Anti-Aliased Fonts and UTF-8 Support in OpenMotif 2.3 White Paper

Integrated Computer Solutions Incorporated

February 2006

Table of Contents

Introduction

OpenMotif 2.3 includes a major feature enhancement: the functionality of Open Motif is now extended to support UTF8 and Anti-Aliased fonts through the XFT library. With these new functionalities, OpenMotif 2.3 reasserts itself as a modern, competitive GUI toolkit.

This document provides an overview of the core technologies used in this newly implemented functionality, as well as details how they can be employed through strategies and usage examples.

Importance of UTF-8 and Anti-Aliasing

Anti-Aliasing and UTF-8 are supported by the majority of modern GUI-toolkits and have become the de facto standard for any up-to-date GUI-program.

This section provides general information on UTF-8 and Anti-Aliased fonts, as well as describes how the new release of OpenMotif 2.3 benefits from their use.

What is UTF-8?

UTF-8 (8-bit Unicode Transformation Format) is a character encoding system that can represent any universal character in the Unicode standard. For this reason, UTF-8 has become the preferred method for encoding email, web pages, and other places where characters are stored or streamed.

Before UTF-8 emerged, UNIX users had to use various ASCII extensions. Support for these encodings was incomplete and unsatisfactory. The advantage of UTF-8 encoding is that the Unicode can be used in a convenient and backwards compatible way in environments that were designed entirely around ASCII, such as UNIX. No conversion is needed for ASCII: every valid ASCII string is also a valid UTF-8 string. Recent open source operating systems such as Linux, FreeBSD, Solaris, etc. use the UTF-8 character encoding system by default.

For more information on UTF-8, please see www.unicode.org.

What is Anti-Aliasing?

Anti-Aliasing is the technique of minimizing jagged edges on graphic images using intermediate shades. Anti-Aliasing of fonts makes them much more readable and visually appealing, especially on TFT displays.

Please see Figure 1, where the font is not anti-aliased, and compare it to Figure 2, where the font is anti-aliased.

Figure 1

The letters below (see Figure 2) show how Anti-Aliasing adds gray pixels around the border between black and white, resulting in a visually smoothed outline.

Figure 2

Graphics are affected by anti-aliasing in the same way that text is.

In OpenMotif 2.3, anti-aliased fonts are supported using the client-side XFT library.

XFT Library for Anti-Aliased Fonts and UTF-8 Support

XFT is a library for displaying fonts with the X Window System. It was designed to support scalable and anti-aliased fonts.

Unlike traditional X11 font rendering, XFT does not use the X Font Server. The XFT library renders fonts within the application space. That way, the application has direct access to the font definitions and full control over rendering character glyphs.

Internally, XFT uses the fontconfig library to locate fonts and the FreeType library to render them.

XFT has become the de facto standard. It is used by the desktop environments KDE and Gnome, as well as the Firefox browser and many other applications.

OpenMotif 2.3 uses XFT to implement Anti-Aliased fonts and UTF-8 support.

Anti-Aliasing Implementation Through XFT

This section describes how the OpenMotif 2.3 GUI toolkit functionality can be extended with Anti-Aliased font support without breaking backward compatibility. The possibility of font Anti-Aliasing in existing applications without recompilation is also discussed.

Implementation Overview

Support for anti-aliased fonts is implemented via XFT. XFT introduces the concept of a “Render Table”, which replaces the use of XFont structures. This approach permits the specification of anti-aliased fonts, not only via API, but also via resource files without recompilation of the program.

Since Render Tables and renditions can be specified not only programmatically but also through resource file settings, programmers will often be able to add Anti-Aliasing support to their applications by simply adding new settings to the application’s resource files.

In some cases, code changes will be necessary to add Anti-Aliasing support. These cases generally exist in applications that use the traditional X font structure, specifically XFontStruct. For anti-aliased font support in such applications, the following possibilities should be considered:

- Using the Render Tables instead of X font structures
- Extending the applications using traditional X font structures with XFT font structures (XftFont) support

Specifying the Render Table with an anti-aliased font in the resource file or fallback resources is straightforward. For example, the following:

```
*List.renderTable: variable
*List.renderTable.variable.underlineType: SINGLE_LINE
*List.renderTable.variable.renditionForeground: Red
*List.renderTable.variable.fontName: Times
*List.renderTable.variable.fontSize: 10
*List.renderTable.variable.fontType: FONT_IS_XFT
```

would set the Render Table resource of the List widget to a Render Table with “variable” rendition, valued for those resources described in the specification.

UTF-8 Support

UTF-8 support is mostly based on the interface provided by the Xlib and XFT libraries. The implementation strategy entails updating existing code with additional cases to handle UTF-8 data. Xlib and XFT provide UTF-8 capable alternatives to almost all text handling functions.

Special cases are implemented for fonts with ISO 10646 character encoding.

ISO 10646 defines several character encoding forms for the Universal Character Set. The simplest, UCS-2, uses a single code value and allows exactly two bytes (one 16-bit word) to represent that value. UCS-2 thereby permits a binary representation of every code point in the Basic Multilingual Plane (BMP), as long as the code point represents a character. UCS-2 cannot represent code points outside the BMP. For more information on the BMP, please see http://www.unicode.org/roadmaps/bmp/.

The UTF-8 support introduced in OpenMotif 2.3 enables the transparent conversion of UTF-8 to UCS-2 that allows ISO 10646 fonts to render multi-language text as easily as single-language text. UTF-8 should be supported together with the new X11 atom UTF8_STRING.

UTF8_STRING Role for UTF8 Support in OpenMotif 2.3

Previously, text data transfers between OpenMotif-based applications and programs supporting UTF8_STRING with limited or no support of COMPOUND_TEXT could cause interoperability issues. Therefore, in this new 2.3 release, OpenMotif now supports a new UTF8_STRING atom together with UTF-8.

UTF8_STRING is a new X11 atom currently in the process of being standardized by X.Org. It seamlessly interchanges international textual data between X11 clients, and is mainly used for selection interchange (“Cut and Paste” and “Drag and Drop”).

UTF8_STRING was carefully designed to preserve compatibility with existing X11 clients. A UTF8_STRING-enabled client will use UTF8_STRING for interchange with other new clients, and fall back to COMPOUND_TEXT when interacting with older clients.

UTF-8 Usage Details

This new feature will allow UTF-8 encoded XmStrings to be specified for all resources that accept XmStrings.

To properly display non-Latin1 text (other UTF-8 characters), one of the following should be used:

- a Render Table with an ISO 10646 font
- a fontset or XFT font type

If no font is specified, OpenMotif will use a "fixed" font (Latin1 charset) by default.

In the source code, UTF-8 strings can be defined by a string in any supported charset, e.g.:

```
s = XmStringCreateLocalized("Мотиф", NULL);
```

In resource files and fallback resources, you can use UTF-8 encoded strings in the same way as strings encoded with other supported charsets, e.g.:

```
*labelString: Мотиф
```

In addition, the new property type and selection target UTF8_STRING that carries UTF-8 encoded Unicode text is supported by the toolkit in the same way as other atoms.

This is the sample "Hello World" program using UTF-8:

```
#include <Xm/XmAll.h>
String fallback[] = {
    "*fontList: -misc-fixed-medium-r-normal--14-130-75-75-c-70-iso10646-1",
/*
 * alternatively you could do the same via Render Table specification:
 *
 *      "*label.renderTable:         rt",
 *      "*rt.fontType:               FONT_IS_FONT",
 *      "*rt.fontName:
-misc-fixed-medium-r-normal--14-130-75-75-c-70-iso10646-1",
 */
        NULL
};
int main(int argc, char *argv[]) {
    Widget        toplevel;
    Arg           al[10];
    Cardinal      ac;
    XtAppContext  app;
    XmString      str;

    XtSetLanguageProc(NULL, NULL, NULL);
    toplevel = XtAppInitialize(&app, "test", NULL, 0, &argc, argv, fallback,
                               NULL, 0);
    ac = 0;
    XtSetArg(al[ac], XmNlabelType, XmSTRING); ac++;
    str = XmStringCreateLocalized("текст");
    XtSetArg(al[ac], XmNlabelString, str); ac++;
    (void)XtCreateManagedWidget("label", xmLabelWidgetClass, toplevel, al, ac);
    XmStringFree(str);
    XtRealizeWidget(toplevel);
    XtAppMainLoop(app);
}
```

Here is a sample utftest.c program:

```
#include <Xm/XmAll.h>

int main(int argc, char *argv[]) {
  Widget toplevel;
  Arg al[10];
  Cardinal ac;
  XtAppContext app;
  XmString str;

  XtSetLanguageProc(NULL, NULL, NULL);

  toplevel = XtAppInitialize(&app, "Utftest", NULL, 0, &argc, argv, NULL, NULL, 0);
  ac = 0;
  XtSetArg(al[ac], XmNlabelType, XmSTRING); ac++;
  (void)XtCreateManagedWidget("label", xmLabelWidgetClass, toplevel, NULL, 0);
  XtRealizeWidget(toplevel);
  XtAppMainLoop(app);
}
```

When run with the resources:

```
*fontList: -misc-fixed-medium-r-normal--14-130-75-75-c-70-iso10646-1
*label.labelString: text-текст
```

The program will produce the output shown in Figure 3.

Figure 3

A common mistake when first using UTF-8 strings is to forget to specify an ISO 10646 font. These resources:

```
*fontList: -misc-fixed-medium-r-normal--14-130-75-75-c-70-iso8859-1
*label.labelString: text-текст
```

Would produce output as shown in Figure 4.

Figure 4

XFT Library Support

Support of the XFT library introduced in OpenMotif 2.3 offers the ability to use client-side anti-aliased fonts from Motif applications.

This feature is available via the new Rendition Font Type named XmFONT_IS_XFT. When specifying these renditions, the following additional resources may be needed:

- xftFont- Initialized XftFont structure; it is an analog to the font resource of renditions of type FONT_IS_FONT
- fontStyle- Font style string (like "Bold Italic")
- fontFoundry- Font foundry string (like "monotype")
- fontEncoding- Font encoding string (like "iso8859-1")
- fontSize- Font size integer

Examples

The following code fragment establishes a rendition that would display a compound string in the Times font, in blue color, and underlined with a single line.

```
int n;
XmRendition Rendition;
XmStringTag RenditionTag;
XtVaGetValues(parent, XmNcolormap, &cmap, NULL);
if ( XAllocNamedColor(XtDisplay(parent), cmap, "blue", &color, &unused))
 {
   pixel_color = color.pixel;
 } else {
   pixel_color = XmUNSPECIFIED_PIXEL;
 }

n = 0;
XtSetArg( args[n], XmNrenditionForeground, pixel_color); n++;
XtSetArg( args[n], XmNfontName, "Times" ); n++;
XtSetArg( args[n], XmNfontType, XmFONT_IS_XFT ); n++;
XtSetArg( args[n], XmNunderlineType, XmSINGLE_LINE );  n++;
RenditionTag = (XmStringTag) "Rendition1";
Rendition = XmRenditionCreate( parent, RenditionTag, args, n );
```

Specifying this type of rendition from UIL is also possible, for example:

```
object
  rendition: XmRendition {
     arguments {
        XmNtag = XmFONTLIST_DEFAULT_TAG;
        XmNfontName = 'Times';
        XmNfontType = XmFONT_IS_XFT;
     };
     controls {
        XmTabList tabl;
     };
  };
```

would declare a rendition with the anti-aliased font Times.

Finally, you can apply render tables to unmodified Motif applications from the command line. The first –xrm specification applies a render table to all widgets, and the second asks for a font scaled to 42 points to be applied to that render table.

```
MotifApp -xrm '*renderTable:rt' \
             -xrm '*rt.font:-urw-urw gothic l-semibold-r-normal--42-0-0-0-p-0-iso10646-1'
```

Summary

The latest improvement of OpenMotif, version 2.3, makes it competitive with any other modern toolkit, providing many benefits in the major features enhancements of UTF_8 support and anti-aliased fonts.

UTF-8 support in this new release has the following advantages:

- Offers a modern, easy approach to render multi-language text
- Allows for a transparent transfer between OpenMotif applications and programs that support the UTF8_STRING atom
- Provides the possibility to render multi-language text as easily as single-language text

Anti-Aliased fonts provide many advantages for both the end users and the developers who implement this feature:

- Provides a more readable and pleasing experience to the end user
- Easy for the developer to implement

Support for anti-aliased fonts can be added without breaking compatibility with existing applications that do not use Render Tables. Applications that already use Render Tables could take advantage of anti-aliased fonts with simple changes to the resource file.

For more information on OpenMotif 2.3, please see the MotifZone, www.motifzone.org.