DEVELOPMENT ENVIRONMENT OF PC_BSD

KievBSD 2012

by Yuri Momotyuk (yurkis@gmail.com)

Development environment of PC-BSD

Background:
C++, Qt, FreeBSD

Development environment of PC-BSD

Plan:

- Subversion layout
- Steps for making PC-BSD utility
- Project file
- libpcbsd
- Translations
- Making single instance app
- Adding control panel item

Subversion layout

PC-BSD svn url: svn://svn.pcbsd.org/pcbsd/

Current branch (trunk): current/

Stable branches: branches/

Subversion layout

build-files/

PC-BSD build related files and settings (out of scope of this presentation)
You may take a look into:
•build-files/conf/port-make.conf – make.conf for PC-BSD build
•build-files/ports-overlay/ – ports which absent into FreeBSD ports tree but uses for PC-BSD build
•build-files/src-overlay/ - Sources which are absent in FreeBSD source tree
•build-files/src-patches/ - Patches for FreeBSD source tree

•overlays/

Overlays will by copied against installed FreeBSD world. This is good place to override FreeBSD config files. For example PC-BSD rc script/rc.conf (with parallel daemons startup) may be found at overlays/desktop-overlay/etc/rc (rc.conf). 64 bit overlay will be copied after 32 bit overlay copy. So it contains only 64 bit specific files.

•src-kde4/

KDE related sources (thumbnailer for PBI files)

•src-qt4/

Source codes of all graphical PC-BSD utils. Every utility places on own directory. Project file should be named just as directory (*src-qt4/about-gui/about-gui.pro* for example) PC-BSD build script use src-qt4/src-qt4.pro project for build all utils. So if some util absent in src-qt4.pro it will not be included into PC-BSD distribution.

•src-sh/

PC-BSD shell scripts. All script should have Makefile with install traget. During build process PC-BSD build script use src-sh/Makefile to install all scripts.

Steps for making PC-BSD utility

- **Make Utility**

  All of graphical PC-BSD utils ia a Qt (or at least qmake based) applications. So in common case below I expect that your application is based on Qt.

- **Change project file for correct PC-BSD distribution build**

  PC-BSD build system require some predefined build and install behavior so you need to make some changes in project file.

- **Add translation**

  All of graphical PC-BSD utils supports multilanguage. So you should add ability of UI and messages translation.

- **Make application single instance (if need)**

  Most of configration utils should be single instance applications (if you try to run another instance, it just display first instance even it minimized)

- **Add item to control panel (if need)**

  Most of system configuration utils are present in PC-BSD control panel

Project file

•Build

PC-BSD build system expect that application builds into /usr/local/bin directory (that binary will be there without making install rule). So you should change DESTDIR variable in your project in that way:

```
DESTDIR=/usr/local/bin
```

•Install

• Desktop file and icon

Your application may (should?) have icon and .desktop file. So you may add:

```
icons.path=/usr/local/share/pcbsd/icons/
icons.files=YOUR_APP.png
desktop.path=/usr/local/share/applications/
desktop.files=YOUR_APP.desktop
INSTALLS += icons desktop
```

•Application shared files

Application should keep own shared files somewere in /usr/local/share/pcbsd/ (dont forget to create and clean destination directory if need).

For example for control panel items:

```
cleanitems.path=/usr/local/share/pcbsd/pc-controlpanel/items
cleanitems.extra=rm -rf /usr/local/share/pcbsd/pc-controlpanel/items
mkdiritems.path=/usr/local/share/pcbsd/pc-controlpanel/items
mkdiritems.extra=mkdir -p /usr/local/share/pcbsd/pc-controlpanel/items
cpitems.path=/usr/local/share/pcbsd/pc-controlpanel/items
cpitems.extra=tar cvf - --exclude '.svn/' -C items . 2>/dev/null | tar xvf - -C
/usr/local/share/pcbsd/pc-controlpanel/items 2>/dev/null
INSTALLS+= cleanitems mkdiritems cpitems
```

libpcbsd

In common case most of PC-BSD utils just do several things: change value in configuration file; run some script or application and anlyze output. libpcbsd provides useful functions for that (and more) actions and provides interface to PC-BSD specific staff.

- **Add to project**

You may add libpcbsd to your project by adding that lines to project file:

```
LIBS    += -L../libpcbsd -L/usr/local/lib -lpcbsd
```

Include file:

```
#include <pcbsd-utils.h>
```

libpcbsd

•Some useful things:

•Run shell command

QStringList output = Utils::runShellCommand(QString(“ls -l”));

(run 'ls -l' command and store output into output string list)

•Run in terminal

Utils::runInTerminal(QString(“portsnap fetch update”), QString(“Ports update”));

(Open desktop environment related terminal application (konsole for kde, gnome termina for gnome, etc) and run 'portsnap fetch update'. Terminal window title should be “Ports update”.)

•Open directory in file manager

Utils::openInFileManager(QString(“/home/yurkis/Downloads/”));

(Open directory in desktop environment related file manager (dolphin for KDE, nautilus for Gnome, etc.)

•Get sysctl value

QString CPU = Utils::sysctl("hw.model");
QString mem = Utils::sysctlAsInt("hw.realmem");

(Get CPU info and amount of memory from hw.model and hw.realmem sysctls)

•Change and get config file value

Utils::setConfFileValue(QString(“/etc/rc.conf”), QString(“sshd_enable”), QString(“\”YES\””);

(set sshd_enable=”YES” in rc.conf file – enable ssh daemon)

libpcbsd

- **Also:**
- **Validation of v4 and v6 IP dress**
- **Format amount of bytes to human readable format (“2.00 Gb” for example)**
- **Get and set pcbsd.conf value**
- **Get and set system proxy settings**
- **Restart networking**
- **Widgets for control meta packages, WiFi security selection, etc**
- **... and more**

Translations

•Changes in sources for translations support

Changes to enable translations for your app against standard generated main.cpp file are shown below (in blue):

```
#include <qtranslator.h>
#include <qlocale.h>
#include <QFile>
#include "mainwindow.h"
#include "../config.h"

int main(int argc, char *argv[])
{
    QtApplication a(argc, argv);

    QTranslator translator;
    QLocale mylocale;
    QString langCode = mylocale.name();
    if ( ! QFile::exists( PREFIX + "/share/pcbsd/i18n/YOUR_APP_" + langCode + ".qm" ) )
        langCode.truncate(langCode.indexOf("_"));
    translator.load( QString("YOUR_APP_") + langCode, PREFIX + "/share/pcbsd/i18n/" );
    a.installTranslator( &translator );

    MainWindow w;
    w.show();
    return a.exec();
}
```

At first we create QTranslator and QLocale objects and then load translations from /usr/local/share/pcbsd/i18n/. Translation files for you app should be /usr/local/share/pcbsd/i18n/YOURAPP_LANGCODE.qm
PREFIX constant defined in ../config.h file and currently is “/usr/local/”

Also dont forget to use _tr() macro for strings in source code:

```
ui->somelabel->setText(_tr(“Some translated text here”));
```

Translations

•Changes in project file for translations support

•Files with translations should be installed in /usr/local/share/pcbsd/i18n. File name should be YOURAPP_LANGCODE.qm (AboutGui_uk.qm for example).

•In subversion translations places in i18n/ subdirectory of your application directory.

Lets add rules for translations instalation to project file:

```
TRANSLATIONS =  i18n/YOUR_APP_af.ts \
                i18n/YOUR_APP_ar.ts \
 ................. and many simular strings here ................................

dotrans.path=/usr/local/share/pcbsd/i18n/
dotrans.extra=cd i18n && lrelease-qt4 -nounfinished *.ts && cp *.qm
/usr/local/share/pcbsd/i18n/

INSTALLS+=dotrans
```

Making single instance app

- **Single instance library**

 To be single instance PC-BSD utils uses SingleApplication library from Qt solutions (devel/qt4-qtsolutions-singleapplication port included in PC-BSD base). To use this library add to your project file:

LIBS += -L/usr/local/lib -lQtSolutions_SingleApplication-head

Making single instance app

Changes in source code

Changes against standard generated main.cpp file to make your app single instance are shown below (in blue):

```
#include <qtsingleapplication.h>
 #include "mainwindow.h"
//////////////////////////////////////////////////////////////////////////////////
  int main(int argc, char *argv[])
  {
      QtSingleApplication a(argc, argv);
      if ( a.isRunning() )
           return !(a.sendMessage("show"));

      MainWindow w;
      w.show();

      // Issue workaround - see explanation in text below
      QObject::connect(&a, SIGNAL(messageReceived(const QString&)), &w, SLOT(slotSingleInstance()) );
      return a.exec();
  }
```

Your application should be child of QtSingleApplication class. That class has isRunning() method to check is application already running.

QtSingleApplication class already has mechanisms to bring to front main window when another instance was run. But that code is not work properly with all window managers. So we should manually send message to previous instance, and when that message will be received bring window to top (See sending message and signal connection example in main.cpp above).

On main window header:

```
public slots:
        void slotSingleInstance();
```

On main window sources:

```
void MainWindow::slotSingleInstance()
{
    this->hide();
    this->showNormal();
    this->activateWindow();
    this->raise();
}
```

Adding control panel item

•Add item

• Control panel item in common case is an ordinary .desktop file
• Control panel items on installed system are placed at /usr/local/share/pcbsd/pc-controlpanel/. Each items category is in separate subdirectory.
• On subversion repo control panel items are placed at src-qt4/pc-controlpanel/items/

So you should only add your application .desktop file to src-qt4/pc-controlpanel/items/ for some category and see your application in control panel in next PC-BSD build.
You also may change project install rule to copy .desktop file to /usr/local/share/pcbsd/pc-controlpanel/ but for me (as for control panel maitainer) really more simple to change something if items will be in single place.

•Aditional features

Control panel items has several enhancement between ordinary .desktop files. You may use this if you need:

•Visibility condition

You may use some shell command to check if your item should be shown. You may use PC-TryCommand field (name is case sensative!). This field may contain shell command. If that command ends with zero return code – your item will be shown.
For example nvidia setting item will be shown only if nvidia kernel module was loaded:
PC-TryCommand= kldstat|grep nvidia.ko

•Desktop envirunment dependent visibility

If you wish to display item only on particular desktop environment you may use PC-RequiredDE field with desktop environment name.
For example:
PC-RequiredDE=KDE

•Other features

•Also you may display item only if some pbi installed and some other

Web resources

Common

- **PC-BSD main site:** http://pcbsd.org/
- **Blog:** http://blog.pcbsd.org
- **Wiki (incl. some development things):** http://wiki.pcbsd.org
- **Forums:** http://forums.pcbsd.org

Development related

- **Bug tracker (trac):** http://trac.pcbsd.org/
- **Translation (pootle):** http://pootle.pcbsd.org
- **PBI build server reporting:** http://pbibuild.pcbsd.org
- **Mail lists:** http://lists.pcbsd.org

Thanks!

... and long live PC-BSD! :)