Contents

Chapter 1

Overview

1.1 Introduction

Recent years have seen an uprising in the use of networking, multitasking environments. Most of these environments make use of the Unix operating system, which allows more than one task to be executed simultaneously. As more powerful computers became available, this power was used to provide better user interfaces, supportive of the user, easier to learn and use. X Windows is a graphical user interface which uses windows, pull down or pop up menus and a pointing device. With X Windows multiple applications can run simultaneously in windows. X Windows provides the means for creating device independent applications, and facilities for generating multifont text and two dimensional graphics. FontCreator uses the X Windows environment, to enable users to create easily, their own fonts for use with their applications or for experimenting. Throughout this dissertation, we concentrate on showing how to use FontCreator effectively to create fonts.

1.1.1 Report Structure

The first chapter is an introduction to the FontCreator project. The second chapter is an overview of X Windows, with special mention on windows, window managers and toolkits. The third chapter explains the character properties and goes on to define the X Logical Font Description and the Bitmap Distribution Format which is used in X Windows. The chapter ends with an example of a font file. The next chapter establishes the need for the FontCreator and describes the way FontCreator was designed and implemented, the FontCreator toolkit and how FontCreator works. We have provided three appendices. The first is a user's manual for FontCreator and the second outlines the utilities provided with X Window to manipulate fonts. The last appendix lists the FontCreator code.

Chapter 2

X Window System.

2.1 The history of X.

X was born in 1984, at Massachusetts Institute of Technology (MIT). X was the result of MIT Project Athena, a project sponsored by Digital and IBM, with a goal of providing a graphical networking environment that was hardware independent. X design was based on the following principles [**?**, p. 5.]:

- Do not add new functionality unless an implementor can not complete a real application without it.
- It is as important to decide what a system is not as to decide what it is. Do not serve all the world's needs; rather make the system extensible so that additional needs can be met in an upwardly compatible fashion.
- The only thing worse than generalising from one example is generalising from no examples at all.
- If a problem is not completely understood, it is probably best to provide no solution at all.
- If you can get 90 per cent of the desired effect for 10 per cent of the work, use the simpler solution.
- Isolate complexity as much as possible.
- Provide mechanisms rather than policy. In particular, place user interface policy in the client's hands.

X Window System was a success. Design of version 11 started in May 1986. During autumn 1986, Digital decided to base its entire desktop workstation strategy for ULTRIX, VMS, and MSDOS on X. The last version 10 release was made available in December 1986.In January 1987 the first X technical conference was held in MIT. In September 1987 the first release of version 11 was made available. In September 1991 the latest release (release 5) of version 11 came out.

2.2 Overview of X.

X is a network transparent window system. X permits application programs to run on machines scattered through out the network and to be device independent. The latter feature allows applications to be portable, that is applications need not be rewritten, recompiled, or even relinked to work with new display hardware. X provides facilities for generating multifont text and two-dimensional graphics in a hierarchy of rectangular windows. Windows can overlap each other, can be moved, resized and restacked dynamically. Windows are inexpensive resources; applications using several hundred subwindows are common. For example, windows are often used to implement individual user interface components such as pushbuttons, dialogboxes, menus etc.

X applications, in terms of the network, are the clients that use the network services of the window system. A program running on the machine with the display hardware provides these services and so is called X server. X server controls and draws all output to the display monitors, reads input and forwards it to the appropriate clients for processing.

X consists of a network protocol, an interface library (Xlib), and client libraries (Xt Intrinsics). The X protocol and X library avoid specifying pushbuttons, dialogue boxes, menus or how an application should respond to user input, but provide a set of mechanisms that can implement a variety of user interface policies. Toolkits (providing menus, scrollbars and so on), higher level graphics libraries and user interface management systems (UIMS) can be all implemented on top of the X library. The most popular specifications for interfaces are Open Look and Motif.

2.3 Windows in X.

Creating windows is what a windowing system is all about. The X Window system supports one or more screens containing overlapping windows or subwindows. A screen is a physical monitor and hardware, which can be either color or black and white. There can be multiple screens for each display or workstation. All windows in an X server are arranged in strict hierarchies. At the top of each hierarchy is a root window which covers each of the display screens. Each root window is partially or completely covered by child windows. All windows, except for root windows , have parents. Child windows may in turn be, parents of other windows. This way an application can create an arbitrarily deep tree of window on each screen. X provides graphics and text operations for windows. A child window can be larger than its parent. That is, part or all of the child window can extend beyond the boundaries of the parent, but all output to a window is clipped by its parent. If several children of a window have overlapping locations, one of the children is considered to be on top of others. X does not guarantee to preserve the contents of windows. When part or all of a window is hidden and then brought back on to the screen, its contents may be lost. The server then sends the client program an Expose event to notify that part or all of the window needs to be repainted. The number of Expose events can be cut down by having the application to ask X server, to provide backing store on windows. This way the X server preserves any obscured areas. However not all X servers provide backing store, and even those that do, do not guarantee to provide backing store, for all windows all the time. Creating and displaying windows is a complex task. Details such as the display, the parent window, the location on the screen, the width and the height, the border width and color, and the background color have to be defined. Once the window is created, the application has to sent hints to the window manager about the window. The window manager may accept or deny any hinted request. Most window managers will try to deliver what the application wants, but there is no guarantee. If the application fights the window manager, the window manager wins. The X server, apart from sending expose events may send keyboard, pointer, window crossing, client communication events and so on. Programs should be able to handle (or ignore) events of all types.

2.4 Window managers.

As already mentioned X protocol provides mechanisms with which a variety of user interfaces can be build. Using these mechanisms a single client, called a window manager, can provide the user interface which is independent of all the other clients. A window manager can enforce a strict window layout policy as well as, automatically provide the following:

- Title bars, borders and other window decorations for each application.
- Uniform icons for applications.
- A uniform means of moving and resizing windows.
- A uniform interface for switching the keyboard between applications.

X itself comes from MIT with twm [**?**, p.3], the Tab (or Tom's) window manager.

2.5 Toolkits

When building X Window applications, one soon discovers that when creating multiple windows, large sections of code are being duplicated. Also soon one discovers that creating subwindows allows for tasks to be encapsulated into windows. These observations make the rationale for creating a toolkit obvious. The framework of a toolkit is an object oriented one. A toolkit contains reusable, configurable user interface components (such as pushbuttons, scroll bars, pop up menus) called widgets. Widgets operate independently of the application [**?**]. For example a pushbutton knows how to draw itself, how to highlight itself when it is clicked on with the mouse, and how to respond to a mouse click by calling an application function. The object oriented approach of building a toolkit means that the application programmer is completely insulated from the code inside the widgets. All he has access to are functions to create, manipulate and destroy widgets. This way the internal implementation of a widget can change, without requiring changes to the application itself. A further benefit is that an object oriented approach forces the programmer to think about the application in a more abstract fashion, which leads to better design and fewer bugs. In an object oriented approach of creating a toolkit certain base classes of widgets are defined, whose behaviour can be inherited and augmented or modified by their subclasses. In FontCreator toolkit the base widget is the ToolWindow.

2.6 Motif and Open Look.

Motif and Open Look are sets of guidelines specifying how a user interface should look (how an application appears on the screen) and feel (how the user interacts with it). Open Look was created by AT&T and Sun Microsystems while Motif was created by Open Software Foundation. Both include many things : specifications for a graphical interface, style guides for providing application consistency, toolkits and window managers. Open Look provides olwm and Motif provides mwm window manager. Motif toolkit is based on Xt Intrinsics while Open Look/XView toolkit is based on X Library. Although many people would think that programs using one or the other toolkit would be incompatible with each other this is not the case. Motif programs can easily be run on an Open Look system and vice versa. Of course without mwm Motif programs will not have the look and feel they do, but they still be able to run. Programs based on X Library will run with both window managers as well as with twm.

Chapter 3

Fonts in X Windows.

3.1 About fonts.

A font is a graphical description of a set of characters. In general, there are two font representations used by X : outline and bitmap. Outline fonts are fonts whose characters are described as outlines. Bitmap fonts are fonts which characters are described as two-dimensional arrays of bits. BDF [?], or Bitmap Distribution Format is the standard font interchange format for X Consortium. It consists of ASCII text, understandable by both human and computers. SNF, or Server Natural Font is the standard format used by X servers, for release 4 of X11. SNF files have an .snf extension. From release 5 of X11 a new format has been added which is portable across all hosts, and is called Portable Compiled Format or PCF. PCF font files have an extension of .pcf. The placement of the character's bitmap on the screen, is controlled by properties such as the ascent, the descent, the character's width, height, and spacing. We will look in more detail in character's properties.

3.2 Character's properties.

We will define the most basic of the character's metrics.

- Character's width is the number of pixels, between the leftmost and the rightmost parts of the character.
- Character's height is the number of pixels, between the uppermost and downmost parts of the character.

- Character's ascent is the extent of the character above the baseline.
- Character's descent is the extent of the character below the baseline.
- Character's x displacement is the horizontal distance, of the lower left corner from the origin of the character.
- Character's y displacement is the vertical distance, of the lower left corner from the origin of the character. Both x and y displacement can be positive or negative. For example, character "p" extents below the baseline, therefore the baseline is in its font box, and y displacement will be negative (and equal to the extent of the character below the baseline). Apostrophe "'" character on the other hand, has the baseline underneath its font box, and the y displacement will be positive.
- Scalable width in x and y of character is a vector, indicating the position of the next character's origin relative to the origin of this character. Scalable widths are in units of 1/1000th of the size of the character. If the size of the character is p points, the width information must be scaled by p/1000 to get the width of the character in printer's points. The scalable width's x value can be computed from the following equation : x = (1000 / charsize) * (72 / resolution) * charwidth, where charsize is the character's size, resolution is the device resolution in pixels per inch, and the charwidth is the character's width. The result is a real number giving the ideal print width in device pixels. The actual device width must be an integral number of device pixels. The scalable width's y value should always be zero for a standard X font.
- Width in x and y of character in device pixels, is also a vector indicating the position of the next character's origin relative to the origin of this character. The device width's y value should always be zero for a standard X font.

3.3 X Logical Font Description.

3.3.1 Requirements and goals.

The X Logical Font Description [**?**, p.53] meets the sort and long term goal to have a font description that :

- Provides unique descriptive font names that support simple pattern matching.
- Supports multiple font vendors, arbitrary character sets, and encodings.
- Is independent of X server and operating or file system implementations.
- Supports arbitrarily complex font matching or substitution.
- Is extensible.

In more detail: It should be possible to have font names that are long enough and descriptive enough to have a reasonable probability of being unique .The name itself should be structured to be amenable to simple pattern matching, making simple for X clients to limit font queries to a subset of all possible fonts. X clients applications that require a particular font should be able to load it, without knowledge of the file system. The X Logical Font Description must be extensible so that new font properties can be added to conforming fonts, without making old files obsolete or incompatible.

3.3.2 FontName description.

Each FontName is composed of two strings : a FontNameRegistry prefix, which is the registration authority that ows the font and a FontNameSuffix. In the X protocol specification, the FontName is required to be a string, therefore numeric fields are represented in the name as string equivalents. The FontName structure is the following:

-FOUNDRY-FAMILY_NAME-WEIGHT_NAME-SLANT-SETWIDTH_NAME-ADD_STYLE_NAME-PIXEL_SIZE- POINT_SIZE-RESOLUTION_X-RESOLUTION_Y-SPACING-AVERAGE-WIDTH-CHARSET_REGISTRY-CHARSET_ENCODING

We will describe each of these fields separately.

FOUNDRY field.

FOUNDRY is an X registered name, the name or identifier of the supplier of the font data, or the identifier of the organisation that last modified the font shape or metric information.

FAMILY_NAME field.

FAMILY_NAME is a string that identifies the family of typeface designs that are all variations of one basic typographic style. This name must be human-understandable. It is up to the type supplier to secure the proper legal title and not to infringe other's copyrights or trademarks. Examples of FAMILY_NAME are : Helvetica, Times, Times Roman, Bitstream Amerigo.

WEIGHT_NAME field.

WEIGHT_NAME identifies the font's typographic weight, that is the blackness of the font, according to FOUNDRY's judgement. The interpretation of this field can be problematic, since it is subjective. It is possible that a font which is identified as Bold from a font family, to have almost the same blackness of a SemiBold font from another font family.

SLANT field.

SLANT is an encoded string, identifying the way in which the character is designed. The following are possible codes:

- "R" for Roman, that is upright design.
- "I" for Italic, that is italic design, slanted clockwise from the vertical.
- "O" for Oblique, obliqued upright design, slanted clockwise from the vertical.
- "RI" for Reverse Italic, italic design, slanted counterclockwise from the vertical.
- "RO" for Reverse Oblique, obliqued design, slanted counterclockwise from the vertical.

SETWIDTH_NAME field.

SETWIDTH_NAME is human-understandable string, identifying the nominal width per horizontal unit of the font, according to the FOUNDRY's judgement. As with WEIGHT_NAME, since its definition is subjective, the interpretation can be problematic. What can be characterised as "Normal" for a family font, may be almost equivalent in typographic feel to a "Narrow" font from another family. Examples of SETWIDTH_NAME are Normal, Condensed, Narrow, and Double Wide.

ADD_STYLE_NAME field.

ADD_STYLE_NAME is a string that identifies additional typographic style information, not captured by other fields. Examples of ADD_STYLE_NAME are Serif, Sans Serif, Informal, and Decorated. Serif characters have a smaller line that finishes off a large stroke, where Sans Serif characters do not have these finishing strokes.

PIXEL_SIZE field.

PIXEL_SIZE is an unsigned integer string, giving the body size of the font at a particular POINT_SIZE and RESOLUTION_Y. PIXEL_SIZE can be approximated by the following equation : PIXEL_SIZE = ROUND((RESOLUTION_Y * POINT_SIZE) / DeciPointsPerInch) where the value of DeciPointsPerInch is 722.7 .

POINT_SIZE field.

POINT_SIZE is an unsigned integer string, giving the body size of the font, in device independent units. POINT_SIZE is not necessarily equivalent to the height of the font bounding box. POINT_SIZE is expressed in decipoints (72.27 points equal one inch).

RESOLUTION_X field.

RESOLUTION_X field is an unsigned integer string, giving the horizontal resolution in dots per inch (dpi), for which the font was designed.

RESOLUTION_Y field.

RESOLUTION_Y field is an unsigned integer string, giving the vertical resolution in dots per inch (dpi), for which the font was designed.

SPACING field.

SPACING is an encoded string that gives the escapement class of the font. A font can be proportional spaced or fixed width. A fixed width allows the same width for all characters. In a fixed width font, a w will take the same space as a l. In a proportional spaced font the space each character takes, is character dependent. The encoding of SPACING is as follows : "P" standing for proportional, "M" standing for monospaced, and "C" standing for CharCell.

CharCell is a monospaced font that follows the standard typewriter character cell model, that is that all characters can be modelled as ”boxes” of the same width and height.

AVERAGE_WIDTH field.

AVERAGE_WIDTH is an unsigned integer string typographic metric value that gives the mean width of all characters in the font. For monospaced and charcell font this is the width of all characters in the font.

CHARSET_REGISTRY and CHARSET_ENCODING fields.

CHARSET_REGISTRY and CHARSET_ENCODING are registered names identifying the registration authority that owns the specified encoding, and the coded character set as defined by that registration authority, respectively. The following CharSet names are registered for use in font names under the X Logical Font Description.

ISO8859-1	Latin alphabet No 1
ISO8859-2	Latin alphabet No 2
ISO8859-3	Latin alphabet No 3
ISO8859-4	Latin alphabet No 4
ISO8859-5	Latin/Cyrillic alphabet
ISO8859-6	Latin/Arabic alphabet
ISO8859-7	Latin/Greek alphabet
ISO8859-8	Latin/Hebrew alphabet
ISO8859-9	Latin alphabet No 5
JISX0201.1976-0	8-bit Alphanumeric Katakana Code
GB2312.1980-0	GL encoding, China (PRC) Hanzi
JISX0208.1983-0	GL encoding, Japanese Graphic Character Set
KSC5601.1987-0	GL encoding, Korean Graphic Character Set

3.3.3 Examples.

The following are examples of font names :

75 dpi fonts

```
-Bitstream-Charter-Medium-R-Normal--12-120-75-75-P-68-ISO8859-1
-Bitstream-Charter-Bold-I-Normal--12-120-75-75-P-75-ISO8859-1
-Adobe-Courier-Medium-R-Normal--8-80-75-75-M-50-ISO8859-1
-Adobe-Courier-Medium-R-Normal--10-100-75-75-P-124-ISO8859-1
```

-Adobe-Times-Medium-R-Normal–24-240-75-75-P-124-ISO8859-1

100 dpi fonts
-Adobe-Times-Bold-R-Normal–14-100-100-100-P-76-ISO8859-1
-Adobe-Times-Bold-I-Normal–14-100-100-100-P-77-ISO8859-1
-Adobe-Times-Medium-I-Normal–14-100-100-100-P-73-ISO8859-1
-Adobe-Times-Medium-R-Normal–14-100-100-100-P-74-ISO8859-1

3.3.4 Font properties.

All font properties [**?**, p.67] are optional, but a font file will generally include the font name fields already discussed and some more useful information. We will describe some of the font properties, not already described.

MIN_SPACE.

MIN_SPACE is an unsigned integer value that gives the recommended minimum word space value, to be used with this font.

NORM_SPACE.

NORM_SPACE is an unsigned integer value that gives the recommended normal word space value, to be used with this font.

MAX_SPACE.

MAX_SPACE is an unsigned integer value that gives the recommended maximum word space value, to be used with this font.

END_SPACE.

END_SPACE is an unsigned integer value that gives the recommended spacing at the end of sentences, to be used with this font.

AVG_CAPITAL_WIDTH.

AVG_CAPITAL_WIDTH is an integer value that gives the mean width of all capital letters in the font (letters A through Z for Latin fonts).

AVG_LOWERCASE_WIDTH.

AVG_LOWERCASE_WIDTH is an integer value that gives the mean width of all lowercase letters in the font (letters a through z for Latin fonts).

SUBSCRIPT_X.

SUBSCRIPT_X is an integer value that gives the recommended horizontal offset in pixels from the position point to the X origin of subscript text.

SUBSCRIPT_Y.

SUBSCRIPT_Y is an integer value that gives the recommended vertical offset in pixels from the position point to the Y origin of subscript text.

SUBSCRIPT_SIZE.

SUBSCRIPT_SIZE is an unsigned integer value that gives the recommended body of subscripts to be used with this font (it is usually smaller than the size of the current font).

UNDERLINE_POSITION.

UNDERLINE_POSITION is an integer value that gives the recommended vertical offset in pixels, from the baseline to the top of the underline.

UNDERLINE_THICKNESS.

UNDERLINE_THICKNESS is an unsigned integer value that gives the underline thickness, in pixels.

ITALIC_ANGLE.

ITALIC_ANGLE is an integer value, that gives the nominal posture angle of the typeface design, in 1/64 degrees, measured from the character origin counterclockwise from the three o'clock position.

CAP_HEIGHT.

CAP_HEIGHT is an unsigned integer that gives the nominal height of the capital letters contained in the font. Where applicable, it is defined to be the height of the Latin uppercase letter X.

X_HEIGHT.

X_HEIGHT is an unsigned integer that gives the nominal height above the baseline of the lowercase characters contained in the font. Where applicable it is defined to be the height of the Latin lowercase letter x.

COPYRIGHT.

COPYRIGHT is a human understandable string containing the copyright information of the legal owner of the digital font data.

NOTICE.

NOTICE is a human understandable string that gives the copyright or the trademark information of the legal owner of the font.

DEFAULT_CHAR.

DEFAULT_CHAR is an unsigned integer value that gives the ASCII code of the default character to be used by the X server when an attempt is made, to display an undefined or non existent character in the font.

3.4 Bitmap Distribution Format Version 2.1

The Bitmap Distribution Format Version 2.1 [**?**, p.50] is an X Consortium standard for font interchange, understandable by both humans and computers.

3.4.1 File format.

Each font file is encoded in ASCII printable characters plus carriage return and linefeed. The general form of a font file is the following :

- The word STARTFONT with the version number (in this case 2.1) indicate the format version used.
- The word COMMENT with comments about the font. These may extend to more than one lines, all lines starting with the word COMMENT. These lines may be ignored by any program reading the file.

- The word FONT followed by either the X Logical Font Description font name or some private font name.
- The word SIZE followed by the point size, the x and the y resolution, described earlier.
- The word FONTBOUNDINGBOX followed by the width, height, x and y displacement of the character.
- The word STARTPROPERTIES with the number p of properties that follow.
- Then follow p lines of properties, starting with the property name and an integer or a string contained in double quotes. Properties like FONT_ASCENT, FONT_DESCENT, and DEFAULT_CHAR should be provided. If these properties are not defined, a compiler may reject the font.
- The property section is ended with the word ENDPROPERTIES.
- The word CHARS followed by the number of character segments contained in the font file.
  - Each character segment starts with the word STARTCHAR followed by the name of the encoded character (up to 14 characters long).
  - Next comes the word ENCODING followed by a positive integer which is the character code as defined by the encoded character set given by CHARSET_REGISTRY-CHARSET_ENCODING font properties. For ISO8859-1, this code is the same as the character ASCII value.
  - The word SWIDTH followed by the scalable width in x and y of character (as defined earlier).
  - The word DWIDTH followed by the width in x and y of the character in device units (as defined earlier).
  - The word BBX followed by the width, height, x and y displacement of the lower left corner from the character's origin.
  - The word BITMAP
  - Then come h lines of the hex-encoded bitmap, padded on the right with zeros to the nearest byte (multiple of eight).
  - The character segment ends with the word ENDCHAR.
- The file is terminated with the word ENDFONT.

3.4.2 File format example.

```
STARTFONT 2.1
FONT -Misc-Greek-Bold-R-Normal--24-240-75-75-P-65-ISO8859-1
SIZE 24 75 75
FONTBOUNDINGBOX 9 24 -2 -6
STARTPROPERTIES 20
FOUNDRY "Misc"
FAMILY "Greek"
WEIGHT_NAME "Bold"
SLANT "R"
SETWIDTH_NAME "Normal"
ADD_STYLE_NAME ""
PIXEL_SIZE 24
PIXEL_POINT 240
RESOLUTION_X 75
RESOLUTION_Y 75
SPACING "P"
AVERAGE_WIDTH 65
CHARSET_REGISTRY "ISO8859"
CHARSET_ENCODING "1"
MIN_SPACE 4
FONT_ASCENT 21
FONT_DESCENT 7
DEFAULT_CHAR 32
COPYRIGHT "public domain"
NOTICE ""
ENDPROPERTIES
CHARS 70
STARTCHAR b
ENCODING 98
SWIDTH 360 0
DWIDTH 8 0
BBX 9 22 -2 -6
BITMAP
1C00
3E00
3700
6300
```

```
6300
6300
6300
6700
FE00
F800
3E00
7F00
6300
6300
7F00
3E00
0000
0000
0000
0000
0000
0000
ENDCHAR

.....
more characters
.....

STARTCHAR 9
ENCODING 57
SWIDTH 360 0
DWIDTH 8 0
BBX 9 22 0 -6
BITMAP
0000
0000
0000
0000
0000
1E00
3F00
3300
3300
```

```
3F00
1E00
0600
0C00
1800
3000
3000
0000
0000
0000
0000
0000
0000
ENDCHAR
ENDFONT
```

Chapter 4

FontCreator.

4.1 The need for FontCreator.

The most common operation in a windowing system is drawing text. Almost every application program, big or small, relies on text in one way or another. Although X Windows support a lot of different fonts, there can be many reasons to justify the need of creating new fonts. First of all, although X Windows support a lot of different fonts, as already mentioned, not all X servers support all fonts. This is the usually the case with foreign alphabets. FontCreator closes this gap, by giving users the means to create their fonts. But FontCreator also gives ground for experimenting. One of the possible experiments could be to investigate the user's response when presented with text, printed in their own handwritting. For this experiment, a sample of users would first create fonts resembling their handwritting and then use these fonts with text operations, such as editing. The experiment could then focus on evaluating whether the users worked more efficiently when they were using fonts similar to their handwritting, rather than the standard X fonts. FontCreator seeks to enable users to create easily, their own font files for use with their applications.

4.2 Graphical User Interfaces.

Graphical User Interfaces (GUIs) [?, p.64] are user interfaces which rely on graphics in order to communicate with the user. WIMP interfaces (derived from Windows, Icons, Menus and Pointing) is a type of graphical user interface which use windows, pull down or pop up menus and pointing devices.

WIMPs have a lot of advantages over traditional user interfaces. They are supportive of the user making users feel in control of the computer, they are easy to learn and use and they provide full screen interaction. WIMPs are characterised by multiple windows, allowing different information to be displayed simultaneously, iconic information representation, command selection via menus and the use of a pointing device such as a mouse or a trackball, to select choices from a menu or indicate objects on the screen. Until recently there were different user interface management systems produced by different vendors, which were incompatible with each other. The Motif interface toolkit however, has been a move to standardisation on the X Window interface.

4.2.1 Interface design

While designing the FontCreator interface, there was an argument whether it should be based upon available toolkits like Open Look or Motif toolkit, or a new toolkit should be built especially for the program needs. The latter approach was chosen, with the rationale of exploring the design decisions that toolkit builders face. This toolkit will be described later. For the moment we will focus on the guidelines upon which, the interface design was based. First of all it was decided that the interface should comply with the Motif style specifications. The Motif interface was modelled after the Microsoft's Common User Access specifications, which define the interface for Microsoft Windows. The key point of the specification is that consistency should be maintained across all applications. Similar user interface elements should look and act similarly regardless of the application that contains them. When the user interface is standardised, the user gets more quickly to the point where he is working with the application, rather than just mastering its mechanics. According to the Motif Style Guide [**?**, p.6], the main window should have a menu bar across the top and a message window at the bottom. This message area is not used for messages that require a response from the user. Messages requiring a response, cause a transient window called a dialogue box, to be displayed. Dialogue boxes are pop up windows that contain a text label on the top, right beneath it there is optionally a text entry window and at the bottom there are two push buttons named "Ok" and "Cancel". The main point when designing the FontCreator interface was to keep it as simple as possible. Having this point in mind the menu bar was designed to contain only five push buttons : "Load", "New", "Edit", "Quit" and "Help". The need for having "Load" and "New" pushbuttons is obvious. The user may want to load an existing font file or create a new one. But along with these two buttons shouldn't there be and a "Save" one. After careful consideration the answer to this question was negative. The user does not have to save the file. The font file loaded or created, exists anyway. What the user saves are the characters. So, if a save question should be asked to the user, that should happen once the user has finished drawing a character. For this reason it was decided that each time the user finishes drawing a character's bitmap, a dialogue box should appear asking him whether to save or not. Simplifying the user interface is more easily said than done. This became apparent when designing the dialogue boxes for the "Edit" procedure. If the font is monospaced, (where all characters can be viewed as boxes of the same width and height), the user has to enter only the character to be edited. The width, height x displacement and y displacement of each character remain the same for all character in the font, and can be calculated from the font properties defined in the font file. But, for a proportional font, (where the size of character is not constant), the user has to enter the width, height, x displacement and y displacement of the character. What makes things worse, is the fact that novice users may find it difficult to understand what exactly is meant by x and y displacement. The best solution would have been, to design the program so that it could scan the font file, to determine whether the font was fixed width or proportional and ask for more information only in the second case. Unfortunately due to lack of time this solution could not be implemented (although we hope that a next version of the program will adopt it). Therefore there were two alternatives : either create a program that could only manipulate fixed width fonts (and reduce the information needed from the user), or create a program that could handle all types of fonts (but on the other hand increase the stress upon the user). The second alternative was chosen since proportional fonts account for the largest part of fonts being used and could not have been excluded so easily. Another point where special attention was paid, was the design of dialogues. All dialogues had to be uniform and their look and feel should remain the same. For this reason the same dialogue box was used through out all the dialogues. What changed was the message at the text label. Clicking on "Ok" pushbutton would always signalled that the user accepted the message (and if appropriate had entered input) and clicking on "Cancel" pushbutton would signalled that the user wanted to abort the operation. Providing a cancel possibility was a way to make the user confident that even if he made an error he could avoid bad consequences, and abort the operation. The next point that needed consideration was the messages that should appear in the message window. The type of messages that should be displayed in the message window, becomes clear once the need for providing such a window is thought about. Its use is to make apparent the state of the program [?], therefore messages such as "Loading a font file", or "Editing character's bitmap" should be printed in it. This way useful feedback is provided to the user and the user can act more readily.

4.2.2 Interface implementation.

FontCreator's main window consists of five pushbuttons, placed at the top of the window and labelled as "Load", "New", "Edit", "Quit" and "Help" and a message window placed at the bottom of the main window, capable of displaying various messages for the user, depending on the state of the program and the user's actions. In more detail : The first push button, from the left, is "Load" button. By placing the mouse pointer on it and holding down the mouse button the user is informed for the "Load" pushbutton operation, with a message displayed at the message window. By clicking the mouse button on "Load" the load procedure is initiated. If a font file was being edited the program will first save the file before continuing the load operation. The user is prompted by a pop up window to enter the file name to be loaded. At any time the user can cancel the operation by clicking on the "Cancel" pushbutton of the dialogue box. If the user chooses to continue and enter a filename the program will try to load the file. On success a message "File loaded ok" will appear on the message window, otherwise "File not found" will be displayed and a warning sound will be heard. The second pushbutton is "New" button. By holding down the mouse button, while the mouse pointer is on "New", the user can be informed of the "New" operation, by a message displayed in the message window. By clicking on "New" pushbutton the user initiates the procedure of creating a new font file. If a font file was being edited the program will first save the file before continuing the new operation. The first pop up window that appears asks for the font family name. Next the user is asked whether he wants to edit the default font properties file. If the user selects to edit the file a full screen editor will appear with the defaults file already loaded, enabling the user to make any alterations required. Once the user exits the editor, the program will scan the file to determine the new file name. If this filename exists, the user will be prompted with a pop up dialogue box, asking whether the existing file will be overwritten. Again, during the whole new procedure, the user can abort at any time by clicking the "Cancel" pushbutton of any of the pop up windows appearing . The third pushbutton is the "Edit" button. By placing the mouse pointer on it and holding down the mouse button a message displayed at the message window informs the user of the "Edit" operation. By clicking on "Edit" pushbutton a series of pop up windows appear, prompting the user to enter the character to be edited, the character width, height x and y displacement. At any time the user can quit the edit procedure by clicking on the "Cancel" pushbutton of any of the pop windows that appear. The program performs validity checks on the user's input. If the input is invalid, or out of range, the user is informed and asked to reenter the value. For example if the user tries to edit a non printable character, enter a non integer value when an integer was expected or enter a negative width or height the program will complain. If the character to be edited already exists, before continuing the program will ask the user whether to overwrite the existing character or abandon the editing operation. When the user finishes entering the required input, two windows appear. The bigger window is the edit window, which has a grid drawn into, where the user can draw the bitmap of the character. A smaller window at the upper right corner of the first, displays the actual character as it is being drawn. The grid displayed in the edit window consists of w columns and h rows, where w and h are the width and height of the character respectively, entered by the user. A line drawn in the edit window shows where the baseline is. At the bottom of the edit window, a small message window reminds the user of the character being edited. When the mouse pointer enters the edit window, it changes from being an arrow to a pencil, implying that this is drawing area. By clicking the left mouse button on grid's square, the square changes color from white to black and vice versa. By pressing the right mouse button, a dialogue box asks the user whether he wants to quit editing. Clicking on "Cancel" pushbutton returns the user back to editing mode, while clicking on "Ok" pushbutton, pops up another dialogue box, asking whether to save the edited character or not. During the editing procedure, the user is assisted by useful messages, printed in the main message window, helping him accomplish the character's editing. The fourth push button is the "Quit" button. Pressing the mouse button, while the mouse pointer is on it, displays a message in the message window, explaining quit's use. Clicking on the "Quit" pushbutton pops up a dialogue box asking the user to confirm his decision to exit FontCreator. Again clicking on "Cancel" aborts the exit procedure and returns the program back to its previous state. If the user insists the program will save the font file edited (if any) and exit. The fifth pushbutton is the "Help" button. Its operation is self evident. Clicking on "Help" pushbutton, makes the help window appear and on line help to be displayed in it. Clicking on "Ok" push button of the help window closes the help window.

4.3 FontCreator toolkit.

4.3.1 ToolWindow structure.

As already mentioned in FontCreator the base widget is the ToolWindow. The ToolWindow structure [**?**, p.49] contains a pointer to the display on which the window is to be mapped, the parent of the window, the window itself, information about the window's graphics context, font, width, height and name. Perhaps the most important element of the structure are three functions, namely drawfunc, userfunc and eventfunc. The drawfunc function redraws the window after an expose event has occurred, the eventfunc function takes care of all events that take place in the window and userfunc can be associated with a user callback function. Not all of these functions need to be defined for every of ToolWindow subclasses, except perhaps for the drawfunc. Since the toolkit is built in an object oriented way the user does not access the ToolWindow fields directly. To manipulate ToolWindow object a set of functions is provided, that allow the user to set or get the values of ToolWindow's fields. In the small toolkit used for FontCreator, an array of ToolWindow structures was used to hold all ToolWindow structures created. For a larger application this would have been inefficient. Instead a linked list of ToolWindow, would permit the user to create as many ToolWindows as the computer's memory allowed and manipulate the ToolWindows in a much more efficient way.

4.3.2 Text Label widget.

The simplest of all widgets is the Text Label widget. The Text Label widget is used only to display text. Since there is no input or user action associated with it, there is no need for an event function or user callback function. The only events the Text Label has to take care of, are expose events. For this reason LabelDraw function is implemented. LabelDraw redraws the Text Label in case of an expose event.

4.3.3 PushButton widget.

The PushButton widget is much more complicated widget. A PushButton can be pressed or released and has a user callback function associated with it. When a ButtonPress event occurs, the 3D bevel of the PushButton is reversed. This way the effect of the button being pushed in, is created. On the arrival of a ButtonRelease event the 3D bevel is restored to its normal state giving the look of the button being pushed out. When the PushButton is pushed in, the ProvidePushButtonMessage function is executed, and if a message exists for that PushButton it will be displayed in the message window. This gives the opportunity to the programmer to display a message explaining the use of the PushButton, that is what will happen when the PushButton's callback function is executed. If the time between the moment the pushbutton is being pressed and released is less than response time (a constant which was set to 500 milliseconds for FontCreator), the callback function will be executed. The reason of implementing the PushButton event function this way is that the user can either press the button to see its operation or click it to execute the callback function. On expose events, PushButtonDraw function redraws the PushButton.

4.3.4 Text Entry widget.

The Text Entry widget is used to get input from the user. The programmer can associate a user callback function with it, that will be executed when the user presses $< Return >$. The TextEntryEvent function handles user's key presses and responds accordingly. If the user enters printable characters, the function will append them to the ones already entered, and if the width of the Text Entry widget is not long enough for all the characters to be displayed, it will scroll them to the left. If the user presses $< Delete >$ or $< BackSpace >$, the last entered character will be deleted, allowing the user to correct wrong input. On expose events the TextEntryDraw function will redraw the Text Entry widget.

4.3.5 Dialogue window.

The dialogue window is a transient pop up window, having a Text Label widget at the top for messages to be displayed, just beneath a Text Entry widget for user's input, and two PushButton widgets at the bottom named "OK" and "Cancel".Pressing the $< Return >$ key has the same result as Clicking on the "OK" button. There are two ways for a pop up window to be mapped on the screen. One way is to find pointer's location and to display the window as close as possible. The other way is to pop up the window anywhere on the screen and move the mouse pointer into the window (often called warping the mouse). It was thought that the first way of implementing a pop up window was better, since warping the mouse is unexpected and tends to confuse the user. Two functions control the display of the dialogue box. Every time the dialogue is popped down, it is not destroyed, as creating a window takes a relatively long time. PopDownDialog simply unmaps the dialog, which can then be used again with another prompt and different purpose, but with the same look and feel. In certain cases, it is necessary to pin the dialogue box. For this reason PinDialog function is provided, which pins the dialogue at a certain position. UnpinDialog function undoes the result of PinDialog. GetDialogData returns a string of characters containing user's input and SetDialogCallback and SetDialogCancelCallback functions can be used to associate callback functions with "OK" and "Cancel" PushButtons of the dialogue box. SetDialogPrompt and SetDialogData can be used to set the dialogue prompt and to initialise the data in the Text Entry field, respectively. There is no need to provide redraw function, since in case of expose events all dialogue components have their own redraw functions to handle expose events.

4.4 How FontCreator works.

FontCreator views each font file as consisting of two separate files: a header file that contains the font properties and a characters' file that contains all information such as the character's encoding, width, height, and bitmap for all characters. Adopting this framework, when a font file is loaded, it is broken down to "header.txt" and "chars.txt" files. "header.txt" contains the font properties and "chars.txt" contains all character segments. The number of characters is not contained in neither file. Instead, this number along with a list of all the characters contained in the file is being kept by the program and updated every time a new character is saved. This way the program can easily check, if the user is trying to edit a character that already exists, and inform the user and ask him about the step to be taken next (overwrite the existing character or abort the editing procedure). Likewise when a new font file is created, the program creates a header file and initialises the value of characters number to zero. The user may select to use the default font properties file as is, or make changes. If the user chooses the latter possibility, a full screen editor is called within the program and the user can easily edit the defaults file. Once the user quits the editor, the program will scan the file to determine the new font file name. This will be the font family name with the .bdf extension. If a file with the same name exists, the user is asked whether to overwrite the file or abort the procedure. If the "New" procedure is successful the message "File created" is displayed, in the message window, otherwise the message "No file created" appears. The edit procedure starts, with the program asking for the character to be edited. If the user inputs a non printable character, the program will issue a warning message and wait for valid input. If the character already exists, the user may select to overwrite the previous character or abort the edit operation. Next the program will ask consecutively for character's width, height, x and y displacement. All input is checked for validity. The program will complain if the input is not integer or an invalid integer (for example if the user enters a negative character width), and wait for valid input to be entered. When the user has input all required values he can draw the character glyph. The program keeps an two dimensional array, holding the bitmap of the character as zero (white) and one (black) values. Each time the user draws a pixel of the character, the program updates the array as well as the edit window and the actual display window. This latter window displays the character as it will actually appear on the screen in its real size. The edit window displays the character magnified, so as to make it easy for the user to draw. When the user finishes the draw procedure, the program asks whether to save the character or not. If the user selects to save the character, the program appends the character to "chars.txt" and updates the character list and character number. When the user clicks on "Quit" pushbutton, the program first checks the state of the program. If there is some other operation on, the program will first ask the user whether to abort it or not. If no other operation is on, the program will ask the user to affirm his choice. If the user insists on his choice, the program will merge "header.txt" and "chars.txt" files and name the new file after the filename. When designing the program it was thought that no special "Save" button had to be implemented. It would rather complicate the interface, rather than make it easier for the user. Instead a mechanism is provided that merges the two files whenever the user quits or loads or creates a new font file. This way the procedure of breaking up and merging the font file is kept transparent to the user. The user has only to decide whether a character should be saved in the font file or not. The program keeps track of its state by use of special flags, thus achieving two things. One, it knows whether the user tries to load or create a font file while he is editing a character, or tries to edit a character while no file has been opened, or tries to make an operation which will put data in risk or simply is unacceptable. This way when such a case occurs the program will involve the user in a dialogue (by means of pop up windows) trying to figure out what the user really wants to do and make sure that is done properly. The other thing achieved is that the user has on line help that corresponds to the operation being done. For this reason four help files, namely ”general.hlp”, ”load.hlp”, ”new.hlp” and ”edit.hlp” are provided. Each time ”Help” pushbutton is clicked the program first checks its state, to decide which help file is appropriate to be loaded, and then displays the help information in the special help window. The program displays all messages to user in its message window. These messages include messages informing the user of the state the program is in, such as ”saving file”, messages that affirm that action has been taken, such as ”File loaded”, or warning messages (with a beep sound). All these messages are stored in ”messages.txt” file. This way messages can easily be changed, if need for such action arises

Chapter 5

References

[1] R. W. Scheifler & J. Gettys, *X Window System*, Second Edition Digital Press, USA 1990.

[2] D. McNutt & M. O'Neal, The Administrator: Font Formats and Utilities. *The X Resource*, Issue 2 Spring 1992.

[3] I. Sommerville, *Software Engineering*, Fourth Edition, Addison Wesley Ltd, New York, 1992.

[4] D. Heller, *Motif Programming Manual* O'Reilly & Associates, Inc., Sebastopol, USA, 1991.

[5] E. F. Johnson & K. Reichard *X Window Applications Programming*, MIS Press, New York, 1992.

Chapter 6

Appendix A : User Manual

6.1 Introduction

Welcome to FontCreator. FontCreator will enable you to create your own fonts for X Windows environment. Although some knowledge about fonts is desirable, we believe that FontCreator can be easily used by novice users. Our intention was to make the more complicated steps of creating a font, transparent to the user. This is the reason why a file with default font properties is provided along with the program. Of course you may select to edit this file to change the predefined font properties. We do not recommend editing this file, unless you are certain of the changes you are making, otherwise undesired effects may occur. We also recommend that you should read this manual carefully, before operating FontCreator. You should also be aware that certain font names are registered trademarks and that for most of the font files copyright exists. Apart from FontCreator itself, the following files should be present to ensure that FontCreator operates properly : defaults.txt, messages.txt, load.hlp, new.hlp, edit.hlp, and general.hlp. If one or more of the above files is missing, FontCreator will issue a message informing the user of the missing file. We hope that you will enjoy using FontCreator. For any questions or suggestions you can contact : G.Lepouras . University of Strathclyde. E-Mail : glepoura@cs.strath.ac.uk

Notice

6.2 Getting started.

FontCreator is designed to run under X Windows version 11, release 4 or later. If you try to execute FontCreator directly from the Unix prompt the message "Cannot connect to X server" will appear. You can use any of the existing window managers such as twm or olwm. However we suggest that you use X Windows window manager (twm). Twm provides special decorations for pop up windows and responds (from our experience), more readily to the program's requests. We have found, that sometimes this is not the case with olwm (Open Look window manager), which sometimes does not respond accordingly, especially when conflicts arise between client programs. Make also sure that at least "defaults.txt", the default font properties file is present. Once you run X Windows, give the FontCreator command at the console, followed by $< Return >$. What you first see is the FonCreator main window. You can use the mouse pointer to resize or move the main (or any other) window of FontCreator. The first step you have to take, before drawing any character bitmap, is to load or create a font file.

6.3 Loading a font file.

"Load" push button is used to load a font file. The file to be loaded should conform to the Bitmap Distribution Format ver 2.1. To load the file, click on "load" push button to initiate the loading process, then enter the full file name in the special text input area, of the pop up window, and click on "OK". if you want to abort the operation just click on "Cancel". If the load operation is successful, "File loaded" is displayed in the message window, otherwise if the file does not exist "File not found" will be displayed. If another font file was being edited, while the load procedure was initiated, it will be automatically saved and the message "File saved" will appear in the message window.

6.4 Creating a new font file.

Click on "New" pushbutton to initiate the procedure of creating a new font file. The first thing you will be asked is to enter the new font file name. The program itself will provide the identifier .bdf at the end of every font file you create. The program will then ask you if you want to edit the default font properties file. You can answer yes or no depending on whether you want to edit the file. If you answer yes a full screen editor will appear containing the defaults file. We give a brief explanation of the font properties included in the file.

- FOUNDRY is an X registered name, the name or identifier of the supplier of the font data, or the identifier of the organisation that last modified the font shape or metric information.
- FAMILY_NAME is a string that identifies the family of typeface designs that are all variations of one basic typographic style. This name must be human-understandable. It is up to the type supplier to secure the proper legal title and not to infringe other's copyrights or trademarks. Examples of FAMILY_NAME are : Helvetica, Times, Times Roman, Bitstream Amerigo.
- WEIGHT_NAME identifies the font's typographic weight, that is the blackness of the font, according to FOUNDRY's judgement. The interpretation of this field can be problematic, since it is subjective. It is possible that a font which is identified as Bold from a font family, to have almost the same blackness of a SemiBold font from another font family.
- SLANT is an encoded string, identifying the way in which the character is designed. The following are possible codes:
  - "R" for Roman, that is upright design.
  - "I" for Italic, that is italic design, slanted clockwise from the vertical.
  - "O" for Oblique, obliqued upright design, slanted clockwise from the vertical.
  - "RI" for Reverse Italic, italic design, slanted counterclockwise from the vertical.
  - "RO" for Reverse Oblique, obliqued design, slanted counterclockwise from the vertical.
- SETWIDTH_NAME is human-understandable string, identifying the nominal width per horizontal unit of the font, according to the FOUNDRY's judgement. As with WEIGHT_NAME, since its definition is subjective, the interpretation can be problematic. What can be characterised as "Normal" for a family font, may be almost equivalent in

typographic feel to a "Narrow" font from another family. Examples of SETWIDTH_NAME are Normal, Condensed, Narrow, and Double Wide.

- ADD_STYLE_NAME is a string that identifies additional typographic style information, not captured by other fields. Examples of ADD_STYLE_NAME are Serif, Sans Serif, Informal, and Decorated. Serif characters have a smaller line that finishes off a large stroke, where Sans Serif characters do not have these finishing strokes.

- PIXEL_SIZE is an unsigned integer string, giving the body size of the font at a particular POINT_SIZE and RESOLUTION_Y. PIXEL_SIZE can be approximated by the following equation :

  $$PIXEL\_SIZE = ROUND((RESOLUTION\_Y * POINT\_SIZE)/DeciPointsPerInch)$$

  where the value of DeciPointsPerInch is 722.7q.

- POINT_SIZE is an unsigned integer string, giving the body size of the font, in device independent units. POINT_SIZE is not necessarily equivalent to the height of the font bounding box. POINT_SIZE is expressed in decipoints (72.27 points equal one inch).

- RESOLUTION_X field is an unsigned integer string, giving the horizontal resolution in dots per inch (dpi), for which the font was designed.

- RESOLUTION_Y field is an unsigned integer string, giving the vertical resolution in dots per inch (dpi), for which the font was designed.

- FONT_ASCENT is the extent of the character above the baseline.

- FONT_DESCENT is the extent of the character below the baseline.

- SPACING is an encoded string that gives the escapement class of the font. A font can be proportional spaced or fixed width. A fixed width allows the same width for all characters. In a fixed width font, a w will take the same space as a l. In a proportional spaced font the space each character takes, is character dependent. The encoding of SPACING is as follows : "P" standing for proportional, "M" standing for monospaced, and "C" standing for CharCell. CharCell is a monospaced font that follows the standard typewriter character cell model, that is that all characters can be modelled as "boxes" of the same width and height.

- AVERAGE_WIDTH is an unsigned integer string typographic metric value that gives the mean width of all characters in the font. For monospaced and charcell font this is the width of all characters in the font.

- CHARSET_REGISTRY and CHARSET_ENCODING are registered names identifying the registration authority that owns the specified encoding, and the coded character set as defined by that registration authority, respectively. The following CharSet names are registered for use in font names under the X Logical Font Description.

ISO8859-1	Latin alphabet No 1
ISO8859-2	Latin alphabet No 2
ISO8859-3	Latin alphabet No 3
ISO8859-4	Latin alphabet No 4
ISO8859-5	Latin/Cyrillic alphabet
ISO8859-6	Latin/Arabic alphabet
ISO8859-7	Latin/Greek alphabet
ISO8859-8	Latin/Hebrew alphabet
ISO8859-9	Latin alphabet No 5

- MIN_SPACE is an unsigned integer value that gives the recommended minimum word space value, to be used with this font.

- DEFAULT_CHAR is an unsigned integer value that gives the ASCII code of the default character to be used by the X server when an attempt to display an undefined or non existent character in the font.

Once you finish editing the defaults file, save it and then exit the editor. Fontcreator scans the defaults file to determine the new font file name. If a font file with the same name already exists, you will be asked whether to overwrite the existing font file or abort the creation of the new file.

6.5 Drawing a new character.

The first step in order to draw a new character is to click the "Edit" pushbutton. By clicking on "Edit" pushbutton a series of pop up windows appear, prompting you to enter the character to be edited, the character width, height x and y displacement. Character's width is the number of pixels, between the leftmost and the rightmost parts of the character. Character's height is the number of pixels, between the uppermost and downmost parts of the character. Character's x displacement is the horizontal distance, of the lower left corner from the origin of the character. Character's y displacement is the vertical distance, of the lower left corner from the origin of the character. Both x and y displacement can be positive or negative. For example, character "p" extents below the baseline, therefore the baseline is in its font box, and y displacement will be negative (and equal to the extent of the character below the baseline). Apostrophe "'" character on the other hand, has the baseline underneath its font box, and the y displacement will be positive. At any time you can quit the edit procedure by clicking on the "Cancel" pushbutton of any of the pop windows that appear. The program performs validity checks on the your input. If the input is invalid, or out of range, then you will be informed and asked to reenter the value. For example if the you try to edit a non printable character, enter a non integer value when an integer was expected or enter a negative width or height the program will complain. If the character to be edited already exists, before continuing the program will ask you whether to overwrite the existing character or abandon the editing operation. When you finish entering the required input, two windows appear. The bigger window is the edit window, which has a grid drawn into, where you can draw the bitmap of the character. A line drawn in the edit window shows the baseline. A smaller window at the upper right corner of the first, displays the actual character as it is being drawn. The grid displayed in the edit window consists of w columns and h rows, where w and h are the width and height of the character respectively, already entered. At the bottom of the edit window, a small message window reminds you of the character being edited. When the mouse pointer enters the edit window, it changes from being an arrow to a pencil, implying that this is drawing area. By clicking the left mouse button on grid's square, the square changes color from white to black and vice versa. By pressing the right mouse button, a dialogue box will ask you whether you wants to quit editing. Clicking on "Cancel" pushbutton returns you back to editing mode, while clicking on "Ok" pushbutton, pops up another dialogue box, asking whether to save edited character or not. You may select to save the character being edited in the font file, by clicking on "OK" pushbutton or exit editing without saving, by clicking on "Cancel".

6.6 Quitting FontCreator.

When you click on "Quit" pushbutton, the program first checks the state of the program. If there is some other operation on, the program will ask you whether to abort it or not. If no other operation is on, the program will ask you to affirm your choice. If you insist on your choice, the FontCreator will save the font file being edited and then exit.

6.7 Geting help.

To get on line help click with the mouse pointer the "Help" pushbutton. Depending on the operation you are on, the relevant help text will appear. However if the help file is missing the message "$< helpfile.hlp >$ is missing" will be displayed in the help window, where $< helpfile.hlp >$ is the name of the help file.

6.8 Compiling a font file and installing a new font.

Once you create your font file, exit FontCreator by pressing "Quit". Then convert the font file from BDF format to PCF format using bdftopcf :

```
% bdftopcf filename> filename.pcf
```

Copy the PCF file to a font directory :

```
% cp filename.pcf $HOME/fonts
```

Since you add a new font file, you should update the directory information with mkfontdir. mkfontdir searches the directory for valid font files in the following order : PCF, SNF, and BDF.

```
% mkfontdir $HOME/fonts
```

mkfontdir creates in fonts directory, a file named fonts.dir. fonts.dir first line is the number of valid font files found in the directory and the second line is the font file name followed by the XLFD font name, defined in the font file. Next, create an alias for the font named $HOME/fonts/fonts.alias. The contents of the alias file will be the alias of the font followed by the XLFD (X Logical Font Description) font name as defined in fonts.dir. Now you can test the new font file. Update your font path using xset :

```
% xset fp+ $HOME/fonts
% xset fp rehash
```

Finally you can start an xterm with the new font file :

% xterm -fn < *aliasname* >

6.9 Example.

We first created a new font file. The font family name was "greek", therefore the program created the font file with a "greek.bdf" name. We then edited some characters of the Greek alphabet, and saved them in the file. Once we left FontCreator we first converted "greek.bdf" file to PCF format :

% bdftopcf greek.bdf> greek.pcf

We then copied the PCF file to a font directory and updated the directory information :

% cp greek.pcf $HOME/fonts
% mkfontdir $HOME/fonts

mkfontdir created a file called "fonts.dir" which looked like that :

1
greek.pcf -misc-greek-bold-r-normal–24-240-75-75-p-65-iso8859-1

The first line of the file is the number of valid font files found and the second line is the name of the file, followed by the XLFD font name. We then created an alias file, in $HOME/fonts named "fonts.alias" which looked like that:

greek -misc-greek-bold-r-normal–24-240-75-75-p-65-iso8859-1

Finally we tested the new font by updating the font path and starting a new xterm :

% xset fp+ $HOME/fonts
% xset fp rehash
% xterm -fn greek

Instead of starting a new xterm we could have used xfd to display the new font:

% xfd -fn greek

Chapter 7

Appendix B : Font Utilities

7.1 Displaying fonts utilities

There are a number of utilities which can be used to help us, display and select a font. We will give a brief description of the operation and syntax of each utility.

- SHOWFONT

  showfont displays information about the specified font. Its syntax is :

  showfont: [-server servername] [-extents_only] [-l] [-m] [-L] -[M] [-unit #] [-pad #] [-bitmap_pad value] [-start first_char] [-end last_char] -fn fontname

  showfont is only available with X11R5.

- XFD

  xfd displays all the characters in a font. Its syntax is :

  xfd [-options ...] -fn font

  where options include:

  -display dpy X server to contact

  -geometry geom size and location of window

  -start num first character to show

  -box show a box around each character

  -center center each character inside its grid

-rows number number of rows in grid

-columns number number of columns in grid

- XFONTSEL

  xfontsel is a program that can be used to select easily a font, by use of its XLFD name. Wildcards (*) can be used, and in the case there are more than one fonts matching the description only the first font found is shown. xfontsel syntax is :

  xfontsel [-options ...] -fn font

  where options include:

  -display dpy X server to contact

  -geometry geom size and location of window

  -pattern fontspec font name pattern to match against

  -print print selected font name on exit

  -noscaled do not use scaled instances of fonts

  -sample string sample text to use for 1-byte fonts

  -sample16 string sample text to use for 2-byte fonts

- XLSFONTS

  xlsfonts lists all fonts available in a X server. Its syntax is :

  xlsfonts [-options] [-fn pattern] where options include: beginitemize

  -l[l[l]] give long info about each font

  -m give character min and max bounds

  -C force columns

  -1 force single column

  -u keep output unsorted

  -o use OpenFont/QueryFont instead of ListFonts

  -w width maximum width for multiple columns

  -n columns number of columns if multi column

  -display displayname X server to contact

7.2 Font conversion utilities.

There are a number of utilities that can be used to convert one font format to another. We briefly describe their use and syntax.

- BDFTOPCF

  bdftopcf converts BDF format fonts to PCF format. bdftopcf replaced in X11R5 the bdftosnf utilitie which compiled BDF font files to SNF format. Its syntax is :

  bdftopcf [-p#] [-u#] [-m] [-l] [-M] [-L] [-t] [-i] [-o pcf file] [bdf file] where # for -p is 1, 2, 4, or 8 and # for -s is 1, 2, or 4

- CONVERTFONT

  convertfont reads the font files and converts them to the specified format. Its syntax is :

  convertfont -abcdnstv files

  -a Write readable text version of file (Adobe ASCII)

  -b Write a CScript format file (the default)

  -c STR Set the comment associated with the font.

  -d STR Set the output destination directory to STR.

  -f N Set the maximum length of the output filename to N characters (default == 8)

  -n STR Set the name of the font.

  -o STR Set the output file name.

  -s STR Set the size of the font.

  -S Set the size of the font by magic.

  -t Tell briefly what is in the font.

  -tv Tell in detail what is in the font.

  -ta Tell in gory detail what is in the font

  -v Write a vfont file

  -vf Write a vfont file, forcing fixed widths

  -x Write BDF 2.1 format file (for X11)

  -M Force matrix font encoding (use carefully!).

Convertfont reads each named font and then writes it out in the specified format.The output name is derived from the input name by replacing the extension and possibly substituting a new directory part.

Chapter 8

Appendix C : Program Listing