

GTK+/GLib のファイル名エンコーディング

岩本一樹
iwm@maid.org

概要

GTK+およびGLibが扱う文字列はUTF-8であるが、GTK+とGLibの一部のAPIはUTF-8以外の文字列を扱う。また一部のAPIは暗にUTF-8を他の文字符号化方式に変換する。本論文ではGTK+とGLibのAPIの文字列の文字符号化方式(Character Encoding Scheme)を検証し、file name encodingを中心にGTK+およびGLibでプログラムを組む上で注意すべき点をまとめる。

1.GTK+/GLib のファイル名の取り扱い

1-1.ファイルやパスを表す文字列

ほとんどのAPIでは文字列はUTF-8であるが、ファイルやパスを表す文字列はUTF-8以外の場合がある。GLib Reference ManualのCharacter Set Conversionではそれをfile name encodingと言っている。UTF-8とfile name encodingの関係はGLib Reference ManualのCharacter Set Conversionで説明されている。openやstat、readdirのような元々GTK+やGLibと関係ないAPIがUTF-8ではないのは言うまでもないことである。しかしGTK+やGLibのAPIであっても、主に前述のopenやstat、readdirのようなAPIに使われるファイルやパスを扱うGTK+やGLibのAPIはUTF-8以外の時がある。

例えばGtkFileChooserで選択されたファイル名はgtk_file_chooser_get_filenameで取得できるが、このAPIが返す文字列はfile name encodingである。またGtkFileChooserにファイル名を設定するgtk_file_chooser_set_filenameの引数の文字列はfile name encodingである。

実際のfile name encodingの文字符号化方式はGLib-2.4以降は環境変数G_FILENAME_ENCODINGで指定される。GLib-2.6以降ならば環境変数G_FILENAME_ENCODINGは複数の文字符号化方式を「,」で区切って指定できる(GLib-2.4であっても「,」で区切る記述は認められるが、先頭の文字符号化方式のみ有効となる)。

例

```
G_FILENAME_ENCODING=ISO-8859-1
```

例

```
G_FILENAME_ENCODING=EUC-JP,CP932
```

文字符号化方式に@localeを指定した時にはそれはロケールの文字符号化方式になる。ロケールの文字符号化方式とは、優先順位が高い順にnl_langinfoの返回值、setlocaleの返回值、環境変数のLC_ALL、LC_CTYPE、LANGの値で示されるいずれか1つの文字符号化方式のことである。Microsoft WindowsではGetACPの返回值が元になる。

例

```
G_FILENAME_ENCODING=@locale,CP932
```

もし環境変数G_FILENAME_ENCODINGが定義されていないか、GLib-2.4より前で環境変数G_BROKEN_FILENAMESが定義されているならば、file name encodingの実体はg_get_charsetの返回值になる。すなわち現在のロケールの文字符号化方式になる。

環境変数G_FILENAME_ENCODINGとG_BROKEN_FILENAMESの両方が定義されてい

い時には、file name encoding の実体は UTF-8 になる。

バージョン	G_FILENAME_ENCODING	G_BROKEN_FILENAMES	文字符号化方式
2.2 以前	無視される	あり	ロケールの文字符号化方式
		なし	UTF-8
2.4	あり	無視される	G_FILENAME_ENCODING の文字符号化方式(単一)
	なし	あり	ロケールの文字符号化方式
			なし
2.6 以降	あり	無視される	G_FILENAME_ENCODING の文字符号化方式(複数可)
		あり	ロケールの文字符号化方式
	なし	なし	UTF-8

なお Microsoft Windows ではどのバージョンも環境変数 G_FILENAME_ENCODING と G_BROKEN_FILENAMES には対応しておらず、これらは無視される。Microsoft Windows における file name encoding の実装は後に述べる。

1-2.file name encoding の変換

これらの変換のために GLib の API には g_filename_to_utf8 と g_filename_from_utf8、GLib-2.6 以降ならば g_filename_display_name、g_filename_display_basename がある。

g_filename_display_basename は引数のファイル名のディレクトリを取り除いた名前部分だけを UTF-8 に変換して返す。

g_filename_to_utf8 と g_filename_display_name の違いは変換を試みる文字符号化方式にある。GLib-2.6 以降には g_get_filename_charsets という API があり、これは file name encoding の実際の文字符号化方式を返す。もし環境変数 G_FILENAME_ENCODING で複数の文字符号化方式を指定していたならば、この API は複数の文字符号化方式を返す。

g_filename_to_utf8 は g_get_filename_charsets が複数の文字符号化方式を返しても最初の文字符号化方式からのみ UTF-8 への変換を試みる。それに対して g_filename_display_name は複数の文字符号化方式が返された時には順番に変換が成功するまで変換を試みる。ゆえに g_filename_display_name の方が UTF-8 に変換できる可能性が高いと言える。しかし g_filename_display_name で UTF-8 に変換されたファイル名は、どの文字符号化方式で変換されたのかわからないため file name encoding に戻す変換はできない。g_filename_to_utf8 で変換されたファイル名は g_filename_from_utf8 で UTF-8 から file name encoding への変換ができる。しかし g_filename_to_utf8 と g_filename_from_utf8 を使った変換では元の同じ文字に変換できるとい保証はない。次節で詳しく述べる。

1-3.libiconv のマッピング問題

g_filename_to_utf8 で file name encoding の文字列を UTF-8 に変換してから UTF-8 の文字列を g_filename_from_utf8 で file name encoding の文字列に戻した時に、最初の文字列と同じ文字列に戻ることは保証されない。g_filename_to_utf8 と g_filename_from_utf8 の動作は iconv に依存する。例えば iconv では CP932→UTF-8→CP932 で変換を行った場合に元の文字に戻らない場合がある。

例

≡:87 90 → E2 89 92 → 81 E0

徹:ED 61 → E5 83 98 → FA 7D

兌:ED 62 → E5 85 8A → FA 7E

黒:EE EC → E9 BB 91 → FC 4B

:::FA 5B → E2 88 B5 → 81 E6

1-4.ファイル名はプログラム内部で file name encoding のまま保持する

1つのプログラムで UTF-8 と file name encoding の2種類の文字符号化方式が混在するのは面倒なので、プログラムは `gtk_file_chooser_get_filename` や `g_dir_read_name` で文字列を取得したならば、直ちに UTF-8 に変換して file name encoding の文字列をその直後に破棄したくなるかもしれない。file name encoding が必要ならば、そのときに UTF-8 から file name encoding に変換して、不要になったらすぐに file name encoding の文字列を破棄すればプログラムの中のすべての文字列は UTF-8 になる。このアイデアはシンプルで良いように見えるが、問題を含んでいる。file name encoding と UTF-8 の変換が失敗した時や前節の `iconv` のマッピング問題が生じた時、プログラムが正常に動作しなくなる。

file name encoding を `g_filename_display_name` や `g_filename_to_utf8` で UTF-8 に変換できなかったとしても、そのファイル名が間違っているというわけではない。変換は失敗するかもしれないが、だからといって `gtk_file_chooser_get_filename` や `g_dir_read_name` が返した文字列を `open` や `stat` のような API で使えないわけではない。プログラム内部では file name encoding のファイル名を保持し続け、`open` や `stat` のような API を呼び出す時には元々の file name encoding のファイル名を使うようにする。もし変換ができなかった時には、例えばファイル名の代わりに「Unknown file name」のような文字列を表示する。file name encoding と UTF-8 の変換に失敗する原因は

- 1.環境変数 `G_FILENAME_ENCODING` と `G_BROKEN_FILENAMES` が正しくない
- 2.ファイル名の文字符号化方式が UTF-8 との変換に対応していない
- 3.対応する文字がない

が考えられる。

1-5.設定ファイルに保存

プログラムを利用するユーザが開いたファイルや入力した文字列の履歴をファイルに保存し、次に起動した時にもその履歴が生かせるようにするのは珍しいことではない。それは GLib の `Key-value file parser` を使って実装されるかもしれないし、それ以外の方法で実装されるかもしれない。いずれにしても、設定をテキストファイルに保存するならば UTF-8 と file name encoding の問題が生じる。

ユーザが入力する文字は UTF-8 であるが、ファイル名は file name encoding である。これらを同じファイルに書き出せば、1つの設定ファイルに異なる文字符号化方式が混在することになる。設定ファイルをエディタで開く場合、UTF-8 のテキストファイルとして読み込めばファイル名は文字化けするし、file name encoding のテキストファイルとして読み込めばファイル名以外の文字は文字化けする。

解決方法の1つとして UTF-8 または file name encoding のどちらかの文字列を `g_strescape` により7ビットの範囲に収めてしまう方法が考えられる。この場合はテキストエディタで矛盾なく編集できるが、可読性が低下する。

もう1つは file name encoding の文字列を UTF-8 に変換して保存する方法である。この場合に

は設定ファイルを UTF-8 のテキストファイルとして開くことができるが、前述の問題が発生する。
設定ファイルにファイル名を保存する場合には、下記の 3 つから選択することになる。

1. 文字列を `g_strescape/g_strcompress` により変換する
2. file name encoding を UTF-8 に変換する
3. テキストファイルとしての編集を諦める

2. Microsoft Windows のファイル名

2-1. Microsoft Windows のファイル名実装

Microsoft Windows の API は文字列を ANSI コードページで扱うものと、UNICODE で扱うものの両方が存在する。ANSI コードページでは 1 文字は 1 バイトまたは 2 バイトになる。日本語版 Microsoft Windows ならば ANSI コードページは CP932 になる。UNICODE と称するものは実際には UTF-16LE に似たもののことである(文字符号化方式を UNICODE と称するのは不適切かもしれないが、Microsoft が発行する文書で UNICODE と書かれているので、以下 UNICODE と記述する)。例えば `CreateFile` という API は実際には `CreateFileA` と `CreateFileW` がある。前者はファイル名を ANSI コードページの文字列で扱うのに対し、後者は UNICODE になる。多くの場合、ソースコードレベルでは `CreateFileA` や `CreateFileW` を直接扱うことはなく `CreateFile` と記述する。ヘッダファイルには UNICODE という名前のマクロが定義されているか否かで、`CreateFile` は `CreateFileA` または `CreateFileW` になる。文字列を表す型は `TCHAR` や `LPTSTR` であり、これも UNICODE というマクロで実際の型が決まる。ほとんどの文字列を引数にとる API と文字列をメンバにもつ構造体でこのような仕組みになっている。

Microsoft Windows NT とそれ以降の OS はファイル名に UNICODE を使うことができる。すなわち ANSI コードページには存在しない文字をファイル名に使うことができる。そのため引数が ANSI コードページな API ではファイルを正しく扱えない。引数が UNICODE な API を呼び出さない限り、ファイルを正しく扱えないのである。

2-2. Microsoft Windows における file name encoding の実装

Microsoft Windows では UTF-8 と file name encoding の関係はさらに複雑になる。ここまでは「file name encoding は `open` や `stat`、`readdir` のような API に使うことができる」という認識であった。GTK+-2.4 までは Microsoft Windows でもその認識で正しかった。Microsoft Windows ではどのバージョンも環境変数 `G_FILENAME_ENCODING` と `G_BROKEN_FILENAMES` には対応しておらず、GTK+-2.4 以前では file name encoding は ANSI コードページに固定されている。

しかし GTK+-2.6 以降は file name encoding の文字列を `open` や `stat`、`readdir` のような API に使うことができない。GTK+-2.4 以前のように file name encoding を ANSI コードページにしてしまうと、`GtkFileChooser` や `g_dir_read_name` で ANSI コードページにはない文字を含むファイル名を取得できなくなる。そこで GTK+-2.6 以降では file name encoding は UTF-8 として実装されている。`GtkFileChooser` や `g_dir_read_name` では可能ならば UNICODE 系の Microsoft Windows の API を呼び出し、UNICODE から UTF-8 への変換を行っている(不可能ならば ANSI コードページ系の Microsoft Windows の API を呼び出し、ANSI コードページから UTF-8 への変換を行っている)。

これに対応して GLib では `g_open` や `g_stat` などの API を新たに用意している。これらは `g_open` ならば `open` に、`g_stat` ならば `stat` と同じ働きをする。UNIX 系 OS ではこれらは単に名前が変わっただけとみなすことができる。Microsoft Windows では `g_open` や `g_stat` などの API の引数のファイル名はやはり file name encoding であり file name encoding の実体は UTF-8 なので、引数のファ

イル名は実際には UTF-8 になる。Microsoft Windows では例えば `g_open` は引数のファイル名を UNICODE に変換して `wopen` を呼び出す (UNICODE が利用できないならば引数のファイル名を ANSI コードページに変換して `open` を呼び出す)。`g_open` や `g_stat` などの API を利用すれば、UNIX 系 OS でも Microsoft Windows でもソースコードの記述を一貫して同じにすることができる。`g_open` や `g_stat` などの API の説明は GLib Reference Manual の File Utilities にある。UNIX 系 OS であっても GTK+-2.6 以降ならば `g_open` や `g_stat` などの API を使うべきである。

2-3. file name encoding と "真の" ファイル名の文字符号化方式の関係

GLib Reference Manual の File Utilities にある API のうち `g_chmod`、`g_access`、`g_creat`、`g_chdir` は GLib-2.8 以降である。GLib-2.6 で `chmod`、`access`、`creat`、`chdir` を使うならば、file name encoding を ANSI コードページに変換しなければならない。file name encoding から ANSI コードページへの変換は `g_locale_from_utf8` で、ANSI コードページから file name encoding への変換は `g_locale_to_utf8` でおこなう。通常は file name encoding を UTF-8 とみなすことはできないし、ロケールの文字符号化方式が実際のファイル名の文字符号化方式と同一とは限らない。これは Microsoft Windows だけの特別な処理となる。また GLib-2.8 以降であっても GLib 以外の対応する API が GLib の File Utilities にはない API を呼ぶ時には、この変換が必要になる。

GLib-2.8 以降ならば `g_win32_locale_filename_from_utf8` という API があるので、前述の `g_locale_from_utf8` の代わりにこれを用いる。`g_win32_locale_filename_from_utf8` は通常は `g_locale_from_utf8` と同じだが、もしファイル名を変換できなくてかつ Microsoft Windows の UNICODE 系の API が使える時には、`GetShortPathNameW` で短いファイル名を取得して、取得できたならばそれを ANSI コードページの文字列で返す。

いずれにしても Microsoft Windows では ANSI コードページを引数にとる API のために、`g_win32_locale_filename_from_utf8` (GLib-2.8 以降) または `g_locale_from_utf8` (GLib-2.6) で変換する特別な処理が必要になる。

バージョン	file name encoding の実体	file name encoding から <code>fopen</code> などの API 呼び出し時の変換	File Utilities の API
2.4 以前	ANSI コードページ	不要	なし
2.6	UTF-8	<code>g_locale_from_utf8</code>	<code>g_open</code> 、 <code>g_rename</code> <code>g_mkdir</code> 、 <code>g_stat</code> 、 <code>g_lstat</code> <code>g_unlink</code> 、 <code>g_remove</code> <code>g_rmdir</code> 、 <code>g_fopen</code> <code>g_freopen</code>
2.8 以降		<code>g_win32_locale_filename_from_utf8</code>	<code>g_chmod</code> 、 <code>g_access</code> <code>g_creat</code> 、 <code>g_chdir</code>

2-4. パスの区切り

パスの区切りに使われる文字は OS により異なる。UNIX 系 OS では「/」(2Fh) であるのに対して、Microsoft Windows では「\」(5Ch) である。GLib では `G_DIR_SEPARATOR` と `G_DIR_SEPARATOR_S` というマクロが定義されている。前者は数値としての区切り文字、後者は文字列としての区切り文字となる。例えば

```
#define G_DIR_SEPARATOR '/'
#define G_DIR_SEPARATOR_S "/"
```

となる。GLib-2.6以降ならばG_IS_DIR_SEPARATOR(c)というマクロも利用できる。このマクロはUNIX系OSならばcがG_DIR_SEPARATORの時に真となる。しかしMicrosoft Windowsでは特別な実装となっており、G_DIR_SEPARATOR(即ち「\」)または「/」で真となる。

GLibにはパスを扱うAPIとしてファイル名を返すg_path_get_basenameとディレクトリを返すg_path_get_dirnameがある。これらのAPIの扱う文字列はfile name encodingである。

Microsoft WindowsではGLib-2.2.3以降からg_path_get_basenameとg_path_get_dirnameは「\」と「/」の両方をパスの区切りとして扱う。上記のマクロ、G_IS_DIR_SEPARATORが定義されるよりも前から「\」と「/」の両方をパスの区切りとして扱っている。

複数のパスを区切るための文字としてG_SEARCHPATH_SEPARATORおよびG_SEARCHPATH_SEPARATOR_Sというマクロが定義されている。これはUNIX系OSでは「:」(コロン)であり、Microsoft Windowsでは「;」(セミコロン)である。Microsoft Windowsでは「:」(コロン)はドライブ名に使われる。したがって、これらのマクロを使わずに複数のパスを区切るための文字を「:」(コロン)でハードコードすると移植性に問題が生じる。

2-5.5Ch 問題

Microsoft Windowsでは「\」(5Ch)をパスの区切りとするが、この5Chはマルチバイト文字の2バイト目になりうる。もし単純に文字列を1バイトずつ「\」(5Ch)と比較した場合、マルチバイト文字の2バイト目をパスの区切りとして処理してしまう可能性がある。例えば「表」という文字はCP932では95h 5Chの2バイトになる。これをファイル名に含む「C:\doc\表 1.txt」(43h 3Ah 5Ch 64h 6Fh 63h 5Ch 95h 5Ch 31h 2Eh 74h 78h 74h)をディレクトリとファイル名に分けた時、43h 3Ah 5Ch 64h 6Fh 63h 5Ch 95hと31h 2Eh 74h 78h 74hに分かれてしまう。

GLib-2.6以降はfile name encodingの実体はUTF-8である。したがって5Ch問題は生じない。しかしGLib-2.4以前はfile name encodingの実体はANSIコードページなので5Ch問題の可能性はある。

GLib-2.0.0からGLib-2.4.8までのGLibの実装を見ると、どのバージョンでも5Ch問題が生じる。単純に文字列を1バイトずつ「\」(5Ch)と比較しているので、GLib-2.4以前のMicrosoft Windowsではg_path_get_basenameとg_path_get_dirnameは使えない。

3. 実際のアプリケーションソフトウェアの状況と提案

3-1. 実装例

画像閲覧ソフトで有名なGImageViewがある。このソフトウェアは元々、GTK+-1.2向けに開発されたもので、後にGTK+-2.0に対応した。しかしGImageViewのファイル名の扱いはそのGTK+-1.2時代のままで、ファイル名に関する文字符号化方式の変換を全く行っていない。よってASCII以外の文字を含むファイル名の表示に失敗する。(なお、私はGImageViewのパッチを作成し、2006年2月9日に作者の足永氏に送っている。まだ完全に修正を終えてはいないが、近い将来にはこの問題を解決する予定である。)

一方、FTPクライアントのgFTPはfile name encodingとUTF-8の変換を行っているので、ASCII以外の文字を含むファイル名も正しく扱える。しかし一部の表示では変換を行わずにfile name encodingの文字列をUTF-8の文字列として表示するので、表示に失敗する。

3-2. 提案

現在の仕様では文字列の文字符号化方式をGTK+/GLibを利用するプログラマ側が管理する必要がある。UTF-8もfile name encodingの文字列も同じgchar型の配列であり、コンパイル時に

警告が出ることはない。また UNIX 系 OS で ASCII の文字だけを使っている限り、問題は発覚しない。そういった事情から file name encoding のことを理解していても、文字符号化方式の変換を忘れてしまうこともある。gFTP の事例はまさに変換を忘れた例である。

GTK+-2.6 以降、file name encoding の文字列はネイティブな API に使うこともできなければ GTK+ のウィジェットで表示することもできない。それにも関わらず、文字列としての体裁をとっていることで混乱を生じていると私は考える。ファイル名をカプセル化した方が良いと思われる。

例えば GFileName という構造体を定義する。この構造体のメンバは実際には gchar 型の配列かもしれないし、Microsoft Windows ならば gunichar2 型の配列として実装されるだろう。しかし利用するプログラマ側からは構造体の実体は見えない。GtkFileChooser の gtk_file_chooser_get_filename はこの構造体を返し、gtk_file_chooser_set_filename はこの構造体を引数に持つようにする。GLib が提供するファイルアクセスのための API である g_open や g_fopen、g_dir_open などは文字列ではなくこの構造体を扱うようにする。また GFileName を扱う下記のような API が必要となるだろう。

```
GFileName *g_filename_get_basename (GFileName *fn)
GFileName *g_filename_get_dirname (GFileName *fn)
gchar *g_filename_get_display_name (GFileName *fn)
GFileName *g_filename_copy (GFileName *fn)
void g_filename_free (GFileName *fn)
gboolean g_filename_compare (GFileName *fn1, GFileName *fn2)
gchar *g_filename_get_native_name (GFileName *fn)
gboolean g_filename_set_native_name (GFileName *fn, const gchar *file)
gunichar2 *g_filename_get_win32_name (GFileName *fn)
gboolean g_filename_set_win32_name (GFileName *fn, const gunichar2 *file)
```

4.まとめ

UNIX 系 OS が一般に普及すれば、ファイルやディレクトリの名前に ASCII 以外の文字を使うことが珍しくはなくなる。また GTK+ は Microsoft Windows 版や Mac OS X 版もあり、GTK+ を使ったプログラムはマルチプラットフォームとなりうる。せっかく GTK+ を使ったのに、file name encoding への対応が不十分であったために UNIX 系 OS でしか正常に動作しないというのでは、ガッカリである。

日本語を母国語とするプログラマは日本語環境のゆえにこの file name encoding の問題を理解しやすい立場にあると思う。私自身も自作のプログラム (Video maid、Text maid など) を通じて文字符号化方式の問題の手本となるように努力したい。

参考文献

- [1] The GTK+ Team, "GLib Reference Manual",
<http://developer.gnome.org/doc/API/2.0/glib/index.html>
- [2] The GTK+ Team, "GTK+ Reference Manual",
<http://developer.gnome.org/doc/API/2.0/gtk/index.html>
- [3] Tony Graham, 乾和志, 海老塚徹, 関口正裕, "Unicode 標準入門", 翔泳社
- [4] Ken Lunde, 小松 章, 逆井 克己, "CJKV 日中韓越情報処理", オライリー