Building an international embedded GUI platform from Microwindows / FLNX

Eero Tamminen
<eero.tamminen@movial.fi>

2002-03-23

Abstract

This document describes the challenges associated with Unicode support in a windowing system, lists Microwindows shortcomings in this area, and explains what could be done about those shortcomings. It also briefly lists Microwindows alternatives, some other areas in Microwindows needing improvement, and changes in different versions of FLNX / FLTK.

Contents

Part I

Platform analysis

1 Audience

The main audience of this document is Microwindows / FLNX platform developers willing to improve internationalization support of this platform. This document should also be of interest to system / software designers and managers considering the given platform for embedded devices with *international* user interfaces and willing to invest their time in its improvement and deployment.

2 Introduction

The Microwindows / FLNX platform was selected as the target of this study as it provides a small, very configurable and portable GUI platform. The section 2.1 discusses Microwindows / FLNX features in more detail, and the section 2.2 presents some of the more notable alternatives.

The main part of this document concentrates on the *current* challenges of the Microwindows / FLNX platform for building user interfaces on embedded devices intended for *international* markets. Support for internationalization in both Microwindows and FLNX is incomplete with regard to Unicode fonts, strings, text layout, user input, etc. I list these issues in the section 4 on page 7.

Other issues should be evaluated according to your project requirements; I list these issues separately, in the section 5 on page 14.

The appendixes II, beginning on page 18, contain lists of changes and bug fixes between FLNX and different versions of FLTK. (FLNX was originally based on FLTK v1.0.7.) This information could be useful with regard to Microwindows / FLNX roadmap issues mentioned in the section 5.2.

Finally, the section 6, beginning on page 17, contains some generic notes about embedded software design.

This document concerns the situation at the start of 2002.

2.1 Why Microwindows / FLNX

Whether you need Microwindows / FLNX depends on your system requirements.

A large portion of the current embedded devices use homegrown solutions. Embedded devices include not just general-purpose devices such as PDAs, but also dedicated single-purpose devices. Open Source technology provides a lot of the software infrastructure needed for future networked embedded devices, as it did earlier for most Internet services. The wide diversity of Open Source projects

and their different goals mean that you always need to do at least *some* integration.

Microwindows / FLNX is not an all-around ideal GUI solution, but it has a very good cross-section of features required for a roll-your-own embedded platform. Evaluation against your requirements is needed, however, since this solution lacks some features that are present in other solutions.

Microwindows / FLNX is a good candidate for embedded devices when compared to Windows CE, EPOC, Qt Embedded, Gtk, and X11, for the following reasons:

Configurability. Source code for both Microwindows and FLNX is freely available, and it's easy to compile in only the features you want. Microwindows and FLNX are separate components and not tightly integrated with any single environment.

Size. Microwindows is significantly smaller than other alternatives with the same or greater functionality. Depending on configured features, the Microwindows Nano-X server binary size is 200 - 450 KB. The full FLNX toolkit is 360 KB.

Portability. Microwindows is ported to at least as many platforms as other alternatives, and it's far easier to cross-compile with Microwindows than with the alternatives. Microwindows has been ported to the Intel 16-bit and 32-bit, MIPS R4000, SH3, StrongARM, and PowerPC CPUs. Operating systems on which it has been run include Linux, Elks, eCos, RTEMS, MS-DOS, and Solaris.

License. The Microwindows source is publicly available, and its core functionality has a dual MPL / GPL license suitable for corporate development.

Microwindows can also emulate the target-device display mode on the X11 desktop computer, and FLTK (of which FLNX is a derivative) applications can be compiled for Windows and X11, which helps in developing the applications. (Similar functionality is also available for some of the alternatives, though.)

2.2 Overview of the platform alternatives

There are several alternatives for embedded user interfaces, each with pros and cons. This section lists some of the alternatives (circa end of 2001).

Windows CE If you're satisfied with what Microsoft has to offer, this is your choice for a *PDA*. Offers the best (Microsoft) office software and multimedia integration but is completely proprietary. Full Unicode support.

EPOC Designed from the bottom up for embedded devices and implemented in its own version of C++. Well-integrated and has the best energy management (battery life). Offers the usual desktop document-synchronization and viewing utilities. Proprietary. Unicode support incomplete.

Qt Embedded The most mature of the Open Source alternatives, packaged and supported. Platform is source-code compatible to Qt on Linux and Windows desktops. Offers package / application management. Desktop office software integration (calendar, address book, etc.) is in the works. Source code license is GPL (or you can negotiate one with Trolltech), but it's still hard to get your *own* changes to Qt. Completely C++ with full Unicode support.

Microwindows / FLNX Window engine is done in C and FLNX uses basic C++ features. If you want to make your own solution and size is crucial, this is your choice. You need to invest more effort for the application-framework development than in other solutions. Microwindows has dual GPL / MPL, FLNX has LGPL, and most of the existing applications have GPL licenses. Smallest of the Open Source solutions.

Gtk2 / DirectFB This combination has features most relevant for digi-TV, such as multimedia devices; not for general-purpose embedded devices. DirectFB is still in development. Gtk2 binary sizes and memory consumption are the largest of the Open Source alternatives. DirectFB is supposed to have support for multiple applications running at the same time. Gtk v2 with its Pango component has full Unicode text-output support. LGPL license, and done in C.

Gtk2 / X11 You can also run Gtk on top of X11, as Linux desktops do,[1] because of the large code base and stability of X11. X11 is the best-supported Gtk2 widget set back end. However, a large programming API, heterogeneous network capabilities, hardware probing, and dynamic component loading with the associated libraries make X11 larger than the above alternatives, both in the base binary sizes and memory footprint[2].

Some other possible Open Source alternatives lack required features and development communities. If you want to make a simple user-interface without any internationalization and advanced color-graphics capabilities, there are several other smaller alternatives.

Note: Full Unicode support mentioned above means having *generic* support for at least one Unicode encoding and supporting *bi-directional* text layout. For general consumer products, this is the state of the art and enough for western and some Asian markets, but the *real* Unicode support includes much more; see the section 4.5 on page 11.

3 Terminology

Microwindows Open Source windowing and graphics engine. Two different programming APIs, one similar to Win32 and another to X11. The Win32 API links the window engine library to the applications (single-application model). See:
`http://www.microwindows.org/`

[1] It's also possible to run X11 on top of DirectFB.

[2] This article contains some notes about downsizing X11:
`http://www.linuxdevices.com/articles/AT9006921228.html`

Nano-X Window server using the Microwindows graphics engine. Applications communicate to the window server through a socket using a client library that provides functionality and an API equivalent to the X11 API. See:
`http://home.twcny.rr.com/embedded/microwin/`

FLNX Version of FLTK v1.0.7 modified to work with Nano-X. All the larger Microwindows applications use FLNX for their user interface.

Century Software Company offering commercial support for the Microwindows / FLNX platform; they have built an embedded desktop environment on top of it called PIXIL. See:
`http://www.centurysoftware.com/`

FLTK User-interface widget / toolkit library supporting Windows and X11. See:
`http://www.fltk.org/`

Fluid FLTK user-interface builder application.

UTF-8 Multi-byte, *variable*-size string encoding in which codes 0-127 are the same as in ASCII, other codes use at least two bytes and have their 8th bit set, and strings end with a NULL byte. Therefore, all the normal C and C++ string routines work with UTF-8, although the *user-visible* string width may differ from what strlen() returns, for example. UTF-8 is becoming the text input- and output-encoding standard on the web and in Open Source because it's byte-order agnostic and backwardly compatible with older applications that can deal only with ASCII.

UCS-2 16-bit fixed-width string encoding. UCS-2 (or UCS-4) is often used inside components that have to do a great deal of Unicode string processing (such as more advanced Unicode text-layout engines, etc.), although these components may outwardly work with UTF-8. Windows supports the little endian version of the UCS-2 encoding.

UCS-4 32-bit fixed-width string encoding that can be used when 65,535 glyphs provided by UCS-2 are not enough. UCS-2 and UCS-4 are not compatible with ASCII and need their own string-manipulation functions. Glibc 'wide char' routines can deal with both UCS-2 and UCS-4, as the 'gcc' wide char (wchar_t) type is 32 bits.

Glyph A single character image in a font is called a glyph.

UTF-8, UCS-2 and UCS-4 encodings are defined in the ISO 10646 standard. For more information on Unicode on UNIX, see:
`http://www.cl.cam.ac.uk/~mgk25/unicode.html`

4 Internationalization issues

The main challenge in internationalization is adding generic-enough support for it without adding significant memory overhead and without doing it in a way that's incompatible with non-international configurations of the platform.

The following sections list issues related to Unicode support and provide some examples of how they could be solved. If you want an overview of the issue dependencies, you could first visit the "text layout" section 4.5 on page 11.

4.1 Unicode string support

Earlier languages had their own code pages and encodings. Today, Unicode is the standard, defining a unique code for each glyph in most languages.

Windows uses UCS-2 (two-byte) representations of strings; UNIX Open Source software generally uses UTF-8 (multi-byte, ASCII compatible) when communicating strings. The Microwindows graphics engine and API support both of these Unicode string representations and can convert between them.

Challenge: FLNX doesn't support UTF-8 or UCS-2 strings.

FLTK is supposed to support UTF-8 sometime in the future. A patch is available for FLTK v1.1 to fix this problem for X11, but the patch doesn't work with FLNX. The patch relies on an X11-specific font-encoding library to map the UTF-8 Unicode strings to several different X11 fonts, each font covering a different Unicode glyph range. There are also some other X11 dependencies. The patch changes FLTK to support UTF-8 strings, it includes some test code, and a bi-directional text editor is available for it. It's available from this address: `http://oksid.ch/fltk-utf/`

Required: FLNX needs to operate with UTF-8 Unicode strings, and its font operations have to tell Microwindows that the strings are in UTF-8 format. The patch mentioned above is a good place to start, as it identifies places that need fixing. When Microwindows supports Unicode fonts, the font-encoding mapping library between Microwindows and FLNX won't be necessary.

Challenge: FLNX doesn't currently support copy / paste between applications.

Required: Both Microwindows and FLTK must support copy / paste and their functionality must include a text clipboard. This has to be added to FLNX in a way that works also with Unicode strings.

4.2 Matching strings to fonts

A full 16-bit Unicode font may include more than 65,000 glyphs. In multiple sizes, bitmap fonts like this can take several megabytes. This is too much space for embedded devices.

To make the font data size-requirements smaller (on Flash and in RAM), Microwindows has to either support fonts that contain only some / several Unicode glyph *ranges*, or it has to have the ability to map a Unicode string to several different fonts containing the required glyph ranges, and load those ranges dynamically.

In the current version of Microwindows, matching strings to fonts is handled by providing separate fonts for required glyph ranges (such as Chinese) and writing a special handler to the Microwindows graphics engine for each of the 'special'

fonts. This handler can then encode and map given strings to glyphs in the font. This is not general enough, however.

Challenge: The application has to know which font contains the characters it wants to show. This makes device localization more difficult; some applications, such as browsers, can't know beforehand what characters they need to show.

There are two solutions:

1. Modify FLNX to handle mapping of Unicode strings into several different fonts. It would need to maintain mappings among Unicode index, font names, and font glyph indexes within those fonts. This is the method used in the above-mentioned FLTK patches for X11. This still isn't general enough, though.
2. Modify Microwindows to handle Unicode fonts. Because of space constraints and the size of more comprehensive Unicode fonts, it has to handle efficiently sparse glyph *ranges*. That way, only the fonts have to be localized (by deciding what Unicode glyphs to include into them). See the next section, "Unicode font support".

4.3 Unicode font support

Some BDF (bitmap) and TrueType (outline) *Unicode* fonts can be used with Microwindows. However, low- and middle-end embedded devices don't have enough CPU power for TrueType font output.

Challenge: The font data offset in the internal Microwindows font structure is an unsigned short. This limits the 'standard' Microwindows font data size to 64 KB. This isn't enough for large Unicode fonts.

Required: This structure member needs to be changed to an unsigned long. This increases the size of the old ASCII character set fonts by 200 bytes.

Challenge: A bitmap font containing *all* the glyphs defined in Unicode standard is too large for embedded use, especially when you need fonts in several different sizes and styles.

Required: Microwindows needs to support fonts with glyph ranges. That way, the same font can contain Latin characters and Chinese pictograms, for example, but skip the rest of the RAM- and Flash-consuming glyphs.

One possible solution would be to have the widths structure for the Unicode font glyphs compiled or loaded into Microwindows with the list of included glyph ranges (ranges and names for glyph range-data files) and pointers to font data for those ranges. The range data itself (offsets and glyph data) could be loaded dynamically. As the font information structure and font data can be quite large, you might want to memory-map them. The font data pointers (which are filled when actual range data is loaded) and any other data that is not read-only should be in a separate structure. Another alternative would be to use short offsets to data with the glyph ranges, but this limits single-range data size to 64 KB.

Challenge: The Nano-X client API for querying font information returns a structure that contains glyph widths in a fixed 256-byte array. Unicode fonts have more than 256 glyphs.

Required: Microwindows font handling and Nano-X API should be modified so that the font glyph width-array has a variable size[3]. The above glyph range requirement will complicate this. To save RAM, it would be better if these font metrics were shared between the Nano-X server and its clients (as XFree v4.2.1 does). This change will be needed by widgets doing text layout.

Note: If fonts are required in many different (and larger) sizes, moving from bitmap fonts to TrueType fonts and using FreeType, for example, should be seriously considered. (It's not very complicated to computationally produce pseudo bold and oblique versions of the same outline font with FreeType. However, glyph caching has to be implemented within the window engine, FreeType doesn't do that, and not doing it at all is too slow. How to do it depends on the memory usage.)

4.4 Input methods

For the user to be able to input characters in different languages, the key input events have to contain Unicode codes, there has to be a method for inputting any of those pages, and there should be a way to change the input method.

Challenge: Microwindows has to output Unicode key events; an input-method application also has to be able to do this.

Microwindows allows applications to inject events to other applications. This feature can be used by input-method applications. Microwindows key events contain a 16-bit key value and OEM scan code. The key value can be a UCS-2 Unicode key code.

Challenge: FLNX doesn't support UCS-2 key events, which Unicode input-method applications would inject to applications.

Required: FLNX should be changed to accept UCS-2 key events and to convert them to a format that FLNX text widgets and applications understand.

Challenge: Just injecting events is not enough for more advanced Asian input methods. These input methods need to be context-aware to be able to offer the user a manageable set of glyph alternatives.

Required: To know the application input context, the input-method application has to know when the context changes and be able to differentiate various contexts. This means that the input method should be signaled about the focus changes with a unique focus and application identifier. Another alternative: If the input method is modal, the context can be changed only after the input method has been terminated.

[3]The Nano-X font info package-marshaling code should be changed to use variable-length packet handling for font information. For every sizeof() for MWFONTINFO or GR_FONT_INFO, the 'widths' array size should be added (to all the functions using GrGetFontInfo() in Nano-X and engine/devfont*.c).

I also considered letting input methods filter application key events[4], but this can be a security issue and doesn't deal with focus / context changes inside an application.

Challenge: There's no application for Unicode input, or framework for switching among different input methods (applications).

Required: These have to be written, but they're a separate issue from Microwindows / FLNX.

One possibility is for the window manager to work as the input-method framework. Using a list of available input methods, it could run, stop, and in general keep track of the current input-method application positioned correctly above the window receiving the input.

Challenge: Some embedded devices have special hw keys.

An alternative that would ease testing is to map the hw keys to function-key codes for Microwindows. That way, developers and testers using the desktop X11 version of Microwindows could emulate the hw keys. It would be a kernel driver issue.

4.5 Text layout widgets

Internationalized text layout and text widgets are a big issue that can only be scratched here. This topic can be summarized as a few separate issues:

- Supporting *Unicode fonts and Unicode-encoded strings.* This issue was introduced earlier; the current methodology is sufficient to handle western languages. Notes about how to support this in a *generic* way are presented in the sections 4.1, 4.2 and 4.3.
- Supporting *different input methods.* Asian languages have a huge alphabet (tens of thousands of glyphs), and it's not practical to have the user select a character from all the possible alternatives. Handling this issue will require *context-sensitive* input methods in which the user is presented with a selection of a few glyphs appropriate to the context in which the glyphs will be used. See the section 4.4.
- Supporting *bi-directional text.* Certain languages can be written both from left to right and from right to left, and these can even be mixed. For example, Arabic text is written right to left, and numbers are written from left to right.
- Supporting *glyph composition.* In some languages, such as Thai, the characters are composed of multiple glyphs. Western languages can use this technique as well (using a German umlaut, for example), but those combinations are so few that Unicode has separate glyphs for them.

[4] In the case of X input methods used on X11, the client side sends the input values through the filter.

- Supporting *vertical text*. Chinese and Japanese are written from top to bottom. In most situations, it's passable to write from left to right, but this still leaves the issue of text input. Context-sensitive input methods are a must for usability.

Challenge: FLNX / FLTK text widgets don't support bi-directional or vertical text or glyph composition.

There are Open Source projects implementing some of this functionality, and even one Unicode text editor[5] for *FLTK v1.1*.

The most advanced of the Open Source text-layout engines is Pango[6], used in Gtk v2. It supports text bi-directionality, glyph composition, and font styles. It has a separate module for each of the language and font-engine pairs, but support for many languages is still incomplete. At the moment, the Microwindows font engine isn't supported.

Integrating this functionality into FLNX would require a great deal of work. For example, in Gtk v2, Pango is used for text output in *all* widgets, not just text-editing widgets. Implementing that type of functionality for FLNX would be a major change and increase its current size significantly. Also, the Pango language modules would need to be made to work with the Microwindows font API. Localization would mean selecting which language modules to include with the device.

4.6 Localization

Localization includes message-string translation, date format, fonts files, etc.

Neither Microwindows nor FLNX contains functionality or widgets that would be affected by localization. For example, FLNX doesn't contain special widgets for dates or have a calendar widget. Applications have to implement this functionality themselves.

Challenge: Applications need localization support (similar to locale, gettext, and wide char functions) from the underlying libraries. Later glibc versions include these features, but none of the libc replacements intended for the embedded devices have all of the features:

- Locale is a set of rules for sorting; presenting dates, currencies, numbers, etc. that affect corresponding C library functions. These rules can be changed as a whole by selecting the locale area (such as Canadian English) and then by editing individual settings. In the Glibc locale model, locale settings are normally controlled by setting certain environment variables before running the application. When the application starts, it calls the locale initialization functions. The application can override these 'automatic' settings by calling separate locale-setting functions from the application code.

[5] `http://www.oksid.ch/flwriter/index.html`
[6] `http://www.pango.org/`

- Gettext is a message catalog library used by most of the internationalized Open Source projects. There's a separate message-catalog file for each language that the application supports. Which message the catalog uses depends on the application environment language settings.
- Glibc wide-char functions can manipulate 16-bit and 32-bit strings, which are often used internally for Unicode text-layout processing, although the outside application would work with UTF-8.

Challenge: Glibc standards-compliant internationalization support is so complex and interdependent that it can't be pruned to a reasonable size for embedded devices. This is mostly the case with rest of the Glibc as well. Embedded devices need something that is just 'good enough'.

Gettext is easy to use; you just surround the user-visible texts in the code with the gettext call and link the application with the gettext library. There are also GUI tools[7] to help translators deal with message catalogs.

Challenge: The gettext system incurs unnecessary overhead because it stores the message index as text in both the application code and the message catalog-data file.

4.7 Conclusions

Adding internationalization through Unicode support is not a major change to Microwindows and shouldn't increase its size significantly. Existing applications might need to make some changes to their font handling, but these should be very minor and significantly increase how easy it will be to internationalize them.

Use of Unicode fonts and encoding would isolate (generalize) part of localization to the selection of what glyphs to include into fonts. No changes are needed to the applications. Supporting sparse Unicode font data is an *important* part of this process, as full Unicode fonts would take a large part of the space needed on the device[8].

Due to the Microwindows design, input methods don't affect Microwindows or FLNX, as long as both of them support Unicode. Input methods and a framework for changing them can be developed separately.

FLNX changes are more controversial. UTF-8 seems an obvious choice for Unicode encoding support. Modifying FLNX to support it just involves manual work in integrating existing FLTK UTF-8 patch code to FLNX code in a Microwindows-compatible way. This shouldn't increase the library size significantly.

Adding Unicode text layout isn't that easy, though. Each language group has separate text-composing rules, and they map to different Unicode font glyph ranges. It all adds significantly to the library size, and it's not clear what layout functionality is actually required. I would suggest that this decision wait until Open Source text-layout engines mature and are in more general use.

[7] Such as KBabel and Gtranslator.

[8] Look at how the X Render extension does things: Clients handle fonts and font metrics; the X11 server just draws (anti-aliased) glyphs with the Render extension.

5 Other issues

This section lists some other issues and characteristics of the Microwindows / FLNX platform. Their significance depends on your requirements.

5.1 Microwindows

These notes concern Microwindows version 0.89pre.

For efficiency, image loading on Microwindows is done completely on the server side. The client gives Microwindows the image path and gets back an ID to the image drawable.

Challenge: The client may need to do image operations that are not provided by Microwindows graphics operations. Microwindows / FLNX doesn't provide a system for this.

Fortunately, Microwindows offers all the graphics operations required by normal applications.

Challenge: Microwindows doesn't provide an efficient and centralized method for dealing with multiple instances of images such as toolbar icons.

This isn't an issue on most embedded environments, but more-graphical user interfaces may utilize the same images multiple times. In this case, you could profit from a centralized image-loading and reference-counting facility[9]. This would be a client-side library, but it might need some help from Microwindows.

Challenge: Automatic switching between portrait and landscape screen mode doesn't work properly yet; sometimes it crashes.

You should disable this feature for the 0.89pre version of Microwindows.

Challenge: Moving the FLNX window so that part of it is outside the screen causes a lot of flickering redraws. Also, the FLNX sliders flicker noticeably when used.

I'm not sure whether this is due to an FLNX application problem (for example, not using Fl_Double_Window) or inefficient Microwindows blitting routines.

5.2 FLNX

These notes concern FLNX version 0.16.

5.2.1 FLNX roadmap

FLNX is a fork of an older FLTK version. Currently, FLNX contains two widgets that are not in the main FLTK library, and Century Software has some additional widgets that are not ported to FLTK.

[9] Such a facility is build into imlib, which is used by many Gtk v1 applications. With it, you can set the amount of used cache memory.

Challenge: It's not clear whether FLNX is intended to be separated from FLTK completely or be more compatible with it in the future. If FLNX and FLTK are completely separated, the user won't be able to code applications for both Microwindows / FLNX and FLTK / X11 / Windows platforms.

Required: Century Software should clarify its intentions for FLNX so that its users can help Century Software to accomplish those objectives, and plan their own projects accordingly.

Challenge: FLNX is a Microwindows port of FLTK version that has some known bugs. See the section B on page 21.

Required: Bug fixes from the latest FLTK 1.0.x version (v1.0.11) should be ported to FLNX.

Challenge: The FLTK Microwindows port is implemented by having #ifdefs all over the FLTK v1.0.7 X11 code. This complicates the porting of fixes and widgets between FLNX and FLTK.

Required: FLNX code should be cleaned to allow easier importing of fixes and additions from mainstream FLTK versions. This could be done by emulating X11 calls with Nano-X calls, as their functionality is similar. Nano-X includes preliminary X11 function-call conversion macros; these could be developed further.

The appendixes, starting from the section B on page 21, list the FLTK changes after FLTK v1.0.7 (on which FLNX is based).

If FLTK compatibility is not a goal, redundant X11 and Windows code should be purged from the FLNX code.

Challenge: Fluid, the FLTK User Interface Builder, doesn't support the new FLNX widgets. Having UI builder support for these widgets would help in development of more complex application interfaces.

Required: Adding support for the new widgets to Fluid is easier than constructing a new UI builder from scratch. Later versions of FLTK have a Fluid version whose user interface is cleaner and somewhat easier to use than the version that comes with FLNX. Converting a later version of Fluid to support FLNX could also be considered.

Challenge: Some of the original FLTK widgets lacked a few allocation failure checks and FLNX has inherited this problem.

Required: These should be fixed.

5.2.2 Main FLNX differences from other widget sets

FLNX is based on FLTK, and has inherited its attributes:

- Minimalistic approach in API; uses only basic C++ features. FLTK doesn't use namespaces, exceptions, RTTI, STL, templates, or iostreams. Using these facilities would increase the C++ application and library sizes. Earlier use of advanced C++ features was a portability issue.

- Uses callbacks instead of the signal/slot methods used in Qt and Gtk for delivering events to widgets.
- Designed to be modular, so it doesn't have many interdependencies. You can drop unwanted parts without worrying too much about unresolved dependencies.
- Unlike many other current widget sets, FLTK doesn't use containers for positioning child widgets. Unlike Gtk widgets, for example, which automatically size themselves according to their contents, with FLTK you set the exact widget positions in pixels. When windows are resizable and resized, containers keep the child widgets at the same offset from container borders. Another use for containers is grouping radio buttons.

  Con: You need to leave enough space in widgets for all localized strings so that strings are not drawn over widget borders or clipped. You also have to test that each translation fits to the widgets.

  Pro: You can position widgets accurately within the screen. On devices where screen size is constrained and you want full control over the look of the GUI, this feature is important.

5.3 Licenses

5.3.1 Microwindows

Unlike some other other Open Source organizations, Microwindows and its support packages utilize code from outside the project and Century Software*without* written copyright transfer. This eases project participation but makes the copyright situation somewhat unclear.

Challenge: Evaluation of the legal risks associated with the software could be easier.

Required:

1. The Microwindows package should list files with licenses that differ from the default dual MPL / GPL license used in the main part of the code, even if the other licenses would be compatible.
2. The Microwindows package should have a file listing the sources (owner, etc.) of the source code being used.

Gregory Haerr, CEO of Century Software Inc., owns the copyright to Microwindows as a whole. Small portions of the code have other copyright owners.

5.3.2 FLNX

Like FLTK, the FLNX library is under the LGPL license. FLTK doesn't contain code from other sources with other licenses, but as there's no organization behind it, code contributions are applied without copyright transfers.

The FLTK copyright is owned by its original author, Bill Spitzak, and other contributors.

6 Notes

6.1 Energy management

Energy management (battery life) gives some general requirements for an embedded platform. These fit into two categories:

1. CPU-intensive jobs should be minimized and *continuous* CPU usage should be eliminated completely. This means that any polling operations should be changed to use status change signaling / waiting.
2. Applications shouldn't draw anything when the screen is off. Screen state has to be communicated to the system so that applications can suspend their screen activity when the screen is off *or* the framework should provide a method to suspend all GUI applications when the screen is off.

The latter option suggests the following architectural constraints:

- Tasks that cannot be interrupted should not be part of GUI applications. Such tasks might include OS event-notice sounds, for example, or downloading of files.
- Tasks that are *not* suspended should not be affected by the suspension of other tasks. Either those tasks must be independent (that is, they don't communicate with suspendable tasks or communication is done asynchronously), or all intercommunicating tasks must be suspended at the same time.

It should also be noted that refreshing RAM consumes energy. The more RAM you have, the faster your battery drains. Therefore, you have to find a good balance between your CPU power and RAM (caching) needs.

Part II

Appendixes

A FLTK 1.0.7 -> FLNX 0.16

FLNX is a source-compatible Microwindows version of FLTK v1.0.7. FLNX v0.16 doesn't have all the functionality of the FLTK v1.0.7 library, but it has two new classes, *Fl_Animator* and *Fl_Window_Resize*.

This section lists how *FLNX* differs from *FLTK v1.0.7*.

A.1 Headers

New headers:

Fl_Animator.H XPM animations.

Fl_Window_Resize.H

n_x.h Added for Nano-X.

Changed headers:

Fl_Menu_Button.H popup(int, int) method changed.

Fl_Slider.H slider_hor_lines() and slider_ver_lines() methods changed.

Fl_Window.H resize_notify() method changed (virtual).

x.H Includes n_x.h if NANO_X is defined.

A.2 Code files

New files:

Fl_Animator.cxx

Changed files:

Fl_x.cxx HUGE load of additions and changes.

Fl.cxx Calls Nano-X graphics instead of X, expose=0.

Fl_Bitmap.cxx Has bitmap decoding code.

Fl_Double_Window.cxx Possibly redundant use_xdbe checks.

Fl_Font.H A couple of items for Nano-X.

Fl_Group.cxx Redraw change (1 line).

Fl_Input.cxx Nano-X modifier and compose code.

Fl_Menu.cxx printf()s and FL_DRAG coordinate change.

Fl_Menu_Button.cxx popup(int, int) method change.

Fl_Menu_Window.cxx mw_parent=0 (1 line).

Fl_Pixmap.cxx Nano-X GC region handling added.

Fl_Scrollbar.cxx Timeout stuff commented out.

Fl_Window.cxx Dummy resize_notify() method.

Fl_Window_iconize.cxx Calls Nano-X graphics instead of X (a few lines).

Fl_arg.cxx X geometry and property argument-handling commented out.

Fl_cutpaste.cxx X selection handling commented out.

Fl_get_key.cxx X keycode mapping commented out.

Fl_get_system_colors.cxx X color database-handling commented out.

Fl_grab.cxx X pointer grabbing commented out.

Fl_own_colormap.cxx X colormap allocation commented out.

Fl_visual.cxx Method outputting debug.

fl_arci.cxx Changes for Nano-X drawing.

fl_ask.cxx Comment XBell() out.

fl_color.cxx Comment out some color-handling code and replace X calls with Nano-X ones.

fl_cursor.cxx Cursor bitmap and replace X calls with Nano-X calls.

fl_draw_image.cxx Comment out XImage stuff and add Nano-X code.

fl_draw_pixmap.cxx Add color-code parsing.

fl_font.cxx HUGE load of additions and changes.

fl_overlay.cxx Nano-X function instead of X functions with temp GC.

fl_rect.cxx HUGE load of changing how Nano-X functions work instead of X.

fl_scroll_area.cxx Minor Nano-X changes.

fl_set_font.cxx Don't use XFreeFontNames() (1 line).

fl_set_fonts.cxx Comment out X function calls.

fl_shortcut.cxx X KeySym stuff replaced with dummy.

fl_vertex.cxx X calls replaced with Nano-X calls.

Test code changes:

checkers.cxx for some reason BLACK / WHITE defines replaced with CBLACK / CWHITE.

color_chooser.cxx X visual handling commented out.

image.cxx X visual handling commented out.

list_visuals.cxx doesn't work with Nano-X.

resize.cxx callback names within code changed.

A.3 Notes

- Some debug printfs have been added to the code.
- Cut-and-paste support hasn't been ported to Microwindows.
- Either separate functions / files should have been written for Microwindows or all required X windows should have alternative / dummy macros.
- #include <stdio.h> added onto many files (could have been done in n_x.h).
- USE_COLORMAP should be configured out, not commented out from the source.

B FLTK 1.0.7 -> FLTK 1.0.11

Source-compatible versions.

Bug fixes:

- Many fixes to FLTK (e.g., in tab, menu, and browser widgets; and key handling, scrolling, and selection; plus a few seg-faults).
- Fixes to Fluid code generation.
- fl_set_fonts() reuses fonts (-> less memory).
- Better timeout and wait() handling (-> faster).
- FLTK palette handling (color cube code) doesn't hog all the palette entries on 8-bit displays.
- Compilation error and warning fixes and other code cleanups.
- Better Windows support.
- Documentation updates.

Changes:

- i18n support (gettext/catgets) for strings added to Fluid.
- All handle() methods are now public and draw() methods are protected.
- Modal and visibility stuff behavior changes.
- New timeout, idle, and line_style functions.
- Dynamic libraries built by default.

C FLTK 1.0.11 -> FLTK 1.1.0

FLTK v1.1.0 will be released within a few months; it's currently in late beta stage (including the functionality below). It's nearly source-compatible to FLTK v1.0.

Changes:

- Improved focus handling. Focus is indicated with a border, and other widgets in addition to text input can get the focus.
- Support for GUI multithreading (wait and set-type locking with lock(), unlock(), and awake() methods).
- Drag-and-drop event types and functionality.
- New File_Chooser, File_Icon, File_Browser, Check_Browser, Help_Dialog, Help_View (html), Progress, Text_Buffer, Text_Display, Text_Editor, Tooltip, and Wizard widgets.
- Updated and improved widget looks and functionality.
- Improvements to the Fluid user interface.
- Remade image support with shared, tiled, and image type classes.
- Fl_Pack container can do some limited autopositioning.
- Widgets can have a tooltip string.
- Fl_Window has an override() method.
- Mouse-wheel and separate key up and down events.
- Fl_Tabs now uses boxtypes to draw the tabs. (It was the only widget that didn't use boxtypes for drawing; changing the widget boxtype changes how the widget looks.)
- Fl_Region used instead of Region in x.H for portability.
- FLTK Forms support and OpenGL interface are compiled into separate libraries.
- Building creates 'fltk-config' script for FLTK application configuration.
- Version and event number (ID) functions.
- Support for anti-aliased text added for X11.
- Mac support.

C.1 Notes

A separate patch adds UTF-8 and Unicode font mapping to FLTK 1.1 using the Xutf-8 library. The Xutf-8 library size depends on the number of font encodings compiled into it. It doesn't compress the encodings in any way.

D FLTK 1.1 -> 2.0 CVS version

FLTK v2 is a development version that is source-incompatible with FLTK v1.x.

Changes:

- Widget coordinates are relative to the parent instead of relative to the window.
- The Fl_Layout container widget provides Gtk/Qt/Motif-style autolayout for its child widgets (with the above relative coordinate change).
- Some methods have changed. For example, resize() doesn't change layout anymore; there's a new method for that called layout().
- Menus and browser are constructed differently. Menus now work more like widget groups, and there are new menu widgets to support this feature. This helps in dynamic menu creation.
- Widget theming with plugin support (adds libdl dependency).
- Include files have different names (FL/*.H -> fltk/*.h).
- The default widget look is changed to be more like Windows 98.
- Image loading and Glut interfaces are compiled into separate libraries.

FLTK v2 will have all the features that are in v1.1 when it's released. The following FLTK v1.1 features are missing from the current FLTK v2 CVS version:

- New Mac and image support and a few of the widgets added to the latest FLTK v1.1.
- Fluid doesn't currently have gettext() support. (FLTK v2 development branched before i18n support was added to v1.x.)

D.1 Notes

Current FLTK v2 CVS version includes theming support[10] and plugins. This means that the default foreground and background colors, box type, and fonts can be changed at runtime. There's a preliminary KDE plugin, for example, but the plugins will not be part of the final FLTK v2. Instead, they'll be offered as a separate package or part of the desktop environments. (For FLTK, these are 'Flek', used in the Agenda VR3 device, and 'Equinox', which is intended for the desktop.)

[10] For your own widget (sub)classes, note that theming works as long as you use the drawing functions provided by the widget set (color(), box(), etc.).

E Trademarks

Microsoft and Windows are registered trademarks of Microsoft Corporation. UNIX is a registered trademark of the X/Open Group, Inc.

F Document distribution

G More documents

You can find other documents of this type from:
`http://www.movial.fi/`

H Change log

2001-12-11 Initial version.

2002-01-04 Proofreading and minor fixes. Added a note about context-aware input methods.

2002-01-11 Added possible solutions, comparison to alternative platforms, and a list of FLNX v0.18 differences.

2002-02-01 Improvements to all sections according to a peer review. Portions concerning text layout and context-sensitive input methods were completely rewritten. Improvements to the logical layout of the document.

2002-03-23 Corrected the list of FLTK changes and updated FLTK v1.1.0 features according to Mike Sweet's feedback.

2002-10-14 Prepared the document for public release.

2003-04-14 Folded feedback from a professional editor into the document. The factual information in the document corresponds to the situation at the start of 2002, but that should still be valid.