標題：Qt/Embedded 與 Qtopia 中文處理實戰
作者：黃敬群 <jimchyun@ccns.ncku.edu.tw>
最後更新：March 11, 2002

Qt/Embedded 中文處理實戰

作者：黃敬群 <jimchyun@ccns.ncku.edu.tw>

[-] 前言

Qt/Embedded 是 Trolltech 進軍 embedded system 的強力武器。秉持著 Qt 在桌面系統 (desktop) 的成功、KDE 成熟而完整的架構 [注 1]，Trolltech 精簡 Qt 的 API，並依據 embedded system 的需求加以作部份功能的擴充與調整。不同於一般 embedded 環境中常見的 toolkit 大多只有處理顯示的機制 (如 FLTK、MiniGUI、Gtk-FB、... 等)，Qt/Embedded 本身就是相當完整的架構，而充分支援 Unicode、依循 i18n (internationalization) / l10n (localization) 標準的發展路線，更是讓 Qt/Embedded 的應用突破語系的限制，本文就是探討如何對 Qt/Embedded 作中文處理，並提及 Qt/Embedded 的若干觀念，希望能對您有幫助。

* 註 1: Trolltech 裡頭不少工程師本身就是 KDE 的核心開發者，當他們設計 Qt/Embedded 與 Qtopia 時，多少也參考了 KDE 的架構

筆者實際上也是初學者，撰寫本文主要是心得筆記，有感一直沒有這類的文件可以參考，所以嘗試著把自己的諸多試驗整理而與大家共享，如有謬誤，還請來信指正，謝謝！

[-] 大綱

[-] Qt/Embedded 與 Qtopia 概況

提到 Qt/Embedded，不免就會聯想到同樣是 Trolltech 推出的重量級武器 -- Qtopia [註 2]。Qtopia 之前的名稱是 Qt Palmtop Environment (QPE)，目標是打造出一個整合性的 PIM (Personal Information Management)、多媒體效果、網路、... 等環境，適合中高階的手持式 (handheld) 應用，而 Qt/Embedded 正是擔任底層的 Framework [註 3] 角色，不僅畫面的繪製、訊息系統 (如著名的 Signals & Slots 機制、Qt/Embedded 仿效 KDE DCOP 的 QCop 等)，甚至連網路、XML 模組 (既然是模組，就可以選擇是否要加入編譯) 都納入核心 API 中。

* 註 2: Qtopia 的開發者首頁在 http://qpe.sourceforge.net/，而 Qt/Embedded 的資訊可在此取得：http://www.trolltech.com/products/embedded/

* 註 3：這邊指的 Framework 是物件導向程式設計中，一群協同合作的類別所構成、可設計出具再利用價值的特定軟體類型的骨幹支架，Qt 整體就是如此的例子

咱們來看看，依據 Trolltech 的規劃，Qtopia 的架構圖是如何：

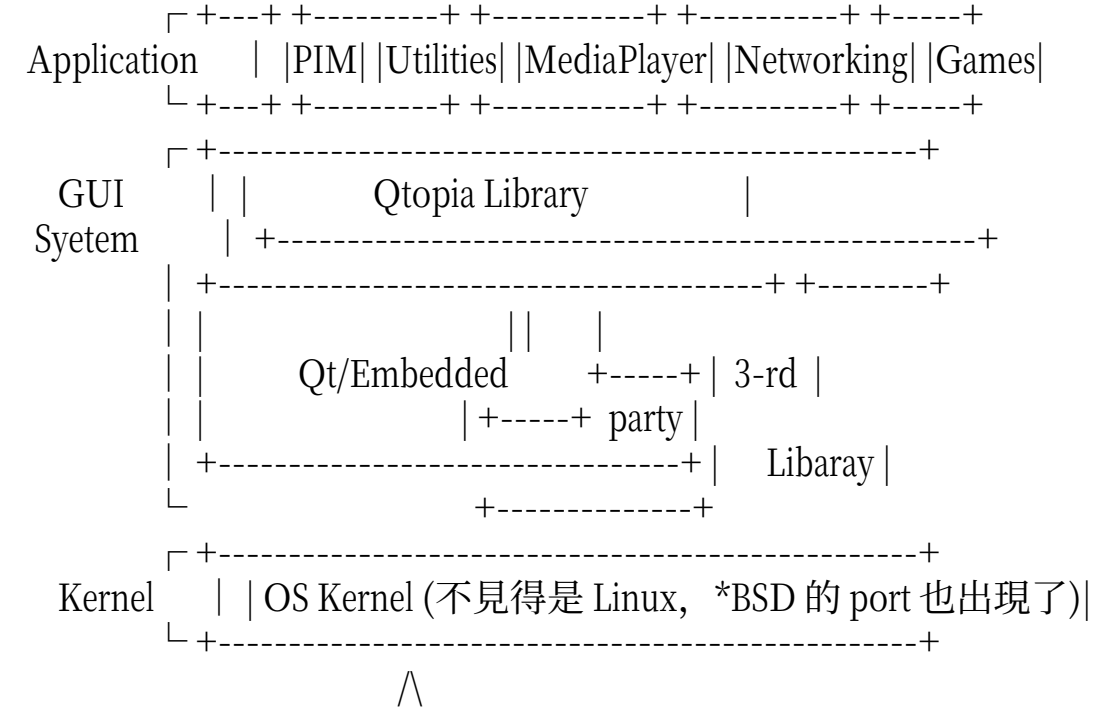


圖 1：Qtopia 架構圖

值得注意的是，Trolltech 的發展 Qt/Embedded 相關產品的過程中，巧妙的顯示出，不少原本是 Qtopia Library 的 API，因為可重用性高，就慢慢移往 Qt/Embedded 中，如 Qtopia Library 負責處理使用者設定的 <qpe/config.h>，就納入 Qt/Embedded (Qt/X11 亦然) 3.x 標準中。Qtopia 中諸多應用程式或

特徵只是使用者所見的部份，實際上，像是韓國嵌入式 Linux 大廠 Mizi Research 在手持式行動通訊環境所推出的產品 -- Linu@ [註 4]，就直接用自行最佳化處理的 Qt/Embedded 搭配專屬研發的功能，頗讓人有耳目一新的感覺。也可以這麼說，在 embedded system 中，Trolltech 真正的核心產品是 Qt/Embedded，一旦經由不同的應用發展，逐漸調適並修改出多樣化的特徵，就更能廣泛的推廣。

* 註 4：關於 Linu@ 的資訊，請參考：
  http://www.mizi.com/en/prod/embed/linuette-int.htm

[-] 手動調整 Qt/Embedded

如果不談 Qt 各項產品的高可攜性，Qt/Embedded 最大的優點必定是相當具有彈性的自訂化 (customization) 能力。藉由功能特徵的增減，Qt/Embedded 可從最陽春的、僅 630 kB 的 [Minimal Configuration]，到最完整、佔 5 MB 的完整 Configuration [註 5] 都有可能，並且你可以輕易修改 Configuration 檔，決定專屬的 Qt/Embedded 環境需要那些特徵，誠如禮運大同篇所說的「各取所需」。

* 註 5：因為必須支援的平台、環境包羅萬象，而面對不同的硬體環境，所提供的軟體功能當然勢必隨著調整，所以 "Configuration" 的概念就是來指定這些對應的硬體環境配置參數與實體功能的關聯性。Configuration 在其他嵌入式環境中也常常被使用，如 Sun Java 2 Micro Edition (J2ME) 就把若干功能、電力供應有限的裝置 (如 PDA、手機等) 定義在 Connected Limited Device Configuration (CLDC)，而把相對具有較佳運算能力、電力供應的裝置 (可能是 set-to box 一類的) 規範成 Connected Device Configuration (CDC)。

Configuration 除了影響 footprint (實際運作時所佔用的資源量)，更是影響在其上層的應用程式 (請留意上面提及 [圖 1] 所列出 Qtopia Library 與 Application)，不但可能造成編譯失敗，更會產生無法二進位相容的程式庫 (binary incompatible library)，就算空有執行檔也不能跑。

回到正題，如果要有完整的中文處理，Configuration 該怎麼調整呢？咱們仔細想想，所謂的中文處理，大概有兩項，首先，是看到的部份，也就是字型的顯示；再者，因為 Qt/Embedded 完整支援 Unicode，負責處理語系的 codec (編碼或解碼成 Unicode) 引擎就相當重要 (尤其在 Konqueror/Embedded 瀏覽非拉丁語文，如繁體中文的網頁，沒有處理 Big5 的 codec，就算光有中文字型也是沒輒)，同時，我們所想顯示的字型也必須以 Unicode 編碼。

理解我們必須處理的項目後，我們決定先來處理字型的部份。Qt/Embedded 內附

一篇很重要的文獻 [Fonts in Qt/Embedded]，你可以用瀏覽器閱讀 $(QTDIR)/doc/fonts-qws.html。在文件中提到，Qt/Embedded 可以支援以下四種型態的字型格式：

. TrueType (TTF) [scalable]
. Postscript Type1 (PFA/PFB) [scalable]
. Bitmap Distribution Format fonts (BDF) [non-scalable]
. Qt Prerendered Font (QPF) [non-scalable]

接觸過 X Window 的人應該對前三者不陌生，那我們來看看 QPF 是怎樣的格式："a light-weight non-scalable font format specific to Qt/Embedded"，一般來說，把 non-scalable 的 BDF 傳換成 QPF 格式的話，可以省下一半以上的空間。至於 TrueType 的支援，目前 Qt/Embedded 採用的是 FreeType2 Library 來顯示字型，可充分享受無段式 anti-aliased 顯示。當然，基於嵌入式環境的錙銖必較，不太可能全都支援，在後面我們會提到更改 Configuration 的方式。

QPF 是個相當有效且經濟的格式，Qt/Embedded 建議盡量都使用此格式的字型，那我們就來看看要如何轉換既有的 BDF 格式成 QPF，主要有四個步驟：

1. 轉換成 Unicode 為基礎的字型

   以繁體中文來說，就是 bdf(big5) -> bdf(unicode)，如台北字型：

   taipei16.bdf(big5) -> qtaipei16.bdf(unicode)

2. 使用 $(QTDIR)/tools/makeqpf 下的 makeqpf 來把 bdf -> qpf

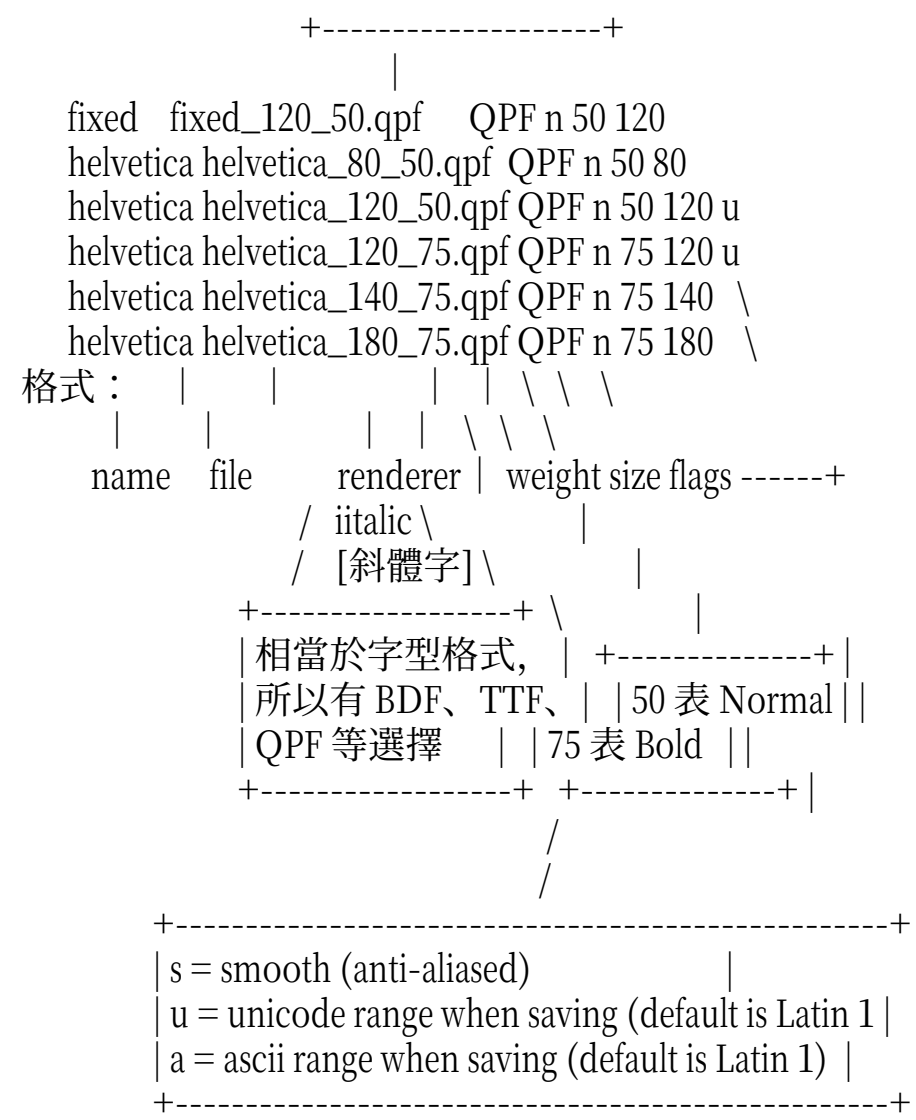
   ```
   # ./makeqpf -qws
   qtaipei16.bdf -> helvetica_160_50.qpf

   # ./makeqpf -qws -display Transformed:Rot270:0
   qtaipei16.bdf -> helvetica_160_50_t10.qpf
   (適用於 iPAQ，因為該機型本身處理顯示時，已經旋轉 270 度，後方的
    t10 表示作 transform 的識別)

   # ./makeqpf -qws -display Transformed:Rot90:0
   qtaipei16.bdf -> helvetica_160_50_t5.qpf
   ```

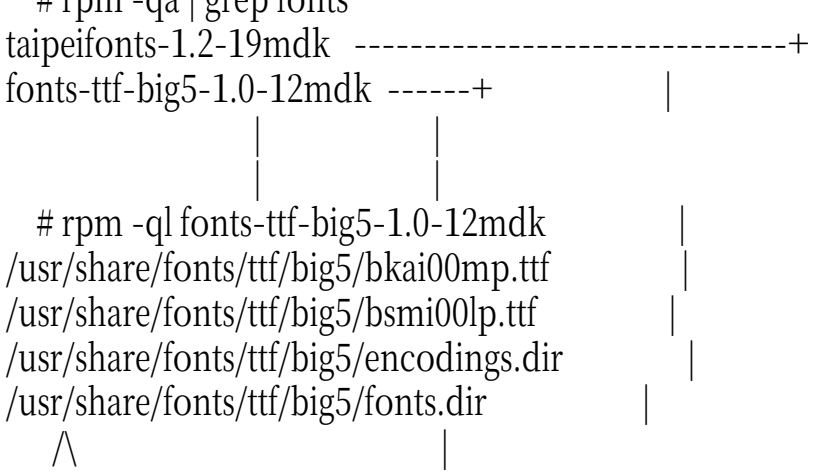
3. 把轉換好的 .qpf 複製到 $(QTDIR)/fonts/
4. 檢查 $(QTDIR)/lib/fonts/fontdir

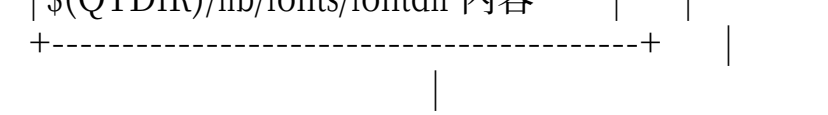
   ```
                        +--------------------+
                        | size 120 表示 12pt |
   ```



以下就以常見的文鼎與台北字型作說明。

首先，我們用用 RPM 來查詢電腦裡頭的字型：(筆者的系統是 Mandrake 8.1)






```
[jserv@venux jserv]$ rpm -ql taipeifonts-1.2-19mdk
/usr/X11R6/lib/X11/fonts/misc/taipei16.pcf.gz
/usr/X11R6/lib/X11/fonts/misc/taipei20.pcf.gz
/usr/X11R6/lib/X11/fonts/misc/taipei24.pcf.gz
/usr/X11R6/lib/X11/fonts/misc/taipeifonts.alias
/usr/X11R6/lib/X11/fonts/misc/vga12x24.pcf.gz
```

如我們所見，台北字型是以 PCF (Portable Compiled Format) 格式存在的，首先我們必須轉換 PCF --> BDF。

感謝李果正 (Edward G.J. Lee) 前輩提供的資訊，我們可以很方便的利用 xmbdfed 來作字型轉換工作，請見：

<%
作者: Edward (Edward G.J. Lee) 看板: statue
標題: 由文鼎 TTF 字型來製作 Unicode bdf/pcf 點陣字(Re: ttf2pt1 ...)
時間: Sun Mar 10 16:40:39 2002

我把標題改了一下，不然離題有點遠了。

運氣好，不小心試出來了。:-)

只須 xmbdfed 就可以了(要把 freetype 編譯進去)，不必靠 getbdf。

方法是啟動 xmbdfed，[Edit] => [Setup]，設定一下預設載入的字型大小，需注意的是，選擇的 point size，和事實可能會有出入，例如，選 20 點，出來的會是 21 點。

然後，[File] => [Import] => [TrueType Font] 來載入文鼎字型，然後 [File] => [Save as] 就成了。千萬不要由 [Server Font] 來 import，否則又會把 Unicode(UCS2) 的 BMP 字面的字(65536)，通通算進去，又會使 X 超載。

如果要使用自己的字型名稱，在 bdf 時就用編輯器改一下(其實在 xmbdfed 裡頭就可以改，不過會搞不清楚要改哪裡就是了)，改好後再使用 bdftopcf。

字型檔倒是小了很多。:)

不過，裡頭只有 big5 的一萬多字，要擴充的話，要由 xmbdfed 自行再加入。而且文鼎字型的一些字是在 private area，這和 Unicode 標準是不合的。

--
Warm Regards,
Edward G.J. Lee（李果正）
%>

因為 TrueType Font 是 scalable 的，在轉換成 non-scalable 的 BDF 時，必須額外指定字型資訊，上面的例子就是用 20 點的。

有了 BDF 字型後，我們就可以依造前面轉換既有的 BDF 格式成 QPF 的程序來

前置動作完畢後，接下來的就是實戰了。

這邊，我們以目前最穩定的 Qt/Embedded 2.3.2 來解說 (雖然 Qt/Embedded 3.x 系列已經推出，但是因為變動不小，需要對 Qtopia 作小幅修正，本文不打算探討)，Trolltech 也推薦用 Qt/Embedded 2.3.x 來編譯 Qtopia [註 6]。

* 註 6：目前這樣的組合 (Qtopia 1.5 + Qt/Embedded 2.3.x)，受限於 Qt/Embedded 2.3.2 無法支援多重執行緒 (multi-threaded)，效能部份受限，不過，Trolltech 已經在 QPE mailing-list (官方 Qt/Embedded 與 Qtopia 的討論區) 上表示會很快納入標準，亦即，在下一代的 Qtopia (版本是 2.0) 會有顯著改進

工欲善其事，必先利其器，在進行以下的操作前，請先準備好若干套件：

1. Linux 系統，雖然 Qt/Embedded 也有 *BSD 的 port，不過筆者沒有測試過，只好預設是 Linux 系統，並且，為了方便，建議用 RPM 為基礎的 Linux 套件，如 RedHat、SuSE，或 Mandrake 等

2. C++ compiler，這裡假設是 g++ (C++ frontend of GCC Compiler) [註 7]

3. Qt/X11 2.x 的發展套件，一般的套件命名是 libqt2-devel，如果沒有，請自行到 RPMFind (http://rpmfind.net/) 安裝備妥

4. Qt/Embedded 2.3.2，您可以到國內的 FTP 站找到 mirror，檔案名稱是 qt-embedded-2.3.2.tar.gz

5. Qtopia 原始程式碼，目前穩定版本是 1.4，不過因為變動很大，筆者建議還是到 Qtopia 開發者首頁用 CVS 取得程式碼。如果您在台灣，而且不想浪費時間或佔用對外頻寬，可以到筆者的網站 [註 8] 抓每天更新的 tarball (如果不介意，也可以順便抓筆者作的 x86 demo 來玩玩)

* 註 7：實際上因為相當多程式會使用到 tmake 來輔助建構 Makefile，但並非所有平台上的 compiler 都支援到，所以這邊使用可攜性最高的 g++。關於 tmake 的資訊，請見筆者撰寫的 [tmake 簡介] 一文

* 註 8：筆者有維護一個專門放置若干 Qtopia 快照與相關檔案的網頁：

http://ccns.ncku.edu.tw/~jimchyun/qpe/

解開 Qt/Embedded 與 Qtopia tarball 後，我們就要開始進行對 Qt/Embedded 作調整設定的動作。

```
# cd qt-2.3.2
# export QTDIR=/home/jserv/qt-2.3.2
  (筆者的環境，請自己修改，且，若未特別指定，本文所出現的 $(QTDIR) 皆指本環境參數)
# export LD_LIBRARY_PATH=/home/jserv/qt-2.3.2/lib:$LD_LIBRARY_PATH
```

-* ./qvfb -depth 16 -width 640 -height 480
-* ./qconfig -qws

在執行 Qt/Embedded 的 configure script 時，你可以下達 -qconfig 後面加上功能的選項，這些選項有 minimal、small、medium、large，與 local。而如果不使用 -qconfig 選項，會讓全部的 Qt/Embedded 特徵有效，也就是 Everything)，如：

```
# ./configure -qconfig minimal
```

以上每個選項會對應到在 $QTDIR/src/tools/ 的標頭檔：

```
qconfig-minimal.h
qconfig-small.h
config-medium.h
qconfig-large.h
qconfig.h
qconfig-local.h
```

值得注意的是 qconfig-local.h 是我們專屬的 Configuration，須自行建立。

Qt/Embedded 把這些標頭檔稱為 "Feature Definition Files" (特徵定義檔)。在 $(QTDIR)/doc/features.html 中，你將會找到一連串的以 QT_NO_ 開頭的巨集定義，而這每個巨集會讓 Qt 的某項特徵無效 (注意，是無效，disable)。所

以，在你配置一個自訂的 Qt 特徵時，把在 src/tools/qconfig-local.h 中，你所不需要的項目記錄下來。你可以使用其他 feature definition 檔作為出發點。

建立一個自訂的 Qt 環境就需要一番嘗試與重新編譯的過程。建立一個移去某些功能的 Feature Definition File 不難，因為這些功能特徵在很多地方都會處理相依性 (dependency) 的問題，這也是為何常在 Qt Libarry 的原始程式碼中看到為數不少的巢狀 preprocessor。要達到瘦身的方式，最簡單的處理就是把 Qt 中若干選擇性的 modules 移除。如果你不需要那些模組，可以這麼做：

```
#define QT_NO_TABLE
#define QT_NO_XML
#define QT_NO_CANVAS
```

光靠以上三個巨集設定，你就在最終 strip 後的 library 中省下 350 kb 的空間。

首先要編撰 qconfig-local.h，因為我們主要是應用在 Qtopia，所以我們直接參考 qconfig-qpe 這個給 Qtopia 的參考 Configuration：

```
# cp src/tools/qconfig-qpe.h  src/tools/qconfig-local.h
```

現在，中文處理的項目只剩下 Big5 的 codec 了，咱們調出 $(QTDIR)/src/tools/qconfig-local.h 的 Configuation 來：

```
# vim src/tools/qconfig-local.h
(不見得要用 vim 啦，只不過因為有 high-lightening，可能比較方便些，
 至少筆者如此認為)
```

於是，我們可以看到相當多選項，但目前我們只需要留意 "codec" 與 Big5 處理的部份，所以，找出以下的項目：

```
#define QT_NO_CODECS
#define QT_NO_BIG5CODEC
```

把以上這兩項的限制取消，怎麼作呢？很簡單，直接在列前打 "//" 註解即可。接著，我們就可以編譯 Qt/Embedded 了：

```
# make
make[1]: Entering directory `/home/jserv/qt-2.3.2'

 Qt must first be configured using the "configure" script.
```

The make process will now run this...

This is the Qt/Embedded Free Edition.

You are licensed to use this software under the terms of
the GNU General Public License (GPL).

Type 'G' to view the GNU General Public License.
Type 'yes' to accept this license offer.
Type 'no' to decline this license offer.

Do you accept the terms of the license?
yes <---- 注意到開發免費軟體的時候適用 GPL
Choose a feature configuration:

1. Minimal (630 kB)
2. Small (960 kB)
3. Medium (1.5 MB)
4. Large (3 MB)
5. Everything (5 MB)
6. Your own local configuration (src/tools/qconfig-local.h)

Sizes are stripped dynamic 80386 build. Static builds are smaller.

Your choice (default 5):
6 <--- 用我們自己的設定

注意：

可以調出 Qt/Embedded 的 $(QTDIR)/include/qfeatures.h，我們可以發現：

```
#if defined(QCONFIG)
#include "qconfig-" QCONFIG ".h"
#else // everything...
#include <qconfig.h>
#endif
```

而在 QPE 的編譯過程中，會指定 -DQCONFIG=\"qpe\"，很自然的，這會使用到
$(QTDIR)/src/tools/qconfig-qpe.h 的 Configuation，而非我們之前弄的
qconfig-local，解決方法有二：

1. 直接改 $(QTDIR)/src/tools/qconfig-qpe.h 內容，並在建構 Qt/Embedded 時，改下 ./configure -qconfig qpe
2. 修改 $(QTDIR)/include/qfeatures.h，把上面幾行直接用

   #include <qconfig-local.h>

   換掉，這是最快且保證對的方式

最後，下達：

```
# ./configure -qconfig local -depths 8,16,24 -qvfb \
  -gif -system-jpeg
# make
```

這時，Qt/Embedded 的編譯過程就完畢了，然後我們可以建構 Qtopia：

```
# cd ../qpe/
# ./configure
# make
```

ok，所有的基礎建設都完畢了。

[-] QVFB (Qt Virtual FrameBuffer) 使用

Qt/Embedded 目前最常被搭配的作業系統就是 Linux，並以 Linux Framebuffer 作底層顯示的基礎，而 qvfb (Qt Virtual FrameBuffer) 則是 Trolltech 設計給 Qt/Embedded 的開發者能夠直接在 X Window 模擬 Linux Framebuffer，而不用很麻煩的切換到終端機下。你大可把 qvfb 想成 Qt/Embedded 的模擬器，因為不僅提供你抓快照 (snapshot) 的功能，還能讓你依喜好，來更改外觀 (skin)。在後面的操作，我們也直接使用 qvfb。

首先，我們來建構 qvfb。

需要留意的是，因為 qvfb 是提供「在 X Window 下模擬 Linux Framebuffer」的機制，所以呢，我們必須把 $(QTDIR) 切換到原本 Qt/X11 的設定，如：

```
# export QTDIR=/usr/lib/qt2
```

接著，切換到 Qt/Embedded 目錄下的 tools/qvfb/，進行建構：

```
# make
```

於是，你可以發現該目錄下多了 "qvfb" 的執行檔，為了方便使用，您可以複製到 $(PATH) 能找到的地方。

qvfb 該怎麼使用呢？

筆者的習慣是，直接用系統的 Launcher (KDE 或 GNOME 都會有的「快速執行」功能) 來執行 "qvfb -depth 16" 的指令，然後保持開著，後續撰寫程式或測試時，直接盯著 qvfb 視窗的輸出即可。

qvfb 有不少參數，不過在此僅說明幾個常用的，舉個例子：

```
# ./qvfb -depth 16  -width 640  -height 480
       [--------] [---------] [----------]
       以 16-bits    寬度        高度
       深度顯示
```

依據不同的 target (應用平台)，我們可依需求調整 depth、width，以及 height，就會有不同的風貌了，大家可以看看小弟的網站，可以很清楚看到箇中差異。

既然提到 qvfb，我們來談談要如何執行 Qt/Embedded 的程式。

透過 Qt/Embedded 編譯的程式，看起來跟一般 Qt 沒有兩樣，但是，實際上，用多了若干命令列參數，其中 "-qws" 就是指定顯示時，要輸出在 QWS (Qt/embedded WorkSpace)，而，如果已經啟動 qvfb，那麼就會轉向到裡頭，不然直接在該 device 輸出。

因為以下的部份都在討論 Qt/Embedded，所以我們把 $(QTDIR) 切換回 Qt/Embedded 目錄下。

$(QTDIR)/tools/qconfig/ 目錄下的 qconfig 工具是 Trolltech 提供給開發者能夠直接更改 Qt/Embedded 設定值的工具 (不過，因為之前編譯時，我們已經手動取消 / 保留若干特徵，所以選項就變少了)，我們用 qconfig 作為範例，看看一個 Qt/Embedded 的程式要如何執行：

```
# cd $(QTDIR)/tools/qconfig
# make
# ./qconfig -qws
```

好，現在把焦點移轉到剛剛啟動的 qvfb，你就可以看到原本黑漆漆的一片，現在已經有個小視窗了。

Qtopia 下的應用程式的啟動就更簡單了，直接切換到 $(QPEDIR)/bin，然後執行

```
# export QPEDIR=/home/jserv/qpe
  (當然是筆者的環境)
# ./qpe
```

你將會看到個典型的 PDA 程式畫面，直接點選圖示即可。

[-] 應用軟體翻譯

前面所說，Qt/Embedded 環境支援完整的 Unicode 處理，所以要進行 i18n 就相當容易。在 $(QTDIR)/doc/i18n.html 就詳細描述這些過程，這邊我們簡單的說明一下。

留意到 $(QPEDIR)/i18n 目錄底下的檔案：

```
# tree -L 1 i18n/
i18n/
|-- de        -------------------\
|-- en                           |
|-- hu                           |
|-- ja                           +--- 這幾個目錄就是存放各區域的訊息
|-- ko                           |    (message) 檔案
|-- no                           |
|-- zh_CN                        |
|-- zh_TW       ------------------/
|-- qpe-i18n-de.control
|-- qpe-i18n-en.control
|-- qpe-i18n-hu.control
|-- qpe-i18n-ja.control
|-- qpe-i18n-ko.control
|-- qpe-i18n-no.control
|-- qpe-i18n-zh-CN.control
`-- qpe-i18n-zh-TW.control
```

咱們看看 $(QPEDIR)/i18n/zh_TW/ 內，正有兩個檔案：

```
textedit.po
textedit.qm
```

.po 代表原始的訊息檔，.qm 則是 Qt/Embedded 所用的格式，要把原始人類可讀的 .po 轉換成 .qm 也是相當容易，用 msg2qm 的工具即可，使用方式如下：

```
  # msg2qm
usage: msg2qm [-scope default] infile [outfile]
```

所以，以上面的 textedit.po 為例，我們可以這麼下：

```
  # msg2qm -scope zh_TW.Big5 textedit.po textedit.qm
```

而，因為 textedit.po 內已經明確指定用 Big5 編碼，所以 -scope 大可省略。

目前 Qtopia 大部份的訊息檔，筆者大致翻譯五成以下，如不想做白工，可到筆者網站下載。

[-] Launcher/Directory 的翻譯

Launcher/Directory 的資料主要在 $(QPEDIR)/apps/ 下，我們可以列表發現：

```
  # tree -L 1
.
|-- Applications      \
|-- Games             |
|-- Settings          +--- 分類
|-- Toys              |
|-- Utilities         /
|-- __7separator.desktop
`-- __9quit.desktop
```

__7separator.desktop 與 __9quit.desktop 則是所謂的 Desktop Description 檔，如果接觸過 KDE 的開發者應該不會陌生，這兩者分別代表選單的分隔線與結束項目的命名。

我們會把重點把在分類的目錄下。

首先，來看看 Applications/：

```
  .directory  <======== 表示該分類的屬性
  addressbook.desktop -\
  calculator.desktop  |
  citytime.desktop    +--- 應用程式分類下各程式的屬性
    ...               |
```

調出 .directory 來看：

```
  # cat .directory
[Desktop Entry]
Name=Applications
Name[ca]=Aplicacions
Name[cs]=Aplikace
Name[da]=Anvendelser
Name[de]=Anwendungen
con=AppsIcon
```

這與 KDE 的處理方式一致，Name[] 所表示的就是不同區域的名稱，不過，請注意，右方區域命名都以 utf8 編碼，所以如果要加入 zh_TW 的項目的話，請先轉碼。

那怎麼轉碼呢？感謝黃志偉前輩的提醒，有兩個方式：

方法一: 使用 KDE2 中的 kbabel, 在 kdesdk 套件中。可以直接存取 utf-8 檔案。

方法二: 將它轉換成 Big5 再行修正，要 commit 時再轉回 utf-8。你可以用任何可轉 utf-8 <-> Big5 的程式，例如 autoconvert、utf-convert ... (CLE 裡面都有)

以方法二來說，可以使用 zh-autoconverter 來看，如：

```
  # autob5 -i utf8 -o big5 < Konq.desktop
        ------- -------
        輸入編碼 輸出編碼
```

所以呢，下次要新增 [zh_TW] 就很簡單啦，先用編輯器打

```
  Name[zh_TW]=
       ^^^ 後面不要有換行符號
```

在某個檔案打入你想要出現的中文名稱，比如「網路征服家」(Konqueror) 存成 tw，然後下：

```
  # autob5 -i big5 -o utf8 < tw >> Konq.desktop
                    -- Append 轉換後的 utf8 格式到原本的檔案,
                      天衣無縫
```

當然，Linux 系統下可以使用現成的 glibc 提供的 iconv，如下：

```
  # iconv -f utf8 -t big5 < input > output
```

不僅 .directory 如此，其他 .desktop 的檔案按照相等程序也可中文區域化完畢。附帶一提，筆者也替大家完成一半以上的翻譯工作了，請自行到筆者的網頁下載。

[-] System Script

在測試平台把大部份的工作都完成後，就可以考慮移轉到 iPAQ 之類的機器去測試了，在許多文獻都提到詳細作法，筆者在此省略。不過，我提醒一下相關的中文問題的注意事項。

在 /usr/etc/rc.local 編撰 $LANG 環境變數，要記得設定成繁體中文，也就是 zh_TW.Big5

然後，檢驗 qpe 的 script，應該是這麼樣的：

```
#!/bin/sh
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/qt/bin
LD_LIBRARY_PATH=/lib:/usr/lib:/qt/lib
QTDIR=/qt
LANG=zh_TW.Big5
export PATH LD_LIBRARY_PATH QTDIR LINUETTEDIR QWS_KEYBOARD LANG
        ...
```

大致是這樣，這部份只需要留意到環境變數即可。

[-] 核心支援項目

在嵌入式環境中，Linux Kernel 可能會被一再的精簡，但，無論怎麼精簡，記得要再 fs 項目支援繁體中文的 [Native Language Support]，以 make xconfig 來說：

```
File systems --->
Native Language Support --->
Traditional Chinese charset (CP950, BIG5)
```

然後，重新編譯即可。