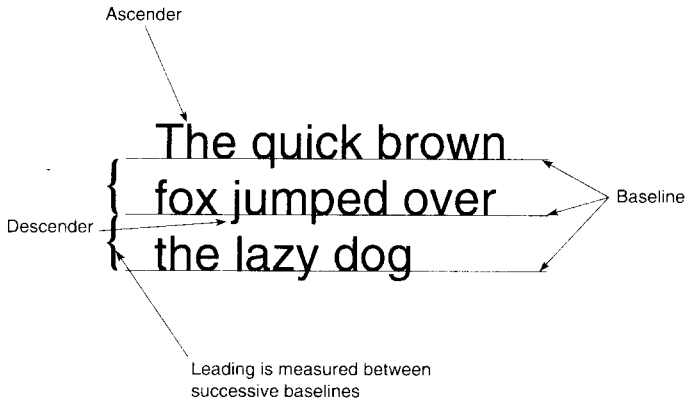
A font is a collection of characters of a particular design and size. The design is known as the typeface. There are thousands of different typefaces in existence. Commonly-used typefaces include Times, Palatino, Helvetica, Univers and Courier. A font may also consist of further modifications to the basic typeface design, for example the characters may be bold or italic. Typefaces are usually the product of meticulous and pains-taking effort by a typographic artist who has designed the shape of each character so that the overall effect of text in the font is pleasing to the eye and easy to read.

3.1.2 *Typeface families*

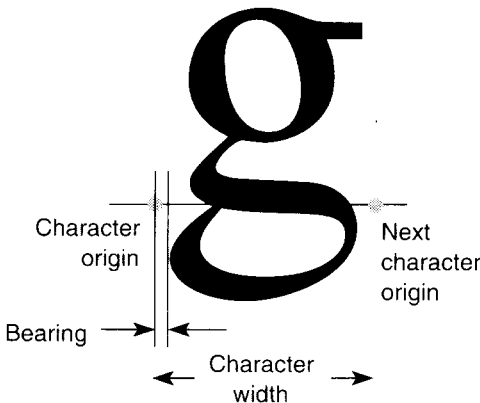
Fonts are often used or distributed as “families”, groups of fonts that are variations on a single typeface design and that combine together well. For example, Times, Times Bold, Times Italic and Times Bold Italic may comprise a family, or alternatively Garamond, Garamond Light, Garamond Italic and Garamond Light Italic. There are numerous ways in which a basic typeface design can be modified; these are described in the “Font attributes” section which follows.

3.1.3 Character features

There are several features of character shape and spacing that affect text placement and appearance. The characters that make up a line of text sit on an imaginary line known as the baseline. Most characters, such as ‘M’, ‘F’ and ‘r’, sit squarely on the baseline. Some characters, such as ‘y’ and ‘g’ extend below the baseline, while others, such as ‘l’ and ‘k’, extend above most other characters, and up close to the baseline of the text line above. The part of a character that goes below the baseline is known as a descender; the part that extends upwards is known as an ascender. Leading is the vertical distance between successive lines of text and is measured from baseline to baseline.



As well the character’s shape, the design of a character in a font includes information that describes how it will be positioned relative to adjacent characters.



The origin of a character is a reference point that defines how the character is positioned relative to the text baseline, and to the preceding character.

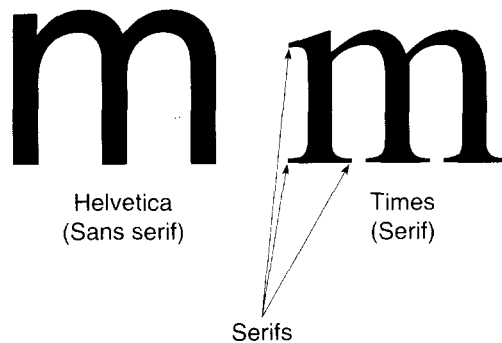
A character's width defines the distance between its origin and the position of the origin of the character which follows. This distance is greater than the actual width of the character's shape.

A character's bearing is the horizontal distance between the left-most part of the character and its origin.

The width and bearing are designed so that adjacent characters are spaced attractively.

3.1.4 Character shape and readability

Some typefaces, for example Times and Palatino, have small curly hooks on the ends of the lines that form the characters. These hooks are known as *serifs* and make body text more readable by leading the eye on from one letter to the next. Other typefaces, for example Univers and Helvetica, do not have these hooks and are referred to as *sans serif* (without *serif*) typefaces. Characters in these typefaces stand out on their own better. Generally fonts with *serif* typefaces are used for body text and *sans serif* typefaces are used for headings and captions.



3.1.5 Document design

Advances in personal computer and laser printer technology have brought high-quality document production within easy reach of anyone with access to a PC and a laser printer. Document design is largely a matter of personal preference, however, there are a few basic guidelines that should be followed.

Do not be tempted to use too many different fonts in a single document, and in particular, on a single page.

In general, use smaller-sized *serif* fonts for body text and larger *sans serif* fonts for headings, captions, titles and any text that is to stand out prominently.

Choose typefaces that work well together. This, too, is to some extent a matter of taste, and experience and experimentation will help you develop good judgement in this matter.

If your software allows, enforce consistency in a document by using paragraph styles. Most desktop publishing packages and word-processors now support this feature. Using this method, you define a number of fixed font formats and assign names to them, e.g. you might define a paragraph style "BodyText" to be 10 point Times Roman. Every time you set a piece of text to be BodyText, it is automatically formatted as 10 point Times Roman.

If your software allows, define document master pages. These are page templates onto which you can place the text and graphic elements of your document. This method also helps to lend your documents a consistent appearance.

3.2 Font attributes

The word “font” is used in a variety of contexts, and is open to a number of different interpretations. For our purposes here, however, a font is a collection of characters with a specific set of attributes. When emulation software (PCL, GL2 or TrueImage) selects a font for printing, the font is usually identified as a specified collection of some or all of these attributes, e.g. 12 point Univers Bold Italic. Font attributes are as follows.

3.2.1 Typeface

The typeface of a font is the design style of the characters. Typeface lends a font its distinctive appearance. The printer has fonts in several typefaces permanently resident in its ROM (read-only memory). Some examples of different typefaces are shown below.

Courier

Palatino

Bookman

Univers

Helvetica

Freestyle script

3.2.2 Spacing type

Fonts are either monospaced (fixed) or proportionally-spaced. The spacing type of a font is inherent in the typeface. Of the typefaces listed above, Courier fonts are monospaced, and the rest are proportionally-spaced.

The characters of a monospaced font all have equal width, and so occupy an equal amount of space on a line. The characters of a proportionally-spaced font take up varying amounts of space depending on each individual character’s design. As a result of this, two different sentences that contain the same number of characters will occupy the same width if printed using a monospaced font, but will usually have different widths if a proportionally-spaced font is used.

An example sentence in Courier.

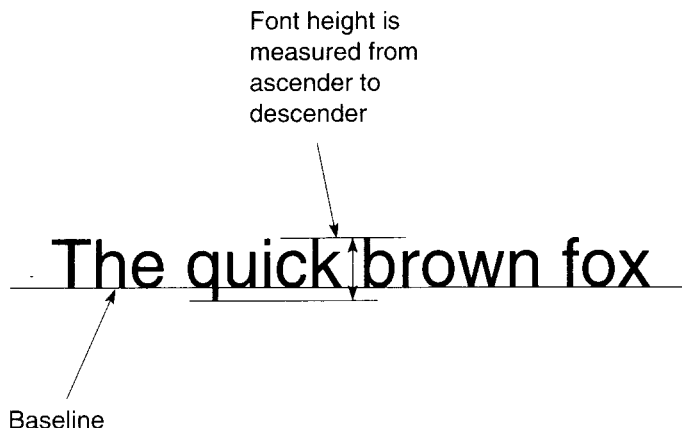
Identical number of characters.

An example sentence in Bookman.

Identical number of characters.

3.2.3 Height

The height or point size of a font is the maximum vertical distance that a single line of text might occupy on the page. Height is measured in typographic points (1/72") and is the vertical distance between the top of the font's highest ascending character, (for example the lowercase l) and the bottom of its lowest descender (for example the lowercase 'p'). The highest ascender and lowest descender depend on the typeface design.



3.2.4 Pitch

The pitch of a monospaced font is the number of characters printed per inch. Proportionally-spaced fonts do not have a pitch attribute, since different characters have different widths.

3.2.5 Weight

Font weight is the thickness of the lines which make up the font's characters. The standard weight is known as medium. Bold fonts, with thicker lines, are also commonly used. Bold text is often used for emphasis or for headings. Light stroke weight fonts have lines that are narrower than the standard weight. Some examples of different stroke weights are shown below.

Univers Light
Univers Medium
Univers Bold
Univers Black

3.2.6 Posture

A font's posture attribute refers to whether it is upright or italic (oblique). Italic text is often used to make particular words or text stand out from the surrounding body text.

Upright text
Italic text

3.2.7 Width

Some fonts are designed as variations on a basic typeface design, but with the character width reduced or enlarged. These types of fonts are generally referred to as condensed (or compressed) and extended.

Ordinary Univers Bold
Condensed Univers Bold

3.2.8 Symbol set

The symbol shapes that a font can display may be varied to meet different printing needs. Most fonts normally use a standardised set comprising upper- and lowercase letters, numerals and punctuation symbols, plus a few extra symbols. However, Symbol or Dingbat typeface fonts use completely different sets of symbols, including bullets, geometric shapes, arrow characters and Greek letters.

Roman-8

ABCDEFGHIJKLMN O PQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789-=[,;'./!@#\$%^&*()_+{|}:<>?-' '... ""

Symbol

ΑΒΧΔΕΦΓΗΙΘΚΛΜΝΟΠΘΡΣΤΥςΩΞΨΖαβγδεφγηιφκλμνοπθρσ
τυωξψζ0123456789-=[;Π,./!≡#%±&*()_+{|}:©<>?∠
≠™Π"”э▷÷ⓂⓈ

Dingbats

⌘ ⦿
⦿
✂ ✂

3.3 Printer fonts

3.3.1 PCL

In HP LaserJet III mode fonts are either bitmap or scalable.

Bitmap fonts

Each character in a bitmap font is defined as a matrix of dots that the printer prints on the page. As a result, bitmap fonts are available in particular point sizes only. For example, the printer contains resident Courier 10 point and 12 point fonts. You can print using Courier 10 or 12 point at any time, simply by selecting the appropriate font. However, if you want to print using Courier 16 point, Courier 16 point must first be made available to the printer either on a cartridge or as a soft font downloaded from a host computer.

Scalable fonts

Each character in a scalable font is defined as an outline shape. The printer converts the outline shape to a matrix of dots that forms the printed character. Hence, scalable fonts are available in any size. For example, the printer contains a resident Univers scalable font. To print using a Univers font of any size, simply select the Univers typeface and specify the size you require. The printer automatically scales the characters to the selected size. Scalable fonts can be scaled to any size from 3 points to 999.75 points, in increments of 0.25 points.

Resident printer fonts

The fonts that are available in HP LaserJet III emulation are listed below. There are 8 scalable typefaces and 14 bitmap fonts; each of the bitmap fonts listed is available in both portrait and landscape versions. Although the printer can automatically rotate any font to fit the current page orientation, the rotated font will take up printer memory space. Thus resident fonts in different orientations can help save memory.

Samples of each font are included for reference in the Technical Supplement at the end of this manual.

Scalable	Bitmap
Univers Medium	Courier 10-point (12 characters per inch)
Univers Medium Italic	Courier Bold 10-point (12 characters per inch)
Univers Bold	Courier Italic 10-point (12 characters per inch)
Univers Bold Italic	Courier 12-point (10 characters per inch)
CG Times	Courier Bold 12-point (10 characters per inch)
CG Times Italic	Courier Italic 12-point (10 characters per inch)
CG Times Bold	Line Printer 8.5-point (16.6 characters per inch)
CG Times Bold Italic	

3.3.2 GL2

In GL2 graphics language mode any font available in PCL mode may be selected. Additionally, GL2 has its own stick font, comprised purely of lines. This font is suitable for use in technical drawings, and is described in chapter 5.

3.3.3 TrueImage

TrueImage has its own native font format, known as TrueType. TrueType is also an integral part of the Macintosh System 7 and MicroSoft Windows 3.1 operating systems. TrueImage can also use PostScript fonts. TrueType fonts define their characters as a set of outline shapes that can be scaled to any size. A Macintosh running system 7 or a PC running Windows 3.1 will have access to TrueType fonts that are part of its system software. The computer can automatically scale these fonts and send them to any printer. This laser printer contains 35 resident TrueType fonts.

TrueType

The printer contains the following TrueType fonts. These fonts can be used at any time. Output that uses any of these fonts will be produced more quickly, since the host computer is spared the task of sending the font to the printer. Samples of each font are included for reference in the Technical Supplement at the end of this manual.

Arial	Courier	ITC Zapf Chancery Medium Italic
Arial Bold	Courier Bold	ITC Zapf Dingbats
Arial Oblique	Courier Oblique	Symbol
Arial Bold Oblique	Courier Bold Oblique	Times New Roman
Arial Narrow	ITC Avant Garde Gothic Book	Times New Roman Bold
Arial Narrow Bold	ITC Avant Garde Gothic Demi	Times New Roman Italic
Arial Narrow Oblique	ITC Avant Garde Gothic Book Oblique	Times New Roman Bold Italic
Arial Narrow Bold Oblique	ITC Avant Garde Gothic Demi Oblique	Zapf Calligraphic Roman
Century School- book Roman	ITC Bookman Light	Zapf Calligraphic Bold
Century School- book Bold	ITC Bookman Light Italic	Zapf Calligraphic Italic
Century School- book Bold Italic	ITC Bookman Demi	Zapf Calligraphic Bold Italic
Century School- book Italic	ITC Bookman Demi Italic	

PostScript

PostScript Type 1 or Type 3 (user-defined fonts) can be used in TrueImage mode. PostScript fonts are also defined as character outlines. The TrueImage interpreter included with the printer scales the outline to produce characters of the required size.

3.4 Font sources

The printer can use fonts from three different sources: its own internal fonts, as listed in the preceding section, fonts that have been downloaded from a host computer, and fonts on a cartridge plugged into the printer's cartridge slot. (Cartridge fonts are only available in HP LaserJet III emulation mode).

3.4.1 Resident fonts

These are listed in the preceding section. These fonts are permanently resident in the printer's ROM (Read-only memory) and thus are always available for selection. Documents that use the internal resident fonts will generally print faster than documents that require fonts to be downloaded.

3.4.2 Downloaded (soft) fonts

Fonts can be downloaded from the host computer to the printer. There is a vast number of fonts available for both HP LaserJet III and TrueImage modes. Fonts can be purchased on floppy disk and on CD-ROM. Copy them to your computer's hard disk and then download them to the printer. Downloaded fonts reside in the printer's RAM (Random Access Memory). The process of downloading will depend on the host computer and the software being used. Downloading is discussed in the section "How applications use fonts" on page 36 of this chapter.

Since the printer also uses its memory to compose pages prior to outputting them and also to store other necessary information, too many downloaded fonts may slow the printer down, or even prevent it from printing complex pages. It is good practice to regulate the number of soft fonts in the printer at any given time.

3.4.3 Cartridge

In HP LaserJet III modes fonts may be installed on cartridge. To make cartridge fonts available to the printer simply plug the cartridge into the cartridge slot. The fonts on the cartridge are then available for selection, just as if they were resident in the printer, or had been downloaded into printer memory. The advantage of cartridge fonts is that they do not consume any of the printer's resources. Also, a cartridge font may be selected as the mode 1 or mode 2 user default font. Provided that the cartridge remains in the printer it will be restored as the user-default font when a hard reset is performed.

3.5 Font selection

So far in this chapter we have described ways in which fonts become available for selection, but have not described exactly how they may be selected. As far as the average user is concerned, printer fonts will usually be selected automatically by the applications software running on the host computer. However, the explicit selection process depends on the printer mode as follows:

3.5.1 HP LaserJet III mode

A font may be selected as the current font using the printer's control panel. This is described in detail on page 12 of Chapter 2 of this manual. Text documents that contain no font selection information will be printed using this default font.

Software applications select printer fonts using PCL escape sequence commands, either specifying font attributes or a unique font ID number. This process is described in detail on page 77 of Chapter 4 of this manual.

3.5.2 TrueImage mode

TrueImage fonts cannot be selected from the control panel. Applications that enable TrueImage output and TrueImage page description programs select TrueType and PostScript fonts using the TrueImage font operators. These are described in the section Font operators starting on page 260 of Chapter 6.

3.6 Special symbols and characters

A typeface may comprise designs for many different characters. In addition to the standard upper- and lowercase letters, numerals and punctuation symbols, there are also currency signs, mathematical symbols, foreign-language accented characters, Greek letters and various others that may be needed from time to time. A printer font can usually represent a maximum of 256 different characters at any one time, as defined by its associated symbol set. Symbol sets are normally designed for a specific purpose, for example to print text in a particular language.

When the need arises, it is easy to switch symbol sets in order to gain access to new characters. In HP LaserJet III mode this may be done in software using escape sequence commands. For example, if you are printing a document that is mostly in French but contains quotations in German, you would first select the ISO69:French symbol set with the appropriate command, switch to the HP German symbol set at the appropriate points in the document, and then revert to the French set after printing each quotation. In this way, both French and German accented characters will appear properly. PCL symbol set selection commands are described on page 81 and 82 of Chapter 4.

PCL symbol set switching can also be performed from the control panel, as described in Chapter 2 on page 12.

The symbol sets available with the printers resident PCL fonts are given for reference in the Technical Supplement at the end of this manual.

The method in which different symbols can be selected in TrueImage mode is outlined in Chapter 6 on page 218.

3.7 How applications use fonts

The way in which applications and fonts interact depends on the hardware and software that is being used with the printer. The documentation accompanying applications software and operating system software should tell you what you have to do in order to use the fonts you require.

3.7.1 Automatic downloading

Nowadays many applications designed for handling text, such as word-processors and desk-top publishing programs, will automatically manage font downloading.

For example, suppose you are composing a document using PageMaker 4.0 running under System 7 on a Macintosh computer, driving the printer in TrueImage mode. If you format a paragraph of body text in New Century Schoolbook 12-point, and then print out the page containing the paragraph, the paragraph will be printed using the resident printer font.

If you format a different paragraph in Palatino 10-point, and print out the page containing the new paragraph, the Macintosh will check the printer's RAM and ROM to see if Palatino is available, and if it is not, will download its own TrueType or PostScript Palatino font. The TrueImage interpreter scales the new font to the correct size and the paragraph is then printed in Palatino.

3.7.2 Manual download

Older operating system or applications software may not perform automatic downloading. In these cases you will need to download soft fonts explicitly. On PC systems running the MS-DOS operating system this can be done by using the DOS COPY /B command to copy font files from the computer to the printer. Usually, however, font vendors supply a downloading utility on floppy disk with their fonts.

On older Macintosh systems, the Font Downloader utility may be used to download fonts to the printer in TrueImage mode.

3.8 Conclusion

Font technology is in a continual state of flux. Virtually every month new products are released and new technological advances are announced. Commercial applications are now available that allow you to convert existing fonts from one format or platform to another. For example, you could create Macintosh TrueType fonts using your existing PC PostScript fonts. Soon we may see font formats that allow the user to derive unlimited numbers of variations from a small set of typeface designs.

For this reason it is well worth regularly reading the industry literature, and monitoring the relevant topics on on-line bulletin boards, in order to keep abreast of new developments.

MEMO

Printer Control Language

CHAPTER 4

4:1 Introduction

The Star LS-5EX and LS-5TT printers emulate the Hewlett Packard LaserJet III. In HP LaserJet III emulation mode, the printer is driven by a control language known as Printer Control Language (PCL), a language that has achieved wide acceptance as a *de facto* printer-control standard. This language has evolved over several years and is now in its fifth major revision, PCL5.

PCL features a wide range of commands and an extensive list of capabilities.

- Job control
- Page set-up
- Precise cursor positioning
- Support for both scalable and bitmap fonts
- Raster and vector graphics
- Macros

There are twenty-two resident PCL fonts already in the printer and you can take advantage of many more by installing font cartridges that plug into the printer's cartridge slots, or by downloading fonts from your computer. There are now thousands of commercially-available fonts, on cartridge, on floppy disk and more recently on CD-ROM.

You can also construct fonts to your own design and download them using PCL commands.

Powerful vector graphics capabilities are available in the shape of the GL2 graphics language, which can be directly accessed from PCL with a single command. PCL and GL2 in conjunction allow you to combine high-quality text and precision graphics in your output. GL2 is described in Chapter 5.

4.2 Printer control language commands

PCL commands are usually sent to the printer together with text and graphic data that is to be printed. Their function is to enable the printer to interpret and format the accompanying data correctly. Whenever the printer receives a command, it executes it. This may simply entail performing a single operation, for example drawing a rectangle on the page, or may determine the way subsequent operations are carried out, for example causing subsequent text to be printed in Times bold.

There are two types of PCL command: control codes and escape sequences.

A control code is a single ASCII code that instructs the printer to perform some simple operation, for example, <CR> (ASCII code 13), causes the printer to perform a carriage return operation. Other common control codes are <LF> (Line feed) and <FF> (Form feed). Control codes are normally described by a two- or three-letter upper case abbreviation, for example <CR>.

An escape sequence is a sequence of characters starting with the <ESC> character. The <ESC> character is a control code (ASCII code 27). The characters following the <ESC> character define the command, for example <ESC>(s3B makes the primary font bold.

Most escape sequence commands include parameters. A typical command is <ESC>(s16V which tells the printer to set the primary font type size to 16 point.

Some escape sequence commands are followed by a stream of data bytes describing, for example, a character or a graphic image.

4.2.1 Applications software

Commercial applications software drives the printer by converting its own commands to the equivalent PCL commands. Hence if you are using a word-processor and want a particular word within a paragraph to appear in italic, you would first select or highlight the word and then select the word-processor's own built-in "italic" command. The word-processor will automatically send the correct commands to the printer at print time.

Some older word-processors require you to enter the PCL escape sequences manually into the document you are working on. In this case you will enter the escape sequence from the keyboard. This involves first positioning the

on-screen cursor and then pressing a combination of keys. The ALT and CTRL keys usually designate an <ESC> sequence. Consult the particular application's manual for specific details.

4.2.2 Programming

If you are writing software to drive the printer, you will need to address the printer directly. This is quite straightforward: PCL commands can be sent to the printer using the same programming language commands that are used to print ordinary text. In BASIC this is the LPRINT command, in C the fprintf function.

A list of guidelines for coding follows:

- Send control codes to the printer as ASCII codes, for example the command LPRINT CHR\$(13); will send a carriage return to the printer.
- Send escape sequences by sending the <ESC> code followed by a text string made up of the letters and numbers which follow. For example, the command LPRINT CHR\$(27);"(s3B"; will transmit the command to set the primary font to bold.
- Encode graphic images and font character definitions as a stream of ASCII codes and send them to the printer using the LPRINT command or equivalent.
- Combine escape sequences into one single sequence when possible, as this makes for more compact code. However, ensure that you arrange commands in the exact order in which you want them executed: combined escape sequences are always executed from left to right.
- Compress character definitions and raster graphic data where possible.
- For complex graphic output switch to GL2 mode from within PCL using the <ESC>%nB command. Perform graphic operations using GL2 commands before reverting to PCL with the <ESC>%nA command.

4.3 Command format

A control code command is a single ASCII code.

An escape sequence command consists of the `<ESC>` character followed by one or more characters which identify the command. Most escape sequence commands require parameter values. These are represented in the sequence by the appropriate numeric characters: that is, in the escape sequence `<ESC>(s16V` the point-size parameter value 16 is represented by the characters '1' and '6'. All the letters in an escape sequence must be lower case except the final letter which must be uppercase.

There are six escape sequences that consist simply of `<ESC>` followed by a single character: `<ESC> E`, `<ESC> 9`, `<ESC> =`, `<ESC> Y`, `<ESC> Z` and `<ESC> z`.

All others consist of `<ESC>` followed by several characters. The standard form is as follows: the first character after `<ESC>` is either `&`, `(`, `)` or `*`, the second character is a lower case letter, the next one or more characters are digits making up a number in the range `-32768` to `32767`, and the final character is an upper case letter (`A - Z`).

If you omit the number parameter, the printer reads its value as 0.

The first, second and final characters of the escape sequence identify its function and the parameter number specifies a setting or value.

There are four commands which have additional data bytes following the final upper case character, for example the `<ESC>(snW` command, which is used to define a downloadable font character.

A few escape sequence commands vary slightly from this form.

Two or more escape sequences can be combined into one if the first two characters of each sequence are the same. For example, `<ESC>(s14V` (which selects a height of 14 points for the primary font) and `<ESC>(s1S` (which sets the primary font style to italic) may be combined and sent to the printer as `<ESC>(s14v1S`. Only the final character of the combined sequence is upper case. The 'V' at the end of the first command is made lower case in the combined sequence.

4.3.1 Syntax

In this chapter commands are printed in the text as follows:

Control codes are represented by a two- or three-letter mnemonic in bold type, e.g. <LF>.

Escape sequence commands are shown as follows:

The letters <ESC> in bold represent the escape character.

Letter and number characters in bold type are literals: they appear in the escape sequence exactly as shown.

n in italics stands for a numeric parameter value.

Words in italics in angle brackets represent a stream of data bytes.

For clarity, lower case L is shown as ‘ℓ’.

4.4 The buffer

When the printer receives data from the computer, it uses the data to build up an image of a complete page in its memory.

When it has received a complete page's data it images the data onto paper and ejects the hard copy page.

The speed of this process is limited by the time the printer takes to process the in-coming data, and the rate at which it can physically transfer the image to paper.

All printable data and commands are stored (buffered) in the printer's memory until the command to print and eject the page (<FF> or <ESC> **E**) is received. Data that has been received by the printer but not yet transferred to paper is described as being in the printer buffer.

4.5 The imaginary cursor

A laser printer does not have a physical cursor: each page is imaged in one fell swoop as the drum rotates over the paper. However, when describing the way in which PCL commands drive the printing process, it is helpful to adopt the concept of a cursor.

The current cursor position is the position on the page from which printing of the next character or graphic object will commence.

The cursor position changes as text and graphics are printed, when explicit cursor repositioning commands are used, and when a new page is begun.

4.6 The page

The sheet of paper on which the printer prints is called the physical page. The printer supports eight different sizes of physical page.

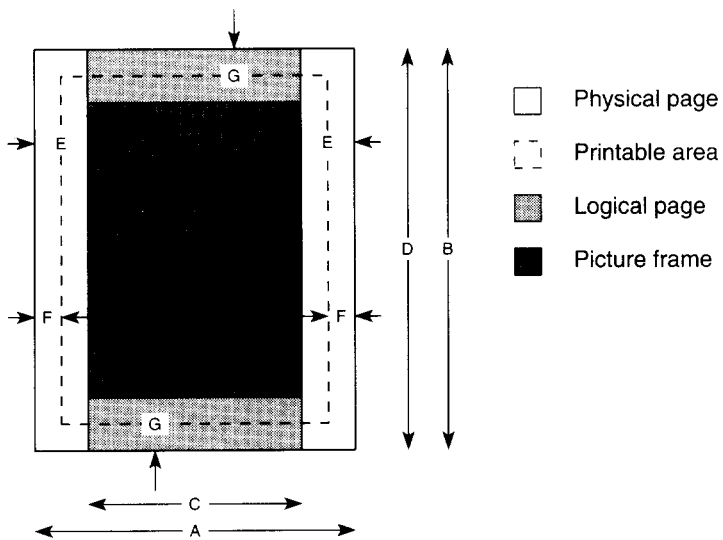
The area of the physical page on which the cursor can be positioned is known as the logical page. The size of the logical page depends on the physical page size. You can use PCL page definition commands to reposition and rotate the logical page.

The printable area is the area of the physical page on which the printer can place a dot. This is not the same as the logical page: the printable area is determined purely by the physical limits of the printer, whereas the logical page location can be altered by the user.

The text area is the area of the page bounded by the margins. Margins can be set using PCL commands or the control panel. The text area must lie wholly within the logical page.

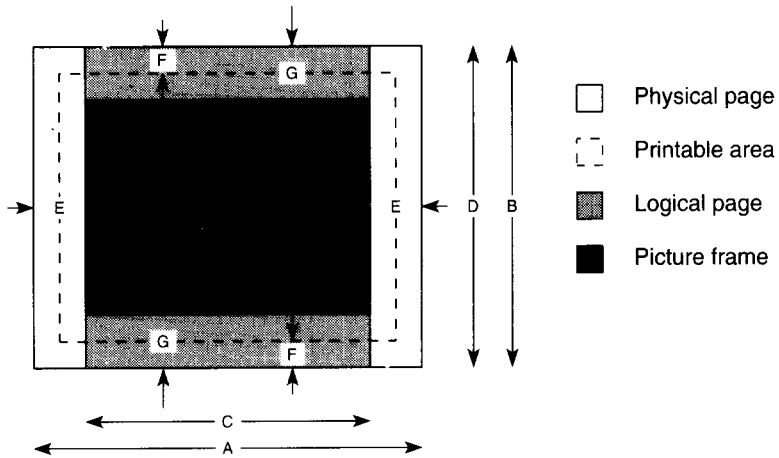
The picture frame is the rectangular area of the page in which GL2 graphic images can be displayed. You can set the size and position of the picture frame using PCL commands. The default picture frame and default text area are the same.

The diagrams which follow show the physical page, printable area and default logical page and picture frames for portrait and landscape pages. The table lists their dimensions for the different physical page sizes available.



	A	B	C	D	E	F	G
Letter	2550	3300	2400	3300	75	50	150
Legal	2550	4200	2400	4200	75	50	150
Executive	2175	3150	2025	3150	75	50	150
A4	2480	3507	2338	3507	71	50	150
Com-10	1237	2850	1087	2850	75	50	150
Monarch	1162	2250	1012	2250	75	50	150
C5	1913	2704	1771	2704	71	50	150
DL	1299	2598	1157	2598	71	50	150

All dimensions are in 1/300".



	A	B	C	D	E	F	G
Letter	3300	2550	3180	2550	60	50	150
Legal	4200	2550	4080	2550	60	50	150
Executive	3150	2175	3030	2175	60	50	150
A4	3507	2480	3389	2480	59	50	150
Com-10	2850	1237	2730	1237	60	50	150
Monarch	2250	1162	2130	1162	60	50	150
C5	2704	1913	2586	1913	59	50	150
DL	2598	1299	2480	1299	59	50	150

All dimensions are in 1/300".

4.7 The PCL coordinate system

The PCL coordinate system has its origin in the top left-hand corner of the current logical page. There are three types of unit: dots (1/300"), decipoints (1/720") and rows and columns. A row is the height of a text line as defined by the vertical motion index (VMI) setting. A column is equal to the width of a single space character and is defined by the horizontal motion index (HMI) setting.

Cursor positioning commands allow you to place the cursor anywhere on the logical page by reference to the coordinate system. Movement can be absolute with respect to the coordinate system origin or relative to the current cursor position.

The default cursor position is the position the cursor is set to on a new page. The default cursor position is at the left margin, 3/4 the height of a row below the top margin. This positions the cursor on the baseline of the first line of text on the new page.

The GL2 graphics language uses a different coordinate system and contains its own cursor positioning command. These are described in Chapter 5.

4.8 The printing environment

An environment is a combination of printer settings. The original settings pre-programmed into the printer that are current when you first use the printer in PCL mode, are known as the factory default environment.

There are four different types of environment to which we shall refer in this chapter.

4.8.1 Factory default environment

The factory default environment is made up of the settings programmed into the printer before it leaves the factory. You can revert to the factory default settings whenever you wish using the printer's control panel. See Chapter 2 for details.

The HP LaserJet III emulation factory default environment consists both of PCL parameter settings and GL2 graphics settings.

The table below shows the factory default PCL settings. The default GL2 settings are listed in the description of the GL2 IN command in Chapter 5.

Number of Copies	1	Stroke Weight	Medium
Registration	Left=0, Top=0	Typeface	Courier
Print Direction	0	Underlining Mode	Off
Orientation	Portrait	Font ID	0
Page Size	A4	Character Code	0
Paper Source	Paper Tray	Left Graphics Margin	0
Vertical Motion Index	8 (6 lpi)	Resolution	75 dpi
Horizontal Motion Index	12 (10 cpi)	Compression Mode	0
Top Margin	1/2" (150 dots)	Raster Height	Not Set
Text Length	64 lines	Raster Width	Logical Page Width
Left Margin	Left edge of logical page	Current Pattern	Solid
Right Margin	Right edge of logical page	Source Transparency Mode	0 (transparent)
Perforation Skip	On	Pattern Transparency Mode	0 (transparent)
Line Termination	CR=CR, LF=LF, FF=FF	Horizontal Rectangle Size	0
Symbol Set	Roman-8	Vertical Rectangle Size	0
Spacing	Fixed	Area Fill ID	0
Pitch	10 cpi	Macro ID	0
Height	12 point	End-of-Line Wrap	Off
Style	Upright	Display Functions	Off

4.8.2 User default environment

The user default environment is the combination of the factory default settings and any settings made by the user from the printer's control panel. Settings that can be made from the control panel include the number of copies, paper feed type and paper size. Most control panel-settable features can also be set using PCL commands.

You can restore the user default environment either by using the PCL command <ESC>E, or by performing a reset from the printer's control panel. When you perform a reset all settings made using PCL commands are lost.

See Chapter 2 for a description of a control panel reset.

The user default environment is retained when the printer is switched off.

The user default environment is made up of the factory default environment settings plus the following:

Number of copies	Symbol set
Font source	Paper tray
Font ID	Page size
Pitch	Feed type (manual or automatic)
Font height	Orientation
Typeface	Form length

4.8.3 Modified print environment

The modified print environment is made up of the current values of all settings that can be made with PCL commands (with a small number of exceptions). Whenever you change a setting with a PCL command, the modified print environment is updated accordingly.

Any parameters that have not been set by PCL commands retain their user default or factory default environment values.

If you use the Call macro or Enable macro for overlay command, the modified print environment is saved. After completion of the macro the modified print environment is restored again. Macros invoked using the Execute macro command can alter the modified print environment.

GL2 settings are not part of the modified print environment.

The settings which make up the modified print environment are listed below.

Page length	Macro ID
Page size	VMI/Line spacing
Orientation	Horizontal rectangle size
Left registration	Vertical rectangle size
Top registration	Area fill ID
Paper source	Raster graphics resolution
Number of copies	Raster graphics presentation mode
Margins	Raster graphics left margin
Perforation skip mode	Pattern ID
Line termination mode	Current pattern
End-of-line wrap	Source transparency mode
Primary font	Pattern transparency mode
Secondary font	Print direction
Current font (Primary or secondary)	Raster graphics compression mode
Primary font characteristics	Underline mode
Secondary font characteristics	Raster graphics height
HMI	Raster graphics width
Font ID	Character code

The following settings are not part of the modified print environment

Current cursor position	Cursor position stack
Downloaded fonts and macros	Picture frame size
GL2 plot size	Picture frame anchor point

4.8.4 Macro overlay environment

The macro overlay environment is a version of the modified print environment in which some user default environment settings override the current settings. The overlay environment becomes current when you invoke a macro using the Enable macro for overlay command. The overlay environment is described in the section Macros at the end of this chapter.

4.9 General printer control commands

This section describes the more general commands to control the printer, including job control, page set-up and cursor positioning commands, and commands for adjusting certain output characteristics.

4.9.1 Job control commands

The commands in this section prepare the printer for a print job. There are two commands for selecting the paper source: the Paper source command and the Select feeder command. Select feeder is a special command that will work in both LaserJet III and TrueImage emulation modes.

Reset - <ESC> E

A Reset restores the user default environment.

Any data still in the printer buffer is printed out.

It is a good idea always to start a print job with a Reset, so that all settings are in a known state.

Any temporary fonts and temporary macros are deleted from memory.

The command also has the following effects on the GL2 vector graphics state.

- All GL2 graphics settings are reset to their default values. Default values of these settings are listed in the description of the IN command in Chapter 5.
- The picture frame is reset to its default size and location.
- The GL2 horizontal and vertical plot sizes are reset to their default values, equal to the width and height of the PCL picture frame. GL2 plot size is explained in Chapter 5.

You can also perform a Reset from the control panel; see Chapter 2.

Select number of copies - <ESC>&nX

This command specifies the number of copies to be printed; the new setting takes immediate effect.

The current page and the following pages will be printed out the specified number of times.

n can be from 1 to 99.

The factory default number of copies is 1.

You can also select the number of copies from the control panel; see Chapter 2.

Select feeder - <ESC>[Cn

The command selects the paper feeder. On receipt of this command any data in the printer buffer is printed out and the new setting is applied to subsequent pages. The cursor is placed at the default cursor position on the new page.

Values for *n* are as follows.

1	Front paper tray
2	Front paper tray for one sheet, then cassette
4	Cassette
5	Cassette for one sheet, then front paper tray
7	Optional cassette
8	Front paper tray for one sheet, then optional cassette
9	Cassette for one sheet, then optional cassette

The factory default paper source is the cassette.

Paper source - <ESC>&lnH

The command selects the paper source. Any data in the printer buffer is printed out and the paper source is set as specified. The cursor is placed at the default cursor position on the new page.

Values for *n* are as follows

0	No change
1	Front paper tray
2	Manual feed
3	Manual envelope feed
4	Cassette
7	Optional cassette

The factory default paper source is the cassette.

Change emulation - **<ESC> [En**

The command switches emulation mode.

The command is effective in any mode.

$n = 0$ selects HP LaserJet III mode

$n = 5$ selects TrueImage mode

You must preface this command with the **<CR>** and **<FF>** control codes to eject the current page, otherwise the command simply ejects the current page and performs a Reset without altering the emulation.

4.9.2 Page definition commands

This section describes commands for selecting the paper size and setting the position and orientation of the logical page. You can set the page size in two ways: either by explicitly selecting a paper or envelope size, or by specifying the number of text lines to be printed per page, in which case the printer calculates the physical page size based on the current VMI value.

There are two commands for selecting the paper size: the Page size command and the Select paper size command. Select paper size is a special command that will work in both LaserJet III and TrueImage emulation modes.

Select paper size - **<ESC>[S n**

The command selects the paper or envelope size that the printer will use.

Values for *n* are as follows.

1	Letter
2	Legal
3	A4 international
4	Executive
5	B5 international
11	Monarch (envelope)
12	Com-10 (envelope)
13	International DL (envelope)
14	International C5 (envelope)

Any data in the printer buffer is printed out. The cursor is placed at the default cursor position on the new page.

If cassette selection has been set to "Automatic" from the control panel, and there is a tray inserted containing the selected size, paper is automatically fed from that tray.

Otherwise, if the paper size you select conflicts with the size of the paper in the selected paper feeder, a message appears on the control panel requesting you to change the paper tray.

You can override this request from the control panel. The printer will then proceed to use the paper in the currently selected feeder.

If n is set to a value other than those shown above, the command is ignored.

The factory default page size is A4.

The logical page size and position, the left, right and top margins, and the text length are set to the default values for the new page size.

The picture frame is set to its default size and position.

Any overlaid macro will be discarded.

The command also has the following effects on the GL2 vector graphics state.

- The GL2 horizontal and vertical plot sizes are reset to their default values, equal to the width and height of the PCL picture frame. GL2 plot size is explained in Chapter 5.
- The scaling points, P1 and P2, the input window (soft clip limits) and the GL2 cursor are all reset to their default positions.
- The polygon buffer is emptied.

Page size - **<ESC>&ℓnA**

The command selects the paper or envelope size that the printer will use.

Values for n are as follows

1	Executive
2	Letter
3	Legal
26	A4 international
80	Monarch (envelope)
81	Com-10 (envelope)
90	International DL (envelope)
91	International C5 (envelope)

Other values of n are ignored.

This command does not support B5 international size paper.

In all other respects the command is the same as the Select paper size - **<ESC>[S n** command.

The factory default page size is A4.

Page length - <ESC>&l nP

The command sets the logical page length in text lines.

n is the length of the logical page in lines (at the current VMI setting).

This command effectively selects the paper size: the smallest available size onto which the logical page can fit.

However, the Page size and Select paper size commands are preferable for page size selection.

It is best to use this command in conjunction with the Vertical motion index command to take advantage of an existing page size setting.

On receipt of this command the printer prints out any pages remaining in the printer buffer.

Paper sizes and the equivalent page length settings in text lines at 6 and 8 lines-per-inch are as follows.

Page	Portrait		Landscape	
	6 lpi	8 lpi	6 lpi	8 lpi
Letter	66	88	51	68
Legal	84	112		
A4	70	93	49	66
Executive	63	84	43	58

If cassette selection has been set to “Automatic” from the control panel, and there is a tray inserted containing the selected size, paper is automatically fed from that tray.

Otherwise, if the paper size selected conflicts with the size of the paper in the selected paper feeder, a message appears on the control panel requesting you to change the paper tray.

You can override this request from the control panel. The printer will then proceed to use the paper in the currently selected feeder.

The left, right and top margins, and the text length are set to the default values for the new page size.

The picture frame is set to its default size and position.
Any overlaid macro will be discarded.

If the value of n that you specify selects a logical page longer than any available paper size, or if the current VMI setting is 0, the logical page length and the paper size are not changed. However, the command still prints out any remaining pages, discards any overlaid macro and resets the margins and text length.

You cannot select Legal size in landscape orientation with this command. To do this first select Legal in portrait orientation, then change the orientation of the logical page to landscape.

You can set text length in lines-per-page from the control panel; however, this can alter the current VMI setting.

The command also has the following effects on the GL2 vector graphics state.

- The GL2 horizontal and vertical plot sizes are reset to their default values, equal to the width and height of the PCL picture frame. GL2 plot size is explained in Chapter 5.
- The scaling points, P1 and P2, the input window (soft clip limits) and the GL2 cursor are all reset to their default positions.
- The polygon buffer is emptied.

Left offset registration - **<ESC>&l nU**

The command sets the horizontal offset of the logical page from its default position.

n specifies the offset in decipoints ($1/720''$) of the left edge of the logical page.

A positive value of n moves the logical page to the right on the physical page, a negative value moves it to the left.

To shift the logical page $1/4''$ to the right, set n to 180 ($180 \times 1/720'' = 1/4''$).

The command always moves the logical page across the width of the physical page, no matter what the current logical page orientation.

n is accurate to 2 decimal places.

Top offset registration - `<ESC>&lnZ`

The command sets the vertical offset of the logical page from its default position.

n specifies the offset in decipoints ($1/720''$) of the top edge of the logical page.

A positive value of n moves the logical page down the physical page, a negative value moves it upwards.

To shift the logical page $1/2''$ down the physical page, set n to 360 ($360 \times 1/720'' = 1/2''$).

The command always moves the logical page up or down the length of the physical page, no matter what the current logical page orientation.

n is accurate to 2 decimal places.

Logical page orientation - <ESC>&l nO

The command sets the orientation of the logical page relative to the physical page.

Values for *n* are as follows.

0	Portrait
1	Landscape
2	Reverse portrait
3	Reverse landscape

Values other than 0, 1, 2 or 3 are ignored.

On receiving this command, the printer prints out any pages remaining in the printer buffer.

The cursor is placed at the default cursor position on the next page.

The command resets page length, text length, top, left and right margins, HMI and VMI to their user default values. The picture frame is reset to its default size and position.

Any macro has been enabled for overlay will be discarded.

The logical page orientation and print direction settings together determine the orientation of text on the page.

To print in more than one orientation on a single page use the Print direction command or Select orientation command. The printer will automatically rotate fonts as necessary.

Portrait or landscape orientation can also be selected from the control panel.

The factory default logical page orientation is portrait.

The command also has the following effects on the GL2 vector graphics state.

- The GL2 horizontal and vertical plot sizes are reset to their default values, equal to the width and height of the PCL picture frame. GL2 plot size is explained in Chapter 5.
- The scaling points, P1 and P2, the input window (soft clip limits) and the GL2 cursor are all reset to their default positions.
- The polygon buffer is emptied.

Select orientation - <ESC> [O *n*

The command rotates the orientation of printing.

This command will work in any emulation mode.

n = 0 selects portrait.

n = 1 selects landscape.

In portrait orientation the coordinate system origin is in the top left-hand corner of the page, in landscape orientation it is in the bottom left-hand corner. Hence, cursor positioning must be amended accordingly.

The command resets page length, text length, top, left and right margins, HMI and VMI to their user default values.

The picture frame is reset to its default size and position.

The printer automatically rotates fonts as necessary.

Portrait or landscape orientation can also be selected from the control panel.

The factory default logical page orientation is portrait.

4.9.3 Margins and line spacing commands

The commands in this section set the row and column size, the coordinate system units, the number of lines per page and the margins.

Horizontal motion index - <ESC>&k*n*H

The command sets the column width in 1/120".

A column is the unit of horizontal movement across the width of the logical page.

If the current font is monospaced, the HMI is the horizontal distance the cursor moves when any single character is printed. The Space (<SP>) and Backspace (<BS>) control codes move the cursor a distance of one column.

If the current font is proportionally spaced, the HMI is the horizontal distance the cursor moves when a Space control code is sent to the printer. The distance the cursor moves when a character is printed depends on its shape.

Switching between the primary and secondary fonts using the Select primary font (<SO>) or Select secondary font (<SI>) control codes, or changing any font attributes (e.g. point size or style) resets the HMI to the new current font's default pitch.

Margin settings are not affected by a change in the HMI.

n must be in the range 0 – 32767 and is accurate to 4 decimal places.

The factory default HMI is 1/10".

Vertical motion index - <ESC>&l*n*C

The command sets the height of a single row in 1/48".

A row is the unit of vertical movement down the length of the logical page.

The VMI is the vertical distance the cursor moves down the page when a Line feed (<LF>) control code is sent to the printer. The setting determines both the Line feed (<LF>) and Half line feed (<ESC>=) distances.

If you try to set the VMI to greater than the length of the logical page, the command is ignored.

The position of the top margin is not affected by a change in the VMI.

The factory default setting is 8: equivalent to 6 lines of text per inch.

If the text length is changed from the control panel, the VMI will automatically be changed.

n must be in the range 0 – 32767 and is accurate to 4 decimal places.

Set line spacing - **<ESC>&l nD**

The command sets the number of text lines printed per inch.

n can be 1, 2, 3, 4, 6, 8, 12, 16, 24 or 48.

Subsequent text is printed at *n* lines per inch.

Values of *n* other than those listed above are ignored.

The command is equivalent to the Vertical motion index command and sets the VMI to $1/n$ "

The setting determines both the Line feed (<LF>) and Half line feed (<ESC>=) distances.

The position of the top margin is not affected by a change in the line spacing.

The factory default setting is 6 lines per inch.

The number of lines per page can also be set from the printer's control panel.

Set left margin - **<ESC>&anL**

The command sets the distance between the left edge of the logical page and the left margin in columns.

The width of a column is set using the Horizontal motion index command.

The left margin setting remains in effect until a new one is set or another command resets the margin to its default position.

Subsequent changes to the HMI do not affect the margin's position.

If you try to set the left margin to the right of the current right margin, the command is ignored.

If the cursor is to the left of the new left margin setting, it is moved to the new left margin.

The factory default left margin is the left edge of the logical page.

Margin settings can also be made from the printer's control panel.

Set right margin - **<ESC>&anM**

The command sets the distance between the left edge of the logical page and the right margin in columns.

The width of a column is set using the Horizontal motion index command.

The right margin setting remains in effect until a new one is set or another command resets the margin to its default position.

Subsequent changes to the HMI do not affect the margin's position.

If you try to place the right margin beyond the right edge of the logical page, the margin is set to the right edge of the logical page.

If you try to set the right margin to the left of the current left margin, the command is ignored.

If the cursor is to the right of the new right margin setting, it is moved to the new right margin.

The factory default right margin is the right edge of the logical page.

Margin settings can also be made from the printer's control panel.

Clear horizontal margins - **<ESC>9**

The command resets the positions of the left and right margins.

The default left margin is the left edge of the logical page.

The default right margin is the right edge of the logical page.

Top margin - `<ESC>&l nE`

The command sets the distance between the top edge of the logical page and the top margin in rows.

The height of a row is set using the Vertical motion index or Set line spacing command.

The top margin setting remains in effect until a new one is set or another command resets the margin to its default position.

Subsequent changes to the VMI do not affect the margin's position.

The command resets the text length: $\text{text length} = \text{logical page length} - (\text{top margin} + 1/2")$. This automatically sets a bottom margin of $1/2"$.

If you try to set the top margin to be greater than the current logical page length, the command is ignored.

If the current VMI is 0, the command is ignored.

The factory default top margin is $1/2"$ below the top of the logical page.

Margin settings can also be made from the printer's control panel.

Text length - **<ESC>&/nF**

The command sets the number of lines of text per page.

Printing starts from the top margin.

The text length and the current VMI (or line spacing) settings together determine the length of the text area: the area of the logical page in which text can be printed.

The text area length and top margin setting effectively set a bottom margin.

If a value is specified that would cause the text area to extend below the bottom of the logical page, the command is ignored.

If the current VMI setting is 0, the command is ignored.

The text length can also be set from the control panel: the VMI is automatically recalculated so that the length of the text area does not change.

Any of the following commands, Page size, Select paper size, Page length, Logical page orientation or Top margin, reset the text length to its user default (control panel) setting.

The factory default text length for a particular page size = (default logical page length - 1") × 6. The result is rounded down to the nearest integer.

Perforation skip - **<ESC>&/nL**

The command turns perforation skip on or off.

$n = 0$ turns off perforation skip. $n = 1$ turns on perforation skip.

Other values are ignored.

The perforation region is the area between the bottom margin of one page and the top margin of the next.

When perforation skip is on, the printer prints inside the text area until it receives a command that would move the cursor below the bottom margin. The printer then ejects the current page and moves the cursor to the default cursor position on the next page. If there is data in the printer buffer, printing continues on the new page.

When perforation skip is off, commands can move the cursor down into the perforation region, enabling printing below the bottom margin and above the top margin.

Changing the perforation skip setting resets the top margin and page length to their default values.

The factory default setting is perforation skip on.

4.9.4 Positioning the cursor

The commands in this section are used to position the cursor. In addition, up to 20 cursor positions can be stored and retrieved.

Space - <SP>

The <SP> control code moves the cursor one column to the right, as defined by the current HMI setting.

If the current font is monospaced, the cursor moves one column to the right.

If the current font is proportionally spaced, the cursor moves one column to the right, unless a special space character is defined in the current symbol set, in which case the defined space character is printed and the cursor is moved to the right by the width of the character.

Carriage return - <CR>

The <CR> control code moves the cursor to the left margin on the current line.

The cursor does not move vertically.

The Line termination command or the control panel auto line feed function can be used to set the <CR> control code to perform a carriage return/line feed, to move the cursor to the left margin on the next line.

Line feed - <LF>

The <LF> control code moves the cursor down the page one row, as defined by the VMI set by the most recent Vertical motion index or Set line spacing command.

The cursor does not move horizontally.

The Line termination command can be used to set the <LF> control code to perform a carriage return/line feed, to move the cursor to the left margin on the next line.

Form feed - <FF>

The <FF> control code ejects the current page and moves the cursor to the first line of the text area on the next page.

The cursor does not move horizontally.

The cursor is positioned 3/4 of a row below the top margin, as defined by the VMI set by the most recent Vertical motion index or Set line spacing command.

The Line termination command can be used to set the <FF> control code to perform a form feed and a carriage return, to move the cursor to the left margin on the first line of the next page.

Backspace - <BS>

The <BS> control code moves the cursor one column or one character width to the left.

If the current font is monospaced, the cursor moves one column width to the left.

If the current font is proportionally spaced, a <BS> code moves the cursor a distance equal to the width of the overstrike character.

If the currently selected font is proportionally spaced, multiple <BS> codes move the cursor a distance equal to the width of the most recently printed character. For example, if you print the string "abcd" followed by four <BS> control codes, the cursor moves to the left four times the width of the 'd'.

If the cursor is on the left margin when a Backspace is sent, the code is ignored.

Horizontal tab - <HT>

The <HT> control code moves the cursor to the next tab stop on the current line.

The tab stops are at the left margin and at every eighth column across the text area.

If there are no tab stops between the current cursor position and the right margin, the cursor moves to the right margin.

If the current HMI setting is 0, the command is ignored.

Horizontal cursor position (columns) - **<ESC>&anC**

The command positions the cursor horizontally in column units.

Movement can either be absolute with respect to the left edge of the logical page, or relative with respect to the current cursor position.

The width of a column is as defined by the current HMI setting.

n specifies the number of columns the cursor is to be moved.

If n is unsigned, the cursor is moved n columns to the right of the left edge of the logical page.

A plus or minus sign before n denotes movement relative to the current cursor position.

A plus sign before n moves the cursor n columns to the right of the current cursor position.

A minus sign before n moves the cursor n columns to the left of the current position.

The cursor's vertical position does not change.

n is accurate to 4 decimal places.

The command can move the cursor outside the horizontal margins but not outside the edges of the logical page.

If a position outside the edges of the logical page is specified, the cursor is moved to the left or right edge of the logical page.

Horizontal cursor position (dots) - **<ESC>*pnX**

This command performs exactly the same function as the Horizontal cursor position (columns) command described above. The only difference is that the units used are dots (1/300"), not columns.

Horizontal cursor position (decipoints) - **<ESC>&anH**

This command also performs exactly the same function as the Horizontal cursor position (columns) command described above. The only difference is that the units used are decipoints (1/720"), not columns.

n is accurate to 2 decimal places.

Vertical cursor position (rows) - **<ESC>&nR**

The command positions the cursor vertically in row units.

Movement can either be absolute with respect to the top of the logical page, or relative with respect to the current cursor position.

The height of a row is as defined by the current VMI setting.

n specifies the number of rows the cursor is to be moved.

If *n* is unsigned, the cursor is moved *n* rows down from the current top margin. (Hence a Top margin command affects subsequent absolute movement specified with this command).

A plus or minus sign before *n* denotes movement relative to the current cursor position.

A plus sign before *n* moves the cursor *n* rows down from the current cursor position.

A minus sign before *n* moves the cursor *n* rows up from the current position.

The cursor's horizontal position does not change.

n is accurate to 4 decimal places.

The command can move the cursor beyond the top and bottom margins but not outside the edges of the logical page.

If a position outside the edges of the logical page is specified, the cursor is moved to the top or bottom edge of the logical page.

Vertical cursor position (dots) - **<ESC>*pnY**

This command performs exactly the same function as the Vertical cursor position (rows) command described above. The only difference is that the units used are dots (1/300"), not rows.

Vertical cursor position (decipoints) - **<ESC>&nV**

This command also performs exactly the same function as the Vertical cursor position (rows) command described above. The only difference is that the units used are decipoints (1/720"), not rows.

n is accurate to 1 decimal place.

Half line feed - **<ESC>=**

The command moves the cursor down the page by half a row.

Row (or line) height is the VMI set by the most recent Vertical motion index or Set line spacing command.

The cursor's horizontal position does not change.

Push/pop cursor position - **<ESC>&fnS**

Up to 20 cursor positions can be stored on the cursor position stack.

This command stores the current cursor position or retrieves a stored position.

If $n = 0$, the current cursor position is placed on the stack. The current position does not change.

If $n = 1$, the cursor position on top of the stack is removed from the stack and made the current cursor position.

Cursor positions are retrieved (popped) from the stack in the opposite order to that in which they were placed (pushed) onto the stack.

If you try to store more than 20 positions, or try to retrieve a cursor position from an empty stack, the command is ignored.

Cursor positions are always interpreted relative to the top left-hand corner of the current logical page in its current orientation. Hence, a cursor position retrieved from the stack may have a new physical location on the page.

If a popped cursor position lies outside the logical page, the cursor is positioned on the edge of the logical page closest to it.

A Reset empties the stack.

4.9.5 Miscellaneous output commands

The commands described in this section change the way in which output appears.

Print direction - `<ESC>&anP`

This command allows printing of text in multiple directions on a single page.

The command changes the orientation of the logical page with respect to the physical page but does not eject the current page.

n specifies the angle or rotation in degrees, and can be 0, 90, 180 or 270. The logical page is rotated counterclockwise through the selected angle. The equivalent orientations are as shown below.

0	Portrait
1	Landscape
2	Reverse portrait
3	Reverse landscape

The cursor position retains the same physical position, thus its coordinates change.

Margins are translated. For example, if $n = 90$ the top margin becomes the new left margin, the left margin becomes the new bottom margin, the bottom margin becomes the new right margin and the right margin becomes the new top margin.

The margin positions relative to the edges of the logical page do not change.

Subsequent text and graphics are printed in the new orientation.

The current HMI setting does not change.

The command has no effect on GL2 graphics state.

If a value of n other than the above is specified, the command is ignored.

The factory default orientation is portrait.

Line termination - **<ESC>&k n G**

The command selects the way in which the printer interprets the carriage return, line feed and form feed control codes.

Set n as follows.

0	CR=CR, LF=LF, FF=FF
1	CR=CR/LF, LF=LF, FF=FF
2	CR=CR, LF=CR/LF, FF=CR/FF
3	CR=CR/LF, LF=CR/LF, FF=CR/FF

If n is set to a value other than 0, 1, 2 or 3, the command is ignored.

End of line wrap - **<ESC>&s n C**

The command specifies the action taken when text is about to go over the right margin.

$n = 0$ turns text wrap on: lines longer than the width of the text area flow onto the next line. An automatic carriage return/line feed is performed so that text is not lost.

$n = 1$ turns text wrap off: lines longer than the width of the text area are clipped at the right margin. When text is clipped any part of a character that would lie beyond the right margin will not appear on the printed page.

When text is clipped, the cursor is moved to the right margin.

This command is mainly for use with the display functions mode.

If n is set to a value other than 0 or 1, the command is ignored.

The factory default setting is End of line wrap off.

Display functions on - <ESC> Y

The command turns on display functions mode.

In display functions mode the printer prints out escape sequences and control codes instead of executing them.

The only commands which do function in this mode are <CR>, which performs a carriage return/line feed, and <ESC> Z, which turns display functions mode off.

Data is printed out in the current font and inside the current text area.

The <ESC> Z command is both printed out and executed.

Display functions mode allows printing of characters defined for character codes 0, 7 – 15 and 27.

Most symbol sets do not have printable characters defined in the code ranges 0 – 31 and 128 – 159. If no character is defined, a space is printed instead.

Display functions off - <ESC> Z

The command turns off display functions mode.

“<ESC> Z” is printed, but all subsequent escape sequences and control codes are executed normally and not printed out.

Self test - <ESC> z

The command prints out a test sheet.

4.10 Fonts

4.10.1 Introduction

Fonts and typography are discussed fully in Chapter 3.

In the context of the HP LaserJet III emulation, a font is a set of symbols of a given size, pitch, typeface, weight and style, for example, 12 point, 10 characters-per-inch Courier medium italic in the Roman-8 symbol set.

Bitmap fonts are fonts whose character size is fixed, for example Courier 12 point.

Scalable fonts are fonts with no implied character size: you can choose any size you want when you select a scalable font for printing.

The printer comes with fourteen bitmap fonts and eight scalable fonts which are stored in the printer's ROM (read-only memory). These are known as internal fonts. You can select these fonts and print using them at any time using PCL commands or the control panel.

Using PCL commands you can also select any other font, providing that you make it available to the printer in one of the following ways. Bitmap and scalable fonts are available on cartridges which you can plug into the printer's cartridge slots. These fonts are then ready for use just as if they were internal fonts. Bitmap and scalable fonts are also available on floppy disk or CD-ROM. You can download these fonts to the printer from your computer. In the same way, you can also download fonts that you have created yourself on your computer. Fonts transferred from a computer are stored in the printer's RAM (random access memory) and are referred to as downloaded or 'soft' fonts.

However, the printer also uses its RAM to compose each page prior to printing it out. If you download too many fonts the printer may run out of memory and be unable to continue printing.

Downloaded fonts are deleted from the printer's memory when you switch it off. To use them again you must redownload them.

Using PCL commands you can select any font by specifying its attributes, for example Roman-8, 12 point, 10 characters-per-inch Courier medium italic. Fonts you select must be available in ROM, RAM or on cartridge, oth-

erwise the printer will not be able to render the exact font specifications that you select. If the printer cannot find the designated font, it will try to match your selection as closely as it can using the fonts available.

Fonts can also be selected by a unique ID number.

The most recently selected font is known as the current font. At all times the printer maintains two font definitions: the primary font and the secondary font. You can define and modify these with PCL commands or from the control panel. A single PCL command sets either of these to be the current font.

Escape codes with ‘(‘ as the second character specify font characteristics of the primary font.

Escape codes with ‘)’ as the second character specify font characteristics of the secondary font.

The factory default primary and secondary fonts are both Courier 12 point, 10 characters per inch, medium, upright, in the Roman-8 symbol set.

4.10.2 *Selecting a font*

To select a font for printing, either use PCL commands to specify its characteristics, or select it with a single PCL command by quoting its unique identification number.

If you select a font by designating its characteristics, the printer will print exactly as you specify, provided that the font is available either as an internal font, on cartridge or in memory. For example, if you specify 12 point, 10 cpi, Courier italic Roman-8 as the primary font, the printer can print in exactly that font, as it is one of the internal fonts.

However, if you specify a font that is not available, the printer will instead select an available font whose characteristics match your specifications in the following order of priority:

symbol set
spacing type
pitch (for fixed space fonts only)
height
style
stroke weight
typeface

These attributes are explained fully in the following section.

4.10.3 *Symbol set*

A symbol set is a pre-defined set of symbols. Symbol sets normally contain lower and upper case letters, the digits 0 to 9, punctuation marks and other commonly used symbols. Some symbol sets are designed for particular purposes, for example, for printing text in a foreign language or for printing mathematical expressions.

Symbol set is the highest priority font attribute. The printer will always ensure that the selected symbol set is used if it is available, even at the expense of all the other specified attributes

4.10.4 Character spacing

There are two types of spacing, fixed spacing (monospacing) and proportional spacing.

The characters of a fixed space font all occupy the same amount of space on a line. Hence all eight-letter words in a particular fixed space font will be of equal length on the printed page.

The characters of a proportionally spaced occupy varying amounts of space on a line according to their design. Hence eight-letter words in a proportionally spaced font will vary in length.

4.10.5 Pitch

The pitch setting is the number of characters per inch on a line. Pitch only applies to fixed space fonts.

4.10.6 Height

A font's height or point size is the vertical distance in points ($1/72''$) between the top of the font's highest ascender and the bottom of its lowest descender. See on page 24 of Chapter 3 for an explanation of the terms "ascender" and "descender". Bitmap fonts are available only in fixed sizes, for example 10 point or 12 point. Scalable fonts are available in any size from 0.25 point to 999.75 point, in steps of 0.25 points.

4.10.7 Style

Font style is the combination of posture (upright or italic), width (condensed, normal or expanded etc.) and structure (solid, outline or shadow etc.). Both upright and italic fonts are available as internal fonts. Fonts with widths other than normal, or structures other than solid must be installed on a cartridge or downloaded.

4.10.8 Stroke weight

Stroke weight is the thickness of the strokes which makes up the font's characters. Normal line thickness is known as medium. Bold and light are thicker and thinner stroke weights respectively.

There are 15 different stroke weights ranging from Ultra Thin to Extra Black.

Both medium and bold fonts are available as *internal fonts*. To print text in other stroke weights you must install the appropriate font on a cartridge or download it to the printer.

If the printer cannot match the selected stroke weight exactly, it matches it as closely as possible.

4.10.9 Typeface

The typeface of a font is the unique style of the characters. Common examples include Times, Univers, Palatino and Courier. The printer's own installed typefaces are Courier, Line Printer, Univers and CG Times. You can select fonts in these typefaces at any time.

Typeface is the lowest priority font attribute. When you select a font in a given typeface, ensure that an exact match is available in one of the font locations. If the exact font is not available, the printer may substitute a font of a different typeface that matches higher priority attributes, such as stroke weight.

4.10.10 Location

If two available fonts match your font specification equally, the printer chooses between them according to their location. Downloaded fonts have the highest priority and internal fonts the lowest. A bitmap font takes priority over a scalable font in the same location.

4.10.11 Orientation

The printer can rotate any font to each of the four logical page orientations. However, this uses up memory and can be slow. Fonts are commercially available in different orientations to make printing in different logical page orientations easier.

If two fonts match the font specification equally in all categories, the printer selects the font whose orientation is the same as the current logical page orientation.

4.10.12 Font selection commands

These commands control and specify the attributes of the primary and secondary fonts. Commands override any previous settings you have made, both those made using PCL commands and those made using the control panel.

Select primary font - <SI>

The command makes the primary font the current font.

Subsequent text is printed in the primary font.

Select secondary font - <SO>

The command makes the secondary font the current font.

Subsequent text is printed in the secondary font.

Set primary font to default values - <ESC>(3@

The command sets the primary font to the user default (control panel) primary font settings

Any primary font settings made using PCL commands are discarded.

Set secondary font to default values - <ESC>)3@

The command sets the secondary font to the user default (control panel) secondary font settings

Any secondary font settings made using PCL commands are discarded.

Select primary font by ID number - <ESC>(nX

The command sets the specified downloaded font to be the primary font.

If *n* matches the ID number of an available font, the primary font attributes are set to that font's attributes.

If the selected font is proportionally spaced, the current primary font pitch setting is retained for possible future use.

When selecting a scalable font with this command, be sure to specify a point size with the <ESC>(snV command. Otherwise, the current primary font point size setting will be adopted.

Use the Font ID command to assign ID numbers to soft fonts.

If there is no available font with the selected ID number, the command is ignored.

Select secondary font by ID number - <ESC>nX

The command sets the specified downloaded font to be the secondary font.

It functions in the same way as the Select primary font by ID number command described above.

Select primary font symbol set - <ESC>(n

The command selects a symbol set for the primary font.

Symbol set may also be selected from the control panel.

The factory default symbol set for the primary font is Roman-8.

n is a one or two-digit number followed by a letter. A list of available symbol sets is as follows.

ISO 60: Norwegian	0D	HP Spanish	1S
Roman Extension	0E	ISO 57: Chinese	2K
ISO 25: French	0F	ISO 17: Spanish	2S
HP German	0G	ISO 2: IRV	2U
ISO 15: Italian	0I	ISO 10: Swedish	3S
JIS ASCII	0K	ISO 16: Portuguese	4S
ECMA-94 Latin 1	0N	ISO 84: Portuguese	5S
ISO 11: Swedish	0S	ISO 85: Spanish	6S
US-ASCII	0U	Roman-8	8U
ISO 61: Norwegian	1D	IBM-PC(US)	10U
ISO 4: UK	1E	IBM-PC(Denmark/Norway)	11U
ISO 69: French	1F	PC-850	12U
ISO 21: German	1G		

Select secondary font symbol set - <ESC>) *n*

The command selects a symbol set for the secondary font.

Symbol set may also be selected from the control panel.

The factory default symbol set for the secondary font is Roman-8.

n is a one- or two-digit number followed by a letter. A list of available symbol sets is shown under the description of the Select primary font symbol set command.

Select primary font spacing type - <ESC>(*s**n*P

The command selects the spacing type for the primary font.

n = 0 selects fixed spacing. *n* = 1 selects proportional spacing.

If you specify proportional spacing and a proportionally spaced font is not available in the current symbol set, a fixed pitch font will be selected instead.

The user default primary font spacing is determined by the typeface you select for the user default primary font. For example, a Courier font always has fixed spacing, a Univers font is always proportionally spaced.

Select secondary font spacing type - <ESC>)*s**n*P

The command selects the spacing type for the secondary font. It functions in the same way as the Select primary font spacing type command described above.

Set primary font pitch - **<ESC>(snH**

The command sets the pitch of the primary font.

The pitch setting is specified in characters per inch.

Pitch only applies to monospaced fonts.

If you specify a pitch while the current primary font is proportionally spaced, the setting is recorded. If you later select a monospaced font, your pitch setting will take effect.

If there is no font available with the specified pitch, the nearest available higher pitch setting is used instead. If no higher pitch setting is available, the closest lower setting is used.

The pitch of a scalable monospaced font is adjusted so that the font height is a multiple of 0.25 points and the ratio of character height to character width is retained.

n is accurate to 2 decimal places.

The user default primary font pitch is determined by the user default primary font.

The factory default pitch setting is 10 characters per inch.

Set secondary font pitch - **<ESC>)snH**

The command sets the pitch of the secondary font.

It functions in the same way as the Set primary font pitch command described above.

Set primary font point size - **<ESC>(snV**

The command sets the point size of the primary font.

One point = 1/72".

For scalable fonts n can range from 0.25 to 999.75. n is rounded to the nearest 0.25.

For bitmap fonts the command will select a font with a point size within 0.25 points of n .

n is accurate to 2 decimal places.

If the specified height is unavailable, the closest available height is selected instead.

When selecting a scalable font by ID number, be sure to specify the point size as well. Otherwise the current primary font point size value will be used.

The point size attribute does not apply to monospaced scalable fonts, whose height is determined by the current pitch setting. If you designate a primary font point size while the current primary font is a monospaced scalable font, the command has no immediate effect. However, the new point size setting is recorded and is applied if you later select a proportionally spaced scalable font or a bitmap font.

The user default primary font point size is determined by the user default primary font.

The factory default primary font height is 12 point.

Set secondary font point size - **<ESC>)snV**

The command sets the point size of the secondary font.

It functions in the same way as the Set primary font point size command described above.

The user default secondary font point size is determined by the user default secondary font.

The factory default secondary font height is 12 point.

Select primary font style - <ESC>(snS

The command sets the primary font style

Set *n* to the number that corresponds to the style you wish to select.

Common style values are shown below.

0	Upright, solid	24	Expanded
1	Italic	32	Outline
4	Condensed	64	Inline
5	Condensed italic	128	Shadowed
8	Compressed (extra condensed)	160	Outline shadowed

In order to take effect straightaway, the specified style must exactly match that of an available font.

If there is no font in the specified style which matches the current settings for the higher priority font attributes (symbol set, spacing, pitch and point size), the command has no immediate effect. However, the style selection is recorded and is applied if changes to the higher priority font attributes allow an exact match between your style selection and an available font.

The user default primary font style is determined by the current user default primary font.

The factory default primary font style is upright, solid.

Select secondary font style - <ESC>)snS

The command sets the secondary font style.

It functions in the same way as the Set primary font style command described above.

The user default secondary font style is determined by the current user default secondary font.

The factory default secondary font style is upright, solid.

Select primary font stroke weight - <ESC>(snB

The command selects the stroke weight for the primary font.

n is in the range -7 to 7 . $n=0$ selects medium stroke weight, negative numbers select lighter stroke weights and positive numbers select bolder stroke weights.

Stroke weight names and the corresponding values for n are shown below.

-7	Ultra Thin	-2	Demi Light	3	Bold
-6	Extra Thin	-1	Semi Light	4	Extra Bold
-5	Thin	0	Medium (Book or Text)	5	Black
-4	Extra Light	1	Semi Bold	6	Extra Black
-3	Light	2	Demi Bold	7	Ultra Block

Bold fonts have a stroke weight of 3.

Light fonts have a stroke weight of -3 .

If there is no font with the selected stroke weight which matches the current settings for the higher priority font attributes, the printer attempts to come as close as possible as follows:

If you select a stroke weight in the range 0 to 7 which is unavailable, the printer selects the closest available bolder stroke weight. If no bolder stroke weight font is available, the closest lighter stroke weight font is selected.

If you select a stroke weight in the range -7 to -1 which is unavailable, the printer will select the closest available lighter stroke weight. If no lighter stroke weight font is available, the closest bolder stroke weight font is selected.

In either case the stroke weight selection is recorded and is applied if changes to the higher priority font attributes allow an exact match between your stroke weight selection and an available font.

The user default primary font stroke weight is determined by the user default primary font setting.

The factory default primary font stroke weight is medium.

Select secondary font stroke weight - <ESC>snB

The command selects the stroke weight for the secondary font.

It functions in the same way as the Set primary font stroke weight command described above.

The user default secondary font stroke weight is determined by the user default secondary font setting.

The factory default secondary font stroke weight is medium.

Select primary font typeface - <ESC>(snT

The command specifies the typeface for the primary font.

The printer's internal typefaces are Courier, Line Printer, Univers and CG Times.

n is in the range 0 to 32767 and designates either the typeface base value, a number between 0 and 511, or the typeface family value, a number which identifies the typeface, the vending company and the font version.

The table below lists typefaces and their corresponding values.

Typeface	Base value	Base value + family value
Line printer	0	4096
Courier	3	4099
Times	5	4101
Univers	52	4148

If the typeface you select is not available, the command is ignored.

The user default primary font typeface is determined by the user default primary font.

The factory default primary font typeface is Courier.

Select secondary font typeface - **<ESC>)snT**

The command specifies the typeface for the secondary font.

It functions in the same way as the Select primary font typeface command described above.

The user default secondary font typeface is determined by the user default secondary font.

The factory default secondary font typeface is Courier.

Turn underlining on - **<ESC>&dnD**

This command turns on underlining.

When underlining is on, all printed text is underlined.

Horizontal cursor movement from left to right on the logical page is also underlined.

$n = 1$ selects fixed underlining, $n = 3$ selects floating underlining.

A fixed underline is drawn in the same place for each character of a given font. If the font does not change, the line is continuous. The line is 5 dots (1/60") below the font's baseline and 3 dots (1/100") thick.

A floating underline is drawn in the same place for every character on a line. The underline is continuous irrespective of changes in font.

The underline and the underscore character may not align or be the same thickness.

Turn underlining off - **<ESC>&d@**

The command turns underlining off.

Transparent print data - **<ESC>&pnX <character data>**

The command allows you to print characters that are normally unprintable.

n specifies the number of data bytes that follow the command.

All bytes are interpreted as character codes. The corresponding characters in the current symbol set are printed.

If there is no character for a particular code, a space is printed instead.

This command is useful for printing special characters in symbol sets such as the IBM All Character Set, in which every code corresponds to a character.

Control codes in the data have no effect.

4.10.13 Font selection examples

The following example sequence serves to illustrate some important points about font selection.

First CG Times 10 point Bold in the Roman-8 symbol set is selected as the primary font.

```
10 LPRINT CHR$(27) + "(8U";  
20 LPRINT CHR$(27) + "(s1p10v0s3b4101T";
```

Next, Univers 12 point in the Roman-8 symbol set is selected as the secondary font, and made the current font.

```
10 LPRINT CHR$(27) + ")8U";  
20 LPRINT CHR$(27) + ")s1p12v0s0b52T";  
30 LPRINT CHR$(14);
```

Next, Courier 12 point Italic in the ISO 69: French symbol set is selected as the secondary font.

```
10 LPRINT CHR$(27) + ")1F";  
20 LPRINT CHR$(27) + ")s0p10h12v1s0b3T";
```

In this case the font selection command sequence must include a command to select fixed spacing type. This is because spacing type has a higher priority than typeface. If fixed spacing selection were omitted, the previous choice of proportional spacing (for Univers 12 point) would still be in force, and the selection of Courier typeface would be ignored, since Courier typeface fonts always employ fixed spacing.

Both height and pitch were specified, however, pitch alone would have been sufficient.

Finally, 14 point Univers Light in the PC-8 symbol set is selected as the secondary font. This assumes that a scalable Univers Light font has been downloaded or is on an installed cartridge.

```
10 LPRINT CHR$(27) + ")10U";  
20 LPRINT CHR$(27) + ")s1p14v0s-3b52T";
```

If Univers Light is unavailable, but a light font in a different typeface, for example Helvetica Light, is available, then Helvetica Light will be selected as the secondary font, rather than a Univers font: stroke weight having a higher priority than typeface. If no light font is available in any typeface, Univers Medium will be selected.

4.10.14 Creating and downloading fonts

You can download both commercial fonts and fonts you design yourself to the printer using PCL commands.

Downloaded fonts can also be controlled using the commands described in this section.

Downloaded fonts are either temporary or permanent. Temporary fonts are deleted when the printer is reset, permanent fonts are retained. Fonts are temporary by default. Use the Font control command to make a soft font permanent.

Font ID - <ESC>*c*n*D

The command either identifies an existing soft font for processing by the Font control command, or assigns a number to a new soft font.

If you issue this command and then download a font to the printer, the number is assigned to the new font. If a font with that ID number was already in the printer's memory, it is overwritten by the new font.

You can also use the command to specify the ID of a font already in the printer's memory in order to perform an operation on it with the Font control command.

Font control - <ESC>*c*n*F

The command performs the specified operation on one or more downloaded fonts.

For single font operations, the font is identified by the Font ID command.

n = 0 deletes all downloaded fonts from the printer's memory.

n = 1 deletes all temporary fonts from memory.

n = 2 deletes a selected font from memory.

n = 3 deletes a specified character from the selected font. The command deletes the character specified by the most recent Character code command.

n = 4 makes the selected font temporary.

n = 5 makes the selected font permanent.

n = 6 makes a copy of the current font and assigns to it the most recently specified font ID number.

Sending a font to the printer involves sending a font descriptor, a block of data describing the font, followed by a character code, character descriptor and character data for every character in the font.

Characters are designed on a grid-shaped character cell. Its position on the grid determines a character's size, shape and alignment to other characters in the font.

Soft fonts created by the user are either in bitmap format or Intellifont scalable format. The Intellifont format is beyond the scope of this manual. The bitmap format is described in the following section.

The sequence of commands is as follows:

```
Send font descriptor
Send character code
Send character descriptor
<data>
Send character code
Send character descriptor
<data>
Send character code
Send character descriptor
<data>
...etc
```

Send font descriptor - <ESC>snW <descriptor>

The command sends the font descriptor to the printer.

n specifies the length of the font descriptor in bytes.

The font descriptor block contains attributes common to all the characters in the font.

Attributes are represented using 8 different data types. The type of each attribute field is indicated by the initials shown in the table.

B	Boolean	SI	Signed integer
UB	Unsigned byte	ULI	Unsigned long integer
SB	Signed byte	SLI	Signed long integer
UI	Unsigned integer	ASC	ASCII string

The font descriptor block for bitmap fonts is shown below.

Byte	MSB	LSB	Byte	MSB	LSB
0	Font descriptor size (64)		26	Typeface MSB	Serif style
2	Descriptor format (0)	Font type	28	Quality	Placement
4	Style MSB	Reserved	30	Underline distance	Underline height
6	Baseline position		32	Text height	
8	Cell width		34	Text width	
10	Cell height		36	First code	
12	Orientation	Spacing	38	Last code	
14	Symbol set		40	Pitch extended	Height extended
16	Pitch		42	Cap height	
18	Height		44-47	Font number	
20	x-Height		48-63	Font name	
22	Width type	Style LSB	64	Copyright (optional)	
24	Stroke weight	Typeface LSB			

The significance of the fields is as follows:

Font descriptor size (UI) - The size in bytes of the font descriptor block. 64 for bitmap format.

Descriptor format (UB) - 0 for bitmap format.

Font type (UB) - 0, 1 or 2. The value sets the range of symbols that can be printed.

Font type	Printable codes
0	32 – 127
1	32 – 127 and 160 – 255
2	0 – 255

To print characters 0, 7 – 15 and 27 in mode 2 use the Transparent print data command.

Style MSB (UI) - Combined with the Style LSB to form the style word.
Style word = posture + (4 × width) + (32 × structure)

Value	Posture
0	Upright
1	Italic
2	Alternate italic
3	Reserved

Value	Width
0	Normal
1	Condensed
2	Compressed (Extra Condensed)
3	Extra Compressed
4	Ultra Compressed
5	Reserved
6	Extended or Expanded
7	Extra Extended or Extra Expanded

Value	Structure	Value	Structure
0	Solid	7	Contour with shadow
1	Outline	8 – 11	Patterned
2	In-line	12 – 15	Patterned with shadow
3	Contour	16	Inverse
4	Solid with shadow	17	Inverse in open border
5	Outline with shadow	18 – 31	Reserved
6	In-line with shadow		

Baseline position (UI) - This field is ignored by the printer.

Cell width (UI) - This field is ignored by the printer.

Cell height (UI) - This field is ignored by the printer.

Orientation (UB) - The font's orientation relative to the current logical page orientation.

Value	Orientation
0	Portrait
1	Landscape
2	Reverse portrait
3	Reverse landscape

Spacing (B) - The spacing type, 0 (fixed pitch) or 1 (proportional spacing).

Symbol set (UI) - The font's symbol set. A symbol set ID consists of a number and a letter. Symbol set attribute = $(32 \times \text{number}) + \text{ASCII value of letter} - 64$. Symbol set IDs are shown in the table with the description of the Select primary font symbol set command.

Pitch (UI) - For bitmap fonts this attribute is combined with Pitch Extended to specify the font's pitch (for proportionally spaced fonts, the width of a space). This attribute field holds the integer part of the font's pitch in 1/1200", e.g. for a 17 cpi font the value would be 70 ($1200/17 = 70.588$).

Height (UI) - For bitmap fonts this attribute is combined with Height Extended to specify the height of the font. This attribute field holds the integer part of the font's height in 1/1200", e.g. for a 10 point font the value would be 166 ($1200 \times 10/72 = 166.667$).

x-height (UI) - This field is ignored for bitmap fonts.

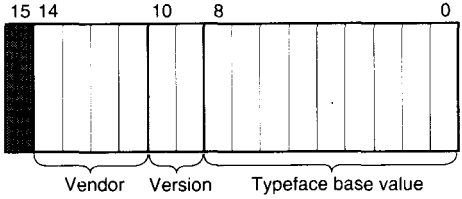
Width type (SB) - This field is ignored by the printer.

Style LSB (UB) - See Style MSB.

Stroke weight (SB) - Values can be from -7 to +7. 0 selects the normal stroke width, -7 the lightest possible stroke weight and 7 the boldest.

	Stroke weight		Stroke weight		Stroke weight
-7	Ultra Thin	-2	Demi Light	3	Bold
-6	Extra Thin	-1	Semi Light	4	Extra Bold
-5	Thin	0	Medium (Book or Text)	5	Black
-4	Extra Light	1	Semi Bold	6	Extra Black
-3	Light	2	Demi Bold	7	Ultra Block

Typeface family (UI) - The field is made up of the Typeface MSB and LSB and identifies the typeface by number. The typeface family field is divided into four parts as follows: bits 0 to 8 hold the typeface base value, bits 9 and 10 hold the version number, bits 11 to 14 identify the vendor and bit 15 is always 0.



Serif style (UB) - Bits 6 and 7 of this byte specify the serif style of the font.

Value	Style
64 - 127	Sans serif
128 - 191	Serif
192 - 255	Reserved

Quality (UB) - This field is ignored by the printer.

Placement (SB) - This field is ignored by the printer.

Underline distance (SB) - For bitmap fonts the field specifies the distance from the baseline to the top row of the underline. A positive value specifies an underline above the baseline, a negative value specifies one below the baseline. A value of 0 specifies an underline on the baseline.

Underline height (UB) - This field is ignored by the printer. Bitmap fonts always print an underline 3 dots thick.

Text height (UI) - This field is ignored by the printer

Text width (UI) - This field is ignored by the printer

First code (UI) - This field is ignored by the printer

Last code (UI) - This field specifies the character code of the last (highest numbered) character in the font. The range of printable codes for each font type is shown in the table. However, the value may be greater than the last code in the symbol set specified by the Font Type byte, since there may be components of compound characters that are not part of the specified symbol set but which still must be downloaded.

Font type	Highest printable code
0	127
1	255
2	255

Pitch extended (UB) - For bitmap fonts this field holds the fractional part of the character pitch. For a 17 cpi font the value is calculated as follows: $1200/17 = 70.588$, the Pitch byte takes the value 70, Pitch Extended = $0.588 \times 256 = 150$ (rounded down).

Height extended (UB) - For bitmap fonts this field holds the fractional part of the height of the font. For a 10 point font the value is calculated as follows: $1200 \times 10/72 = 166.667$, the Height byte takes the value 166, Height Extended = $0.667 \times 256 = 170$ (after rounding down).

Cap height data (UI) - This field is ignored by the printer

Font number (ULI) - For bitmap fonts this field is ignored.

Font name (ASC) - This 16-byte field can be used to specify the name of the font's typeface. The name is used when the printer prints out a list of available fonts.

Send character code - **<ESC>*cnE**

This command sends a character code to the printer.

n specifies the code.

The character is defined by the Character descriptor command.

The command can also select a character for deletion from a font with the Font control command.

Send character descriptor and data

- **<ESC>(snW** *<descriptor and data>*

The command sends to the printer a character descriptor block followed by the data that makes up the character.

n is the total number of bytes, both descriptor and data, that follow the command.

n can be up to 32767. If it takes more than 32767 bytes to describe a character, split the description into blocks of 32767 bytes or less, and use the command to send each block separately. A character descriptor field specifies whether the data is the first block of a character description or a continuation.

The character descriptor for the first block of data describing a bitmap format character is as follows:

Byte	MSB	LSB
0	Format (4)	Continuation (0)
2	Descriptor size (14)	Class (1)
4	Orientation	Reserved (0)
6	Left offset	
8	Top offset	
10	Character width	
12	Character height	
14	Delta X	

The character descriptor for a continuation block is as follows:

Byte	MSB	LSB
0	Format (4)	Continuation (not 0)

The bytes following the header are the character's raster data.

Character descriptor attributes are represented using 5 different data types. The data type of each attribute field is indicated by the initials shown in the table.

B	Boolean
UB	Unsigned byte
SB	Signed byte
UI	Unsigned integer
SI	Signed integer

Format (UB) - 4 for bitmap fonts. If the setting does not match the descriptor format setting in the font descriptor, the character is not downloaded.

Continuation (B) - This field specifies whether the data block describes a new character (0) or is the continuation of a character description (1). If the continuation byte is 1, all subsequent bytes are interpreted as character data.

Descriptor size (UB) - 14 for bitmap fonts.

Class (UB) - 1 for bitmap fonts, 2 for compressed bitmap fonts. Ordinary bitmap fonts are sent as uncompressed raster data. Compressed bitmap font character data is encoded as follows.

The first byte of a line of data specifies the number of times that the line is repeated. The second byte indicates the number of successive white pixels at the start of the line. The third byte indicates the number of successive black pixels that follow the white pixels. The fourth byte indicates the number of successive white pixels following the black etc. Odd- and even-numbered bytes specify the number of successive black and white pixels making up the line. If there are more than 255 successive pixels of one color, this is represented by a byte set to 255, followed by a byte set to 0, followed by a byte

indicating the number of pixels of the color in excess of 255. The width of a line is determined by the Character width attribute. The number of pixels in each row must equal the Character width attribute setting.

Orientation (UB) - This setting determines the orientation of the character. 0 specifies portrait, 1 specifies landscape, 2 specifies reverse portrait and 3, reverse landscape.

If the setting does not match the Orientation attribute setting in the font descriptor block, the character is not downloaded.

Left offset (SI) - The horizontal distance between the character reference point (cursor position) and the leftmost character dot on the character cell grid in the physical page coordinate system. The value must be in the range -16384 to 16383.

Top offset (SI) - The distance between the character reference point (cursor position) and the topmost character dot on the character cell grid in the physical page coordinate system. The value must be in the range -16384 to 16383.

Character width (UI) - For bitmap fonts the attribute setting specifies the width of the character in the physical page orientation in dots. The value must be in the range 1 to 16383.

Character height (UI) - The setting specifies the height of the character in the physical page orientation in dots. The value must be in the range 1 to 16383.

Delta X (UI) - The setting specifies the horizontal distance the cursor moves from the character reference point after the character has been printed. The value is specified in units of 1/1200" and must be in the range 32768 to 32767.

The character data follows these header bytes. Bitmap characters are encoded as raster data. The data bytes build up an image of the character from left to right, from top to bottom. The Character width and Character height attribute settings determine the dimensions of the character cell grid.

4.11 Graphics

4.11.1 The print model

Using PCL commands you can control the way graphic elements combine on the page. The model used to describe the process details how a source image (an image to be drawn) is applied to a destination image (an image that has already been drawn).

The printer constructs each page in its memory before printing it out. Thus, at a given time, it will have received some text and graphics commands and will be about to receive more. The data received so far make up the destination image.

The source image may consist of a rectangle, a raster image or text. It consists of white areas and patterned areas. The pattern may be solid black, a shade of gray, or may itself be comprised of white and non-white areas. For example, the pattern may consist of grid of lines. You can specify the way in which you want the white and patterned areas of the whole source image and the white and non-white areas of the source image's pattern to interact with the destination image to produce the final result.

4.11.2 Source transparency

The source image can be either transparent or opaque.

When a transparent source image is superimposed on the destination image, the destination image is visible through the white (non-patterned) parts of the source image.

When an opaque source image is superimposed on the destination image, no part of the destination image is visible through the source image.

4.11.3 Pattern transparency

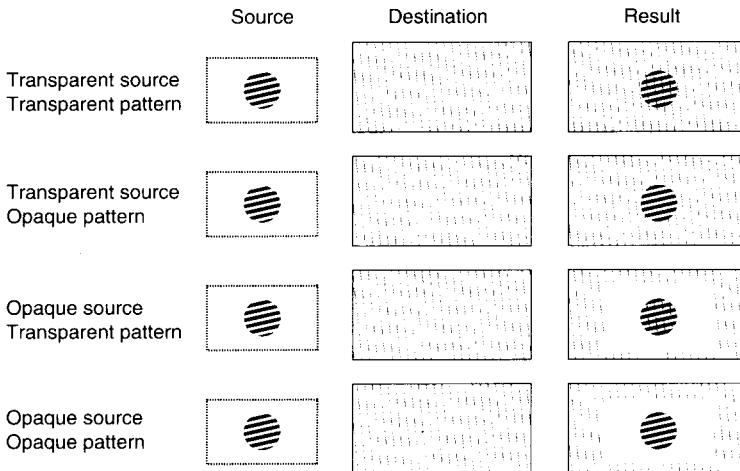
The source image's pattern can also be either transparent or opaque.

When a source image with a transparent pattern is superimposed on the destination image, the destination image is visible through any white parts of the patterned areas of the source image. This is true even if the source image itself is opaque.

When a source image with an opaque pattern is superimposed on the destination image, no part of the destination image is visible through the patterned areas of the source image.

Black-filled or gray scale patterns do not have any white areas, hence no part of the destination image is visible through the pattern, irrespective of the pattern transparency setting.

White-filled and cross-hatched patterns are comprised wholly or partly of white areas, however, a white filled rectangle constructed with the Draw filled rectangle command is always opaque.



Set source transparency - <ESC>*vnN

The command specifies the source image transparency.

$n = 0$ selects transparent mode.

$n = 1$ selects opaque mode.

In transparent mode the destination image is visible through the white (non-patterned) areas of the source image after the source image has been superimposed on it.

In opaque mode the destination image is not visible through the white (non-patterned) areas of the source image after the source image has been superimposed on it.

Set pattern transparency - <ESC>*vnO

The command specifies the pattern transparency.

$n = 0$ selects transparent mode.

$n = 1$ selects opaque mode.

In transparent mode the destination image is visible through the white parts of the patterned areas of the source image after the source image has been superimposed on it.

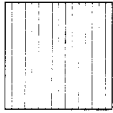
In opaque mode the destination image is not visible through the white parts of the patterned areas of the source image after the source image has been superimposed on it.

A white-filled rectangle drawn with the Draw filled rectangle command is always opaque, no matter what the pattern transparency setting.

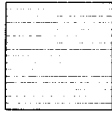
Set area fill identity - `<ESC>*cnG`

The command selects a cross-hatch or gray scale pattern which can then be selected with the Set pattern type command.

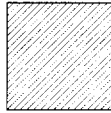
Select a cross-hatch pattern by setting n to the appropriate value (1 – 6).



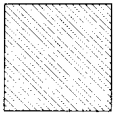
(#1)



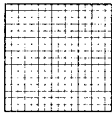
(#2)



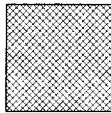
(#3)



(#4)



(#5)



(#6)

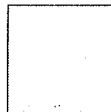
Select a shade of gray by setting n to the gray scale percentage you require (1 – 100). n selects the shade whose percentage range it falls in.



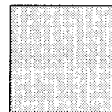
(1-2%)



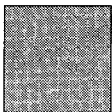
(3-10%)



(11-20%)



(21-35%)



(36-55%)



(56-80%)



(81-99%)



(100%)

Having selected a pattern or gray scale, you can enable it for printing with the Set pattern type command.

Set pattern type - <ESC>*vnT

The command selects a fill pattern type: black, white, gray scale or cross-hatch.

Text and graphics are printed with the selected fill.

$n = 0$ selects solid black.

$n = 1$ selects white.

$n = 2$ selects a gray scale. You must first have selected a gray scale percentage (0 – 100%) using the Set area fill identity command.

$n = 3$ selects a cross-hatch pattern. You must first have selected a cross-hatch pattern using the Set area fill identity command.

4.11.4 Rectangle graphics

Using the commands described in this section you can draw filled rectangles. The rectangles have no outline: they are simply blocks of a given shade or pattern.

When drawing a filled rectangle position the cursor at the point where the top left hand corner of the rectangle will be. Then specify the height and width of the rectangle using the Set rectangle height and Set rectangle width commands. You can now draw the rectangle with the Draw filled rectangle command.

After the rectangle has been drawn the cursor position is still the top left-hand corner of the rectangle.

If the current pattern is a cross-hatch pattern, the current pattern transparency setting determines whether the destination image is visible through the rectangle. The source transparency has no significance, since the source image consists of a patterned area without any accompanying white space.

A white-filled rectangle is always opaque, just like a black or gray scale filled rectangle, no matter what the current pattern transparency setting. A white rectangle appears simply as a solid white block.

Set rectangle width in dots - <ESC>*cnA

The command specifies the width in dots of a rectangle to be printed.

n sets the width in dots.

Set rectangle width in decipoints - **<ESC>*cnH**

The command specifies the width in decipoints of the rectangle to be printed.

n sets the width decipoints.

Set rectangle height in dots - **<ESC>*cnB**

The command specifies the height in dots of the rectangle to be printed.

n sets the height in dots.

Set rectangle height in decipoints - **<ESC>*cnV**

The command specifies the height in decipoints of the rectangle to be printed.

n sets the height in decipoints.

Draw filled rectangle - **<ESC>*cnP**

The command draws a rectangle filled with a pattern of the specified fill type.

n = 0 selects solid black as the fill.

n = 1 selects white as the fill.

n = 2 selects the gray scale selected by the Set area fill identity command as the fill.

n = 3 selects the cross-hatch pattern specified by the Set area fill identity command as the fill.

n = 5 selects the current pattern as specified by the most recent Set pattern type command and the Set Area Fill Identity command which preceded it.

A white-filled rectangle is always opaque. The destination image is not visible through it.

4.11.5 Raster graphics

A raster graphic consists of a matrix of white and black dots. The image is represented by a matrix of zeroes and ones which correspond to the white and black dots comprising the image. The areas of the image represented by ones (the non-white areas) are output using the current pattern when the image is printed.

The source transparency, pattern transparency and pattern settings affect raster images as described in the Print model.

Transmit raster data a line at a time using Send raster data commands. Precede the Send raster data commands with a Start raster transfer command and terminate the transmission with an End raster transfer command.

It is a good idea to define a rectangular raster area within which the image is to lie. Specify the height and width of the area with the Set raster area height and Set raster area width commands. A raster area is not strictly necessary, however, it often facilitates the transmission of images.

If you have defined a raster area you can use the Raster y-offset command to skip any all-white lines. The printer automatically prints the number of white lines which you specify. You can also omit trailing zeroes representing white space on the end of a line. The printer pads out the line with white space up to the edge of the raster area. The printer will also pad out the bottom of the raster area with white space, obviating the need to transmit trailing blank rows. Any raster data that would lie outside the raster area is not printed.

Raster images can consist of a lot of numerical data. There are a number of ways to compress the data using the Set compression mode command.

After the image has been printed, the cursor is positioned at the bottom right-hand corner of the raster area or, if no raster area was defined, at the end of the last transmitted row of data.

Set raster resolution - **<ESC>*tnR**

The command sets the raster image resolution in dots per inch.

$n = 75, 100, 150$ or 300 .

Use this command before you the Start Raster Graphics command, otherwise it does not take effect until after the next End Raster Graphics command.

Lower resolution images consume less printer memory.

The factory default raster resolution is 75 dots per inch.

Set raster image orientation - **<ESC>*rnF**

The command specifies the orientation in which a raster image will be printed with respect to the logical page orientation.

$n = 0$ causes the image to be printed in the current logical page orientation.

$n = 3$ causes the image to be printed in the current physical page orientation, irrespective of the logical page orientation.

Use this command before you the Start Raster Graphics command, otherwise it does not take effect until after the next End Raster Graphics command.

The factory default setting is the current physical page orientation.

Set raster area height - **<ESC>*rnT**

The command determines the height of the raster area.

n specifies the height in raster rows.

The height of single raster row is either 1/300", 1/150", 1/100" or 1/75" as determined by the current raster resolution setting.

A change in the raster resolution setting will change the physical height of the raster area.

Use this command before you use the Start Raster Graphics command, otherwise the setting does not take effect until after the next End Raster Graphics command.

Set raster area width - **<ESC>*rnS**

The command determines the width of the raster area.

n specifies the width in raster rows.

The width of a single raster dot is either 1/300", 1/150", 1/100" or 1/75" as determined by the current raster resolution setting.

A change in the raster resolution setting will change the physical width of the raster area.

Use this command before you use the Start Raster Graphics command, otherwise the setting does not take effect until after the next End Raster Graphics command.

Set raster y-offset - <ESC>*bnY

The command specifies how many rows should be skipped.

This command tells the printer to insert the specified number of white lines in the image.

n is in the range 0 – 32767

Use this command after a Start Raster Graphics command and before an End Raster Graphics command, otherwise it will have no effect.

Set compression mode - <ESC>*bnM

The command specifies the compression method that has been used to encode a raster image.

$n = 0$ specifies no compression.

$n = 1$ selects run-length encoding.

$n = 2$ selects tagged image file format (TIFF) encoding.

$n = 3$ selects delta row compression.

Run-length encoding

Data bytes are transmitted in pairs. The first byte of a pair specifies the number of times the second byte is repeated successively. The second byte is raster image data. If the first byte has the value x , the second byte is repeated $x+1$ times.

Tagged image file format

An image consists of groups of bytes, each group consisting of a control byte followed by one or more data bytes.

The control byte specifies how many data bytes follow and how they are to be interpreted.

If the two's complement value of the control byte is between -1 and -127 , the byte which follows is successively repeated. The number of times the data byte is repeated equals the absolute value of the control byte plus one: e.g. if the control byte's two's complement value is -31 (11100001), the data byte is repeated 32 times.

If the two's complement value of the control byte is between 0 and 127, the bytes which follow are normal uncompressed raster data. The number of data bytes is the value of the control byte plus one: e.g. if the control byte is 30, the following 31 bytes are unencoded raster data.

A control byte with two's complement value -128 (binary 1000000) is ignored and the byte which follows is interpreted as a control byte.

Delta row compression

An image is transmitted as a sequence of groups of bytes, each group consisting of a command byte followed by one or more data bytes. Byte groups specify a raster row by modifying the last transmitted row (the seed row).

The command byte identifies a sequence of bytes in the seed row that needs to be changed. The data bytes that follow replace the specified seed row sequence. The number of bytes to be changed equals the value held in the top 3 bits of the command byte plus 1. The position in the row of the first byte of the seed row sequence to be changed equals the value held in the lower 5 bits of the command byte plus 1. For example, if the command byte is 222 (binary 11011110), the 31st – 37th bytes of the seed row will be replaced by the 7 data bytes which follow.

If the lower 5 bits are 11111 (31 decimal), the following byte (all 8 bits) is added to 32 to calculate the total offset. If this offset byte equals 255, the next byte is also treated as a further offset value and is added to the offset total. This process continues until a byte whose value is less than 255 is encountered. This byte is treated as the final byte in the offset sum.

Each row is specified in terms of the preceding row. If there is more than one byte sequence in a raster row that must be changed, the second (and subsequent) offsets are counted from the first byte following the last byte that was changed.

When a complete row has been transmitted it becomes the seed row.

Start raster transfer - **<ESC>*rnA**

The command signals the start of a transmission of raster data to the printer.

$n = 0$ prints the image starting at the left edge of the logical page.

$n = 1$ prints the image starting from the current cursor position.

The transfer continues until an End raster transfer command or until a command other than Transfer raster data, Set compression mode, or Set raster y-offset is transmitted.

Transfer raster data - **<ESC>*bnW<data>**

The command transmits a row of raster data to the printer.

n specifies the number of bytes to be transmitted.

If more data is transmitted than will fitted onto one raster area row, the line is clipped.

End raster transfer - **<ESC>*rB**

The command signals the end of a transmission of raster data to the printer.

If a raster area has been defined, the cursor is positioned one raster dot below the raster area.

The delta row compression seed row is set to all zeroes.

Any raster settings made since the last Start raster transfer command now take effect.

4.12 Macros

A macro is a predefined series of PCL commands which can be downloaded to the printer and run automatically with a single command.

A typical macro application would be a set of commands to draw a company logo in a set position on a page.

Downloaded macros take up printer memory in the same way as downloaded fonts do. However, some macros are available on cartridge, allowing you to use macros without sacrificing memory.

Assign macros unique ID numbers with which they can then be referenced. Cartridge macros already have ID numbers assigned to them.

A macro cannot enter GL2 mode, change the size or location of the picture frame or change the GL2 plot size.

The <ESC> E Reset command cannot be used within a macro.

All macros in use at a given time must have a unique ID number. If a cartridge macro has the same ID number as a downloaded macro, the downloaded macro takes precedence: the cartridge macro cannot be accessed until you delete the downloaded macro. To avoid further conflict the deleted macro could be assigned a different ID number and redownloaded.

One macro can call another. Only two levels of nesting are allowed.

Macros can be either temporary or permanent.

A Reset deletes all temporary macros from the printer's memory.

4.12.1 Running Macros

Macros can be executed, called or enabled for overlay.

When a macro is executed, it uses the current modified print environment. Any changes it makes to the environment are permanent.

A called macro also uses the current modified print environment. However, changes are not retained when the macro has finished running.

A macro enabled for overlay automatically runs as the final operation every time a page is printed. Overlay macros use the macro overlay environment: a combination of the user default environment and the modified print environment. The macro overlay environment is only in effect while the macro is running.

The macro overlay environment consists of the user default environment settings for all features except those listed below, which retain their current modified print environment settings.

Page length	Paper source
Page size	Number of copies
Orientation	Cursor position stack
Registration	

See the Environments section for a description of the different printer environments.

4.12.2 Macro definitions

A macro definition consists of three macro commands: Macro ID, Macro Control (start macro definition) and Macro Control (end macro definition), and the PCL commands which the macro will perform.

The sequence of commands is as follows:

Macro ID command

Macro Control (start macro definition) command

PCL commands

Macro Control (end macro definition) command

The sequence of PCL commands may contain the Macro Control (execute macro) or Macro Control (call macro) command, invoking another macro. No other macro commands are allowed within the definition.

4.12.3 Macro commands

Macro ID - **<ESC>&fnY**

The command assigns an ID number to a macro that is to be downloaded, or identifies a macro in the printer's memory.

Before downloading a macro, use this command to assign an ID to it. If you use a number belonging to a macro already in printer memory, the new macro overwrites the existing macro.

When using the Macro control command to perform an operation on a macro already in the printer's memory, for example, making the macro permanent, use this command to select it.

The factory default macro ID number is 0.

Macro control - **<ESC>&fnX**

The command performs a specific action on one or all macros. When performing an action on a single macro first select the macro using the Macro ID command, then use this command to perform the appropriate action.

The start and end macro definition options apply to a macro to be downloaded, all other options apply to a macro (or all macros) in memory.

n specifies the operation to be performed as follows:

n = 0 starts macro definition

This option signals the start of a macro definition.

n = 1 ends macro definition

This option signals the end of a macro definition.

n = 2 executes a macro

Changes that the macro makes to the modified print environment, e.g. font selection, are retained after the macro has finished running.

n = 3 calls a macro.

Changes that the macro makes to the modified print environment are temporary and are not retained after the macro has finished running.

n = 4 enables a macro for overlay.

The macro is run as the final operation each time a page is printed.

$n = 5$ disables an overlaid macro.

$n = 6$ deletes all macros from the printer's memory.

$n = 7$ deletes all temporary macros from the printer's memory.

All macros are temporary unless they have been made permanent with the Make macro permanent command.

$n = 8$ deletes a macro from the printer's memory.

$n = 9$ makes a macro temporary.

Temporary macros are not retained after a Reset.

This command only applies to downloaded macros.

$n = 10$ makes a macro permanent.

Permanent macros are retained after a Reset.

This command only applies to downloaded macros.

MEMO

Vector graphics

CHAPTER 5

Printer Control Language does not contain any vector graphics commands of its own. However, PCL commands allow you to switch from PCL mode into GL2 vector graphics mode and use the powerful vector drawing commands of the GL2 graphics language. Originally devised for pen-plotters, GL2 is a powerful drawing tool that precisely defines images with reference to a grid coordinate system, and includes commands to draw lines and shapes, apply shading patterns and fills to shapes, and handle text. In GL2 mode, you can draw images to appear on the same page as text and graphics generated in PCL mode. On completion of vector graphics operations you can then switch back into PCL mode.

5.1 GL2 concepts

The **picture frame** is the rectangular area of the page in which GL2 graphic images can appear. The default picture frame for a given page size is the same as the default text area. Picture frame dimensions for the different page sizes are given in tables on pages 45 and 46 of Chapter 4. Before entering GL2 mode you can specify the size and location of the picture frame using PCL commands. Specify the size and location in terms of the **anchor point** (the top left-hand corner of the picture frame) and height and width.

The GL2 coordinate system has its default origin in the bottom left-hand corner of the picture frame. Hence, in contrast to the PCL coordinate system, the x-coordinate value of the pen position increases as it moves up the page. The default units, known as **plotter units**, are 1/1016" (0.025mm) on both axes. Alternatively, you can specify a more convenient unit size using the **SC** command. These custom units are known as **user units** and you can define x- and y-axis user units of different sizes. The printer automatically converts user units to plotter units at print time. The units in use at a particular time (plotter or user) are called the **current units**.

GL2 drawing commands can be described using the notion of an imaginary pen, which can be either “up” or “down”. When the pen is “down”, a GL2 plotting command, e.g. the command to move the pen to a specified coordinate location, will draw a line on the page. When the pen is “up”, the same command will not draw a line. Thus, when the pen is in the “up” state, you can position it without marking the page.

Two GL2 commands, the **PU** and **PD** commands, allow you to set the pen to be “up” or “down” before you issue a command to move the pen.

Some GL2 commands always draw on the page, irrespective of the current pen state (“up” or “down”). Hence you do not need to precede these commands with a “pen down” instruction.

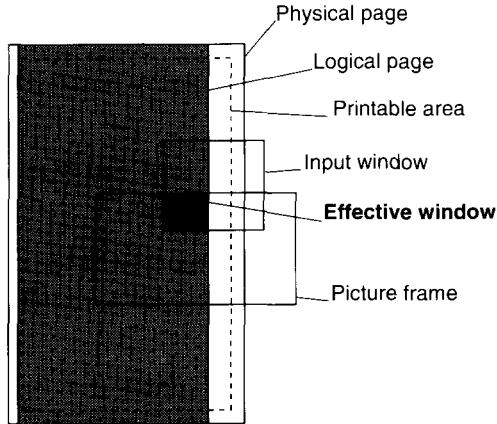
GL2 makes use of two reference **scaling points**, **P1** and **P2**, whose default positions are the bottom left- and top right-hand corners of the picture frame. The **IP** and **IR** commands alter the positions of P1 and P2.

By altering the relative positions of P1 and P2 and scaling the user units with the **SC** command, you can dynamically rotate, reflect, skew and scale images.

Pen movement is either absolute or relative. Absolute movement coordinates are specified with reference to P1, the origin of the coordinate system. Relative movement coordinates are specified relative to the current pen position. Coordinates are always expressed in the current units.

The **input window**, also known as the soft clip limits, is a user-defined rectangular window, outside which GL2 graphic output cannot appear. In this respect, it is like the picture frame, however, the difference is that you can define an input window in GL2 mode, whereas you can only modify the picture frame in PCL mode.

The area formed by the overlapping of the printable area, logical page, picture frame and input window is called the **effective window**. Only GL2 output that lies within the effective window will appear on the printed page. The printable area and the logical page are defined in “The page” in Chapter 4.



5.2 Managing GL2 mode from PCL mode

The PCL commands in this section allow you to switch back and forth between PCL and GL2 mode, to determine the size and position of the picture frame, and to scale a GL2 image to fit the area you require. In some instances scaling is performed automatically.

If you do not specify picture frame position, height and width, the default picture frame is used. Default picture frame sizes are shown on pages 45 and 46 of Chapter 4.

The `<ESC>*c0T` command makes the current PCL cursor position the picture frame anchor point. The `<ESC>*cnX` command sets the picture frame height and the `<ESC>*cnY` command sets the width.

5.2.1 *Scaling an imported image*

An imported image will automatically be scaled to fit the picture frame if the image is **page-size independent**. An image is page-size independent if the GL2 code that defines it meets the following conditions:

- 1) User units are used exclusively: i.e. a **SC** scaling command must precede all plotting commands and any others that take current unit parameters.
- 2) All pen movement is relative: i.e. only relative plotting commands may be used.
- 3) All measurements are relative: character size, line type pattern length and pen width must be specified as relative distances.
- 4) No commands that imply absolute pen movement are used: i.e. any command, such as **IP** or **PA**, that has a relative equivalent (**IR** and **PR**) may not be used, even without parameters.

If the image you want to import does not satisfy the above conditions, use the `<ESC>*cnK` and `<ESC>*cnL` plot size commands to specify the horizontal and vertical dimensions of the original image. The printer will then scale the x- and y-axis dimensions so that the image fits the picture frame exactly. If you fail to specify the original height and width, the imported image is printed actual size, and may be clipped as a result.

If the image you want to import is the same size as the picture frame, you do not need to use the plot size commands.

5.2.2 Set-up commands for GL2 mode

Set picture frame anchor point - <ESC>*c0T

The command sets the picture frame anchor point to the current cursor position.

First position the cursor using the appropriate PCL commands, then use the command to make the current position the anchor point.

The command has the following effects on the GL2 vector graphics state.

The scaling points, P1 and P2, are set to their respective default positions, the bottom left-hand and top right-hand corners of the picture frame.

The input window is set to its default position, the picture frame limits.

The polygon buffer is emptied.

The GL2 cursor is set to its default position, P1.

Set picture frame vertical size - <ESC>*cnY

The command sets the height of the picture frame in decipoints (1/720").

The command has the same effects on the GL2 vector graphics state as the Set picture frame anchor point command.

Set picture frame horizontal size - <ESC>*cnX

The command sets the width of the picture frame in decipoints (1/720").

The command has the same effects on the GL2 vector graphics state as the Set picture frame anchor point command.

Specify vertical plot size - <ESC>*cnL

The command specifies the height in inches of an imported image.

Only use this command if importing an existing image.

n must be between 0 and 32767 and is accurate to four decimal places.

The imported graphic is scaled vertically to fit the height of the picture frame.

Specify horizontal plot size - **<ESC>*cnK**

The command specifies the width in inches of an imported image.

Only use this command if importing an existing image.

n must be between 0 and 32767 and is accurate to four decimal places.

The imported graphic is scaled horizontally to fit the width of the picture frame.

Enter GL2 mode - **<ESC>%nB**

The command switches the printer from PCL mode into GL2 graphics mode.

n = 0 positions the pen at the previous GL2 pen position. If this is the first switch into GL2 mode since a Reset or since the printer was switched on, the pen is positioned at the lower left-hand corner of the picture frame.

n = 1 positions the pen at the current PCL cursor position.

All commands that follow are interpreted as GL2 vector graphics commands until the printer receives an **<ESC> E** Reset, **<ESC>%nA** Enter PCL mode or **<ESC> [E n** Change emulation mode command, or until a control panel reset is performed.

When the printer is first switched into GL2 graphics mode after switch-on or a reset, all GL2 settings have their default values. These are listed with the description of the **IN** command on page 128.

Enter PCL mode - **<ESC>%nA**

The command switches the printer from GL2 graphics mode back into PCL mode.

n = 0 positions the PCL cursor at the position it was in when the printer entered GL2 graphics mode.

n = 1 positions the PCL cursor at the current GL2 pen position.

All commands that follow are interpreted as PCL commands.

5.3 GL2 syntax

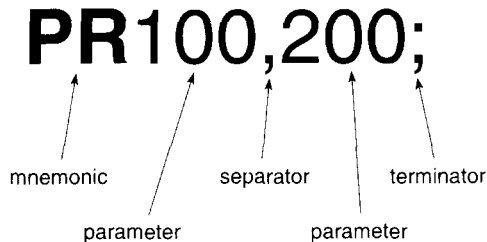
GL2 commands can consist of up to four components: a two-letter mnemonic, parameters, separator characters and a terminating character.

The **mnemonic** is an abbreviation for the name of the command and helps remind you of the command's purpose.

Almost all GL2 commands can have one or more numerical **parameters**.

Parameters must be delimited from one another by **separators**: valid separators are a space, a comma, and the + and – signs.

A **terminator** is not usually necessary, though a semi-colon may be used to terminate any command explicitly. The final command used before quitting GL2 mode must be terminated by a semi-colon, as must the **PE** Polyline encode command. In all other instances, however, a command may be implicitly terminated by the first letter of the mnemonic of the command which follows.



In this chapter command mnemonics are shown in bold upright type and parameters are shown in italics. Optional elements, i.e. optional parameters and the terminator, are enclosed in square brackets. Parameters that may optionally be repeated a number of times are followed by dots "...".

Parameters fall into the following five format categories:

Integer - any integer from -2^{30} to $2^{30}-1$. Real numbers are rounded to the nearest integer.

Clamped integer - any integer from -32768 (-2^{15}) to 32767 ($2^{15}-1$). Numbers outside the range are clamped to the nearest integer inside the range, e.g. 33000 would be clamped to 32767. Real numbers are rounded to the nearest integer inside the range.

Real number - any real number whose integer component is in the range -2^{30} to $2^{30}-1$. Numbers that are out of range cause the command to be ignored. If the number has no fractional component the decimal point may be omitted. Real numbers are accurate to six significant digits.

Clamped real number - any real number whose integer component is in the range -32768 (-2^{15}) to 32767 ($2^{15}-1$). Numbers outside this range are clamped to the nearest real number inside the range. If the number has no fractional component the decimal point may be omitted.

Label - any text string.

5.4 Programming with GL2

Send GL2 commands to the printer using the same programming language commands that are used for printing ordinary text. In BASIC this is the LPRINT command. The two example programs which follow, in BASIC and C, demonstrate how an IBM PC might send the printer GL2 commands.

5.4.1 BASIC program

```
100 WIDTH "LPT1:",255
110 LPRINT CHR$(27);"E"; :REM ESC E - Reset
120 LPRINT CHR$(27);"%0B"; :REM ESC %0B - Enter GL2
graphics mode
130 LPRINT "IN"; :REM Initialize GL2 graphics variables
140 LPRINT "IP0,0,6096,6096"; :REM Initialize P1 and P2
150 LPRINT "SC0,24,0,24"; :REM Set user units to 1/4"
160 LPRINT "SP1PA6,6"; :REM Select black pen & move to
(6,6)
170 LPRINT "PUEA18,18"; :REM Draw square
180 LPRINT CHR$(27);"%0A"; :REM Revert to PCL mode with
cursor in its pre-GL2 mode position
190 LPRINT CHR$(27);"E"; :REM Reset printer & eject page
```

5.4.2 C program

```
#include <stdio.h>
main()
{
FILE *prn; /* Access printer port */
prn = fopen("PRN","wb");
fprintf(prn,"\33E"); /* ESC E - Reset */
fprintf(prn,"\33%0B"); /* ESC%0B - Enter GL2 graphics
mode */
fprintf(prn,"IN"); /* Initialize GL2 graphics variables
*/
fprintf(prn,"IP0,0,6096,6096"); /* Initialize P1 and P2
*/
fprintf(prn,"SC0,24,0,24"); /* Set user units to 1/4" */
fprintf(prn,"SP1PA6,6"); /* Select black pen & move to
(6,6) */
fprintf(prn,"PUEA18,18"); /* Draw square */
fprintf(prn,"\33%0A"); /* Revert to PCL mode with cursor
in its pre-GL2 mode position */
fprintf(prn,"\33E /* Reset printer & eject page */
}
}
```

5.4.3 Automatic “Pen down”

Some drawing commands draw on the page irrespective of the current pen state (up or down). It is advisable to immediately precede these commands with a Pen up command (**PU**). This precludes the possibility of unwanted dots on the final output.

5.4.4 Lost mode

If a command parameter value causes overflow, the printer can lose track of the current pen position. The printer then enters “lost” mode. In “lost” mode the printer raises the pen and ignores the following commands: **AA, AR, AT, CL, CP, EA, ER, EW, LB, PE, PM0, PR, RA, RR, RT** and **WG**.

The printer can still perform the following commands: **AC, AD, CF, CO, DF, DI, DR, DT, DV, ES, FT, IN, IP, IR, IW, LA, LO, LT, PA, PD, PG, PM1, PM2, PU, PW, RF, RO, RP, SA, SB, SC, SD, SI, SL, SM, SP, SR, SS, TD, UL** and **WU**.

You can get out of “lost” mode by using the **IN, PA, PG** or **RP** commands with valid parameters, or the **PU** or **PD** commands with valid absolute parameters. The **PD** (Pen down) command draws a line from the last known pen position to the first point in its parameter list. If the **PA** command is used to clear “lost” mode, the pen stays in the “up” position until the printer receives a **PD** (Pen down) command.

5.5 GL2 graphics commands

GL2 graphics commands are classified in five groups. Each group consists of commands whose uses are related. The five command groups are as follows:

Configuration and status group
Vector group
Polygon group
Line and fill attributes group
Character group

5.5.1 Configuration and status group

The commands that make up the configuration and status group are as follows:

Initialize	IN	Rotate coordinate system	RO
Default values	DF	Input window	IW
Input scaling points	IP	Advance full page	PG
Input relative scaling points	IR	Replot	RP
Scale	SC		

These commands set up an environment in which the remaining GL2 commands can operate.

IN and **DF** set GL2 variables to default values.

IP and **IR** position the scaling points P1 and P2, and hence determine image size and rotation. They can be used to effect a variety of image transpositions and duplications.

SC sets the size of the user units and can thus be used to resize or distort an image.

RO rotates the coordinate system and can thus also be used to rotate images.

IW defines a window outside of which no GL2 graphics or text can appear.

Initialize

IN [:]

The command initializes all GL2 graphics mode variable settings to their default values.

The table shows the default GL2 graphics mode settings and the command equivalents for resetting them.

It is a good idea to use **IN:** each time you switch the printer into GL2 mode, unless you specifically want to retain some variable settings from the last time that GL2 mode was used.

Function	Setting	Equivalent instruction
Plotting mode	Absolute	PA;
Window	Current picture frame	IW;
Anchor corner	Bottom left corner of picture frame	AC;
Scaling	No scaling; plotter units in use	SC;
Scaling points	Picture frame bottom left and top right corners	IP;
Rotation	0 degrees	RO;
Line type	Solid	LT;
Line pattern length	4% of distance from P1 to P2	LT;
Line attributes	Butt caps, mitered joins, miter limit=5	LA;
User-defined line type	All line types set to default	UL;
Pen	White pen selected	SP;
Pen position	Lower left corner of picture frame	PA0,0;
Pen state	Up	PU;
Pen width type	Metric	WU;
Pen width	0.35mm	PW;
Fill type	Type 1 - solid	FT;
Raster fill	Solid black	RF;
Transparency mode	On	TR1;
Screened vectors	No screening	SV;
Polygon mode	Polygon buffer empty	PM0PM2;
Standard font	Stick font	SD;
Alternate font	Stick font	AD;
Character set	Standard font selected	SS;
Character slant	0 degrees	SL0;
Character fill	Solid	CF;
Character direction	Horizontal	DI1,0;
Character size transformation	Off	SI;
Symbol mode	Off	SM;
Scalable or bitmap fonts	Scalable fonts only	SB0;
Label terminator	CHR\$(3), non-printing.	DTCHR\$(3);
Label origin	Current pen position	LO1;
Text path	Left to right with normal line feed.	DV;
Extra space	No extra space	ES;
Transparent data	Normal printing mode	TD;

Default values

DF [;]

The command sets all GL2 graphics mode variable settings to their factory default values, except for the following:

The position of P1 and P2, the GL2 coordinate system rotation, and the current pen position, pen state (up or down), pen number, pen width and width unit.

The printer sets the carriage return point for labelling to the current pen position. See the Character group section on page 171 for a description of labelling.

This command allows you to reset GL2 variables without affecting the current plotting characteristics.

The table below shows the default GL2 graphics mode settings which **DF**; resets and the command equivalents for resetting them.

Function	Setting	Equivalent instruction
Plotting mode	Absolute	PA ;
Window	Current picture frame	IW ;
Anchor corner	Bottom left corner of picture frame	AC ;
Scaling	No scaling: plotter units in use	SC ;
Line type	Solid	LT ;
Line pattern length	4% of distance from P1 to P2	LT ;
Line attributes	Butt caps, mitered joins, miter limit=5	LA ;
User-defined line type	All line types set to default	UL ;
Fill type	Type 1 - solid	FT ;
Raster fill	Solid black	RF ;
Transparency mode	On	TR1 ;
Screened vectors	No screening	SV ;
Polygon mode	Polygon buffer empty	PM0PM2 ;
Standard font	Stick font	SD ;
Alternate font	Stick font	AD ;
Character set	Standard font selected	SS ;
Character slant	0 degrees	SL0 ;
Character fill	Solid	CF ;
Character direction	Horizontal	DI1.0 ;
Character size transformation	Off	SI ;
Symbol mode	Off	SM ;
Scalable or bitmap fonts	Scalable fonts only	SB0 ;
Label terminator	CHRS(3), non-printing.	DTCHRS(3) ;
Label origin	Current pen position	LO1 ;
Text path	Left to right with normal line feed.	DV ;
Extra space	No extra space	ES ;
Transparent data	Normal printing mode	TD ;

Input scaling points

IP [*P1x*, *P1y* [, *P2x*, *P2y*]] [;]

P1x: x-coordinate of P1

P1y: y-coordinate of P1

P2x: x-coordinate of P2

P2y: y-coordinate of P2

The command defines the positions of P1 and P2 in absolute plotter units relative to the lower left-hand corner of the picture frame.

Plotter units are 1/1016" and coordinate values are integers.

The next **SC** command received by the printer assigns user coordinate values to P1 and P2. This effectively sets the size of the user units.

If you omit the parameters, the command sets the scaling points to their default positions, the lower left- and upper right-hand corners of the PCL picture frame in the current GL2 coordinate system orientation. See the Rotate coordinate system command on page 136 for a description of how to rotate the coordinate system.

If you omit the P2 parameters, P2 is repositioned so that it stays in the same position relative to P1. Hence if you want to plot the same image several times in different positions, simply move P1 and redraw the image.

P1 and P2 may be positioned anywhere, as long as the specified coordinates are inside the integer range. However, only the parts of an image that lie inside the effective window will appear on the final output.

P1x must be set to a different value from *P2x*, and *P1y* must be set to a different value from *P2y*. If *P1x* and *P2x*, or *P1y* and *P2y*, are set to equal values, the P2 coordinate is set to be 1 plotter unit greater than the corresponding P1 coordinate.

The scaling point settings remain in effect until the printer receives another **IP** command, or an **IR** or **IN** command.

Input relative scaling points

IR [*P1x*, *P1y* [, *P2x*, *P2y*]] [;]

P1x: x-coordinate of P1

P1y: y-coordinate of P1

P2x: x-coordinate of P2

P2y: y-coordinate of P2

The command defines the positions of P1 and P2 as a percentage of the width and height of the picture frame.

Coordinate values are clamped real numbers.

The next **SC** command received by the printer assigns user coordinate values to P1 and P2. This effectively sets the size of the user units.

If you omit the parameters, the command sets the scaling points to their default positions, the lower left- and upper right-hand corners of the PCL picture frame in the current GL2 coordinate system orientation. See the Rotate coordinate system command on page 136 for a description of how to rotate the coordinate system.

If you change the size of the PCL picture frame, P1 and P2 are repositioned so that their relative distances from each corner of the picture frame remain the same.

The plotter unit coordinates of the scaling points are stored in the printer. If you subsequently change the orientation of the coordinate system using the Rotate coordinate system command, P1 and P2 are repositioned so that they have the same plotter unit coordinates in the new orientation.

If you omit the P2 parameters, P2 is repositioned so that it stays in the same position relative to P1. Hence if you want to plot the same image several times in different positions, simply move P1 and redraw the image.

P1 and P2 may be positioned anywhere, as long as the specified coordinates are inside the real number range. However, only the parts of an image that lie inside the effective window will appear on the final output.

The scaling point settings remain in effect until the printer receives another **IR** command, an **IP** command or an **IN** command.

Scale

SC [*Xmin*, *Xmax*, *Ymin*, *Ymax* [, *type* [, *left*, *bottom*]]] [;] (Types 0 & 1)

SC [*Xmin*, *Xfactor*, *Ymin*, *Yfactor*, *type*] [;] (Type 2)

Xmin: x-coordinate of P1

Xmax: x-coordinate of P2

Ymin: y-coordinate of P1

Ymax: y-coordinate of P2

type: scaling type

left: percentage of unused space to left of scaling area

bottom: percentage of unused space below scaling area

Xfactor: ratio of plotter units to user units on x-axis

Yfactor: ratio of plotter units to user units on y-axis

The command assigns user unit coordinates to P1 and P2, and makes user units the current units.

In effect, this command sets the size of the user units, which are calculated from the positions of P1 and P2.

Coordinates can now be specified in user units: the printer interprets coordinate parameters with reference to the positions of P1 and P2.

There are three different types of scaling: anisotropic, isotropic and point factor.

P1 and P2 are not graphic limits. You can print an image that lies wholly or partly outside the P1-P2 rectangle, so long as it lies within the effective window.

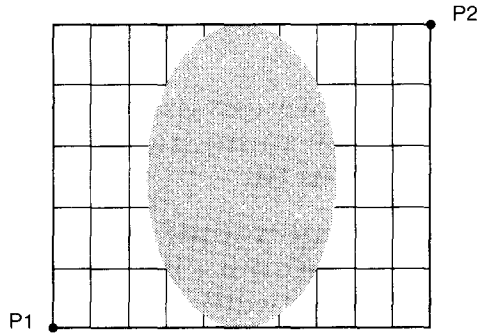
P1 does not have to be (0,0). Both the unit-size and origin can be selected to fit the requirements of the task at hand.

The order in which the coordinate parameters are specified for the **SC** command differs from the order other commands use: the two x-coordinates are specified first, then the two y-coordinates.

type = 0, 1 or 2. 0 selects anisotropic scaling, 1 selects isotropic scaling and 2 selects point factor scaling.

In anisotropic and isotropic scaling *Xmin* must be different from *Xmax*, and *Ymin* must be different from *Ymax*.

Anisotropic scaling, the default, allows x-axis and y-axis units of different sizes. As a result, the rectangle defined by $Xmin$, $Xmax$, $Ymin$, and $Ymax$ occupies the entire area defined by P1 and P2.

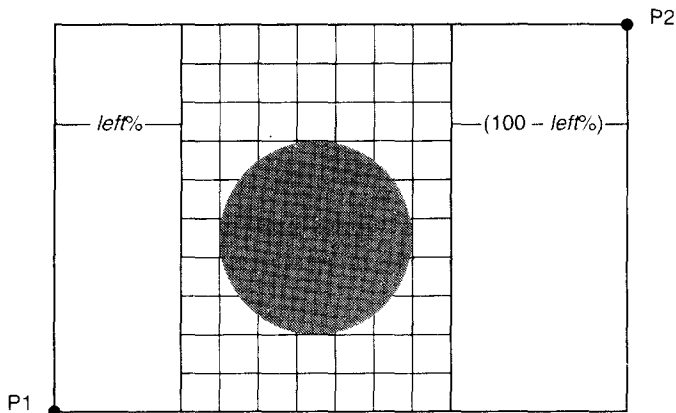


Anisotropic units need not be square, so for example, the **CI** command to draw a circle may be used to generate an ellipse.

The *left* and *bottom* parameters are not used in anisotropic scaling. If they are specified, they will be ignored.

Isotropic scaling forces x-axis and y-axis units to be the same size. As a result, the rectangle defined by $Xmin$, $Xmax$, $Ymin$, and $Ymax$ (the isotropic area) may not occupy the entire area defined by P1 and P2.

If the isotropic area does not fit exactly, it is sized so that either its height or its width matches that of the P1/P2 rectangle, and so that it fits entirely within the rectangle. This results in unused space either above and below, or to the sides of the isotropic area.



Isotropic units are always square. So for example, when the **CI** command is used, a circle is always drawn.

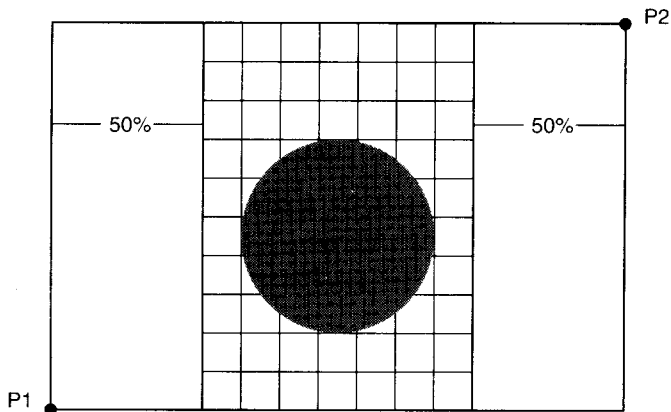
You can precisely position the isotropic area within the P1-P2 rectangle by specifying the percentage of unused space that should lie below, or to the left of, the isotropic area.

left determines the percentage of unused space to lie to the left of the isotropic area, if the width of the isotropic area is less than the width of the P1/P2 rectangle. *left* is in the range 0 to 100.

bottom determines the percentage of unused space to lie below the isotropic area, if the height of the isotropic area is less than the height of the P1/P2 rectangle. *bottom* is in the range 0 to 100.

Specify either both *left* and *bottom* parameters, or neither. Although only one of them will apply, both must be supplied.

If you omit the *left* and *bottom* parameters, the isotropic area is centered within the P1/P2 rectangle.



Point factor scaling specifies the number of plotter units in each user unit and assigns user unit coordinates to P1.

Xfactor specifies the number of plotter units in one x-axis user unit, and *Yfactor* the number of plotter units in a y-axis user unit. *Xfactor* and *Yfactor* are both integers.

By varying the parameters supplied to successive **SC** commands you can manipulate images in a number of ways, for example, you can invert images and create mirror images.

To invert an image set Y_{min} to be greater than Y_{max} . To generate the mirror image of an image, set X_{min} to be greater than X_{max} .

Because the **SC** command sets the size of user x- and y-units in terms of the scaling points, changes to the relative positions of P1 and P2 will cause the size of one or both of the user units to change as well.

Possible parameter errors and the action the printer takes in each case are as follows:

Condition	Printer's response
Types 0, 1 & 2: No parameters	Ignores command, turns scaling off
Types 0 & 1: Fewer than 4 parameters	Ignores command
Types 0 & 1: 6 parameters	Ignores command
Types 0 & 1: More than 7 parameters	Executes command using first 7 parameters
Types 0 & 1: $X_{min} = X_{max}$	Ignores command
Types 0 & 1: $Y_{min} = Y_{max}$	Ignores command
Type 2: Fewer than 5 parameters	Ignores command
Type 2: More than 5 parameters	Ignores command
Type 2: More than 7 parameters	Ignores command
Type 2: $X_{factor} = 0$ or $Y_{factor} = 0$	Ignores command

The **SC** command with no parameters makes plotter units the current units.

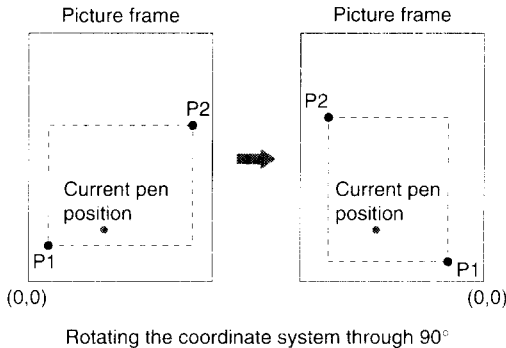
An **SC** command remains in effect until the printer receives another **SC** command or a **DF** or **IN** command.

Rotate coordinate system

RO [*angle*] [;]

The command sets the orientation of the GL2 coordinate system relative to the orientation of the picture frame.

angle, which can be 0, 90, 180, or 270, specifies the angle of rotation counter-clockwise from the default orientation, in which the origin of the GL2 coordinate system corresponds to the bottom left-hand corner of the picture frame.



The plotter unit origin, (0,0), is set to one of the four corners of the picture frame, according to the specified rotation.

The current pen position does not change: the pen's current coordinates are changed to reflect the new orientation.

The positions of P1 and P2 move with the coordinate system so that they retain the same coordinates. However, this may place either or both of them outside the picture frame. To reposition P1 and P2 at the lower left- and upper right-hand corners of the picture frame in the new orientation, use the **IP;** command.

The contents of the polygon buffer are rotated.

An input window will be rotated with the coordinate system. However, this may place part of the window outside the picture frame. The input window will then be clipped to the overlap of the picture frame, the logical page and the printable area. To reset the input window to the picture frame limits, use the **IW;** command.

The command with no parameter sets the rotation of the coordinate system to 0 degrees.

The command remains in effect until the printer receives another **RO** command or an **IN;** command.

Input window

IW [*X1*, *Y1*, *X2*, *Y2*] [;]

X1: input window bottom left corner x-coordinate

Y1: input window bottom left corner y-coordinate

X2: input window upper right corner x-coordinate

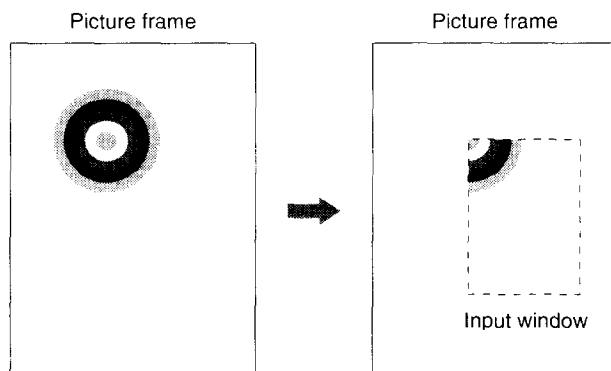
Y2: input window upper right corner y-coordinate

The command defines an input window, a rectangular area on the page, outside which no printed output can appear. Only GL2 graphics output that lies within the input window will appear on the printed page.

The input window is also known as the soft clip limits.

Coordinates are specified in current units and are integers.

If the current units are user units when the input window is defined, subsequent **IP** or **IR** commands will move the window on the page, so that the user coordinate values of the window's corners remain the same. However, a subsequent **SC** command fixes the input window on the page: its position is then unaffected by any further **IP** or **IR** commands.



An input window can be used to mask out portions of an image

The corners of the window can be set to lie outside the picture frame. However, only the parts of an image which fall within the effective window will be printed.

The command with no parameters sets the input window to the picture frame limits.

The command remains in effect until the printer receives another **IW** command or a **DF**; or **IN**; command.

Advance full page

PG [:]

The command clears “lost” mode but is otherwise ignored by the printer.

Use the PCL Form feed control code to eject a page. This is described in Chapter 4 on page 68.

A Form feed does not affect the GL2 pen position.

Replot

RP [:]

The command clears “lost” mode but is otherwise ignored by the printer.

Use the PCL Select number of copies command to print multiple copies of a graphics plot. This is described in Chapter 4 on page 52.

5.5.2 Vector group

The commands that make up the vector group are as follows:

Pen up	PU	Draw circle	CI
Pen down	PD	Draw absolute arc	AA
Plot absolute	PA	Draw absolute three point arc	AT
Plot relative	PR	Draw relative arc	AR
Polyline encoded	PE	Draw relative three point arc	RT

The commands in this group control the pen state (up or down), position the pen and draw lines. **PU** (Pen up) and **PD** (Pen down) set the pen state, determining whether plotting commands such as **PA** (Plot absolute) and **PR** (Plot relative) plot lines or simply move the pen without drawing. **PE** (Polyline encoded) combines a sequence of vector group commands into a single command. **CI** (Draw circle) and the arc drawing commands add the facility to draw circles, ellipses and curves.

Pen up

PU [*X*, *Y* [,...]] [;]

X: x-coordinate of destination point

Y: y-coordinate of destination point

The command raises the pen and moves in turn to each of the destination points specified.

If no parameters are supplied, the command raises the pen without moving it.

Coordinates are in current units, are real numbers, and can be either relative or absolute. If a **PA** (Plot absolute) command was used most recently, coordinates are absolute. If **PR** (Plot relative) was used most recently, coordinates are relative. If neither **PA** or **PR** has been used, coordinates are absolute.

There is no limit on the number of coordinate pairs you can specify.

If an odd number of coordinates is specified, the final coordinate is ignored.

In symbol mode the selected symbol is drawn at each point in the list. For a description of symbol mode refer to the **SM** command on page 166.

In polygon mode the destination points are stored in the polygon buffer and used when an Edge polygon or Fill polygon command is executed. For a description of polygon mode refer to the **PM** command on page 149.

Pen down

PD [*X*, *Y* [,...]] [;]

X: x-coordinate of destination point

Y: y-coordinate of destination point

The command lowers the pen and draws a line from the current pen position to the first destination point, and then successively from each destination point to the next.

If no parameters are supplied, the command lowers the pen without moving it.

In all other respects the command is the same as the **PU** (pen up) command.

Plot absolute

PA [*X*, *Y* [,...]] [;]

X: x-coordinate of destination point

Y: y-coordinate of destination point

The command moves the pen to each of the destination points in turn, and sets the plotting mode to absolute plotting.

If no parameters are supplied, the command simply makes absolute plotting the current plotting mode.

The parameters of commands which follow are treated as absolute coordinates.

Coordinates are in current units and are real numbers.

If the pen is “down”, a line is drawn from the current pen position to the first specified position, and then between each successive pair of points in the parameter list.

If an odd number of coordinates is specified, the final coordinate is ignored.

In symbol mode the selected symbol is drawn at each point in the list. For a description of symbol mode refer to the **SM** command on page 166.

In polygon mode the destination points are stored in the polygon buffer and used when an Edge polygon or Fill polygon command is executed. For a description of polygon mode refer to the **PM** command on page 149.

Plot relative

PR [*X* , *Y* [,...]] [;]

X: x-coordinate of destination point

Y: y-coordinate of destination point

The command moves the pen to each of the destination points in turn, and sets the plotting mode to relative plotting. The coordinates of the first point in the list are interpreted relative to the current pen position; the coordinates of each subsequent point are interpreted relative to the preceding point.

If no parameters are supplied, the command simply makes relative plotting the current plotting mode.

The parameters of commands which follow are treated as relative coordinates.

Coordinates are in current units and are real numbers.

If the pen is “down”, a line is drawn from the current pen position to the first specified position, and then between each successive pair of points in the parameter list.

If an odd number of coordinates is specified, the final coordinate is ignored.

If the command moves the pen to a position whose absolute plotter unit coordinates are outside integer range, all following commands are ignored until a **PA** or **PE** command clears “lost” mode.

In symbol mode the selected symbol is drawn at each point in the list. For a description of symbol mode refer to the **SM** command on page 166.

In polygon mode the destination points are stored in the polygon buffer and used when an Edge polygon or Fill polygon command is executed. For a description of polygon mode refer to the **PM** command on page 149.

Polyline encoded

PE [[*flag*] [*value*] | *Xi Yi* ... [*flag*] [*value*] | *Xi Yi*]] ;

flag: a command, number type, plotting mode or data mode

value: parameter data for preceding flag

Xi: x-coordinate of destination point

Yi: y-coordinate of destination point

The command incorporates a sequence of **PA**, **PR**, **PU**, **PD** and **SP** commands into a coded form, resulting in smaller graphics files and reducing data transmission times.

Flags within the parameter list determine the way in which data is interpreted.

The command draws lines to all coordinate points in the list except those preceded by a "<" (pen up) flag.

All coordinates are relative except those preceded by the absolute mode flag (=).

The command must be explicitly terminated by a semi-colon.

Flags are as follows:

:	Select pen	The number that follows selects the pen. A PE command without this flag uses the current pen.
<	Pen up	The pen is raised, moved to the point specified by the coordinates that follow, and lowered. Lines are drawn to all points not preceded by this flag.
>	Fractional data	The number that follows specifies the number of fractional binary bits in the data.
=	Absolute plotting	The coordinate pair following the flag are interpreted as absolute coordinates. Any coordinates not preceded by this flag are relative.
7	7-bit mode	All coordinate values following the flag are interpreted as 7-bit (base 32) values.

Send flags to the printer as ASCII character codes. The MSB of the code is ignored, so '<', the fractional data flag, can be either 62 or 190.

The ':' select pen flag has no effect in polygon mode.

Values and coordinates are encoded in base 64 or base 32 using ASCII character codes. Codes 0 – 62, 127 – 190 and 255 are not used. (Number encoding schemes are described below).

Valid ranges for values and coordinates are as follows:

Pen number	0 (white) or 1 (black)
Number of fractional binary bits	0 to 26 (default 0)
Coordinates	-2^{30} to $2^{30}-1$ plotter units

If the command moves the pen to a position whose absolute plotter unit coordinates are outside integer range, all following coordinates are discarded up to the next absolute flag, '=', that is followed by in-range coordinates.

Encode coordinate values as either base 64 (the default) or base 32 numbers, and send them to the printer using the corresponding ASCII character codes. Use base 32 on systems requiring a parity bit and base 64 on systems that do not.

To encode an integer, multiply its absolute value by 2, and, if the original value was negative, add 1, e.g. represent -50 as 101 and +50 as 100. Convert this number to base 64 or base 32, and encode each base 64 or 32 digit as an ASCII character code.

To encode a real number, multiply the number of decimal places by 3.33 and round this result up to the next integer (e.g. round 23.31 up to 24). This is the number of binary bits needed to represent the fractional part of the real number – the value that follows the '>' flag. Call this number a. Multiply the original real number by 2a, round it to the nearest integer and encode it as an integer as described above.

Numbers must be transmitted to the printer least significant digit first, and the last (most significant) digit of a number must be specified using a different ASCII range from that used for the preceding digits, as follows.

Base	Non-terminating codes	Terminating codes
32	63 – 94	95 – 126
64	63 – 126	191 – 254

For example, to encode a two-digit base 64 number with least significant digit 2, and most significant digit 7, encode 2 as 65 (63+2) and 7 as 198 (191+7).

Commas are not permitted within a **PE** command.

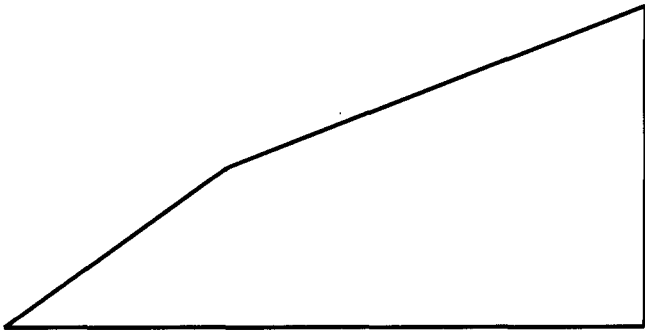
In symbol mode the selected symbol is drawn at each specified point. For a description of symbol mode refer to the **SM** command on page 166.

In polygon mode the specified points are stored in the polygon buffer and used when an Edge polygon or Fill polygon command is executed. For a description of polygon mode refer to the **PM** command on page 149.

After the command has been executed the previous plotting mode (absolute or relative) is restored and the pen is set in the "down" position, unless the **PE** command's final pen movement was with the pen "up".

The **PE** command with no parameters updates the carriage return point. For an explanation of the carriage return point refer to the **LO** Label origin command on page 179.

```
100 WIDTH "LPT1:",255;
110 LPRINT CHR$(27);"E";
120 LPRINT CHR$(27);"%0B";
130 LPRINT "INSC0,500,0,500,1,50,0";
140 LPRINT "PE:1<=150,150,100,0,0,50,-70,-25,-30,-25";
150 LPRINT CHR$(27);"%0A";
160 LPRINT CHR$(27);"E";
170 END
```



Draw circle

CI *radius* [, *chord*] [;]

radius: circle radius in current units

chord: chord angle in degrees

The command draws a circle of radius *radius*, whose center is the current cursor position.

The circle is comprised of equal chords which subtend an angle of *chord* degrees.

radius is a real number, and *chord* a clamped real number with a valid range of 0.5 to 180. The default value is 5, giving a default 72 chords to the circle. The smaller the angle *chord*, the smoother the circle.

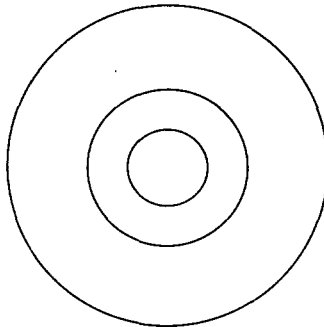
The command plots irrespective of the current pen state.

It is advisable to precede the command with a **PU** command, to avoid an unwanted dot at the center of the circle.

Anisotropic or point factor scaling may cause the circle to appear as an ellipse.

When the command has executed, the previous pen position (the center of the circle) and pen state (up or down) are restored.

```
100 WIDTH "LPT1:" ,255;  
110 LPRINT CHR$(27); "E";  
120 LPRINT CHR$(27); "%0B";  
130 LPRINT " INSC0,500,0,500,1,50,0";  
140 LPRINT " PU250,250CI100,CI50,CI25";  
150 LPRINT CHR$(27); "%0A";  
160 LPRINT CHR$(27); "E";  
170 END
```



Draw absolute arc

AA *X*, *Y*, *arc* [, *chord*] [;]

X: arc center x-coordinate

Y: arc center y-coordinate

arc: arc angle in degrees

chord: chord angle in degrees

The command draws an arc starting from the current cursor position. The arc's center is the specified point.

Coordinates are in current units and are absolute.

The radius of the arc is the distance between the current position and the point (*X*,*Y*).

The arc subtends an angle of *arc* degrees and is made up of equal chords, each subtending an angle of *chord* degrees. The smaller the value of *chord*, the smoother the arc.

If *arc* is positive, the arc is drawn counterclockwise; if it is negative, the arc is drawn clockwise.

An arc is only plotted if the pen is down.

After the command has finished, the pen position is at the opposite end of the arc from the starting point (even if the pen was up).

X and *Y* are real numbers.

arc is a clamped real number and *chord* a clamped real number with a valid range of 0.5 to 180. The default value is 5.

Anisotropic or point factor scaling may make the arc elliptical rather than circular.

It is advisable not to use an adaptive line type with this command. For an explanation of line types refer to the **LT** Line type command on page 162.

Draw absolute three point arc

AT *X1*, *Y1*, *X2*, *Y2* [, *chord*] [;]

X1: x-coordinate of intermediate point

Y1: y-coordinate of intermediate point

X2: x-coordinate of end point

Y2: y-coordinate of end point

chord: chord angle

The command draws an arc that starts at the current pen position, passes through the intermediate point, and finishes at the end point.

Coordinates are in current units and are absolute.

An arc is only plotted if the pen is down.

After the command has finished, the pen position is at the opposite end of the arc from the starting point (even if the pen was up).

Whether the arc is drawn clockwise or counterclockwise depends on the position of the intermediate point relative to the start and end points.

X1, *Y1*, *X2* and *Y2* are real numbers.

chord is a clamped real number with a valid range of 0.5 to 180. The default value is 5.

Anisotropic or point factor scaling may make the arc elliptical rather than circular.

If the intermediate point does not lie between the start and end points, an arc is not drawn. Instead two straight lines are plotted: one from the current pen position through the intermediate position to the edge of the effective window, and one from the opposite edge of the effective window to the end point.

Draw relative arc

AR *X*, *Y*, *arc* [, *chord*] [;]

X: arc center x-coordinate

Y: arc center y-coordinate

arc: arc angle in degrees

chord: chord angle in degrees

The command draws an arc starting from the current cursor position. The arc's center is the specified point.

Coordinates are in current units and are relative.

In all other respects the command functions in the same way as the **AA** Draw absolute arc command.

Draw relative three point arc

RT *X1*, *Y1*, *X2*, *Y2* [, *chord*] [;]

X1: x-coordinate of intermediate point

Y1: y-coordinate of intermediate point

X2: x-coordinate of end point

Y2: y-coordinate of end point

chord: chord angle

The command draws an arc that starts at the current pen position, passes through the intermediate point, and finishes at the end point.

Coordinates are in current units and are absolute: the intermediate and end point coordinates are specified relative to the start point (the current cursor position).

In all other respects the command functions in the same way as the **AT** Draw absolute three point arc command.

5.5.3 Polygon group

The commands that make up the polygon group are as follows:

Polygon mode	PM	Fill absolute rectangle	RA
Edge absolute rectangle	EA	Fill relative rectangle	RR
Edge relative rectangle	ER	Fill polygon	FP
Edge polygon	EP	Fill wedge	WG
Edge wedge	EW		

Polygon group commands store, plot, and fill polygons. The polygon buffer, a temporary printer storage area, holds coordinate pairs that define one or more polygons. The buffer has enough space for at least 512 points, and may be able to hold many more if printer memory is available. This depends in part on the number of fonts and macros downloaded in PCL mode. Multiple polygons in the buffer are referred to as sub-polygons. A polygon or sub-polygon stays in the buffer until overwritten by another polygon, or until the printer receives a **DF**; or **IN**; command. Some commands automatically use the contents of the polygon buffer; others only use the buffer in polygon mode. The **PM** command is used to enter polygon mode.

Polygon Mode

PM [*mode*] [;]

mode: command mode

The command enters or exits polygon mode, or closes a sub-polygon.

In polygon mode vector group commands, such as **PA** and **PR**, can be used to define the outline of a polygon.

A polygon in the buffer will not be plotted until polygon mode has been exited.

Multiple polygons in the buffer are known as sub-polygons.

The value of *mode* determines the action of the command.

mode = 0 empties the polygon buffer, enters polygon mode and stores the current pen position as the first vertex of the new polygon. Make sure you position the pen at the first point in the polygon before using a **PM0**; command.

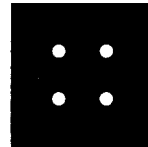
mode = 1 closes the current polygon or sub-polygon without exiting polygon mode. To close a polygon the command adds a point whose coordinates are the same as those of the starting point, so that the polygon is a closed shape.

The vector group commands that follow a **PM1**; define a single sub-polygon. A subsequent **PM1**; closes the polygon and marks the start of a new sub-polygon definition.

mode = 2 closes the current polygon or sub-polygon and exits polygon mode.

Unlike a **PM0**; command, a **PM1**; command does not store the current pen position as the first point of a new sub-polygon.

```
100 WIDTH "LPT1:",255;
110 LPRINT CHR$(27);"E";
120 LPRINT CHR$(27);"%0B";
130 LPRINT "INSC0,500,0,500,1,50,0";
140 LPRINT "PU100,100";
150 LPRINT "PM0PA100,400,400,400,400,100,100,100PM1";
160 LPRINT "PU200,200CI30PM1,PU200,300CI30PM1";
170 LPRINT "PU300,200CI30PM1,PU300,300CI30";
180 LPRINT "PM2;FP;EP";
190 LPRINT CHR$(27);"%0A";
200 LPRINT CHR$(27);"E";
210 END
```



The command with no parameters is equivalent to a **PM0**; command.

After polygon mode has been exited, the **EP** or **FP** command can be used to edge or fill the polygon or polygons in the buffer.

When a polygon is edged or filled, the pen is automatically raised and moved to the first point of the polygon in the “up” state.

The **EP** command only draws between points defined with the pen “down”.

The **FP** command fills a polygon irrespective of the pen state at the time the polygon was defined.

Only vector group commands, **IN**; and **DF**; have any effect in polygon mode. **IN**; and **DF**; clear the buffer and exit polygon mode.

If a Reset is performed while the printer is in polygon mode, the printer exits polygon mode, empties the polygon buffer, exits GL2 mode and ejects the current page.

Edge absolute rectangle

EA X, Y [:]

X : x-coordinate of rectangle opposite corner

Y : y-coordinate of rectangle opposite corner

The command draws a rectangle with the current cursor position and (X,Y) in opposite corners.

Coordinates are in current units and are absolute. Coordinate values are real numbers.

The rectangle is drawn irrespective of the pen state, using the current pen, line width and line attributes.

The command first empties the polygon buffer and then makes use of it to define the rectangle, however, you do not have to enter polygon mode to use the command. After the command has executed the buffer contains the rectangle vertices.

The current pen position and pen state do not change.

Edge relative rectangle

ER X, Y [:]

X : x-coordinate of rectangle opposite corner

Y : y-coordinate of rectangle opposite corner

The command draws a rectangle with the current cursor position and (X,Y) in opposite corners.

Coordinates are in current units and are relative. Hence the position of (X,Y) is specified relative to the current cursor position. Coordinate values are real numbers.

In all other respects the command functions in the same way as the **EA** Edge absolute rectangle command.

Edge polygon

EP [;]

The command plots the outline of the polygon or polygons in the buffer.

The command only draws between points defined while the pen was “down”.

Polygons are plotted irrespective of the current pen state, using the current pen, line width and line attributes.

All polygons in the buffer are plotted, including those implicitly defined by any previous **EA**, **ER**, **RA**, **RR**, **EW** or **WG** commands.

The data in the polygon buffer is not altered by the command.

The current pen position and pen state do not change.

Edge wedge

EW *radius*, *start*, *arc* [, *chord*] [;]

radius: radius

start: start point angle (degrees)

arc: arc angle (degrees)

chord: chord angle (degrees)

The command draws a wedge of radius *radius* whose center is the current cursor position.

The radius is in current units; *radius* is a real number.

The command first empties the polygon buffer and then makes use of it to define the wedge, however, you do not have to enter polygon mode to use the command. After the command has executed the buffer contains the wedge's vertices.

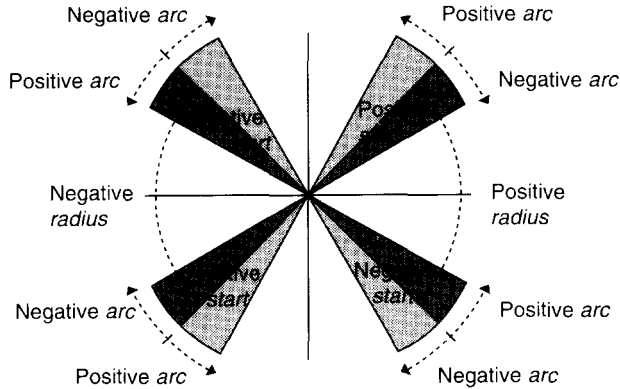
The wedge is plotted irrespective of the current pen state, using the current pen, line width and line attributes.

The current pen position and pen state do not change.

The starting point from which the arc of the wedge is plotted can be determined in terms of a reference radius that extends horizontally from the current pen position. Any point on the reference radius has the same y-coordinate value as the current pen position.

If *radius* is positive, the reference radius extends in the positive x-direction. If *radius* is negative, the reference radius extends in the negative x-direction.

If *start* is positive, the arc is drawn from a point *start* degrees counterclockwise from the reference radius. If *start* is negative, the arc is drawn from a point *start* degrees clockwise from the reference radius.



The arc is comprised of chords that subtend an angle of *chord* degrees. The smaller the value of *chord*, the smoother the arc.

start is a clamped real number.

If *start* is greater than 360, a start angle of *start* modulo 360 degrees is used.

arc should be in the range -360 to 360. If *arc* is greater than 360, a circle is drawn.

If *arc* is positive the arc is drawn counterclockwise. If *arc* is negative the arc is drawn clockwise.

chord should be in the range 0.5 to 180. The default value is 5.

Anisotropic or point factor scaling may cause the wedge to appear distorted.

Fill absolute rectangle

RA X, Y [;]

X : x-coordinate of rectangle opposite corner

Y : y-coordinate of rectangle opposite corner

The command draws and fills a rectangle with the current cursor position and (X,Y) in opposite corners.

Coordinates are in current units and are absolute. Coordinate values are real numbers.

The rectangle is drawn irrespective of the pen state, using the current pen, fill type, line width and line attributes.

The command first empties the polygon buffer and then makes use of it to define the rectangle, however, you do not have to enter polygon mode to use the command. After the command has executed the buffer contains the rectangle vertices.

The current pen position and pen state do not change.

Fill relative rectangle

RR X, Y [;]

X : x-coordinate of rectangle opposite corner

Y : y-coordinate of rectangle opposite corner

The command draws and fills a rectangle with the current cursor position and (X,Y) in opposite corners.

Coordinates are in current units and are relative. Hence the position of (X,Y) is specified relative to the current cursor position. Coordinate values are real numbers.

In all other respects the command functions in the same way as the **RA** Fill absolute rectangle command.

Fill polygon

FP [:]

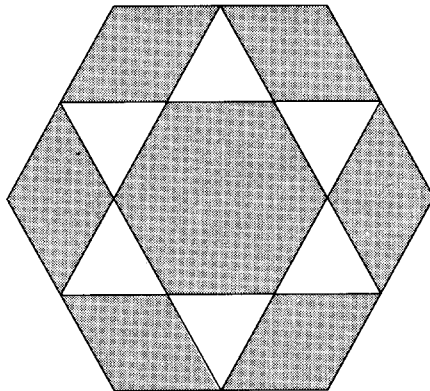
The command plots and fills the polygon or polygons in the buffer.

The command plots and fills irrespective of the pen state at the time the polygon was defined.

Polygons are plotted irrespective of the current pen state, using the current pen, fill type, line width and line attributes.

All polygons in the buffer are plotted and filled, including those implicitly defined by any previous **EA**, **ER**, **RA**, **RR**, **EW** or **WG** commands.

Areas formed by overlapping sub-polygons are alternately filled and left blank. The command fills a particular area enclosed by overlapping sub-polygons if an imaginary line drawn into the area from outside all the sub-polygons would intersect the sub-polygons' edges an odd number of times.



The data in the polygon buffer is not altered by the command.

The current pen position and pen state do not change.

Fill wedge

WG *radius, start, arc* [, *chord*] [:]

radius: radius

start: start point angle (degrees)

arc: arc angle (degrees)

chord: chord angle (degrees)

The command draws and fills a wedge of radius *radius* whose center is the current cursor position.

The command first empties the polygon buffer and then makes use of it to define the wedge, however, you do not have to enter polygon mode to use the command. After the command has executed the buffer contains the wedge's vertices.

The wedge is plotted and filled irrespective of the current pen state, using the current pen, fill type, line width and line attributes.

In all other respects the command functions in the same way as the **EG** Edge wedge command.

5.5.4 Line and fill attributes group

The commands that make up the line and fill attributes group are as follows:

Anchor corner	AC	Symbol mode	SM
Fill type	FT	Select pen	SP
Line attributes	LA	Screened vectors	SV
Line type	LT	Transparency mode	TR
Pen width	PW	User-defined line type	UL
Raster fill definition	RF	Select pen width unit	WU

The commands in this group establish the line and fill types that are used by vector and polygon group commands.

Anchor corner

AC [*X*, *Y*] [;]

X : x-coordinate of fill pattern anchor corner

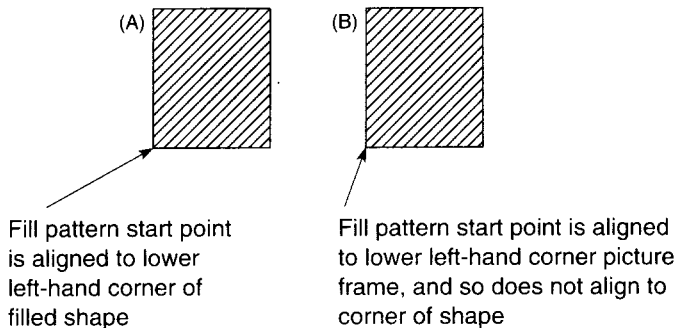
Y : y-coordinate of fill pattern anchor corner

The command establishes the starting point of the currently selected fill pattern.

Coordinates are in current units and are real numbers.

The command can be used to align a fill pattern with the shape which is to be filled, or to fill adjacent shapes with a continuous pattern.

The command with no parameters sets the anchor position to the lower left-hand corner of the picture frame in the current coordinate system rotation.



Fill type

FT [*fill* [, *op1* [, *op2*]]] [:]

fill: fill type (1, 2, 3, 4, 10, 11 or 21)

op1, *op2* : options

The command selects a shading pattern.

fill = 1 or 2 selects solid black. *op1* and *op2* are disregarded.

fill = 3 selects parallel hatching, *fill* = 4 selects cross-hatching. *op1* sets the distance in current x-axis units between the lines of the pattern. *op1* = 0 selects a spacing of 1% of the distance between P1 and P2. *op2* selects the angle in degrees between the hatching lines and the x-axis. The hatching lines are drawn using the current line type, pen width and line attributes. Subsequent changes in the position of P1 and P2 affect this spacing if user units were current at the time that the hatching fill was selected, but not if plotter units were in force. The lines in a cross-hatch pattern are drawn at the selected angle to the x-axis and at 90 degrees to the selected angle.

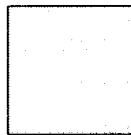
fill = 10 selects a gray scale. *op1* selects the tone (0 – 100%); 0% selects the lightest tone, and 100% the darkest. There are eight levels of gray available. *op2* is disregarded.



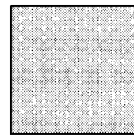
(0-2%)



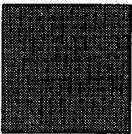
(3-10%)



(11-20%)



(21-35%)



(36-55%)



(56-80%)



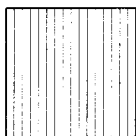
(81-99%)



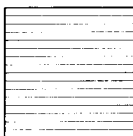
(100%)

fill = 11 selects a raster fill previously defined with the **RF** command. *op1* identifies the fill by index number (1 – 8). *op2* is disregarded. If no raster fill has been defined for the selected index number, a solid black fill is used instead.

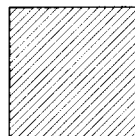
fill = 21 selects a PCL cross-hatch pattern. *op1* selects one of six predefined PCL cross-hatch patterns (1 – 6). *op2* is disregarded.



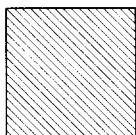
(#1)



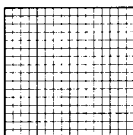
(#2)



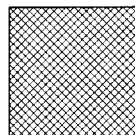
(#3)



(#4)



(#5)



(#6)

If *op1* or *op2* are omitted, the most recently supplied values for the selected fill type are used. If no values have been set, the default values for the fill type are used.

The command with no parameters defaults all fill type parameters and sets the fill type to solid fill.

Line attributes

LA [*attribute* , *value* [, *attribute* , *value* [, *attribute* , *value*]]] [;]

attribute: line attribute

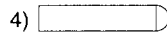
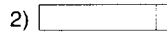
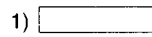
value: attribute value

The command specifies the shape of line joins and line ends by setting the three line attributes: line end type, line join type and miter limit.

attribute = 1 specifies line end type, *attribute* = 2 line join type, and *attribute* = 3 miter limit.

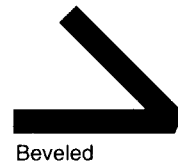
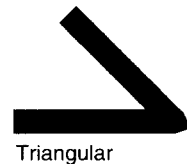
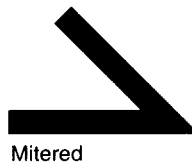
The four line end types and corresponding *value* values are as follows:

Value	Line end
1	Butt
2	Square
3	Triangular
4	Round

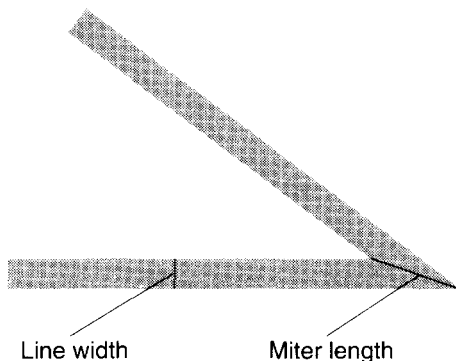


Line join types and corresponding *value* values are as shown below:

Value	Line join
1	Mitered
2	Mitered/beveled
3	Triangular
4	Round
5	Beveled
6	No join

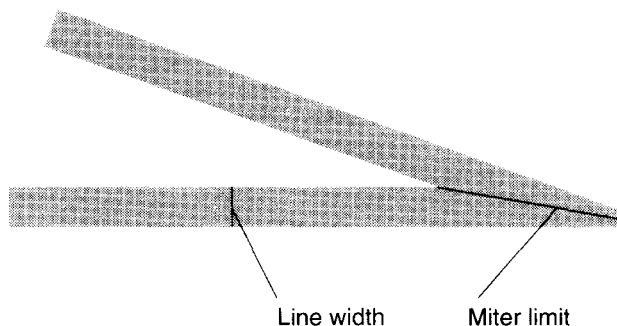


Miter length is the dimension shown below:



The miter limit is the maximum ratio of miter length to line width and is expressed as a number. e.g. if the miter length can be up to 8 times the line width, the miter limit is 8.

If mitered line join type is selected and the miter limit is exceeded, the join is beveled instead, the miter limit determining the cut off point.



If mitered/beveled line join type is selected and the miter limit is exceeded, a beveled line join is also used. In this case, however, the cut off point is determined by the lines' lengths and relative positions.

Lines 0.35mm wide or less always have butt line ends and no line join.

Labels are always drawn with rounded line ends and line joins, regardless of the current line attributes.

The command with no parameters sets line ends to butt, line joins to mitered and the miter limit to 5.

The line attributes settings remain in effect until the printer receives another **LA** command, or a **DF**; or **IN**; command.

Line type

LT [*type* [, *length* [, *mode*]]] [;]

type: line type

length: pattern length

mode: pattern length mode

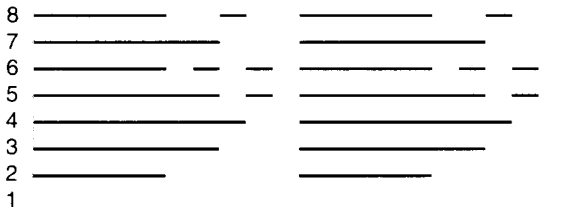
The command selects the line pattern which vector and polygon group commands will use.

type selects the line type. *type* is from -8 to 8 or is 99.

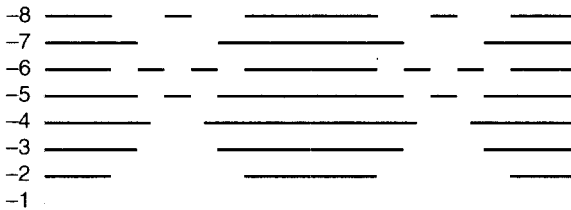
All line types are drawn using the current line attribute settings.

You can define custom line types with the **UL** command.

type = 1 to 8 selects a fixed pattern length line type. Any unused part of a pattern is carried over and used at the start of the next line.



type = -1 to -8 selects an adaptive pattern length line type. When a line is drawn the pattern is scaled to fit the line exactly.



type = 0 plots a single dot at each vertex of a rectangle plotted with an **AA**, **AR**, **AT** or **RT** command, each point in the parameter list of a **PA**, **PD** or **PR** command, and at the center of a circle drawn with a **CI** command.

length is specified either as a percentage of the distance between P1 and P2, or in millimeters, depending on the value of *mode*. If *length* is omitted, the most recently specified pattern length is used. If no value has been specified previously, the default value is used.

If *length* is zero or negative, the command is ignored.

mode = 0 specifies that *length* is specified as a percentage of the distance between P1 and P2. *mode* = 1 specifies that *length* is defined in millimeters. If *mode* is omitted, the most recently specified mode value is used. If no value has been specified previously, the default value is used.

LT99; restores the most recent previous line type. For fixed line types the pattern residue is also restored. **LT99**; only works if the following three conditions are true: 1) the current line type is solid, 2) since the current line type was selected the pen position has not changed and, 3) since the current line type was selected none of the following commands have been used.

Line and fill attributes group	Configuration and status group
AC, LA, LT (except LT; and LT99;), PW, RF, SP, TR, UL, WU	DF, IN, IP, IR, IW, RO, SC

LT; sets the line type to be solid. The previous line type, pattern length and line residue are saved.

It is advisable to use a fixed pattern line type to plot circles, arcs, wedges and polygons.

The line type settings remain in effect until the printer receives another **LT** command, or a **DF**; or **IN**; command.

Pen width

PW [*width* [, *pen*]] [;]

width: pen width

pen: pen number

The command sets the width of the specified pen.

The setting determines the width of subsequent lines drawn with the selected pen.

Pen width is either specified in millimeters, or as a percentage of the distance between P1 and P2; the current pen width unit selection determines which method is used. If no pen width unit has been specified, the width is set in millimeters.

If *width* = 0, a width of 1/300" is selected.

Widths specified in millimeters are scaled by the ratio of the PCL picture frame size to the GL2 plot size. If the ratio is different for the two axes, the thinner pen width (smaller ratio) is used. If this is less than 1/300", then a width of 1/300" is used.

If *pen* is not specified, both pens are set to the specified width. If *pen* is not 0 or 1, the command is disregarded.

The command with no parameters set both pens to a width of either 0.35mm or 0.1% of the P1-P2 distance, according to the current pen width unit.

The pen width setting does not affect the width of label characters.

The **DF**; command does not reset the pen width.

The pen width setting remains in effect until the printer receives another **PW** command or an **IN**; command.

Raster fill definition

RF [*index* [, *width*, *height*, *pixel* [...]]] [;]

index: pattern index

width: fill width

height: fill height

pixel: pixel setting

The command defines a raster fill pattern. Up to 8 fill patterns may be defined.

The **FT** command can be used to select a pattern.

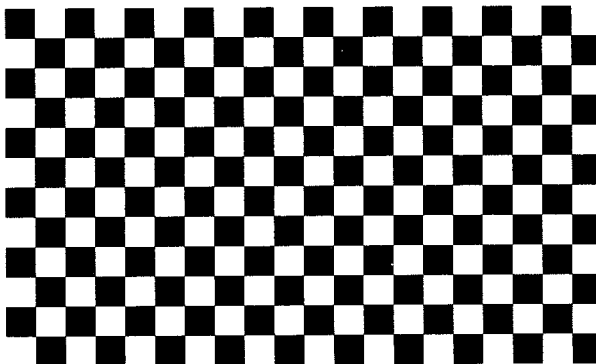
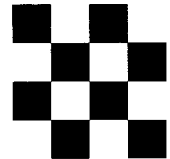
index is the index number of the pattern which is used to reference it when it is selected. *index* can be from 1 to 8. *width* is the width of the pattern in pixels and *height* is the height. *width* and *height* can both be from 1 to 255.

pixel represents a single pixel in the fill pattern. 0 stands for a white pixel, and any other value for a black pixel. There can be as many *pixel* parameters as there are pixels in the pattern i.e. *width* × *height*. Pixels are set from left to right and from top to bottom. If there are fewer than (*width* × *height*) pixel parameters, the trailing pixels are set to white.

If *width*, *height* and *pixel* are omitted, the pattern is defined as a solid black fill.

The command with no parameters sets all 8 raster fill patterns to solid black fill.

```
100 WIDTH "LPT1:",255;
110 LPRINT CHR$(27);"E";
120 LPRINT CHR$(27);"%0B";
130 LPRINT "INSP1SC0,500,0,500,1,50,0";
140 LPRINT "RF1,4,4,"
150 LPRINT "1,0,1,0"
160 LPRINT "0,1,0,1"
170 LPRINT "1,0,1,0"
180 LPRINT "0,1,0,1"
190 LPRINT "FT11,1PU100,100RA105,103";
200 LPRINT CHR$(27); "%0A";
210 LPRINT CHR$(27);"E";
220 END
```



Symbol mode

SM [*char*] [;]

char: ASCII character code

The command specifies a symbol for use with vector group commands, and initiates symbol mode.

In symbol mode the specified symbol is drawn at each point in the parameter list of any **PA**, **PR**, **PD**, **PU** or **PE** command, regardless of the pen state. If the pen is down, lines are plotted as well. The symbol is centered on the specified point.

char can be from any of the following character code ranges: 33 – 58, 60 – 126, 161 or 254. The semi-colon, ‘;’, character code 59, cannot be used, as it is the GL2 command terminator.

The symbol appears in the current font. If a new symbol set is selected, the symbol may change.

The symbol’s appearance is also determined by the current character size, slant and direction settings. See the Character group section on page 171.

The command with no parameters exits symbol mode. Subsequent **PA**, **PR**, **PD**, **PU** and **PE** commands do not cause the symbol to be plotted.

The command does not alter the pen position or pen state.

An **SM** command remains in effect until the printer receives another **SM** command, a **DF**; or an **IN**; command.

Select pen

SP [*pen*] [:]

pen: pen number

The command selects a pen color for drawing and filling.

pen = 0 selects the white pen. Output from the white pen is only visible on a non-white background and when transparency mode has been turned off with the **TR0**; command.

pen = 1 selects the black pen. Plotting commands produce output as normal.

Any other integer values of *pen* also select the black pen.

A change to the pen width does not change the current pen number selection.

The command with no parameter selects the white pen.

Screened vectors

SV [*screen* [, *op1* [, *op2*]]] [:]

screen: screen type (0, 1, 2 or 21)

op1 and *op2*: screen type options

The command selects the type of screening (shading) for use with lines, hatching patterns, arcs, circles and the edges of polygons, rectangles and wedges.

screen = 0 turns screening off.

screen = 1 selects a gray scale. *op1* specifies the percentage of shading required (0 – 100%); *op2* is disregarded. There are 8 gray scales available. See the description of the **FT** (Fill type) command on page 158.

screen = 2 selects a raster fill defined by the **RF** command. *op1* specifies its index number. If *op2* = 0, the fill uses the color of pen number 1. If *op2* = 1, the fill is in the current pen's color.

screen = 21 selects a PCL cross-hatch pattern. *op1* selects the pattern (1 – 6). See the description of the **FT** (Fill type) command on page 158 for a diagram showing the patterns available.

If *op1* or *op2* are omitted, the most recently set values for the selected screen type are used. If no values have been set, the values are defaulted.

The command with no parameters turns screening off.

Transparency mode

TR [*setting*] [;]

setting: transparency mode

The command turns transparency mode on or off, determining how source and destination images interact.

setting = 1 turns transparency mode on. The destination image may be seen through the white areas of the source image.

setting = 0 turns transparency mode off. The destination image cannot be seen through the white areas of the source image.

Refer to the description of the PCL print model on page 101 of Chapter 4 for a discussion of source transparency.

The command with no parameter turns transparency mode on.

A Reset or an **IN**; or **DF**; command turns transparency mode on.

User-defined line type

UL [*index*] [, *gap* [,...]] [;]

index: pattern index

gap: pattern gap

The command redefines line types. Up to 8 line patterns may be defined.

The **LT** command can be used to select the defined line types.

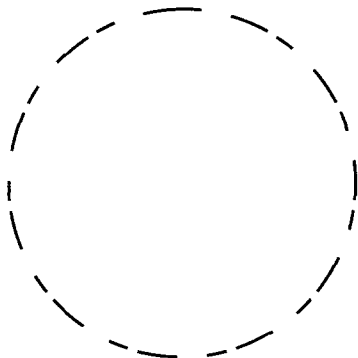
index (1 – 8) specifies the index number of the line type to be defined.

gap sets the length of alternate pen down and pen up segments on the line. A line type can have up to 20 segments; the first must always be a pen down segment.

```
100 WIDTH "LPT1:" ,255 ;
110 LPRINT CHR$(27) ; "E" ;
120 LPRINT CHR$(27) ; "%0B" ;
130 LPRINT "INSP1" ;
140 LPRINT "IP0,0,4064,4064" ;
150 LPRINT "SC0,100,0,100" ;
160 LPRINT "PU50,50" ;
170 LPRINT "UL4,40,30,20,10" ;
180 LPRINT "PW0.5LT4,12.5,0CI30" ;
```



```
190 LPRINT CHR$(27); "%0A";  
200 LPRINT CHR$(27); "E";  
210 END
```



gap is a positive clamped integer. The **LT** command automatically converts the gaps to percentages.

index is an absolute value. **UL-6** and **UL6** are equivalent. If a fixed line type is redefined, the corresponding adaptive line type is automatically redefined to match.

The sum of the *gap* parameters must be greater than 0. If an index number is specified but no *gap* parameters, the line type is set to the default for the index number.

The command with no parameters defaults all line types.

Select pen width unit

WU [*unit*] [;]

unit: unit type

The command specifies a pen width unit for use with the **PW** command. The unit selection applies to both pens.

The **WU** command always sets both pens' widths to default values.

unit = 0 selects millimeters as the pen width unit and sets all pen widths to 0.35mm.

unit = 1 specifies that pen width is to be designated as a percentage of the distance between P1 and P2. Both pens' widths are set to 0.1% of the current distance.

The command with no parameter sets the unit type to millimeters and both pens' widths to 0.35mm.

The unit type setting is not altered by a **DF**; command.

The width unit setting remains in effect until the printer receives another **WU** command or an **IN**; command.

5.5.5 Character group

The commands that comprise the character group are as follows:

Define standard font	SD	Relative direction	DR
Define alternate font	AD	Define variable text path	DV
Select standard font	SS	Character plot	CP
Select alternate font	SA	Character fill mode	CF
Select primary font	FI	Set absolute character size	SI
Select secondary font	FN	Set relative character size	SR
Define label	LB	Set character slant	SL
Define label terminator	DT	Scalable or bitmap fonts	SB
Label origin	LO	Extra space	ES
Absolute direction	DI	Transparent data	TD

The commands are used to print and manipulate text. Any font available in PCL mode can also be used in GL2 mode. In addition, the size, direction, fill pattern and slant of characters can be altered. As in PCL mode, two font definitions are always maintained, the standard font and the alternate font. You can switch between the two with a single command.

Define standard font

SD [*attribute*, *value*] [...] [;]

attribute: font attribute

value: attribute value

The command defines the standard font in terms of the seven font attributes.

Attributes are as follows: symbol set, spacing, pitch, height, posture, stroke weight and typeface.

attribute (1 – 7) identifies which attribute is to be set, as shown: .

<i>attribute</i>	Attribute
1	symbol set
2	spacing type
3	pitch
4	height
5	posture
6	stroke weight
7	typeface

Any number of attributes can be set: the current standard font settings are retained for any attributes not specified in the command.

value selects the setting for the selected attribute.

Available options for each attribute are as follows:

Symbol set

ISO 60: Norwegian	4	HP Spanish	51
Roman Extension	5	ISO 57: Chinese	75
ISO 25: French	6	ISO 17: Spanish	83
HP German	7	ISO 2: IRV	85
ISO 15: Italian	9	ISO 10: Swedish	115
JIS ASCII	11	ISO 16: Portuguese	147
ECMA-94 Latin 1	14	ISO 84: Portuguese	179
ISO 11: Swedish	19	ISO 85: Spanish	211
US-ASCII	21	Roman-8	277
ISO 61: Norwegian	36	IBM-PC(US)	341
ISO 4: UK	37	IBM-PC(Denmark/Norway)	373
ISO 69: French	38	PC-850	405
ISO 21: German	39		

Spacing type

Fixed spacing (default)	0
Proportional spacing	1

Pitch

Pitch setting	0 – 32767
---------------	-----------

Height

Height setting	0 – 32767
----------------	-----------

Posture

Upright (default)	0
Italic	1
Alternate italic	2

Stroke weight

-7	Ultra Thin	1	Semi Bold
-6	Extra Thin	2	Demi Bold
-5	Thin	3	Bold
-4	Extra Light	4	Extra Bold
-3	Light	5	Black
-2	Demi Light	6	Extra Black
-1	Semi Light	7	Ultra Block
0	Medium	9999	Stick font

If the stick font is selected (typeface 48), selecting a stroke weight of 9999 causes stick font characters to be rendered at the current pen width.

Typeface

Line printer	0 or 4096
Courier	3 or 4099
Times	5 or 4101
Stick font	48
Univers	52 or 4148

If no font with all the specified attribute values is available, the printer attempts to match the requested font as closely as possible using an available font. Attribute number determines the priority order in which attributes are matched; symbol set has the highest priority and typeface the lowest. This is analogous to the font selection procedure in PCL mode. See the explanation of font selection on page 77 of Chapter 4.

The command without parameters defaults the standard font attribute settings. The default standard font settings are as follows:

Default font settings

Attribute	<i>attribute</i>	Setting	Equivalent <i>value</i>
Symbol set	1	Roman-8	277
Font spacing	2	Fixed	0
Pitch	3	9 cpi	9
Point size	4	11.5 point	11.5
Posture	5	Upright	0
Stroke weight	6	Medium	0
Typeface	7	Stick font	48

Define alternate font

AD [*attribute*, *value*] [..] [;]

attribute: font attribute

value: attribute value

The command defines the alternate font in terms of the seven font attributes.

Any number of attributes can be set: the current alternate font settings are retained for any attributes not specified in the command.

The command without parameters defaults the alternate font attribute settings. The default alternate font settings are the same as the default standard font settings.

The command functions in the same way as the **SD** Define standard font command.

Select standard font

SS [;]

The command makes the standard font the current font. Subsequent label text is printed in the standard font.

The standard font remains selected until the printer receives an **SA** command or **<SO>** control code (ASCII code 14).

An **<SI>** control code (ASCII code 15) in a label string also selects the standard font.

The default standard font is the GL2 stick font. The **DF**; and **IN**; commands make the stick font the standard font and select it as the current font.

Select alternate font

SA [;]

The command makes the alternate font the current font. Subsequent label text is printed in the alternate font.

The alternate font remains selected until the printer receives an **SS** command, an **<SI>** control code (ASCII code 15) or a **DF**; or **IN**; command.

An **<SO>** control code (ASCII code 14) in a label string also selects the alternate font.

The default alternate font is the GL2 stick font.

Select primary font

FI *id* [;]

id: font identity number.

The command selects a font as the primary (standard) font. The font is identified by number.

Any font to which a font identity number has previously been assigned in PCL mode, may be selected. The font can be an internal font, a downloaded font or a cartridge font.

The standard font's attributes are set to those of the selected font.

If the selected font is scalable, a point size should first be specified with the **SD** command. Otherwise the current standard font point size is adopted.

If the selected font is proportionally spaced, the current standard font pitch is stored for future use.

If no font with the specified ID number is available, the command is ignored.

Select secondary font

FN *id* [;]

id: font identity number.

The command selects a font as the secondary (alternate) font. The font is identified by number.

Any font to which a font identity number has previously been assigned in PCL mode, may be selected. The font can be an internal font, a downloaded font or a cartridge font.

The alternate font's attributes are set to those of the selected font.

If the selected font is scalable, a point size should first be specified with the **AD** command. Otherwise the current alternate font point size is adopted.

If the selected font is proportionally spaced, the current alternate font pitch is stored for future use.

If no font with the specified ID number is available, the command is ignored.

Define label

LB [*char*] *term* [;]

char: character string

term: terminator

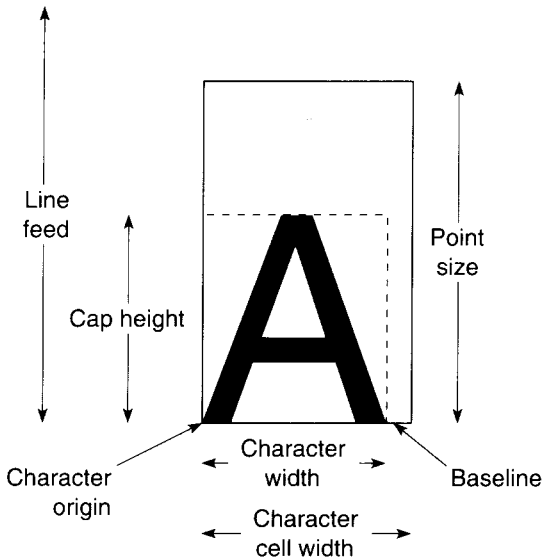
The command prints character string labels.

Labels can include non-printing characters, such as carriage returns or line feeds and must be terminated with a terminating character. The default terminator is the <ETX> code (ASCII code 3). A different label terminator may be defined with the **DT** command.

Printing starts from the current pen position unless a label origin has been set by an **LO** command.

Labels are printed even if the pen is up.

After the label has been printed the pen is at the bottom left-hand corner of the next character's character cell. The character cell is an imaginary bounding box enclosing a single character. Each character in a font has its own character cell definition. The character cell definitions determine the spacing and alignment between successive characters.



Define label terminator

DT [*char* [, *mode*]] [;]

char: character

mode: printing mode

The command defines a character as the label terminator.

mode = 0 causes the terminator to be printed as part of each string.

mode = 1 causes strings to be printed without the terminator appearing.

If *mode* is omitted, the terminator is not printed.

There must not be a space character between the letters **DT** and the terminator character. If there is, the space character will be made the terminator.

If no parameters are supplied, the default terminator <ETX> (ASCII 3) is used.

Label origin

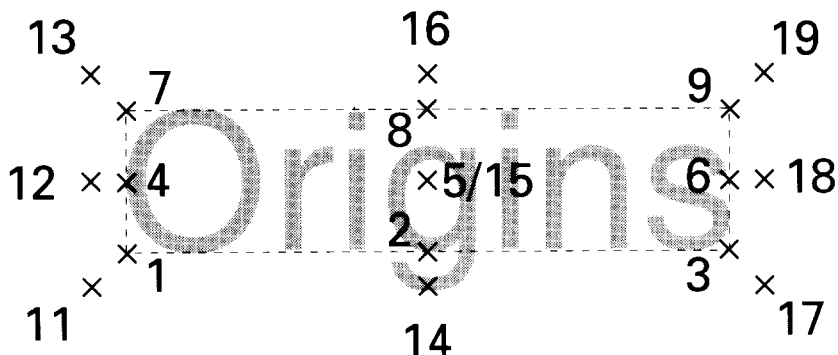
LO [*origin*] [;]

origin: label origin

The command determines the position relative to the current pen position from which label printing starts.

There are eighteen possible settings: labels can be centered, left-justified or right-justified relative to the current pen position, and vertically centered on, above or below the current pen position. Each of these positions can be offset 25% of the currently selected font's point size.

origin is either 1 – 9 or 11 – 19 and determines the offset as shown.



After a label has been printed, the pen position is at the bottom right-hand corner of the final character's character cell.

See the description of the **LB** Define label command on page 178 for a diagram of the character cell.

The command sets the carriage return point to the new label origin; a carriage return after the last character of the label and before the terminating character repositions the pen back at a label's origin after printing.

Each sequence of characters that follow a carriage return in a label will be printed from the label origin. Hence, several strings may be over-printed

The default label origin is at the bottom left-hand corner of the first character's character cell.

The command with no parameter defaults the label origin.

An **LO** command remains in effect until the printer receives another **LO** command or a **DF**; or **IN**; command.

Absolute direction

DI [*run*, *rise*] [;]

run: label direction x-component

rise: label direction y-component

The command determines the direction in which labels are printed relative to the coordinate system x-axis.

$rise / run =$ the tangent of the angle between the label's baseline and the x-axis.

run and *rise* are clamped integers.

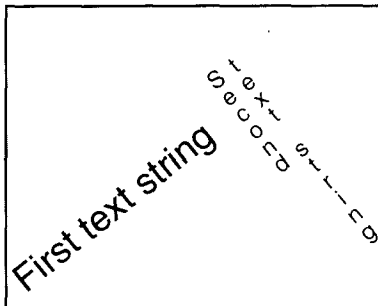
The command sets the carriage return point to the current pen position.

The text path set by the **DV** command and the absolute print direction together determine the orientation and direction of text.

If *run* and *rise* are both 0, the command is ignored.

The command without parameters sets the printing direction to horizontal.

```
100 LPRINT CHR$(27);"E";
110 LPRINT CHR$(27);"%0B";
120 LPRINT "INSP1IP0,0,4000,3200SC0,10,0,5";
130 LPRINT "DI8,4"; :REM Set text direction
140 LPRINT "DT*";
150 LPRINT "SD2,1,4,18,7,4";
160 LPRINT "PU0,0";
170 LPRINT "LB First text string";
180 LPRINT "DV1,1"; :REM Set text path to vertical
190 LPRINT "SD4,12";
200 LPRINT "PU5,4";
210 LPRINT "LBSecond";CHR$(13);CHR$(10); "text string";
220 LPRINT CHR$(27);"%0A";
230 LPRINT CHR$(27);"E";
240 END
```



Relative direction

DR [*run*, *rise*] [;]

run: label direction x-component

rise: label direction y-component

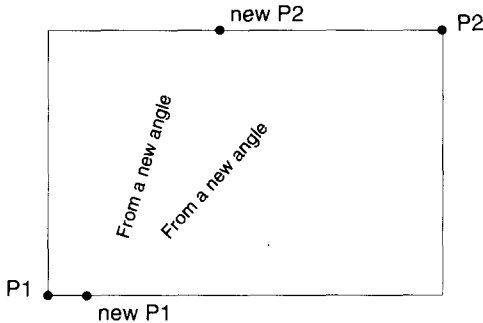
The command determines the direction in which labels are printed as a percentage of the horizontal and vertical distances between P1 and P2.

run and *rise* are clamped integers.

The command sets the carriage return point to the current pen location.

Changes to the relative positions of P1 and P2 cause the relative printing direction to change.

```
100 LPRINT CHR$(27);"E";
110 LPRINT CHR$(27);"%0B";
120 LPRINT "INSP1IP0,0,3000,1500SC0,10,0,10";
130 LPRINT "DR3,5"; :REM Set relative text direction
140 LPRINT "DT*";
150 LPRINT "SD2,1,4,10,7,52";
160 LPRINT "PU3,2";
170 LPRINT "LBFrom a new angle*";
180 LPRINT "IP300,0,1300,1500";
190 LPRINT "PU3,2";
200 LPRINT "LBFrom a new angle*";
210 LPRINT CHR$(27);"%0A";
220 LPRINT CHR$(27);"E";
230 END
```



The text path set by the **DV** command and the relative print direction together determine the orientation and direction of text.

If *run* and *rise* are both 0, the command is ignored.

The command without parameters sets the printing direction to horizontal.

Define variable text path

DV [*path* [, *action*]] [;]

path: text printing path

action: line feed action

The command sets the label printing direction and the carriage return point.

The text path is defined relative to the current absolute or relative printing direction. If no **DI** or **DR** command has been used, the text path is defined relative to the coordinate system x-axis.

path is 0, 1, 2 or 3 and sets the text path as shown.

```
100 LPRINT CHR$(27); "E";
110 LPRINT CHR$(27); "%0B";
120 LPRINT "INSP1SC0,100,0,100";
130 LPRINT "DT*";
140 LPRINT "SD2,1,4,18,7,4";
150 LPRINT "PU52,50";
160 LPRINT "LBText path=0*";
170 LPRINT "DV1";
180 LPRINT "PU50,48";
190 LPRINT "LBText path=1*";
200 LPRINT "DV2";
210 LPRINT "PU48,50";
220 LPRINT "LBText path=2*";
230 LPRINT "DV3";
240 LPRINT "PU50,52";
250 LPRINT "LBText path=3*";
260 LPRINT CHR$(27); "%0A";
270 LPRINT CHR$(27); "E";
280 END
```

3
=
h
t
a
p

t
x
e
T
S = rftsqt xet Text path = 0
T
e
x
t

p
a
t
h
=
1

action = 0 causes a line feed to reposition the pen clockwise at an angle of 90° to the text path. For example, if *path* = 3, a new line of text will be printed to the right of the previous line.

action = 1 causes a line feed to reposition the pen counterclockwise at an angle of 90° to the text path. For example, if *path* = 2, a new line of text will be printed below the previous line.

Changes to the positions of P1 and P2 do not affect the text path.

If *action* is omitted, a line feed repositions the pen clockwise at 90° to the text path.

The default text path is horizontal; printing is from left to right. A line feed repositions the pen clockwise at 90° to the text: equivalent to a **DV0,0**; command.

The command with no parameters sets the text path to be horizontal, with printing from left to right.

Character plot

CP [*spaces* [, *lines*]] [;]

spaces: pen movement in spaces

lines: pen movement in lines

The command repositions the pen a specified distance.

The distance is specified as a number of lines and spaces. No plotting is performed.

Pen movement is specified relative to the current printing direction.

spaces is the number of spaces the pen moves horizontally. A positive value moves the pen to the right, a negative value moves it to the left.

lines is the number of lines the pen moves vertically. A positive value moves the pen upwards, a negative value moves it downwards.

The height of a line and width of a space are determined by the current font's pitch (or space character width, if it is proportionally spaced) and line-spacing. The line-spacing, the vertical distance the pen moves after a line feed, is defined for every font, and may be adjusted with the **ES** Extra space command.

spaces and *lines* are clamped integers.

As the pen is repositioned, the carriage return point is adjusted accordingly.

The command does not affect the current pen state or margin settings.

The command with no parameters repositions the pen one line below the carriage return point: equivalent to a carriage return/ line feed.

Character fill mode

CF [*fill* [, *pen*]] [;]

fill: fill pattern

pen: pen number

The command specifies how text characters are to be edged and filled.

Scalable font characters may be edged and filled with any pattern that can be selected with the **FT** command. Bitmap font characters and stick font characters cannot be edged and may only be filled with a raster fill, shading pattern or a PCL cross-hatch pattern.

fill may be 0, 1, 2 or 3.

fill = 0 fills all characters with a solid fill and edges scalable font characters.

fill = 1 edges scalable font characters but does not fill them, and fills bitmap font and stick font characters.

fill = 2 fills all characters using the current fill type but does not edge them.

fill = 3 fills all characters with the current fill type, and edges scalable font characters.

pen = 0 selects the white pen for edging.

pen = 1 selects the black pen.

If no pen is specified, the current pen is used.

The width of the pen used to edge a character is proportional to the character's point size.

The line width used in cross-hatch patterns can be set with the **PW** command.

The command with no parameters selects a solid fill and the white pen: equivalent to **CF0,0**;

A **CF** command remains in effect until the printer receives another **CF** command, a **DF**; or an **IN**; command.

Set absolute character size

SI [*width*, *height*] [;]

width: character width in centimeters

height: character height in centimeters

The command determines the size of label characters.

width is a clamped integer. A negative *width* produces mirror-image characters, except after an **SB1**; command.

height is a clamped integer. A negative *height* produces upside-down characters.

```
100 LPRINT CHR$(27); "E";
110 LPRINT CHR$(27); "%0B";
120 LPRINT "INSP1";
130 LPRINT "IP0,0,4000,2500,SC0,100,0,100";
140 LPRINT "DT*";
150 LPRINT "SD2,1,4,12,7,52";
160 LPRINT "SI.5,.75";
170 LPRINT "PU50,160";
180 LPRINT "LBNormal text*";
190 LPRINT "SI.5,-.75";
200 LPRINT "PU50,140";
210 LPRINT "LBUpside-down text*";
220 LPRINT "SI.5,-1.5";
230 LPRINT "PU50,100";
240 LPRINT "LBBackward text*";
250 LPRINT "SI@-.5,-.75";
260 LPRINT "PU50,80";
270 LPRINT "LBBackward, upside-down text*";
280 LPRINT CHR$(27); "%0A";
290 LPRINT CHR$(27); "E";
300 END
```

Normal text

Upside-down text

Backward text

Backward, upside-down text

A change to the character size may alter the line width of stick font characters.

The **SB1**; command may cause unexpected variations in character size.

Set relative character size

SR [*width*, *height*] [;]

width: character width

height: character height

The command determines the size of label characters relative to P1 and P2.

width and *height* are specified respectively as a percentage of the horizontal and vertical distances from P1 to P2.

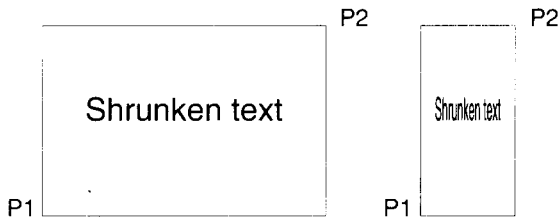
width is a clamped integer. A negative *width* produces mirror-image characters.

height is a clamped integer. A negative *height* produces upside-down characters.

Changes to the relative positions of P1 and P2 will alter the character size.

```
100 REM Program one
110 LPRINT CHR$(27);"E";
120 LPRINT CHR$(27);"%0B";
130 LPRINT "INSP1IP0,0,3000,2000SC0,100,0,100";
140 LPRINT "LO4";
150 LPRINT "DT*";
160 LPRINT "SD2,1,4,30,7,52";
170 LPRINT "PU50,50";
180 LPRINT "LBShrunken text*";
190 LPRINT CHR$(27);"%0A";
200 LPRINT CHR$(27);"E";
210 END

100 REM Program two
110 LPRINT CHR$(27);"E";
120 LPRINT CHR$(27);"%0B";
130 LPRINT "INSP1IP1000,0,2000,2000SC0,100,0,100";
140 LPRINT "LO4";
150 LPRINT "DT*";
160 LPRINT "SD2,1,4,30,7,52";
170 LPRINT "PU50,50";
180 LPRINT "LBShrunken text*";
190 LPRINT CHR$(27);"%0A";
200 LPRINT CHR$(27);"E";
210 END
```



If P2 is to the left of P1, characters will appear in mirror-image. If P2 is below P1, characters will appear upside-down.

A change to the relative character size may alter the line width of stick font characters.

The **SB1**; command may cause unexpected variations in character size.

The command without parameters sets the character width to 0.75% of the horizontal distance between P1 and P2, and the height to 1.5% of the vertical distance between P1 and P2.

Set character slant

SL [*tan*] [;]

tan: tangent of character slant angle

The command specifies a slant angle for label characters.

Only scalable font and stick font characters can be slanted.

A positive *tan* causes characters to slope forwards.

A negative *tan* causes characters to slope backwards.

```
100 LPRINT CHR$(27);"E";
110 LPRINT CHR$(27);"%0B";
120 LPRINT "INSP1IP0,0,3000,3000SC0,30,0,30";
130 LPRINT "DR3,5";
140 LPRINT "DT*";
150 LPRINT "SD2,1,4,25,7,52";
150 LPRINT "SI.7,1";
170 LPRINT "SL.3";
180 LPRINT "PU10,10";
190 LPRINT "LBForward slanting text*";
200 LPRINT "SL -.3";
210 LPRINT "PU10,3";
220 LPRINT "LBBackward slanting text*";
230 LPRINT CHR$(27);"%0A";
240 LPRINT CHR$(27);"E";
250 END
```

Forward slanting text
Backward slanting text

If *tan* = 0, characters are upright.

tan is a clamped real number.

The command without parameters causes characters to be printed upright.

Scalable or bitmap fonts

SB [*type*] [:]

type: font type

The command enables or disables the use of bitmap fonts.

type = 0 specifies that only scalable fonts and the stick font may be selected.

type = 1 specifies that any font may be selected.

An **SB1**; command may change the current standard or alternate font selection by allowing a bitmap font that better matches the most recently specified font attributes to be selected in preference to a scalable font.

An **SB0**; command will change the current standard or alternate font selection if either is currently a bitmap font.

Bitmap characters cannot be edged, can only be printed in orientations of 0°, 90°, 180° or 270°, and cannot be slanted. Bitmap font character sizes are approximate only.

The **FI** and **FN** commands automatically enable bitmap font selection when they select a bitmap font.

The command without parameters is equivalent to an **SB0**; command.

The default setting is scalable fonts and the stick font only.

Extra space

ES [*width* [, *height*]] [;]

width: character spacing change

height: line-spacing change

The command adjusts label character and line spacing.

width specifies the number of extra spaces between characters. A negative *width* removes spaces.

height specifies the number of extra lines between label lines. A negative *height* removes lines.

The width of a space is either the current pitch setting (for a fixed-pitch font), or the width of the space character (for a scalable font).

The line-spacing setting is determined by the current font.

Fractional values of *width* and *height* allow character and line spacings to be fine-tuned.

width and *height* are clamped real numbers.

The command without parameters specifies no extra character or line spacing.

An **ES** command remains in effect until the printer receives another **ES** command or a **DF**; or **IN**; command.

Transparent data

TD [*mode*] [;]

mode: data transparency mode

The command determines whether control characters are printed.

mode = 0 selects normal printing: control codes perform their normal operations and are not printed.

mode = 1 selects transparent mode printing: characters are printed if possible and control characters do not perform any operation (unless a control code is defined as the label terminator, in which case it will still perform this function). Non-printable or undefined characters appear as a space.

The command without parameters selects normal printing: equivalent to **TD0**;

MEMO

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 <i>ddd</i> - used to specify a character outside the standard character set.
\b	backspace		
\f	form feed	\newline	end of line (without the <i>newline</i> 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.

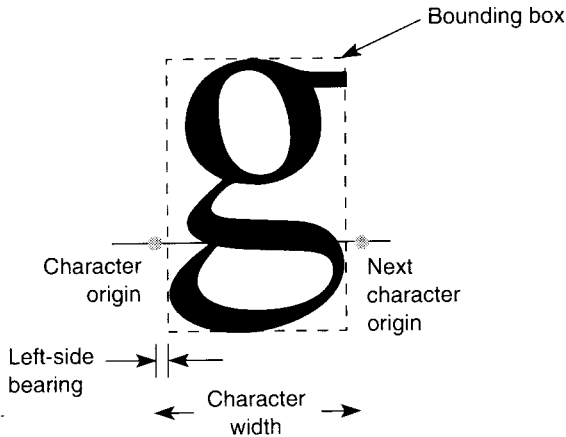
When a printing operator such as **show** attempts to print a character within a string (say, character code 65) it first looks up element 65 in the encoding vector to find the name of the character. Supposing the name of the character is ‘A’, it then looks up the procedure value associated with the 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

```

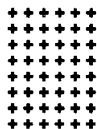
6.6.6 Creating a new font

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 appear 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.

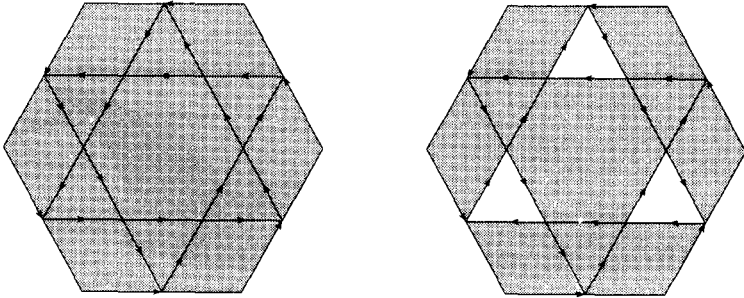
If gray scales specified by TrueImage are not accurately reflected on the printer, a new mapping of specified gray levels to printer gray levels may be defined using the **settransfer** function.

6.7.2 Filling complex paths

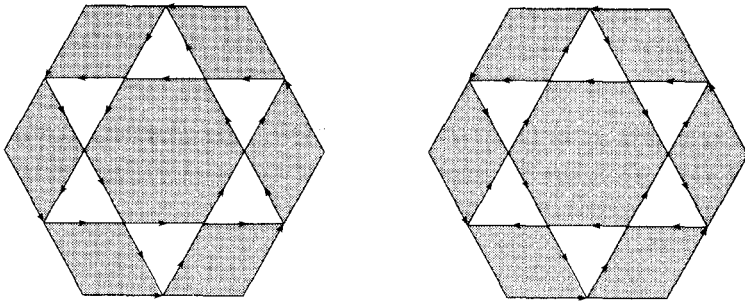
Complex paths that intersect themselves, or that contain subpaths that enclose other subpaths, are filled according to one of two rules: the non-zero winding rule and the even-odd rule. In either case, areas that are judged inside the path are painted, areas outside the path are left blank.

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.

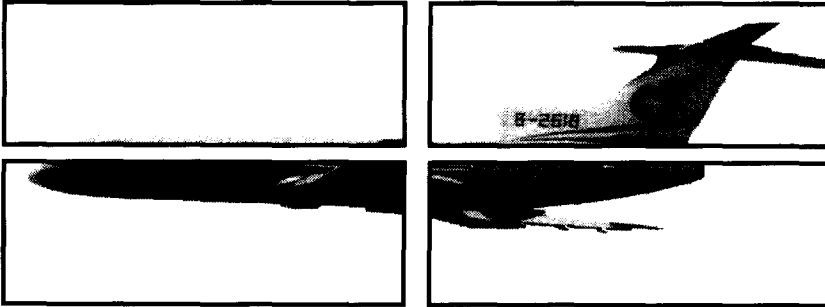


Polygons are filled in the same manner
irrespective of the direction of the
constituent sub-paths

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

any₁ any₂ **exch** any₂ any₁

exchanges the top two stack elements.

Errors - stackunderflow

dup

any **dup** any any

duplicates the top stack element.

Errors - stackoverflow, stackunderflow

copy

any₁ any₂ ... any_n n **copy** any₁ any₂ ... any_n any₁ any₂ ... any_n

duplicates the *n* stack elements *any*₁ to *any*_n.

Errors - invalidaccess, rangecheck, stackunderflow, stackoverflow,
typecheck

index

$any_n \dots any_0$ **index** $any_n \dots any_0$ 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

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

rotates the elements $any_{n-1} \dots any_0$ 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

$l-$ $any_1 \dots any_n$ **clear** -

discards all elements from the stack.

count

$l-$ $any_1 \dots any_n$ **count** $l-$ $any_1 \dots any_n$ 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 $obj_1 \dots obj_n$ **cleartomark** -

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

Errors - unmatchedmark

counttomark

mark $obj_1 \dots obj_n$ **counttomark** mark $obj_1 \dots obj_n$ 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

`num1 neg num2`

returns the result of multiplying num_1 by -1 .

Errors - stackunderflow, typecheck

ceiling

`num1 ceiling num2`

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

`num1 floor num2`

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

`num1 round num2`

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

`num1 truncate num2`

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

`num1 num2 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

`any1 any2 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 *any₁* and *any₂* need not be the same for them to be considered equal.

Errors - invalidaccess, stackunderflow

ne

`any1 any2 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

`num1 num2 ge` bool

returns *true* if *num₁* is greater than or equal to *num₂*, and *false* if *num₁* is less than *num₂*.

Errors - invalidaccess, stackunderflow, typecheck

gt

`num1 num2 gt` bool

returns *true* if *num₁* is greater than *num₂*, and *false* if *num₁* is less than or equal to *num₂*.

Errors - invalidaccess, stackunderflow, typecheck

le

`num1 num2 le` bool

returns *true* if *num₁* is less than or equal to *num₂*, and *false* if *num₁* is greater than *num₂*.

Errors - invalidaccess, stackunderflow, typecheck

lt

`num1 num2 lt` bool

returns *true* if *num₁* is less than *num₂*, and *false* if *num₁* is greater than or equal to *num₂*.

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

rline

`dx dy rline` -

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**

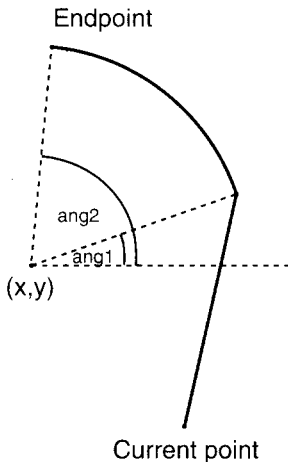
arc

`x y radius ang1 ang2 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, *ang₁* the angle of elevation of the arc's start point and *ang₂* 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

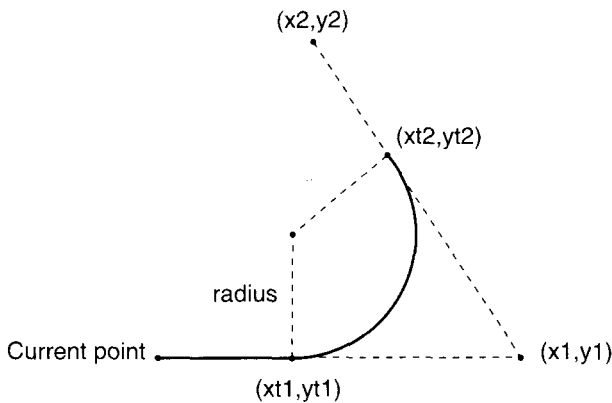
adds a circular arc to the current path, optionally preceded by a straight line segment. The arc is defined by the radius *radius* and two lines, a line from the current point to (x_1, y_1) , and a line from (x_1, y_1) to (x_2, y_2) . These lines are tangential to the arc.

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.

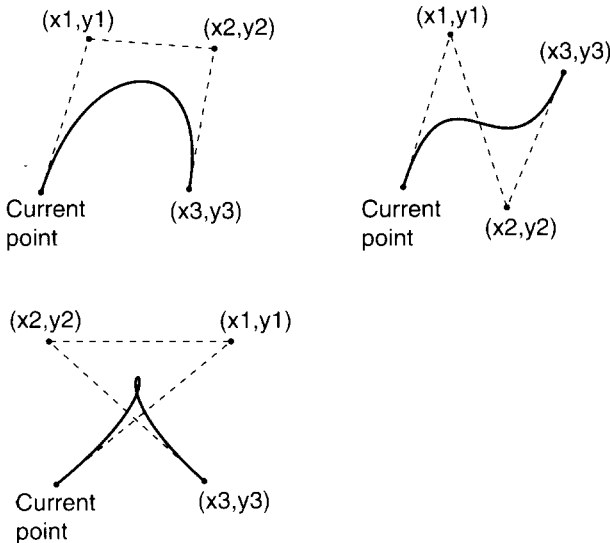


Errors - nocurrentpoint, rangecheck, limitcheck, stackunderflow, typecheck, undefinedresult

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

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

string₁ index string₂ **putinterval** -

copies *string₂* into *string₁*, replacing the sub-sequence of characters of *string₁* beginning with the character identified by *index*.

Errors - invalidaccess, rangecheck, stackunderflow, typecheck

copy

string₁ string₂ **copy** substring

copies all characters of *string₁* into *string₂*, returning the initial substring of *string₂* that contains the copied characters. The executable and access attributes of *substring* are the same as those of *string₂*. *string₁* cannot be longer than *string₂*.

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 oper- and 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

string₁ string₂ **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 *string₁* and *string₂* need not be the same for them to be considered equal.

Errors - *invalidaccess*, *stackunderflow*

ne

string₁ string₂ **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

`string1 string2 ge bool`

returns *true* if *string₁* is greater than or equal to *string₂*, and *false* if *string₁* is less than *string₂*. 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

`string1 string2 gt bool`

returns *true* if *string₁* is greater than *string₂*, and *false* if *string₁* is less than or equal to *string₂*. String ordering is as described under the **ge** operator above.

Errors - `invalidaccess`, `stackunderflow`, `typecheck`

le

`string1 string2 le bool`

returns *true* if *string₁* is less than or equal to *string₂*, and *false* if *string₁* is greater than *string₂*. String ordering is as described under the **ge** operator above.

Errors - `invalidaccess`, `stackunderflow`, `typecheck`

lt

`string1 string2 lt bool`

returns *true* if *string₁* is less than *string₂*, and *false* if *string₁* is greater than or equal to *string₂*. 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 obj₀ ... 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** element₀ ... 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

any₀ ... any_{n-1} array **astore** array

fills the array with the *n* objects *any₀* to *any_{n-1}*, where *n* is the array's length. *any₀* becomes the first element of the array and *any_{n-1}* the last.

Errors - invalidaccess, stackunderflow, typecheck

copy

array₁ array₂ **copy** subarray

copies all elements of *array₁* into *array₂*, returning the initial subarray of *array₂* that contains the copied objects. If elements of *array₁* are composite objects, their values are shared between *array₁* and *array₂*. The executable and access attributes of *subarray* are the same as those of *array₂*. *array₁* cannot be longer than *array₂*.

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** element₀ ... 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

packedarray₁ array₂ **copy** subarray

copies all elements of *packedarray₁* into *array₂*, returning the initial subarray of *array₂* that contains the copied objects. If elements of *packedarray₁* are composite objects, their values are shared between *packedarray₁* and *array₂*. The executable and access attributes of *subarray* are the same as those of *array₂*. *packedarray₁* cannot be longer than *array₂*.

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

dict₁ dict₂ **copy** dict₂

copies all key-value pairs in *dict₁* into *dict₂*, returning *dict₂*. If some values in *dict₁* are composite objects, they are shared between *dict₁* and *dict₂*. The executable and access attributes of *dict₂* are the same as those of *dict₁*. *dict₂* must initially contain no key-value pairs, and must be at least as long as *dict₁*.

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 proc₁ proc₂ **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	realttype	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

cv\$

any string **cv\$** 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 **cv\$**.

If *any* is a number, **cv\$** returns a string representation of the number. If *any* is a boolean, **cv\$** returns either the string *true* or *false*. If *any* is a string, **cv\$** simply copies its contents into *string*. If *any* is a name or an operator, **cv\$** returns the text representation of the name or operator name. If *any* is of any other type, **cv\$** 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

font₁ matrix **makefont** font₂

returns a new font whose characters are the characters of *font₁*, transformed by *matrix*. **makefont** creates a copy of *font₁*'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 *font₁*.

Errors - stackunderflow, typecheck

scalefont

font₁ scale **scalefont** font₂

returns a new font whose characters are the characters of *font₁*, scaled by a factor of *scale*. **scalefont** creates a copy of *font₁*'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 *font₁*.

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** *b*size *b*max *m*size *m*max *c*size *c*max *b*limit

returns current consumption and maximum space available for the following: bytes of bitmap storage (*b*size and *b*max), font/matrix combinations (*m*size and *m*max), and number of cached characters (*c*size and *c*max), plus the maximum number of bits that may be used to cache a single character, *b*limit.

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 permissible 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

`string1 string2 file file`

creates a file object for the file specified by *string₁*. The access type is specified by *string₂*: '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

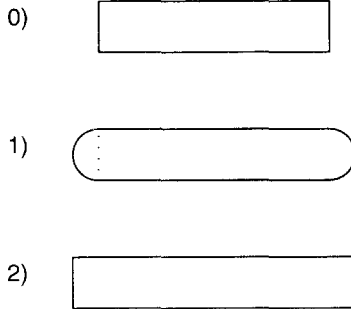
returns the current line width in the current graphics state.

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

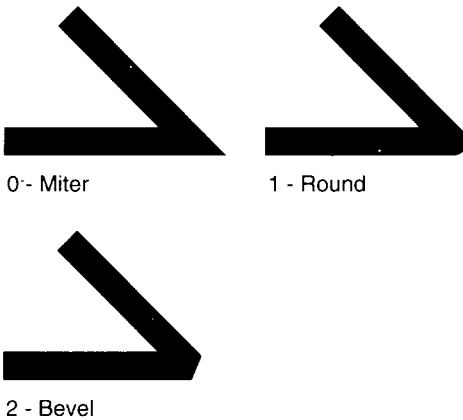
setlinejoin

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.



Errors - rangecheck, stackunderflow, typecheck

currentlinejoin

- **currentlinejoin** int

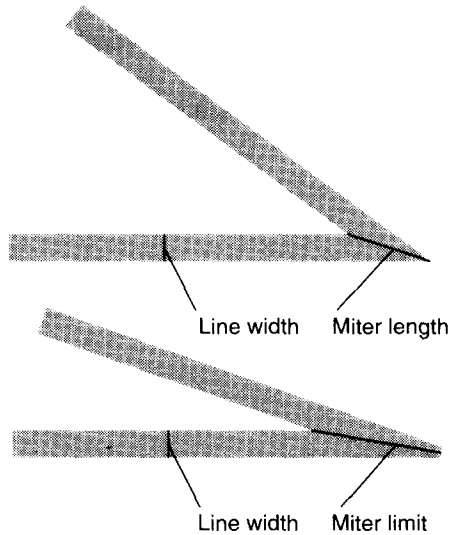
returns the current line join setting in the current graphics state.

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** -

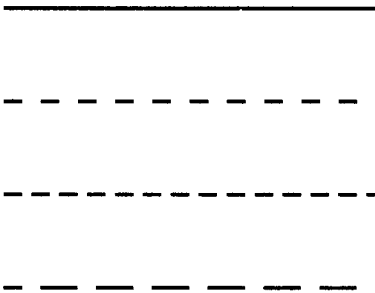
sets the current line dash pattern for the current graphics state. The dash pattern is specified by an array of numbers that specify alternating lengths of line and spacing. A single number defines a dash pattern that alternates equal lengths of line and spacing. The numbers in *array* should be non-negative 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

$$\begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x & y & 1 \end{matrix}$$

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

$$\begin{matrix} x & 0 & 0 \\ 0 & y & 0 \\ 0 & 0 & 1 \end{matrix}$$

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

$$\begin{matrix} \cos(\text{angle}) & \sin(\text{angle}) & 0 \\ -\sin(\text{angle}) & \cos(\text{angle}) & 0 \\ 0 & 0 & 1 \end{matrix}$$

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:

<i>papersize</i>	size	<i>papersize</i>	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**

searches 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 <descriptor and data>		
	Send character descriptor 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 <descriptor>		
	Send font descriptor	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 anchor 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 <i>any</i> to standard output file	269
==	Write syntactic representation of <i>any</i> to standard output file	269
a4tray	Look for A4 size paper tray	288
abs	Absolute value of <i>num1</i>	225
add	<i>num1</i> plus <i>num2</i>	225
aload	Push all elements of <i>array</i> on stack	246
aload	Push all elements of <i>packedarray</i> on stack	248
anchorseach	Determine if <i>seek</i> is initial substring of <i>string</i>	242
and	Logical bitwise and	229
arc	Append counterclockwise arc	232
arcn	Append clockwise arc	233
arcto	Append tangent arc	233
array	Create array of length <i>int</i>	245
ashow	Add (<i>x</i> , <i>y</i>) to width of each character while showing <i>string</i>	261
astore	Pop elements from stack into <i>array</i>	246
atan	Arctangent of <i>num1/num2</i> in degrees	226
awidthshow	Combine effects of ashow and widthshow	261
b5tray	Look for B5 size paper tray	288
begin	Push <i>dict</i> on dictionary stack	249
bind	Replace operator names in <i>proc</i> by operators	271
bitshift	Bitwise shift to <i>int1</i> (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 <i>num1</i>	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 <i>file</i>	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 <i>matrix</i> × CTM	281
concatmatrix	Fill <i>matrix3</i> with <i>matrix1</i> × <i>matrix2</i>	282
copy	Duplicate top <i>n</i> elements	223
copy	Copy elements of <i>string1</i> to initial substring of <i>string2</i>	242
copy	Copy elements of <i>array1</i> to initial subarray of <i>array2</i>	246
copy	Copy elements of <i>packedarray1</i> to initial subarray of <i>array2</i>	248
copy	Copy contents of <i>dict1</i> to <i>dict2</i>	251

Operator	Function	Page
copypage	Transmit current page	283
cos	Cosine of <i>angle</i> (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 <i>mark</i>	224
currentcacheparams	Return current font cache parameters	264
currentdash	Return current dash pattern	277
currentdict	Push current dictionary on operand 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 <i>matrix</i> 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 scion	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 <i>key</i> and <i>value</i> in current dictionary	249
defaultmatrix	Fill <i>matrix</i> with device default matrix	280
definefont	Register <i>font</i> as a font dictionary	260
dict	Create dictionary with capacity for <i>int</i> elements	249
dictfull	No more room in dictionary	290
dictstack	Copy dictionary stack into <i>array</i>	252
dictstackoverflow	Too many begins	290
dictstackunderflow	Too many ends	290
div	<i>num1</i> divided by <i>num2</i>	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 (<i>dx1</i> , <i>dy1</i>) by CTM or <i>matrix</i>	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 <i>array</i>	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 <i>num</i> to <i>exponent</i> power	227
false	Push boolean value <i>false</i>	229
file	Open file identified by <i>string1</i> with access <i>string2</i>	265
fill	Fill current path with current color	238
findfont	Return font dictionary identified by <i>key</i>	260
findtray	Find the specific paper tray	286
flattenpath	Convert curves to sequences of straight lines	235
floor	Floor of <i>num1</i>	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 <i>proc</i> with values form <i>start</i> by steps of <i>increment</i> to <i>finish</i>	253
forall	Execute <i>proc</i> for each element of <i>string</i>	242
forall	Execute <i>proc</i> for each element of <i>array</i>	246
forall	Execute <i>proc</i> for each element of <i>packedarray</i>	248
forall	Execute <i>proc</i> for each element of <i>dict</i>	251
framedevice	Install frame buffer device	283
ge	Test greater or equal	228, 244
get	Get string element indexed by <i>index</i>	241
get	Get array element indexed by <i>index</i>	245
get	Get <i>packedarray</i> element indexed by <i>index</i>	247
get	Get value associated with <i>key</i> in <i>dict</i>	250
getinterval	Substring of <i>string</i> at <i>index</i> for <i>count</i> elements	241
getinterval	Subarray of <i>array</i> starting at <i>index</i> for <i>count</i> elements	245
getinterval	Subarray of <i>packedarray</i> starting at <i>index</i> for <i>count</i> 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 <i>matrix</i> with identity transform	280
idiv	Integer divide	225
idtransform	Inverse transform distance (<i>dx₁</i> , <i>dy₁</i>) by CTM or <i>matrix</i>	282
if	Execute <i>proc</i> if <i>bool</i> is true	253
ifelse	Execute <i>proc1</i> if <i>bool</i> is true, <i>proc2</i> if <i>bool</i> 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 <i>matrix2</i> with inverse of <i>matrix1</i>	282
ioerror	Input/output error occurred	290
itransform	Inverse transform (<i>x₁</i> , <i>y₁</i>) by CTM or <i>matrix</i>	282
known	Test whether <i>key</i> is in <i>dict</i>	250
kshow	Execute <i>proc</i> between characters shown from <i>string</i>	262
le	Test less or equal	228, 244
legaltray	Look for Legal size paper tray	288
length	Number of elements in <i>string</i>	241
length	Number of elements in <i>array</i>	245
length	Number of elements in <i>packedarray</i>	247
length	Number of key-value pairs in <i>dict</i>	249
lettertray	Look for Letter size paper tray	287
limitcheck	Implementation limit exceeded	290
lineto	Append straight line to (<i>x</i> , <i>y</i>)	231
ln	Natural logarithm (base <i>e</i>)	227
load	Search dictionary stack for <i>key</i> and return associated <i>value</i>	250
log	Logarithm (base 10)	227
loop	Execute <i>proc</i> an indefinite number of times	253
lt	Test less than	228, 244
makefont	Transform <i>font1</i> by <i>matrix</i> to produce new <i>font2</i>	260
mark	Push mark on stack	224
matrix	Create identity matrix	280
maxlength	Current capacity of <i>dict</i>	249
mod	<i>int1</i> mod <i>int2</i>	225
monarcenvelope	Establish imaging area to Monarch size envelope	287
monarcenvelopetray	Look for Monarch size envelope tray	288
moveto	set current point to (<i>x</i> , <i>y</i>)	231

Operator	Function	Page
mul	<i>num1</i> times <i>num2</i>	225
ne	Test not equal	228, 243
neg	Negative of <i>num1</i>	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 <i>null</i> on operand stack	271
nulldevice	Install no-output device	283
or	Logical bitwise inclusive or	229
packedarray	Create packed array consisting of the specified <i>n</i> 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 <i>string</i> to standard output file	268
prompt	Executed when ready for interactive input	269
pstack	Print stack non-destructively using ==	269
put	Put <i>int</i> into <i>string</i> at <i>index</i>	241
put	Put <i>any</i> into <i>array</i> at <i>index</i>	245
put	Associate <i>key</i> with <i>value</i> in <i>dict</i>	250
putinterval	Replace substring of <i>string1</i> starting at <i>index</i> by <i>string2</i>	241
putinterval	Replace subarray of <i>array</i> starting at <i>index</i> by <i>subarray</i>	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 <i>file</i>	265
readhexstring	Read hex from <i>file</i> into string	266
readline	Read line from <i>file</i> into string	266
readonly	Reduce access to read-only	257
readstring	Read string from <i>file</i>	266
repeat	Execute <i>proc int</i> 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 <i>n</i> elements up <i>j</i> times	224
rotate	Rotate user space or define rotation by <i>angle</i> degrees	281
round	Round <i>num1</i> 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
scaleftont	Scale <i>font1</i> by <i>scale</i> to produce new <i>font2</i>	260
search	Search for <i>seek</i> 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 <i>matrix</i>	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 <i>string</i> on page	261
showpage	Transmit and reset current page	283
sin	Sine of <i>angle</i> (degrees).	227
sqrt	Square root of <i>num</i>	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 <i>file</i>	268
stop	Terminate stopped context	254
stopped	Establish context for catching stop	254
store	Replace topmost definition of <i>key</i>	250
string	Create string of length <i>int</i>	241
stringwidth	Width of <i>string</i> in current font	262
stroke	Draw line along current path	238
strokepath	Compute outline of stroked path	235
sub	<i>num1</i> minus <i>num2</i>	225
syntaxerror	Language syntax error	291

Operator	Function	Page
systemdict	System dictionary	252
timeout	Time limit exceeded	291
token	Read token from start of <i>string</i>	243
token	Read token from <i>file</i>	267
transform	Transform (x_1, y_1) by CTM or <i>matrix</i>	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 <i>true</i>	229
truncate	Remove fractional part of <i>num1</i>	226
type	Return name identifying the type of <i>any</i>	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 <i>key</i> is defined	251
widthshow	Add (x, y) to width of <i>char</i> while showing <i>string</i>	261
write	Write one character to <i>file</i>	265
writehexstring	Write <i>string</i> to <i>file</i> as hex	266
writestring	Write <i>string</i> to <i>file</i>	266
xcheck	Test executable attribute	256
xor	Logical 1 bitwise exclusive or	229

7.2 Character set tables

ISO 60: Norwegian

	0	1	2	3	4	5	6	7
0	<input type="text" value="0"/>	<input type="text" value="16"/>	<input type="text" value="32"/>	<input type="text" value="48"/>	<input type="text" value="64"/>	<input type="text" value="80"/>	<input type="text" value="96"/>	<input type="text" value="112"/>
1	<input type="text" value="1"/>	<input type="text" value="17"/>	<input type="text" value="33"/>	<input type="text" value="49"/>	<input type="text" value="65"/>	<input type="text" value="81"/>	<input type="text" value="97"/>	<input type="text" value="113"/>
2	<input type="text" value="2"/>	<input type="text" value="18"/>	<input type="text" value="34"/>	<input type="text" value="50"/>	<input type="text" value="66"/>	<input type="text" value="82"/>	<input type="text" value="98"/>	<input type="text" value="114"/>
3	<input type="text" value="3"/>	<input type="text" value="19"/>	<input type="text" value="35"/>	<input type="text" value="51"/>	<input type="text" value="67"/>	<input type="text" value="83"/>	<input type="text" value="99"/>	<input type="text" value="115"/>
4	<input type="text" value="4"/>	<input type="text" value="20"/>	<input type="text" value="36"/>	<input type="text" value="52"/>	<input type="text" value="68"/>	<input type="text" value="84"/>	<input type="text" value="100"/>	<input type="text" value="116"/>
5	<input type="text" value="5"/>	<input type="text" value="21"/>	<input type="text" value="37"/>	<input type="text" value="53"/>	<input type="text" value="69"/>	<input type="text" value="85"/>	<input type="text" value="101"/>	<input type="text" value="117"/>
6	<input type="text" value="6"/>	<input type="text" value="22"/>	<input type="text" value="38"/>	<input type="text" value="54"/>	<input type="text" value="70"/>	<input type="text" value="86"/>	<input type="text" value="102"/>	<input type="text" value="118"/>
7	<input type="text" value="7"/>	<input type="text" value="23"/>	<input type="text" value="39"/>	<input type="text" value="55"/>	<input type="text" value="71"/>	<input type="text" value="87"/>	<input type="text" value="103"/>	<input type="text" value="119"/>
8	<input type="text" value="8"/> <BS>	<input type="text" value="24"/>	<input type="text" value="40"/>	<input type="text" value="56"/>	<input type="text" value="72"/>	<input type="text" value="88"/>	<input type="text" value="104"/>	<input type="text" value="120"/>
9	<input type="text" value="9"/> <HT>	<input type="text" value="25"/>	<input type="text" value="41"/>	<input type="text" value="57"/>	<input type="text" value="73"/>	<input type="text" value="89"/>	<input type="text" value="105"/>	<input type="text" value="121"/>
A	<input type="text" value="10"/> <LF>	<input type="text" value="26"/>	<input type="text" value="42"/>	<input type="text" value="58"/>	<input type="text" value="74"/>	<input type="text" value="90"/>	<input type="text" value="106"/>	<input type="text" value="122"/>
B	<input type="text" value="11"/>	<input type="text" value="27"/> <ESC>	<input type="text" value="43"/>	<input type="text" value="59"/>	<input type="text" value="75"/>	<input type="text" value="91"/>	<input type="text" value="107"/>	<input type="text" value="123"/>
C	<input type="text" value="12"/> <FF>	<input type="text" value="28"/>	<input type="text" value="44"/>	<input type="text" value="60"/>	<input type="text" value="76"/>	<input type="text" value="92"/>	<input type="text" value="108"/>	<input type="text" value="124"/>
D	<input type="text" value="13"/> <CR>	<input type="text" value="29"/>	<input type="text" value="45"/>	<input type="text" value="61"/>	<input type="text" value="77"/>	<input type="text" value="93"/>	<input type="text" value="109"/>	<input type="text" value="125"/>
E	<input type="text" value="14"/> <SO>	<input type="text" value="30"/>	<input type="text" value="46"/>	<input type="text" value="62"/>	<input type="text" value="78"/>	<input type="text" value="94"/>	<input type="text" value="110"/>	<input type="text" value="126"/>
F	<input type="text" value="15"/> <SI>	<input type="text" value="31"/>	<input type="text" value="47"/>	<input type="text" value="63"/>	<input type="text" value="79"/>	<input type="text" value="95"/>	<input type="text" value="111"/>	<input type="text" value="127"/>

Roman Extension

	0	1	2	3	4	5	6	7	
0	0	16	32	48	â	Å	Á	Ɔ	
1	1	17	33	49	ê	î	Ã	Ɔ	
2	2	18	34	50	ô	Ø	ã	·	
3	3	19	35	51	û	Æ	Ð	μ	
4	4	20	36	52	á	å	ø	¶	
5	5	21	37	53	é	í	Í	¾	
6	6	22	38	54	ó	ø	Ï	-	
7	7	23	39	55	ú	æ	Ó	¼	
8	<BS>	8	24	40	ì	à	Ä	Ò	½
9	<HT>	9	25	41	ç	è	ì	Õ	à
A	<LF>	10	26	42	α	ò	Ö	õ	º
B	<ESC>	11	27	43	£	ù	Ü	Š	«
C	<FF>	12	28	44	¥	ä	É	š	■
D	<CR>	13	29	45	§	ë	ï	Ú	»
E	<SO>	14	30	46	ƒ	ö	ß	ÿ	±
F	<SI>	15	31	47	£	ç	ü	ô	ÿ

ISO 25: French

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

HP German

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

ISO 15: Italian

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

JIS ASCII

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

ECMA-94 Latin 1

	0	1	2	3	4	5	6	7
0	0	16	32	0	@	P	`	p
1	1	17	33	1	A	Q	a	q
2	2	18	34	2	B	R	b	r
3	3	19	35	3	C	S	c	s
4	4	20	36	4	D	T	d	t
5	5	21	37	5	E	U	e	u
6	6	22	38	6	F	V	f	v
7	7	23	39	7	G	W	g	w
8	<BS>	24	40	8	H	X	h	x
9	<HT>	25	41	9	I	Y	i	y
A	<LF>	26	42	:	J	Z	j	z
B	<ESC>	27	43	;	K	[k	{
C	<FF>	28	44	<	L	\	l	
D	<CR>	29	45	=	M]	m	}
E	<SO>	30	46	>	N	^	n	~
F	<SI>	31	47	?	O	-	o	☒


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	ì 161	± 177	Á 193	Ñ 209	á 225	ñ 241
2	130	146	ç 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	ö 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	$\frac{1}{4}$ 188	Ì 204	Ü 220	ì 236	ü 252
D	141	157	­ 173	$\frac{1}{2}$ 189	Í 205	Ý 221	í 237	ý 253
E	142	158	® 174	$\frac{3}{4}$ 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	<input type="checkbox"/> 0	<input type="checkbox"/> 16	<input type="checkbox"/> 32	0 <input type="checkbox"/> 48	É <input type="checkbox"/> 64	P <input type="checkbox"/> 80	é <input type="checkbox"/> 96	p <input type="checkbox"/> 112
1	<input type="checkbox"/> 1	<input type="checkbox"/> 17	! <input type="checkbox"/> 33	1 <input type="checkbox"/> 49	A <input type="checkbox"/> 65	Q <input type="checkbox"/> 81	a <input type="checkbox"/> 97	q <input type="checkbox"/> 113
2	<input type="checkbox"/> 2	<input type="checkbox"/> 18	" <input type="checkbox"/> 34	2 <input type="checkbox"/> 50	B <input type="checkbox"/> 66	R <input type="checkbox"/> 82	b <input type="checkbox"/> 98	r <input type="checkbox"/> 114
3	<input type="checkbox"/> 3	<input type="checkbox"/> 19	# <input type="checkbox"/> 35	3 <input type="checkbox"/> 51	C <input type="checkbox"/> 67	S <input type="checkbox"/> 83	c <input type="checkbox"/> 99	s <input type="checkbox"/> 115
4	<input type="checkbox"/> 4	<input type="checkbox"/> 20	¤ <input type="checkbox"/> 36	4 <input type="checkbox"/> 52	D <input type="checkbox"/> 68	T <input type="checkbox"/> 84	d <input type="checkbox"/> 100	t <input type="checkbox"/> 116
5	<input type="checkbox"/> 5	<input type="checkbox"/> 21	% <input type="checkbox"/> 37	5 <input type="checkbox"/> 53	E <input type="checkbox"/> 69	U <input type="checkbox"/> 85	e <input type="checkbox"/> 101	u <input type="checkbox"/> 117
6	<input type="checkbox"/> 6	<input type="checkbox"/> 22	& <input type="checkbox"/> 38	6 <input type="checkbox"/> 54	F <input type="checkbox"/> 70	V <input type="checkbox"/> 86	f <input type="checkbox"/> 102	v <input type="checkbox"/> 118
7	<input type="checkbox"/> 7	<input type="checkbox"/> 23	' <input type="checkbox"/> 39	7 <input type="checkbox"/> 55	G <input type="checkbox"/> 71	W <input type="checkbox"/> 87	g <input type="checkbox"/> 103	w <input type="checkbox"/> 119
8	<BS> <input type="checkbox"/> 8	<input type="checkbox"/> 24	(<input type="checkbox"/> 40	8 <input type="checkbox"/> 56	H <input type="checkbox"/> 72	X <input type="checkbox"/> 88	h <input type="checkbox"/> 104	x <input type="checkbox"/> 120
9	<HT> <input type="checkbox"/> 9	<input type="checkbox"/> 25) <input type="checkbox"/> 41	9 <input type="checkbox"/> 57	I <input type="checkbox"/> 73	Y <input type="checkbox"/> 89	i <input type="checkbox"/> 105	y <input type="checkbox"/> 121
A	<LF> <input type="checkbox"/> 10	<input type="checkbox"/> 26	* <input type="checkbox"/> 42	: <input type="checkbox"/> 58	J <input type="checkbox"/> 74	Z <input type="checkbox"/> 90	j <input type="checkbox"/> 106	z <input type="checkbox"/> 122
B	<input type="checkbox"/> 11	<ESC> <input type="checkbox"/> 27	+ <input type="checkbox"/> 43	; <input type="checkbox"/> 59	K <input type="checkbox"/> 75	Ä <input type="checkbox"/> 91	k <input type="checkbox"/> 107	ä <input type="checkbox"/> 123
C	<FF> <input type="checkbox"/> 12	<input type="checkbox"/> 28	' <input type="checkbox"/> 44	< <input type="checkbox"/> 60	L <input type="checkbox"/> 76	Ö <input type="checkbox"/> 92	l <input type="checkbox"/> 108	ö <input type="checkbox"/> 124
D	<CR> <input type="checkbox"/> 13	<input type="checkbox"/> 29	- <input type="checkbox"/> 45	= <input type="checkbox"/> 61	M <input type="checkbox"/> 77	Å <input type="checkbox"/> 93	m <input type="checkbox"/> 109	å <input type="checkbox"/> 125
E	<SO> <input type="checkbox"/> 14	<input type="checkbox"/> 30	• <input type="checkbox"/> 46	> <input type="checkbox"/> 62	N <input type="checkbox"/> 78	Ü <input type="checkbox"/> 94	n <input type="checkbox"/> 110	ü <input type="checkbox"/> 126
F	<SI> <input type="checkbox"/> 15	<input type="checkbox"/> 31	/ <input type="checkbox"/> 47	? <input type="checkbox"/> 63	O <input type="checkbox"/> 79	- <input type="checkbox"/> 95	o <input type="checkbox"/> 111	☒ <input type="checkbox"/> 127

US-ASCII

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

ISO 61: Norwegian

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

ISO 4: UK

	0	1	2	3	4	5	6	7
0	0	16	32	48	64	80	96	p
1	1	17	33	49	65	81	97	q
2	2	18	34	50	66	82	98	r
3	3	19	35	51	67	83	99	s
4	4	20	36	52	68	84	100	t
5	5	21	37	53	69	85	101	u
6	6	22	38	54	70	86	102	v
7	7	23	39	55	71	87	103	w
8	<BS>	24	40	56	72	88	104	x
9	<HT>	25	41	57	73	89	105	y
A	<LF>	26	42	58	74	90	106	z
B	<ESC>	27	43	59	75	91	107	{
C	<FF>	28	44	60	76	92	108	
D	<CR>	29	45	61	77	93	109	}
E	<SO>	30	46	62	78	94	110	~
F	<SI>	31	47	63	79	95	111	⌘

ISO 69: French

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

ISO 21: German

	0	1	2	3	4	5	6	7	
0	0	16	32	0	48	64	80	96	p
1	1	17	!	1	49	A	Q	a	q
2	2	18	"	2	50	B	R	b	r
3	3	19	#	3	51	C	S	c	s
4	4	20	\$	4	52	D	T	d	t
5	5	21	%	5	53	E	U	e	u
6	6	22	&	6	54	F	V	f	v
7	7	23	'	7	55	G	W	g	w
8	<BS>	24	(8	56	H	X	h	x
9	<HT>	25)	9	57	I	Y	i	y
A	<LF>	26	*	:	58	J	Z	j	z
B	<ESC>	27	+	;	59	K	Ä	k	ä
C	<FF>	28	,	<	60	L	Ö	l	ö
D	<CR>	29	-	=	61	M	Ü	m	ü
E	<SO>	30	.	>	62	N	^	n	ß
F	<SI>	31	/	?	63	O	-	o	☒

HP Spanish

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

ISO 57: Chinese

	0	1	2	3	4	5	6	7
0	0	16	32	48	@	P	`	p
1	1	17	33	49	A	Q	a	q
2	2	18	34	50	B	R	b	r
3	3	19	35	51	C	S	c	s
4	4	20	36	52	D	T	d	t
5	5	21	37	53	E	U	e	u
6	6	22	38	54	F	V	f	v
7	7	23	39	55	G	W	g	w
8	<BS>	24	40	56	H	X	h	x
9	<HT>	25	41	57	I	Y	i	y
A	<LF>	26	42	58	J	Z	j	z
B	<ESC>	27	43	59	K	[k	{
C	<FF>	28	44	60	L	\	l	
D	<CR>	29	45	61	M]	m	}
E	<SO>	30	46	62	N	^	n	~
F	<SI>	31	47	63	O	_	o	⦿

ISO 17: Spanish

	0	1	2	3	4	5	6	7
0	<input type="text" value="0"/>	<input type="text" value="16"/>	<input type="text" value="32"/>	<input type="text" value="48"/>	<input type="text" value="64"/>	<input type="text" value="80"/>	<input type="text" value="96"/>	<input type="text" value="112"/>
1	<input type="text" value="1"/>	<input type="text" value="17"/>	<input type="text" value="33"/>	<input type="text" value="49"/>	<input type="text" value="65"/>	<input type="text" value="81"/>	<input type="text" value="97"/>	<input type="text" value="113"/>
2	<input type="text" value="2"/>	<input type="text" value="18"/>	<input type="text" value="34"/>	<input type="text" value="50"/>	<input type="text" value="66"/>	<input type="text" value="82"/>	<input type="text" value="98"/>	<input type="text" value="114"/>
3	<input type="text" value="3"/>	<input type="text" value="19"/>	<input type="text" value="35"/>	<input type="text" value="51"/>	<input type="text" value="67"/>	<input type="text" value="83"/>	<input type="text" value="99"/>	<input type="text" value="115"/>
4	<input type="text" value="4"/>	<input type="text" value="20"/>	<input type="text" value="36"/>	<input type="text" value="52"/>	<input type="text" value="68"/>	<input type="text" value="84"/>	<input type="text" value="100"/>	<input type="text" value="116"/>
5	<input type="text" value="5"/>	<input type="text" value="21"/>	<input type="text" value="37"/>	<input type="text" value="53"/>	<input type="text" value="69"/>	<input type="text" value="85"/>	<input type="text" value="101"/>	<input type="text" value="117"/>
6	<input type="text" value="6"/>	<input type="text" value="22"/>	<input type="text" value="38"/>	<input type="text" value="54"/>	<input type="text" value="70"/>	<input type="text" value="86"/>	<input type="text" value="102"/>	<input type="text" value="118"/>
7	<input type="text" value="7"/>	<input type="text" value="23"/>	<input type="text" value="39"/>	<input type="text" value="55"/>	<input type="text" value="71"/>	<input type="text" value="87"/>	<input type="text" value="103"/>	<input type="text" value="119"/>
8	<input type="text" value="8"/> <BS>	<input type="text" value="24"/>	<input type="text" value="40"/>	<input type="text" value="56"/>	<input type="text" value="72"/>	<input type="text" value="88"/>	<input type="text" value="104"/>	<input type="text" value="120"/>
9	<input type="text" value="9"/> <HT>	<input type="text" value="25"/>	<input type="text" value="41"/>	<input type="text" value="57"/>	<input type="text" value="73"/>	<input type="text" value="89"/>	<input type="text" value="105"/>	<input type="text" value="121"/>
A	<input type="text" value="10"/> <LF>	<input type="text" value="26"/>	<input type="text" value="42"/>	<input type="text" value="58"/>	<input type="text" value="74"/>	<input type="text" value="90"/>	<input type="text" value="106"/>	<input type="text" value="122"/>
B	<input type="text" value="11"/> <ESC>	<input type="text" value="27"/>	<input type="text" value="43"/>	<input type="text" value="59"/>	<input type="text" value="75"/>	<input type="text" value="91"/>	<input type="text" value="107"/>	<input type="text" value="123"/>
C	<input type="text" value="12"/> <FF>	<input type="text" value="28"/>	<input type="text" value="44"/>	<input type="text" value="60"/>	<input type="text" value="76"/>	<input type="text" value="92"/>	<input type="text" value="108"/>	<input type="text" value="124"/>
D	<input type="text" value="13"/> <CR>	<input type="text" value="29"/>	<input type="text" value="45"/>	<input type="text" value="61"/>	<input type="text" value="77"/>	<input type="text" value="93"/>	<input type="text" value="109"/>	<input type="text" value="125"/>
E	<input type="text" value="14"/> <SO>	<input type="text" value="30"/>	<input type="text" value="46"/>	<input type="text" value="62"/>	<input type="text" value="78"/>	<input type="text" value="94"/>	<input type="text" value="110"/>	<input type="text" value="126"/>
F	<input type="text" value="15"/> <SI>	<input type="text" value="31"/>	<input type="text" value="47"/>	<input type="text" value="63"/>	<input type="text" value="79"/>	<input type="text" value="95"/>	<input type="text" value="111"/>	<input type="text" value="127"/>

ISO 2: IRV

	0	1	2	3	4	5	6	7
0	0	16	32	48	@	P	`	p
1	1	17	33	49	A	Q	a	q
2	2	18	34	50	B	R	b	r
3	3	19	35	51	C	S	c	s
4	4	20	36	52	D	T	d	t
5	5	21	37	53	E	U	e	u
6	6	22	38	54	F	V	f	v
7	7	23	39	55	G	W	g	w
8	<BS>	24	40	56	H	X	h	x
9	<HT>	25	41	57	I	Y	i	y
A	<LF>	26	42	58	J	Z	j	z
B	<ESC>	27	43	59	K	[k	{
C	<FF>	28	44	60	L	\	l	
D	<CR>	29	45	61	M]	m	}
E	<SO>	30	46	62	N	^	n	-
F	<SI>	31	47	63	O	_	o	⦿

ISO 10: Swedish

	0	1	2	3	4	5	6	7	
0	0	16	32	0	48	ø	P	˘	p
1	1	17	33	1	49	A	Q	a	q
2	2	18	34	2	50	B	R	b	r
3	3	19	35	3	51	C	S	c	s
4	4	20	36	4	52	D	T	d	t
5	5	21	37	5	53	E	U	e	u
6	6	22	38	6	54	F	V	f	v
7	7	23	39	7	55	G	W	g	w
8	<BS>	24	40	8	56	H	X	h	x
9	<HT>	25	41	9	57	I	Y	i	y
A	<LF>	26	42	:	58	J	Z	j	z
B	<ESC>	27	43	;	59	K	Ä	k	ä
C	<FF>	28	44	<	60	L	Ö	l	ö
D	<CR>	29	45	=	61	M	Å	m	å
E	<SO>	30	46	>	62	N	ˆ	n	ˆ
F	<SI>	31	47	?	63	O	—	o	☒

ISO 16: Portuguese

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

ISO 84: Portuguese

	0	1	2	3	4	5	6	7
0	0	16	32	48	64	P	96	p
1	1	17	33	49	A	Q	a	q
2	2	18	34	50	B	R	b	r
3	3	19	35	51	C	S	c	s
4	4	20	36	52	D	T	d	t
5	5	21	37	53	E	U	e	u
6	6	22	38	54	F	V	f	v
7	7	23	39	55	G	W	g	w
8	<BS>	24	40	56	H	X	h	x
9	<HT>	25	41	57	I	Y	i	y
A	<LF>	26	42	58	J	Z	j	z
B	<ESC>	27	43	59	K	Ã	k	ã
C	<FF>	28	44	60	L	Ç	l	ç
D	<CR>	29	45	61	M	Õ	m	õ
E	<SO>	30	46	62	N	^	n	~
F	<SI>	31	47	63	O	-	o	☒

ISO 85: Spanish

	0	1	2	3	4	5	6	7
0	0	16	32	0	·	P	`	p
1	1	17	!	1	A	Q	a	q
2	2	18	"	2	B	R	b	r
3	3	19	#	3	C	S	c	s
4	4	20	\$	4	D	T	d	t
5	5	21	%	5	E	U	e	u
6	6	22	&	6	F	V	f	v
7	7	23	'	7	G	W	g	w
8	<BS>	24	(8	H	X	h	x
9	<HT>	25)	9	I	Y	i	y
A	<LF>	26	*	:	J	Z	j	z
B	<ESC>	27	+	;	K	ı	k	´
C	<FF>	28	,	<	L	Ñ	l	ñ
D	<CR>	29	-	=	M	Ç	m	ç
E	<SO>	30	·	>	N	¿	n	¨
F	<SI>	31	/	?	O	—	o	⌘
	15	31	47	63	79	95	111	127

Roman-8

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

Roman-8

	8	9	A	B	C	D	E	F
0	128	144	160	176	192	208	224	240
1	129	145	À	Ý	ê	î	Ã	þ
2	130	146	Â	Ý	ô	ø	ã	·
3	131	147	È	°	û	Æ	Ð	μ
4	132	148	Ê	Ç	á	å	Ö	¶
5	133	149	Ë	Ç	é	í	Í	$\frac{3}{4}$
6	134	150	Î	Ñ	ó	ø	Ï	-
7	135	151	Ï	ñ	ú	æ	Ó	$\frac{1}{4}$
8	136	152	ˆ	i	à	Ä	Ò	$\frac{1}{2}$
9	137	153	˘	ı	è	ì	Õ	æ
A	138	154	ˆ	α	ò	Ö	õ	ø
B	139	155	˙	£	ù	Ü	š	«
C	140	156	˘	¥	ä	É	š	■
D	141	157	Û	§	ë	ï	Ú	»
E	142	158	Û	f	ö	ß	ÿ	±
F	143	159	£	Ç	ü	Ô	ÿ	

IBM-PC(US)

	0	1	2	3	4	5	6	7
0	<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="@"/>	<input type="text" value="P"/>	<input type="text" value="´"/>	<input type="text" value="p"/>
1	<input type="text" value="!"/>	<input type="text" value="1"/>	<input type="text" value="!"/>	<input type="text" value="1"/>	<input type="text" value="A"/>	<input type="text" value="Q"/>	<input type="text" value="a"/>	<input type="text" value="q"/>
2	<input type="text" value="\"/>	<input type="text" value="2"/>	<input type="text" value="\"/>	<input type="text" value="2"/>	<input type="text" value="B"/>	<input type="text" value="R"/>	<input type="text" value="b"/>	<input type="text" value="r"/>
3	<input type="text" value="#"/>	<input type="text" value="3"/>	<input type="text" value="#"/>	<input type="text" value="3"/>	<input type="text" value="C"/>	<input type="text" value="S"/>	<input type="text" value="c"/>	<input type="text" value="s"/>
4	<input type="text" value="\$"/>	<input type="text" value="4"/>	<input type="text" value="\$"/>	<input type="text" value="4"/>	<input type="text" value="D"/>	<input type="text" value="T"/>	<input type="text" value="d"/>	<input type="text" value="t"/>
5	<input type="text" value="%"/>	<input type="text" value="5"/>	<input type="text" value="%"/>	<input type="text" value="5"/>	<input type="text" value="E"/>	<input type="text" value="U"/>	<input type="text" value="e"/>	<input type="text" value="u"/>
6	<input type="text" value="&"/>	<input type="text" value="6"/>	<input type="text" value="&"/>	<input type="text" value="6"/>	<input type="text" value="F"/>	<input type="text" value="V"/>	<input type="text" value="f"/>	<input type="text" value="v"/>
7	<input type="text" value="´"/>	<input type="text" value="7"/>	<input type="text" value="´"/>	<input type="text" value="7"/>	<input type="text" value="G"/>	<input type="text" value="W"/>	<input type="text" value="g"/>	<input type="text" value="w"/>
8	<input type="text" value="("/>	<input type="text" value="8"/>	<input type="text" value="("/>	<input type="text" value="8"/>	<input type="text" value="H"/>	<input type="text" value="X"/>	<input type="text" value="h"/>	<input type="text" value="x"/>
9	<input type="text" value=")"/>	<input type="text" value="9"/>	<input type="text" value=")"/>	<input type="text" value="9"/>	<input type="text" value="I"/>	<input type="text" value="Y"/>	<input type="text" value="i"/>	<input type="text" value="y"/>
A	<input type="text" value="*"/>	<input type="text" value=":"/>	<input type="text" value="*"/>	<input type="text" value=":"/>	<input type="text" value="J"/>	<input type="text" value="Z"/>	<input type="text" value="j"/>	<input type="text" value="z"/>
B	<input type="text" value="+"/> <input type="text" value=";"/>	<input type="text" value="+"/> <input type="text" value=";"/>	<input type="text" value="+"/> <input type="text" value=";"/>	<input type="text" value="+"/> <input type="text" value=";"/>	<input type="text" value="K"/>	<input type="text" value="["/> <input type="text" value="k"/>	<input type="text" value="["/> <input type="text" value="k"/>	<input type="text" value="{"/> <input type="text" value="{"/>
C	<input type="text" value=","/> <input type="text" value="<"/>	<input type="text" value=","/> <input type="text" value="<"/>	<input type="text" value=","/> <input type="text" value="<"/>	<input type="text" value=","/> <input type="text" value="<"/>	<input type="text" value="L"/>	<input type="text" value="\"/> <input type="text" value="l"/>	<input type="text" value="\"/> <input type="text" value="l"/>	<input type="text" value=" "/> <input type="text" value=" "/>
D	<input type="text" value="-"/> <input type="text" value="="/>	<input type="text" value="-"/> <input type="text" value="="/>	<input type="text" value="-"/> <input type="text" value="="/>	<input type="text" value="-"/> <input type="text" value="="/>	<input type="text" value="M"/>	<input type="text" value="]"/> <input type="text" value="m"/>	<input type="text" value="]"/> <input type="text" value="m"/>	<input type="text" value="}"/> <input td="" type="text" value="}" }}<=""/>
E	<input type="text" value="."/> <input type="text" value=">"/>	<input type="text" value="."/> <input type="text" value=">"/>	<input type="text" value="."/> <input type="text" value=">"/>	<input type="text" value="."/> <input type="text" value=">"/>	<input type="text" value="N"/>	<input type="text" value="^"/> <input type="text" value="n"/>	<input type="text" value="^"/> <input type="text" value="n"/>	<input type="text" value="~"/> <input type="text" value="~"/>
F	<input type="text" value="/"/> <input type="text" value="?"/>	<input type="text" value="/"/> <input type="text" value="?"/>	<input type="text" value="/"/> <input type="text" value="?"/>	<input type="text" value="/"/> <input type="text" value="?"/>	<input type="text" value="O"/>	<input type="text" value="_"/> <input type="text" value="o"/>	<input type="text" value="_"/> <input type="text" value="o"/>	<input type="text" value="△"/> <input type="text" value="△"/>

IBM-PC(US)

	8	9	A	B	C	D	E	F
0	Ç 128	É 144	á 160	⋮ 176	Ł 192	⋈ 208	α 224	≡ 240
1	ü 129	æ 145	í 161	⋱ 177	⊥ 193	⌒ 209	β 225	± 241
2	é 130	Æ 146	ó 162	⋴ 178	⌞ 194	π 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	ƒ 213	σ 229	∫ 245
6	å 134	û 150	ä 166	⌣ 182	ƒ 198	π 214	μ 230	÷ 246
7	ç 135	ù 151	ó 167	π 183	⌣ 199	⌣ 215	τ 231	≈ 247
8	ê 136	ÿ 152	ç 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	Ç 155	½ 171	π 187	⌒ 203	■ 219	δ 235	√ 251
C	î 140	£ 156	¼ 172	⌣ 188	⌣ 204	■ 220	∞ 236	ⁿ 252
D	ì 141	¥ 157	ı 173	⋈ 189	= 205	■ 221	φ 237	² 253
E	Ä 142	℞ 158	« 174	∫ 190	⌣ 206	■ 222	€ 238	▪ 254
F	Å 143	f 159	» 175	∫ 191	± 207	■ 223	∩ 239	

IBM-PC(Denmark/Norway)

	0	1	2	3	4	5	6	7
0	<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="@"/>	<input type="text" value="P"/>	<input type="text" value="´"/>	<input type="text" value="p"/>
1	<input type="text" value="!"/>	<input type="text" value="1"/>	<input type="text" value="!"/>	<input type="text" value="1"/>	<input type="text" value="A"/>	<input type="text" value="Q"/>	<input type="text" value="a"/>	<input type="text" value="q"/>
2	<input type="text" value="\"/>	<input type="text" value="2"/>	<input type="text" value="\"/>	<input type="text" value="2"/>	<input type="text" value="B"/>	<input type="text" value="R"/>	<input type="text" value="b"/>	<input type="text" value="r"/>
3	<input type="text" value="#"/>	<input type="text" value="3"/>	<input type="text" value="#"/>	<input type="text" value="3"/>	<input type="text" value="C"/>	<input type="text" value="S"/>	<input type="text" value="c"/>	<input type="text" value="s"/>
4	<input type="text" value="\$"/>	<input type="text" value="4"/>	<input type="text" value="\$"/>	<input type="text" value="4"/>	<input type="text" value="D"/>	<input type="text" value="T"/>	<input type="text" value="d"/>	<input type="text" value="t"/>
5	<input style="width: 20px; height: 20px; border: 1px solid black; display: flex; align-items: center; justify-content: center; font-size: 10px; font-weight: bold;" type="text" value="%"/> 5	<input type="text" value="5"/>	<input style="width: 20px; height: 20px; border: 1px solid black; display: flex; align-items: center; justify-content: center; font-size: 10px; font-weight: bold;" type="text" value="%"/> 5	<input type="text" value="5"/>	<input type="text" value="E"/>	<input type="text" value="U"/>	<input type="text" value="e"/>	<input type="text" value="u"/>
6	<input type="text" value="&"/>	<input type="text" value="6"/>	<input type="text" value="&"/>	<input type="text" value="6"/>	<input type="text" value="F"/>	<input type="text" value="V"/>	<input type="text" value="f"/>	<input type="text" value="v"/>
7	<input type="text" value="´"/>	<input type="text" value="7"/>	<input type="text" value="´"/>	<input type="text" value="7"/>	<input type="text" value="G"/>	<input type="text" value="W"/>	<input type="text" value="g"/>	<input type="text" value="w"/>
8	<input style="width: 20px; height: 20px; border: 1px solid black; display: flex; align-items: center; justify-content: center; font-size: 10px; font-weight: bold;" type="text" value="("/> 8	<input type="text" value="8"/>	<input style="width: 20px; height: 20px; border: 1px solid black; display: flex; align-items: center; justify-content: center; font-size: 10px; font-weight: bold;" type="text" value="("/> 8	<input type="text" value="8"/>	<input type="text" value="H"/>	<input type="text" value="X"/>	<input type="text" value="h"/>	<input type="text" value="x"/>
9	<input type="text" value=")"/>	<input type="text" value="9"/>	<input type="text" value=")"/>	<input type="text" value="9"/>	<input type="text" value="I"/>	<input type="text" value="Y"/>	<input type="text" value="i"/>	<input type="text" value="y"/>
A	<input type="text" value="*"/>	<input type="text" value=":"/>	<input type="text" value="*"/>	<input type="text" value=":"/>	<input type="text" value="J"/>	<input type="text" value="Z"/>	<input type="text" value="j"/>	<input type="text" value="z"/>
B	<input style="width: 20px; height: 20px; border: 1px solid black; display: flex; align-items: center; justify-content: center; font-size: 10px; font-weight: bold;" type="text" value="+"/> 11	<input type="text" value=";"/>	<input style="width: 20px; height: 20px; border: 1px solid black; display: flex; align-items: center; justify-content: center; font-size: 10px; font-weight: bold;" type="text" value="+"/> 11	<input type="text" value=";"/>	<input type="text" value="K"/>	<input type="text" value="["/>	<input type="text" value="k"/>	<input type="text" value="{"/>
C	<input type="text" value="´"/>	<input type="text" value="<"/>	<input type="text" value="´"/>	<input type="text" value="<"/>	<input type="text" value="L"/>	<input type="text" value="\"/>	<input type="text" value="l"/>	<input type="text" value=" "/>
D	<input type="text" value="-"/>	<input style="width: 20px; height: 20px; border: 1px solid black; display: flex; align-items: center; justify-content: center; font-size: 10px; font-weight: bold;" type="text" value="="/> 13	<input type="text" value="-"/>	<input style="width: 20px; height: 20px; border: 1px solid black; display: flex; align-items: center; justify-content: center; font-size: 10px; font-weight: bold;" type="text" value="="/> 13	<input type="text" value="M"/>	<input type="text" value="]"/>	<input type="text" value="m"/>	<input type="text" value="}"/>
E	<input type="text" value="·"/>	<input type="text" value=">"/>	<input type="text" value="·"/>	<input type="text" value=">"/>	<input type="text" value="N"/>	<input type="text" value="^"/>	<input type="text" value="n"/>	<input type="text" value="˘"/>
F	<input style="width: 20px; height: 20px; border: 1px solid black; display: flex; align-items: center; justify-content: center; font-size: 10px; font-weight: bold;" type="text" value="/"/> 15	<input style="width: 20px; height: 20px; border: 1px solid black; display: flex; align-items: center; justify-content: center; font-size: 10px; font-weight: bold;" type="text" value="?"/> 31	<input style="width: 20px; height: 20px; border: 1px solid black; display: flex; align-items: center; justify-content: center; font-size: 10px; font-weight: bold;" type="text" value="/"/> 15	<input style="width: 20px; height: 20px; border: 1px solid black; display: flex; align-items: center; justify-content: center; font-size: 10px; font-weight: bold;" type="text" value="?"/> 31	<input type="text" value="O"/>	<input type="text" value="‑"/>	<input type="text" value="o"/>	<input type="text" value="△"/>

IBM-PC(Denmark/Norway)

	8	9	A	B	C	D	E	F
0	Ç 128	É 144	á 160	⋮ 176	Ł 192	⋈ 208	α 224	≡ 240
1	ü 129	æ 145	í 161	⋯ 177	⊥ 193	⌒ 209	β 225	± 241
2	é 130	Æ 146	ó 162	⋱ 178	⌞ 194	π 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	ƒ 213	σ 229	J 245
6	å 134	û 150	õ 166	‖ 182	‡ 198	π 214	μ 230	÷ 246
7	ç 135	ù 151	Õ 167	π 183	‖ 199	‡ 215	τ 231	≈ 247
8	ê 136	ÿ 152	ı 168	ƒ 184	⋈ 200	‡ 216	Φ 232	° 248
9	ë 137	Ö 153	ã 169	‖ 185	ƒ 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	π 187	⌒ 203	■ 219	δ 235	√ 251
C	î 140	£ 156	ñ 172	⋈ 188	‖ 204	■ 220	∞ 236	ⁿ 252
D	ì 141	Ø 157	i 173	⋈ 189	= 205	■ 221	φ 237	² 253
E	Ä 142	Ł 158	³ 174	∩ 190	‖ 206	■ 222	€ 238	▪ 254
F	Å 143	ł 159	⁴ 175	∩ 191	⊥ 207	■ 223	∩ 239	

PC-850









	0	1	2	3	4	5	6	7
0	<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="@"/>	<input type="text" value="P"/>	<input type="text" value="´"/>	<input type="text" value="p"/>
1	<input type="text" value="1"/>	<input type="text" value="1"/>	<input type="text" value="1"/>	<input type="text" value="1"/>	<input type="text" value="A"/>	<input type="text" value="Q"/>	<input type="text" value="a"/>	<input type="text" value="q"/>
2	<input type="text" value="2"/>	<input type="text" value="2"/>	<input type="text" value="2"/>	<input type="text" value="2"/>	<input type="text" value="B"/>	<input type="text" value="R"/>	<input type="text" value="b"/>	<input type="text" value="r"/>
3	<input type="text" value="3"/>	<input type="text" value="3"/>	<input type="text" value="3"/>	<input type="text" value="3"/>	<input type="text" value="C"/>	<input type="text" value="S"/>	<input type="text" value="c"/>	<input type="text" value="s"/>
4	<input type="text" value="4"/>	<input type="text" value="4"/>	<input type="text" value="4"/>	<input type="text" value="4"/>	<input type="text" value="D"/>	<input type="text" value="T"/>	<input type="text" value="d"/>	<input type="text" value="t"/>
5	<input type="text" value="5"/>	<input type="text" value="5"/>	<input type="text" value="5"/>	<input type="text" value="5"/>	<input type="text" value="E"/>	<input type="text" value="U"/>	<input type="text" value="e"/>	<input type="text" value="u"/>
6	<input type="text" value="6"/>	<input type="text" value="6"/>	<input type="text" value="6"/>	<input type="text" value="6"/>	<input type="text" value="F"/>	<input type="text" value="V"/>	<input type="text" value="f"/>	<input type="text" value="v"/>
7	<input type="text" value="7"/>	<input type="text" value="7"/>	<input type="text" value="7"/>	<input type="text" value="7"/>	<input type="text" value="G"/>	<input type="text" value="W"/>	<input type="text" value="g"/>	<input type="text" value="w"/>
8	<input type="text" value="8"/>	<input type="text" value="8"/>	<input type="text" value="8"/>	<input type="text" value="8"/>	<input type="text" value="H"/>	<input type="text" value="X"/>	<input type="text" value="h"/>	<input type="text" value="x"/>
9	<input type="text" value="9"/>	<input type="text" value="9"/>	<input type="text" value="9"/>	<input type="text" value="9"/>	<input type="text" value="I"/>	<input type="text" value="Y"/>	<input type="text" value="i"/>	<input type="text" value="y"/>
A	<input type="text" value="*"/>	<input type="text" value=":"/>	<input type="text" value="*"/>	<input type="text" value=":"/>	<input type="text" value="J"/>	<input type="text" value="Z"/>	<input type="text" value="j"/>	<input type="text" value="z"/>
B	<input type="text" value="+"/>	<input type="text" value=";"/>	<input type="text" value="+"/>	<input type="text" value=";"/>	<input type="text" value="K"/>	<input type="text" value="["/>	<input type="text" value="k"/>	<input type="text" value="{"/>
C	<input type="text" value="´"/>	<input type="text" value="<"/>	<input type="text" value="´"/>	<input type="text" value="<"/>	<input type="text" value="L"/>	<input type="text" value="\"/>	<input type="text" value="l"/>	<input type="text" value=" "/>
D	<input type="text" value="-"/>	<input type="text" value="="/>	<input type="text" value="-"/>	<input type="text" value="="/>	<input type="text" value="M"/>	<input type="text" value="]"/>	<input type="text" value="m"/>	<input style="font-size: 0.8em; vertical-align: middle;" type="text" value="}"/>
E	<input type="text" value="·"/>	<input type="text" value=">"/>	<input type="text" value="·"/>	<input type="text" value=">"/>	<input type="text" value="N"/>	<input type="text" value="^"/>	<input type="text" value="n"/>	<input type="text" value="˘"/>
F	<input style="font-size: 0.8em; vertical-align: middle;" type="text" value="/"/>	<input style="font-size: 0.8em; vertical-align: middle;" type="text" value="?"/>	<input style="font-size: 0.8em; vertical-align: middle;" type="text" value="/"/>	<input style="font-size: 0.8em; vertical-align: middle;" type="text" value="?"/>	<input type="text" value="O"/>	<input type="text" value="‑"/>	<input type="text" value="o"/>	<input type="text" value="△"/>

PC-850

	8	9	A	B	C	D	E	F
0	Ç 128	É 144	á 160	⋮ 176	Ł 192	ø 208	Ó 224	- 240
1	ü 129	æ 145	í 161	⋮ 177	⊥ 193	Ð 209	β 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	$\frac{3}{4}$ 243
4	ä 132	ö 148	ñ 164	⊥ 180	- 196	È 212	ō 228	¶ 244
5	à 133	ò 149	Ñ 165	Á 181	⊥ 197	ı 213	Õ 229	§ 245
6	å 134	û 150	ä 166	Â 182	ã 198	Î 214	μ 230	÷ 246
7	ç 135	ù 151	å 167	Ã 183	Ä 199	Ï 215	þ 231	˘ 247
8	ê 136	ÿ 152	ç 168	© 184	Ł 200	ÿ 216	ƒ 232	° 248
9	ë 137	ö 153	® 169	185	⊥ 201	⊥ 217	Û 233	¨ 249
A	è 138	ü 154	¬ 170	186	⊥ 202	⊥ 218	Û 234	· 250
B	ï 139	ø 155	$\frac{1}{2}$ 171	⊥ 187	⊥ 203	■ 219	Û 235	¹ 251
C	î 140	£ 156	$\frac{1}{4}$ 172	⊥ 188	⊥ 204	■ 220	Ÿ 236	³ 252
D	ì 141	ø 157	i 173	ç 189	= 205	221	Ÿ 237	² 253
E	Ä 142	× 158	« 174	¥ 190	⊥ 206	Ï 222	- 238	■ 254
F	Å 143	f 159	» 175	⊥ 191	⊥ 207	■ 223	- 239	■ 255

7.3 Resident font samples

PCL5 fonts

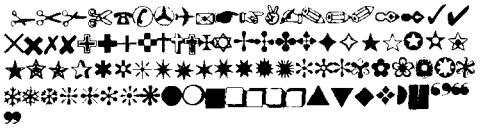
Courier 12-point (10 cpi)	!"#\$%&'()*+,-./0123456789: ;<=>?@ABCDEFGHIJKLMNOPQRST UVWXYZ[\]^_`abcdefghijklmnop opqrstuvwxyz{ }~ 
Courier Bold 12-point (10 cpi)	!"#\$%&'()*+,-./0123456789: ;<=>?@ABCDEFGHIJKLMNOPQRST UVWXYZ[\]^_`abcdefghijklmnop opqrstuvwxyz{ }~ 
Courier Italic 12-point (10 cpi)	!"#\$%&'()*+,-./0123456789: ;<=>?@ABCDEFGHIJKLMNOPQRST UVWXYZ[\]^_`abcdefghijklmnop opqrstuvwxyz{ }~ 
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;<=>@ABCDEFGHIJK LMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuv wxyz{ }~ 
Univers Medium	!"#\$%&'()*+,-./0123456789;<=> ?@ABCDEFGHIJKLMNOPQRSTUVWXYZ [\]^_`abcdefghijklmnopqrstuvwxyz{ }~ 

Univers Medium Italic	!"#\$%&'()*+,-./0123456789:;<=>? @ABCDEFGHIJKLMN ^O QRSTUVWXYZ [\]^_`abcdefghijklmnopqrstuvwxyz{ }~ ☒
Univers Bold	!"#\$%&'()*+,-./0123456789:;<=>? @ABCDEFGHIJKLMN^OQRSTUVWXYZ [\]^_`abcdefghijklmnopqrstuvwxyz{ }~ ☒
Univers Bold Italic	!"#\$%&'()*+,-./0123456789:;<=>? @ABCDEFGHIJKLMN^OQRSTUVWXYZ [\]^_`abcdefghijklmnopqrstuvwxyz{ }~ ☒
CG Times	!"#\$%&'()*+,-./0123456789:;<=>?@AB CDEFGHIJKLMN ^O QRSTUVWXYZ[\]^_` abcdefghijklmnopqrstuvwxyz{ }~☒
CG Times Italic	!"#\$%&'()*+,-./0123456789:;<=>?@AB <i>CDEFGHIJKLMN^OQRSTUVWXYZ[\]^_` abcdefghijklmnopqrstuvwxyz{ }~☒</i>
CG Times Bold	!"#\$%&'()*+,-./0123456789:;<=>?@AB CDEFGHIJKLMN^OQRSTUVWXYZ[\]^_` abcdefghijklmnopqrstuvwxyz{ }~☒
CG Times Bold Italic	!"#\$%&'()*+,-./0123456789:;<=>?@AB <i>CDEFGHIJKLMN^OQRSTUVWXYZ[\]^_` abcdefghijklmnopqrstuvwxyz{ }~☒</i>

Truelmage fonts

Arial	!"#\$%&'()*+,-./0123456789:;<=>?@AB CDEFGHIJKLMNOPQRSTUVWXYZ[\]^ _`abcdefghijklmnopqrstuvwxyz{ }~
Arial Bold	!"#\$%&'()*+,-./0123456789:;<=>?@AB CDEFGHIJKLMNOPQRSTUVWXYZ[\]^ _`abcdefghijklmnopqrstuvwxyz{ }~
Arial Bold Oblique	!"#\$%&'()*+,-./0123456789:;<=>?@AB CDEFGHIJKLMNOPQRSTUVWXYZ[\]^ _`abcdefghijklmnopqrstuvwxyz{ }~
Arial Oblique	!"#\$%&'()*+,-./0123456789:;<=>?@AB CDEFGHIJKLMNOPQRSTUVWXYZ[\]^ _`abcdefghijklmnopqrstuvwxyz{ }~
Arial Narrow	!"#\$%&'()*+,-./0123456789:;<=>?@ABCDEFGHI JKLMNOPQRSTUVWXYZ[\]^`_abcdefghijklmnopqrstuvwxyz qrstuvwxyz{ }~
Arial Narrow Bold	!"#\$%&'()*+,-./0123456789:;<=>?@ABCDEFGH IJKLMNOPQRSTUVWXYZ[\]^`_abcdefghijklmnopqrstuvwxyz mnopqrstuvwxyz{ }~
Arial Narrow Bold Oblique	!"#\$%&'()*+,-./0123456789:;<=>?@ABCDEFGH IJKLMNOPQRSTUVWXYZ[\]^`_abcdefghijklmnopqrstuvwxyz mnopqrstuvwxyz{ }~
Arial Narrow Oblique	!"#\$%&'()*+,-./0123456789:;<=>?@ABCDEFGHI JKLMNOPQRSTUVWXYZ[\]^`_abcdefghijklmnopqrstuvwxyz qrstuvwxyz{ }~
Century Schoolbook Bold	!"#\$%&'()*+,-./0123456789:;<=>?@AB CDEFGHIJKLMNOPQRSTUVWXYZ[\]^`_abcdefghijklmnopqrstuvwxyz yz{ }~
Century Schoolbook Bold Italic	!"#\$%&'()*+,-./0123456789:;<=>?@A BCDEFGHIJKLMNOPQRSTUVWXYZ[\]^`_abcdefghijklmnopqrstuvwxyz xyz{ }~

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

ITC Bookman Demi	!"#\$%&'()*+,-./0123456789;:<=>? @ABCDEFGHIJKLMNopQRSTUVW XYZ[\]^_`'abcdefghijklmnopqrstuv wxyz{ }~
ITC Bookman Demi Italic	!"#\$%&'()*+,-./0123456789;:<=>? @ABCDEFGHIJKLMNopQRSTUVW XYZ[\]^_`'abcdefghijklmnopqrstuv wxyz{ }~
ITC Bookman Light	!"#\$%&'()*+,-./0123456789;:<=>?@ ABCDEFGHIJKLMNopQRSTUVWXY Z[\]^_`'abcdefghijklmnopqrstuvwxyz{ }~
ITC Bookman Light Italic	!"#\$%&'()*+,-./0123456789;:<=>?@A BCDEFGHIJKLMNopQRSTUVWXYZ[\]^_`'abcdefghijklmnopqrstuvwxyz{ } ~
ITC Zapf Chancery Medium Italic	!"#\$%&'()*+,-./0123456789;:<=>?@ABCDEF G HIJKLMNopQRSTUVWxyz[\]^_`'abcdefghijk lmnopqrstuvwxyz{ }~
ITC Zapf Dingbats	
Symbol	!∀#∃%&∞()*+,-./0123456789;:<=>?@ABX ΔΕΦΓΗΙΘΚΛΜΝΟΠΘΡΣΤΥζΩΞΨΖ[.:]⊥_ αβχδεφγηηιφκλμνοπθρστυπωξψζ{ }~
Times New Roman	!"#\$%&'()*+,-./0123456789;:<=>?@ABC DEFGHIJKLMNopQRSTUVWXYZ[\]^_`a bcdefghijklmnopqrstuvwxyz{ }~
Times New Roman Bold	!"#\$%&'()*+,-./0123456789;:<=>?@AB CDEFGHIJKLMNopQRSTUVWXYZ[\]^_`'abcdefghijklmnopqrstuvwxyz{ }~
Times New Roman Bold Italic	!"#\$%&'()*+,-./0123456789;:<=>?@ABC DEFGHIJKLMNopQRSTUVWXYZ[\]^_` 'abcdefghijklmnopqrstuvwxyz{ }~

Times New Roman Italic	<i>!"#\$%&'()*+,-./0123456789;:<=>?@ABC DEFGHIJKLMNOPQRSTUVWXYZ[\]^_`ab cdefghijklmnopqrstuvwxyz{ }~</i>
Zapf Calligraphic Bold	!"#\$%&'()*+,-./0123456789;:<=>?@ABC DEFGHIJKLMNOPQRSTUVWXYZ[\]^ ^_`abcdefghijklmnopqrstuvwxyz{ }~
Zapf Calligraphic Bold Italic	<i>!"#\$%&'()*+,-./0123456789;:<=>?@ABC DEFGHIJKLMNOPQRSTUVWXYZ[\]^ _`abcdefghijklmnopqrstuvwxyz{ }~</i>
Zapf Calligraphic Italic	<i>!"#\$%&'()*+,-./0123456789;:<=>?@ABC DEFGHIJKLMNOPQRSTUVWXYZ[\]^_ `abcdefghijklmnopqrstuvwxyz{ }~</i>
Zapf Calligraphic Roman	!"#\$%&'()*+,-./0123456789;:<=>?@ABC DEFGHIJKLMNOPQRSTUVWXYZ[\]^ _`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 coordinates 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 mono-spaced 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 limit 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. <i>Serif</i> typeface characters have small curly hooks (serifs) that serve to make <i>serif</i> -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

A

Alternate font (GL2), 171
Anchor corner, 157
Anchor point, 117
Array, 201, 208
Array operators, 245–246
Ascender, 24
Automatic downloading, 36

B

Backspace, 68
Baseline, 24
Binary, 4
Bitmap fonts, 30, 75
Boolean, 201, 208
Bounding box, 217
Buffer, 43

C

Caching, 214
Carriage return, 67
Cartridge, 2
Cartridge fonts, 33
CD-ROM, 2, 39, 75
Character code, 98
Character encoding, 218
Character encoding (TrueImage), 216
Character features, 24
Character group commands, 171–191
Character spacing, 78
Characters
 special, 35
Clipping path, 222
Columns, 69
Configuration and status group commands, 127–138

Control codes, 40, 67, 68
 Backspace, 68
 Carriage return, 67
 Form feed, 68
 Horizontal tab, 68
 Line feed, 67
 Space, 67
Control operators, 253–255
Control panel, 9
 setting parameters, 12
Coordinate operators, 280–282
Coordinate system (GL2)
 rotating, 136
Coordinate system (PCL), 47
Coordinate system (TrueImage), 198
CTM, 198, 199
Current path, 196
Current settings, 11
Current transformation matrix, 198, 199
Current units, 117
Cursor positioning commands, 67–71

D

Data LED, 9
Decipoints, 69, 70
Delta row compression, 110
Descender, 24
Device set-up operators, 283
Device space, 198
Dictionaries
 errordict, 210
 statusdict, 284
 systemdict, 206
 userdict, 206
Dictionary, 202, 208
Dictionary operators, 249–252
Dictionary stack, 206
Document design, 26

Dots, 47, 69, 70
Downloaded fonts, 33, 75

E

Effective window, 119
Emulations, 2
End of line wrap, 73
ERROR SKIP button, 10
Errordict, 210
Errors, 210
Escape sequences, 40
Execution of objects, 208
Execution stack, 206

F

Factory default environment, 48
Factory settings, 11
FEEDER SELECT button, 10
File, 203, 208
File operators, 265–269
Fill type (GL2), 158
Filling paths, 220
Font attributes, 27
Font cache operators, 263
Font caching, 214
Font descriptor, 92
Font dictionaries, 214
Font Downloader utility, 36
Font height, 28
Font location, 79
Font metrics, 217
Font operators, 260–262
Font orientation, 79
Font pitch, 28
Font posture, 29
Font selection (GL2), 176–177
Font selection (PCL), 77
Font selection commands, 80–88
Font selection examples, 89
Font selection from control panel, 12
Font stroke weight, 79
Font style, 78
Font symbol set, 29
Font typeface, 79
Font weight, 28
Font width, 29

FontID, 203, 208
Fonts, 23, 211
 bitmap, 75
 bitmap fonts, 30
 cartridge fonts, 33
 character spacing, 78
 creating, 90
 downloaded, 33, 75
 downloading, 90
 automatic, 36
 manual, 36
 GL2 alternate font, 171
 GL2 standard font, 171
 monospaced, 27
 PCL fonts, 75
 pitch, 78
 PostScript type 1, 32, 211
 PostScript type 3, 32, 211
 primary font, 76
 proportionally-spaced, 27
 resident fonts, 33
 resident printer fonts, 30
 scalable, 75
 scalable fonts, 30
 secondary font, 76
 soft fonts, 33
 symbol set, 81, 82
Form feed, 68

G

GL2, 117
GL2 graphics commands, 127
GL2 mode
 entering, 122
GL2 pen, 118
GL2 syntax, 123
Graphics (PCL), 101
Graphics (TrueImage), 220
Graphics state, 197, 199
Graphics state stack, 206

H

Half-tone screen, 220
Hex dump mode, 22
Hexadecimal, 4
Horizontal tab, 68

I

Importing images (TrueImage), 222
Initial settings, 11
Input window, 118, 137
Interface settings, 17
Interpreter, 200

J

Job control commands, 52–54

L

Label, 178
Label origin, 179
Line and fill attributes group commands, 157–170
Line end type (GL2), 160
Line feed, 67
Line join type (GL2), 160
Line join type (TrueImage), 274
Line termination, 73
Line type (GL2), 162
Logical operators, 228–230
Logical page, 44
Lost mode, 126

M

Macro commands, 114
Macro overlay environment, 51
Macros, 112
 defining, 113
 running, 113
Manual downloading, 36
Margins and line spacing commands, 62–66
Mark, 203, 208
Maths operators, 225–227
Miscellaneous operators, 271–279
Miter limit, 160, 161
MODE button, 10
Modified print environment, 49
Name, 202, 208

N

Null, 208
Number systems, 4

O

Objects, 200, 201–203
 execution, 208
ON-LINE button, 9
On-line LED, 9
Operand stack, 204
Operators, 223

P

Packed array, 201, 208
Packed array operators, 247–248
Page definition commands, 55–61
Painting operators, 238–240
Paragraph styles, 26
Path, 196
Path construction operators, 231–237
Paths
 filling, 220
Pattern transparency, 102, 103
PCL, 2, 39
 programming in PCL, 41
PCL command syntax, 42, 43
PCL coordinate system, 47
PCL decipoints, 69, 70
PCL fonts, 75
PCL graphics, 101
PCL macros, 112
PCL mode
 entering, 122
PCL raster graphics, 107
PCL rectangle graphics, 105
Pen, 118
Pen movement
 absolute, 118, 140
 relative, 118, 141
Pen width, 164
Physical page, 44
Picture frame, 44, 117
Pitch, 28, 78
Plot size, 121, 122

- Plotter units, 117
- Polygon buffer, 149
- Polygon group commands, 149–156
- Polygon mode, 149
- PostScript, 2, 32, 193
- Posture, 29
- Primary font, 76
- PRINT button, 9
- Print model (PCL), 101
- Print model (TrueImage), 196
- Printer Control Language, 39
- Printing process, 1
- Printing to disk, 194
- Procedures, 194, 206
- PROGRAM button, 10
- Program mode, 8
- Programming, 41, 125
- Programming in GL2, 125

R

- Raster graphics, 107
- Rectangle graphics, 105
- REP, 16
- RESET button, 10
- Resident fonts, 33
- Resident printer fonts, 30
- Resolution enhancement, 16
- Rows, 70
- Run-length encoding, 109

S

- Scalable fonts, 30, 75
- Scale command, 132
- Scaling
 - anisotropic, 132
 - isotropic, 132
 - point factor, 132
- Scaling a GL2 image, 120
- Scaling points, 118
 - inputting, 130, 131
- Screen, 220
- Secondary font, 76
- Self test, 74
- Setting parameters, 12
- Soft fonts, 33
- Source transparency, 101, 103

- Space, 67
- Spacing type, 27
- Special symbols, 35
- Stack operators, 223–224
- Stacks, 204–206
 - dictionary stack, 206
 - execution stack, 206
 - graphics state stack, 206
 - operand stack, 204
- Standard font, 171
- statusdict, 284
- String, 202, 208
- String operators, 241–244
- Stroke weight, 79
- Subpath, 196
- Superset commands, 22
- Symbol set, 29, 77, 81, 82
- Symbol set selection, 81, 82
- Symbol sets, 35
- Symbols
 - special, 35
- System 7, 31, 193
- systemdict, 206

T

- Tagged image file format, 109
- TEST button, 10
- Text area, 44
- TIFF, 222
- Transparency mode, 101, 102
- Transparency mode (GL2), 168
- TrueImage, 2
- TrueImage extensions, 284–289
- TrueImage interpreter, 200
- TrueImage operators, 223
- TrueImage syntax, 207
- TrueType, 2, 31, 32, 193
- TrueType fonts
 - TrueType fonts, 211
- Type and attribute operators, 256–258
- Typeface, 27, 79
 - sans serif, 25
 - serif, 25
- Typefaces, 23

U

Units

current units, 117

plotter units, 117

user units, 117

User default environment, 49

User settings, 11

User space, 198

User units, 117

userdict, 206

V

Vector graphics, 117

Vector group commands, 138–148

Virtual memory, 210

Virtual memory operators, 270

W

Weight, 28

Windows, 193

Windows 3.1, 31

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