Uniplex Device Configuration Guide

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

About this Guide

THIS PAGE INTENTIONALLY LEFT BLANK

About this Guide

This guide is intended for anyone who wishes to carry out advanced configuration in the following areas:

o   Configuring Terminals
o   Configuring Printers
o   Configuring Uniplex Windows

Note for existing Uniplex users:

This information in this guide was previously detailed in two Uniplex Version 7 technical guides which no longer exist:

o   Uniplex Configuration Guide (Volumes 1 and 2)

         The chapters: "Configuring Terminals" and "Configuring Printers" are now part of this guide.

         Note: The greater part of the Configuration Guide can now be found in the Uniplex Technical Guide.

o   Uniplex Windows Configuration Guide

         Relevant information from this guide can now be found in this guide.

The remaining information in the Uniplex technical guides have been reworked and distributed as follows:

o   Uniplex Administration Guide

         This information has been divided between the Uniplex Technical Guide and the Uniplex Form-Building Tools guide.

o   Uniplex Installation Guide and Release Notes (Versions 7.02, 8.00)

         Installation information is now in the retitled Uniplex Installation Guide. Specific information from the Release Notes have been incorporated into the appropriate guide; current Release Notes can now be found on-line.

Assumptions

In order to carry out the configuration tasks described here, it is essential that you have a good general knowledge of computing and that you are an experienced UNIX user. You should also be familiar with:

o   The major Uniplex applications

o   The way that Uniplex is used in your organization

Note: We expect you to use this guide in conjunction with an installed version of Uniplex. This enables you to browse the configuration files to view real examples of the information contained in this guide.

Shell Syntax

While configuring Uniplex, you may need to set environment variables. The way to do this depends on the operating system shell you use.

For the Bourne shell, set variables using the syntax:

VARIABLE=value; **export** *VARIABLE*

For example:

TERM=ansi; export TERM

For the C shell, set variables using the syntax:

setenv *VARIABLE value*

For example:

setenv TERM ansi

Note: Throughout this guide, all examples for setting environment variables show the Bourne shell. If you are using a different shell, substitute the given command for the one appropriate to the shell your system uses.

Chapter 1

Configuring Terminals

THIS PAGE INTENTIONALLY LEFT BLANK

Introduction to Configuring Terminals

You can use Uniplex on almost any character-based terminal. You need a high-resolution graphics terminal to use the Presentation Editor application of the Uniplex Advanced Graphics System module.

This chapter explains how Uniplex provides support for different types of terminals, and how to change or create the configuration for a particular terminal. It contains the following sections:

Determining the Terminal Type

Each Uniplex application determines the user's terminal's capabilities at invocation time by:

o   Reading the terminal *type* from the setting of the TERM environment variable.

        Note: See the chapter The Uniplex Environment in the Uniplex Technical Guide for details of environment variables.

o   Reading the capabilities of the *type* from Uniplex's database of compiled terminal descriptions.

        The Uniplex front-end script checks for the existence of a compiled description before loading the Uniplex main menu program, and if there is none, attempts to compile one. If it cannot find an entry to compile, then it abandons trying to load Uniplex and displays an appropriate message.

When a user invokes either Presentation Graphics or Presentation Editor, then Uniplex determines whether the terminal can support high-resolution graphics by checking the Tcap FILTER setting. If the setting defines a valid graphics filter for this terminal, then Uniplex enables high resolution graphics mode.

If the terminal cannot display high-resolution graphics, the Presentation Graphics module can only display the character-mode representation of graphs. However, you can print the graphs you create using a Uniplex-supported printer.

You cannot use the Presentation Editor on character-based terminals, since it requires high resolution graphics capabilities.

Compiling Terminal Descriptions

Uniplex uses compiled terminal descriptions to improve performance. As described in the previous section, when you invoke Uniplex, it checks for the existence of a compiled terminal description, and if there is none, compiles the appropriate entry automatically.

When you are modifying the terminal configuration files, you must compile the appropriate descriptions to check your entry works. You can compile terminal descriptions using the *syscomp* utility program:

Enter:	To:
syscomp	Compile the description for the current setting for the environment variable $TERM.
syscomp *TERM*	Compile the description for the named *TERM* (where *TERM* is the terminal type).
	See the Note under the first bullet point below.
syscomp ALL	Compile all the types defined in Tcap.

Note: For full details of **syscomp**, see the appendix Program Usage and Invocation in the Uniplex Technical Guide.

When compiling a terminal description, *syscomp* reads:

o   The *termcap* or *terminfo* description (as appropriate to your system) for basic terminal operations. See the later section Configuring Terminfo and Termcap.

   Note: Some operating systems misreport the 'lines' and 'columns' settings from terminfo or termcap when syscomp is used to compile a named terminal. If this occurs, 'disconnect' syscomp from the named terminal using a command sequence similar to the following:

   syscomp TERM < /dev/null > /tmp/syscomp.out 2 >&1
   cat /tmp/syscomp.out
   rm /tmp/syscomp.out

o   The system command file *uniplex.cmd* for all Uniplex command sequences, including mapping the function keys to the softkeys.

o   The Uniplex-specific terminal configuration file *Tcap* for output definitions (for example, how to draw boxes, where to display the softkey line.)

The utility *syscomp* places the compiled description in *UAP/commands* and *UAP/termdef*. It places the description in the local or central UAP, depending on whether the Tcap it finds is local or central.

Developing Terminal Drivers - Conventions

The symbol set used by Uniplex is the X/OPEN symbol set. This provides the basis for transportable documents. Most printers can accept a document containing X/OPEN characters and print an acceptable representation of them. To edit a document requires all terminals to operate in the same manner. For single language organizations this does not pose much of a problem as they probably have the same terminal language across all their terminals. To cope with the ever increasing requirement to support mixed language environments a consistent approach to developing terminals drivers has been adopted.

Tcap

Although the entries in the Tcap file are capable of handling both input and output mapping of characters only the output mapping facility is utilized. Thus any new terminal entry should have all the required maps to output the X/OPEN symbol set.

termset/termreset

Some terminals do not contain the necessary symbols in order to display the X/OPEN symbols. When the terminal has a font download capability then the required characters are downloaded using the files in the *UAP/termset* directory. The original fonts are reinstated by the files in the *UAP/termreset* directory after exiting Uniplex.

Modifying or Creating a Terminal Description

Follow these steps when creating or modifying a terminal description:

1   Set up the environment so that it correctly addresses Uniplex programs and configuration files. You must set up at least the environment variables *PATH*, *TERMINFO* or *TERMCAP* as appropriate and *Uredirect*.

The easiest way to do this is using a terminal that already supports Uniplex. Invoke Uniplex, and press **ESC ESC $** to enter the shell with the correct environment set.

2   Change to the home UAP directory, creating one if required:

cd $HOME/UAP

3   Find out whether your system uses either the *terminfo* system, or the *termcap* system, by running
        uinfo:

        $Uredirect/UAP/install.cmds/uinfo

        Uniplex displays a message indicating which system Uniplex uses.

        If your system uses the *terminfo* system:

        a)   Create an appropriate *terminfo* source file. For example, if you are creating a terminal
                definition for terminal type ansi, create a file called *ansi.src*.

                Consult your operating system manuals for details of creating *terminfo* source files, and the
                terminfo format.

                See the next section Configuring Terminfo and Termcap for details of the keywords that you
                must set for Uniplex to work correctly.

                If required, you can copy a suitable entry from the *UAP/terminfo* file to use as a basis for your
                source file.

        b)   When you have created your source file, compile it using the operating system program *tic*, for
                example:

                tic -v ansi.src

                See your operating system documentation for details of the *tic* utility.

        If you do not want to use the Uniplex *terminfo* compiled database, you must set the TERMINFO
        environment variable to point to where the database to use resides.

        If your system uses the *termcap* system:

        a)   Create or edit the appropriate entry in the *UAP/termcap* source file. For example, the *ansi*
                entry.

        b)   Consult your operating system manuals for details of creating *termcap* source files, and the
                termcap format.

    c)   See the next section Configuring Terminfo and Termcap for details of the keywords that you must set for Uniplex to work correctly.

    If you do not want to use the Uniplex *termcap* source file, you must set th TERMCAP environment variable to point to where the file to use resides.

4   Make a copy of *uniplex.cmd* in your local UAP directory and make any necessary changes to it. See the chapter Configuring Uniplex Commands (uniplex.cmd) in the Uniplex Technical Guide for details.

5   Make a copy of *Tcap* in your local UAP directory and make any necessary changes to it. See the later section Configuring Tcap.

6   If you are configuring a graphics terminal, make a copy of *Gcap* in your local UAP directory and make any necessary changes to it. See the later section Configuring Gcap for details.

7   Run *syscomp* to compile the terminal description. For example:

syscomp ansi

8   Check the terminal description by setting your terminal type to the type you have created or modified (by setting the $TERM variable), and re-executing Uniplex, on the terminal for which you have created or modified the terminal type. See the section checking your Terminal Description at the end of this chapter. Remember to re-syscomp the terminal description if you need to make any changes (and re-tic if you modify *terminfo*).

When you are satisfied with the terminal descriptions, modify the main UAP configuration source files in the same way you modified your local copies. Then re-compile your new terminal type(s) globally. This ensures that the information is duplicated elsewhere in the system.

Configuring Terminfo and Termcap

The Uniplex files *UAP/terminfo* and *UAP/termcap* are normally used to define the general capabilities of the terminals on which you can run Uniplex.

The format and layout of these files is the same as that used by their standard operating system counterparts. Refer to your operating system documentation for information on how to build entries in these files.

Uniplex will use either *termcap* or *terminfo*, so only one set of entries will be required. The appropriate entry is used by the *syscomp* program to produce a Uniplex-compiled terminal description, it is not used directly by the main Uniplex programs.

The following table shows all the entries read by *syscomp* from *termcap* and *terminfo*. For most terminals, you need only a small subset of these. Those marked with an asterisk (*) are the most commonly defined.

Termcap Entry	Terminfo Entry		Description
cm	cup	*	Cursor motion string.
up	cuu1	*	Move cursor up one line in the same column.
ku	kcuu1	*	String sent by up arrow key.
kd	kcud1	*	String sent by down arrow key.
kr	kcuf1	*	String sent by right arrow key.
kl	kcub1	*	String sent by left arrow key.
am	am	*	Terminal automatically wraps when a character is printed in the rightmost column.
co	cols	*	Number of columns across the screen.
li	lines	*	Number of lines on screen.
so	smso	*	Start of standout mode. Used as the default effect for effects not defined in Tcap.

Termcap Entry	Terminfo Entry		Description
se	rmso	*	End of standout mode.
cs	csr	*	Define scrolling region.
sf	ind	*	Scroll forward one line (top line disappears) within a scrolling region.
sr	ri	*	Scroll back one line (bottom line disappears) within a scrolling region.
al	il1	*	Add a blank line on current cursor line. Note that you will, generally, only require either the cs/sf/sr or the al/dl strings defined.
dl	dl1	*	Delete current line.
ws	wsl		Set number of columns in status line.
ms	msgr		Allow movement of the cursor while in highlight mode.
pt	-		The terminal supports tabbing and has the tabs set every 8 characters.
sg#1	xmc#1		The terminal uses a character cell on screen both before highlights and after. Since most of Uniplex will not be able to use this sort of highlighting, the result of including this keyword is to disable much highlighting, and this keyword should not, generally be used.
ug#1	xmc#1		Same as sg#1.
xs	xhp		This keyword is used only for terminals of a certain class. If required, it is documented in your terminal manual.
bc	cub1		Sequence to move cursor left one space. Defaults to backspace character.
bt	cbt		Sequence for backtab.

Termcap Entry	Terminfo Entry	Description
cd	ed	Clear from current cursor position to end of the screen.
ce	el	Clear from current cursor position to end of line.
cl	clear	Clear screen and move cursor to home.
do	cud1	Move cursor down one line in the same column.
ho	home	Move cursor to top left of screen.
le	cub1	Same as bc/cub1 (see above). The logic is: If **le** set, use it, otherwise, if **bc** set, use that, otherwise use ASCII backspace character.
nd	cuf1	Move cursor right one space without erasing text.
ti	smcup	Start of terminal mode (see ke/ks below).
te	rmcup	End of full terminal mode.
us	smul	Start of underscore mode. Only used if so/smso is not set - in which case used as the default standout mode.
ue	rmul	End of underscore mode.
ks	smkx	Start of keypad mode / terminal initialization. Terminal initialization is performed as follows: 1 Send ti/smcup string if set. 2 Send ks/smkx string if set. Terminal de-initialization is performed as follows: 1 Send te/rmcup string if set. 2 Send ke/rmkx string if set.

Termcap Entry	Terminfo Entry	Description
ke	rmkx	End of keypad mode.
km	km	If present, signifies the terminal can input 8 bit characters from the keyboard and not to strip the high order bit on input (assuming, on most systems, that **stty -istrip** is also used).
kh	khome	String sent by home key.
CC	cmdch	Command character(s). This is not normally present and defaults to escape. However it can be used to specify an alternative to the escape key (**&** in *uniplex.cmd*) in all commands, such as the DO key on vt220.

Configuring Tcap

In the Uniplex Tcap file, you define Uniplex-specific terminal capabilities that are not definable in *terminfo* or *termcap*.

File Format and Layout

The Tcap file uses the standard Uniplex file format and layout. The following table shows this layout, together with the names of the sections where each part of the file is explained in detail.

Section of Chapter	Tcap File
Provide Terminal Name	#*term_name*[,*term_name* ...]
Display the Softkey Menu Line	INIT=*escape_sequence* DEINIT=*escape_sequence*
Set the Print Effect Video Attributes	X=*on_sequence, off_sequence* X=*on_sequence, off_sequence* . . X=*on_sequence, off_sequence*
Set the Character Graphics	[=*on_sequence, off_sequence* BOXC=*'string'* BOXE=*'string'* FILLC=*'string'* FILLE=*'string'* PLOTC=*'string'* PLOTE=*'string'* JOINC=*'string'* JOINE=*'string'*
Set the General Video Display Attributes	PAINT=*'string'*
Set the Optional Video Display Attributes	co HIGHBIT nt WIDTH=*nnn* ns

Section of Chapter	Tcap File
Provide Maps for Special Characters	MAP=*value1,value2,value3* MAP=*value1,value2,value3* . . MAP=*value1,value2,value3*
Set High-resolution Graphics Keywords	ss FILTER='*string*'))

To see how these sections relate to a terminal type, see the supplied Tcap file.

Tcap Entry Syntax

Each entry is in standard Uniplex file section format and obeys the following syntax rules:

Entry	Rule
Literals	Enclose in single quotes. For example: '[7m'
ASCII decimals	Do not enclose in quotes. For example: 114
ESC (Escape)	Use the $ (dollar) sign to indicate ESC.
- (dash)	Link values together using dash. For example: I=$-'[7m'-$-'[5m',$-'[m'
, (comma)	Use the comma to separate sequences.

Provide Terminal Name

At the beginning of a terminal section, you must provide the terminal name using the following syntax:

#terminal_name **[,**terminal_name ...**]**

This *terminal_name* must be the same as the corresponding setting for the *TERM* environment variable and the one used in the corresponding entry in *UAP/terminfo* or *UAP/termcap*, and optionally in *uniplex.cmd*. If you want, you can specify more than one name.

When you invoke Uniplex, it finds out which terminal the user is running by reading the *TERM* environment variable.

Display the Softkey Menu Line

By default, Uniplex automatically displays the softkey menu on the bottom line of the screen. If the terminal has a programmable status line (often the 25th line), you can configure the Tcap file to display the softkey menu on this line.

Note: Use the INIT and DEINIT keywords for this purpose only. They are not general terminal initialization keywords.

To configure the softkey menu line:

1   Use the terminal manual to find the sequences to enable and to address the programmable line.

        Note: Often, the terminal manual refers to these as the initialize and de-initialize status line sequences.

2   If possible, set the enable sequence in the terminal initialization string in TERMINFO/TERMCAP (smkx/ks).

3   Enter the sequence to address the line in the Tcap entry for the particular terminal, using the syntax:

        INIT=_sequence1_
        DEINIT=_sequence2_

        Where:

        sequence1 addresses the programmable line and sets any desired highlights.

        sequence2 turns off any highlights.

Make sure *sequence1* performs the following functions on the terminal:

o   Enables the programmable line (if not possible, once-off, in the initialization string).

o   Moves the cursor to the start of the programmable line.

o   Clears the line.

o   Turns on highlighting, if required.

Make sure *sequence2* performs the following functions on the terminal:

o   Turn off highlighting (if it was enabled).

For example:

```
INIT=$-'[>1h'-$-'[25;H'-$-'[K'-$-'[7m'
DEINIT=$-'[m'
```

Set the Print Effect Video Attributes

In Tcap you can set video attributes for Uniplex print effects. In this way, when a user effects text, Uniplex displays the text with a corresponding video attribute. For example, when a user selects the bold effect, Uniplex usually displays the selected text in reverse video.

Uniplex relates the print effect video attribute defined in Tcap for a terminal, and the attribute defined for a printer in Pcap as follows:

1   When the user selects a print effect, Uniplex displays the contents of the #EFFECTS section of uniplex.eff.

2   The user presses a character (for example, A for Bold) to identify the required print effect.

3   Uniplex uses the video attribute definition in Tcap (for example, reverse video) to highlight the effected text on the screen.

4   When the user prints the text, Uniplex uses the equivalent definition in Pcap (for example, the escape sequences to produce emboldened text) to effect the printed copy.

You define video attributes for a terminal using the following syntax:

X=on_sequence, off_sequence

Where:

X   is a single uppercase character in the range A - Z or [.

on sequence Turns on the required attribute.

off sequence Turns off the required attribute.

For example:

A=$-'[7m',$-'[m'   Effect A is defined with the sequence $-'[7m' to turn on inverse video, and the
                           sequence $-'[m' to turn off reverse video.

C=$-'[4m',$-'[m'   Effect C is defined with the sequence $-'[4m' to turn on underscore, and the
                           sequence $-'[m' to turn off underscore.

It is recommended that you define video attributes for the most commonly-used print effects: bold (A),
underline (C) and italic (I). Try to use terminal attributes that identify these uniquely. You can leave other
effects undefined. If they are undefined, Uniplex uses the default standout mode (as defined by *smso* in
terminfo and *so* in *termcap*), which is usually reverse video.

Do not use blinking effects, most terminals have them, but they are irritating to users. Do not use
double-wide or double-high character sets since Uniplex does not support them.

Set the Character Graphics

Many terminals are capable of displaying character graphics. Usually the terminal performs this using an
alternate character set, accessible via an escape sequence. On some terminals, such as PC ansi
monitors, the character graphics are accessed using an 8 bit extension. Uniplex makes use of this
capability, allowing the user to draw boxes and create character graphics. You use definitions in the Tcap
file to define the character graphics.

The Tcap keywords for setting the character graphics are detailed below.

Keyword	Summary
[	Enables/disables character graphics on this terminal, if necessary
BOXC	Defines the boxing characters
BOXE	Defines an effect for each BOXC character
FILLC	Defines up to 10 fill or shade characters
FILLE	Defines an effect for each FILLC character
PLOTC	Defines up to 6 plot characters to use for plotting points
PLOTE	Defines an effect for each PLOTC character
JOINC	Defines up to 6 join characters
JOINE	Defines effects for each JOINC character

The following sections describe how to set each of these keywords. If the terminal does not have a character graphics set, see the later section Terminals without a Line Draw Character Set.

Enable Character Graphics

You define the alternate line draw character set keyword using the syntax:

[=on_sequence, off_sequence

Where:

on_sequence turns on the line draw/alternate character set.

off_sequence turns off the alternate/line draw character set and returns to the default ASCII character set.

For example:

[=$-'(0',$-'(B'

Note: On terminals such as the PC ansi monitor, the line draw characters are part of an 8 bit extension, so you do not need to include this entry, as all graphics characters are accessible in the same character set.

Set the Line Draw Characters

The BOXC keyword defines the characters to send to the terminal to display graphics boxing characters. These characters differ from terminal to terminal. The characters you specify are the character equivalent of the ASCII value to send to the terminal. Specify these characters in the following order:

Relative Character Position	Used to display	Stored Character (See below)
character 1	—	A
character 2	│	B
character 3	┼	C
character 4	┌	D
character 5	┬	E
character 6	┐	F
character 7	┤	G
character 8	┘	H
character 9	┴	I
character 10	└	J
character 11	├	K

You use the BOXE string to define either:

o   The character to invoke the terminal's graphic character set.

       For example: [

o   If no graphic character set is available, specify an effect to use instead. For example, if a terminal does not have a particular graphics character you can specify to effect this particular character with a space, when requested to display it.

In this way, when Uniplex displays or prints the file, it can interpret all graphically effected characters as their graphical equivalent.

For example:

BOXC='D3EZB?4YA@C'
BOXE='[[[[[[[[[[['

If you accidentally type over a graphics character, rather than redraw the complete box or line, you can enter just the single character by entering:

ESC <[and then the appropriate *stored character* from the table above.

Set Fill or Shade Characters

If your terminal can display special shading characters, you can specify these to produce the shaded or filled areas used in bar graphs and in the Word Processor. Choose patterns that allow you to easily distinguish successive adjacent fill characters. For example, do not make character 2 and 3 generate the same pattern. You can define up to 10 fill or shade characters.

The FILLC keyword defines the characters to send to the terminal to display fill characters. These characters differ from terminal to terminal. The characters you specify are the character equivalent of the ASCII value to send to the terminal.

You use the FILLE string to define either:

o   The character to specify you want to use the terminal's graphic character set. For example: [

o   If no graphic character set is available, an effect to use instead. For example, if a terminal does not have a particular fill pattern, you could specify to display this particular pattern as an effect.

For example:

FILLC='[02ˆ_NEX1:'
FILLE='[[[[[[[[[['

Set Join and Plot Characters

You can set the characters used by Uniplex to produce plotted points for line or scatter graphs, and the characters used to join plotted characters. By default these are not included in Tcap entries, as they look best in the default ASCII character set, as described in the later section Terminals without a Character Draw Set.

You can set these characters using the following keywords:

PLOTC=*up to 6 plot characters to use for plotting points*
PLOTE=*effect for each PLOTC character*
JOINC=*up to 6 join characters*
JOINE=*effects for each JOINC character*

For example:

PLOTC='******'
PLOTE='TBKQWY'
JOINC='L'
JOINE='['

Terminals without a Line Draw Character Set

If the terminal does not have a line draw character set, Uniplex accomplishes boxing and shading for graphs using the standard character set. You do not require any entries in Tcap to produce this. Uniplex automatically uses the following characters:

o   Box Characters (for drawing lines and boxes)

plus	+	for all corners and intersections
minus	-	for all horizontal lines
pipe	¦	for all vertical bars

o   Fill Characters (for producing bar graphs/shading)

hash	#	(fill character 1)
slash	/	(fill character 2)
asterisk	*	(fill character 3)
plus	+	(fill character 4)
percent	%	(fill character 5)
equals	=	(fill character 6)
dollar	$	(fill character 7)
colon	:	(fill character 8)
uppercase X	X	(fill character 9)
bracket	)	(fill character 10)

o   Plot Characters (for plotting line and scatter graphs)

lowercase x	x	(plot character 1)
asterisk	*	(plot character 2)
plus	+	(plot character 3)
lowercase o	o	(plot character 4)
minus	-	(plot character 5)
equals	=	(plot character 6)

o   Join Characters (for joining the points on line graphs)

Period   .

Set the General Video Display Attributes

You use the PAINT keyword to set the general display attributes for a terminal. The syntax for the PAINT keyword is:

PAINT='*string***'**

where *string* is up to 17 position dependent characters describing terminal display in the Uniplex environment. The following table shows the purpose of each character, with its default setting. The notes are explained after the table.

PAINT

Character	Description	Default	Note
character 1	Character or effect to use to indicate start of input field	[	*1*
character 2	Character to use to indicate end of input field	]	
character 3	Character or effect to use to indicate start of pick and point field	Z	*1*
character 4	Character to use to indicate end of pick and point field	*SPACE*	*1*
character 5	Effect for menu title line	Z	
character 6	Character to indicate spaces in input field	_ (underscore)	
character 7	Effect for popup	[	*1,2*

PAINT

Character	Description	Default	Note
character 8	Effect to box menu	[	*1,2*
character 9	Effect for ruler line	Z	
character 10	Effect for error messages	Z	
character 11	Effect for softkey line if no INIT string.	Z	
character 12	Effect for cut and paste area marks and effect for displaying dot commands.	Z	
character 13	Reserved (BLINK)	*SPACE*	
character 14	Reserved (BOLD)	A	
character 15	Reserved (ULINE)	C	
character 16	Reserved (REVRS)	Z	
character 17	Default 8-bit display effect	Z	

Notes:

1   If these character positions contain effect letters, the corresponding effect is used, otherwise the character is used.

2   If effect letter [is used in either of these character positions, then Uniplex will use the appropriate line draw character set (defined in BOXC/BOXE) to draw the menu boxes.

In all the above, a SPACE where an effect letter could be used signifies no effect required.

Effect **Z** is used as the default for many PAINT fields, since this is usually an undefined effect, which therefore defaults to the standout rendition defined in terminfo/termcap. This is most commonly set to reverse video for monochrome terminals.

For example:

PAINT='[]A A_[[AAAZ'

Set the Optional Video Display Attributes

The following section describes additional optional keywords you can include in a Tcap entry.

Keyword	Description
co	Use full screen width, (default is width -1). In particular this allows the use of the bottom right hand corner of the screen, which normally scrolls the whole screen.
HIGHBIT	The terminal supports direct 8-bit output. This disables Uniplex's default output processing of characters in the range 160 to 255 (with the high or 8th bit set), and outputs them unchanged so that the terminal can display them as required. (For example, the X/OPEN recommended representation of character 233 is e acute. See the appendices in the Uniplex Technical Guide for ASCII and X/OPEN tables.)
	For example, if HIGHBIT is not defined, and Uniplex needs to display character 233, it will, by default display this in the default display effect. If, however, HIGHBIT is set, the character will be written directly to the screen - on the assumption that the terminal has a character set that will display character 233 as e acute.
	Note that display defaults can always be overridden -regardless of HIGHBIT - using MAP statements to select more accurate representations - where the terminal supports them. See the later section Provide Maps for Specials Characters for details.
nt	A graphics initialization sequence is required before every graphics character.
WIDTH=*nnn*	Maximum width of a line in the word processor. Defaults to its maximum value: WIDTH=254.
ns	Optimize performance by disabling line-by-line scrolling when in the Word Processor. If set, and the user attempts to move the cursor one line down when at the bottom of the screen, the active window on the document is automatically moved down so that the selected line is about one quarter of the way down the screen.

Keyword	Description
UNOOPT	Disables cursor optimization for all products. The product uses the cursor movement command as defined in *termcap* or *terminfo* instead of optimizing.
rw	Reserved.
nu	No longer used.
FILTER=*'string'*	See below.
ss	See below.

Provide Maps for Special Characters

You use the MAP keyword in a Tcap entry to provide support for special characters, such as 8-bit characters and foreign language characters. You use the syntax:

MAP=*value1,value2,value3*

Where:

value1   Is the input from the keyboard you want to map.

   If the input mapping for *value2* is defined in *uniplex.cmd*, you can specify 0 for *value1*.

   In *uniplex.cmd*, specify the values using the syntax **M**value2=value1. See the chapter Configuring Command Keystrokes (uniplex.cmd) in the Uniplex Technical Guide for details.

value2   Is the single character to store in the text file.

value3   Is the string you want displayed on the terminal screen, when *value2* is displayed.

For example, if the input map for a hard space is defined in *uniplex.cmd* as:

m160=&-' '   * Hard space entered as ESC SPACE

then the Tcap mapping for a terminal might be:

MAP=0,160,$-'G4='-$-'G0'

Where:

160 is the decimal value of the character that Uniplex recommends for hard space.

$-'G4='-$-'G0' is the sequence on the terminal to display the equals sign in reverse video.

If the input map for a hard space is not defined in *uniplex.cmd*, then the input mapping can be specified by writing the Tcap statement as:

MAP=$-' ',160,$-'G4='-$-'G0'

Set High-resolution Graphics Keywords

There are two keywords in the Tcap terminal section which specify the terminal's high-resolution capabilities. Include the graphics keywords only if it has these capabilities. You must have a high-resolution graphics terminal to use the Uniplex Advanced Graphics System applications.

Set the following keywords in the Tcap section for graphics terminals:

o   FILTER='*string*'

This keyword specifies the *filter* name for the terminal and any command arguments it requires. To run the high-resolution graphics applications on a terminal, Uniplex requires a filter for the terminal.

For example:

FILTER='gd_tek4010 -p fal5600'

See the appendix Program Usage and Invocation in the Uniplex Technical Guide for details of the command arguments you can use with these filters.

o   ss

This keyword specifies whether or not the terminal has split screen capabilities. That is whether it can display alphanumerics and graphics concurrently. Include this keyword if the terminal has these capabilities, otherwise do not include it.

Configuring termset and termreset

The directories *UAP/termset* and *UAP/termreset* contain shell scripts relevant to a particular terminal type. The script is given an identical name to the terminal type, for example *ansi* or *ansicolor*.

When you invoke Uniplex on a terminal type for which there is a shell script, the script in *termset* is executed before the binary is run.

The shell scripts contained in *termset* should include details for a terminal type that are only necessary when running Uniplex. For example, the setting of function keys or special effects.

Configuring Gcap

You define the capabilities of a high-resolution graphics terminal in the file *UAP/Gcap*.

The Gcap file also defines the high-resolution graphics capabilities of printers and plotters. See the chapter Configuring Printers in this guide for details.

To use a high-resolution graphics terminal with Uniplex, there must be a *filter* program for that type of terminal. The filter programs provide all the low level interaction with the terminal. When a user invokes either Presentation Graphics or Presentation Editor, Uniplex reads the Tcap FILTER keyword to determine which filter program to use. See the previous section Configuring Tcap for details of FILTER.

There are a number of filter programs available. Each filter provides support for a particular class of device. Sometimes you need to create an entry in Gcap for a device, depending on which filter program it uses and whether you are satisfied with the default settings.

There are the following filters:

Filter	Supported Terminals	Gcap Requirement
gd_tekconfig	Tektronix 4207 Tektronix 4208 Tektronix 4207/8 emulations.	You must provide a Gcap entry for each device that uses the gd_tekconfig filter.
gd_tek4010	Tektronix 4010 Tektronix 4014 Tektronix 4010/14 emulations including: Wyse 99GT Visual 550 Visual 603	You need only to provide a Gcap entry for devices that use the gd_tek4010 filter, if you do not want the default settings.

Filter	Supported Terminals	Gcap Requirement
gd_regis	DEC Regis type. For example: vt240 vt300 vt330 vt340	Gcap entries required. Information available from Uniplex if required but not present in this guide.

The following sections provide information on the Gcap entries required by each of these filters. You can also refer to the supplied Gcap file for details.

Gcap Entry Syntax

UAP/Gcap contains sections in the standard Uniplex file format. The following table shows the syntax you can use to make entries.

Possible Entry	Example	Default
escape_sequence	OPEN=$-'Pt'	Null String
boolean	TEK4010=TRUE	FALSE
real	WIDTH=9.5	0.0
integer	XPIXELS=1023	0

Gcap Entry for gd_tekconfig

The following table shows the entries you need to make in a Gcap section for a terminal that uses the *gd_tekconfig* filter.

Keyword	Description
OPEN=*escape_sequence* CLOSE=*escape_sequence*	Enter graphics mode. Return from graphics mode to normal. Most terminals that support high resolution graphics have two separate modes: alphanumeric mode and graphics mode. You specify the escape sequences to open and close graphics mode using the keywords OPEN and CLOSE.

Keyword	Description
XPIXELS=*integer* YPIXELS=*integer*	Maximum number of pixels horizontally. Maximum number of pixels vertically. For Tektronix emulations, set these as follows: XPIXELS=4095 YPIXELS=3132
YREV=*boolean*	Set to TRUE if the Y axis is equal to 0 at top left of screen, otherwise leave as FALSE. Generally, this is not true for any terminal, so in most cases set this keyword to FALSE.
HEIGHT=*real* WIDTH=*real*	Vertical size of screen in inches. Horizontal size of screen in inches. These keywords specify the size of the area addressed by the graphics functionality and are in decimal inches. Few terminal handbooks state these sizes, but you can measure the screen using a standard ruler.
BLX_DEF=*real* BLY_DEF=*real*	Default bottom left x of the screen. Default bottom left y of the screen. Note: These keywords are currently unimplemented. Therefore, you do not need to include them in a Gcap section.
ALIGNX=*integer* ALIGNY=*integer*	For split screen graphics and alpha plane alignment. See the later section Configuring for Split Screens.
BACKGROUND=*integer*	The background color number. Note: This keyword is currently unimplemented. Therefore, you do not need to include it in a Gcap section.
FOREGROUND=*integer*	The foreground color number. Set this to a color that is distinct from the background color. For example 7 (white).

Keyword	Description
NUM_FNTS=*integer*	The number of font faces available. Leave at 1.
DESADJ=*real*	Character descender adjustment factor.
	Later in the Gcap section, you can define character cell sizes. The lower part of the cell height is expressed as carrying the character descenders. Specify this keyword as the character descender, expressed as a decimal proportion. For example 0.2 if the descender occupies 2/10ths of the height of the character cell. This enables the filter to align graphical text around the underline.
CLRO=*escape_sequence*	The escape sequence to draw background color.
RED=*escape_sequence* YELLOW=*escape_sequence* GREEN=*escape_sequence* CYAN=*escape_sequence* BLUE=*escape_sequence* MAGENTA=*escape_sequence* WHITE=*escape_sequence* BLACK=*escape_sequence*	Sequences to set the colors listed.
LINE0=*escape_sequence* LSOLID=*escape_sequence* LDOT=*escape_sequence* LDASH=*escape_sequence* LDASHDOT=*escape_sequence* LDOTDOT=*escape_sequence* LDASHDASH=*escape_sequence* LDOTDASHDASH_*escape_sequence* LDASHDOTDOT=*escape_sequence*	RGIP line style 0 - no effect RGIP line style 1 - solid RGIP line style 2 - dotted RGIP line style 3 - dashed RGIP line style 4 - dash dot RGIP line style 5 - fine dot RGIP line style 6 - dash dash RGIP line style 7 - dot dash dash RGIP line style 8 - dash dot dot
	You can change the order of the line styles as required.

Keyword	Description
HERSHEYTEXT=*boolean*	TRUE if using Hershey fonts for text.
HERSHEYMARK=*boolean*	TRUE if using Hershey fonts for marker drawing.
HERSHEYMENU=*boolean*	TRUE if using Hershey stroke fonts for drawing menu and softkey text in the Presentation Editor; FALSE if using hardware fonts to draw them.
OWN_SEG=*boolean*	TRUE if device draws its own segment.
OWN_ARC=*boolean*	TRUE if device draws its own arc.
OWN_REC=*boolean*	TRUE if device draws its own rectangle.
OWN_CLEAR=*boolean*	TRUE if device has full screen clear.
OWN_LWID=*boolean*	TRUE if device has own line width.
OWN_MCVT=*boolean*	TRUE if device has mouse conversion.
OWN_TFACE=*boolean*	TRUE if using specific character faces.
OWN_VPORT=*boolean*	TRUE if using firmware viewport.
OWN_WIND=*boolean*	TRUE if using firmware window.
OWN_HLT=*boolean*	TRUE if device has highlight mode.
OWN_CLIP=*boolean*	TRUE if device has clipping mechanism.
	Note: The above eleven keywords are currently unimplemented. Therefore, you do not need to include them in a Gcap section.
OWN_TSIZE=*boolean*	TRUE if using specific character sizes.
OWN_FSTY=*boolean*	TRUE if device uses its own fill styles.
OWN_MARK=*boolean*	TRUE if device uses its own markers.
OWN_PFILL=*boolean*	TRUE if device has its own polygon fill styles.
OWN_TSTYLE=*boolean*	TRUE if using specific character styles.
OWN_INIT=*boolean*	TRUE if device has specific initialization.
OWN_MOVE=*boolean*	TRUE if device has its own mouse.
OWN_RBAN=*boolean*	TRUE if device has a rubber band line.
OWN_DOT=*boolean*	TRUE if device has its own dot.
OWN_TEXT=*boolean*	TRUE if device uses inbuilt text.
TEK40TEN=*boolean*	TRUE if using Tektronix 4010 emulation.
TEK40HUN=*boolean*	TRUE if using Tektronix 4100 emulation.
ST_AG=*escape_sequence*	Enter alphagraphics mode.
	Set to $-'LT'
ST_GR=*escape_sequence*	Enter graphics mode.

Keyword	Description
KEYLOAD=*escape_sequence*	Load special function keys.
	Note: This keyword is currently unimplemented. Therefore, you do not need to include it in a Gcap section.
DOTSON=*escape_sequence* DOTSOFF=*escape_sequence*	Makes plotted data visible. Makes plotted data invisible.
	If the terminal is a monochrome device, use these codes to enable draw mode and erase mode respectively. These two codes are sometimes called dark and light vector modes. They are not part of any specific Tektronix code sequence, but are implicit in the correct operation of a graphics raster-type terminal.
	If there is not a DOTSOFF code sequence, then some control string to disable all drawing is required. This is because, when Uniplex draws, for example, a bar, it draws it first in DOTSOFF mode, so if there is not a DOTSOFF code sequence, you require a control string to disable all drawing. This clears the area ready for Uniplex to fill the bar correctly. If the code for DOTSOFF is wrong, Uniplex may draw all bars as solids, regardless of data grouping, with no differentiation in the data plots.
VBS=*escape_sequence*	The backspace. For most terminals, set this to CTRL H. That is, VBS=8.
CELL0=*escape_sequence* CELL1=*escape_sequence* CELL2=*escape_sequence* CELL3=*escape_sequence* CELL4=*escape_sequence*	Text size control string for size 0 Text size control string for size 1 Text size control string for size 2 Text size control string for size 3 Text size control string for size 4
	These keywords specify the text sizes 0, 1, 2, 3, 4, as distinct from pixel sizes described by XSIZE1, YSIZE1 and so on.

Keyword	Description
ALIAS=*boolean*	Set this to TRUE if using a monochrome device.
	The Uniplex graphics applications use color information as the major attribute. If the terminal is monochrome only, then Uniplex generates fill patterns to use in place of colors. If ALIAS is set to TRUE, Uniplex uses color information to set fill patterns and the color output remains the same.
GIN_MODE=*boolean*	Set to TRUE if the device supports Tektronix 4100 advanced input handling.
	The Presentation Editor requires advanced Tektronix input capabilities. These capabilities are available on terminals such as the 4207 and 4208 devices. Terminals such as the 4105 offer only 4010 capabilities, which generate cursor position information, whenever a key is pressed. Uniplex requires that graphics input information is only sent when a button is pressed. This allows its KEY_IN mechanism to work.
	The advanced input handling Tektronix commands that the terminal needs to support are:
	SET REPORT SIG CHARS ENABLE GIN DISABLE GIN
	The command:
	ENABLE 4010 GIN
	will not operate correctly.

Keyword	Description
CAN_90=*boolean* CAN_180=*boolean* CAN_270=*boolean*	TRUE if device can print text at 90 degrees TRUE if device can print text at 180 degrees TRUE if device can print text at 270 degrees
	These keywords specify a device's ability to perform graphics text rotation. If any of these is set to FALSE, the filter generates a nearest possible synthesis. For example, for 180 degrees requirement, the filter right-justifies text.
INIT_90=*escape_sequence* INIT_180=*escape_sequence* INIT_270=*escape_sequence*	String to start 90 degree printing. String to start 180 degree printing. String to start 270 degree printing.
	These keywords specify the strings to invoke text rotation.
STOP_ROT=*escape_sequence*	String to stop rotation. This string is not required by terminals that are unable to rotate text. It sets 0 degrees of rotation.
MAXFONTS=*escape_sequence*	The maximum number of text fonts.
TEK_MODE=*boolean*	TRUE if Tektronix emulation.
LINESTYLE=*boolean*	TRUE if terminal supports linestyles.
TEK_MAX=*boolean*	Set to FALSE if the terminal sends both sets of coordinates to draw lines.
	Note: This keyword is currently unimplemented. Therefore, you do not need to include it in a Gcap section.

Keyword	Description
TEK_SLEEP=*integer*	Causes the filter to pause every *integer* line.
	Sometimes, the handshaking on the communication line does not operate correctly. If this is the case, Uniplex may not be able to draw the required graphic correctly. You can make the filter pause for one second every *integer* line. This allows the terminal to catch up on its backlog in the event of handshake failure. Try a starting point of 50 for this keyword.
	Note: At least 9 bytes are sent to describe each line.
TEK_LN=*boolean*	TRUE if device supports Tektronix line vectors.
MAX_SIZ=*integer*	Max. number of font styles. Minimum is 1.
MAXLNSTY=*integer* MAXSTY=*integer* MAXCLR=*integer*	Max. number of line styles. Minimum is 1. Max. number of fill styles. Minimum is 1. Max. number of colors. Minimum is 1.
	Set these to 8.
XSIZE0=*integer* YSIZE0=*integer* XSIZE1=*integer* YSIZE1=*integer* XSIZE2=*integer* YSIZE2=*integer* XSIZE3=*integer* YSIZE3=*integer*	Character cell x and y sizes. They are used for 4010 mode only. These keywords specify the character cell sizes in pixels, for the four sizes of graphic text that the applications can call. Size 0 is the smallest. The filter uses these sizes to calculate the positions for text justification, relative to the reference point supplied by the application.
	4100 series terminals have stroke defined text which is defined by the keywords:
	OWN_TSIZE=TRUE OWN_TEXT=TRUE OWN_TSTYLE=TRUE

Keyword	Description
	The filter changes the size of stroke-defined text, so if the above keywords are present then omit the XSIZE and YSIZE keywords.
M_XLOC=*integer* M_YLOC=*integer* M_BUTT=*integer*	Mouse x location. Mouse y location. Mouse button status.
	Note: These keywords are currently unimplemented. Therefore, you do not need to include them in a Gcap section.
M_CAP=*integer*	Mouse capability. This keyword defines whether the terminal can accept graphics input data from the keyboard.
	If the terminal has no graphics input mechanism via the mouse, set to 0. If the graphics input mechanism is initiated by a start character indicating the number of bytes in a message string, set to 1. If the graphics input mechanism is initiated by a start and stop character, set to 2.
M_START=*integer*	If you set M_CAP to 1 or 2, with this keyword, specify the character code to send to the terminal to start transmission of position and button information for the mouse (bit-pad or other input device).
	With this keyword, specify the ASCII decimal equivalent for the byte to start all graphics information with. Until now 29 (the GS character) has been used.
M_BCOUNT=*integer*	If you set M_CAP to 1, with this keyword, specify the number of bytes in the message string. If you set M_CAP to 2, specify the termination character (ADE).
	This keyword specifies the number of bytes that follow the start character of the graphic input string. This will always be 6 when M_CAP is set to 1.

Keyword	Description
MOUSE_DEV=*escape_sequence*	String for Tektronix/keyboard/mouse.
	Conventionally, this is set as follows:
	If it is a mouse: MOUSE_DEV='D0' If is a bit-map: MOUSE_DEV='8'
	This keyword specifies an internal address for the Tektronix terminal to connect in to its graphics cursor control mechanism. If MOUSE_DEV is set to 'D0', then Uniplex uses the mouse plugged into the back of the terminal. The code for the bit pad assumes it is plugged into port 0. Use the device handbook to use other configurations.
GIN_WIN=*esc_seqence* M_WIN=*esc_sequence*	String to set the gin window. String to set the overall window.
	These two strings (as supplied by the tek4207 entry in Gcap) allow re-mapping of the input devices. Those supplied should suffice in most cases, but refer to your terminal manual for details on re-mapping the graphics input co-ordinate range (GIN_WIN), and the overall screen co-ordinate (M_WIN).

Configuring for Split Screens

Many terminals support what Uniplex calls *split-screen mode*. This means the terminal can display graphical images at the same time as alphanumeric characters. For example, the terminal can display both a menu and a high-resolution graph on the screen at the same time. The terminal achieves this by displaying graphs on the non-scrolling background of the screen, known as the *graphic area* and displaying alphanumeric characters (such as menus) on the scrolling foreground of the screen, known as the *character area*. Usually, these two areas (graphic and character) are slightly different shapes and sizes:

For example:

```
                                    ┌──────────────────────────────────┐ <-----graphic area
                              ┌─────┼──────────────────────────────┐   │
                              │     │                              │   │
                              │     │                              │   │
                              │     │                              │   │ <---character area
                              │     │                              │   │
                              │     │                              │   │
                              │     └──────────────────────────────┼───┘
                              └────────────────────────────────────┘
```
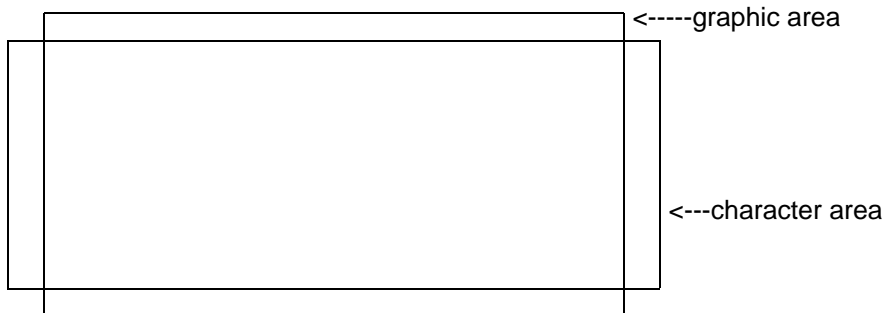
This section describes how to optimize the alignment between the graphic and character area.

Firstly, check whether you need to configure these two areas by checking the alignment as follows:

1   Include the split-screen keyword (ss) in the Tcap section for the terminal.

2   Set the ALIGNX and ALIGNY keywords in the Gcap section for this terminal to 0.

3   Invoke Presentation Graphics. Create a graph. When specifying the graph text, include a main title, a footnote, a value axis label and a group heading. See the Uniplex Advanced Graphics System User Guide for details.

4   When you have created a graph, select the Display Format option.

If the terminal has split screen capabilities, Uniplex displays the menu on the lefthand side of the screen, and the graph on the righthand side of the screen.

If any of the graph text is obscured by the menu, you need to configure the plane alignment as described below.

To align the graphics area and the character area, you re-align the graphics addressing. This ensures that Uniplex only draws graphical images within the graphics area that are coincident with the character area. In the previous example, the horizontal graphical (x) axis does not extend to the full width of the character area, so you cannot improve the graphical alignment. However, you can modify the vertical (y) axis to specify an offset (the ALIGNY keyword in Gcap). Uniplex adds this offset to all incoming y coordinates in the filter, preventing drawing in the non-coincident graphics area. Additionally, you need to modify the XPIXELS value in the Gcap entry for the terminal, to reduce it to the overlapped height of the two areas.

To evaluate the height of the two areas (graphics and character), you need to generate a full screen graphics rectangle and a full screen alphanumeric rectangle, and superimpose them to assess the overlap.

Write a shell script to generate the alphanumeric rectangle. The rectangle should be 24 lines of 80 uppercase X characters.

To generate the graphics rectangle:

1   Create a file with the following single text line in it:

     0 0 10000 10000 1.0 1 7 1 7 BOX

2   Redirect this file to the filter using the command:

     gd_tekconfig -p *Gcap_section* **<** *file*

     Where *file* is the file containing the above line, and *Gcap_section* is the section in Gcap for your terminal.

Use the rectangles you generate to assess the overlap, proportionally.

For example, if the overlap in the Y axis starts an eighth of the way from the graphics area, set ALIGNY to one eighth of the possible range. Reiterate this process to provide alignment of the bottom left corners of the graphics and alphanumeric areas. Lastly, specify a revised number of vertical pixels for YPIXELS, since fewer than the full height will be required. If an eighth of the graphics area extends beyond the overlap, then two eighths of the Y axis will not be used and you need to set YPIXELS accordingly.

Adopt the same method for width alignment if required.

Gcap Entry for gd_tek4010

The following sections describe the entries you can make in a Gcap section for a terminal that uses gd_tek4010 filter.

You do not have to create an entry for a terminal that uses the gd_tek4010 filter. Only create one if you want to change any of the default entries.

The Gcap keywords for an entry for the gd_tek4010 filter are described in the following sections:

Section of Chapter	Description
Initialize the Device	Describes the keywords that switch the device between graphics mode and normal mode.
Configure the Device Parameters	Describes the keywords that specify the physical characteristics of the device.
Configure the Color	Describes the keywords to render color.
Configure the Line Styles	Describes the keywords to select the line styles.
Configure the Fill Styles	Describes the keywords to select the fill styles.
Configure the Text and Markers	Describes the keywords to select the text and markers.
Configure the Miscellaneous Keywords	Describes the miscellaneous keywords.

Initialize the Device

Keyword	Definition
INIT=$key\_sequence$ DEINIT=$key\_sequence$ OSRESET=$key\_sequence$ URESET=$key\_sequence$	These four key sequences bound graphical output from the filter. Their purpose is to set the device to some known state and restore it to some reset state. For example, to set the terminal to Tektronix emulation mode and restore normal mode when output ends.

Two *default* states have been identified. These are the states the operating system environment and Uniplex assumes. These are the states that key sequences *OSRESET* and *URESET* attempt to set. If the filter is invoked *standalone* ("-m" keyword then the *URESET* sequence is sent before any filter output, and the *OSRESET* sequence at the end. If the filter is not invoked standalone, then these sequences are not output.

The *INIT* key sequence performs the action of setting the device in the required graphic mode. It is sent when the RGIP *Open* command is executed. The *DEINIT* key sequence takes the device out of graphic mode and may in addition undo other effects the filter may have set. It is sent when an RGIP *Close* command is executed.

Configure the Device Parameters

The following Gcap keyword entries are used to specify physical characteristics of the device:

Keyword	Default	Description
XPIXELS=*integer*	800	Size of the graphic display in pixels.
YPIXELS=*integer*	312	
XTEKPOINTS=*integer*	1024	The number of Tektronix tek_points that map to the
YTEKPOINTS=*integer*	780	above given number of pixels.
HEIGHT=*real*	9.0	The size of the display area in inches.

The default values are arbitrary.

The standard Tektronix escape sequences for setting drawing modes are built into the filter. However, as a precaution, you can override them using the following Gcap entries. This is only necessary if the emulation is not very close.

Keyword	Default	Description
TEK_GMODE=*escape_sequence*	ˆ] or GS	Set Vector Plot mode
TEK_PPMODE=*escape_sequence*	\ or FS	Set Point Plot mode
TEK_EXITMODE=*escape_sequence*	ˆ_ or US	Return to Alphagraphic mode points
TEK_CLRSCREEN=*escape_sequence*	$-12	Erase Screen and home alphagraphic cursor
FIVE_BYTES=*boolean*	FALSE	See below

The filter uses device pixel coordinates to draw and fill areas. Using the given number of screen pixels and screen tek_points, pixel coordinates are transformed to tek_point coordinates which are sent to the device. The standard format for encoding 4010/14 coordinates involves a 4 byte sequence. However, there is a 5 byte format that will be used if FIVE_BYTES=TRUE.

Configure the Color

Since the Tektronix 4010/14 is a monochrome device, no color mapping entries are applicable. All RGIP colors are rendered in White except Black which is rendered as black. Note, however, that not all emulations can draw black lines. When this is the case, Black graphics are simply not drawn.

The filter is informed of the devices rendering capabilities through the following optional entries in the specified Gcap section:

Keyword	Default
TEK_DOESBLACK=*boolean*	FALSE
TEK_DOESINVERT=*boolean*	FALSE
TEK_SETWHITE=*escape_sequence*	<null>
TEK_SETBLACK*escape_sequence*	<null>
TEK_SETINVERT*escape_sequence*	<null>

TEK_DOESBLACK and TEK_DOESINVERT take TRUE or FALSE values to indicate to the filter if the device can render black lines and perform inverted line drawing respectively. If either or both are true, then the remaining keywords should be set to the escape sequences that place the terminal into the require drawing mode.

Configure the Line Styles

Eight line styles are supported. The five standard Tektronix line styles are employed plus three of the defocused line styles. The escape sequences to select line styles are built in to the filter, but can be overridden by optional Gcap section entries.

Mapping the RGIP line style index to the appropriate Tektronix line style index is a two level process.

The following table lists the RGIP line style Gcap entries with their default values. RGIP line styles can be swapped simply by assigning different numbers to these keywords.

Keyword	Default Value	RGIP Name
LSOLID=*integer*	1	1 - Solid
LDOT=*integer*	2	2 - Dot
LDASH=*integer*	3	3 - Dash
LDASHDOT=*integer*	4	4 - Dash Dot
LDOTDOT=*integer*	5	5 - Dot Dot
LDASHDASH=*integer*	6	6 - Dash Dash
LDOTDASHDASH=*integer*	7	7 - Dot Dash Dash
LDASHDOTDOT=*integer*	8	8 - Dash Dot Dot

The following table gives the Tektronix line style escape sequences. The default values shown map the RGIP index to the closest Tektronix line style. You can change the assignment of the values of the LSTYLE_<n> keywords:

Keyword	Tek. Seq.	CGI Name
LSTYLE_1=*escape_sequence*	$-'‘	Solid
LSTYLE_2=*escape_sequence*	$-'a'	Dotted
LSTYLE_3=*escape_sequence*	$-'c'	Short Dash
LSTYLE_4=*escape_sequence*	$-'b'	Dot Dash
LSTYLE_5=*escape_sequence*	$-'i'	Dotted Defocused
LSTYLE_6=*escape_sequence*	$-'k'	Short Dash Defocused
LSTYLE_7=*escape_sequence*	$-'j'	Dot Dash Defocused
LSTYLE_8=*escape_sequence*	$-'d'	Long Dash

Configure the Fill Styles

The Tektronix 4010/14 does not support area filling, so it is performed wholly by the gd_tek4010 filter.

Seventeen fill styles are supported. The mapping of RGIP fill style index to the fill pattern actually invoked is controlled by overriding the default values of the following optional Gcap entry keywords:

Keyword	Default Value	RGIP Name
FSOLID=*integer*	1	Solid
FHORIZ=*integer*	2	Horz. Lines
F45DEG=*integer*	3	45deg. Lines
FVERT=*integer*	4	Vert. Lines
F135DEG=*integer*	5	135deg. Lines
FOLATTICE=*integer*	6	Horz. Lattice
FDLATTICE=*integer*	7	45deg. Lattice
FHOLLOW=*integer*	8	Hollow
FBRICKS=*integer*	9	not defined
FROOFTILES=*integer*	10	not defined
FBASKET=*integer*	11	not defined
FCHECKER=*integer*	12	not defined
FDIAMOND=*integer*	13	not defined
FSFRIEZE=*integer*	14	not defined
FVBRICKS=*integer*	15	not defined
FSCALES=*integer*	16	not defined
FVSFRIEZE=*integer*	17	not defined

So, for example, if you want to render a brick patten fill in place of the vertical line fill, add the following line to the Gcap section:

FVERT=9

Configure the Text and Markers

The Tektronix 4010/14 only has limited text output capability. The 4010 only has one size and the 4014 has four sizes. They can only output text along a horizontal baseline.

Because of these limitations the gd_tek4010 filter employs the Hershey text output mechanism as used by gd_matrix. However, by specifying a text face of '5' you can get access to the Tektronix native text output method.

Markers are not supported by the 4010/14, so these are rendered using the Hershey mechanism.

The table below gives the Gcap keywords using the text and marker attributes.

Keyword	Default	Description
MPOINT=*integer*	1	Marker - point
MPLUS=*integer*	2	Marker - plus sign
MSTAR=*integer*	3	Marker - asterisk
MSQUARE=*integer*	4	Marker - small square
MCROSS=*integer*	5	Marker - multiply sign
MDIAMOND=*integer*	6	Marker - diamond shape
MHBOW=*integer*	7	Marker - bow tie
MVBOW=*integer*	8	Marker - bow tie rotated 900
TFACE1=*integer*	1	Text Face - simple Helvetica like
TFACE2=*integer*	2	Text Face - complex Roman like
TFACE3=*integer*	3	Text Face - script like
TNORMAL=*integer*	1	Text Type - standard appearance
TBOLD=*integer*	2	Text Type - bold
TITALIC=*integer*	3	Text Type - italic
TBOLDITALIC=*integer*	4	Text Type - bold and italic

If you decided that Hershey text output is too slow, you can switch to the Tektronix text by making the following entries in your Gcap section:

TFACE1=5
TFACE2=5
TFACE3=5

By default this will give access to the standard 4010 size text. If the device emulates a 4014, or has extra text sizes available, these can be accessed using the following configuration keywords. Enough have been provided to allow configuration of up to 8 text sizes.

Keyword	Description
TEK_ALPHA_CODE_1=*escape_sequence* through TEK_ALPHA_CODE_8=*escape_sequence*	Escape sequences to select available text sizes.
TEK_ALPHA_WIDTH_1=*escape_sequence* through TEK_ALPHA_WIDTH_8=*escape_sequence*	The respective pixel widths of the text characters.
TEK_ALPHA_HEIGHT_1=*escape_sequence* through TEK_ALPHA_HEIGHT_8=*escape_sequence*	The respective pixel heights of the text characters.

Keyword	Description
TEK_ALPHA_DECEND_1=*escape_sequence* through TEK_ALPHA_DECEND_8=*escape_sequence*	The respective pixel descender extent of text characters

These entries are only required if you want to get at more than the 1 standard Tektronix text size.

Configure the Miscellaneous Keywords

There are a number of Gcap keywords that do not fall into any of the above categories. These are defined here.

Keyword	Default	Description
PATTERNMAP=*boolean*	TRUE	If set TRUE then line color and fill color are used to select line style and fill style respectively. This is the default since 4010/14's are monochrome devices.
MENU_FACE=*integer* MENU_TYPE=*integer* MENU_SIZE=*integer*	5 1 1	These entries control the text information returned by the RGIP command GetSurface. The default instructs ped to use face 5 for its menu text. Note: Ped cannot be used with this filter.
TEK_OPAQUEFILL=*boolean*	TRUE	For a non-solid fill style the area under a box, for example, may either be shown through the fill pattern or be set to the background color. The choice is made by setting this entry FALSE or TRUE respectively. The default TRUE was chosen because this is what uchart prefers.

Keyword	Default	Description
FILL_WINDING=*boolean*	FALSE	When a polygon intersects with itself there is a choice. Is the self overlapping area filled or not? The default is to use the odd-even fill rule that will fill such an area. By setting this entry TRUE then the winding fill rule is used; this will not fill such an area.
FNONE=*integer*	1	This is a special hook that allows area erasure to background color to be disabled by FNONE=0!. It is recommended that you do this, since clearing any substantial area of the screen is slow. Erasure is only required if Uchart is operating in *split-screen* mode.

Checking your Terminal Description

When you have completed a terminal description, you can check it by completing this set of tests:

Terminfo/Termcap Checking

o   **Basic Operation**

Invoke the Word Processor. Enter some text. Move the cursor using all four arrow keys.

o   **Scrolling**

Invoke the Word Processor. Enter a few screenfuls of text. Position the cursor on the first and line last of the screen, and use the up and down arrow keys. Check the text scrolls correctly. Move to the middle of the screen. Insert a line, and then delete it.

o   **Backspace and Left Arrow Difference**

Invoke the Spreadsheet in ucalc mode. Check that using the BACKSPACE key on the command line, backspaces or erases, the same as the DELETE key. Check that LEFT ARROW moves the cursor.

Tcap

o   **Effects**

Use the Word Processor to edit the file:

UAP/demo/SAMPLES/EFFECTS

Check that each effect is displayed correctly.

o   **X/OPEN mapping**

Use the Word Processor to edit the file:

UAP/demo/SAMPLES/x.open.table

Check that each character is mapped correctly.

APP Terminal Driver Conventions and Documentation

This section describes the conventions used in developing terminal drivers for the APP and all subsequent driver configuration produced by Uniplex.

The symbol set adopted by Uniplex is the X/OPEN symbol set. This provides the basis for transportable documents. Most printers can accept a document containing X/OPEN characters and print an acceptable representation of them. To edit a document requires all terminals to operate in the same manner. For single language organizations this does not pose much of a problem as they probably have the same terminal language across all their terminals. To cope with the ever increasing requirement to support mixed language environments a consistent approach to developing terminals drivers needs to be adopted.

The following sections detail the approach Uniplex has adopted in consolidating terminal support so that it is consistent across Uniplex translations, terminal types, and terminal languages.

Supersections

Following on from the supersections introduced for printer configuration, the terminal entries are collected together within supersections. This allows all the relevant details for a terminal to be kept together and treated as one entity within each of the configuration files.

A terminal supersection is started by a *{comment}*TERMNAME line and terminated by a *{comment}***))**TERMNAME line.

Where:

{comment} is the relevant comment character for the file in question. All Uniplex configuration files use the asterisk (*) character as a comment character. The UNIX files *terminfo* and *termcap* use the hash (#) character as a comment introducer.

A Uniplex file supersection for the Wyse 120 terminal would therefore appear as:

*WYSE120
...
*))WYSE120

A UNIX file supersection for the Wyse 120 terminal would appear as:

#WYSE120
...
#))WYSE120

The supersection name is always an uppercase derivative of one of the terminal's names that can be used for setting the UNIX environment variable TERM.

Tcap File

Although the entries in the Tcap file are capable of handling both input and output mapping of characters only the output mapping facility is utilized. Thus any new terminal entry should have all the required maps to output the X/OPEN symbol set.

termset/termreset Directories

Some terminals do not contain the necessary symbols in order to display the X/OPEN symbols. When the terminal has a font download capability then the required characters are downloaded using the files in the *UAP/termset* directory. The original fonts are reinstated by the files in the *UAP/termreset* directory after exiting Uniplex.

uniplex.cmd File

The APP appends the section *#COMMANDS2-$TERM* to the uniplex.cmd file for each of the newly installed terminals. These sections consist of include statements referring to other sections which are added as part of the installation procedure. The *#COMMANDS2-$TERM* section defines the character sequences sent by the keyboard. The Uniplex functions they invoke are basically independent from the product translation but dependent upon the terminal language and the type of keyboard used with the terminal.

Generally, the *#COMMANDS-$TERM* sections are structured as follows:

```
#COMMANDS2-TERM
include=#TERM-COMMANDS-LANGUAGE
include=#SOFTKEYS-CONFIGURATION
include=#TERM-FKEYS
include=#UNIPLEX-CONFIGURATION
include=#TERM-KEYBOARDTYPE
include=#TERM-KEYBOARD-MAPS
include=#OTHER-MAPS
include=#XOPEN-MAPS
))
```

Where:

#TERM-COMMANDS-LANGUAGE is optional. It would be included where the terminal's control sequences clash with Uniplex sequences. For example, **ESC Q** being a cursor key. Being defined after the COMMANDS sections causes these sequences to take precedence. There would be a section for each language supported by the driver.

#SOFTKEYS-CONFIGURATION defines the fallback softkeys, for example, **ESC 1** for **F1**. This can be omitted should any of the terminal's keys send out conflicting sequences.

#TERM-FKEYS defines the sequences sent by the terminal for each of its function keys. Included here is the definition for K020 which is used for the X/OPEN lead-in sequence. If no specific key is assigned K020, is normally assigned to be **ESC ESC 0** (Escape, Escape, zero).

#UNIPLEX-CONFIGURATION defines the mapping between the terminal's function keys and Uniplex's softkeys. For example, **F1** = **S1**.

#TERM-KEYBOARDTYPE contains definitions for keys specific to the keyboard type. Predominantly, these are the Edit Keypad, containing keys labelled **Insert**, **Home**, **End**. These are assigned the closest Uniplex function matching the key legend.

#TERM-KEYBOARD-MAPS contains the maps required to convert the terminal's character code to the correct X/OPEN code. For example, a French keyboard may send the code 123 for the key labelled **é**, this would be output as a **{** as this is the ASCII representation for this code. The terminal's **é** code needs to be mapped to the correct X/OPEN value (233) so that it is output correctly.

#OTHER-MAPS contains a set of maps to convert from a specific symbol set to their X/OPEN equivalents. Currently, Data General terminals output character codes according to the International symbol set definition. For example, a French DG terminal sends the code 232 for an **é**, as above, the X/OPEN value should be 233, so it is mapped on input to its correct value.

#XOPEN-MAPS contains the default sequences to enter after the X/OPEN lead-in sequence to generate X/OPEN characters that do not appear on the terminal being used.

The order in which the include sections are included is significant. This new mode of operation is dependent upon the order of definition of the Uniplex functions. Uniplex commands are prioritized in reverse order of definition.

For example, the *#COMMANDS-ENGLISH* section normally defines the Quit function (F062) as:

F062=&-'Q'

If a terminal sends **ESC Q** in response to depressing the Cursor Left key Uniplex would attempt **quit** whenever the left cursor key was depressed as the Quit command is defined as case insensitive and occurs after the **F012=L** definition of the cursor left function.

To resolve this conflict the *#TERM-COMMANDS-ENGLISH* section for the terminal in question would define the Quit function as **f062=&-'q'**, making the function case sensitive. Being defined last means that the definition takes precedence over any previously defined sequences. The result of redefining the function is that Uniplex attempts to quit upon receipt of the sequence **ESC q** but attempts a cursor left upon receipt of **ESC Q**, even if the user explicitly enters this sequence.

Character mapping definitions work the opposite way to command definitions. The first map sequence defined takes precedence. In the *#STD-MAPS* section the sequence to produce a sterling (**£**) symbol is defined as **M163=&-'%#'**.

A UK type keyboard has the sterling symbol where the hash symbol should be. Just depressing the sterling symbol key will normally send the code 35 which is treated as a hash symbol. In order to store the correct code the *#TERM-UKENGLISH* section contains the definitions **m35=&-'%'-35 * Hash Symbol**, and **m163=35 * Sterling symbol**.

This causes the sterling code (163) to be stored when **Shift 3** is depressed and the hash code (35) to be stored when **ESC %£** is entered. One side effect of this set of mappings is that in order to invoke the Folios system a user has to enter the sequence for generating a hash symbol (**ESC %£**).

Supported Terminal Specifications (STS)

For each terminal supported in the APP a **S**upported **T**erminal **S**pecification (**STS**) is included in the on-line documentation section of the APP. The STS is comprised of seven sections which detail the terminal facilities supported by Uniplex and any non-standard ways of using the printer within the Uniplex applications. Below is a description of each section within an STS.

1   Terminal Details

This table lists the terminal manufacturer, terminal model, installed options required, keyboard types and languages supported, and the Uniplex name(s) of the driver.

2   Capability Summary

This table lists the general capabilities of the terminal. It details whether the terminal is monochrome or color, if color how many colors are supported, whether mouse input is supported, what key is used as the escape key, and the number of columns supported by the driver.

3   Uniplex Print Time Effects

This table lists the Uniplex effects and the terminal screen effect used to represent it on the terminal's screen.

4   Uniplex Extended Character Set

This section defines whether the full Uniplex character set can be displayed, and if not which characters are unavailable.

5   termset/termreset

The driver can make use of shell scripts in the *UAP/termset* and *UAP/termreset* directories to set and reset the terminal before and after running Uniplex. If this is the case, this section explains what the scripts do.

6   Terminal Set-up Details

This section details how the terminal used for developing and testing the terminal driver is set up. Where the terminal has a specific keyboard, the set-up option is covered separately in the keyboard description.

7   Keyboard Details

For each keyboard type and language combination supported, there is a further set of tables detailing the driver's implementation.

a)   Terminal Set-up Details

This details the set-up of the keyboard specific settings.

b)   Additional Function Keys

This section lists any keyboard keys that are assigned specific Uniplex functions. Where the key label is a symbol a definition for the key label is given.

c)   Input Maps/Functions

This table details what key sequences are required to enter the full Uniplex character set where these sequences differ from the standard sequences. As a minimum this table details the X/OPEN lead-in sequence to use for generating characters not on the keyboard.

All the STS files can be found in the *UAP/documents/APP/sts* directory, if the Install Documentation option is selected when installing the APP. The file *INDEX* in the STS directory lists the terminal driver name against the filename used to store the driver information.

Chapter 2

Configuring Printers

THIS PAGE INTENTIONALLY LEFT BLANK

Overview

Proportionally spaced printing is a method of printing where characters take up differing amounts of space. For example, a lower case i takes up less space than an upper case W. Documents printed in this way are easier to read and more professional than documents printed in fixed pitch, where each character takes up the same amount of space.

You print Uniplex files proportionally spaced or in fixed pitch, using the *uprop* program. This chapter describes how to configure your printer to take full advantage of uprop's capabilities.

Uprop achieves proportional spacing by selecting a proportional font, as defined in the relevant configuration files, and outputting the characters of each word. The physical size of each character is determined by the printer and the printer cartridge you are using. Where necessary, uprop adjusts inter-word white space to achieve the desired justification of text.

Uprop also allows the expansion and shrinking of graphic images when embedded within text, so that graphics fit exactly within the area left after text has been reformatted during printing.

This chapter describes how to configure Pcap, Fcap and Gcap as follows:

o   Pcap: Describes how to define the attributes of your printer. For example, you define the way print effects appear at print-time, the amount of horizontal and vertical head movement, and the entries required for printing character graphics and ruled graphics.

o   Fcap: Describes how to configure Fcap to define the paper sizes you use, and define the sizes of characters for each font (typeface/effect combination) supported by your printer. These sizes are used by uprop to scale the character widths depending on the point size requested.

o   Gcap: Describes how to configure Gcap for printing high-resolution graphics. Parameters within this file consist of sequences to switch the printer in and out of high-resolution mode, values for the size of paper, maximum addressable pixels, and definitions of available colors, line types and fill patterns.

Configuring Pcap

You define the attributes of your printer in the Pcap file. For example, you define the way effects will print, the amount of horizontal and vertical print head movement, and the parameters required for printing character graphics and ruled graphics.

In addition to this chapter, refer to the programmer's documentation for the particular printer you are working on. (Note that this documentation is not normally supplied with the printer). Because the configuration of the Hewlett Packard (HP) Laserjet^{TM} was used as a model for uprop configuration, you may also find it useful to refer to the Laserjet configuration and reference manuals.

The Pcap file contains an entry for each printer and cartridge combination supported by Uniplex. You can modify these entries or add new entries. The complete set of definitions for each printer is known as a *printer section*.

Each entry is indexed at the top of the Pcap file showing the initial line number of each printer section. If you make changes to the file, you should re-index the file to speed up access. For example:

uindex Pcap

Uniplex uses the *Horizontal Motion Index* (HMI) to achieve correct spacing between words. The horizontal motion index specifies the distance the print head moves horizontally. HMI units are usually measured in 1/120 inches or 1/300 inches. If your printer is not capable of this type of movement, then true proportionally spaced printing is not possible.

Uprop uses the *Vertical Motion Index* (VMI) to achieve correct spacing between lines. The vertical motion index specifies the distance the print head moves vertically. VMI units are usually measured in 1/48 inches. If your printer does not support VMI, uprop may fail to support the double underline effects, and vary the line spacing.

Character graphics are formed from individual characters, unlike high resolution graphics which are formed from a series of dots. Uprop makes use of the ability of some laser printers to draw lines, boxes and fills. Each graphics character is scaled to fit within an allocated area to produce accurate graphic images. In Pcap you define which characters uprop uses to draw boxes, and fill areas.

In addition to each individual printer section, you also specify a set of generic keywords to define indexes, contents and section numbering.

File Format and Layout

Function	Pcap File Entry
Define a Printer	#*printer_name*[**/***cartridge***/**] or #[***L/***] *modelsymset*[**/***card***/**]
Include	INCLUDE='*section_name*'
Initialize/De-initialize Printer	INIT=*escape_sequence* DEINIT=*escape_sequence*
Define Printer Effects	*Effect_letter=turn_on_sequence*, *turn_off_sequence*,['FONT:*font_name*'] or ['FONT=*font_name*']
	FONTCAPS=*'typeface','sizes', 'effect', 'Fcap* *entry',start init,end init, deinit*
	FONTDEF='*font_name*','*typeface*', '*point_size*','*effect*'
	DFONT=*fontname*
Paper Control	LF=*escape_sequence* CR=*escape_sequence* FF=*escape_sequence* NEWPAGE=*escape_sequence* DPAPER=*paper_size* BIN=*sequence.1*,*sequence.2*
Define Horizontal Head Movement	LEADIN=*escape_sequence* LEADOUT=*escape_sequence* HMIPIN=*integer* DCMLOFS=*type* HMIMAX=*integer* OFFSET=*integer* HMIFRC PRSPC=*on*,*off* RELHMI=*boolean*
Define Vertical Head Movement	VMIBEG=*escape_sequence* VMIEND=*escape_sequence* VMIDCML=*type* VMIOFS=*integer*

Function	Pcap File Entry
Define Vertical Head Movement *(continued)*	VMIPIN=*integer* VMIMAX=*integer* VHFORMAT=*type*
Set other Text Parameters	DFHMI=*integer* DFVMI=*integer* RVLNFD=*escape_sequence* DSCHITE=*integer* PITCH=*sequence_1,sequence_2*
Define Character Graphics	CRPUSH=*escape_sequence* CRPOP=*escape_sequence* CRMAX=*integer* HDOTBEG=*escape_sequence* HDOTPIN=*integer* HDOTEND=*escape_sequence* VDOTBEG=*escape_sequence* VDOTPIN=*integer* PPTNBEG=*escape_sequence* FILLFMT=*n,format_sequence* UMOVEND=*escape_sequence* RPTNBEG=*escape_sequence* PPTNEND=*escape_sequence* RPTNEND=*escape_sequence* PPRNBLK=*integerlescape_sequence* RGRYFRM=*integer* RGRYTO=*integer* PPTNSTP=*integer* GRYGRPH PPTNGRY=*escape_sequence*
Define High Resolution Graphics	FILTER='*program_name parameters*' RESTORE
Map Characters and Words	MAP=*character1,character2* MAP=*sequence1,sequence2*
Configure Multiple Copy Facilities of Printers	CPYBEG=*escape_sequence* CPYEND=*escape_sequence* CPYDCML=*type* CPYOFS=*integer* CPYMAX=*integer*

Pcap Syntax

Each entry is in standard Uniplex file section format and obeys the following syntax rules:

Entry	Rule
Literals	Enclose in single quotes. For example: '[7m'
Nulls	Use the @ (at) sign to indicate a null character. It will only be interpreted as a null if it occurs outside quotes. For example:
	CN=9-@-'Example of using @'
	The first @ will be interpreted as a null character, the second as an @ sign.
	Note: You must not specify a null character in the #UPROP section, in the MAP from strings of MAP statements or in non-printer-control sequences (for example: DPAPER).
0 (zero)	Empty. This entry is normally only used where no string is required. For example:
	U=0,0,'/'
ASCII decimals	Do not enclose in quotes. For example: 114.
ESC (Escape)	Use the $ (dollar) sign to indicate ESC.
- (dash)	Link values together using dash. For example:
	I=$-'[7m'-$-'[5m',$-'[m'
, (comma)	Use the comma to separate sequences.
>	Precede send-to-printer string with font and margin selection data.
<	Follow send-to-printer string with font and margin selection data.

> and < are only relevant in .Sx strings (for example .SN).

Note: You must never define the character with the decimal value of 255 within quotes (this is because uprop will confuse it with the escape sequence). Instead, define it outside the quotes using the decimal representation.

For example: NEWPAGE=10-'aaa'-255-'aa'

Defining a Printer Section

You need to define a printer section for any new printer, or printer and cartridge combination added to your system. Each section contains the printer capabilities for that device.

For each individual printer you need to define the following:

o   The way the printer prints different effects (for example **bold** and <u>underline</u>).

o   The horizontal movement parameters.

o   The vertical movement parameters.

o   The parameters for printing character graphics.

Create a Printer Section

When you create a printer section, you must give that printer a name. The syntax for this name used to be:

*printer_name***[/***cartridge***/]**
))

where *printer_name* is the name of the printer (for example, Epson, Kyocera, Laserjet) and *cartridge* is the name of the cartridge used (if any) with the printer.

For example:

#laserjet+/F/
))

Two entries were normally defined for most laser printers, one for portrait and the other for landscape. You specified landscape mode by preceding the cartridge letter with an upper case L. For example:

#laserjet2000/LF/
))

However, since the introduction of the Additional Peripheral Pack (APP), the syntax for naming printers has evolved and now follows the convention:

[*L/*]*modelsymset*[*/card/*]

Where:

L/	Indicates the Landscape name for the printer
model	Identifies the printer, for example hpIII for the HP III printer
symset	Indicates the symbol set that the printer uses
/card/	Indicates any font card or cartridge being used

Therefore, the Hewlett Packard IID printer with its supplied S2 cartridge would have a portrait name of hpIIDecma/S2/ and a landscape name of L/hpIIDecma/S2/.

Where possible the same Pcap entry has been used for both the portrait and landscape orientation of a printer. The orientation is now selected by the INIT token within the Fcap paper definition for the printer. Where possible, this token sets up the printer margins so that they match the margins defined in the paper section. The paper sections define the largest possible printable area for the standard paper sizes.

If you want to specify more than one printer which share common characteristics, you separate the printer and cartridge names with a comma. For example:

#laserjet+/F/,mt910/F/

Include Other Sections

You can put common printer definition information in sections by themselves, and then include them within a printer section using the keyword **INCLUDE='***section_name***'**.

For example, you might put all the common HP printer information in a section called **HP-GENERIC** and then include it using:

#HP_GENERIC
common HP definitions. Can occur
before or after an INCLUDE= that references it
...
))
#hpIIecma
... *a few model-dependent definitions*
INCLUDE='HP_GENERIC'
))

Initialize the Printer

Whenever you start to use the printer, you need to *initialize* it. This sets up the printer ready for use.

The INIT keyword in the printer section is used to specify the escape sequence which initializes the printer. You define the INIT keyword using the following syntax:

INIT=*escape_sequence*

The escape sequence you use will vary according to the printer you are configuring. Refer to the programming manual for information on your specific printer.

For example, the Diablo 630 printer uses the escape sequence ESC SUB I. To initialize the Diablo printer, enter:

INIT=$-26-'I'

In both the Pcap and Fcap files, both initialization strings (using the flag **INIT**) and initialization files (using the flag **INITFILE**) can contain entries to be downloaded to the printer. The initialization files should be placed in the directory *UAP/filters* and the name of a file described in INITFILE should reflect its path relative to the UAP directory.

For example:

/filters/mylaser.in

The order in which the INIT information is sent to the printer is:

1   INIT = string (from Pcap)

2   INITFILE = data (from Pcap)

3   INIT = string (from Fcap)

4   INITFILE = data (from Fcap)

Note: If your printer does not have an INIT escape sequence, it is recommended that you enter the escape sequences which reset the printer to its default state. For example, select the default font, character pitch, ribbon and margins.

Deinitialize the Printer

The DEINIT keyword in a printer section is used to specify the escape sequence to restore the printer to its original state after completing the printout. You define the DEINIT keyword using the following syntax:

DEINIT=*escape_sequence*

The *escape_sequence* varies according to the printer you are using. Refer to the programmer's guide for your printer for details.

Define Printer Effects

You can apply printing effects to documents you prepare using the word processor. For example, you can effect text in **bold**, *italic* or underline.

When you apply an effect to a piece of text, Uniplex highlights the text on the screen. The way the text is highlighted depends on the capabilities of the individual terminal. See the chapter Configuring Terminals in this guide for more details.

You must define the way Uniplex prints each effect. You define the printer effects for each printer. Uniplex uses the following effects:

A=BOLD font	O=PS-SMALL font
(Usually printed overstruck)	(Proportionally spaced small font)
B=Doublestrike	P=PS-NORMAL font
(Often printed using a font)	(Proportionally spaced normal font)
C=Underline all	Q=FX-SMALL font
D=Underline text	(Fixed pitch small font)
E=BOLD & underline	R=FX-NORMAL font
G=SMALL font	(Fixed pitch normal font)
H=LARGE font	S=NORMAL font
I=ITALIC font	T=Shaded overstrike
J=Superscript	U=Strikeout slash
K=Subscript	V=Strikeout dash
M=Double underline	X=Index leader
N=Double underline text	]=Graphics

You define these effects by giving an escape sequence to the printer to enable the effect, and another sequence to disable the effect. The escape sequence you use will vary according to the printer. Use the following syntax:

Effect_letter= turn_on_sequence, *turn_off_sequence*, **[FONT:**'*font_name*'**]**

Refer to the programmer's manual for your printer to find the sequences to use. The font name is the name of a FONTDEF parameter. See the following section for details of defining fontnames.

For example, to define subscript (Effect K) for a Canon printer, enter:

K=$-'L'-$-'[81 C'-$-'[3Y',$-'[0 K'-$-'[120 C'-$-'[3Y'-$-'L'

Note: Do not include any spaces in this sequence.

If your printer does not support a particular effect, omit the effect entry.

Uprop supports the following standard effects:

A   Shadowing
B   Double strike
C   Underscore all
D   Underscore Text
M   Double underline all
N   Double underline text

If you do not define these effects uprop provides them automatically. If you define these effects they are printed according to their definitions.

If an effect requires a font change, for example, change to small font, this effect must be defined as a FONT effect. Use the following syntax:

*Effect letter***=***on_sequence***,***off_sequence***,'FONT:***font_name***'**

where the *on_sequence* selects the font and the *off_sequence* deselects the font. The *font_name* is the name of the font you selected. For example:

E=$-'G'-$-'-1',$-'H'-$-'-0','FONT:BOLD'

If the associated FONTCAPS contains font initialization and de-initialization sequences, the on and off sequences can be substituted with a 0. For example:

E=0,0,'FONT:BOLD'

The *font_name* is used as follows:

1   If *font_name* is either of the words **BOLD** or **ITALIC**, then Uniplex will attempt to find a FONTCAPS definition matching the current font, but with the appropriate effect entry.

       For example, if the font currently in effect is defined with:

       FONTCAPS='Symbol','1-128',ITALIC',...

       and Uniplex needs to change to Bold+Underline effect defined as:

       E=$-'&dD',$-'&d@','FONT:BOLD'

       Uniplex would try to use the entry:

       FONTCAPS='Symbol','1-128','BOLD ITALIC',...

2   If *font_name* is not **BOLD** or **ITALIC**, or the matching FONTCAPS definition cannot be found,
         Uniplex will switch to the FONTDEFS font defined by *font_name*.

3   Should you want to always use the FONTDEFS font named **BOLD** or **ITALIC**, use an "=" instead of
         ":" in the effect definition. For example:

         E=$-'&dD',$-'&d@','FONT=BOLD'

The output sequence when changing effect is:

1   Any *turnon-sequence* specified in the effect definition line.

2   The font initialization sequence from the FONTCAPS line (whether located directly, or via FONTDEF
         lookup).

3   The text in this effect, until the effect is turned off (or a ".FN" font change).

4   Any *turnoff-sequence* specified in the effect definition line.

5   If the end of effect implies a change to a new font, then the new font's initialization sequence,
         otherwise any font de-initialization sequence from the FONTCAPS line (whether located directly, or
         via FONTDEF lookup) of the old font.

Define Overstrike Effects

Some effects, for example overstrike with slash or overstrike dash, require the printer to use overstrike
characters. When you define these effects, use the following syntax:

Effect letter**=**on sequence**,**off sequence**,**overstrike character

U=0,0,'/'

Characters effected with the U effect, are printed as follows:

~~Characters with a strike-out slash.~~

Specify Fonts

The syntax of the command that the user puts in his document to select a particular font is shown below:

> **.FN [***effect***,] [***typeface***,] [***point size***]**

> or:

> **.FN** *fontdef_name*

where:

effect is one of the following: NORMAL, BOLD, ITALIC or BOLD ITALIC, as defined in uniplex.eff.

typeface can include any of the following: Helvetica, Times, Courier, Symbol, Avant Garde Gothic, Bookman, New Century Schoolbook, Helvetica Narrow, Palatino, Zapf Chancery, Zapf Dingbats, or any other that has been defined in the Pcap file.

point size is any value specifying the size of the character, in points, from the range 1 to 128.

fontdef_name is the first argument to a FONTDEF definition in Pcap.

To allow the user to be able to specify any or all of these variables for a font change in his text document, the following entries need to be made in the corresponding Pcap section for the printer to be used:

o   FONTCAPS : these entries specify available fonts for a given printer;

o   **SIZEFMT** : this line specifies the required format of the point size value which the given printer needs to work;

o   **FONTDEF** : these entries specify the features of standard default fonts that can be used by the printer; such as LARGE or BOLD.

Each of these entries is described below, together with the syntax to use and examples.

FONTCAPS

This entry specifies each supported combination of typeface, effect and point size for the given printer in whose section it appears.

The initialization and de-initialization sequences for the specified font are included, together with a reference to the Fcap file entry containing the standard character width table, to be used for scaling by the print formatting program uprop, according to the required point size. (See the later section Configuring Fcap for more details on character width definition.)

The syntax for the entry is as follows:

FONTCAPS='typeface','sizes','effect','Fcap entry', start init, end init, deinit

where:

typeface	is the name of a typeface supported by the printer, including:
	Helvetica, Times, Courier, Symbol, AvantGarde, Bookman, NewCenturySchlbk, HelveticaCondensed and Palatino
sizes	are the supported point sizes for the given printer and typeface combination. This can be one or more individual sizes separated by commas, or a range of sizes with a dash (-) separator. The maximum permitted point size is 128.
	For example: **24** or **8, 10, 36** or **8-24**.
effect	is the printing effect that is to be used with that particular typeface and point size, which must be one of:
	NORMAL, BOLD, ITALIC, or BOLD ITALIC.
	Note: Uniplex only actually recognises the last three of these; any other string is treated as NORMAL.
Fcap entry	is the name of the Fcap section (see later on in this chapter), where the individual character widths are defined for this typeface and point size combination. Those widths are then scaled by uprop to the nearest character width for the point size requested.
	Note: There must be a POINTS= entry in the Fcap section for that printer for point size scaling to work (see the later section Configuring Fcap for details).
start init	is the part of the initialization sequence that comes before the point size.

end init   is the part of the initialization sequence that comes after the point size.

deinit   is the de-initialization sequence for the font.

You should include a separate FONTCAPS entry in the Pcap section for each supported font that is to be available for the given printer.

For example, the following shows a valid FONTCAPS entry for a Kyocera printer in HP emulation mode.

FONTCAPS='Times','10','NORMAL','PS-NORMAL',$-'(s1p','v0s0b5T',0

SIZEFMT

This line specifies the number format of the point size value to be sent to the given printer.

The syntax is shown here:

SIZEFMT=f

where:

f is a value specifying the number format, and can be one of the following:

   0   specifies a single-byte value; e.g. size 20 would be **CTRL-T**.

   -1   specifies a floating-point value.

   n   specifies an integer comprising n digits.

So, if the entry is:

SIZEFMT=2

and the required point size is 8, the value 08 will be sent to the printer between the two initialization strings.

FONTDEF

These entries define the standard fonts available for a given printer, whose Pcap section they appear in. They include such font names as PS-NORMAL, and FX-SMALL.

The syntax for this entry is as follows:

FONTDEF='font_name','typeface',point_size,'effect'

where:

font_name   is the name of the font that will be defined; e.g. LARGE, PS-SMALL. These are the
                    names used in the document for selecting a particular font; for example, .FN LARGE
                    and .FN PS-SMALL.

typeface   is one of the typefaces supported by the specified printer.

point_size   is one of the sizes that are supported by the given typeface as specified in the
                    FONTCAPS entries.

effect   is the one of the printing effects that is available for the typeface/point_size
                    combination, as specified in the FONTCAPS entries.

                    In other words, NORMAL, ITALIC, BOLD or BOLD ITALIC.

A separate FONTDEF entry should be included in the Pcap section for each standard font that the given
printer is going to use.

For example, the following FONTDEF entry would be included for a printer to use a standard font called
FX-NORMAL, printing in normal, Courier, 12-point type.

FONTDEF='FX-NORMAL','Courier',12,'NORMAL'

As shipped, Uniplex contains a number of standard fonts described below that can be used with any of the
supported printers, these printers are listed in the latest version of the Release Notes.

These fonts can be defined by using the FONTDEF command described above.

Note: Uniplex also supports the previous version of the font definition command, FONTINIT, used by
earlier releases of the product.

Name	Description
FX-NORMAL	Specifies a font to print normal text in fixed pitch. If the printer does not have this font, then define the font name to select an alternative font to be used instead.
PS-NORMAL	Specifies a font to print normal text proportionally spaced. If the printer does not have this font, then define the fontname to select an alternative font to be used instead.
NORMAL	Selects either FX-NORMAL or PS-NORMAL, depending on which is most likely to be used as the default font.
FX-SMALL	Specifies a small fixed pitch font.
PS-SMALL	Specifies a small proportionally spaced font.
SMALL	Selects either FX-SMALL or PS-SMALL, depending on which is most likely to be used more frequently.
ITALIC	Specifies an italic font.
BOLD	Specifies a bold font.
LARGE	Specifies a large font.

In addition to specifying the fontnames to use, you need to specify which of those fontnames uprop will use as the default font.

Specify the following keywords:

Keyword	Description
DFONT=*fontname*	Specifies the default font to use.
	For example, to specify the NORMAL font, enter:
	DFONT=NORMAL
	Note: Do not enclose the name of the font in quotes.

Controlling the Paper

You define keywords in Pcap to control the way paper is fed through the printer and to define the default paper size.

Define the following keywords:

Keyword	Description
LF=*escape_sequence*	Specifies the escape sequence to perform a line feed (moving the print head down by one line). By default uprop uses CTRL j as the line feed which is the default for most machines.
CR=*escape_sequence*	Specifies the escape sequence to perform a carriage return (returning the print head to the leftmost print position). If this is omitted then uprop assumes it to be CTRL m, which is the default for most printers.
FF=*escape_sequence*	Specifies the escape sequence which performs a form feed.
	Note: Some printers like the laserjet family use the INIT sequence to eject the page. Do not use this sequence as it resets the printer as well as ejecting the page.
NEWPAGE=*string*	Specifies a new page. This is only required by some printers (for example postscript printers). For example, for a postscript printer, enter:
	NEWPAGE=10-'%% Page:??'-10-'PAGE ('
DPAPER=*string*	Specifies the default paper size to be used when using this printer. Use the syntax:
	DPAPER=*paper-size*

Keyword	Description
DPAPER (*continued*)	where *paper-size* is a paper size defined in Fcap. Do not include the prefix from the Fcap keyword which specifies the printer. For example if the entry in Fcap is as follows: #laserjet/F/:A4)) then the **DPAPER** keyword for that printer section may be defined as: DPAPER='A4' Note: DPAPER can be overriden by the paper styles setting in the print styles file. Refer to the section, Configuring Fcap, for details of setting the paper size.
BIN=*sequence.1,sequence.2*	For use with printers with more than one paper bin. Specifies the bin from which to select paper. *sequence.1* is the escape sequence to select bin 1, and *sequence.2* is the escape sequence to select bin 2. If uprop was *invoked* with the -b option, (see the appendix Program Usage and Invocation in the Uniplex Technical Guide for more details), uprop sends sequence.2 to switch to bin 2, prints n pages of the document then sends sequence.1 to switch back to bin 1. Note: Bin selection is usually more usefully performed by defining appropriate .SN strings in Pcap, so the user can insert ".SNBIN1" and ".SNBIN2" directives within the document. See the section Sending Direct Command Sequences to the Printer later in this chapter.

Define Horizontal Head Movement

Uprop uses the *horizontal motion index* (HMI) to achieve correct spacing between words and characters. The horizontal motion index specifies the distance the print head moves horizontally. HMI units are usually measured in 1/120 inches or 1/300 inches. If your printer is not capable of this type of movement, then true proportionally spaced printing is not possible.

Horizontal head movement works in two different modes:

o   Fixed pitch mode

       When the printer is in fixed pitch mode, then each printable character, including the space character, moves the print head by the specified number of HMI units after printing each character.

o   Proportional Spacing mode

       When the printer is in Proportional Spacing (PS) mode, then setting the HMI will effect the print-head movement in one of the three ways listed below.

       1   For printers that fully implement HMI, the print head moves the number of HMI units you specify each time it prints a space character. When it prints any other character, the print head moves the width of the character. Examples of printers that fully implement HMI are the Laserjet, Laserjet+, Laserjet2000 and Canon LBP-8II.

       2   Some printers do not support HMI when printing in PS mode. When printing characters in PS mode, the print head moves by the width of each character irrespective of the HMI setting. Examples of printers that implement HMI in this way are the Qume laser installed with the Qume driver.

       3   Some printers use HMI to indicate if they are printing in PS or fixed pitch mode. Usually you set the HMI to 0 to print in PS mode, and set the HMI to any other value to print in fixed pitch.

If your printer supports HMI, set the following keywords:

Keyword	Description
LEADIN=*escape_sequence*	Indicates a change of size of a space character. The initial value of HMI is usually a set number of units of movement. This value stays the same for each space character sent until reset.
	For example, the LEADIN sequence for a laserjet:
	LEADIN=$-'&k'
LEADOUT=*string*	Indicates the end of an HMI size parameter. For example, the LEADOUT sequence for a laserjet:
	LEADOUT='H'
	The complete sequence to change the HMI for the laserjet is therefore:
	$-'&k#H'
	where # is the new value for HMI.
HMIPIN=*integer*	Indicates the denominator for size unit in fractions of an inch, multiplied by 100. The multiplication by 100 is to cater for printers that allow movement of *fractions of units*, and also to help internal calculations of head movement by *uprop* no matter what size units any printer may use.
	For example, a printer with a head movement size unit of 1/300 inches would have the following definition:
	HMIPIN=30000

Keyword	Description
DCMLOFS=*type*	Indicates the format of the number of units to alter HMI movement by. Use any of the following valid types:

0   indicates that the number is a single byte, the decimal equivalent of which forms the required number. For example, to alter movement to twenty units enter CTRL-T.

-1   indicates that the number is a string of ASCII digits, made up of any number of digits followed by an optional decimal point and any number of decimal places.

-2   specifies that the value is to be output as a 2 byte binary value with the high order byte first.

-3   specifies that the value is to be output as a 2 byte binary value with the low order bit first.

Any other positive integer value indicates that the number is sent as an ASCII string of digits. The size of the string is specified by the parameter. For example, setting DCMLOFS=2 requires a two digit number. To move twenty units, enter 20. Note that single digit parameters must be prefixed by zero, so movement of nine units would require the number, 09.

| HMIMAX=*integer* | Indicates the maximum number of units that the print head can be moved. Note that this may not necessarily be a hardware maximum, but rather the maximum number that uprop is capable of handling. |

Keyword	Description
HMIMAX (*continued*)	For example, the Toshiba P1351 always requires a two character number to be sent. Values between 1 to 99 are sent as decimal numbers, but any numbers above this are sent in a hexadecimal/decimal format to restrict length of a string. In cases such as this, it is best to set HMIMAX=9900. Note: Like HMIPIN, this number must also be multiplied by 100.
OFFSET=*integer*	Indicates the number you need to add to the HMI movement number, so that the location is correctly processed by the printer. The concept is similar to screen cursor addressing, where it is required to add an offset parameter to line/column number. Normally, OFFSET will be zero. Refer to the programming manual for your printer to find the OFFSET to use.
HMIFRC	No value is required. By default all head movement values are sent as whole numbers. However, if the printer is capable of moving the print head in fractions of a size unit, include this flag. If the printer is not capable of this type of movement, omit this flag.
PRSPC=*on,off*	In the majority of cases, a printer is capable of printing in both proportionally spaced mode, where each character takes up a varying amount of space, or fixed pitch mode, where all characters occupy the same amount of space. When any given font is selected, uprop examines the definitions for that font, and if all characters are defined as being the same size, prints in fixed pitch mode. Once a different font is selected, where characters are defined as being of different sizes, uprop prints proportionally spaced.

Keyword	Description
PRSPC (*continued*)	If this flag is defined, the *on* sequence turns off proportionally space mode and prints in fixed pitch. The *off* sequence turns on proportionally spaced mode.
	This parameter is particularly useful for printers such as the Toshiba P1351, where the current HMI setting is used to set the spaces between the characters as well as words. In such a case, the *on* string sets HMI to a minimal value, typically 1. The *off* string is set to null (0), as spaces between words are dynamically calculated by uprop, and so the HMI value will be reset at the end of each word.
	In some cases, such as the Canon LBP-811, this flag is not required, as font selection results in correct character spacing.
RELHMI=*boolean*	Boolean flag. Present for Relative Horizontal Movement capability.

Uprop usually sends the keywords described above in the following sequence for each printed word:

1   LEADIN   Start of HMI information.

2   *nnn*   Where *nnn* is the number of size units to which HMI value is to be set.

3   LEADOUT   End of HMI information.

4   SPC   Space character. Width of this space character will be nnn/HMIPIN/100 inches. (nnn/HMIPIN x100)

5   PRSPC ON   If defined, this string is output to correctly set the space between characters within a word.

6   TEXT   One word is printed. Characters are evenly spaced, according to the new value of HMI set by the PRSPC_OFF keyword, or according to the characteristics of the current font.

| 7 | PRSPC OFF | If defined, this string sets the printer back to proportional spacing mode. |
| 8 | NEXT WORD | The cycle is repeated again to define space size, print a space between words, then print the next word. |

Defining Vertical Head Movement

Uprop uses the *vertical motion index* (VMI) to achieve correct spacing between lines. The vertical motion index specifies the distance the print head moves vertically. VMI units are usually measured in 1/48 inches. If your printer does not support VMI, uprop may fail to support the double underline effects, and vary the line spacing.

If your printer supports VMI, specify the following keywords:

Label	Description
VMIBEG=*escape_sequence*	Changes the size of a line space. The line space size stays the same until reset. For example, for the VMIBEG keyword for the laserjet, enter: VMIBEG=$-'&l'
VMIEND=*escape_sequence*	Indicates the sequence to end the VMI size parameter. For example, the VMI sequence for a laserjet: VMIEND='H' The complete VMI sequence is therefore: $-'&l#H' where # is the VMI number.

Label	Description
VMIDCML=*escape_sequence*	This defines the format of the VMI value. You can use any of the following values:
	0 indicates that the number is a single byte, the decimal equivalent of which forms the required number. For example, to alter movement to twenty units enter CTRL t.
	-1 indicates that the number is a string of ASCII digits, made up of any number of digits followed by an optional decimal point and any number of decimal places.
	Any other positive integer value indicates that the number is sent as an ASCII string of digits. The size of the string is specified by the parameter. For example, setting DCMLOFS=2 requires a two digit number. To move twenty units, enter 20. Note that single digit parameters must be prefixed by zero, so movement of nine units would require the number, 09.
VMIOFS=*integer*	You only need to define this keyword for printers which require the VMI to be offset by a fixed number. For example, the NEC Spinwriter. Refer to the programming manual for your printer for details.
VMIPIN=*integer*	This specifies the number of VMI units per inch. The units per inch will vary according to the printer you are configuring. For example, the VMI resolution for the laserjet is 48 units per inch, and the VMI resolution for the Canon LBP is 720 units per inch.
VMIMAX=*integer*	This specifies the maximum value allowed at any time. For example, the Laserjet has a limit of 125 units. You may need to enter the value in hex if the value is over 99, in such a case, the value should be 99. Refer to the programming manual for your printer for details.

Label	Description
VHFORMAT=*type*	Specifies the output format for width & height. *Type* is a number with the following meanings:

0	Any number of ASCII digits but no decimals, for example 12.
-1	Any number of ASCII digits and decimals, for example 12.5.
-2	specifies that value is to be output as a 2 byte binary value with the high order byte first.
-3	specifies that value is to be output as a 2 byte binary value with the low order bit first.
n	(where n > 0) Ascii string of digits **n** bytes long. for example: 012 where *n*=3.

Setting Other Text Parameters

In addition to setting the parameters described above, you need to specify the text-related keywords detailed below. They tell uprop how far the print head should move (horizontally and vertically) when sent a space or linefeed character immediately after being powered on.

Keyword	Description
DFHMI=*integer*	Defines the space width of the default font. You need to define this if you want to emulate *uprint* for compatibility with previous versions of Uniplex.
	Set this keyword to the number of HMIPIN units for a space character defined in Fcap for the default font. This is specified in DFONT.
DFVMI=*integer*	Defines the default line height. Enter the number of VMI size units equivalent to the height of characters defined in Fcap for the default font.

Keyword	Description
DFVMI (*continued*)	Use the syntax: **DFVMI**=*line_height* If the printer has no VMI capability, define the keyword in 1/48 units per inch. Fonts are usually defined in *points*; one point equals 1/72 inch. Therefore: - 12 point font = 12/72inch = 1/6 x 48 VMI = 8 - DFVMI=8 for 12 point font with 1/48 inch VMI. A 12 point font is generally printed at 10 pitch, that is, each character = 1/10 inch. Therefore: - DFHMI = 1/10 x 12000 (HMIPIN) = 1200.
RVLNFD=*escape_sequence*	Defines the sequence to carry out a reverse line-feed (move up the paper by one line) via the printer. In cases where the printer is not capable of producing double underlining as a built-in attribute, this parameter is used to produce the double underline print effect by drawing an underline, then reverse line feeding to produce second underline. (The number of units for this will have already been set to a minimal VMI value by the DSCHITE parameter described below.) If your printer has this capability, use the syntax: **RVLNFD**=*escape_sequence* For example, the reverse line feed sequence for the Diablo 630 is **ESC LF**, therefore the keyword RVLNFD for the Diablo may be defined as follows: RVLNFD=$-10

Keyword	Description
DSCHITE=*integer*	This parameter defines the number of VMIPIN size units that separates the two lines drawn for double underline effect.

Only define this parameter if the printer cannot produce double underline as a built-in attribute. Otherwise omit this parameter. |
| PITCH=*on,off* | This parameter is used to achieve a bold attribute with printers that do not support emboldening as an internal effect. This effect is achieved by printing the character, moving the print head by a minimal amount of horizontal movement using the *on* sequence, reprinting the character, then restoring the original print head position by using the *off* sequence.

Where possible, the *on* sequence will specify movement using HMI, and the *off* sequence will specify the appropriate re-positioning. The *off* sequence can be specified as null, as HMI is self-compensating, though this is usually less efficient.

Use the following syntax:

PITCH=*sequence_1***,***sequence_2*

For example, define the pitch keyword for the Laserjet as follows:

PITCH=$-'&a+6H',$-'&a-6H'

For example, the Toshiba P1351 has this parameter specified as:

PITCH=$-'E01',0 |

Keyword	Description
PITCH (*continued*)	The LEADIN sequence for HMI is ESC e. HMI values for this printer are required in the form of a two digit ASCII string, so 01 specifies that the next space character will be one HMIPIN size unit wide. The space is then printed to move the print head. This printer has no LEADOUT sequence. Following this, the character is reprinted.
TABFONT=*string*	To save having to work out the sequence to restore the print head position, the *off* sequence is specified as 0 (zero). Define this keyword if your printer does not have HMI capability. You define the font by moving the print head for outputting a space. Use the following syntax: **TABFONT=**ic*font_name*

Define Ruled Graphics

Some printers (for example, the HP Laserjet+, Kyocera, Canon LPB-8II and the Apple Laserwriter), have the capability of printing rectangular areas (rules) to any size (within the paper limits) with various patterns or shades. This capability is referred to as ruled graphics.

Uprop can take advantage of this capability to draw lines, boxes and filled areas. Each graphics character is scaled to fit within the allocated area, adjusting both the line height and character width as necessary.

Note: Uprop draws lines using small filled rectangles.

Print Head Push/Pop Position

Some printers are capable of storing the current head position, then at a later stage restoring the print head to its original position. This is achieved by sending an escape sequence to tell the printer to store the current position (*pushing* the cursor position), and another escape sequence to tell the printer to restore the last cursor position (*popping* the cursor position). This optimizes the amount of data sent to the printer, since graphics requires a lot more print head movement than text.

If your printer has this capability, refer to the programmer's manual for your printer. You need to define the following keywords:

Keyword	Description
CRPUSH=*escape_sequence*	Defines the sequence used to *push* the current print head position into memory.
CRPOP=*escape_sequence*	Defines the sequence used to *pop* the original cursor position stored by CRPUSH.
CRMAX=*integer*	Enter the maximum number of different cursor positions that may be stored or restored by the CRPUSH/CRPOP parameters.
	Note: If this feature is available on the printer, (even if it supports only one level), it should be defined for optimization purposes. In addition, these flag is vital for the operation of ruled graphics.

Types of Ruled Graphics

Uprop supports two implementations of ruled graphics capabilities:

Type One

Some printers (for example the HP Laserjet+) draw graphics using the following sequence:

1   Positions the cursor in the upper left corner of the graphic area to be printed.

2   Sends both horizontal and vertical rule/pattern size escape sequences to the printer.

3   If the graphic area is to be filled with a predefined pattern or gray scale then sends the escape sequence to select that pattern.

4   Instructs the printer to print the rule or pattern.

Type Two

Some printers (for example, the Canon LBP-8II) draw graphics in the following sequence.

1   Positions the cursor in the upper left corner of the graphic area to be printed.

2   Marks that position and selects the pattern to be printed.

3   Positions the cursor in the lower right corner of the graphic area to be printed.

4   Commands the printer to print the rule or pattern.

Define Ruled Graphic Labels

To print ruled graphics, you need to define the following keywords:

Keyword	Description
HDOTBEG=*escape_sequence*	Defines the start sequence for the horizontal width of a fill pattern.
HDOTPIN=*integer*	Defines the value for the horizontal resolution of graphic images.
HDOTEND=*escape_sequence*	Defines the end sequence for setting the horizontal width of graphic images.
VDOTBEG=*escape_sequence*	Defines the start sequence for the vertical height for graphic images.
VDOTPIN=*integer*	Defines the vertical resolution in increments per inch for graphic images.
VDOTEND=*escape_sequence*	Defines the end sequence for setting the vertical height for graphic images. If the printer has no leadout sequence, then omit this keyword.

Keyword	Description
PPTNBEG=*escape_sequence*	When Uniplex prints fill patterns, including lines, the print head moves to the top left corner of the area to fill, sends a start sequence then moves to the bottom right corner of the area and sends an end sequence.
	Defines the start sequence for marking the top left of an area to fill.
PPTNEND=*escape_sequence*	Defines the end sequence for marking the bottom right of an area to fill.
PPTNRUL=[*int/str*]	Selects a *ruled* or *hatched* fill pattern rather than a *gray-scaled* pattern. The type of fill pattern used is decided internally by *uprop*, dependant on the keywords described below. This parameter may be either a string or a single decimal number. If gray-scaled patterns are required, or are not supported, then define this parameter as null (0).
FILLFMT=*n,Format_Sequence*	Where *n* can be any of the following:

Where *n* can be any of the following:

1-10	for the ten fill characters (**FILL** in Tcap)
11-16	for the six plot characters (**PLOT** in Tcap)
17-22	for the six join characters (**JOIN** in Tcap)

Any pattern not defined with a FILLFMT is drawn using the old PPTN? and RPTN? entries.

Format_Sequence is an escape sequence which can include:

String	Replaced by:
%up	*height* of area to be filled
%down	*(-height)* of area to be filled
%right	*width* of area to be filled
%left	*(-width)* of area to be filled

Keyword	Description
	For example:
	FILLFMT=1,$-'*c'-%up-'B'-$-'*c'-%right-'A'-$-'*c1G'-$-'*c2P'
	VLINEFMT Used to compose the vertical line sequence. Same format as **Format_Sequence** above.
	HLINEFMT Used to compose the horizontal line sequence. Same format as **Format_Sequence** above.
	Notes:
	1 The entries **MHIPIN**, **HDOTPIN** and **VDOTPIN** are unchanged and are still required to control the range of numeric values in the sequences above.
	2 The presence of the new flags **VHFORMAT**, **FILL**, **FILLFORMAT**, **VLINEFORMAT** and **HLINEFORMAT**, indicating the new approach to drawing graphics, mean that the existing flags **HDOTBEG**, **HDOTEND**, **VDOTBEG**, **VDOTEND**, **PPTNBEG**, **PPTNEND**, **PPTNBLK**, **PPTNSTP**, **RGRYFRM**, **RGRYTO**, **RPTNFRM**, **RPTNTO**, **RPTNBEG** and **RPTNEND** are not required.
	Attempts to mix these new and old flags results in the uprop message:
	uprop : Old style entries no longer required: XXXX
RPTNBEG=*escape_sequence*	Defines the sequence to start selecting a fill pattern. This sequence is followed by a code to select the required fill pattern.
RPTNEND=*escape_sequence*	Defines the sequence to end the fill pattern selection.
PPRNBLK=[*int/str*]	Selects the black fill pattern for drawing lines. This may be a string or a single decimal number, depending on the printer you are configuring.

Keyword	Description
PPTNRUL=[*int/str*]	Selects a *ruled* or *hatched* fill pattern rather than a *gray-scaled* pattern. The type of fill pattern used is decided internally by *uprop*, dependant on the keywords described below. This parameter may be either a string or a single decimal number. If gray-scaled patterns are required, or are not supported, then define this parameter as null (0).
RGRYFRM=*integer*	Uprop uses many different values for contrasting fill styles. The majority of printers have these fill patterns defined internally. There may be any number of these internal patterns, usually numbered. This parameter defines the first pattern to be used as fill character 0.
RGRYTO=*integer*	This parameter defines the last pattern number to be used.
PPTNSTP=*integer*	Define the increments for scaling. For example, if all ten fill patterns are gray-scales, then setting this parameter to: PPTNSTP=10 results in pattern 0 being 10% dark, pattern 1 20% dark, and so on. If this parameter is set to 0, or not defined, then *uprop* calculates the scaling proportions internally.
GRYGRPH	Defines the type of pattern to be used, *gray-scale* or *hatched*. By default, uprop uses hatched patterns rather than *gray-scales*. By default, pattern selection is done using the PPTNRUL parameter. Only define this parameter if gray-scales are required.

Keyword	Description
PPTNGRY=*escape_sequence*	This keyword specifies the escape sequence to send after PPTNBEG and before PPTNEND to request the printer to print a gray-scale.
RPTNFRM=*integer*	Specifies the number of the first predefined pattern shape.
RPRNTO=*integer*	Specifies the number of the last pattern shape.

How Uprop Uses these Keywords

Uprop uses the ruled graphics capability to emulate a line draw character set. It draws each graphic character scaled to the current line height and column width. Each graphic character is drawn in a rectangular area filling the height of the line and width of the column.

Usually the cursor is positioned at the bottom of a line. Uprop moves it to the left edge of the column (where the graphic character is to be drawn) before drawing the character.

The following lists the sequence in which parameters are sent when drawing a line.

Keyword	Description
GRAPHICS ON	This sequence is always sent when using ruled graphics rather than high-resolution graphics.
CRPUSH	Stores the current cursor position, prior to printing the graphic image.
UMOVBEG	The print head is normally moved up to the center of the character cell, so that the graphic image starts at the center of the matrix. With most printers, the default position to start printing is at the extreme bottom of the character matrix. This sequence indicates that movement is about to begin.
nnn	The head moves *nnn* units as specified in the UMOVPIN parameter.
UMOVEND	The end sequence for print head movement upwards.

Keyword	Description
VDOTBEG	The leadin sequence for vertical size specification for this image.
nnn	The vertical size for this image is *nnn*/VDOTPIN inches long.
VDOTEND	The leadout sequence for vertical sizing of the image.
HDOTBEG	Leadin sequence for horizontal size specification for this image.
nnn	Horizontal size of this image is to be *nnn*/hdotpin inches wide.
HDOTEND	Leadout sequence for horizontal sizing of image.
PPTNBEG	Start sequence for drawing fill pattern.
PPTNBLK	When drawing a line, uprop fills the area with a black fill pattern.
PPTNEND	End sequence for drawing fill pattern. A thin box is drawn and filled with black to produce the required line.
CRPOP	Restore print head to original position saved with CRPUSH.
GRAPHICS OFF	Now that the cursor is correctly positioned, go back to text mode. Either print text, or repeat for next graphic image.

When drawing fills as opposed to lines, parameters are sent in the following sequence:

Keyword	Description
GRAPHICS ON	This sequence is always sent when using ruled graphics as opposed to high-resolution graphics.
CRPUSH	Stores current cursor position prior to printing graphic image.

Keyword	Description
UMOVBEG	The print head is normally moved up to the center of the character cell so that the graphic image starts at the center of the matrix. With most printers, the default position to start printing is at the extreme bottom of the character matrix. This sequence indicates that movement is about to begin.
nnn	Head is moved *nnn* units as specified in UMOVPIN parameter.
UMOVEND	End sequence for print head movement upwards.
VDOTBEG	Leadin sequence for vertical size specification for this image.
nnn	Vertical size for this image to be *nnn*/VDOTPIN inches long.
VDOTEND	Leadout sequence for vertical sizing of image.
HDOTBEG	Leadin sequence for horizontal sizing.
nnn	Horizontal size of this image is to be *nnn*/HDOTPIN inches wide.
HDOTEND	Leadout sequence for horizontal sizing of image.
RPTNBEG	Sequence to start selection of required fill pattern.
nnn	This is a number selected internally by uprop based on either RGRYFRM (if GRYGRPH is specified), or RPTNFRM (if GRYGRPH is not specified).
RPTNEND	This sequence indicates the end of pattern selection.
PPTNBEG	Start sequence for drawing fill pattern.
xxx	This sequence will be either PPTNGRY (if GRYGRPH is set) or PPTNRUL (if GRYGRPH is not set).

Keyword	Description
PPTNEND	This is the end sequence for drawing a fill pattern. A thin box is drawn and filled with black to produce the required line.
CRPOP	Restore print head to original position saved with CRPUSH.
GRAPHICS OFF	Now that the cursor is correctly positioned, go back to text mode. Either print text, or repeat this sequence for next graphic image.
UMOVBEG=*string*	Specifies to move the print head upwards.
	This parameter is similar to VMIBEG, except to VMIBEG, except movement is up rather than down.
UMOVPIN=*integer*	This parameter is similar to VMIPIN, except it is used for upward rather than downward movement.
UMOVEND=*escape_sequence*	Specifies the end sequence to move the print head upwards.

Defining High Resolution Graphics

Uprop can use high-resolution graphics if your printer is capable of this type of output. Define the following keywords:

Keyword	Description
FILTER='*program_name*'	Specifies the graphics filter program name to use for high-resolution graphics. Usually, this will be *gd_matrix*.
	The argument for gd_matrix specifies a definition for this printer in Gcap. For example, the value for this parameter for the HP Laserjet is:
	FILTER='gd_matrix -p hp_jet+.MEDIUM'
	Refer to the section Configuring Gcap later in this chapter for further details.

Keyword	Description
RESTORE	Specifies that the graphics filter restores (or repositions) the print head position to the top of the window after drawing that graph. If not defined, the keyword indicates that the graphic filter leaves the print head position at the bottom of the graph. In this case, uprop attempts to reposition the print head to the top of the graph if possible.

Mapping Characters

Uprop allows you to use the upper half of the ASCII character set (characters 128 to 255). This is useful, for example, if you want to print out foreign text, or mathematical symbols.

You need to refer to the programming manual for your printer to find the ASCII codes to use for your printer.

Use the following syntax:

MAP=*char,sequence*

where *char* is the character to be mapped and *sequence* is the sequence to replace that character on output.

Uprop expects the width of the mapped character to be set in the font section in the Fcap file to reflect the width of the sequence output.

For example:

MAP=167,129
MAP=129,167
MAP=130,'Uniplex'

In this example, the first map maps character 167 to character 129. Uprop expects the print head will move by the width of the character 167 after printing 129. Therefore when a character map is added in the printer section you should modify the width of the character you are mapping to reflect the distance the print head moves after outputting the sequence.

The second line in the example maps character 129 to character 167. The third line shows a map which maps character 130 to the word uniplex, in this case the word uniplex is treated as one character. Note that uprop expects that the print head will move by the distance specified by the width of the character 130.

Mapping Words

In addition to mapping characters, you can create maps to map a word or phrase to another word or phrase. Uprop calculates the head movement from the characters in the sequence. After the map is performed, characters in the sequence are subject to character mapping.

Use the following syntax:

MAP=_sequence1.sequence2_

where _sequence1_ is a string of more than one character to be mapped to _sequence2_.

For example:

MAP='UII','Uniplex-II'

In the example above, the characters UII are mapped to Uniplex-II.

Making Multiple Copies

Some printers have a multiple copy facility which allows each page to be duplicated automatically to the required number of copies. This is similar to the facility available on photocopiers. When you print using the printer's own multiple copy facility, the first page is printed the required number of times, followed by the second page and so on.

If the printer does not have a multiple copy facility uprop reprints the file repeatedly to the required number of copies (which is much slower).

To use the multiple copy facilities of your printer, set the following keywords:

Keyword	Description
CPYBEG=_escape_sequence_	Defines the part of the escape sequence to send to the printer before outputting the value to specify the number of copies.
	For example, to specify multiple copies the Laserjet requires the following character sequence:
	$&l#X

Keyword	Description
CPYBEG (*continued*)	where $ is the ESC character and # is the number which specifies the number of copies you require. In this example, the lead-in escape sequence is $&l, therefore the CPYBEG keyword for the Laserjet would be defined as: CPYBEG=$-'&l'
CPYEND=*escape_sequence*	Specifies the escape sequence to terminate the multiple copy setting. For example, the CPYEND string for the Laserjet is the X character, so enter: CPYEND='X' Note: Some devices do not have a CPYEND sequence. In this case, omit the keyword.
CPYDCML=*type*	Defines the format for sending the value to specify the number of copies, where it could be one of the following formats: 0 indicates that the number is a single byte, the decimal equivalent of which forms the required number. -1 indicates that the number is a string of ASCII digits, made up of any number of digits followed by an optional decimal point and any number of decimal places. Any other positive integer value indicates that the number is sent as an ASCII string of digits. The size of the string is specified by the parameter. For example, setting CPYDCML=2 requires a two digit number. To make twenty copies, enter 20. Note that single digit parameters must be prefixed by zero, so nine copies would require the number, 09.

Keyword	Description
CPYOFS=*integer*	Some printers require the value to be offset by a fixed number (similar to the offset on cursor addressing for terminals).
CPYMAX=*integer*	Specifies the maximum value allowed for setting the multiple copies value at any time. For example, the HP Laserjet has a limit of 99, therefore the CPYMAX for the laserjet may be defined as: CPYMAX=99 Note: Some printers allow the value to be sent in digits from 1-99. For values greater than 99 they use a Hex format. For printers like this, define CPYMAX as 99.

Define the #UPROP Section in Pcap

The #UPROP section of Pcap contains configuration information for the print formatting program, *uprop* to use in creating the desired output for a document. Each keyword is listed below with an explanation of its function.

Keyword	Description
LEADER=*char*	Defines which effect character is to be used for the *dot leader* effect used in indexes and tables of contents.
STYLE=*char*	Defines the characters to use in section numbering (.NS) commands. The default is SPF.

Position	Description
1	Character to use to denote section numbering.
2	Character to use to denote paragraph numbering.
3	Character to use to denote footnote numbering.

Keyword	Description
MODS=*char*	Is a two character position dependent string. The first position denotes the character to be used for setting the display level of paragraph numbering. By default, uprop uses / for the last level only, and // to denote all levels. The second position defines the character to use when resetting all levels of paragraph numbering.
INDEX=*string*	This is an eight character, positionally dependent string, denoting the characters to be used for the .IC and .IX commands. A full description of these characters can be found in the Word Processor section of the Uniplex II Plus Users Guide.

Each position and its meaning is listed below.

Position	Description
1	Delimiter character. Precede the following characters with this in order to put them into the .IC or .IX definition.
2	End of label character (<).
3	Effect character prefix (@).
4	Multiple index entry (+).
5	Sublevel indicator character (>).
6	End of field definitions (!).
7	Start block character (B) this is prefixed by the # character.
8	End block character (E) again prefixed by the # character.

Keyword	Description
NMBSPR=*char*	Defines the separator to use between the .NS numbering options. 0 indicates not to use any separator. For example, if #P#S was specified for an index entry and '+' was defined here, then the output would look like 12+34, where 12 is the paragraph number and 34 the section number.

Keyword	Description
RNGSPR=*string*	This is only used if message 118 (which defines the separator string in indexes) is not present in the file uprop.msg.
	Specifies the string to use to separate ranges in an index. If set to ' to ' then the output could look like Page 3 to 7 after having specified the #B and #E block commands.
GRPSPR=*string*	Specifies the string to use to separate the elements in an index group. For example, if you set the string to be a comma, the output looks like this:
	Birds 1,3,5
TYPSTL=*char*	Typographical characters to use. The character sequence is used as a number sequence. The default is '*+%&', giving a base radix of 4 (always starting with 1). After & the marker will be ** etc.
FTDFLT=*char*	Indicates the numbering style to use for typographical markers. This can be any of the options described in the Word Processor chapter of the Uniplex II Plus Users Guide.
EFFECTS=*string*	Is a two character string, defining the effects characters in use for double underline all and double underline text, respectively.

Sending Direct Command Sequences to the Printer

The .SN, .ST and .SB commands (.S*x* commands) are used to output strings directly to the printer.

Uniplex does not add printer data before or after such .S*x* strings. That is, the string is not automatically preceded by font and margin change data or followed by a carriage return and font and margin reset data.

If you want such printer data to be used, you must specify it. You use two *Pcap* syntax directives to do this. These directives are the characters less than (<) and greater than (>), and have the following effect in .S*x* commands.

o   <   Specifies that the string is to be preceded by font and margin selection data. When the string
                  is output, the following events occur:

          -          The print head moves to the effective print left margin position (as specified by the print style).

          -          The default font is selected if it is not currently selected.

          -          HMI (space width) is set to the width of a space in the default font.

          -          If this is a .SN operation, and no text has yet been output on the current page, the header is
                     output (if there is one).

o   >   Specifies that the string is to be followed by font and margin selection data. When the string is
                  output, the following events occur:

          -          If the output before this .Sx command was in PS mode, PS mode is reselected.

          -          A flag is set so that the following text (if any) will be preceded by HMI and VMI setting
                     sequences.

These directives are not position-dependent, they can occur anywhere in a string, but will usually be placed
before and after it. Following Pcap conventions, they must be separated from the string by a hyphen (-).

For example, the Pcap line:

BIN2=$-'xy'

would cause any **.SN BIN2** directive in a document to result in the three characters **ESC x y** being output,
without any associated printer data.

However, the Pcap line:

BIN2=<-$-'xy'->

would cause any **.SN BIN2** directive to result in these characters being preceded and followed by the
printer-data listed above.

Note: These directives can also be specified in a document in conjunction with a .Sx command, using the
standard .Sx string_in_Pcap syntax (for example: .ST <-$-'xy'-> instead of .ST BIN2).

Configuring Fcap

The Fcap (Font Capabilities) file is split into two sections; paper definitions and font definitions. You define the size of paper on which you print in the paper definitions section. In the font definitions section, you define the physical size of each character printed in each font type.

File Format and Layout

Section of Chapter	Fcap File
Define Paper Type	**#***printer_name*:*paper-name*
Define Paper Size	LENGTH=*integer* WIDTH=*integer* LMARGIN=*integer* RMARGIN=*integer* TMARGIN=*integer* BMARGIN=*integer* OFFSET=*integer*
Define Font Sizes	**#***printer-name*:*font-name* POINTS=*floating-point to 1 decimal place* UPPER=*integer* LOWER=*integer* DIGIT=*integer* SIZES=*char*=*n***[,***char*=*n***..]** OTHER=*integer* HEIGHT=*integer* CAVG=*integer*

Fcap Syntax

Each entry is in standard Uniplex file section format and obeys exactly the same syntax rules as Pcap (see the earlier subsection Pcap Syntax in the section Configuring Pcap).

Define Paper Type

You need to define each paper type that you use with each printer. In addition, you must define the default paper size you use with each printer.

Usually, you need to define the following paper sizes for use with Uniplex:

o   A4 (Standard European Size)
o   A5 (Standard European Size)
o   8x11 (Standard American Size)
o   5x8 (Standard American Size)
o   Wallet (Personal Organizer Size)

You can make your entries printer dependent, or general. Uprop first searches for a printer dependent entry. If none is found, then uprop searches for a paper name only entry.

Define a paper type as follows:

1   Edit the Fcap file. This is located in the UAP directory.

2   Make the entry:

        #*printer-name***:***paper-name*
        ))

        where *printer-name* is the name of a printer defined in Pcap, and *paper-name* is the name of the paper defined by the DPAPER keyword in Pcap. Alternatively, you can specify a paper name using the -P flag in uprop.

        For example, if you are using a LBP-8II printer with a default paper size of A4, make the following entry:

        #lbp_8II:A4
        ))

        If uprop cannot find a printer-specific entry, then uprop searches for a general entry of A4.

3   When the entry is as you require, re-index the file to speed up access. For example:

        uindex Fcap

Define Paper Size

Define the following keywords:

Keyword	Description
LENGTH=*integer*	Defines the physical length of the paper in units of 1/6 inches.
	For example the length of an A4 sheet is 70 lines. Therefore, define the LENGTH keyword for an A4 sheet of paper as:
	LENGTH=70
WIDTH=*integer*	Defines the physical width of the paper in units of 1/10 inches.
	For example, the width of an A4 sheet is 8.25 inches. Therefore the WIDTH keyword for an A4 sheet may be defined as:
	WIDTH=83
LMARGIN=*integer*	Specifies the size of any area on the left edge of the paper that cannot be printed on due to hardware restrictions. Define this in units of 1/10 inches.
	For example, the Laserjet printer has a left margin of 2 characters for A4 paper. Therefore, define the keyword as:
	LMARGIN=2
RMARGIN=*integer*	Specifies the size of any area on the right edge of the paper that cannot be printed on due to hardware restrictions. Define this in units of 1/10 inches.
	For example, the Laserjet printer has a right margin of 4 characters for A4 paper. Therefore, define the keyword as:
	RMARGIN=4

Label	Description
TMARGIN=*integer*	Specifies the size of any area on the top edge of the paper that cannot be printed on due to hardware restrictions. Define this in units of 1/6 inches. For example, the Laserjet printer has a top margin of 1/2 inch (3 lines). Therefore define this keyword as: TMARGIN=3
BMARGIN=*integer*	Specifies the size of any area on the bottom edge of the paper that cannot be printed on due to hardware restrictions. Define this in units of 1/6 inches. For example, the Laserjet has a bottom margin of 1/2 inch (3 lines), therefore the keyword BMARGIN for A4 paper for the laserjet printer may be defined as: BMARGIN=3
OFFSET=*integer*	Specifies the default actual margin to use at print time in units of 1/10 inches. This parameter has the same effect as specifying a left margin on the print document screen.

Define Font Sizes

You do not need to define the size of all fonts for all printers you have defined. Once you have defined the character widths for a font with a specified point size for a given printer, the print formatting program, uprop, will automatically scale the characters to the nearest correct size for the font you want to print using that printer.

Font definitions are printer-dependent and take the form:

#*printer-name***:***font-name*

where *printer-name* is the name of the *Pcap* entry, and *font-name* is a value specified in a FONTDEF parameter within Pcap.

For example, using a LBP 8II printer, and a font called NORMAL, make the following entry:

```
#lbp_8II:NORMAL
))
```

If uprop cannot find an entry for a specific printer, it displays an appropriate message and abandons the process.

Note that unlike size parameter entries in Pcap, where units may be selected by the configurer, all width sizes in font definitions within Fcap must be in units of 1/12000 inches, and all height definitions must be in units of 1/48 inches.

You can define the following keywords for each font:

Keyword	Description
POINTS=*floating point*	Specifies the point size, to one decimal place, of the font whose character widths are being defined (to be used by uprop for scaling).
UPPER=*integer*	Defines the width of all upper case characters.
LOWER=*integer*	Defines the width of all lower case characters.
DIGIT=*integer*	Defines the width of all numeric characters.
SIZES=*c=n* [,*c=n*..]	Defines the size of an individual character *c* as size *n*. Character *c* may be defined as a character itself within single quotes, or a decimal equivalent value. The maximum permitted point size is 128.
OTHER=*integer*	Defines the width of all characters not already defined using any of the above parameters.
HEIGHT=*integer*	Defines the height of all characters. This keyword is used to set line spacing via the .SP command with no arguments.
SCALING=*integer*	Defines the scaling rules for scaleable fonts (see the next section).
	Note: Either HEIGHT or SCALING must be specified, but not both.

Keyword	Description
CAVG=*integer*	Defines the width of the average character size for printing normal text. This setting is used by uprop when printing to a terminal. The setting should reflect the average pitch of the font when printing normal text.

Note that the parameters defined above are read by uprop in the order in which they appear. The exception to this is the OTHER keyword which is used as a default for any undefined characters. Any subsequent duplicate definition overrides previous definitions. For example, if you have the following definitions:

SIZES='A'=1300,'B'=1500, 'C'=1300...
UPPER=1100

The definition of UPPER (all upper case letters) overrides the definition of the SIZES line. For example all upper case characters would be taken to be 1100.

Fcap Syntax to Describe Scaleable Fonts

The SCALING keyword is used to support scaleable fonts such as those in HP's PCL-5:

SCALING=<DPI>,<PTperIN>,<DesignUnits>[,<Algorithm>]

If this is present, then the units in the entry's SIZES table specify the Horizontal Escapements (Hesc) as supplied by the printer vendor's Font Matrix tables (for example, HP's TFM files), rather than Uniplex's usual 1/12000 units.

When using such a SCALING entry, Uniplex uses the following formula to compute the size that any given character takes on the printer:

$$\text{size} = \text{ROUND} \left(\frac{\text{Hesc} * \text{DPI} * \text{PTperIN} * \text{PointSize}}{\text{DesignUnits} * \text{POINTS}} \right) * \frac{12000}{\text{DPI}}$$

Where:

Hesc	is the **H**orizontal **Esc**apement from the SIZES table.
DPI	is the **D**ots **P**er **I**nch resolution of the printer (for example 300 or 600). This is always an integer value.
PTperIN	is the **P**oin**t** size **per In**ch for the font.

PointSize   is Point Size being used.

DesignUnits   are the Design Units for the font.

POINTS   is the value specified in the POINTS keyword for this font.

ROUND (...)   rounds the value using one of four algorithms defined by the optional field on the SCALING line. This field can be one of:

Algorithm   0   (or omitted; this is the default.) Use classical rounding algorithm (that is, <0.5 rounds down, and >=0.5 rounds up.

    1   Round down. That is, it takes only the integer value. This is required for IBM scaled font support.

    2   Round value up. That is, it takes the next integer value.

    3   Use a double-rounding algorithm. This is for use with TrueType fonts. The algorithm is:

$$ppem = ROUND (DPI * PTperIN * PointSize)$$

$$size = ROUND (\frac{Hesc}{DesignUnits} * ppem) * \frac{12000}{DPI}$$

Configuring Gcap

The Gcap file defines the capabilities of terminals, printers and plotters for producing high-resolution graphics. This section describes the keywords that you need to set for printers and plotters. For information on the Gcap keywords to set for terminals, see the chapter Configuring Terminals in this guide.

Three filters exist to support the production of graphics:

o   The *gd_matrix* filter supports many types of printers, including laser and dot-matrix printers.

o   The *gd_pscript* filter is used to print graphics using a *postscript* printer (for example, an Apple Laserwriter). gd_pscript outputs graphics information in postscript format. You cannot configure gd_pscript except by editing the postscript. Refer to your postscript manuals for details of how to achieve this.

o   The *gd_plotter* filter produces plotter output in Hewlett Packard Graphics Language (HP-GL) format from Uniplex graphics files (RGIP format). This filter works for HP-GL Plotter emulations. Currently the filter has been configured to drive the HP 7475A and GP 1760 plotters.

You set the graphics filter that Uniplex uses by setting the FILTER keyword in Pcap. See the sub-section, Defining High Resolution Graphics in the section Configuring Pcap, earlier in this chapter.

Printer and plotter filters share many keywords. However, their configurations differ in several respects. Accordingly, this section comprises of:

o   keywords used by printer and plotter filters

o   keywords used specifically by printer filters

o   keywords used specifically by plotter filters

        All gd_plotter entries adopt a sensible default value if not set by the Gcap section or if a filter is invoked without a Gcap section. The default values for plotter keywords are included in this chapter.

File Format and Layout

Gcap uses the standard Uniplex file format and layout. Include a section for each printer in the following format:

#section_name
keyword
keyword
keyword
.
.
.
keyword
))

Gcap Entries Using Shared Keywords

Initialization Keywords for Printers and Plotters

Keyword	Description
INIT=*initial_sequence*	Defines the initialisation sequence for a printer or plotter, for example setting the required graphic mode, selecting the correct flow control and defining the sequence for setting the resolution for graphics. The resolution you define depends on the printer, but you will usually have a choice of 300 (high resolution), 150 (medium resolution) and 75 (normal text).
DEINIT=*escape_sequence*	Defines the escape sequence for resetting the printer or plotter to its default state.
OSRESET=*escape_sequence*	Defines the escape sequence for resetting the printer or plotter to its default state for printing directly from the operating system.
URESET=*escape_sequence*	Defines the escape sequence for resetting the printer or plotter to its default state for printing from Uniplex.

Note: The gd_plotter default values for the above keywords is null,"".

Plotter Interfacing

When setting up a plotter, you need to check how handshaking is controlled. On the plotters Uniplex has configured, it was discovered that the method of handshaking was controlled through the first string sent to the plotter and not through the DIP switches. For example, if you wish to use XON/XOFF handshaking, you must inform the plotter of this by way of a string. In addition, it was found that for XON/XOFF, the plotter buffer switch threshold had to be set. On the Hewlett Packard 7475a, it was achieved by placing the following string in the Gcap entry:

INIT=$-'.I512;;17:'-$-'.N;19:'

This string could also have been sent by way of the print spooler initialization file or interface.

Since the method of handshaking varies from machine to machine, Uniplex has not included any of these strings.

Device Parameter Keywords for Printers and Plotters

The following keywords tell the filter about the physical characteristics of the output device used.

The gd_plotter defaults assume A4 paper is used in landscape orientation.

Keyword	Description
XPIXELS=*integer*	Defines the number of pixels across the horizontal. Calculate the number of pixels horizontally by multiplying the number of dots per inch by the number of inches across on the printable area of the paper. gd_plotter default=11041
YPIXELS=*integer*	Defines the number of pixels vertically. You calculate the number of pixels vertically by multiplying the number of dots per inch by the number of inches in the printable length of the paper. gd_plotter default=7722

Keyword	Description
YREV=*boolean*	Defines whether printing should start at the top left of the paper or the bottom left. Set to TRUE to start at the top left, FALSE to start at the bottom left.
	gd_plotter default=FALSE
XY90=*boolean*	Defines whether to rotate by 90 degrees.
	gd_plotter default=FALSE
HEIGHT=*decimal*	Defines the height of the drawing surface in inches. Enter the height of the printable area of the paper in inches.
	gd_plotter default=7.56
WIDTH=*decimal*	Defines the width of the drawing surface in inches. Enter the width of the printable area of the paper in inches.
	gd_plotter=10.81
NCOLS=*integer*	Defines the number of available colors on the printer or plotter.
	gd_plotter default=6

If a plotter is being used, paper size is usually set by switches on the plotter. Alteratively, it may be included as part of the sequence defined for the INIT label, for example:

INIT='IN;PS3'

defines the paper size as A3.

Printer and Plotter Color Keywords

Keyword	Description
RED=*integer* YELLOW=*integer* GREEN=*integer* CYAN=*integer* BLUE=*integer* MAGENTA=*integer* WHITE=*integer* BLACK=*integer*	These integer values allow a value for a given color to be specified. The color keywords relate to RGIP. The integer values are set to default, although may be altered so the value passed to the filter is different. In this way, eight color charts output by uchart can be mapped to device specific colors if supported. For most printers, the following will be found: BLACK=0 WHITE=7
LRED=*integer* LYELLOW=*integer* LGREEN=*integer* LCYAN=*integer* LBLUE=*integer* LMAGENTA=*integer* LGREY=*integer* DGREY=*integer*	You can define these eight colors in addition to the eight standard colors described above.

Note: The default color values for the gd_plotter filter are listed in the gd_plotter section.

Printer and Plotter Fill Style Keywords

Keyword	Description	gd_plotter Default
FSOLID=*integer*	Maps the uplot pattern to the	1
FHORIZ=*integer*	pattern provided by the printer.	2
F45DEG=*integer*		3
FVERT=*integer*		4
F135DEG=*integer*		5
FOLATTICE=*integer*		6
FDLATTICE=*integer*		7
FHOLLOW=*integer*		8
FBRICKS=*integer*		9
ROOFTILES=*integer*		10
FBASKET=*integer*		11
FCHECKER=*integer*		12
FDIAMOND=*integer*		13
FSFRIEZE=*integer*		14

Printer and Plotter Text/Marker Keywords

Keyword	Description	gd_plotter Default
MPOINT=*integer*	Maps the uplot pattern to the	1
MPLUS=*integer*	pattern provided by the printer.	2
MSTAR=*integer*		3
MSQUARE=*integer*		4
MCROSS=*integer*		5
MDIAMOND=*integer*		6
MHBOW=*integer*	These integer values allow the	7
MVBOW=*integer*	default RGIP text face and text	8
TFACE1=*integer*	type orders to be remapped.	1
TFACE2=*integer*		2
TFACE3=*integer*		3
TNORMAL=*integer*		1
TBOLD=*integer*		2
TITALIC=*integer*		3
BOLDITALIC=*integer*		4

Miscellaneous

There are a number of Gcap labels that do not fall into any of the above categories. These are:

Gcap Label	Description
PATTERNMAP=*boolean*	If set TRUE, then line color and fill color are used to select line style and fill style respectively. Since plotters are color devices this mode of operation is turned off by default in gd_plotter.
FILL_WINDING=*boolean*	When a polygon intersects with itself there is a choice. Is the self overlapping area filled or not? The default is to use the "odd-even" rule that fills the area. By setting this entry TRUE then the "winding" fill rule is used; this will not fill such an area.
SMALLER_BOX_FILL=*boolean*	Defines whether the top and right edges of an RGIP BOX are included in the fill. If set TRUE, they are not filled, generally giving better output from TIFF files that have been converted to RGIP using *tiff2rgp*. (Only supported by gd_matrix and gd_x11.)

gd_matrix

The following keywords are used by gd_matrix to specify Gcap printer entries.

Keyword	Description
NPLANES=$integer$	Defines the number of color planes on the printer ribbon you are using. The values stored in the planes can be specified by the color keywords defined below. With a dot matrix printer, this might be done to cause a double strike of a particular ribbon color for some pixels. This can increase the total number of colors that can be displayed.
HALFTONE=$boolean$	Maps colors displayed on a color terminal to half-toned shades. Set to TRUE if you want to map halftones, set to FALSE if you don't want to map halftones.
	By setting the HALFTONE flag to TRUE, filled areas are printed in black through a halftone mask of a density related to the luminosity of the requested color. This takes priority over the setting of PATTERNMAP. However, line colors map to line styles as before.
DRAW_ZERO_OUTLINE=$boolean$	If set TRUE, gd_matrix draws the outline of an RGIP BOX, even if the line width is specified as zero. If set FALSE, gd_matrix behaves like all the other drivers. The default is TRUE for compatability with Version 9.00 and earlier.

The following keywords are used to define halftones used by the gd_matrix filter, in addition to the colors used by printers and plotters described in the Shared Keywords section.

Keyword	Description
HTONE_[1-16]=*integer*	To match the 16 colors discussed in the previous section, 16 halftone masks have been designed. The default mapping of color to halftone is based on the perceived luminosity of the requested color. The keyword HTONE_1 corresponds to RED, this continues through to HTONE_16 that corresponds to color DGREY. There are 16 halftone masks which vary in *darkness*. They are selected by setting the halftone for the color of interest to the number of *black* pixels required out of a count of 16.

For example:

HTONE_5=10

sets the halftone for the color BLUE to a mask that has 10 pixels set out of 16 pixels.

The following keywords are used by the gd_matrix filter to configure various printer features.

Keyword	Description
MTRX_BAND1=*escape_sequence*	Defines the ESC sequence which selects ribbon band 1.
MTRX_BAND2=*escape_sequence*	Defines the ESC sequence which selects ribbon band 2.
MTRX_BAND3=*escape_sequence*	Defines the ESC sequence which selects ribbon band 3.
MTRX_BAND4=*escape_sequence*	Defines the ESC sequence which selects ribbon band 4.
MTRX_GMODE=*escape_sequence*	Defines the ESC sequence which enters graphic mode.
MTRX_GFORMAT=*integer*	Matrix printer sequence, graphic format. Allowable formats:

Entry	Format
0	No format
1	Epson format
2	HP Laserjet format
3	Canon 1000 series format
4	HP Paintjet compressed format
5	HP Paintjet non-compressed format

Keyword	Description
MTRX_LSPACE=*escape_sequence*	Sets the line spacing for the matrix printer sequence.
MTRX_LDEPTH=*integer*	Sets the number of scan lines per linefeed.

The following keywords define Fill Styles used by the gd_matrix filter in addition to the Fill Styles shared by printers and plotters.

Keyword	Description
FSCALES=*integer* FVBRICKS=*integer* FVSFRIEZE=*integer*	Maps the uplot pattern to the pattern provided by the printer.

gd_plotter

The following keywords are used specifically by the gd_plotter filter.

HP-GL Commands

The following Gcap label entries are used to tell gd_plotter about the HP-GL standard command language. The filter generates standard HP-GL commands by default. If you wish to alter the behavior of any standard command, you can do so by adding a Gcap entry to override the default command sequence.

Keyword	Default	Description
HPGL_TERMINATOR	';'	Command Terminator
HPGL_SEPARATOR	','	Parameter Separator
HPGL_END_OF_TEXT	'003'	Label Text Termination Character
HPGL_INITIALISE	'IN'	Initialise
HPGL_VELOCITY	'VS'	Set Pen Velocity
HPGL_SELECT_PEN	'SP'	Select Pen
HPGL_PEN_DOWN	'PD'	Pen Down
HPGL_PEN_UP	'PU'	Pen Up
HPGL_PEN_THICKNESS	'PT'	Set Pen Thickness
HPGL_PLOT_ABS	'PA'	Plot Absolute
HPGL_FILL_TYPE	'FT'	Set Fill Type
HPGL_FILL_RECT	'RA'	Shade Rectangle Absolute
HPGL_LINE_TYPE	'LT'	Set Line Type
HPGL_INPUT_WINDOW	'IW'	Input Window - Clipping
HPGL_CHAR_SIZE	'SI'	Set Character Size
HPGL_CHAR_SLANT	'SL'	Set Character Slant (Italic)
HPGL_CHAR_DIRECTION	'DI'	Set Text Direction (Angle)
HPGL_CHAR_PLOT	'CP'	Plot Character widths
HPGL_DEFINE_EOT	'DT'	Define End of Text Character
HPGL_LABEL_TEXT	'LB'	Output Label Text

Color

The gd_plotter filter can support up to 16 colors, although plotters generally only provide 6 or 8 pens. The RGIP color values output to gd_plotter are mapped directly to pen positions on the plotter's carousel. The following table gives the 16 Gcap color labels together with the default values of pen location.

Keyword	Default Pen	Keyword	Default Pen
RED	2	LRED	2
YELLOW	3	LYELLOW	3
GREEN	4	LGREEN	4
CYAN	2	LCYAN	2
BLUE	5	LBLUE	5
MAGENTA	6	LMAGENTA	6
WHITE	1	LGREY	1
BLACK	0	DGREY	1

The default entries assume only six pens are available. CYAN is therefore mapped to RED.

If the default color mappings are to be altered, each color should have assigned to it a pen number.

Pens should be loaded into the carousel in the following order:

Pen Position	Pen Color
1	Black
2	Red
3	Yellow
4	Green
5	Blue
6	Magenta

It is also possible to redefine the line thickness of the pens used. The default value is 0.3mm, but if this is incorrect, the labels HPGL_THICK1 to HPGL_THICK16 should be assigned appropriate values in the Gcap entry.

For example, if pen 4 is 2mm thick, the following line should be added to the Gcap:

HPGL_PTHICK4=2.0

Line Styles

Eight line styles are supported by gd_plotter. HP-GL defines 6 line patterns that can be repeated over a programmable length. Thus a particular line pattern can be condensed or stretched to yield a distinct line style. The eight standard Gcap line style labels are supported to allow line styles to be swapped around. These are shown below:

RGIP Name	Gcap Label	Default Value
1 - Solid	LSOLID	1
2 - Dot	LDOT	2
3 - Dash	LDASH	3
4 - Dash Dot	LDASHDOT	4
5 - Dot Dot	LDOTDOT	5
6 - Dash Dash	LDASHDASH	6
7 - Dot Dash Dash	LDOTDASHDASH	7
8 - Dash Dot Dot	LDASHDOTDOT	8

The actual HP-GL line pattern and repeat length are also configurable using the following labels:

Gcap Label	Default Sequence	Gcap Label	Default Sequence
HPGL_LTYPE1	''	HPGL_LENG1	''
HPGL_LTYPE2	'1'	HPGL_LENG2	'2.0'
HPGL_LTYPE3	'2'	HPGL_LENG3	'2.0'
HPGL_LTYPE4	'4'	HPGL_LENG4	'2.0'
HPGL_LTYPE5	'1'	HPGL_LENG5	'1.0'
HPGL_LTYPE6	'2'	HPGL_LENG6	'1.0'
HPGL_LTYPE7	'6'	HPGL_LENG7	''
HPGL_LTYPE8	'6'	HPGL_LENG8	'2.0'

Please note that the above labels are assigned character sequences, not integers or decimals. The repeat length unit is a percentage of the diagonal of the drawing area.

Fill Styles

The gd_plotter filter provides 14 fill styles. The rectangle fill capacity makes use of the plotter's built-in ability. Other shapes, such as Arcs, Segments and Polygons are filled using filter-generated line drawing commands. A table of the Fill Style Keywords together with default values is provided in the Shared Keyword section of this chapter.

gd_plotter receives an RGIP fill style index value which is mapped by the table mentioned above to give the fill type to be used. This value is used as an index to the following tables to obtain the HP-GL characteristics of the fill style. These characteristics are: fill type; fill line spacing; and fill line angle.

Gcap labels are provided to override the default HP-GL sequences that select these fill characteristics:

Gcap Label (Fill Type)	Default Sequence	Gcap Label (Fill Space)	Default Sequence	Gcap Label (Fill Angle)	Default Sequence
HPGL_FTYPE1	<solid>	HPGL_FSPACE1	''	HPGL_FANGLE1	
HPGL_FTYPE2	<paral>	HPGL_FSPACE2	<loose>	HPGL_FANGLE2	<00>
HPGL_FTYPE3	<paral>	HPGL_FSPACE3	<loose>	HPGL_FANGLE3	<450>
HPGL_FTYPE4	<paral>	HPGL_FSPACE4	<loose>	HPGL_FANGLE4	<900>
HPGL_FTYPE5	<paral>	HPGL_FSPACE5	<loose>	HPGL_FANGLE5	<1350>
HPGL_FTYPE6	<cross>	HPGL_FSPACE6	<loose>	HPGL_FANGLE6	<00>
HPGL_FTYPE7	<cross>	HPGL_FSPACE7	<loose>	HPGL_FANGLE7	<450>
HPGL_FTYPE8	''	HPGL_FSPACE8	''	HPGL_FANGLE8	''
HPGL_FTYPE9	<paral>	HPGL_FSPACE9	<tight>	HPGL_FANGLE9	<00>
HPGL_FTYPE10	<paral>	HPGL_FSPACE10	<tight>	HPGL_FANGLE10	<450>
HPGL_FTYPE11	<paral>	HPGL_FSPACE11	<tight>	HPGL_FANGLE11	<900>
HPGL_FTYPE12	<cross>	HPGL_FSPACE12	<tight>	HPGL_FANGLE12	<00>
HPGL_FTYPE13	<cross>	HPGL_FSPACE13	<tight>	HPGL_FANGLE13	<900>

The angle brackets in the above table are references to the standard HP-GL string in the table below. Thus, it is possible to either override specific fill type entries in the above table or, for example, swap all the 450 and 1350 fill types by overriding the sequences of the labels given below:

Gcap Label	Default Sequence	Description	
HPGL_FSOLID	'1'	Select HP-GL Solid Fill Type	<solid>
HPGL_FPARALLEL	'3'	Select HP-GL Parallel Lines Fill Type	<paral>
HPGL_FCROSSHATCH	'4'	Select HP-GL Cross Hatch Fill Type	<cross>
HPGL_FLOOSE	'100'	Select a fill line gap of 100 units	<loose>
HPGL_FTIGHT	'50'	Select a fill line gap 0f 50 units	<tight>
HPGL_F0DEG	'0'	Set angle of fill lines to 00	<00>
HPGL_F45DEG	'45'	Set angle of fill lines to 450	<450>
HPGL_F90DEG	'90'	Set angle of fill lines to 900	<900>
HPGL_F135DEG	'135'	Set angle of fill lines to 1350	<1350>

APP Printer Driver Conventions and Documentation

This section describes the conventions used in developing printer drivers for the APP and all subsequent printer driver configuration produced by Uniplex.

As described earlier, APP Pcap names have adopted the following naming convention:

[*L*/]*modelsymset*[*/card/*]

The naming of Fcap font name sections has been standardized to help cut down duplication of entries and keep the section naming line short. This makes the name more relevant to the printer, and more intelligible to anyone configuring a printer. An Fcap font name takes the form:

MAN-SYMSET-tfaceEFFECT[*-psize*][*/CART/*]

Where:

MAN   Indicates the manufacturer, for example HP for Hewlett Packard.

SYMSET   Specifies the symbol set being used. For example, ROMAN8.

tface   Is the Typeface name. For example, Times.

EFFECT   Is the character effect of the font, one of NORMAL, BOLD, ITALIC, or BOLDITALIC.

psize   Is the point size that the character widths have been entered at, if the font is not truly scalable by Uniplex. This should be the same value as defined by the POINTS token.

/CART/   Denotes which font cartridge or card the font is on.

For example, the font name for the 11 point CG Times typeface of the HP Great Start font cartridge would be:

HP-ECMA-CGTimesNORMAL-11/GSTART/.

Note: Names are case insensitive.

The Fcap and Gcap files in the APP contain *supersections*, which contain a group of sections that belong together. For instance, all the font definitions for a particular font cartridge are grouped together in one supersection. The supersections start with a **name* line and terminate with a **))* line. The format of the name is dependent upon the type of supersection.

The Gcap file supersection names are of the form:

MAN.gcap

Where MAN is an abbreviation of the printer manufacturer's name, For example, HP for Hewlett Packard, CN for Canon.

The Fcap file consists of three different supersections:

-        a fixed pitch font supersection

-        a printer paper definition supersection

-        a proportional font definition supersection which can be either the internal fonts of a printer model or the fonts belonging to a particular font card or cartridge

The fixed pitch font supersection is named *fonts.fixd* and contains all the fixed pitch font variants configured to date. The font name for these fonts takes the form:

*pitch***CPI***points***PT**

Where:

pitch   is the characters per inch value of the font. For example, 16.66.

CPI   is the string **CPI**.

points   is the point size of the font, for example 8.5. This matches that defined in the $POINTS$ token of the font definition.

PT   is the string **PT**.

The paper supersection name takes the form:

ptr.**paper**

Where:

ptr   Is the Pcap printer name.

.**paper**   is the string .**paper**.

The proportional font supersection name takes the form:

srce.*symset*.**prop**

Where:

srce   Is either the font cartridge/card name or the model/series name of the printer. For example, S2 for the HP S2 cartridge or hpIII for any of the HP series III printers.

symset   Is the symbol set name. For example, ecma.

.**prop**   is the string .**prop**.

For example, the S2 cartridge supersection would be named S2.ecma.prop, the HPIII internal font supersection would be named hpIII.ecma.prop as all the series III HP printers have the same internal proportional typefaces.

Where a supersection is relevant to a number of different printer models each model is also included in a comment section within the supersection. Each entry takes the same format as the supersection name.

For example, the hpIII.ecma.prop supersection contains the following comment:

*   hpIIID.ecma.prop, hpIIIP.ecma.prop, hpIII.ecma.prop

WARNING: Do **NOT** remove any of the supersection names or comments as these are used for checking whether entries exist or not before adding them to your existing files.

Supported Printer Specifications (SPS)

For each printer supported in the APP a **S**upported **P**rinter **S**pecification (**SPS**) is included in the on-line documentation section of the APP. The SPS is comprised of seven sections which detail the printer facilities supported by Uniplex and any non-standard ways of using the printer within the Uniplex applications. Below is a description of each section within an SPS.

1   Printer Details

This table appears at the start of each Supported Printer Specification and details the following:

-        printer manufacturer
-        printer model
-        font options installed
-        page orientation
-        Pcap name(s) of the driver

2   General Facilities

This table details the availability of:

-        input paper bin selection
-        duplex printing
-        high resolution graphics
-        local copy facility
-        the type of box graphics the fill method used
-        any other general facilities available on the printer (for example, change bars)

When Yes or YES are allowed, the fully uppercase YES indicates that the option has been tested by Uniplex. The Yes indicates that the option does not normally exist for that printer but that the controls have been included in the driver as some part of them may be relevant.

For example, bin selections normally include a manual feed command. Not-Tested indicates that it was not possible to test the facility at all.

A brief explanation of the general facilities table follows.

Bin Selection can either be set up in a print style which is restricted to specifying that the first N pages of any print should come from an alternate bin, with the rest from the main bin or by use of Uniplex printer commands, for example, UPPER and LOWER. The configuration for this latter method may need modification for Uniplex version 7.00 and earlier. The setting Not-Tested indicates that Uniplex has been configured to use bin selection, but that this could not be tested since a multi-bin printer was not available.
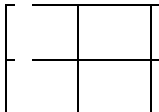
Duplex Control indicates whether Uniplex can switch the printer into duplex mode, printing both sides of each sheet of paper, in response to Uniplex printer commands such as DUPON and DUPOFF.

High Resolution printing (only available if Uniplex AGS is installed) shows whether printing of high quality graphics, for example, pie charts is possible. On some printers this is not possible in Landscape mode.

Local copy (if available) allows Uniplex to ask the printer to replicate pages, rather than having to send multiple copies of each page.

Box Graphics defines the ability to print line-drawing and boxing. If No, then lines and boxes can only be drawn using a combination of **+**, **-** and ¦ characters. If Fixed-pitch-only, then the boxing may not all join up properly when printing in proportional fonts.

For example:

No: *Yes:*   *Fixed-pitch-only*
                                                   with prop. font:

```
+--+--+
¦  ¦  ¦
+--+--+
¦  ¦  ¦
+--+--+
```

Fill method defines the method Uniplex uses to print *fill patterns*; these are used to print graphics files when AGS is not installed, and for printing patterns, such as bar charts, generated in the Word Processor. Depending on the printer's capabilities, Uniplex uses characters, some form of fill pattern, or a mixture of the two.

For example:

Characters: *Patterns:*

####////&&&&
  @@@@%%%%****

£ - Pound Sterling specifies whether Uniplex can print the **£** symbol (entered into documents as **ESC % #**). On some printers it can only be printed in a certain font, indicated as Fixed-Font which in a complex document might look out-of-place amongst text of a different font. For example, a pound sign in the middle of some small font text might appear as:

Regardless of currently selected
font, can only print £ in normal font.

Change Bars specifies whether the set of Printer Commands (**.SNCN**, **.SNC2**, **.SNCS** and **.SNCE**) used to create change bars on the right hand side of a page are available. They place the following symbols on the right hand side of a page:

Note: When A5 printing on a larger page, the bar may be well beyond the right hand text margin.

.**SNCN**   (one bar on next line; **C**hange **N**ext)

.**SNC2**   (one bar on next 2 lines; **C**hange next **2**)

.**SNCS**   (one **C**hange **S**tart mark on next line)

.**SNCE**   (one **C**hange **E**nd mark on next line)

3   Paper Sizes

Where relevant, separate tables for portrait and landscape paper sizes are included which detail the various paper sizes tested for the particular printer driver.

Uniplex Name is the name that appears in the Fcap file for a particular paper size.

Manufacturers Name is mainly for laser printers, where the specific paper tray usually bears the commonly used name for that paper.

Lines per Page column shows how many lines Uniplex can print using default font and line spacing, provided the document's page length (**.PL** command) is at least as long as the number in the column.

Many printers cannot print all the way to the edge of a page. Where this is the case, the Unprintable inches column indicates the areas that Uniplex can not print to.

Unless noted otherwise, prints using the following Uniplex paper sizes also work on each listed paper size (printing an appropriately small page image on the paper):

A5, **5x8**, **Wallet**

4   Fonts Supported

This consists of three tables each detailing which fonts are defined and how they interact with effects and pre-defined fonts.

Uniplex provides three subtly different ways for users to select a font:

a)   **Pre-defined** - using the Set Font command to specify the name of a Uniplex pre-defined font. For example **.FN LARGE**.

The font named NORMAL from this set is also the default font used when printing in Quality style.

b)   **Dialog Box** - select the Select Font dialog box from the ring menu:

Layout
Set_Font
Font_start

This allows specification of the typeface, effect and point size. The use of this font selection mechanism is of most use with a printer that has scalable fonts. However, where the exact selection is not available, Uniplex always uses the closest available font, often selected from the pre-defined set.

c)   **Effect selected** - using Uniplex print effects to select from a subset of the pre-defined fonts.

This mechanism allows the selection of a font for a single character or word, as opposed to the other two methods which define the font for all following lines.

Since these effects do cause an actual font change, users need to be aware that effecting text beneath a Set Font command, overwrites the Set Font command. The text is printed in the selected effect.

5   Uniplex Extended Character Set

This table identifies whether the full set of accented and other special characters (character values 161-255; X/Open - ISO 8859/1) are available to print on the printer. Some printers can only print accented and special characters in a certain font.

6   Print Time Directives

This table details the print time directives that have been defined for this entry, such as DUPON, which are used with the **.SN** dot command.

7   Printer Hardware Specification and Set-up

This table details the configuration of the printer used to test the entry.

All the SPS files can be found in the *UAP/documents/APP/sps* directory, if the Install Documentation option is selected when installing the APP. The file *INDEX* in the SPS directory lists the printer driver name against the filename used to store the driver information.

Chapter 3

Configuring Uniplex Windows

THIS PAGE INTENTIONALLY LEFT BLANK

Overview of X

Uniplex Windows allows you to access standard Uniplex applications which have been modified to support windowing concepts.

X is a network-based graphical windowing environment. It is built on the server-client model:

o   The *server* is the screen driver and provides the user's access to X applications. For example, X terminals are displays which run a dedicated X server.

o   The *client* is any application currently associated with the server. There may be more than one client. That is, the user's application (for example, **uterm**, the front-end program which provides an interface between X and the Uniplex character-based applications) is a client of the server concerned, as is the X application being accessed.

In addition:

o   The *host* is the machine running the X application being accessed. It makes requests to the server for windows and fonts via the X protocol.

The server and client need not run on the same machine, provided that they are networked together.

The server and client communicate through a well-defined protocol. This can be either across a network (for example, TCP/IP running over Ethernet) or through an internal channel (for example, UNIX streams) when both are running on the same machine.

The following illustration shows a greatly simplified example of how these components fit together.

HOST

```
X Clients:
mwm (window manager)
uterm (Uniplex text
front-end program)
gd_x11 (Uniplex RGIP
front-end program)
Desktop (eg. VUE, CDE,
X.Desktop)

Non-X Applications:
Uniplex
Ucalc
```

X Protocol

SERVER

```
Screen driver

Local X clients
```
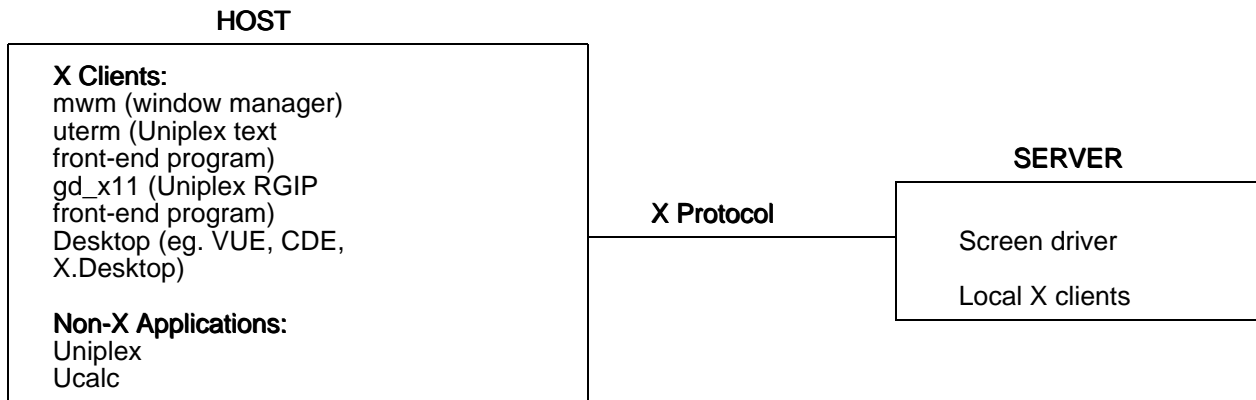
Defining the Screen Driver Server

Applications (clients) are directed to communicate with a particular screen driver (server) via the $DISPLAY$ environment variable. $DISPLAY$ contains the server's Internet address as one of its components. It has the format:

host:server.display

where:

$host$ is the name of the machine running the server.

$server$ is the server number (which is always zero on a single-user machine).

$display$ is the screen number of the machine to be used (some servers may address the screen as black and white or as color). For further information, refer to the documentation supplied with your server.

For example, to specify the server on the single-user machine **pluto**, on the default screen, set $DISPLAY$ as follows:

DISPLAY=pluto:0.0; export DISPLAY

or, more commonly:

DISPLAY=pluto:0; export DISPLAY

If the server (the screen driver) and the client (the user's application) are running on the same machine, the *host* parameter of *DISPLAY* may be set to **unix:0**, which indicates that communication between the two may be via an internal mechanism.

Fonts

Note: This section applies to X11 Release 4; for details of Release 5 or later, refer to your X documentation.

Fonts used by X applications are files of character bitmaps (or scalable outlines) usually held in the directory */usr/lib/X11/fonts* on machines running X servers, although this is configurable. X terminals may load their fonts from a host machine running a *font server* which sends font files held on this host to the terminal. Often X terminals have one or two default fonts *hard-wired* into them.

Fonts distributed for X Windows systems are generally held in a portable format known as BDF; these files are usually suffixed *.bdf*. Before a server can use them, they must be converted to a server-dependent form, which is often known as SNF. These files will have names suffixed with *.snf*. Server vendors should provide utilities for converting BDF files to files readable by their servers.

There is a file in each font directory (for example, the font directory */usr/lib/X11/fonts/misc*) called *fonts.dir*. It contains the names of all font files in the directory, together with the names which an application or resource should use in referring to them.

Entries in the *fonts.dir* file have the syntax:

font_filename font_name

Sometimes another file, *fonts.alias*, is also present (or you can create it). This allows other names to be associated with a font.

Entries in the *fonts.alias* file have the syntax:

font_name alternative_name

On most systems, there are two standard X utilities in */usr/bin/X11*: **xlsfonts** and **xfd** *font_name*. These are used to retrieve information about a machine's fonts. **xlsfonts** lists all of the available fonts on a system by their font names (as opposed to filenames). **xfd** *font_name* opens an X window containing the characters of the named font.

Window Managers

Window managers are clients of X; they are programs with special privileges that allow control of windows on the server's screen. They govern much of the look and feel of the X environment.

Window managers detect requests from other client applications to map top-level windows to the screen. Most window managers intercept such requests and stop the window from appearing on the screen until it has been formatted correctly. This allows resizing of windows, shuffling of window stacking order, iconizing of windows, and program termination through menus and window controls. It is the window manager which places title bars around application windows and selects active windows. It also controls the position of each window on the display.

Desktops

A desktop is an application which provides a visual way of viewing directories, files and applications. In earlier releases of Uniplex Windows, Uniplex supplied rules for and, on some systems, a copy of IXI's X.desktop product.

uterm and gd_x11

uterm and **gd_x11** are the Motif programs which act as front-ends to Uniplex character and graphics applications, respectively. They both support the use of push buttons and scroll bars and allow use of the mouse to interact with text and graphics.

Communication between Uniplex applications and their front-ends is via a well-defined communications protocol which is extensive enough to allow the reporting of such things as resize events, mouse clicks and drag areas. In addition, the Uniplex binary may instruct its front-end to adjust the scroll bar, change its push button text or emulate different terminal window-sizes.

Note: Standard Uniplex applications do not need to understand any X concepts to support the front-end protocol.

Like many X applications, **uterm** and **gd_x11** are highly configurable using X resources.

For further details, see the later section Uniplex Windows Resources.

Bitmap Files

All icons and graphical symbols used by Uniplex Windows are found in the directory *UAP/XW/bitmaps/uniplex*. These include the Uniplex logo used for the Uniplex Windows copyright window and the pictures used by the window manager when a Uniplex application window is iconized.

Startup of Uniplex Windows Applications

The later section Starting X and Applications describes the commands used to start Uniplex Windows. The following sections describe the main programs that will be run as a result of using the **uxwindows** script.

uxuniplex Front-end Script

Each invocation of an application or a utility starts with the execution of the **uxuniplex** front-end script. This is a link to the standard **uniplex** front-end script, which then tests the name with which it was invoked to determine whether it should run in X mode.

uxuniplex sets up a standard Uniplex environment (for example, setting the $Uredirect$ and $PATH$ variables). It then runs the command **exec uxlaunch -exec $*.**

uxlaunch and uxspawn

uxlaunch is a program which launches a new Uniplex application in its own window. It does this by:

1   Checking for the existence of the user's process spawner, **uxspawn**.

2   Creating **uxspawn** if it is not already running.

3   Placing its arguments (the requested user program) on **uxspawn**'s process queue.

4   Informing **uxspawn** that a process request has been made.

On initialization, **uxspawn** sets up certain user-configurable environment variables (for example, TERM, thus enabling the user to make use of **uterm**, **utermcolor** or **utermgrey** terminal definitions). It then loads all relevant X resources (for example, configuration strings which affect the appearance and behavior of X applications). Finally, it displays the Uniplex copyright message in a Motif dialog box.

uxspawn also sets itself up as a UNIX process group leader. This means that, if it detects a server failure or the user makes a request to shut down Uniplex Windows, it can immediately kill all of its child processes (see below).

uxspawn then sleeps until one of the following occurs:

o   A timeout. Timeouts are set up to tell **uxspawn** to look for any Uniplex Mail or Time Manager alarm messages.

         If there are any messages, **uxspawn** displays them on the screen in a dialog box.

         Even if there are no messages, **uxspawn** exercises its link to the server to check that it is still running (for example, in case the user has switched off the terminal without closing down Uniplex Windows).

         If **uxspawn** discovers that the server is not running, it will shut down cleanly, removing all associated Uniplex processes.

o   A pending process request from **uxlaunch**. As described above, **uxlaunch** is run every time a user requests a new Uniplex application window.

         Once **uxlaunch** has verified that **uxspawn** is already running, it places the command line concerned in a file in **uxspawn**'s process queue (which is a directory known to both **uxlaunch** and **uxspawn**). It then informs **uxspawn** that a process should be invoked; **uxlaunch** does this by sending a character to a named pipe in **uxspawn**'s process queue. **uxspawn** constantly waits for this, using an Xt Intrinsic mechanism.

         When **uxspawn** receives a pending process request in this way, it reads each pending process file, forks a child process and executes the requested command. The child process inherits **uxspawn**'s environment and also makes use of the X resources which **uxspawn** loaded on initialization.

         Note: The command executed by uxspawn can, in theory, be anything. In practice, it is usually uxstartup or uxinvoke (see below).

uxspawn may be asked to shut down via the special request **@KILL**. When it receives this command, for example uxuniplex @KILL, **uxspawn** runs its close-down procedure rather than attempting to execute a new process.

All programs executed by the process spawner **uxspawn** are members of the same UNIX process group and are all terminated when it finishes. This occurs either when a user terminates Uniplex Windows or when **uxspawn** detects that its X server has shut down.

uxinvoke

uxinvoke is a Uniplex application which connects a standard Uniplex application to the relevant X front-end process. It is called by **uxspawn** to connect the application required (for example, **uniplex**, **ucalc** or **ped**) to either **uterm** or **gd_x11**, as appropriate.

Communications between the Uniplex process and the front-end are via pseudo-terminal devices. **uxinvoke** selects a free pseudo-terminal device and makes any necessary **ioctl** calls for it before executing the Uniplex application and the front-end as child processes.

uxinvoke uses the file *UAP/XW/xrule* to determine which front-end the application concerned is to be connected to and which front-end options (if any) are to be used. *xrule* contains a line in the following format for each application:

application_name.button_pressed   frontend_command

uxinvoke looks for the line starting with the *application_name* passed to it as an argument by **uxspawn** (for example, **-name Word_Processor**) or, if no **-name** argument was passed, it looks for the line starting with the Uniplex binary name.

button_pressed indicates which mouse button the user clicked or dragged and may be passed to **uxinvoke** via its **-b** argument. This feature allows the front-end to be invoked differently depending upon a button action. It is not often used.

frontend_command is the full UNIX command required to execute the front-end process. For example, **uterm -vscrollbar** to run **uterm** with a vertical scroll bar.

uxinvoke executes the required application and front-end process as described above, as child processes. It then waits until it detects the death of either child, at which point it ensures that the other child process exits as well.

uniplex.start

This is called by the **uniplex**, **uxuniplex**, and **uxwindows** front-end scripts to perform their main processing.

It is invoked with the **-uxrestart** argument from the restart button on the Exit Uniplex Windows dialog to relaunch a main menu window.

It is also called from the main menu file with the **-application** argument to allow it to start applications in separate windows when running under Uniplex Windows.

In this mode, if Uniplex Windows is running, it simply issues the necessary **uxuniplex** command. If Uniplex Windows is not running, this calls the application directly.

A Sequence of Events

The following sequence of steps, summarizing what happens when a user starts Uniplex Windows and runs another application, illustrate how the main programs are invoked:

1   The user starts Uniplex Windows by typing, typically in an *xterm* window:

         uxwindows

2   This front-end script does some processing and then displays the word UNIPLEX in an asterisked box in the *xterm* window.

         It then ensures that **uxspawn** is running by issuing the command:

         uxlaunch -exec echo

3   Once **uxlaunch** starts, it displays the Uniplex Copyright message in a dialog window.

4   The **uxwindows** script ends by issuing the following command to start a window running the main Uniplex menu:

         uxlaunch -exec uxinvoke -name Menus -b 1 uniplex -q

5   **uxlaunch** puts this command in **uxspawn**'s queue, and signals **uxspawn**. In due course, **uxspawn** issues the command:

         uxinvoke -name Menus -b 1 uniplex -q

6   This then results in a **uterm** window running the menus system.

7   The calls from the main menu file to main applications create a new application window.

Note: Calls from other places, such as the pop-up desk, always run applications within the same **uterm** window.

For example, if the user selects Mail from the main menu, this runs the command:

uniplex.start -application E-Mail umail

8   **uniplex.start** then issues the command:

uxuniplex uxinvoke -name E-mail umail

9   **uxuniplex** then issues:

uxlaunch -exec uxinvoke -name E-mail umail

10   Which, once processed by **uxspawn**, results in a new **uterm** window created by the command:

uxinvoke -name E-mail umail

Configuration Files

Uniplex Windows differs from other Uniplex products in the way in which it may be configured since it uses X resource files in addition to the usual Uniplex style of configuration.

This section provides brief details of the location and content of the files used to hold configuration information.

Wherever possible, relevant X resource files are maintained in Uniplex directories and are searched for in standard local, central and network layers. For further details, see the list of files below.

Configuration files for Uniplex Windows are maintained in a subdirectory of *UAP* named *XW*. Subdirectories within *XW* are:

Directory	Purpose
config	This directory is referenced by the program **uxinvoke** when starting up a Uniplex Windows application.
	The file *xrule* holds invocation lines for the front-end process to connect an application to, usually **uterm** or **gd_x11** with appropriate arguments.
servers	Holds generally non-translatable X resource files for tailoring the appearance of Uniplex Windows applications for different terminals. Uniplex Windows loads the file *default* unless the user does one of the following:

- Sets the resource *Uxwindows.server* in the user's *.Xdefaults* file to a particular server name.

- Sets the environment variable *UXENVIRONMENT* to the name of the required server file.

- Passes its name as the first argument to the *uxwindows* script.

Resources for individual servers are held in files called *server_name* in this directory. For example, *servers/ncd*.

For more details, see the section Uniplex Windows Resources.

Directory	Purpose
resources.comm	Holds common resources for all servers, for example, icons.
	For more details, see the section Uniplex Windows Resources.
resources.host	Holds resources applicable to the host machine.

Environment Variables

Uniplex Windows makes use of a number of environment variables in addition to those used by the standard Uniplex product. Where necessary, these are propagated by the Uniplex process spawner, **uxspawn**, when it is invoked, so that Uniplex applications executed by **uxspawn** inherit these variables.

These environment variables are described in the chapter The Uniplex Environment although most users and system administrators should not need to set to alter them.

Uniplex Windows Resources

X resources are configurable strings for various properties of an X application (within Uniplex Windows this includes such applications as, for example, **uterm**, **gd_x11** or **uxspawn**). Resources define such attributes as the color of a window's background or the number of softkey buttons to be displayed.

Location, Format and Loading of Resources

Location of Resource Definitions

Applications that use resources make an order-dependent search of several sources. Each time a match is found, the value loaded overrides any current value. The sources used in the search are listed below in the order they are accessed:

o   Resources specified by **-xrm** command line argument

o   Defaults loaded into the server by **xrdb** (see the X11R4 or X11R5 user guide)

o   *$XENVIRONMENT* (contains a resource filename)

o   **/usr/lib/X11/[$LANG/]app-defaults/application-class** (for example, **Uterm**)

o   *$HOME*/**.Xdefaults**

The server defaults are overridden by the application-class defaults, which are overridden by the user's home directory defaults, and so on.

The *XENVIRONMENT* variable is used by the **uxspawn** program to point to a temporary file containing all resources required by Uniplex applications.

Uniplex Windows does not make use of **-xrm**, **app-defaults** or **xrdb** to set resources in its default configuration, but the user or System Administrator may wish to do so.

Format of Resource Definitions

In brief, a resource definition takes the following format:

object.subobject [**.** *subobject ...*] **.** *attribute* **:** *value*

where:

object is a client name or class;
subobject is a widget instance or class.

Note: An asterisk (*) may be used instead of period (.) to represent zero or more subobjects.

For example:

xterm*font: 8x13

or:

xclock*background: red

For full details of the format of resource definitions and an explanation of the components, refer to the appropriate X user guide.

How Uniplex Windows Loads X Resources

When the **uxspawn** process is initialized, it loads all required X resources from three files in addition to the usual set of X resource files and saves the resulting *resource database* to the temporary file **xenvironment** in the user's **$Utemp/uxwindows** subdirectory.

The environment variable *XENVIRONMENT* is then set to the path of this file, so that all child processes inherit these resources.

The additional files loaded by **uxspawn** are:

UAP/XW/resources.comm	A set of common resources for all servers, namely translatable icon and window titles, and the icons to use to represent Uniplex applications.
UAP/XW/resources.host	A set of resources applicable to the host machine.
UAP/XW/servers/<server>	A file of resources for the named server, such as **decstation** or **ncd**. If no server has been specified by the user, the file **default** is used.

How uxspawn Determines the Server File to Use

uxspawn uses the server file specified in any of the following ways, which are ordered from highest to lowest precedence:

o   The user may specify the name of the server file he wishes to use by passing its name as the first
         argument to the **uxwindows** script. For example:

         uxwindows ncd.adobe

o   The Uniplex environment variable *UXENVIRONMENT* may be set to the name (not full path) of the
         server file to use. For example:

         UXENVIRONMENT=ncd.adobe
         export UXENVIRONMENT
         uxwindows

o   The resource **Uxwindows.server** or **uxspawn.server** may be set to the name (not full path) of the file
         to use. This resource should be placed in the user's *.Xdefaults* file. For example:

         Uxwindows.server : ncd.adobe

o   The required server file can be placed in the user's local *UAP/XW/servers* directory, and named
         default, to be loaded without specifically identifying a server filename. For example:

         cp $Uredirect/UAP/XW/servers/ncd.adobe $HOME/UAP/XW/servers/default

Once **uxspawn** has determined the server filename to use, it searches the *UAP/XW/servers* directories in the user's local, central and network layers, in that order, for the file to use.

If the named file cannot be found in any of these directories, then the default server file is used in whichever Uniplex configuration layer it may be found; that is, the user's local default server file if it exists.

Since **uxspawn** saves its loaded resources to a temporary file, users must exit Uniplex Windows and reinvoke it before any changes made to server files take effect.

The following sections outline the resources that may be used with Uniplex applications. Any of them may be placed in the user's *.Xdefaults* file if they are relevant to all the servers he uses, or placed only in relevant server files.

Loading Different Resources for an Application

As mentioned previously, resource definitions usually begin with the name of the relevant application. Some applications however, including **uterm**, support the use of a **-name** command line argument to override the application's name. This allows the application to load different resource values depending on the string passed with **-name**.

The technique is used by Uniplex Windows to set **uterm**'s geometry and its title and closed icon depending upon which Uniplex application it is currently front-ending. For example, **uterm -name Word_Processor** would find the resource entry **Word_Processor*geometry**, whereas **uterm -name Spreadsheet** would match with the resource **Spreadsheet*geometry**.

This enables **uterm** to appear in different places on the screen, depending upon its associated Uniplex application. The program **uxinvoke** passes its own **-name** argument to **uterm** or **gd_x11** to make use of this feature.

Resource Details

The following subsections list individual resources by group (for example, all those which apply to **uterm** are grouped under the same subheading).

Each resource is given a code indicating the type and interpretation of the resource:

B   (boolean) an option which is either true or false.

I   (uppercase 'i') an integer; the units are described in the corresponding notes.

S   a string, used directly with some specific meaning.

P   a string used to hold a pathname for a file. However, this X resource will only be used to find the file if the file is NOT found in the **XW/bitmaps/uniplex** sub-directory within either the local or central UAP directory.

Uxwindows

Several programs used by Uniplex Windows are defined to be of class **Uxwindows** (that is, they make use of resources which begin with the name Uxwindows). These are **uxlaunch**, **uxspawn** and **uxmsg**. The following resources are used by one or all of them. Each should be prefixed by the string **Uxwindows*** or **Uxwindows**.

Resource	Details
copyright (P)	This is the name of a pixmap to be placed placed in the copyright dialog box when the user first invokes a Uniplex application from the desktop. The default is **$Unode/XW/bitmaps/uniplex/copyright.px**.
copyrightTime (I)	This is the time in milliseconds for which the Uniplex copyright message is to be displayed. The default is **7000** (7 seconds).
font (S)	This indicates the name of the font to be loaded into Motif information boxes for warnings and mail messages.
msgIcon (P)	This is the name of a pixmap to be placed in alarm message boxes. The default is **$Unode/XW/bitmaps/uniplex/menus.px**.
QueryShutdown (B)	If set to FALSE, uxspawn terminates the Uniplex Windows session when the last application exits, rather than prompting the user with the "Exit Uniplex Windows?" dialog. The default is **TRUE**.
server (S)	This is set to a string indicating the name of the server file that **uxspawn** should load. Server files may be placed in Uniplex local or central layers and **uxspawn** uses the appropriate file according to Uniplex search rules.
term (S)	This is used to change the **TERM** used from **uterm** to any other user-defined terminal name. For example, giving **term** the value **utermcolor** results in **uterm** displaying text effects in different colors on a color X terminal since **utermcolor** is predefined in standard Uniplex terminal configuration.
time (I)	This is the time in milliseconds for **uxspawn** to sleep between checks for pending **uclock** messages. The default is **15000** (15 seconds).
usePipe (B)	If set to TRUE, **uxspawn** actively reads the named pipe via which it communicates with **uxlaunch**, rather than relying on Xt Intrinsic mechanisms. The default is **FALSE**.
WaitStateIcon (P)	Name of the pixmap used by the Exit Uniplex dialog. The default is **$Unode/XW/bitmaps/uniplex/menus.px**.

uterm

uterm is the X window terminal widget in which character-based Uniplex products are run. It is highly configurable, with resources affecting color, fonts, underlining and character spacing. Resources unique to **uterm** are described below. These resources should be preceded by the string **Uterm*** in the resource file.

Resource	Details
bitmapA to **bitmapP** (S)	Pixmaps used for Uniplex grayscale characters. The default is a set of hard-coded patterns.
caretAppearance (S)	This may take the values **CaretBlock**, **CaretUnderline** or **CaretOutline**. Affects the appearance of uterm's text cursor. The default is **CaretBlock**.
Font (S)	Default font displayed by **uterm** in its main window. The default is **XtDefaultFont**.
fontA to **fontP** (S)	**uterm** allows sixteen different fonts, named **A** to **P**, to be displayed concurrently. These resources specify which X font to use for each **uterm** font. The values used are font names in **fonts.dir** or **fonts.alias** files. They do not necessarily relate to Uniplex effect letters. The default is the value of **uterm*main.font** or **Uterm*Font**.
fontAAscent to **fontPAscent** (I)	Used to adjust the baseline of a font for use in superscripting. The default is **0**.
fontAEffect to **fontPEffect** (S)	These resources may be set to the value **reverse** if the user wishes the relevant Uniplex effects to be displayed in reverse video if the associated X font for the effect is not defined or cannot be loaded. The font used is the default font. The resource **FontEffect** may be set to cause all Uniplex effects to display in reverse video if their preferred X fonts are not available. The default is **None**.

Resource	Details
fontAGLTranslations to **fontPGLTranslations** (S)	These are strings enclosed in double quotes, which contain the characters to use to represent the X/Open characters in the range **32** to **127** if they are not available in the current X font.
	The first character in the string corresponds to X/Open character 32, the last mapping to X/Open character 127. To use the same X/Open mapping for all Uniplex effect fonts, the resource **FontGLTranslations** may be used.
fontAGRTranslations to **fontPGRTranslations** (S)	These are strings enclosed in double quotes, which contain the 7-bit characters to use to represent the X/Open characters in the range **160** to **255** if they are not available in the current font.
	The first character in the string corresponds to X/Open character 160, the last mapping to X/Open character 255. To use the same X/Open mapping for all Uniplex effect fonts, the resource **FontGRTranslations** may be used.
fontASize to **fontPsize** (S)	May take any of the values **DontUse**, **UseBoxAndWidth**, **UseBox** and **UseWidth**. Those resources not set to **DontUse** are combined to compute the column spacing for text in **uterm**, such that characters always vertically align, no matter what font they are in. Those fonts set to **UseBoxAndWidth** take the maximum of the font's character bounding box and the widest character in the font. (The latter may be larger than the bounding box width for italic fonts, for example.)
	Fonts set to **UseBox** only take into account the font bounding box width, whilst those set to **UseWidth** only use the maximum character width. If all fonts are set to **DontUse**, spacing is set to fontA's bounding box width. The default is **UseBoxAndWidth**.
innerBorder (I)	Pixels of white space between edge of the **uterm** window and the area where text drawing takes place. The default is **2**.

Resource	Details
lazyRedraw (B)	If set to TRUE, **uterm** delays printing text when it receives it to see if more will follow. This may actually increase the speed of display since fewer redraws take place. The default is **FALSE**.
main.font (S)	Default for any of fontA to fontP not set. It defaults to either the value of **Uterm*Font** or **XtDefaultFont**.
maxClickTime (I)	If the event defined in the **clickTable** (see the later section Mouse Actions) for a particular mouse button does not end in '+', then **uterm** waits this many milliseconds after one click of this button, to determine whether a double-click will occur. The default is **750**.
maxMovement (I)	Allowable movement of the mouse in pixels during a mouse-click, before the click is interpreted as a drag. The default is **5**.
pixelsBelow (I)	The space in pixels between the descent line of one line of text and the ascent line of the one below (for example, in interline spacing). The default is **4**.
reportScrollDrag (B)	If this is set TRUE, then scroll-bar drags are reported to the application as appropriate. If set FALSE, they are not. The default is **FALSE**. This resource is set to FALSE when an application is interested only in the final position of the scroll bar.
scrollConvention (S)	If set to **DontUseThumbSize**, the scroll bar always returns values between 0.0 and 1.0 irrespective of the thumb size. If set to **UseThumbSize**, then the size of the thumb is taken into account and the value returned varies between 0.0 and 1.0-thumbsize. This resource affects the range of possible values seen in scroll bar movement messages. The default is **DontUseThumbSize**.

Resource	Details
underline1 (I)	Number of pixels above or below font baseline to draw underlines. Positive values are interpreted as pixels above the baseline whilst negative values are below the baseline. The default is **-1**.
underline2 (I)	Similar to underline1, but applies to the second line used when double underlining. The default is **-3**.

gd_x11

gd_X11 is the widget which acts as a 'front-end' filter for **ped** and **uchart**. It has many similarities with **uterm**, the text widget, such as its handling of softkey lines. Resources unique to **gd_X11** are described below. These resources should be preceded by the class name **GdX11***.

Resource	Details
flushFlag (B)	If set to TRUE, an Rgip flush command is executed after every draw request. This ensures that the screen is constantly updated. The default is FALSE.
interactMode (B)	When set to TRUE, this is equivalent to running **gd_x11** with the **-i** (interaction mode) flag. This allows **gd_x11** to return information to the Uniplex application. The default is FALSE.
maskOpen (B)	If set to TRUE, it is equivalent to the **-m** option (that is, it enables manual mode). Alters action of filter so as not to automatically generate Open, Initialize and Close sequences. The default is FALSE.
noOptRedraw (I)	If set to 1, no backing store is used by **gd_x11** for fast redraws. It is necessary to set this resource for servers whose pixmap copying does not work well. The default is **0**.

Resource	Details
noXtraPixel (B)	It should be set to TRUE for servers which do not draw rectangles wide or high enough. Symptoms are drawing errors in **ped** windows where white rectangles painted to remove objects underneath them leave pixels behind. The default is **FALSE**.
Rgip.noAspectRatioAdjust (B)	It affects whether **gd_x11** uses its entire drawing area or adjust its **Rigp** window to maintain aspect ratio (and therefore possibly not fill its entire area). For example, a 400x400 pixel **gd_x11** window appears wider than it is tall on a 100x80 dpi screen. By default, or if the resource is set to FALSE, **gd_x11** only uses a "visually square" region of its window area by reducing the larger size until it is the correct proportion of the smaller dimension. If set to TRUE, **gd_x11** uses its entire window area. The default is **FALSE**.
Rgip.stipple (I)	If set to an integer value, this resource overrides the server's recommended **best_stipple_size** and uses this value instead for tiling patterns and half-tones. This resource should be used where using the server's recommended size causes draw errors such as striping in filled areas. The default is **0**, implying that the server's value should be used.
wideLines (B)	If set to TRUE, then **Rgip** wide-line drawing is enabled. If set to 0, line width commands are ignored. Equivalent to running **gd_x11** with the **-w** flag when set to 1. The default is FALSE.

uterm and gd_x11

As mentioned in the previous section, several of the features of **gd_x11** are shared by **uterm**, and this is reflected by the number of resources which may be applied to both **gd_x11** and **uterm**. If any of these resources is not preceded by the widget name or class (for example, **GdX11** or **Uterm**), then the resource is applied to both. They may be set to different values for **GdX11** and **Uterm** if these names are used (for example, **Uterm*resizeShell**). If **uterm** or **gd_x11** is invoked with a **-name** argument this name may be used (for example, **Spreadsheet*geometry**).

Some of these resources are available as a direct result of using the Motif widget set. Any other resources available to Motif widgets used by **uterm** and **gd_x11** (for example, the PushButton widget) may, therefore, be used by them as well. One example is the text for softkey buttons, which may be set through the resource **XmPushButton*fontList**.

Resource	Details
buttons (I)	If set to zero, the number of softkey buttons appearing at the bottom of a **uterm** or **gd_x11** window is determined by how many may fit, using **buttonSize**, **buttonSpacing** and **buttonRows**. The number of buttons may change as the window is resized. If set to a non-zero value, then no more buttons than the number specified will appear, even when the window is stretched. Button sizes may change to fit them across the window. The default is **0**.
buttonRows (I)	Takes an integer number of rows of softkey buttons in **uterm** or **gd_X11** windows. The default is **1**.
buttonSize (I)	When applied to **uterm** this specifies the number of characters wide to make the softkeys. When used with **gd_x11**, this value is interpreted as pixels. The default for **uterm** is **8**; the default for **gd_x11** is **48**.
buttonSpacing (I)	Takes an integer number of pixels to space buttons by. The default is **10**.
buttonStart (I)	This is an integer which defines the softkey number to start from (that is, what the top left button is associated with). The default is **1** (that is, pressing the top left button generates F1).

Resource	Details
clickTable (S)	The **clickTable** resource specifies what mouse button specifies which mouse click event codes should be sent to Uniplex applications for each mouse action. This is described in detail in the later section Mouse Actions. There is no default.
cursor (S)	Determines the bitmap to use for the X mouse pointer when it moves into the **uterm** or **gd_x11** window. Values may be taken from any of the names in **/usr/include/X11/cursorfont.h**. The default is **left-ptr**.
dragTable (S)	The **dragTable** resource is similar to **clickTable** except that it specifies event codes to be sent to Uniplex to initiate mouse dragging. The resource has the same syntax as **clickTable**. There is no default.
forceClosedown (B)	A Boolean. If set to TRUE, **uterm** or **gd_x11** explicitly clears away pending events and free X structures such as widgets and display when exiting. This should ensure that the **uterm** or **gd_x11** window is removed from the server's screen. This resource is necessary for use with the Interactive server. The default is **FALSE**.
gap (I)	The border (in pixels) between all the widgets of the **xst** widget. The default is **4**.
geometry (S)	Standard X resource. Takes syntax WidthxHeight+X_offset+Y_offset. If x and y offset values are preceded by minus (-), the offsets are taken to be from the bottom right of the screen as opposed to the top-left corner. **uterm** interprets the width and height values as numbers of characters, whilst **gd_x11** interprets them as pixels. In both cases, offsets are in pixel co-ordinates.
hScrollbar (B)	If TRUE, **uterm** or **gd_x11** creates a horizontal scroll bar. The default is **FALSE**.
keyMapping (S)	This resource allows the definition of function keys and other keys such as arrows for use with Uniplex Windows. This is described in detail later. There is no default.

Resource	Details
noTitle (B)	If TRUE, the standard Uniplex application banners do not appear on the first line inside **uterm**. Since the title bar of the **uterm** window is set with an appropriate banner anyway, this line is not usually desired. The default is **FALSE**.
pollInterval (I)	This is the number of milliseconds for which **uterm** or gd_x11 waits between polls of their **stdin**; only used when **usePipe** is set to TRUE.
ptyDelay (I)	This controls the number of attempts a paste operation tries to paste text. Typically, this resource is set to values in the range 10 to 100. The default is **0**.
ttyInputSize (I)	If a tty driver's input buffer receives more data than allowed for by the size of the buffer, the excess data is lost. Therefore, to not overfill the buffer, this resource has been added to define the size of the input buffer. The default value is set to **128** characters. If the size of the input buffer is smaller than the default value, you must reconfigure the value to the correct size. For example, if it is known that the tty driver has a 64 character input buffer, the following line should be added to either the resources.host file or the user's .Xdefaults file: *ttyInputSize: 64 Equally, you may increase the value if the tty input buffer size is known to be higher than the default.
resizeShell (B)	If set to TRUE, then the **uterm** window grows to incorporate new vertical or horizontal scroll bars; that is, the text area remains constant. This situation arises when Uniplex Menus is invoked and the Word Processor or Spreadsheet is selected. If set to FALSE, the scroll bars take space from **uterm**'s work area; that is, the outer dimensions of the window remain constant. The default is **FALSE**.
usePipe (B)	If set to TRUE, then **uterm** or **gd_ x11** actively polls their **stdin** (as opposed to relying on the Xt Intrinsics). The default is **FALSE**.

Resource	Details
vScrollbar (B)	If TRUE, **uterm** or **gd_x11** creates a vertical scroll bar. The default is **FALSE**.
XmPushButton*fontList (S)	This is a mixture of **uterm** or **gd_x11** geometry manager widget and Motif PushButton widget resources. If used, it determines the font to use inside softkey buttons. For example: Uterm*XmPushButton*fontList: user6x13

Application Trigger Resources

Several Uxwindows resources are specific to a feature known as the *application trigger*. This is a small pop-up which appears when a user invokes a Uniplex application to notify him that the action has been detected. It has been added to reinforce the change in the mouse pointer which also occurs at this time, but which is not very noticeable.

The following resources may be preceded either by the string **uxapptrig*** or the string **Uxwindows***:

Resource	Details
grab (B)	When TRUE, the pop-up window *grab*s the mouse pointer, thus stopping any further applications from being invoked until it disappears. This stops any windows from being mapped or restacked on top of the alert and, therefore, obscuring it. If set to FALSE, the user is free to double-click on other icons, and so on, while the alert is still mapped. The default is **TRUE**.
placement (S)	Specifies where the pop-up should be placed on the screen. A value of **absolute** indicates that the resources **x** and **y** should be used to map the pop-up using standard X **geometry** X and Y values (that is, negative values may be used to measure distance from right or bottom of the screen). A value of **Pointer** maps the pop-up under the current mouse position, whilst **Centre** centers the pop-up on the screen. The default is **Pointer**.
trigger (I)	The time (in milliseconds) for which the pop-up should be left on the screen. The default is **2000**.

Resource	Details
triggerIcon (P)	This resource specifies the "busy" icon to display in a small window to indicate that an application is being invoked. The default is **$Unode/XW/bitmaps/wait_d.xbm**.
x (I)	The X co-ordinate to use for Absolute placement of the trigger pop-up. If it is negative, then it is offset from the right of the screen. The default is **0**.
y (I)	The Y co-ordinate to use for Absolute placement of the trigger pop-up. If it is negative, then it is offset from the bottom of the screen. The default default is **0**.

The resource **uxspawn.useTrigger** or **Uxwindows.useTrigger** may be set to FALSE or OFF to stop trigger pop-ups from appearing. This is best set in the user's *.Xdefaults* file.

Mouse Actions

Mice may be configured such that different actions are used for the same effect in a Uniplex application. Configurers should be aware of the difference between a **Uniplex** mouse button and a **physical** mouse button. Uniplex mouse buttons are in reality codes. When a Uniplex application receives a mouse event code of 1, it determines that mouse button 1 was used, an event code of 2 indicates that mouse button 2 was used, an event code of **4** specifies that button **3** was used, etc. The reason for the discrepancy between button 3 and its event code is that mouse buttons are each given one bit of a byte and, therefore, event codes increase by powers of two. Sending event codes this way means that it is possible to detect two mouse buttons used simultaneously.

By changing the **clickTable** resource, it is possible to make any physical mouse button return the event code for Uniplex mouse button 1, for example. Also, combinations of mouse clicks may be used to send one event.

Event definitions in a **clickTable** resource are semicolon separated and take the form:

step_set=event

where:

step_set   may include one or more physical mouse actions.

event   is a Uniplex event code.

Button clicks are defined by the string bn where n is a physical button number. Multiple clicks in one step_set are comma separated.

If an event is ended with the '+' symbol, it instructs **uterm** or **gd_x11** that double-clicks have no meaning for the given step_set, so that no delay should occur after one click, in anticipation of another. If the '+' symbol is not included, users may find that they cannot click buttons in quick succession.

For example:

o   **b1=1+;b2=2+;b3=4+**   ties physical mouse buttons to Uniplex mouse buttons, and instructs **uterm** or **gd_x11** not to expect double-click events (that is, to act on each single mouse click).

o   **b1=2+;b2=1+;b3=4+**   swaps the actions of buttons one and two.

o   **b1,b1=1;b2,b2=2;b3,b3=4**   causes **uterm** to only send events when buttons are double-clicked.

The **dragTable** resource specifies which Uniplex event codes to send when physical mouse buttons are used for dragging. It uses similar syntax to the **clickTable**.

Configure a Two-button Mouse

Uniplex Windows can be used with a mouse with two or more buttons. Most X display stations and terminals supply a three-button mouse. However, if you have a two-button mouse, you can customize a server description so that pressing both buttons together simulates Uniplex button three operations. For example, when drawing lines using the Word Processor.

Uniplex provides various customized server descriptions, one of which is called **odt**. This server file defines the operation of a color display using a three-button mouse running under the SCO Open Desktop 1.1. A second server file is provided, called **odtmb2**, which is a modified copy of the **odt** file for use with a two-button mouse. These files are held in the directory UAP/XW/servers. For SCO Open Desktop versions 2.0 and 3.0, these files are not required if Open Desktop has been configured correctly. See your Open Desktop documentation.

The changes made to the **odt** file to produce **odtmb2** are:

o   On the four lines that start:

        Uterm*clickTable
        GdX11*clickTable
        Uterm*dragTable
        GdX11*dragTable

        the **b3=** assignments have been changed to **b1b3** and the **b2=** assignments are changed to **b3=**. These lines now read:

        Uterm*clickTable   : b1=1+;b3=2+;b1b3=4+
        GdX11*clickTable   : b1=1+;b3=2+;b1b3=4+
        Uterm*dragTable   : b1=1;b3=2;b1b3=4
        GdX11*dragTable   : b1=1;b3=2;b1b3=4

Recommended Procedure

The following procedure is recommended when configuring Uniplex Windows for use with a two-button mouse:

1   Create the directory **$HOME/UAP/XW/servers**.

2   Change to the **$HOME/UAP/XW/servers** directory.

3   Copy the standard Uniplex server file that is most appropriate to your display. Use a name that indicates the file is for two-button mouse operation. You could enter, for example:

        cp /usr/UAP/XW/servers/generic genericmb2

4   Edit the copied file as described for the file **odtmb2** in the previous section.

5   Invoke Uniplex Windows using the filename as an argument. For example:

uxwindows genericmb2

6   If you are satisfied with the two-button mouse operations, move the copied file to the central Uniplex area. For example, enter:

mv genericmb2 /usr/UAP/XW/servers

Each time you invoke Uniplex Windows, use this new server name as the argument as shown in step 5.

Keyboard Mapping

The resource **keyMapping** is the mechanism used to tailor a server's keyboard to work with Uniplex. It allows function keys to be used as softkeys, and arrow keys to work sensibly.

To understand how the **keyMapping** resource is set, the configurer must be aware of the X keyboard portability concept known as the **keysym**. Each server returns a well-defined string, or **keysym**, for any key pressed. Function keys pressed typically return strings **F1** to **Fx**. Arrow keys return the strings "Up", "Down", "Left" and "Right". There are many other **keysym** strings, listed in the *X Window System Protocol*. X assumes every key on any keyboard has a unique **keysym**, and it is up to the server to decide which **keysym** a key should return.

The **keyMapping** resource tells **uterm** and **gd_x11** what strings, such as escape sequences or control characters, to send to a Uniplex application when a key with the relevant **keysym** name is pressed. Key definitions in the **keyMapping** resource are comma separated and take the form:

keysym=*send_string*

where *send_string* may be simple ASCII characters or include any of the following special strings:

String	Meaning
\\e	Comma
\\,	Carriage return
\\n	Tab
\\t	Backslash character
\\\\	Octal representation for ASCII character
nnn	Hexadecimal representation of ASCII character
\\x*nn* or \\X*nn*	

For example:

o   **F1=\\e1** makes **uterm** or **gd_x11** send ESCAPE 1 to the Uniplex application when F1 is pressed.

o   **Up=\\xb** makes **uterm** or **gd_x11** send hexadecimal b, or decimal 11, to the application.

o   An entire **keyMapping** resource might look like:

         Uterm*keyMapping : **Up=\\xb,Down=\\xa,Left=\\x8,Right=\\xc,**
         F1=\\e1,F2=\\e2,F3=\\e3,F4=\\e4,F5=\\e5,F6=\\e6,F7=\\e7,F8=\\e8

         Notice that each line in the resource string, except the last, is ended with a backslash to indicate that more of the resource string is to follow.

Note: On some verions of X, you may need four backslashes instead of two, for example, F1=\\\\e1.

See also the **setservers** command in the appendix Program Usage and Invocation in the Uniplex Technical Guide.

Starting X and Applications

Starting X

X must be running before you can start Uniplex Windows. If you are using an X terminal, X need not be specifically started since X terminals run dedicated X servers. If you are not using an X terminal, you must start X as described below.

Starting X and Uniplex Windows from an X Terminal

Users of X terminals need not start X since the server is an integral part of the terminal.

Starting Uniplex Windows from an X terminal involves logging into the client machine, setting the *DISPLAY* environment variable to point to the X terminal, and invoking **uxwindows**.

Starting X from a Standard Workstation

If you have a standard UNIX workstation capable of running X, you need to start X before invoking Uniplex Windows:

1   Set the *DISPLAY* environment variable to **unix:0** if Uniplex Windows is on the same machine as you are; otherwise, set it to *machine_name*:**0**.

2   Change the *PATH* environment variable to point to the server executables. *PATH* must include */usr/bin/X11*.

3   Start X. X is usually invoked using the standard program **xinit**.

         For example:

         xinit - - X -bs -su

         where:

         -bs   tells the X server not to use backing store.
         -su   indicates that save-unders should not be used.

These two mechanisms reduce the number of application redraws required when windows are exposed. For many servers they give faster response although they can be memory intensive. If you are experiencing memory problems, it is probably better not to use them.

xinit runs an **xterm** by default. This is a shell window incorporating vt100 and tek graphics emulation.

4   If Uniplex Windows is on another host, rlogin to the remote machine and set the $DISPLAY$ environment variable to $machine\_name$**:0** (where $machine\_name$ refers to the server workstation).

5   To start Uniplex Windows, enter:

uxwindows

This will try and run the Motif window manager (mwm). Window manager protocols mean that, if another manager is already running, this call to **mwm** will be ignored. However, if you want to stop **uxwindows** even trying to run **mwm**, invoke it with the flag **-manager**.

Logging out of the xterm shell window terminates X (even if other X applications are still running).

Configuring an X Terminal

If you have a new X terminal for which there is no suitable server definition file, you need to make some configuration changes before you can run Uniplex Windows.

Four files contain terminal-related configuration information:

UAP/XW/servers/server_name	The server definition file for your terminal.
UAP/terminfo and *UAP/termcap*	The files used to define the general terminal capabilities.
Tcap	The file used to define the Uniplex-specific terminal capabilities that are not definable in *terminfo* or *termcap*.

You must create a new server definition file each time a new type of X terminal device is added to the system for use with Uniplex Windows.

In addition, if the new terminal is grayscale or color, you may want to create a new **uterm** terminal configuration entry so that you can then take advantage of all the features of the X terminal device. You do this by creating a new entry in *Tcap* and modifying existing entries in *termcap* or *terminfo*.

The specific changes you need to make to these files are described separately in later subsections.

Configuring uterm Fonts in the Server Definition File

Uniplex uses fonts configured into **uterm** to highlight print effects on the terminal, so that the user has an idea of what the final printed copy will look like. Each font in **uterm** is associated with a resource letter (which is different from the effect letters in *Tcap*, described later), and each indicates a particular print effect. **uterm** simulates print effects by either changing the current font to another supported by the terminal or by manipulating the current font in some manner.

When you are setting up a new terminal, you need to select a subset of its available fonts and define how they are to be used within **uterm**.

1   Output a list of the fonts available on this terminal. Do this using either the standard **xlsfonts** command (if available) or the command appropriate to your system. For details of **xlsfonts**, refer to the X Window System Reference Manual.

2   Select an appropriate set of fonts (see below for hints on how to do this).

3   Associate these fonts with **uterm** resource letters in the relevant *UAP/XW/servers* file, as described
        below.

Choosing Fonts

o   The Default Font

        When choosing the set of fonts for **uterm** to use when indicating print effects, the most important
        choice is that of the default font size (the normal font, mapped to resource letter A in the appropriate
        servers file and usually to effect A in the *Tcap* file). This choice is important because the size of the
        default font determines the size of the **uterm** window and the readablity of the text within it.

        Generally speaking, the larger the pixel size and the lower the pixel density, then the smaller the size
        of the font which should be used. A suitable size should be chosen to ensure that the **uterm** window
        fits on the screen and that the text in it is easily legible.

        Ideally, you should choose normal, bold and italic fonts which are all the same type and size (so that
        when the print effect is applied, the font-style remains the same).

o   Proportionally-spaced Fonts

        A fixed-pitch font is recommended. However, if you do choose proportional fonts, you will probably
        need to associate an extra fixed-pitch font with one of the unused resource letters and tell **uterm** to
        use this font to determine the cell's size. Do this by setting the **FontSize** resource in the *servers* file
        as follows:

        Uterm*main.FontSize : DontUse

        Then, for the font you want **uterm** to use to set its cell size, set the **font[A-P]Size** resource as follows:

        Uterm*main.font[A-P]Size : UseBoxAndWidth

        Note: Full details of uterm resources, how to set them and what the effect is, are given in the earlier
        chapter Uniplex Windows Resources.

o   The Small Font

When selecting the small font, choose one which is large enough to read, but small enough to be used for superscripts and subscripts (the standard configuration simply shifts the small font up or down for superscripts and subscripts).

o   The Large Font

When selecting the large font, choose one which will fit inside the cell size calculated and still be readable.

Associating Fonts with Resources

The following table shows the mapping of print effects to **uterm** resource letters. Resource letters A to K should be allocated fonts suited to their associated print effect in the *servers* file; L to P may be allocated a font if required, but need not be.

Resource Letter	Print Effect
A	Normal
B	Bold
C	Small
D	Large
E	Italic
F	Superscript
G	Subscript
H	PS-SMALL
I	PS-NORMAL
J	FX-SMALL
K	FX-NORMAL
L	Undefined
M	Undefined
N	Undefined
O	Undefined
P	Undefined

The print effects are associated with effect letters (those used, for example, by the Word Processor) in the *Tcap* file.

The resource letters are associated with suitable fonts (using the guidelines given in the previous subsection) by editing the appropriate *UAP/XW/servers* file.

Note: The font definition data (that is, the data following the colon) should be in exactly the same form as the data output by the xlsfonts command used to display a list of available fonts.

THIS PAGE INTENTIONALLY LEFT BLANK