“SL シリーズザウルス”　＋　“Qtopia”

開発チュートリアル

(第 1.10 版　2003 年 1 月 22 日)

シャープ株式会社
通信システム事業本部
モバイルシステム事業部

改訂履歴

2003 年 1 月 22 日	control ファイル設定例、QtDesigner 説明について修正、Qt 関連開発ツールの紹介を追加、コンパイラのバージョン注意事項を追加
2002 年 12 月 13 日	初版リリース

目次

はじめに

この資料では、Linux がはじめての方にもわかるように、クロスコンパイラや Qtopia 開発環境の設定、そしてテストアプリケーションの作成手順を解説します。

1. Linux マシンのターミナルでコマンドを入力する例が多数出てきますが、資料の中の表記は

   $ command ….[Enter]

   です。最後の[Enter]はキーボードの Enter キーの入力を示しています。

   また、何か任意のものを入力する場合には、

   $ （パスワードを入力）[Enter]

   のように、（）で囲んでいます。

2. 文中に「SL ザウルス」という製品名がでてきますが、これは国内SL シリーズのザウルスを指しています。2003 年 1 月 22 日現在、SL-A300、SL-B500、SL-C700 が発売されています。それぞれの機種の製品概要に関しましては、

   http://more.sbc.co.jp/sl_j/info/common_spec.htm

   でご確認いただけます。

関連サイト

国内 SL ザウルスソフト開発サポートサイト宝箱 Pro

http://more.sbc.co.jp/sl_j/sl_top.asp

Qt 関連情報、各種ツールダウンロード　　trolltech 社

http://www.trolltech.com/

第 1 章：開発環境のセットアップ

この章では、Qtopia のアプリケーションを開発するために必要な OS やツール、それらを PC にインストールする方法を解説します。

1.1　開発用 PC

開発用 PC は、Linux をインストールしておきます。RedHat や SuSE、Mandrake、または、Caldera などの RPM をサポートするディストリビューションを使用してください。Slackware、及び、Debian のディストリビューションも使用できますが、RPM フォーマットをサポートするために RPM 変換ユーティリティを使う必要があるかもしれません。この資料では、Redhat7.3、シェルには bash を使用しています。
また、SL ザウルスにアプリケーションソフトをインストールする場合、SD カードや CF カードを使用することになりますので、カードや PC カードアダプタをご用意ください。

Linux マシンに USB が搭載されている場合、PC-Linux 用の USB ドライバをインストールすれば、SL ザウルスに付属の USB 接続ケーブルを使用して、ファイルを転送することも可能です。PC-Linux 用の USB ドライバについては、次の URL から入手してください。

http://more.sbc.co.jp/sl_j/tool/tools.htm#Support_Tool（開発サポートツール）
http://more.sbc.co.jp/sl_j/tool/usb_driver/kernel-zaurus-2.4.18.5-4a.i386.rpm（ドライバの RPM）

今後、SL シリーズのカーネルの再構築や rootfs の再構築も行うことになると思われますので、HDD の空容量や RAM、CPU 性能については次の内容を参考にしてください。

・HDD の空き容量　　約 400MB
・CPU、RAM 容量　　PC-Linux が快適に動作する程度

1.2　ツールのセットアップ

以下、1.2.1 以降では、開発に使用する各種ツールをインストールしていきます。資料に記載のとおり操作してインストールされるディレクトリがデフォルトのものですので、特に理由が無い限り、インストールされるディレクトリを変更しないで下さい。もし変更された場合、その後の説明どおりに処理が進まないばかりでなく、いろいろな設定をインストール状況に応じて変更する必要があります。

1.2.1 クロスコンパイラのセットアップ

開発用 PC に LinuxOS をインストールした後、gcc のクロスコンパイラ関連のツールをインストールします。各パッケージの入手先は、宝箱 Pro の次の URL から参照できます。

http://more.sbc.co.jp/sl_j/tool/tools.htm#Linux（宝箱 Pro　Linux 関連開発ツールのページ）

インストールするパッケージは次の 4 つです。新しい Linux ディストリビューションには、バージョン 2.95 より新しいものがインストールされていることもありますが、必ずこのバージョン 2.95 をインストールしてご使用ください。

gcc-cross-sa1100-2.95.2-0.i386.rpm	クロスコンパイラ(gcc)
glibc-arm-2.2.2-0.i386.rpm	ライブラリ(glibc)
linux-headers-arm-sa1100-2.4.6-3.i386.rpm	ヘッダファイル
binutils-cross-arm-2.11.2-0.i386.rpm	ユーティリティ

上記の RPM ファイルは、次のコマンドを用いてインストールしてください。

```
$ rpm -Uvh filename.rpm[Enter]
```

例えば、ARM 用 gcc コンパイラをインストールする場合には、次のようにコマンドを実行します。

```
$ rpm -Uvh gcc-cross-sa1100-2.95.2-0.i386.rpm[Enter]
```

初期設定では、これらのクロスコンパイラ関連のツールは【/opt/Embedix/tools】ディレクトリ以下にインストールされます。また、RPM のインストールは root で行います。root でログインしてインストールするか、su コマンドで root になった上でインストールしてください。

```
$ su[Enter]
Password:  (root のパスワードを入力) [Enter]
$ whoami[Enter]
root
$
```

上記操作のように whoami コマンドで root と表示されれば、root 権限になっています。

1.2.2 Qtopia のセットアップ

SL ザウルス用の Qtopia アプリケーションソフトは、C/C++プログラミング言語、及び、TrollTech 社の Qt を用いて開発します。Qtopia は、SL ザウルスを持っていなくても PC-Linux の x11 上でアプリケーションソフトのテストをするための仮想フレームバッファ(qvfb)に対応しています。SL ザウルス上（または qvfb 上）でアプリケーションソフトを動作させるには、ビルド時に Qtopia や Qt のライブラリをリンクする必要があります。

Qtopia アプリケーションソフトを開発するために必要な QtopiaSDK として、ここでは無償版の QtopiaSDK（GPL SDK Linux386）へのリンクを示しています。この無償版は GPL に基づいて提供されていますので、このツールで開発したアプリケーションソフトは GPL に基づいて公開する必要があります。

http://www.trolltech.com/developer/download/qtopia.html

（qtopia-free-1.5.0-1.i386.rpm を入手）

商用ソフトの開発を行う場合、有償版の SDK（Qtopia commercial SDK）を入手してください。

https://www.regnow.com/softsell/nph-softsell.cgi?item=7131-1

GPL については、以下の URL を参照してください。

http://www.gnu.org/home.ja.html

QtopiaSDK をインストールするには、クロスコンパイラと同じように RPM ツールが必要です。初期設定では、Qtopia SDK は【/opt/Qtopia/】ディレクトリ以下にインストールされます。

1.2.3　シャープ用 tmake 設定ファイル

tmake とは、Trolltech 社から提供されているソフトウェアプロジェクトの make ファイル管理ツールです。クロスコンパイル環境においては複数のターゲットや複数のコンパイラ別に make ファイルを管理しますが、この tmake を使用することにより、make ファイルの管理が非常に簡単になります。設定用のファイル入手先は、別途宝箱 Pro の次の URL から参照できます。

http://more.sbc.co.jp/sl_j/tool/tools.htm#Linux

次のものを入手してください。

http://more.sbc.co.jp/slj/tool/tmake-sharp.tar.gz　tmake 設定ファイル for SL シリーズ

入手した tar.gz ファイルを次の操作でインストールします。

```
$ gzip –d tmake-sharp.tar.gz[Enter]
$ tar xvf tmake-sharp.tar[Enter]
$ find linux-sharp-g++ | cpio –amp /opt/Qtopia/tmake/lib/qws[Enter]
```

上記操作の結果、カレントディレクトリの linux-sharp-g++ディレクトリ以下にある SL シリーズ用の tmake 設定ファイルは【/opt/Qtopia/tmake/lib/qws/linux-sharp-g++】ディレクトリ以下にインストール（コピー）されます。

1.3　コンパイル環境設定用スクリプトファイル

クロスコンパイラ、Qtopia SDK、SL シリーズ用 tmake 設定ファイルをインストールした後に、ユーザーの home ディレクトリ（例えば /home/user1）に以下の 2 つのスクリプトファイルを作成し

ます。ARM ターゲットでSLザウルス用にクロスコンパイルするための環境変数を設定するものと、x86 ターゲットでコンパイルするために環境変数をセットアップするものです。

スクリプトファイルの内容は、次のものを参考にしてください。

```
#!/bin/bash
CROSSCOMPILE=/opt/Embedix/tools
QPEDIR=/opt/Qtopia
QTDIR=/opt/Qtopia
PATH=$QTDIR/bin:$QPEDIR/bin:$PATH:/opt/Embedix/tools/bin
TMAKEPATH=/opt/Qtopia/tmake/lib/qws/linux-x86-g++/
LD_LIBRARY_PATH=$QTDIR/lib:$LD_LIBRARY_PATH
export QPEDIR QTDIR PATH TMAKEPATH LD_LIBRARY_PATH PS1
echo “Altered environment for sharp Zaurus Development x86”
```

X86 ターゲット用スクリプト例「dev-x86-qpe.sh」

```
#!/bin/bash
CROSSCOMPILE=/opt/Embedix/tools
QPEDIR=/opt/Qtopia/sharp
QTDIR=/opt/Qtopia/sharp
PATH=$QTDIR/bin:$QPEDIR/bin:$CROSSCOMPILE/bin:$PATH
TMAKEPATH=/opt/Qtopia/tmake/lib/qws/linux-sharp-g++/
LD_LIBRARY_PATH=$QTDIR/lib:$LD_LIBRARY_PATH
export QPEDIR QTDIR PATH LD_LIBRARY_PATH TMAKEPATH
echo "Altered environment for sharp Zaurus Development ARM"
```

ARM ターゲット用スクリプト例「dev-arm-qpe.sh」

上記のスクリプトの内容は、使用するシェルにより修正が必要になる場合もありますのでご注意ください。ツールをインストールしたディレクトリがデフォルトではない場合、適宜該当する PATH になるように必ず修正してください。インストールする PATH を変更してしまうと、これ以外にも影響が出てくる可能性があります。また、実行前にスクリプトが実行可能かどうかを以下のコマンドでご確認ください。

```
$ chmod +x dev-arm-qpe.sh[Enter]
```

x86 用アプリケーションソフトをコンパイルしたりテストする際には、dev-x86-qpe.sh を実行してください。逆に、SL ザウルス用にコンパイルする際には、dev-arm-qpe.sh を実行してください。

1.4　インストール結果の確認

以上のツールをインストールした結果、次のようなフォルダ構成になります。このフォルダ構成と異なる場合、各種 PATH を変更しないといけませんのでご注意ください。

■ Qtopia 関連

/opt/Qtopia の下にインストールされます。

■ gcc 関連

/opt/Embedix/の下にインストールされます。

■ コンパイル環境設定用スクリプトファイル

ユーザーのホームディレクトリ。ユーザー名が user1 の場合、/home/user1 にスクリプトファイルを保存します。コンパイル前に、ホームディレクトリのスクリプトファイルを実行して、ターゲット毎に違うコンパイラやライブラリ、tmake 関連の PATH などを設定します。もちろん、別途 PATH の通っているディレクトリに保存してもかまいません。

■ tmake 関連

tmake 自体は Qtopia のツールですので、/opt/Qtopia/bin/tmake にあります。シャープ用の設定ファイルは別途インストールしますが、こちらは /opt/Qtopia/tmake/lib/qws/linux-sharp-g++ にインストールされます。

1.5　コンパイラのテスト

コンパイラをテストするために、QtopiaSDK に同梱されているサンプルアプリケーション example にてテストを行います。以下の手順を実行します。

1.5.1　x86 用コンパイル環境の確認

1. 最初にユーザーの home ディレクトリで x86 環境用スクリプトを実行し、各種環境変数の設定を行います。

```
$ cd /home/user1[Enter]
$ . dev-x86-qpe.sh[Enter]
```

スクリプトファイル名の前の”. “（ドット・スペース）は、設定した内容を現在使用中のシェルに引き継ぐために必要です。ご注意ください。

2. /opt/Qtopia/example/ディレクトリに移動し、tmake コマンドで Make ファイルを作成します。

```
$ cd /opt/Qtopia/example
$ tmake -o Makefile example.pro
```

3. 同じディレクトリで

```
$ make
```

を実行してアプリケーションをビルドします。

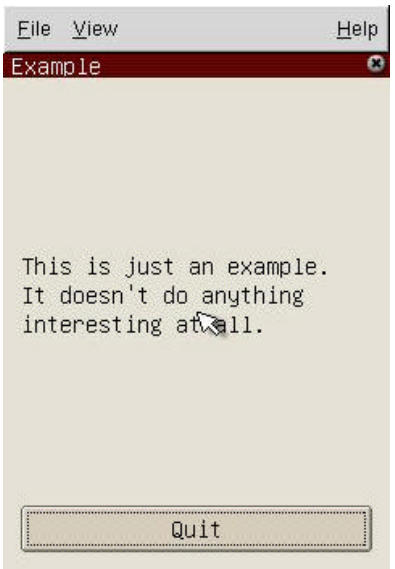
4. コマンドラインから qvfb を起動して、example アプリケーションの動作を確認します。

```
$ qvfb &
```

で qvfb の実行を開始し、

```
$ ./example -qws example
```

アプリケーションをサーバーモードで起動します。以下の画面が表示されれば、成功です。



1.5.2 ARM 用コンパイル環境の確認

1. 最初にユーザーの home ディレクトリで ARM 環境用スクリプトを実行し、各種環境変数の設定を行います。

```
$ . dev-arm-qpe.sh[Enter]
```

スクリプトファイル名の前のドット・スペースを忘れないで下さい。

2. 次に/opt/Qtopia/example ディレクトリに移動します。

```
$ cd /opt/Qtopia/example[Enter]
```

3. x86 環境で作成された一時ファイルおよび Make ファイルを削除します。

```
$ make clean[Enter]
$ rm Makefile[Enter]
```

を実行してください。

4. ARM コンパイル用の Make ファイルを作成します。

```
$ tmake –o Makefile example.pro[Enter]
```

5. このディレクトリで *make* コマンドを実行して、ARM 用バイナリを作成します。

```
$ make[Enter]
```

6. example（ファイル名も同様で拡張子なし）が作成できた後、そのファイルを SL ザウルスの /home/QtPalmtop/bin などにカード経由などでコピーしてターミナルソフトから起動します。コピー方法、インストール方法などはSL ザウルスの取扱説明書をご参照ください。SL ザウルス用のターミナルソフトは、次の URL から入手できます。

http://more.sbc.co.jp/sl_j/tool/tools.htm#Support_Tool

1.6　トラブルシューティング

ここまでの手順でコンパイルができなかった場合のトラブルシューティングです。

1. rpm をインストールできない。
   RPM に対応した Linux ディストリビューションを PC にインストールされていますか？
   root 権限でインストールされていますか？
   HDD の空き容量は足りていますか？
   コマンドは正しく入力されていますか？
2. QtopiaSDK をインストールすると、途中で　libc.so.6　が見つからないというエラーになる。
   Linux のバージョンが古いと発生する可能性があります。これは QtopiaSDK インストール時に必要になるライブラリです。/lib の下に無ければ、バージョンが古いと思われます。Redhat の 7.1 以降などにアップしてください。
3. tmake でプロジェクトファイルを作成しようとすると、../../unix/app.t が見つからないというエラーになる。
   ターゲット別環境設定用のスクリプトファイル実行時に、". "をつけて実行していないと、このエラーが発生する可能性があります。
4. example を実行しようとすると、コマンドが無いというエラーになる。
   カレントディレクトリのコマンドを実行する場合でも、PATH を明示的に指定する必要があります。コマンド名の前に"./"をつけて、現在のカレントディレクトリを指定します。

```
$ ./example –qws example[Enter]
```

5. gcc や arm-linux-gcc などコンパイラが見つからないというエラーになります。

opt/embedix/tools/bin に PATH が通っていないと発生します。別途PATH を設定するか、シェル起動時に読み込まれる.bashrc ファイルなどに設定内容を追加します。

6. example をコンパイルすると、エラーが発生する。
   SL シリーズザウルスの開発で使用する gcc は、バージョン 2.95 です。最新のディストリビューションにはもっと新しい gcc があらかじめインストールされているものがありますが、2.95 以外の gcc は使用できません。このドキュメントに記載されている gcc のバージョン 2.95 をインストールしてご使用ください。

Qt、Qt Deigner (Qt の GUI 設計ツール)、アプリケーション用のプロジェクトファイルの作成方法、qmake、tmake、及び、Qt に関連する Q&A などは、TrollTech 社のサイト:http://doc.trolltech.com や SDK ドキュメンテーションでも確認できます。

Qtopia の SDK ドキュメンテーションは、PC の/opt/Qtopia/doc/index.htmlにあります。そして http://doc.trolltech.comでアップデートされた内容を確認できます。ただし、2002 年 10 月現在、SL ザウルスに搭載されている Qt/Embedded のバージョンは 2.3.2、Qtopia のバージョンは 1.5.0 です。Trolltech 社のサイトには Qt/Embedded のバージョン 3.0.4 が掲載されていますのでご注意ください。

1.7　Qt 関連開発ツールのご紹介

この資料では Trolltech 社の QtopiaSDK について説明しておりますが、それ以外にも Qt 関連開発ツールは販売されています。

1. シャープビジネスコンピュータソフトウェア株式会社（http://www.sbc.co.jp/）
   Trolltech 社様の商用版 QtopiaSDK とコンパイラやライブラリ、各種資料などを 1 枚の CD-ROM にまとめたものを販売されています。この CD-ROM があれば、すぐに開発をはじめていただけます。詳細は以下の URL をご確認ください。

   SL シリーズザウルス用 Qt アプリケーションソフト開発ツール
   QtopiaSDK 通信販売のご案内のページ
   http://www.sbc.co.jp/shop/qtopia/qtopia.html

2. 株式会社コンピューテックス（http://www. computex.co.jp/）
   Windows マシン上で Qt の開発ができる統合開発環境です。Commercial 版（商用版）とスタンダード版（フリーソフト開発用）の 2 種類があります。この開発環境を使用すれば、Linux マシンを別途用意することなく、お手持ちのWindowsPC で開発していただけます。詳細は

以下の URL でご確認ください。

シャープ製 Linux 搭載ザウルス専用統合開発環境の紹介
http://www.computex.co.jp/products/zaurus/

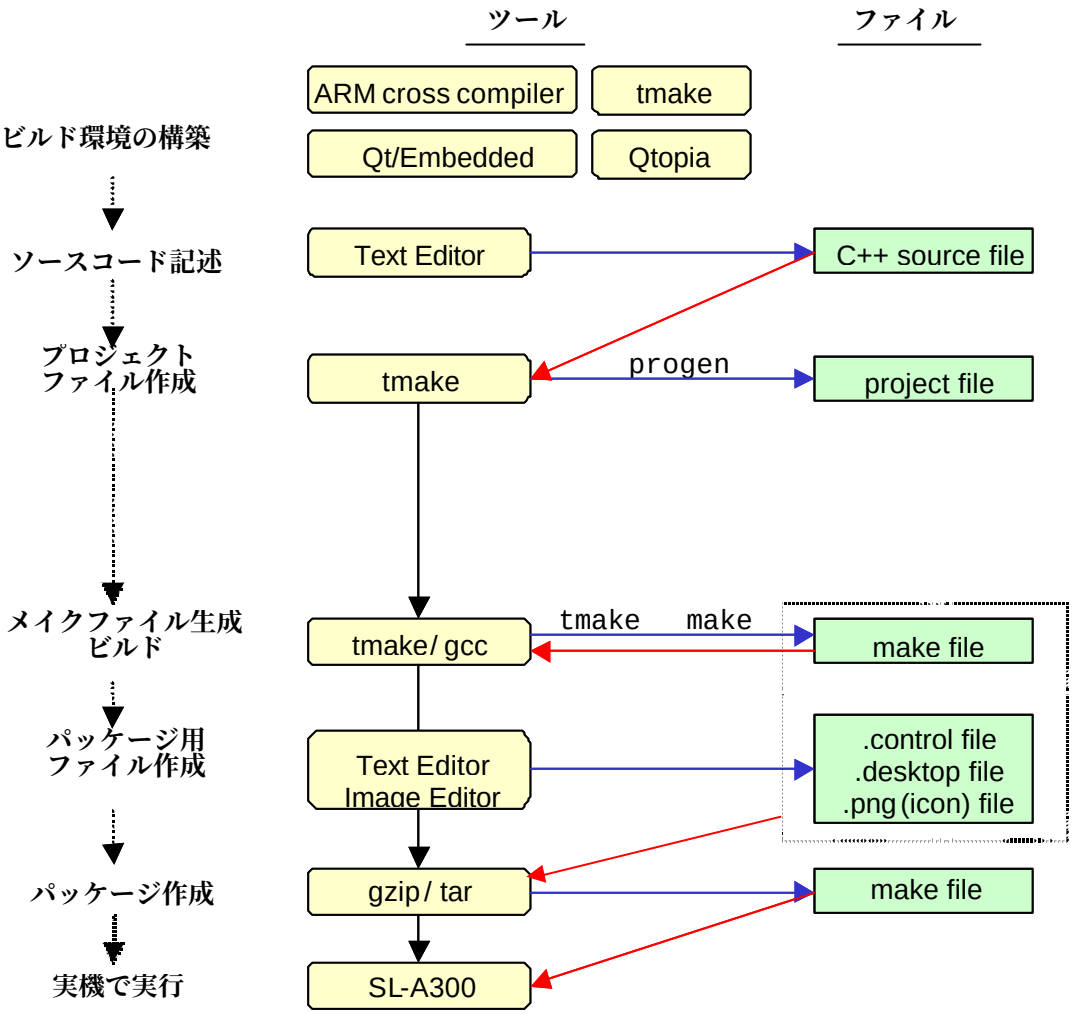
第 2 章 Qtopia アプリケーションの開発手順

この章では、第 1 章でセットアップできた開発環境を使用して、一から Qtopia アプリケーションを開発する手順を解説します。

2.1　開発概略

ここでは、開発の大まかな手順や使用するツール（コマンド）などを説明しています。

2.1.1　開発ステップ



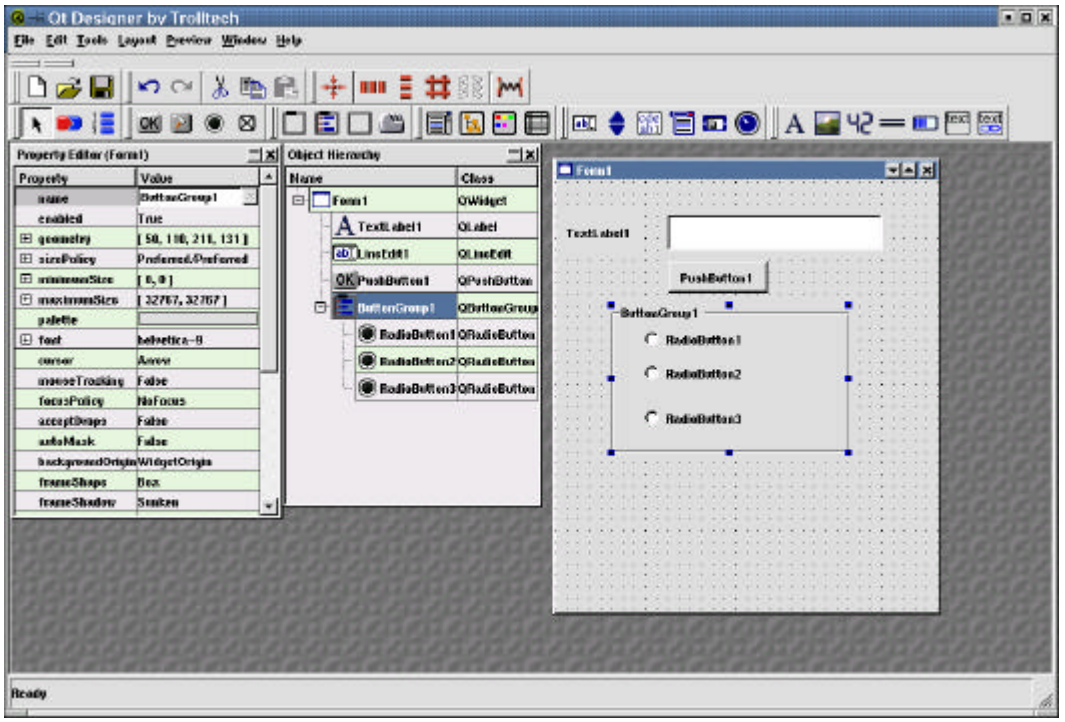
このように第 1 章で説明されているコンパイル環境構築後は、他のプラットフォーム用のプログラム開発と同じように、プロジェクトの作成やソースコーディング、ビルド、インストール用パッケージの作成という手順になります。ただし、現状ではビルドやパッケージの作成などに Linux のコマンドを指定して実行する必要があります。

2.1.2　使用するツール

ここでは、第 1 章のコンパイル環境構築で説明されているツール以外のものについて概要を説明しています。

2.1.2.1QtDesigner とは？

QtDesigner とは、Qt アプリケーションの画面を設計するツールで、【/opt/Qtopia/bin/designer】です。Qt アプリケーションで使用できるリストやボタン、コンボボックスなどのパーツを設計中の画面に貼り付けていき、それらのパーツの属性を設定していくことにより画面を設計していきます。パーツを操作した時の処理である、SIGNAL/SLOT の関係も設定できますので、画面の内容や動作に関する部分は、ほとんど QtDesigner で設計できます。なお、QtDesigner を使用しなくても、コーディングすれば動的にパーツのインスタンスを作成することもできますので、QtDesigner の使用は必須ではありませんが、管理が非常に楽になります。
次の画面は、QtDesigner を使用中の画面例です。



QtDesigner で画面を設計して保存すると、*.ui というファイルが作成されます。このファイルそのままでは c++コンパイラが解釈できませんので、uic コマンドを使用してソースファイルを生成します。

2.1.2.2uic とは？

uic とはユーザーインターフェースコンパイラと呼ばれるツールで、QtDesigner で作成した画面設計情報のファイル*.ui から、c++コンパイラが解釈できるソースファイルを生成するツール

です。基本的な使い方は次のとおりです。

```
$ uic baseform.ui -o baseform.h[Enter]
$ uic baseform.ui -i baseform.h -o baseform.cpp[Enter]
```

1 行目で*.ui ファイルからヘッダファイルを、2 行目で*.ui ファイルと 1 行目で生成したヘッダファイルからソースファイルを生成します。

2.1.2.3moc とは？

moc とはメタオブジェクトコンパイラと呼ばれるツールで、Qt のイベント処理である signal/slot を定義したソースファイルから、c++コンパイラが解釈できるソースファイルを生成するツールです。Qt で提供されているクラスライブラリには、あらかじめ signal や slot も定義されています。クラスの定義部分で signal や slot を含めるには、キーワードとして「Q_OBJECT」「slots」「signals」を使用しますが、このキーワードはc++コンパイラは解釈できません。そのため、moc を使用してソースファイルを生成します。
以下は、これらのキーワードを使用したヘッダファイルの例です。

```
class MyTestClass : public Qobject
{
Q_OBJECT
        ...
signals:
        // シグナル
public slots:
        // public スロット
private slots:
        // private スロット
};
```

moc の基本的な使い方は以下のとおりです。

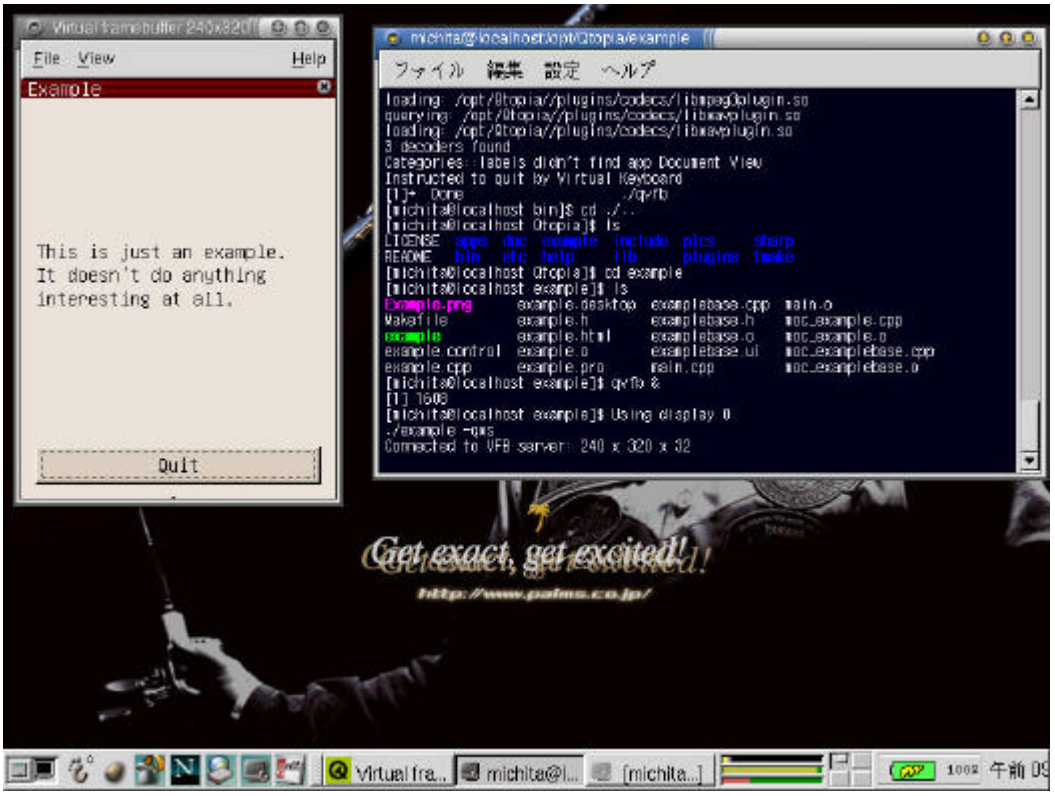
```
$ moc mytestclass.h -o moc-mytestclass.cpp[Enter]
```

上記の操作で、c++コンパイラが解釈できるソースファイルが作成されますので、コンパイル・リンクできます。moc で*.moc ファイルを作成し、その moc ファイルをソースにインクルードするという方法もあります。

2.1.2.4qvfb とは？

qvfb とは、x11 上で SL ザウルス用のアプリケーションソフトをテストするためのシミュレーション用ツールです。既に第 1 章の中で qvfb を使用して example アプリケーションの動作を確認しましたが、このようにターゲットにアプリケーションソフトをダウンロードしなくてもある程度のデバッグができますので、開発効率が大幅に向上します。以下は、Qtopia に付属し

ている example アプリケーションをPC-Linux の qvfb 上で実行した例です。ターゲット用のバイナリではなく、x86 用のバイナリを使用してシミュレーションします。ただし、実機とPC-Linux で相違のある部分（例えばストレージのデバイス名　dev/hdc1　など）については、当然そのままでは実行できませんので、別途ソースファイル中で#ifdef～#endif で区切るなどの対応が必要です。



2.1.2.5progen とは？

progen とは、tmake コマンドで make ファイルを作成する際に必要な、.pro ファイルを生成するためのツールで、【/opt/Qtopia/tmake/progen】にあります。.pro ファイルには make したい.cpp や.h ファイルを記述しますが、新規に作成するプロジェクトやファイルが増えたときなどに手で一つ一つ追加していくことは間違いが発生しやすいものです。progen を使用すれば、ディレクトリ内のソースファイルなどをスキャンして.pro ファイルに登録してくれますので、漏れなどが発生しません。progen で作成した.pro ファイルを tmake で使用して make ファイルを作成するという手順になります。これらの一連の作業をスクリプトで用意しておいてもいいでしょう。

2.1.2.5多言語対応で使用するツール群

その他、多言語対応時に使用するツールとしては、次のものがあります。日本語の文字列を直接コーディングして表示させる場合には必要ありません。

1. findtr

このツールは、QObject::tr("this is a test string.")のように tr()で囲まれている文字列をソースファイル中から探し出し、.po ファイルに出力するツールです。使用方法は次のとおりです。

```
$ findtr test.cpp > test.po[Enter]
```

この.po ファイルには、ヘッダ部分と実際の文字列設定部分が存在します。

ヘッダ部分で重要な点は、文字コードの指定です。

```
"ContentType=  ~  charset=****¥n"
```

という行がありますので、このcharset の指定と.po ファイル自身のエンコードが同じでないと正常に表示されません。

文字列設定部分には、ファイル名と行番号および元の文字列が記載されていますので、その文字列を翻訳した文字列を設定していきます。

```
#: test.cpp:36
msgid "test::this is a test"
msgstr ""
```

上記の msgstr の部分に、翻訳後の文字列を入力します。

```
#: test.cpp:36
msgid "test::this is a test"
msgstr "これはテストです"
```

このようにして作成した.po ファイルを次の msg2qm コマンドで実際に使用する翻訳ファイルに変換します。

2. msg2qm

findtr コマンドで作成した.po ファイルを.qm ファイルに変換するツールです。使用方法は次のとおりです。

```
$ msg2qm test.po > test.qm[Enter]
```

このコマンドで作成した翻訳ファイル.qm を使用するには、アプリケーションの実行開始部分に処理を追加します。

```
int main( int argc, char *argv[] )
{
        QPEApplication app( argc, argv );
        QTranslator  translator( 0 );
        translator.load( "test.qm" );
        app.installTranslator( &translator );
        …
}
```

このように、Qtranslator クラスに翻訳ファイルを読み込み、QApplication クラスにインストールします。ただし、この msg2qm コマンドは、QtopiaSDK には入っていませんので、別途 Qt/Embedded のフリー版に入っているソースから x86 用にビルドする、または Qt/X11 のツールを使用する、などで対応してください。

2.1.3 SL ザウルス用アプリケーションソフトで推奨される内容

ここでは、各 SL ザウルス用アプリケーションで操作や動作などを統一するために推奨される内容を列挙しています。アプリケーションソフト開発者様のご理解ご協力をお願いいたします。

1. Menu キーの動作

   SL ザウルス本体には Menu キーがありますが、このキーを押下した際の動作としては、アプリケーションソフト自体のメニューを開くように、メニューを開いている状態で Menu キーを押下された場合には、開いたメニューを閉じるようにお願いします。また、メニュー項目が複数ある場合には、メニューの移動には横スクロールキーを使用するようにお願いいたします。

2. Cancel/OK キーの動作

   OK キーは、データの選択や画面上の[OK]ボタン押下と同等に処理をお願いいたします。Cancel キーは、ダイアログや画面を閉じる際の画面上の[X]ボタン押下と同等の処理をお願いいたします。

3. VRAM へのアクセス

   描画速度を早くするなどの目的で VRAM に直接アクセスするアプリケーションの場合、機種によって VRAM の扱いが変わった場合に動作しなくなる可能性があります。そのようなことを避けるため、Qt で提供されている

   ```
   QDirectPainter
   ```

   クラスを使用してください。これを利用することで、VRAM への直接描画の処理を機種依存を回避して実装することができます。

   ただし、

   ```
   void QDirectPainter::setAreaChanged(const QRect &);
   ```

   のメソッドは Qt3.0 から追加されていますので、SL ザウルスには実装されていません。ご注意ください。

2.2　実際の開発

ここでは、QtopiaSDK に用意されている example ではなく、一からプログラムを作成する手順を説明していきます。なお、作業用のディレクトリとして、/home/user1/work-dir を使用しますので、

```
# cd /home/user1[Enter]
# mkdir work-dir[Enter]
```

で作成しておいてください。また、user1 のアカウントも作成しておいてください。

2.2.1　ソースファイルの作成

作成した work-dir に、cpp ファイルを作成します。ここでは、おなじみの"Hello World!!"を表示するだけのプログラムにします。

```
#include <qpe/qpeapplication.h>
#include <qlabel.h>

int main( int argc, char *argv[] )
{
        QPEApplication app( argc, argv );
        QLabel *label = new QLabel( "Hellow World!!" ,
                (Qwidget*)0 );
        label->resize( 100, 50 );

        app.setMainWidget( label );
        label->show( );
        return app.exec( );
}
```

2.2.2　プロジェクトファイルの作成

progen コマンドを使用して、.pro ファイルを作成します。tmake ディレクトリに PATH を通すため、以下のコマンドを実行します。/home/user1 ディレクトリに dev-x86-qpe.sh ファイルが存在する場合の例です。

```
# . ../dev-x86-qpe.sh[Enter]
# progen –o qpe-test.pro[Enter]
```

これで、work-dir に存在するソースやヘッダファイルをすべて含むプロジェクトファイルが作成されました。ただし、このままでは不足があり使用できませんので、以下の内容を追加します。

```
DESTDIR = ./
INCLUDEPATH += $(QTDIR)/library
DEPENDPATH += $(QTDIR)/library
TARGET = qpe-test
LIBS += -lqpe
```

ソースファイル以外に、*.ui ファイルもプロジェクトに存在する場合、INTERFACE タグを追加すれば、make ファイル作成時に自動的に make ファイルに処理内容が追加されます。

```
DESTDIR = ./
INCLUDEPATH += $(QTDIR)/library
DEPENDPATH += $(QTDIR)/library
TARGET = qpe-test
LIBS += -lqpe
INTERFACE = qpe-test.ui
```

2.2.3　make ファイルの作成

2.2.2 で作成した qpe-test.pro をもとに、tmake コマンドで make ファイルを作成します。次のコマンドを実行してください。

```
# tmake –o Makefile qpe-test.pro[Enter]
```

これで、main.cpp をビルドするための make ファイルが作成できました。

2.2.4　make の実行

2.2.3 で作成した make ファイルを使用して、プログラムをビルドします。次のコマンドを実行してください。

```
# make[Enter]
```

一度 make コマンドを実行した後は、main.cpp を修正するか、make コマンドで作成された.o ファイルなどビルド生成物を削除しておかないと make できません。そのような場合には、

```
# make clean[Enter]
```

で初期状態に戻したあと、再度 make コマンドを実行してください。

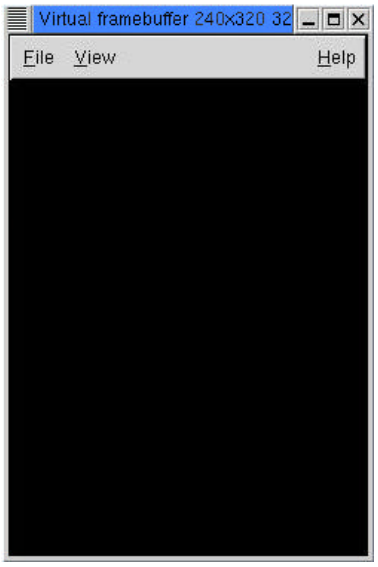
2.2.5　qvfb 上でプログラムを実行

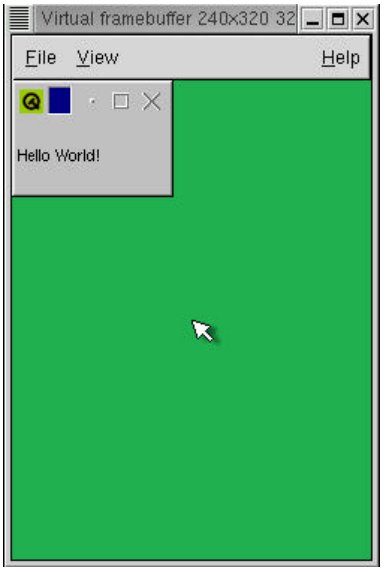
qvfb 上で単体で実行して動作を確認する方法です。とりあえず動作するかどうかを確認する目的に適しています。

2.2.4 で作成した実行ファイルを qvfb 上で実行します。次のコマンドを実行してください。

```
# qvfb &[Enter]
# ./qpe-test -qws qpe-test[Enter]
```

なお、qvfb の起動中に qpe-test を実行してしまうとエラーが発生してプログラムが起動できませんので、ご注意ください。





左が qvfb を起動した画面、右がアプリケーションを起動した画面です。

2.2.6　qpe 上でプログラムを実行

これは、実機上と同じように qpe（Qtopia 環境）を起動してその上で起動する方法で、単体で動作させるよりもさらに実機の状態に近づきます。第 3 章で説明しています desktop ファイルの記述やインストール先の確認もできます。

2.2.6.1各ファイルの配置

PC-Linux 上の qpe 上でアプリケーションソフトを実行するために、以下のようなディレクトリに各ファイルを配置します。desktop ファイルやヘルプファイルの内容については、第 3 章をご確認ください。

ディレクトリ	保存するファイル
opt/Qtopia ── apps/Applications	desktop ファイル
├── bin	実行ファイル
├── pics	アイコンファイル
├── i18n/ja	qm ファイル（翻訳ファイル）
└── help/ja/html	ヘルプ（html ファイル）

2.2.6.2プログラムを実行

qpe も qvfb 上で動作しますので、次のコマンドを実行してください。

```
# qvfb &[Enter]
# qpe[Enter]
```

なお、qvfb の起動中に qpe を実行してしまうとエラーが発生してプログラムが起動できませんので、ご注意ください。

無事 qpe が起動できると次のようにホーム画面が表示されますので、表示されているアプリケーションのアイコンから qpe-test を起動してください。

2.3　ヘルプ対応手順

アドレス帳などの本体に搭載されているアプリケーションには、ウィンドウのタイトルバー部分に[?]ボタンがあり、このボタンを押下するとヘルプが表示されます。この機能は、ヘルプ内容を html ファイルで用意して所定のディレクトリに配置するだけで実現できます。配置するディレクトリは　/opt/QtPalmtop/help/ja/html　です。通常の html ドキュメントと同じように作成しますが、html の charset タグと実際の文字コードを合わせておかないと文字化けしますのでご注意ください。

2.4　日本語対応手順

翻訳ファイルを作成する方法でも日本語に対応できますが、ソースファイル中に直接日本語を記述しても日本語を表示できます。ここでは、直接記述する方法について説明します。
ソースファイル中に直接日本語を記述して日本語を表示するには、QString クラスの fromUtf8() メソッドを使用します。
このメソッドでは引数に指定した UTF8 を Unicode に変換します。
従って fromUtf8 に渡す日本語は UTF8 で書かれていなければなりません。
文字コード変換ツール等を利用して UTF8 で保存して下さい。
以下、ラベルを作成し「ラベル」と表示する場合の例です。

```
QLabel *mylabel= new QLabel( "dummylabel" , this );
mylabel->setGeometry( 10, 10, 100, 30 );
mylabel->setText( QString::fromUtf8( "ラベル" ) );
```

※“ラベル“はUTF8で保存されていますので、通常Linux上で見ると文字化けしています。

この際に注意するのは、QApplication クラスではなく、QPEApplication クラスのオブジェクトを作成することです。QApplication クラスのオブジェクトを使用すると日本語が表示されません。

ヘッダのインクルードは

```
#include <qpe/qpeapplication.h>
```

オブジェクトの生成は

```
QPEApplication myapp( argc, argv );
```

等としてください。

2.5　イベント処理

実際のアプリケーションを組むにはボタンが押されたら何か処理をするといった
イベントの処理が不可欠です。ここでは、イベント処理の方法について説明します。

2.5.1 定義済みシグナルとスロットを使用する

Qt ではシグナルとスロットという仕組みを使ってイベント処理を行ないます。
次の例はボタンを作成し、ボタンが押されるとアプリケーションを終了する処理です。

```
QPushButton *quitbutton = new QPushButton( "quit", this );
connect( quitbutton, SIGNAL(clicked()), qApp, SLOT(quit() );
```

1 行目が【quitbutton】ボタンを作成する処理です。
2 行目が quitbutton がクリックされるとアプリケーションを終了させる処理です。
quitbutton がクリックされると clicked というシグナルが発信され qApp の quit()で設定されているプログラム終了の処理が行なわれます。 qApp は、Qt で定義されたポインタで QApplication オブジェクトをポイントします。
ここで使用しているシグナルのclicked()、スロットの quit() はQt で定義されていますので、記述するだけで動作します。
定義済みのシグナルやスロットはクラスリファレンスをご参照下さい。
例えば、QButton クラスのシグナルには、pressed, released, clicked, toggled,stateChanged があります。QLabel クラスのスロットには setText,setPixmap,SetPicture 等があります。

2.5.2 スロットを自作する

定義済みのシグナルやスロットでは希望の処理が行なえない場合、自分でシグナルやスロットを作成することもできます。
シグナルを作成するケースは少ないと思われますので略します。書籍などをご参照下さい。

自分でスロットを作成する場合は次の処理が必要です。
・クラス定義へのスロット定義追加
・スロットの作成
・シグナルとスロットの接続
・moc(メタオブジェクトコンパイラ)を通す

2.5.2.1クラス定義への追加

自分でスロットを作成する場合は、通常のクラス定義内に Q_OBJECT と記述します。
更に作成するスロットの定義も行ないます。
これは public slots:あるいは private slots:にスロット名を記述します。

```
class myMainWindow  :  public Qwidget
{
 Q_OBJECT
  public:
        myMainWindow();
  public slots:
        void testSlot();
  private:
        Qlabel  *mylabel;
];
```

2.5.2.2スロットの作成

他の関数と同じ要領で作成します。

2.5.2.3シグナルとスロットの接続

connect メソッドを使ってシグナルとスロットをつなぎます。

```
connect( mybutton, SIGNAL(clicked()), this, SLOT( testSlot()) );
```

mybutton がクリックされると自作の testSlot が動作します。

2.5.2.4moc (メタオブジェクトコンパイラ)

シグナルとスロットという仕組みは Qt 独自の仕組みなので、コンパイルには注意が必要です。
moc（メタオブジェクトコンパイラ）というツールを使って、クラスの定義がされているファイルを C++コンパイラが認識できるコードに変換する処理を行います。
moc については　[2.1.2.3　moc とは？]をご参照下さい。
なお、moc の処理は Makefile を使ったコンパイルを行なうのであれば、プロジェクトファイルにファイルを指定するだけで行なわれます。
次のコードはプロジェクトファイル（.pro）の例です。このファイルを元に tmake を使用して Makefile を作成し、コンパイル･リンクできます。
Makefile に moc の処理を行うよう記述されますので、別途 moc を通す必要はありません。

```
TEMPLATE     = app
CONFIG       = qt warn_on release
HEADERS      = tut1.h
SOURCES      = tut1.cpp
INTERFACES   =
INCLUDEPATH  += $(QPEDIR)/include
DEPENDPATH   += $(QPEDIR)/include
TARGET       =  tut1
LIBS         += -lqpe
```

2.5.3　日本語表示及びスロットの簡単な例

以下に、日本語表示と QUIT の処理、自作のスロットの機能を持たせたプログラムを示します。「ボタン」を押すと「Hello zaurus world」のラベルが「ボタンが押されました」に変わります。また、「quit」ボタンでアプリを終了します。

このプログラムは tut1.h,tut1.cpp,tut1.pro のファイルから構成されています。
以下リストを記載します。

tut1.h

```
#include <qlabel.h>
#include <qpushbutton.h>

  class myMainWindow:public QWidget
  {
    Q_OBJECT
    public:
      myMainWindow();

    public slots:
      void changeLabelSlot();

    private:
      QLabel        *mylabel;
      QPushButton    *quitbutton;
      QPushButton    *mybutton;
  };
```

tut1.cpp　(＊UTF8 の文字コードで保存します。)

```
#include <qpe/qpeapplication.h>
#include <qwidget.h>
#include <qlabel.h>
#include <qpushbutton.h>
#include "tut1.h"

void myMainWindow::changeLabelSlot()
{
  mylabel->setText( QString::fromUtf8("ボタンが押されました"));
}

myMainWindow::myMainWindow()
{
  setGeometry( 0, 0, 240, 320 );
  mylabel = new QLabel( "Hello zaurus World", this );
  mylabel -> setGeometry( 50, 80, 160, 30 );
  mybutton = new QPushButton( "button", this );
  mybutton -> setGeometry( 70, 140, 100, 30 );
  mybutton -> setText( QString::fromUtf8("ボタン") );
```

```
   quitbutton = new QPushButton( "quit", this );
   quitbutton -> setGeometry( 180, 5, 50, 30 );

   connect( quitbutton, SIGNAL( clicked()), qApp, SLOT( quit()) );
   connect( mybutton, SIGNAL( clicked()), this,
        SLOT( changeLabelSlot()) );
}


int main( int argc, char** argv )
{
   QPEApplication myapp( argc, argv );
   myMainWindow mywidget;
   myapp.setMainWidget( &mywidget );
   mywidget.show();
   return myapp.exec();
}
```

tut1.pro

```
TEMPLATE      = app
CONFIG        = qt warn_on release
HEADERS       = tut1.h
SOURCES       = tut1.cpp
INTERFACES    =
INCLUDEPATH   += $(QPEDIR)/include
DEPENDPATH    += $(QPEDIR)/include
TARGET        =  tut1
LIBS          += -lqpe
```

2.6　QtDesignerを使った開発

QtDesigner を使用した場合は、上記とは多少異なる開発手順となります。QtDesigner で画面設計を行ない、その保存ファイル(*.ui)からソースファイルとヘッダファイルを生成します。生成されたファイルに画面設計以外の処理を加えてコンパイル・リンクする流れとなります。以下、画面の設計を QtDesigner で行ない、その他の処理は自分で記述する手順を示します。
※QtDesigner については[2.1.2.1 の QtDesigner とは?]をご参照下さい。

1. QtDesigner（/opt/Qtopia/bin/designer）を起動します。
   Linux のディストリビューションによっては、QtopiaSDK に収録されている QtDesigner よりも新しいバージョンがインストールされていることもありますので、QtopiaSDK1.5.0 に付属している QtDesigner が起動されるよう、必ず上記のパス（QtopiaSDK をインストールしたパス）の QtDesigner を起動してください。

2. QtDesigner 上で画面を作成します。
   form1.ui ファイルに保存します。ボタンやリストなどのパーツを貼りつけて画面を作成します。日本語を表示したい箇所があれば、プロパティエディタで日本語を設定します。

3. main.cpp を作成します。
   QtDesigner を使用して画面を設計するとクラス（この例では Form1）が作成されます。QtDesigner のプロパティエディタの name に入れた名称がクラス名です。main.cpp では、7 行目のように、そのクラスのオブジェクトを作成して下さい。また、2 行目のように作成されたクラスのヘッダファイルをインクルードするよう記述して下さい。

   main.cpp

```
1     #include <qpe/qpeapplication.h>
2     #include "Form1.h"
3
4     int main( int argc , char **argv )
5     {
6             QPEApplication  myapp(argc, argv);
7             Form1  w;
8             myapp.showMainWidget( &w );
9             return myapp.exec();
10    }
```

4. プロジェクトファイルを作成します。
   form1.ui と main.cpp がある状態で progen コマンドを実行して下さい。
   `$ progen  -o  demo.pro[Enter]`

生成された tut2.pro には SOURCES タグに main.cpp、INTERFACES タグに form1.ui が記述されています。更に INCLUDEPATH タグ以降の項目を追加して下さい。

demo.pro

```
TEMPLATE      = app
CONFIG        = qt warn_on release
HEADERS             =
SOURCES             = main.cpp
INTERFACES    = form1.ui
INCLUDEPATH   += $(QPEDIR)/include
DEPENDPATH    += $(QPEDIR)/include/qpe
TARGET        = demo
LIBS          += -lqpe
```

5. tmake コマンドを使用して Makefile を作ります。
   `$ tmake -o Makefile demo.pro[Enter]`

6. make します。
   プロジェクトファイルの INTERFACES タグに form1.ui ファイルを指定して make すると、form1.ui ファイル を元にソースファイルとヘッダファイル等が作成されます。

7. Form1 クラスに画面設計以外の処理を記述します。
   make で生成されたソースファイルやヘッダファイルにスロットの処理など必要な内容を記述して下さい。

8. Designer で日本語を入力した箇所があれば次のように変更して下さい。
   ソースファイルの日本語処理部分は次のようになっていますが、文頭の tr を削除し保存し直して下さい。
   `tr( QString::fromUtf8("utf8 保存なので Linux では化けている"))`
   ↓
   `QString::fromUtf8("utf8 保存なので Linux では化けている")`

9. プロジェクトファイルを修正します。
   HEADERS に Form1.h、SOURCES に main.cpp と Form1.cpp を指定し、INTERFACES の form1.ui ファイルの記述は削除します。INTERFACES に ui ファイルを指定したまま tmake コマンドで Makefile を作成すると、make する度に.ui ファイルからソースファイルを再度生

成してしまい、8 で変更した内容が消えてしまいます。8 で記述した処理が消去されないように、プロジェクトファイルを修正します。

demo.pro(修正後)

```
TEMPLATE      = app
CONFIG        = qt warn_on release
HEADERS       = form1.h
SOURCES             = main.cpp form1.cpp
INTERFACES    =
INCLUDEPATH   += $(QPEDIR)/include
DEPENDPATH    += $(QPEDIR)/include/qpe
TARGET        = demo
LIBS          += -lqpe
```

10. 再び tmake コマンドで Makefile を作成し、make します。
    Designer で画面を変更した場合は、再度プロジェクトファイルの作成から実行してください。

上記で説明している手順で、とりあえず QtDesigner で作成した画面を表示させることができますが、画面デザインを変更するたびにプロジェクトファイルや uic で生成されるソースファイルを修正しなくてはなりません。実際には、別途 uic で生成されるソースに記述されているクラス（この例では　Form1 クラス）を継承したサブクラスを宣言し、実際の処理や修正などはサブクラス側に実装するようにしたほうがいいでしょう。

この場合の手順としては、上記手順 3 で main.cpp と同じように、QtDesigner で作成したクラスを継承したサブクラス用のソースファイルを作成します。手順 7、8 で form1 と記載している部分は、サブクラス側で実装します。よって、手順 9 は不要になります。

第 3 章 アプリケーションのインストール

この章では、第 2 章で作成したアプリケーションソフトを SL ザウルスにインストールする手順について解説します。

3.1　ipkg ファイル

ipkg とは、RPM のようなパッケージングシステムで、組み込み機器などに適したものです。詳細は、http://www.handhelds.org/z/wiki/iPKG　でご確認ください。

3.1.1　ディレクトリの作成

ipkg ファイルを作成するためには、まずインストールする実機上のディレクトリ構造と同じものを PC-Linux 上に作成します。SL ザウルスでは、Qt アプリケーションは【opt/QtPalmtop】にインストールしますので、用意するディレクトリ構造は次のとおりです。

ディレクトリ	保存するファイル
Work-dir ── CONTROL	coontrol ファイル
Work-dir ── opt/QtPalmtop ── apps/Applications	desktop ファイル
Work-dir ── opt/QtPalmtop ── bin	実行ファイル
Work-dir ── opt/QtPalmtop ── pics	アイコンファイル
Work-dir ── opt/QtPalmtop ── i18n/ja	qm ファイル（翻訳ファイル）
Work-dir ── opt/QtPalmtop ── help/ja/html	ヘルプ（html ファイル）

work-dir 以下には、CONTROL および opt ディレクトリ以外のファイルを保存しないで下さい。また、CONTROL ディレクトリには、control というファイルだけを保存します。

3.1.2　control ファイルの作成

control ファイルとは、このパッケージの内容を各種定義しているファイルです。インストーラーはこのファイルの情報を使用して、ソフトウェアのインストールを行います。設定できる項目は主に次のとおりで、必須項目は★印をつけています。

1. Package★

   パッケージの名前を指定します。使用できるキャラクタは、

   a-z、0-9、.、+、-

   です。

2. Files

パッケージに含まれるファイルを列挙します。通常はプログラムファイル、アイコンファイル、desktop ファイルを指定します。

3. Priority
   このパッケージの種類を指定します。通常は「optional」を指定します。
4. Section
   適宜指定します。Qtopia アプリケーションは「Qtopia」を指定しておくといいでしょう。
5. Maintainer★
   このパッケージの管理者（作成者）およびメールアドレスを記載します。
6. Architecture★
   このパッケージのソフトウェアが動作するプラットフォームのアークテクチャを指定します。SL ザウルス用なら「arm」です。
7. Version★
   このパッケージのバージョンを指定します。適宜$Qtopia_VERSOIN-10 などのように指定します。バージョンにもルールがあり、「ソフトウェアのバージョン」-「パッケージのバージョン」というかたちで設定します。0.33-1 などのようになります。
8. Depends
   依存するパッケージを指定します。
9. Description★
   簡単な説明を記載します。

設定ファイル例です。

```
Package: qpe-test
Installed-Size:3k
Version:0.1-1
Depends:qpe-base (1.5.0)
Priority:optional
Section:Qtopia
Maintainer:foo <foo@foo>
Architecture:arm
Description:test application
 The test application for the Qtopia.
```

Packege 項目に設定する内容には、_（アンダースコア）は使用できません。また、':'の後ろに一つ以上のスペースもしくは TAB を入れてください。

3.1.3 desktop ファイルの作成

desktop ファイルとは、このパッケージの内容を各種定義しているファイルです。設定すべき

内容は次のとおりです。なお、このファイルに 2 バイト文字を設定する場合、文字コードは UTF-8 で入力しておく必要があります。

1. Comment
   簡単な説明です。
2. Exec
   プログラムファイル名です。
3. Icon
   ホーム画面に表示するアイコンのファイル名です。拡張子は除きます。
4. Type
   種類を指定します。通常は「Application」です。
5. Name
   ホーム画面に表示するアプリケーション名です。日本語で表示する場合には、タグ名を Name[ja]にしてタイトルそのものを utf-8 で入力しておきます。

設定ファイル例です。

```
[Desktop Entry]
Comment = A File Manager Program
Exec = example
Icon = example
Name = exapmle
Type = Application
```

Exec や Icon に設定するプログラムファイル名やアイコンファイル名は、すべて本当のファイル名と同じでないとインストールや起動に失敗します。なお、Icon ファイルは PNG で作成します。画像のサイズは任意でかまいません。大きいと自動的に縮小して表示されます。

3.1.4 ipkg ファイルの作成

3.1.1 で説明されているディレクトリ構造をワークディレクトリ下に用意し、その各ディレクトリに必要なファイルを配置した後、ipkg ファイルを作成します。

ipkg を作成するための便利なツールが、ipkg-build です。次の URL から入手できます。
http://ipkgfind.handhelds.org/result.phtml?section=base（ipkg find）
http://ipkgfind.handhelds.org/details.phtml?package=ipkg-build&official=&format=（ipkg-build）

このコマンド（実際にはスクリプトファイル）を/usr/bin などの PATH の通ったディレクトリに解凍します。ipkg-build の使い方は次のとおりです。

```
$ cd /home/user1[Enter]
$ ipkg-build work-dir[Enter]
```

上記の操作で、/home/user1/work-dir 以下が丸ごと ipkg ファイルに含まれるように作成されます。

3.2　インストール

3.1 で作成できた ipkg ファイルを実際に SL ザウルスにインストールします。

3.2.1　SL ザウルスへの ipkg ファイルの転送

ここでは、作成した ipkg ファイルを SL ザウルスに転送する方法について説明しています。ご使用のシステムや PC に適した方法で転送してください。

3.2.1.1メモリカード経由

SL ザウルス本体には SD カードスロットがあります。また、コミュニケーションアダプター CE-JC1 を装着すれば、CF カード Type2 のスロットも装備されていますので、CF メモリカードも使用できます。PC から、このカードの適当なディレクトリに ipkg ファイルをコピーします。ただし、カードのファイルフォーマットは FAT にしておく必要があります。

カードへのファイルコピー方法は次のとおりです。事前に/mnt/card ディレクトリの作成やカードのデバイス名（例では/dev/hdc1）を確認しておいてください。

```
$ mount –t vfat /dev/hdc1 /mnt/card[Enter]
$ cp qpe-test_0.1-1_arm.ipk /mnt/card[Enter]
```

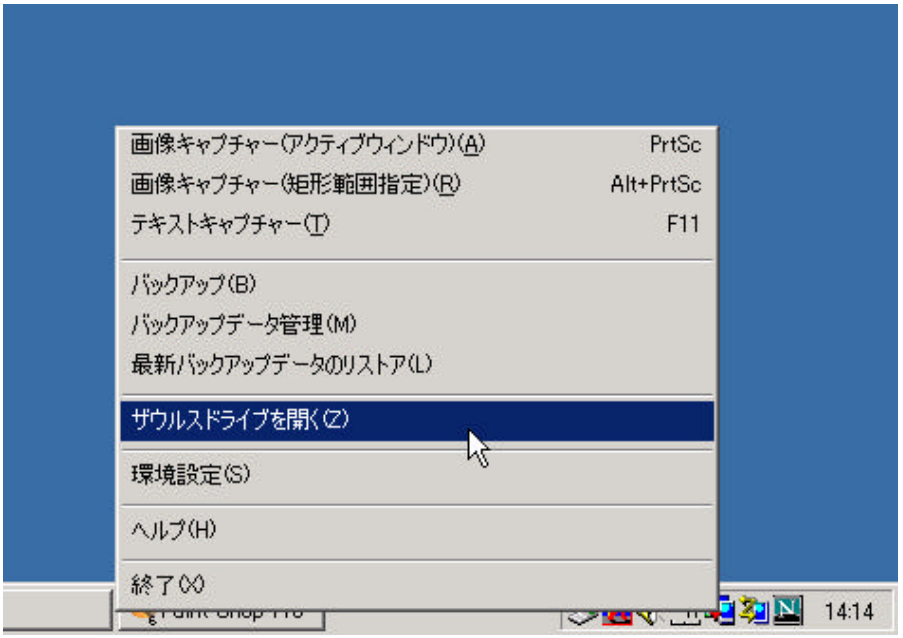
カードを取り外すには、次のコマンドを実行します。完全にファイルにデータを出力させるために、取り外す前に sync コマンドも実行しておきます。

```
$ sync[Enter]
$ umount /mnt/card[Enter]
```

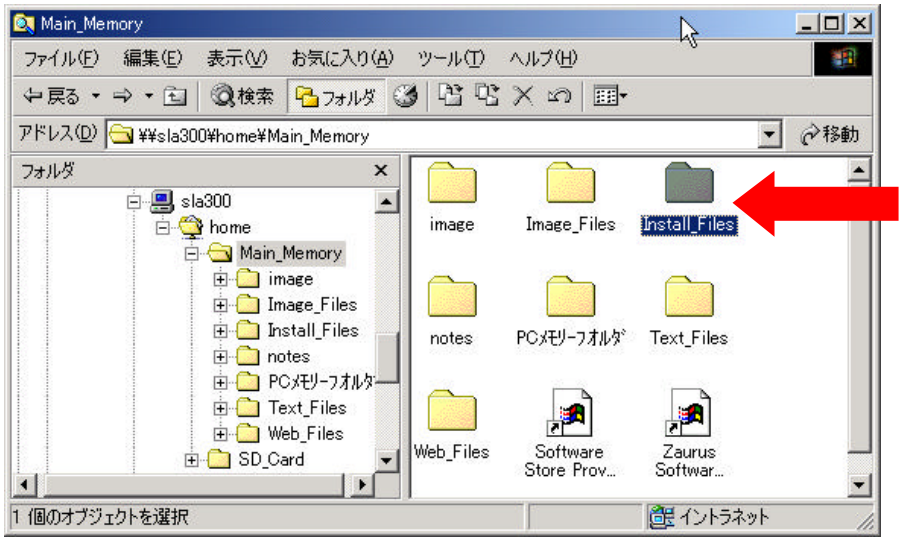
3.2.1.2ザウルスドライブ経由

SL ザウルスに同梱されている“ザウルスドライブ”というソフトウェアを使用すれば、WindowsPC と SL ザウルスを SL ザウルスに付属している USB ケーブルで接続し、簡単にファイルをコピーすることができます。ザウルスドライブのインストール方法や使用方法の詳細は、取扱説明書をご確認ください。

ザウルスショットをインストールすると、タスクバーに「ザウルス通信マネージャー」「ザウルスショット」のアイコンが常駐します。ザウルスショットのアイコン上でマウスを右クリックすると以下のメニューが表示されますので、「ザウルスドライブを開く」を選択します。

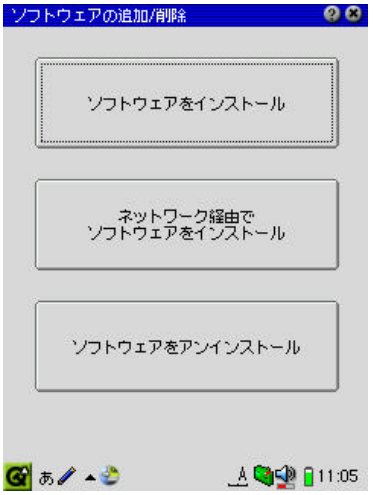


ザウルスドライブを起動すると、SL ザウルスの home ディレクトリが表示されますので、Main_Memory¥Install_Files に ipkg ファイルをドラッグ&ドロップしてコピーします。



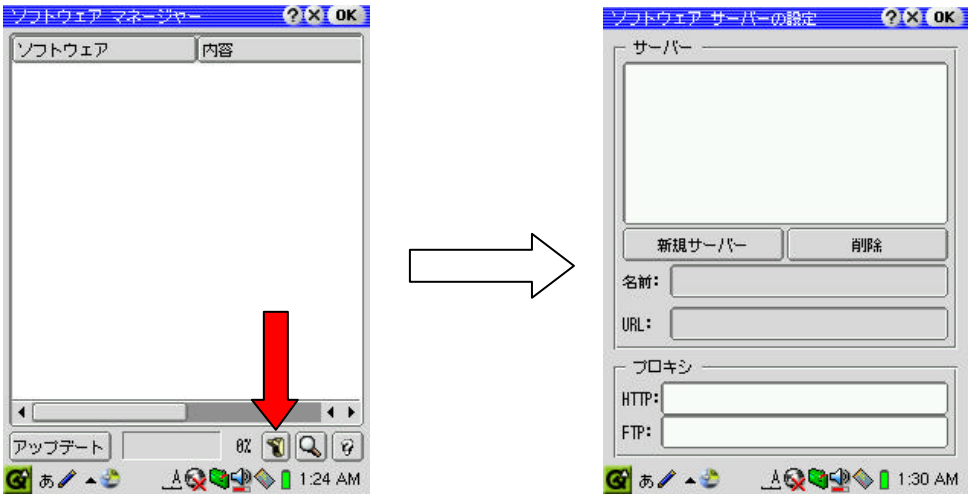
3.2.1.3ネットワーク経由

SL ザウルスの設定画面から"ソフトウェアの追加/削除"を起動すると、画面上に「ネットワーク経由でインストール」というボタンが表示されています。



このボタンからネットワーク上に存在する ipkg ファイルを直接インストールすることができます。インストールそのものの操作などは 3.3 を参照いただきますが、ここではネットワーク上のサーバーに接続できるように設定する方法を解説します。

「ネットワーク経由でソフトウェアをインストール」を選択すると、「ソフトウェアマネージャー」画面が表示されますので、この画面からサーバーを設定/選択するアイコンを選択し、次に表示される「ソフトウェア　サーバーの設定」画面で、接続するサーバーを設定します。



この「ソフトウェア　サーバーの設定」画面で、接続するサーバーの名前（任意）URL、接続に使用するプロキシサーバーなどが必要な環境の場合、http や ftp に使用するプロキシサーバー

の URL を設定します。設定後、[OK]で画面を閉じます。なお、プロキシサーバーのポート番号の指定は、「proxy.mycorp.co.jp:8080」のように指定します。詳しくは社内ネットワーク管理者の方、または ISP の窓口にご確認ください。

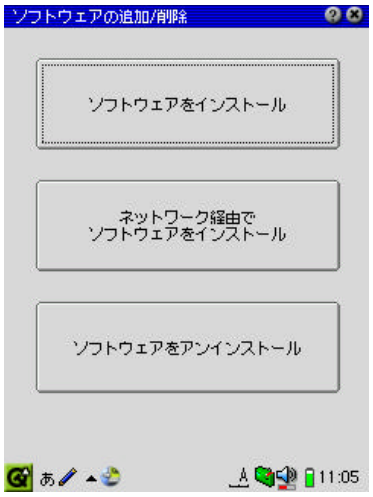
「ソフトウェア　サーバーの設定」画面で[OK]を押下すると、上記のように、そのサーバーに存在する ipkg ファイルがリストされます。リストされない場合は、サーバーの設定を再確認していただく、[アップデート]ボタンを押下していただくなどご対応ください。リストされている ipkg ファイルをダブルタッチすると、Package の詳細が表示されますので、確認後「インストール」ボタンを押下します。あとの操作方法は、3.3 をご確認ください。

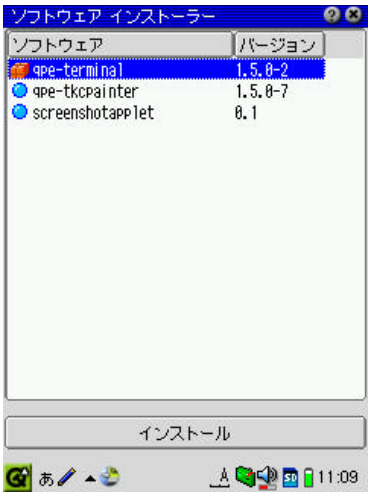
3.3　SL ザウルスのインストール/アンインストール操作

3.3.1 ソフトウェアの追加/削除

設定ホーム画面にインストーラーとして"ソフトウェアの追加/削除"があります。これによりアプリケーションソフトのインストール、アンインストールができます。このインストーラーは、本体メモリやSD カード/CF カード上にある ipkg ファイルを検索して表示します。さらに、本体メモリの他に SD カード、または CF カードにインストールすることもできます。

この[ソフトウェアの追加/削除]を起動するとメニュー画面が表示されますので、一番上の「ソフトウェアをインストール」ボタンを押下し、インストールできるソフトウェアの一覧を確認します。





ソフトウェアの一覧表示で、各ソフトウェアの左のアイコンが、インストール可能かインストール済みなのかを表しています。

赤い箱のようなアイコンは、インストール可能なもの、青くて丸いアイコンはインストール済みのものです。ソフトウェアをインストールする場合には、赤いアイコンのものを選択して、画面下部の「インストール」ボタンを押下します。アンインストールする場合には、メニュー画面で「アンインストール」ボタンを押下して表示される画面で操作してください。

インストールを選択すると、インストール先を選択するダイアログが表示されますので、適当なインストール先を選択します。ソフトウェアによっては、カードにインストールできないものもありますので、その場合は本体にインストールするようにしてください。

Ipkg ファイルについての FAQ が公開されています。宝箱 Pro の以下のページをご確認ください。
http://more.sbc.co.jp/sl_j/faq/a300_faq_ipk.htm

（SL-A300　アプリケーションソフトパッケージ（ipkg ファイル）に関する FAQ）

ダイアログの「OK」ボタンを押下するとインストールが始まります。画面の指示どおり、[ソフトウェアの追加/削除]を終了させてください。また、ソフトウェアによっては、SL ザウルスの再起動が必要になるものもありますので、ソフトウェアの説明書をご確認ください。

3.3.2　コマンドライン

3.3.1 の方法で ipkg をインストールできますが、インストール中のエラー内容の詳細については確認できません。インストール中の問題について内容を確認したい場合には、別途 ipkg コマンドを使用して、実行中のログを確認します。

```
$ ipkg install qpe-test_0.1-1_arm.ipk[Enter]
```

空きメモリが不足している場合のログは以下のとおりです。

```
# ipkg install qpe-test_0.1-1_arm.ipk
Unpacking qpe-test...Done.
Configuring      qpe-test...tar:      ./opt/QtPalmtop/bin/qpe-test:
input/output error -- No space left on device
tar: Bad tar header, skipping
tar: Bad tar header, skipping
tar: Error exit delayed from previous errors
rm: /: is a directory
rm: /opt: is a directory
rm: unable to remove `/opt/QtPalmtop': Read-only file system
rm: /opt/QtPalmtop/bin: is a directory
rm: /opt/QtPalmtop/apps: is a directory
rm: /opt/QtPalmtop/apps/Applications: is a directory
rm: /opt/QtPalmtop/pics: is a directory
#
```

上記のログの最初の部分で「no space left on device」と出ていますので、空き容量が不足していると判断できます。

3.3.3 インストール後の注意

インストール後の注意点として、次のような内容があります。

1. ipkg ファイルが残る
   本体やカードにコピーした ipkg ファイルはインストール後も残っていますので、そのままにしておくと空きメモリを圧迫します。不要な場合は適宜削除してください。

2. カードにインストールしたアプリケーションのアイコン
   カードを抜いている状態でもホーム画面に表示され、アイコンを選択しても起動できませんのでご注意ください。

3. カードにインストールしたアプリケーションの再インストール
   カードを抜いた状態でもインストール済みの状態になります。本体に再度インストールする場合などは、アンインストール後再度インストールしてください。

4. カードにインストールしたアプリケーションの利用
   インストールしたカードを他の SL ザウルスに装着しても利用することはできません。インストール時に使用した SL ザウルスとの組み合わせでのみ使用することができます。

3.3.4 アンインストール

アプリケーションをアンインストールするには、"ソフトウェアの追加/削除"の起動画面から「ソフトウェアのアンインストール」ボタンを押下してリスト表示されるアプリケーションからアンインストールするアプリケーションソフトを選択します。選択した後は、画面の指示に従います。

付録：シャープ株式会社運営サイトの紹介

この章では、シャープ株式会社が運営するサイトのサポートメニューについて紹介しています。

付録 1 ザウルス宝箱

従来の MI ザウルスのころからユーザーが開発されたソフトウェア（MORE ソフト）を掲載しています。MORE ソフトでは、国内最大のダウンロード/リンクサイトです。2002 年 8 月から Linux ザウルス用のソフトウェアを募集開始し、2003 年 1 月 22 日現在、30 本の掲載があります。

URL は次のとおりです。

http://more.zaurusworld.ne.jp/download-sl/slsoft-post-general-1.html（ダウンロードページ）

http://more.zaurusworld.ne.jp/（ザウルス宝箱トップページ）

応募要領は　http://more.zaurusworld.ne.jp/sl-post-method.asp　に記載されています。詳細はこちらでご確認いただきますが、主な内容は、

1. Qtopia アプリ、Java アプリ、など種類を問いません。
2. フリー、シェアウェア、商用を問いません。
3. シャープ株式会社は、投稿されてきたソフトウェアについてサポートや動作確認を行いません。
4. 著作権を侵害している恐れのあるものなど、掲載できない場合もあります。判断はシャープ株式会社が行います。
5. 投稿は、必要事項記入の上、more-post@more.sbc.co.jp　宛てにメールを送付いただきます。

皆様の投稿をお待ちしております。

付録 2 ザウルス宝箱 Pro

この資料が掲載されているサイトでもありますが、このサイトではザウルス用アプリケーションソフトを開発するための技術サポートを行っています。

http://more.sbc.co.jp/（ザウルス宝箱 Pro トップページ）

http://more.sbc.co.jp/sl_j/sl_top.asp（SL ザウルスサポート　トップページ）

SL ザウルス向けのサポート内容としては、

1. 開発ツールダウンロード（リンク）
2. SL ザウルス技術資料（ハードウェア、ソフトウェア）の公開
3. 開発プログラミングガイド、開発環境セットアップガイドの提供
4. Linux ソースコード、ROM イメージのダウンロードサービス

5. 関連情報へのリンク
6. FAQ の公開

などです。今後もさまざまな情報を提供していく予定です。

付録 3 ザウルスサポートステーション

ザウルスの本体やザウルスで使用できる周辺機器についての情報を掲載しているサイトです。

http://zaurus.spacetown.ne.jp/（トップページ）
http://zaurus.spacetown.ne.jp/sl-a300/menu-sla300.asp（SL-A300 のトップページ）
http://zaurus.spacetown.ne.jp/sl-b500/menu-slb500.asp（SL-B500 のトップページ）
http://zaurus.spacetown.ne.jp/sl-c700/menu-slc700.asp（SL-C700 のトップページ）

周辺機器の動作確認状況などはこちらでご確認ください。

付録 4 ザウルスビジネスソリューション

ザウルスの法人向けソリューションのサポートサイトです。

http://ezaurus.com/zbsolution/index.html（ZaurusBusinessSolution）

法人向け導入のご相談や導入事例などについては、こちらをご活用ください。