User's Guide to the GNUstep GUI Library

Version 0.9.3

A. Fedor (fedor@gnu.org), N. Pero (n.pero@mi.flashnet.it)

This manual documents some configuration and installation issues with the GNUstep GUI Library and also differences between the GUI Library and libraries that implement the OpenStep AppKit specification and the MacOS-X AppKit implementation.

1 Overview

The GNUstep GUI Library is a library of objects useful for writing graphical applications. For example, it includes classes for drawing and manipulating graphics objects on the screen: windows, menus, buttons, sliders, text fields, and events. There are also many peripheral classes that offer operating-system-independent interfaces to images, cursors, colors, fonts, pasteboards, printing. There are also workspace support classes such as data links, open/save panels, context-dependent help, spell checking.

It provides functionality that aims to implement the '`AppKit`' portion of the OpenStep standard. However the implementation has been written to take advantage of GNUstep enhancements wherever possible.

The GNUstep GUI Library is divided into a front and back-end. The front-end contains the majority of implementation, but leaves out the low-level drawing and event code. Different back-ends will make GNUstep available on various platforms. The default GNU back-end currently runs on top of the X Window System and uses only Xlib calls for graphics. Another backend uses a Display Postscript Server for graphics. Much work will be saved by this clean separation between front and back-end, because it allows different platforms to share the large amount of front-end code. Documentation for how the individual backends work is coverered in a separate document.

2 Implementation Details

Following are some implementation details of the GUI library. These will mostly be of interest to developers of the GUI library itself.

2.1 Drag and Drop

The drag types info for each view is kept in a global map table (protected by locks) and can be accessed by the backend library using the function -

```
NSArray *GSGetDragTypes(Nsview *aView);
```

Drag type information for each window (a union of the drag type info for all the views in the window) is maintained in the graphics context. The backend can get this information (as a counted set) using -

```
- (NSCountedSet*) _dragTypesForWindow: (int)winNum;
```

Whenever a DnD aware view is added to, or removed from a window, the type information for that view is added to/removed from the type information for the window, altering the counted set. If the alteration results in a change in the types for the window, the method making the change returns YES.

```
- (BOOL) _addDragTypes: (NSArray*)types toWindow: (int)winNum;
- (BOOL) _removeDragTypes: (NSArray*)types fromWindow: (int)winNum;
```

The backend library should therefore override these methods and call 'super' to handle the update. If the call to the super method returns YES, the backend should make any changes as appropriate (in the case of the xdnd protocol this means altering the XdndAware property of the X window).

You will notice that these methods use the integer window number rather than the NSWindow object - this is for the convenience of the backend library which should (eventually) use window numbers for everything

2.2 NSWorkspace

Here is (I think) the current state of the code (largely untested) -

The make_services tool examines all applications (anything with a .app, .debug, or .profile suffix) in the system, local, and user Apps Directories.

In addition to the cache of services information, it builds a cache of information about known applications (including information about file types they handle).

NSWorkspace reads the cache and uses it to determine which application to use to open a document and which icon to use to represent that document.

The NSWorkspace API has been extended to provide methods for finding/setting the preferred icon/application for a particular file type. NSWorkspace will use the 'best' icon/application available.

To determine the executable to launch, if there was an Info-gnustep.plist/Info.plist in the app wrapper and it had an NSExecutable field - use that name. Otherwise, try to use the name of the app - eg. foo.app/foo The executable is launched by NSTask, which handles the addition of machine/os/library path components as necessary.

To determine the icon for a file, use the value from the cache of icons for the file extension, or use an 'unknown' icon.

To determine the icon for a folder, if the folder has a '.app', '.debug' or '.profile' extension - examine the Info.plist file for an 'NSIcon' value and try to use that. If there is no value specified - try foo.app/foo.tiff' or 'foo.app/.dir.tiff'

If the folder was not an application wrapper, just try the .dir.tiff file.

If no icon was available, use a default folder icon or a special icon for the root directory.

The information about what file types an app can handle needs to be in the MacOS-X format in the Info-gnustep.plist/Info.plist for the app - see `http://developer.apple.com/techpubs/macosxserver/System/Documentation/Developer/YellowBox/ReleaseNotes/InfoPlist.html`.

In the NSTypes fields, I used NSIcon (the icon to use for the type) NSUnixExtensions (a list of file extensions corresponding to the type) and NSRole (what the app can do with documents of this type). In the AppList cache, I generate a dictionary, keyed by file extension, whose values are the dictionaries containing the NSTypes dictionaries of each of the apps that handle the extension.

I tested the code briefly with the FileViewer app, and it seemed to provide the icons as expected.

With this model the software doesn't need to monitor loads of different files, just register to recieve notifications when the defaults database changes, and check an appropriate default value. At present, there are four hidden files used by the software:

'`~/GNUstep/Services/.GNUstepAppList`'
: Cached information about applications and file extensions.

'`~/GNUstep/Services/.GNUstepExtPrefs`'
: User preferences for which apps/icons should be used for each file extension.

'`~/GNUstep/Services/.GNUstepServices`'
: Cache of services provides by apps and services daemons

'`~/GNUstep/Services/.GNUstepDisabled`'
: User settings to determine which services should not appear in the services menu.

Each of these is a serialized property list.

Almost forgot - Need to modify NSApplication to understand '-GSOpenFile ...' as an instruction to open the specified file on launching. Need to modify NSWorkspace to supply the appropriate arguments when launching a task rather than using the existing mechanism of using DO to request that the app opens the file. When these changes are made, we can turn any program into a pseudo-GNUstep app by creating the appropriate app wrapper. An app wrapper then need only contain a shell-script that understands the -GSOpenFile argument and uses it to start the program - though provision of a GNUstep-info.plist and various icons would obviously make things prettier.

For instance - you could set up xv.app to contain a shellscript 'xv' that would start the real xv binary passing it a file to open if the -GSOpenFile argument was given. The Info-gnustep.plist file could look like this:

```
{
  NSExecutable = "xv";
  NSIcon = "xv.tiff";
  NSTypes = (
    {
      NSIcon = "tiff.tiff";
      NSUnixExtensions = ( tiff, tif );
    },
    {
      NSIcon = "xbm.tiff";
      NSUnixExtensions = ( xbm );
    }
  );
}
```

3 Contributing

Contributing code is not difficult. Here are some general guidelines:

- FSF must maintain the right to accept or reject potential contributions. Generally, the only reasons for rejecting contributions are cases where they duplicate existing or nearly-released code, contain unremovable specific machine dependencies, or are somehow incompatible with the rest of the library.
- Acceptance of contributions means that the code is accepted for adaptation into libgnustep-gui. FSF must reserve the right to make various editorial changes in code. Very often, this merely entails formatting, maintenance of various conventions, etc. Contributors are always given authorship credit and shown the final version for approval.
- Contributors must assign their copyright to FSF via a form sent out upon acceptance. Assigning copyright to FSF ensures that the code may be freely distributed.
- Assistance in providing documentation, test files, and debugging support is strongly encouraged.

Extensions, comments, and suggested modifications of existing libgnustep-gui features are also very welcome.

Concept Index

(Index is nonexistent)

Short Contents

1 Overview 3
2 Implementation Details 5
3 Contributing 9
Concept Index 11

Table of Contents

1 Overview **3**

2 Implementation Details **5**

2.1 Drag and Drop 5

2.2 NSWorkspace 5

3 Contributing **9**

Concept Index **11**