マニュアル

Algo Smart Display 用Linuxディストリビューション

『Algonomix』について

目　次

はじめに

１）　お願いと注意 …… １

２）　保証について …… １

第１章　概要

１－１ Algonomixとは …… １－１

１－２ Linux OSを使用するメリット …… １－２

１－３ Linuxの仕組み …… １－３

第２章　システム構成

２－１ Algonomix用開発環境について …… ２－１

２－２ Algonomix用開発環境のハードディスクへのインストール …… ２－４

２－３ 他のLinuxに開発環境を入れる方法 …… ２－７

２－４ CFカードの内容 …… ２－８

２－５ CFカードの作成方法 …… ２－１４

２－６ Algo Smart Displayコンフィグプログラムについて …… ２－１７

２－６－１  Network Setting …… ２－１９

２－６－２  Backlight Adjustment …… ２－２１

２－６－３  Volume Setting …… ２－２２

２－６－４  Touch Panel Calibration …… ２－２３

２－６－５  Server Setting …… ２－２５

２－６－６  Setting At Date …… ２－２６

２－６－７  Kernel Update …… ２－２７

２－６－８　Misc. Settings ･･ ２－２８

２－６－９　Hardware Information ･･ ２－２９

２－６－１０　Algonomix Shutdown･･ ２－３０

２－７ Linuxカーネルの復旧 ･･ ２－３２

２－７－１　AP110 の場合･･ ２－３２

２－７－２　AP210/310/315 の場合･･･ ２－３３

第３章　開発環境

３－１ クロス開発環境 ･･ ３－１

３－２ Algonomix開発環境の起動 ･･ ３－２

３－３ WideStudio/MWTによるアプリケーション開発 ･･････････････････････････ ３－４

３－３－１　WideStudioの起動 ･･･ ３－４

３－３－２　プロジェクトの新規作成 ･･ ３－４

３－３－３　アプリケーションウィンドウの作成 ････････････････････････････････ ３－８

３－３－４　部品の配置 ･･･ ３－１０

３－３－５　イベントプロシージャの設定 ･････････････････････････････････････ ３－１２

３－３－６　イベントプロシージャの編集 ･････････････････････････････････････ ３－１３

３－３－７　セルフコンパイル ･･ ３－１６

３－３－８　クロスコンパイル ･･ ３－１７

３－３－９　ファイルの転送 ･･ ３－１８

３－３－１０　WideStudio/MWTの開発例 ･･････････････････････････････････････ ３－２２

３－４ DDDについて ･･ ３－２８

３－５ GDBによるデバッグ方法 ･･ ３－３２

３－６ シリアルコンソールについて ･･ ３－３７

第４章　Algo Smart Display について

４－１ ALGO Smart Displayのデバイスについて ･････････････････････････････････ ４－１

４－１－１　デバイスドライバアクセスについて ････････････････････････････････ ４－１

４－１－２　AP110 のパネルスイッチ･･･ ４－１

４－１－３　AP210/310/315 の汎用入出力･･････････････････････････････････････ ４－９

４－２ その他のデバイスについて ･･ ４－１５
４－２－１　シリアルポート ･･ ４－１５
４－２－２　ネットワークポート ･･･ ４－１８
４－２－３　オーディオ出力 ･･ ４－２４
４－２－４　ウォッチドッグタイマ ･･ ４－２６
４－２－５　RAS機能 ･･ ４－２８
４－３ サンプルプログラム ･･ ４－３５
４－３－１　起動ランチャー ･･ ４－３５
４－３－２　多言語表示 ･･ ４－３７
４－４ Algonomix設定ファイルについて ･････････････････････････････････ ４－３８
４－４－３　/home/asdusr/autostart ･･････････････････････････････････････ ４－３８
４－４－４　/etc/network/interface ･･････････････････････････････････････ ４－３８
４－４－５　/etc/hosts ･･ ４－３８
４－４－６　/etc/resolv.conf ･･ ４－３８
４－４－７　/etc/profile ･･ ４－３９
４－４－８　/etc/inittab ･･ ４－３９
４－５ 動作確認済みUSB機器一覧 ･･ ４－４０
４－６ 起動画面の変更について ･･･ ４－４１
４－６－１　起動画面用の画像について ･････････････････････････････････････ ４－４１
４－６－２　画像バイナリデータの作成 ･････････････････････････････････････ ４－４２
４－６－３　画像バイナリデータの書き込み ･････････････････････････････････ ４－４２

付録

Ａ－１　参考文献 ･･ １

はじめに

この度は、アルゴシステム製品をお買い上げ頂きありがとうございます。
弊社製品を安全かつ正しく使用して頂く為に、お使いになる前に本書をお読み頂き、
十分に理解して頂くようお願い申し上げます。

１）　お願いと注意

本書では、ALGO Smart Display 用 Linux ディストリビューション（以降 **Algonomix**）に特化した部分について説明します。一般的な Linux についての詳細は省略させていただきます。Linux に関する資料および文献は、現在インターネット上や書籍にて大量に出回っています。これらの書籍等と併せて本書をお読みください。

本書を執筆するに当たり、可能な限り誤記のないように努めていますが、間違った表現や使い方および説明をする可能性があります。前もってご了承の上、本書をお使いください。

２）　保証について

Algonomix の動作は現状のバージョンおよび設定でのみ動作確認しております。Algonomix はオープンソース形式で提供されるため、お客様でソースの改変、ライブラリの追加と変更、プログラム設定の変更等を行うことができます。お客様がどのような変更をされるか弊社で予想することは不可能です。そのため、これらの変更を行われた場合は動作保証することができません。変更される際にはすべて自己責任にてお願いします。

第１章　概要

本章では、**Algonomix** の具体的な内容を説明する前に、Algonomix の概要について説明します。

１－１　Algonomixとは

「Linux」というのは、カーネルのみを指す言葉です。Linux カーネルのみでは、オペレーティングシステム（以下 OS）としての役割を果たすことができません。OS として使えるようになるためには、Linux カーネルのほかに、以下のような各種ソフトウェアパッケージと併せて使用する必要があります。

- シェル（bash, ash, csh, tcsh, zsh, pdksh, ・・・・）
- util-linux（init, getty, login, reset, fdisk, ・・・・）
- procps（ps, pstree, top, ・・・・）
- GNU coreutils（ls, cat, mkdir, rmdir, cut, chmod, ・・・・）
- GNU grep, find, diff
- GNU libc
- 各種基本ライブラリ（ncurses, GDBM, zlib・・・・）
- X Window System

Linux カーネルといくつかの必要なソフトウェアパッケージをまとめて、OS として使えるようにしたものを Linux ディストリビューションといいます。

最初に述べましたとおり、「Linux」という言葉は、本来カーネルを指す言葉です。そのため、「カーネルとしての Linux」と「OS としての Linux」を厳密には区別する必要がありますが、本書では「Linux」とは「OS としての Linux」を指す言葉として使用します。

Algonomix は弊社が、ALGO Smart Display シリーズの OS としてまとめた、Linux ディストリビューションの一つです。

Algonomix 用の開発環境イメージを図 1-1-1 に示します。Algonomix 用の開発環境を動作させる PC の Linux OS として「KNOPPIX」と「Red Hat Enterprise Linux WS Ver4」の２通りを推奨しています。

- KNOPPIX ： KNOPPIX とは Debian 系パッケージで、CD のみでブート可能な Linux ディストリビューションです。
- Red Hat Enterprise Linux WS Ver4 ： Red Hat Enterprise Linux はレッドハット株式会社が販売している商用ディストリビューションです。企業での利用を主に考えて作られており，セキュリティ情報などもしっかりサポートしています。

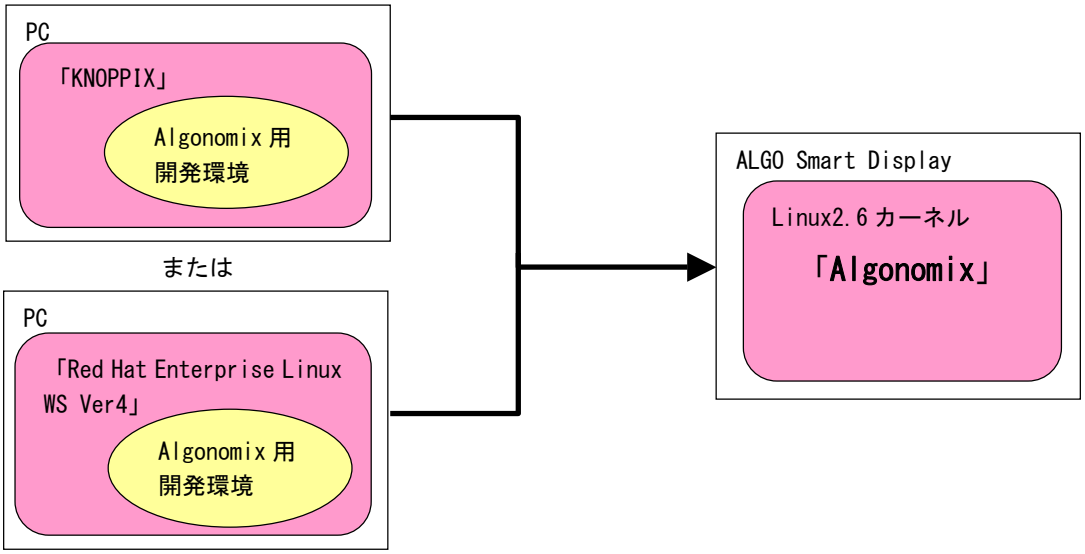

図 1-1-1. Algonomix の開発環境

このマニュアルではKnoppixにAlgonomix用開発環境が組み込こまれたPC Linuxをベースにして説明します。

１－２　Linux OSを使用するメリット

以下に、OSとしてLinuxを選択した理由を記述します。

① オープンソース

オープンソース定義の原本（英語）はOpen Source Initiativeのホームページにかかれています。日本語に訳されたものもインターネット上にアップロードされていますので詳しくはそれらで確認してください。オープンソース定義を要約すると、ソースコードを入手でき、改変ができ、再配布することができるというものです。

Linux は完全にオープンソースであるため、何か問題が生じたときすぐに対処することが可能です。また、お客様のシステムに合わせた改良も容易にできます。ただし、ライセンス上ソースの公開等の制限はつきます。

Algonomix の開発環境を使い、お客様によって作成されたアプリケーションについてはソース開示の義務はありません。

② ロイヤリティフリー

ALGO Smart Displayのように、CFカード、LCD、LAN、USB等のデバイスが搭載されているボードを動作させるためには、核となるRTOSのほかに、それぞれのドライバ、およびファイルシステム、TCP/IP、USBの各種デバイスドライバ等が必要です。

これらのプログラムは本来なら自作する必要がありますが、膨大な時間が必要です。短期間で開発するならそれぞれのミドルウェアを個別に購入する必要があり、また製品ごとにロイヤリティが発生する場合があります。

Linux ではこれらのプログラムはすでに含まれており、ロイヤリティフリーで使用することができます。

③ 安定供給可能

組み込み用途向けの製品は１度開発された後、５年、１０年は同じバージョンで量産されて欲しいものです。しかし、ハードは開発されていても、基本となるOSが販売停止やサポート停止となった場合、新しいOSにて再度動作確認する必要があります。

Linuxにはこれらの制限はありません。

１－３　Linuxの仕組み

Linux のソフトウェア構成を図 1-1-1 に示します。

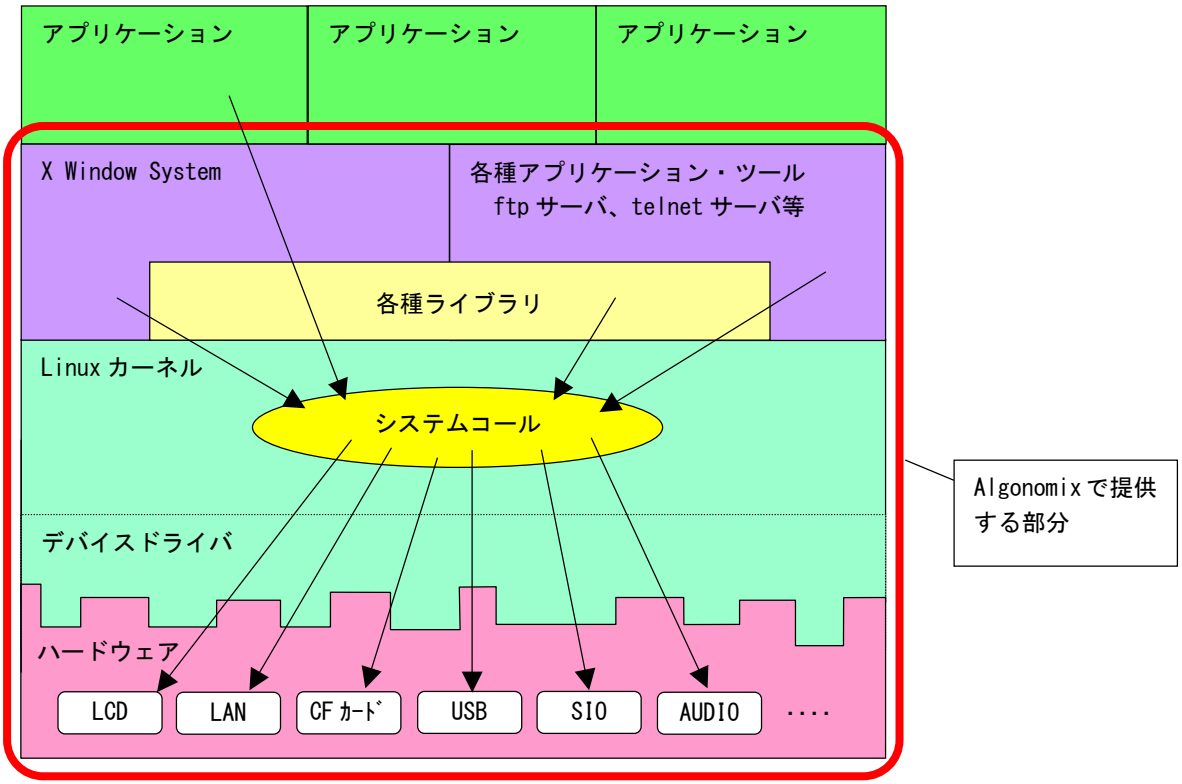

図 1-3-1. Linux ソフトウェア構成図

OS として重要な役割の一つに、**ハードウェアアクセスの複雑さを隠し、統一されたプログラミングインターフェース（システムコールや API と呼ばれる）をアプリケーションに提供する**というものがあります。Linux ではハードウェアを制御するのにドライバに関連付けられた「デバイスファイル」を読み書きすることで制御します。これは UNIX 系 OS の大きな特徴であり、ファイルを扱う感覚でハードウェアを制御することができます。Linux の代表的なシステムコールとして、open, close, read, write 等があります。これらのシステムコールは特別な呼び方をしているわけではなく、関数の呼び出しと同じように呼び出すことができます。

Linux にどのようなデバイスファイルがあるか詳細は次章以降で説明しますが、Linux の GUI 環境である X Window System も各種アプリケーションやツールもこのデバイスファイルを読み書きすることで LCD への描画やイーサネットへのアクセスなどを行っています。お客様で作成されるアプリケーションの中で、ALGO Smart Display のハードウェアを扱いたい場合でも、対象となるハードウェアのデバイスファイルをオープンし、読み書きすることで制御することができます。

もう一つ OS の重要な役割として、**CPU 時間、メモリ、ネットワーク等のリソースをプログラムやプロセス、スレッドに分配する**というものもあります。このあたりは Linux がすべて行ってくれるので、アプリケーション作成時に特に意識する必要はありません。しかし、図 1-1-1 にある X Window System も、telnet サーバや ftp サーバもプロセスの一つです。CPU 時間やメモリなどのリソースは無限ではないため、複数のプロセスを同時に実行すれば、それぞれのパフォーマンスは落ちます。必要最低限のプロセスで実行効率のよいプログラムを作成する必要があります。

第２章　システム構成

２－１　Algonomix用開発環境について

Algonomix 用開発環境は DVD-ROM１枚で提供しています。DVD-ROM をドライブに入れて、PC を起動すると、knoppix という Linux ディストリビューションが起動します。

DVD-ROM は表 2-1-1 のような構成になっています。

表 2-1-1. Algonomix 用開発環境 DVD-ROM の内容

DVD-ROM のディレクトリ	内容
L:¥ALGO¥doc	Algo Smart Display の取扱説明書が格納されています。 ・ASD Software Users Manual.pdf 本書です。 ・ASD Hardware Users Manual.pdf Algo Smart Display のハードウェアについて書かれた詳細マニュアルです。
L:¥ALGO¥packages	knoppix 以外の Linux ディストリビューションに Algonomix 用開発環境をインストールするためのパッケージが格納されています。 ・asd-dev.tgz Algonomix 用開発環境の最低限必要なものがパッケージされています。WideStudio でアプリケーションを開発するにはこれで十分です。 ・asd-dev-src.tgz asd-dev.tgz を構成するためのソースコードがパッケージされています。 **※注：Algonomix 用開発環境のソースコード改変はお客様の自己責任となり、サポートの範囲外になります。** ・asd-package.tgz Algo Smart Display の CF イメージがパッケージされています。
L:¥boot	knoppix X をブートするためのファイルが格納されています。
L:¥KNOPPIX	knoppix 本体です。

knoppix を DVD-ROM から起動する PC の推奨環境を表 2-1-2 に示します。

表 2-1-2. Algonomix 用開発環境（KNOPPIX DVD-ROM Boot）推奨 PC 環境

CPU	Intel(R) Celeron(R) M 1.00GHz 以上
メモリ	256MByte 以上
ドライブ	DVD-ROM（必須）

次に、Algonomix 用開発環境のディレクトリ構成の説明を行います。リスト 2-1-1 に Algonomix 用開発環境のディレクトリ構成を示します。表 2-1-3 にそれぞれのディレクトリの意味を示します。

リスト 2-1-1. Algonomix 用開発環境のディレクトリ構成

```
usr--+--local--+--ws
               +--sh4-linux-dev--+--tools-0010--+--bin
                                                +--bin2
                                                +--include
                                                +--info
                                                +--lib
                                                +--libexec
                                                +--man
                                                +--sh4-linux--+--bin
                                                              +--etc
                                                              +--include
                                                              +--info
                                                              +--lib
                                                              +--libexec
                                                              +--man
                                                              +--sbin
                                                              +--share
                                                              +--usr--+--X11R6--+--bin
                                                                              +--include
                                                                              +--lib
                                                                              +--share
                                                                      +--bin
                                                                      +--etc
                                                                      +--include
                                                                      +--info
                                                                      +--lib
                                                                      +--local--+--include
                                                                                +--lib
                                                                                +--ssl
                                                                                +--ws
                                                                      +--man
                                                                      +--share
                                                                      +--var
                                                              +--var
                                                +--share
                                  +--image--+--dd
                                            +--pacages
home/knoppix/wsproject
```

表 2-1-3. ディレクトリの内容（抜粋）

ディレクトリ名	内容
/usr/local/ws	PC 上で動作する WideStudio 本体のインストールディレクトリです。
/usr/local/sh4-linux-dev/tools-0010 （以降 SH4TOOL とします）	Algonomix 用開発環境の核ともいうべき、SH4 用のコンパイラや SH4CPU で動作する実行ファイルやライブラリが格納されています。
SH4TOOL/bin SH4TOOL/bin2	PC 上で動作する SH4 用のコンパイラやリンカが格納されています。 bin2 は bin のリンクファイルが格納されています。
SH4TOOL/sh4-linux/bin SH4TOOL/sh4-linux/usr/bin	名前が変更された、PC 上で動作する SH4 用のコンパイラとリンカが入っています。 また、SH4 上で動作する実行プログラムが格納されています。
SH4TOOL/sh4-linux/etc SH4TOOL/sh4-linux/usr/etc	SH4 上で動作するプログラムの設定ディレクトリです。 実際に使用されるのは、対象のプログラムが SH4 上で動作したときです。
SH4TOOL/sh4-linux/include SH4TOOL/sh4-linux/usr/include SH4TOOL/sh4-linux/usr/local/include	コンパイルするときに使用するヘッダファイルが格納されています。
SH4TOOL/sh4-linux/lib SH4TOOL/sh4-linux/usr/lib SH4TOOL/sh4-linux/usr/local/lib	SH4 上で動作する実行可能ファイルが利用できる､コードを格納したライブラリが格納されています。
SH4TOOL/sh4-linux/usr/X11R6	SH4 上で動作する X Window System です。
SH4TOOL/sh4-linux/usr/X11R6/bin	SH4 上で動作する X Window System 本体とアプリケーションが格納されています。
SH4TOOL/sh4-linux/usr/X11R6/lib	SH4 上で動作する X Window System が使用するライブラリが格納されています。
SH4TOOL/sh4-linux/usr/X11R6/include	X Window System のヘッダファイル
SH4TOOL/sh4-linux/usr/local/ws	SH4 上で動作する WideStudio 本体が入っています。実際に使用できるのは、ライブラリのみです。
/usr/local/sh4-linux-dev/image （以降 CFIMAGE とします）	CF イメージとアプリケーションパッケージが格納されているディレクトリです。
CFIMAGE/dd	Algonomix の CF イメージファイルが格納されています。
CFIMAGE/pacages	Algonomix 用に用意したアプリケーションが格納されています。
/home/knoppix/wsproject	WideStudio で開発したサンプルコードが格納されています。

表 2-1-4 に Algonomix 開発環境構築で使用したライブラリパッケージやアプリケーションパッケージのバージョンを示します。

表 2-1-4. 組み込まれているパッケージとそのバージョン

パッケージ	内容	バージョン
Linux	Linux カーネル	2.6.15.7
binutils	リンカ、アセンブラ等のソフトウェア開発ツール	2.16.1
gcc	GNU C コンパイラ	3.4.4
glibc	GNU C ライブラリ	2.3.6
gdb	GDB デバッガ	6.3
gdbserver	GDB デバッガのサーバ	6.3
busybox	UNIX 基本コマンドのマルチコールバイナリ	1.1.0
zlib	圧縮・解凍ライブラリ	1.2.3
jpeg	JPEG ライブラリ	6
png	PNG ライブラリ	1.2.8
xorg-x11	X Window System	6.8.1
blackbox	ウィンドウマネージャ	0.70.1
expat	XML パーサ	2.0.0
freetype	Free な TrueType フォントのレタリング用ライブラリ	2.1.10
fontconfig	システム中のフォント管理するためのライブラリ	2.3.94
WideStudio	統合開発環境	3.92.1
xpdf	X Window System 用の PDF リーダ	3.01
dillo	ブラウザ	0.8.5
plaympeg	MPEG プレーヤ	0.4.3
alsa	オーディオと MIDI 機能を提供します。	1.0.10

２－２　Algonomix用開発環境のハードディスクへのインストール

Algonomix用開発環境(Knoppix版)はDVDのみでも起動しアプリケーションの開発を行うことができますが、実際に開発を行うには、やはり、ハードディスクへのインストールを行ったほうが、実行速度も速くなり、ストレス無く開発を行うことが出来ます。
ここでは、Algonomix 用開発環境（Knoppix 版）のハードディスクへのインストール方法の説明を行います。

１）root のシェルを起動

① デスクトップ上の  をクリックするとコンソールが立ち上がります。
② root 権限で実行するために su コマンドを実行します。

```
$ su
#
```

③ プロンプトが#になったことを確認してください。

２）knoppix-installer を実行

① インストールプログラムを実行します。

```
# knoppix-installer
```

３）**注意書きに OK**

①「このスクリプトは、Knoppix を Hard Disk へインストールするものです。注意：このバージョンは、とても初期段階でまだかなり開発中のものです・・・。」というメッセージが表示されるので、確認のうえ OK を押します。

４）**インストールの一覧を選択**

①次の一覧が表示されます。

```
1.インストールの設定　新しい設定を作成します
2.インストールの開始　インストールを開始します
3.パーティション設定　ハードディスクを分割します
4.設定の読み込み　　既存の設定を読み込みます
5.設定の保存　　設定を保存します
```

②「1.インストールの設定」を選ぶと次の 3 つの選択肢が表示されます。

```
debian：　Debian 形式のシステム（推奨）
beginner: マルチユーザーシステムと自動認識
Knoppix：　CD 起動のような Knoppix-System
```

③「debian：　Debian 形式のシステム（推奨)」を選択します。

５）**パーティションの選択**

①次に、パーティションのフォーマット形式の選択画面になります。最も適当と思われるパーティションが自動的に選択されるので、確認のうえ作業を進めてください。

６）**システムタイプの選択**

①次に、パーティションの選択画面になります。次の２つの選択肢が表示されます。

```
Ext3:journal サーポート付き Extended2 ファイルシステム
Reiserfs:ReiserFS 3.6:Namesys 開発によるジャーナリングファイルシステム
```

②ここでは Ext3 を選択します。

７）**あなたの全ての名前を入力**

①次に、システム名の設定をします。ここでは適当な名前を付けておきます。（例： name surname）この名前は、初期ユーザー作成に使用されます。入力が済めば、Next を押します。

８）**ユーザー名の設定**

①ユーザー名を入力します。好きなユーザー名（例：root）を入れて、Next を押します。

９）**ユーザーパスワードの設定**

①ユーザーのパスワードを設定し入力します。確認のため２回入力してから、Next を押します

１０）**root パスワードの設定**

①管理者(root)用のパスワードを設定し入力します。確認のため２回入力してから、Next を押します。

１１）**ホスト名の設定**

①ホスト名を設定します。そのパソコンの名前になります。（例：box）

１２）**ブートローダのインストール先を指定**

①ブートローダー（lilo）を入れる場所を指定します。多くの場合、Master Boot Record を選ぶことになります。

１３）インストールの開始

①自動的に最初の一覧画面に戻ります。この中から、「2.インストールの開始」を選びます。

１４）確認画面に Next

①「これらの設定で実行してもよろしいですか？」というメッセージが表示されます。

②これでよいのであれば、 Next を押します。この画面は、インストール直前の最終確認の画面です。インストールの設定に関する変更を希望するときは、この画面をキャンセルし元に戻って再度設定し直してください。 Next ボタンを押すと、取り消すことはできません。

１５）フォーマット＞ファイルのコピー

①/dev/hda* の初期化が開始され、続いて CD からハードディスクへのコピーが行なわれます。そのあいだの経過状況は、進捗バーで表示されます。ここでの一連の所要時間は、およそ２０分以上かかります。やがてインストールは、自動的に完了します。完了した時点で、ブートディスク作成の画面に切り替わります。

１６）ブートディスク作成の確認

①ブートディスク（起動用ディスク）を作成をするかどうか尋ねられます。

②作成を希望する場合は、 フロッピーディスクを挿入して Yes を押します。

１７）完了

①「インストール成功」の画面が出ます。

②「ハードディスクへ Knoppix を入れることに成功しました。」というメッセージが表示されます。

１８）停止

①システムを終了させます。 コンソール は、 exit で閉じ、ログアウトします。

②正常に終了できれば CD が自動的に排出されるので、忘れずに取り出してからリターンキーを押します。これでパソコンの電源が切れます。

１９）再起動

①再び、パソコンの電源を入れます。途中でログイン画面が表示されるので、登録したパスワードを入力します。

２０）終了

①パソコンが完全に起動したら、 KNOPPIX を使い始めることができます。

２－３　他のLinuxに開発環境を入れる方法

Algonomix 開発環境は、「Knoppix」の他に「RedHat Enterprise LinuxWSVer4」にも対応しています。以下に Algonomix 開発環境を「RedHat Enterprise LinuxWSVer4」にインストールする方法を説明します。

１）**開発環境のインストール**

①　デスクトップ上のメニューからアプリケーション→システムツール→GNOME 端末をクリックするとコンソールが立ち上がります。

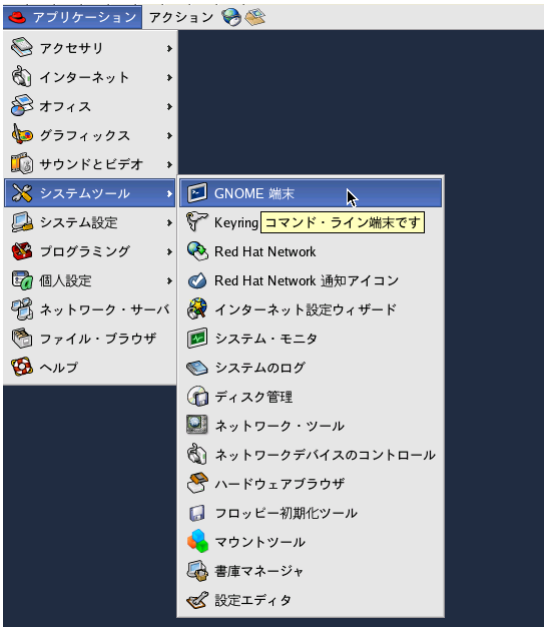

図 2-3-1.　システムツールメニュー画面

②　開発環境 DVD をマウントします。(ここでは DVD-ROM ドライブが/dev/hdc1 に割り当てられているとします。)

```
$ sudo mount /dev/hdc1 /mnt/DVD-ROM
$
```

③　開発環境をインストールします。

```
$ sudo tar zxvf /mnt/DVD-ROM/ALGO/development/asd-dev-1.61.tgz –C /
$
```

④　必要があれば、開発環境のソースファイルをインストールします。

```
$ sudo tar zxvf /mnt/DVD-ROM/ALGO/development/asd-dev-src-1.61.tgz –C /
$
```

⑤　ＡＳＤのインストール用パッケージをインストールします。

```
$ sudo tar zxvf /mnt/DVD-ROM/ALGO/development/asd-packages-1.61.tgz –C /
$
```

⑥　/usr/local/ws/wsstart.sh を実行すると Wide Studio が実行できます。
必要に応じて、以下の環境変数を設定してください。

```
export WSDIR=/usr/local/ws
export LD_LIBRARY_PATH=/usr/local/ws/lib
export PATH=/usr/sh4-linux/bin:/usr/local/ws/bin:$PATH
```

２－４　CFカードの内容

ALGO Smart Display では、Linux カーネル本体は Flash ROM に書き込まれています。CF カードには、ルートファイルシステムと呼ばれる Linux システムに必要なファイルやディレクトリが入っています。ルートファイルシステムにはデバイスファイル等のハードウェア依存のファイルも含まれているため、製品毎に内容が違います。以下に CF カードの内容について説明します。

CF カードのイメージは「/usr/local/sh4-linux-dev/image」に入っています。それぞれのディレクトリの意味と内容物について説明します。

＜ベースルートファイルシステム（base-x.x.tgz）＞

ここには、各製品に共通となるベースルートファイルシステムが格納されています。CF カードをフォーマットした後、このディレクトリの中身を最初にコピーします。

リスト 2-4-1 にディレクトリ構成を、表 2-4-1 にディレクトリ内容を示します。

リスト 2-4-1. ベースルートファイルシステムのディレクトリ構成

```
+--bin
+--dev
+--etc
+--home--+--asdusr
+--lib
+--media
+--mnt
+--proc
+--root
+--sbin
+--sys
+--tmp
+--usr--+--bin
        +--lib
        +--sbin
        +--share
+--var
```

表 2-4-1. ベースルートファイルシステムのディレクトリ内容（抜粋）

ディレクトリ名	内容
/bin	ls や cp など基本操作に必要なコマンドが格納されています。
/sbin	/bin の内容物は一般ユーザも使用するコマンドですが、ここには、システムよりのコマンドであまり一般的でないものが格納されています。
/dev **※注：/dev は RAM ディスク上で構成されています。新規にデバイスファイルを作成しても電源を入れ直した場合は保存されません。保存したい場合は「/dev/.static/dev」ディレクトリでデバイスファイルを作成してください。**	ハードウェアをコントロールするためのデバイスファイルが格納されています。基本的なデバイスの一覧を以下に示します。 ・ハードディスク（CF カード）：/dev/hd*　　例：hda1,hda2,hdb hd の後ろにつく文字がディスクを特定し、数字がパーティションを表しています。 ・ターミナル：/dev/tty*　　例：tty0,tty1 キーボードとプリンタにより構成され、文字を入出力できます。 ・シリアルポート：/dev/ttyS*　　例：ttyS0,ttyS1,ttyS2 SH4 のシリアルインターフェースを制御するためのデバイスファイルです。 AP110,AP210,AP310,AP315 すべて外にでているシリアルポートは ttyS2 です。 ・FlashROM：/dev/mtd*,/dev/mtdblock*　　例：mtd1,mtdblock2 FlashROM をアクセスするためのデバイスです。Linux カーネルを書き換えたいときは mtdblock3 に Linux カーネルをコピーすることで書き換えることができます。 **※注：mtdblock0 には IPL、mtdblock1 には各種設定データ、mtdblock2 には IPL 動作時にでる bmp ファイルが書き込まれています。IPL は変更しないでください。また、書き込み途中に電源を落とすと、起動しなくなる可能性があります。ご注意ください。** ・フレームバッファ：/dev/fb*　　例：fb0 LCD に表示されるグラフィックイメージを保持する領域です。X Window などはこれをアクセスすることで、LCD に描画します。 ・マウス：/dev/input/mice,/dev/psaux　　例：psaux,mice マウスデバイスをコントロールします。 ・SCSI ディスク：/dev/sd*　　例：sda1,sda2,sdb USB メモリや DVD-ROM 等はこのデバイスになります。sd の後ろにつく文字がディスクを特定し、数字がパーティションを表しています。
/etc	設定ファイルが格納されています。ここでは、変更する可能性のあるものについて説明していきます。 ・/etc/profile 環境変数を登録することができます。 ・/etc/network/interfaces ここで、イーサネットの IP 設定を行います。デフォルト設定は「192.168.0.1」となっています。 ・/etc/inittab Linux が起動されたとき、初期化すべき項目が書かれています。ここに、「ttySC1::respawn:/sbin/getty -L 38400 ttySC1 vt100」という行があります。これは、シリアルポート「ttySC1」をシリアルコンソールとして使用するための設定です。 **※注：シリアルポート「ttySC1」はケース外にはでていません。使用する場合は、ケースを開ける必要があります。**
/home	システムに登録された通常ユーザの個人用ディレクトリが入っています。ALGO Smart Display では、「asdusr」といわれるユーザが登録されています。ftp や telnet ではこのユーザに対してアクセスします。 ユーザ名　：asdusr パスワード：asdusr
/lib /usr/lib	SH4 上で動作する実行可能ファイルが利用できる、コードを格納したライブラリが格納されています。
/mnt	USB メモリ等をマウントするときのテンポラリディレクトリです。

/root	システムの管理者権限を持ったユーザのホームディレクトリです。
/usr/bin /usr/sbin	/bin,/sbin よりもユーザよりのコマンドが格納されています。

＜AP315 用の追加ファイル（ap315-x.x.tgz）＞

ここには、ベースルートファイルシステムに対して、追加書き込みできる、AP315 用の設定ファイルやコマンドがあります。ベースファイルシステムをコピーした後、上書きコピーしてください。

リスト 2-4-2 にディレクトリ構成を示します。AP315 は、ネットワークポートが標準実装されているため、IP 設定用ファイルが追加されています。また、AP315 は汎用入出力（IN6、OUT4)、RAS 機能デバイスを持っています。これらのデバイスを制御できるようにするためのファイルが追加されています。

リスト 2-4-2. AP315 用追加ファイルシステムのディレクトリ構成

```
+--etc
+--lib
+--sbin
```

＜AP310、AP210 用の追加ファイル（ap310-x.x.tgz）＞

ここには、ベースルートファイルシステムに対して、追加書き込みできる、AP310、AP210 用の設定ファイルやコマンドがあります。ベースファイルシステムをコピーした後、上書きコピーしてください。

リスト 2-4-3 にディレクトリ構成を示します。AP310、AP210 は、ネットワークポートが標準実装されているため、IP 設定用ファイルが追加されています。また、AP310、AP210 は汎用入出力（IN6、OUT4）を持っています。汎用入出力を制御できるようにするためのファイルが追加されています。

リスト 2-4-3. AP310、AP210 用追加ファイルシステムのディレクトリ構成

```
+--etc
+--lib
+--sbin
```

＜AP110 用の追加ファイル（ap110-x.x.tgz）＞

ここには、ベースルートファイルシステムに対して、追加書き込みできる、AP110 用の設定ファイルやコマンドがあります。AP110 を使用する場合に、ベースファイルシステムをコピーした後、上書きコピーしてください。

リスト 2-4-4 にディレクトリ構成を示します。AP110 はパネルスイッチを５つ持っています。パネルスイッチの状態をリードするためのファイルが追加されています。

リスト 2-4-4. AP110 用追加ファイルシステムのディレクトリ構成

```
+--etc
+--lib
```

＜X Window System の実装（xwindow.tgz）＞

ベースルートファイルシステムでは、X Window System は実装されていません。X Window System を実装する場合は、ベースファイルシステムと機種用追加ファイル(AP110 または AP310、AP315)をコピーした後、上書きコピーしてください。

リスト 2-4-5 にディレクトリ構成を、表 2-4-2 にディレクトリ内容を示します。

リスト 2-4-5. X Window System 追加用ファイルシステムのディレクトリ構成

```
+--etc--+--X11
+--lib
+--root
+--usr--+--X11R6--+--bin
                  +--lib
        +--etc
        +--lib
        +--sbin
+--var
```

表 2-4-2. X Window System 用ファイルシステムのディレクトリ内容（抜粋）

ディレクトリ名	内容
/etc	X Window 用の設定が格納されています ・/etc/profile 以下の環境変数が追加されています。 export DISPLAY=localhost:0.0 ・/etc/init.d/startup X Window System が自動的に起動されるように変更されています。 ・/etc/X11 X Window System 用の設定ファイルが格納されています。 ・/etc/xunicoderc WideStudio のフォント設定ファイルです。フォントの設定形式は X Window System のフォント設定方法を採用されています。
/root	blackbox の初期設定ファイルが格納されています。
/home/asdusr	アプリケーション自動起動設定ファイルが格納されています。 ・/home/asdusr/autostart 初期状態では asd_config プログラムが設定されいます。このファイルに実行ファイル名を記述することで、次回起動時に自動的にプログラムが起動されます。
/lib /usr/lib /usr/X11R6/lib	X Window System で使用するライブラリが追加されています。WideStudio のライブラリもここに格納されています。
/usr/sbin /usr/X11R6/bin	X Window System 本体と X Window System で使用するコマンドが格納されています。

＜TrueType フォントの実装（sharefonts.tgz）＞

Algo Smart Display で TrueType フォントを実装するときにコピーします。
リスト 2-4-6 にディレクトリ構成を、表 2-4-4 にディレクトリ内容を示します。

リスト 2-4-6. sharefonts 追加用ファイルシステムのディレクトリ構成

```
+--usr--+--share--+--truetype
                  +--type1
```

表 2-4-4. sharefonts 追加用ファイルシステムのディレクトリ内容（抜粋）

ディレクトリ名	内容
/usr/share	TrueType フォントと Type1 フォントが格納されています。

＜PDF リーダの実装（xpdf-3.01.tgz）＞

Algo Smart Display で PDF リーダを実装したときにコピーします。X Window System を実装したあとにコピーしてください。XPDF を使用するには、同時に「append-sharefonts」を実装してください。
リスト 2-4-7 にディレクトリ構成を、表 2-4-5 にディレクトリ内容を示します。

リスト 2-4-7. XPDF 追加用ファイルシステムのディレクトリ構成

```
+--etc--+--xpdf
+--usr--+--bin
        +--share--+--xpdf--+--Japanese
```

表 2-4-5. XPDF 追加用ファイルシステムのディレクトリ内容（抜粋）

ディレクトリ名	内容
/etc/xpdf	xpdf の設定ファイルが格納されています。 ・/etc/xpdf/xpdfrc この設定ファイルで使用するフォントを指定しています。
/usr/bin	xpdf という実行ファイルが格納されています。
/usr/share	xpdf で使用する標準のフォントが格納されています。

＜MPEG プレーヤの実装（plaympeg-0.4.3.tgz）＞

Algo Smart Display で MPEG プレーヤを実装するときにコピーします。X Window System を実装したあとにコピーしてください。
リスト 2-4-8 にディレクトリ構成を、表 2-4-6 にディレクトリ内容を示します。

リスト 2-4-8. MPEG プレーヤ追加用ファイルシステムのディレクトリ構成

```
+--usr--+--bin
```

表 2-4-6. MPEG プレーヤ追加用ファイルシステムのディレクトリ内容（抜粋）

ディレクトリ名	内容
/usr/bin	plaympeg という実行ファイルが格納されています。

※ 現バージョンでの使用上での注意

1) フル画面での動画再生は音声なしで 5 コマ/秒程度です。
2) サンプリング 44.1KHz の音声付で動画を再生する場合、画像サイズを（240x160 ドット[VGA]）程度にしてください。画質により音とびが発生する場合があります。
3) 音声のサンプリング速度を遅くすると動画への影響は減少します。

＜ロケールデータの実装（local.tgz）＞

Algo Smart Display でロケールデータを実装するときにコピーします。
リスト 2-4-9 にディレクトリ構成を、表 2-4-7 にディレクトリ内容を示します。

リスト 2-4-9. ロケールデータ追加用ファイルシステムのディレクトリ構成

```
+--usr--+--bin
        +--lib
        +--sbin
        +--share
```

表 2-4-7. ロケールデータ追加用ファイルシステムのディレクトリ内容（抜粋）

ディレクトリ名	内容
/usr/bin /usr/sbin	ロケールデータを作成する実行ファイルが格納されています。
/usr/lib	ロケールデータ用のライブラリが格納されています。
/usr/share	ロケールデータが格納されています。

＜dillo ブラウザの実装（dillo-0.8.5.tgz）＞

Algo Smart Display で dillo ブラウザを実装するときにコピーします。X Window System を実装したあとにコピーしてください。日本語を表示させるには、TrueType フォントとロケールデータをコピーする必要があります。
リスト 2-4-9 にディレクトリ構成を、表 2-4-7 にディレクトリ内容を示します。

リスト 2-4-9. dillo 追加用ファイルシステムのディレクトリ構成

```
+--root
+--usr--+--lib
        +--local--+--bin
                  +--dillo
                  +--dillo-libs
                  +--etc--+--dillo
                  +--lib
                  +--sbin
                  +--share
```

表 2-4-7. dillo 追加用ファイルシステムのディレクトリ内容（抜粋）

ディレクトリ名	内容
/root	dillorc という dillo 設定ファイルが格納されています。 起動したときの画面サイズやフルスクリーン設定等を設定することができます。
/usr/lib	dillo で使用するライブラリが格納されています。
/usr/local/bin	dillo 実行ファイルが格納されています。
/usr/local/dillo	dillo プログラム本体が格納されています。
/usr/local/etc/dillo	dillorc という dillo 設定ファイルが格納されています。 /root/dillorc の設定が優先されます。実行者が/root 権限でない場合はこちらの設定ファイルが反映されます。同じ設定にしておくことを推奨します。
/usr/local/dillo-libs	dillo のみが使用するライブラリが格納されています。

２－５　CFカードの作成方法

ALGO Smart Display で使用する CF カードを作成する方法を以下に示します。

①USB 接続の CF カードリーダを PC に接続します。今回は、CF が/dev/sda として認識されたとして説明します。

②「fdisk」というコマンドを使用し、CF カードにパーティションを作成します。

```
$ su
# /sbin/fdisk /dev/sda
コマンド (m でヘルプ): m
コマンドの動作
   a   ブート可能フラグをつける
   b   bsd ディスクラベルを編集する
   c   dos 互換フラグをつける
   d   領域を削除する
   l   既知の領域タイプをリスト表示する
   m   このメニューを表示する
   n   新たに領域を作成する
   o   新たに空の DOS 領域テーブルを作成する
   p   領域テーブルを表示する
   q   変更を保存せずに終了する
   s   空の Sun ディスクラベルを作成する
   t   領域のシステム ID を変更する
   u   表示/項目ユニットを変更する
   v   領域テーブルを照合する
   w   テーブルをディスクに書き込み、終了する
   x   特別な機能（エキスパート専用）
```

③「p」で現状のパーティションを見ることができます。「d」でパーティションを削除します。

```
コマンド (m でヘルプ): p

Disk /dev/sda: 259 MB, 259080192 bytes
16 heads, 63 sectors/track, 502 cylinders
Units = シリンダ数 of 1008 * 512 = 516096 bytes

 デバイス Boot      Start         End      Blocks   Id  System
/dev/sda1   *           1         501      252472+   6  FAT16
領域 1 は異なった物理/論理終点になっています。:
     物理=(1012, 15, 63) 論理=(500, 15, 63)

コマンド (m でヘルプ): d
Selected partition 1

コマンド (m でヘルプ): p

Disk /dev/sda: 259 MB, 259080192 bytes
16 heads, 63 sectors/track, 502 cylinders
Units = シリンダ数 of 1008 * 512 = 516096 bytes

 デバイス Boot      Start         End      Blocks   Id  System
```

④「n」でパーティションを作成します。「p」で基本領域を選択し、領域番号は「1」を選択します。最初シリンダと終点シリンダはデフォルトでいいのでそのまま次へいきます。パーティションができたことを確認してください。

```
コマンド (m でヘルプ): n
コマンドアクション
   e   拡張
   p   基本領域 (1-4)

コマンドアクション
   e   拡張
   p   基本領域 (1-4)
p
領域番号 (1-4): 1
最初 シリンダ (1-502, default 1):
Using default value 1
終点 シリンダ または +サイズ または +サイズ M または +サイズ K (1-502, default 502):
Using default value 502

コマンド (m でヘルプ): p

Disk /dev/sda: 259 MB, 259080192 bytes
16 heads, 63 sectors/track, 502 cylinders
Units = シリンダ数 of 1008 * 512 = 516096 bytes

 デバイス Boot      Start         End      Blocks   Id  System
/dev/sda1               1         502      252976+  83  Linux
```

⑤「w」で CF カードに設定を書き込みます。これでパーティションの作成は完了です。

```
コマンド (m でヘルプ): w
領域テーブルは交換されました！

ioctl() を呼び出して領域テーブルを再読込みします。
ディスクを同期させます。
#
```

⑥作成したパーティションをフォーマットします。以下のコマンドを実行します。これでフォーマットが完了します。

```
# /sbin/mkfs.ext3 /dev/sda1
mke2fs 1.38 (30-Jun-2005)
Filesystem label=
OS type: Linux
～～～～～省略～～～～～
Writing inode tables: done
Creating journal (4096 blocks): done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 32 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
#
```

⑦CF カードをマウントします。

```
# mount /dev/sda1 /mnt/sda1
```

⑧CF カードに「base」をインストールします。

```
# tar zxvf /usr/local/sh4-linux-dev/image/Packages/Packages-1.61/base-1.61.tgz -C /mnt/sda1
```

⑨CF カードに「append-apXXX」(ap110, ap310, ap315)をインストールします。

```
# tar zxvf /usr/local/sh4-linux-dev/image/Packages/Packages-1.61/ap310-1.61.tgz –C /mnt/sda1
```

⑩CF カードに「xwindow」をインストールします。

```
# tar zxvf /usr/local/sh4-linux-dev/image/Packages/Packages-1.61/x-window-1.61.tgz –C /mnt/sda1
```

⑪CF カードに「sharefonts」をインストールします。

```
# tar zxvf /usr/local/sh4-linux-dev/image/Packages/Packages-1.61/sharefonts-1.1.1.tgz –C /mnt/sda1
```

⑫ここまでが、標準で CF カードに実装するべきものです。あと、MPEG プレーヤや PDF リーダ等を入れたい場合は、⑨,⑩,⑪を参考に「xpdf-3.02.tgz」や「plaympeg-0.4.4.tgz」等を CF カードにインストールします。

⑬CF カードをアンマウントします。

```
# umount /mnt/sda1
```

以上で CF カード作成が完了しました。ALGO Smart Display に CF カードを挿して動作確認してみてください。

２－６　Algo Smart Displayコンフィグプログラムについて

出荷状態で、Algo Smart Display を起動した時、図 2-6-1 のようなコンフィグプログラムが起動されます。ここで、ネットワーク設定やバックライト調整等を行うことができます。

AP110 の起動画面

AP210 の起動画面

AP310/315 の起動画面

図 2-6-1. asd_confg メイン画面

表 2-6-1 にそれぞれのボタンの意味について説明します。

表 2-6-1. メイン画面ボタン一覧

ボタン	名称	内容
	Network Setting	IP アドレスと DNS アドレスを設定します。
	Backlight Adjustment	液晶パネルのバックライトの明るさを調節できます。 AP210/310/315 は 255 段階、AP110 は 3 段階の調節が可能です。
	Volume Setting	ステレオオーディオ出力端子の音量を 96 段階で調節できます。 **※ AP210/310/315 のみ有効です。**
	Touch Panel Calibration	タッチパネルのキャリブレーションを実行できます。 **※ AP210/310/315 のみ有効です。**
	Server Setting	telnet サーバと ftp サーバを起動時に、実行するかしないかを設定できます。 設定は Algo Smart Display の再起動後に反映されます。
	Setting At Date	Algo Smart Display の現在日時を設定できます。
	Kernel Update	Linux カーネルの書き換えを実行します。
	Misc. Setting	その他の設定を行います。
	Hardwear Information	Algo Smart Display ハードウェア情報を表示します。
	Algonomix Shutdown	Algonomix を再起動またはシャットダウンします。 設定値を初期値に戻してから終了することもできます。
	Exit	コンフィグプログラムを終了します。

２－６－１　Network Setting

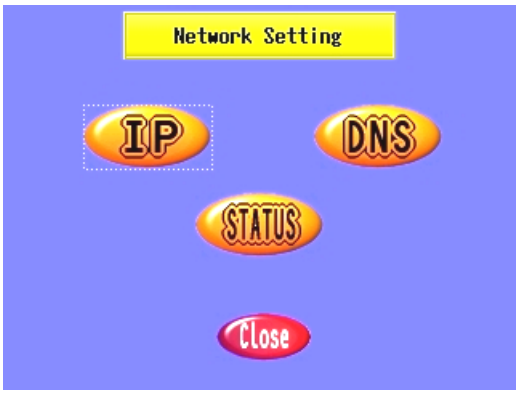

図 2-6-1-1. ネットワーク設定メイン画面

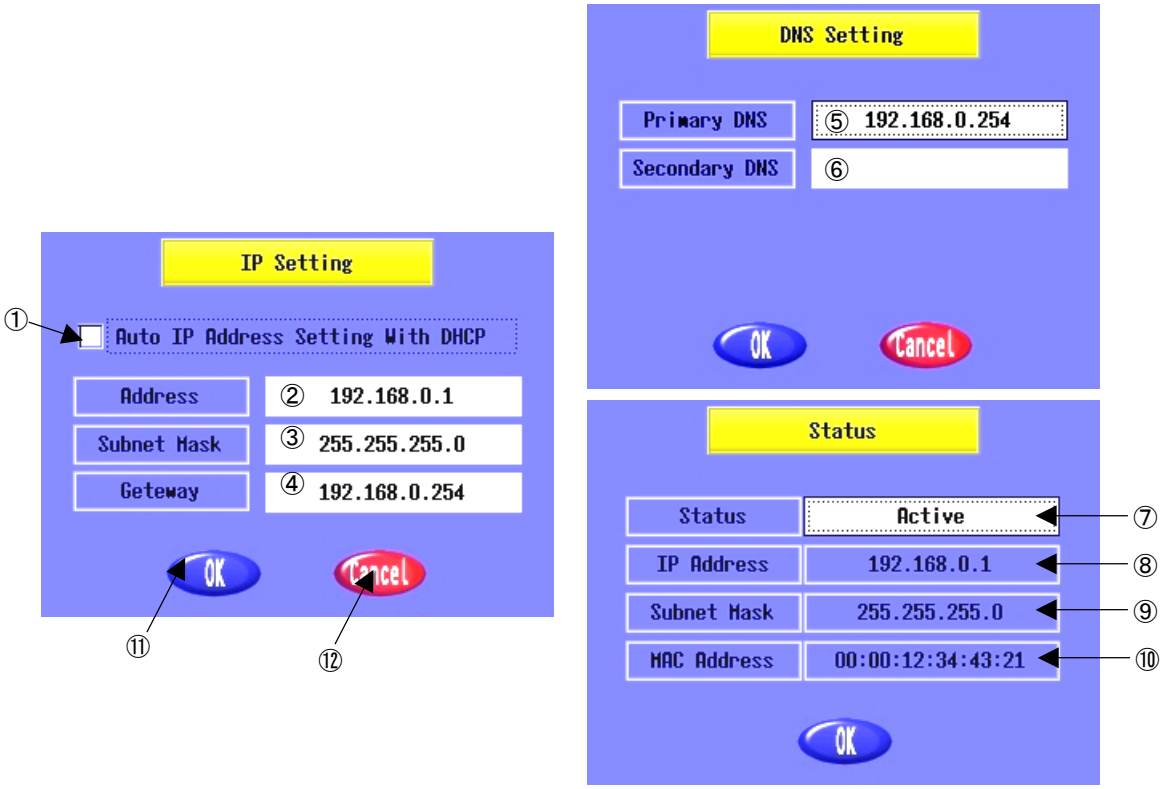

図 2-6-1-2. ネットワーク設定画面

① DHCP 有効／無効
　チェックをつけると、DHCP による IP アドレスの自動取得を行います。この場合、②～⑥の設定は無効となります。
　チェックをはずすと、IP アドレスを設定することができます。

② IP アドレスの設定
　クリックすると図 2-6-1-3 のような数値入力画面が表示されます。ここで IP アドレスを設定してください。

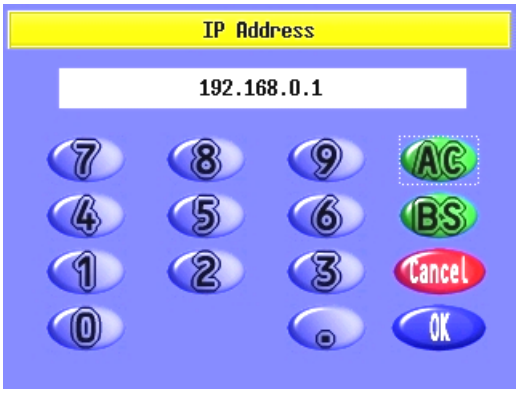

図 2-6-1-3. 数値入力画面

③　サブネットマスクの設定
　　クリックすると図 2-6-1-3 のような数値入力画面が表示されます。ここでサブネットマスクを設定してください。

④　ゲートウェイの設定
　　クリックすると図 2-6-1-3 のような数値入力画面が表示されます。ここでゲートウェイを設定してください。

⑤　優先 DNS アドレス設定
　　クリックすると図 2-6-1-3 のような数値入力画面が表示されます。ここで優先 DNS アドレスを設定してください。

⑥　サブ DNS アドレス設定
　　クリックすると図 2-6-1-3 のような数値入力画面が表示されます。ここでサブ DNS アドレスを設定してください。この設定は、優先 DNS アドレスが見つからなかった場合に、このアドレスが検索されます。

⑦　ネットワークステータス
　　現在のネットワークの状態を表示しています。
　　「ACTIVE」　：ネットワーク確立
　　「INACTIVE」：ネットワーク切断
　　「ACTIVE」のときにクリックすると「INACTIVE」になります。「INACTIVE」のときにクリックすると「ACTIVE」になります。

⑧　取得 IP アドレス
　　現在、取得している IP アドレスを表示しています。

⑨　取得サブネットマスク
　　現在、取得しているサブネットマスク

⑩　MAC アドレスの表示
　　固有の MAC アドレスが表示されています。変更は不可です。

⑪　OK ボタン
　　設定を保存して、ネットワーク設定メイン画面へ戻ります。メイン画面へ戻った時点で変更内容は反映されます。

⑫　キャンセルボタン
　　設定を保存せずにネットワークメイン画面へ戻ります。変更内容は無視されます。

２－６－２　  Backlight Adjustment

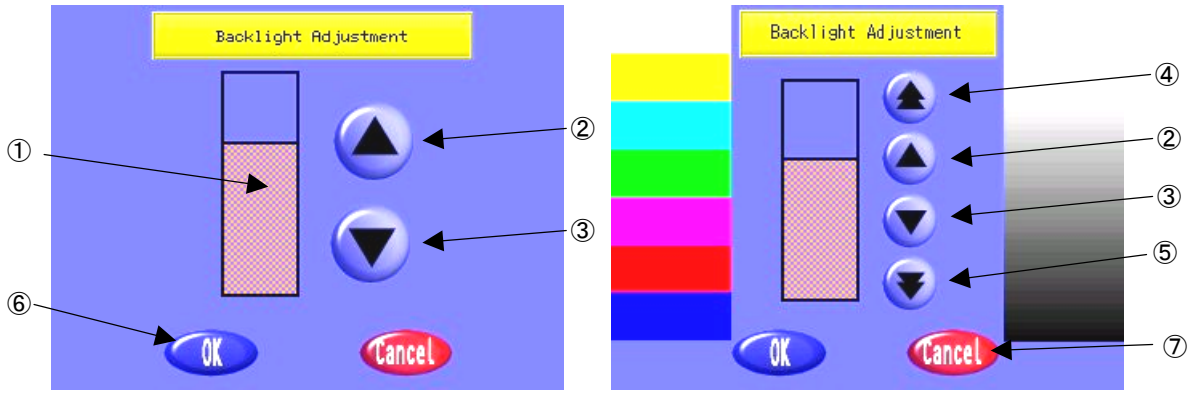

図 2-6-2-1. バックライト調節画面（左：AP110 右：AP210/310/315）

① メータ
　現在のバックライトの明るさレベルを表示しています。②～⑤のボタンをクリックすることで上下します。
　AP210/310/315 では 255 段階で調節可能です。
　AP110 では 3 段階で調節可能です。

② 微 UP ボタン
　クリックすると液晶のバックライトが 1 段階明るくなります。

③ 微 DOWN ボタン
　クリックすると液晶のバックライトが 1 段階暗くなります。

④ 粗 UP ボタン
　クリックすると液晶のバックライトが 25 段階明るくなります。(AP210/310/315 のみ有効)

⑤ 粗 DOWN ボタン
　クリックすると液晶のバックライトが 25 段階暗くなります。(AP210/310/315 のみ有効)

⑥ OK ボタン
　クリックすると現在のバックライトの明るさを保存してメイン画面へ戻ります。
　次回起動時でも、設定したバックライトの明るさが保持されます。

⑦ キャンセルボタン
　クリックすると設定前のバックライトの明るさに戻したうえで、メイン画面へ戻ります。

２－６－３ Volume Setting

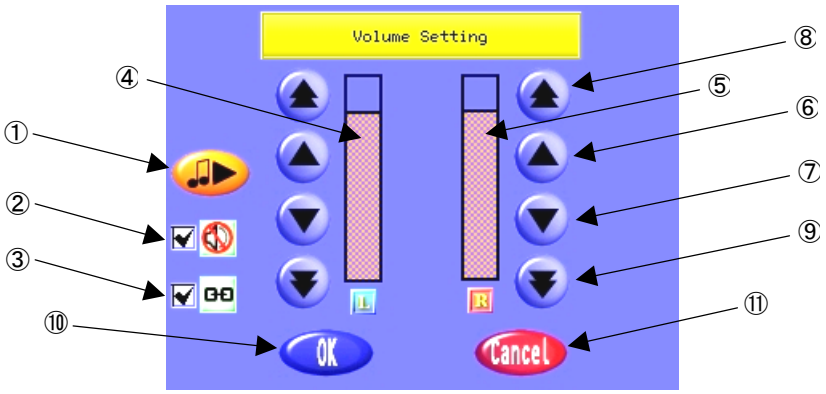

図 2-6-3-1. ボリューム調節画面

① サンプル音再生
　クリックするとサンプル音を再生します。
② ミュート設定
　チェックを入れると、ミュートにします。
③ シンクロ設定
　チェックを入れると、L と R で音量を同じにします。⑥～⑨のボタンをクリックすると、両方のメータが連動して上下します。
④ 左音量メータ
　ステレオ左の音量を表示しています。⑥～⑨のボタンをクリックすることで上下します。
　音量は 96 段階で調整可能です。
⑤ 右音量メータ
　ステレオ右の音量を表示しています。⑥～⑨のボタンをクリックすることで上下します。
　音量は 96 段階で調整可能です。
⑥ 微 UP ボタン
　クリックすると音量が 1 段階大きくなります。
⑦ 微 DOWN ボタン
　クリックすると音量が 1 段階小さくなります。
⑧ 粗 UP ボタン
　クリックすると音量が 10 段階大きくなります。
⑨ 粗 DOWN ボタン
　クリックすると音量が 10 段階小さくなります。
⑩ OK ボタン
　クリックすると現在の音量を保存してメイン画面へ戻ります。
　次回起動時でも、設定した音量が保持されます。
⑪ キャンセルボタン
　クリックすると設定前の音量に戻したうえで、メイン画面へ戻ります。

２－６－４ Touch Panel Calibration

メイン画面で ボタンをクリックした後、図 2-6-4-1 のようなタッチパネル初期起動画面が表示されます。準備が整うまで 5 秒ほどお待ちください。「Cancel」ボタンをクリックするとメイン画面へ戻ります。

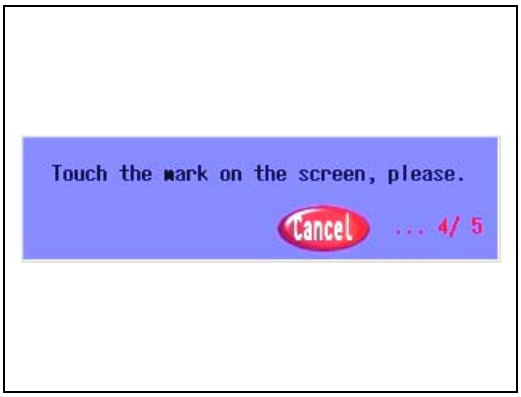

図 2-6-4-1. タッチパネル初期起動画面

準備が整ったら、図 2-6-4-2 のようなマークが表示されます。マークは全部で 6 ヶ所表示されます。ので順次タッチしてください。

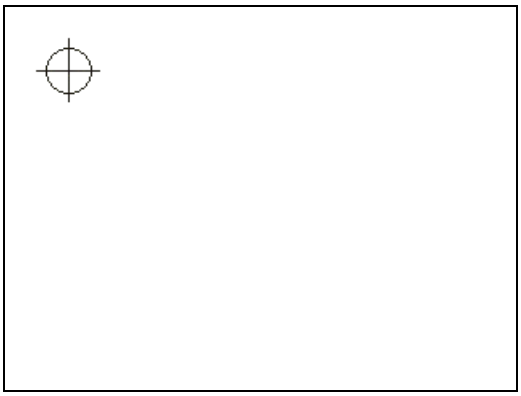

図 2-6-4-2. タッチパネルキャリブレーションポジションマーク

※注：なるべく中心を正確にタッチしてください。キャリブレーション結果に影響されます。

※注：マークがでてから 20 秒以内にタッチしてください。タッチされずに 20 秒経過します。と図 2-6-4-3 のタイムアウト時の画面が表示されます。

6 ヶ所クリックしたあと、図 2-6-4-3 の 6 ヶ所タッチ完了時の画面が表示されます。

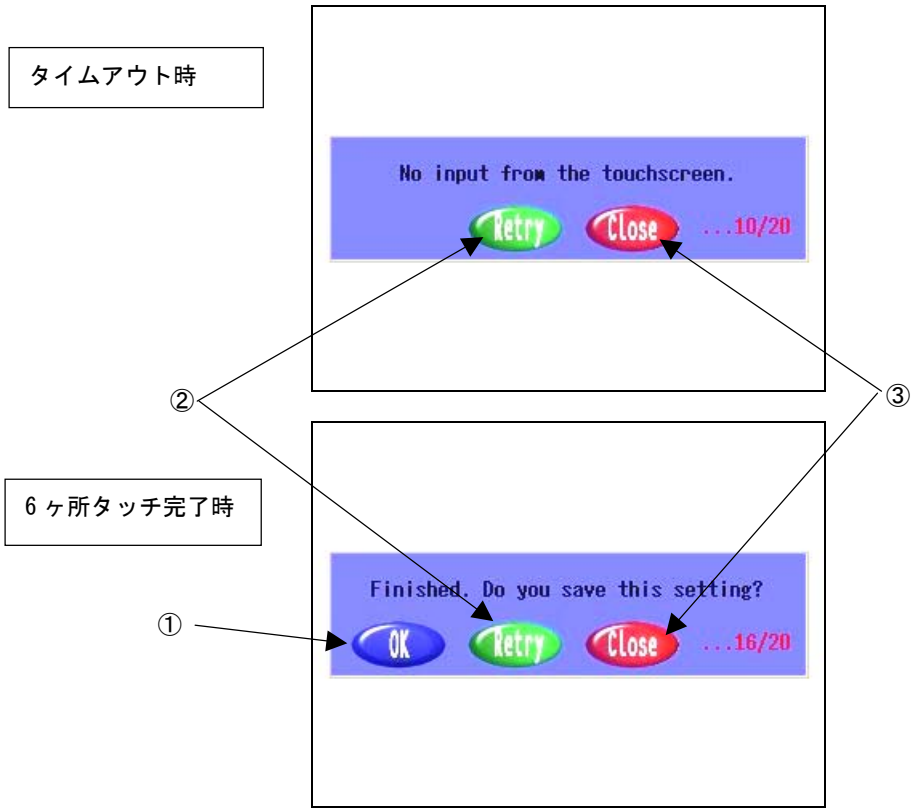

図 2-6-4-3. タッチパネルキャリブレーション完了問い合わせ画面

① OK ボタン
　クリックすると現在のキャリブレーション結果を保存します。
　実際にタッチパネルをタッチして、ポインタとタッチした位置が同じかチェックしてから OK ボタンをクリックしてください。

② Retry ボタン
　クリックすると再度タッチパネルキャリブレーションを実行します。

③ Close ボタン
　キャリブレーション結果を出荷時状態に戻して終了します。

※注：設定前の状態ではなく出荷時状態に戻します。再度タッチパネルキャリブレーションを実行してください。

※注：①～③のいずれのボタンもクリックせずに 20 秒経過すると、キャリブレーション結果を出荷時状態に戻した上でメイン画面へ戻ります。再度タッチパネルキャリブレーションを実行してください。

２－６－５　  Server Setting

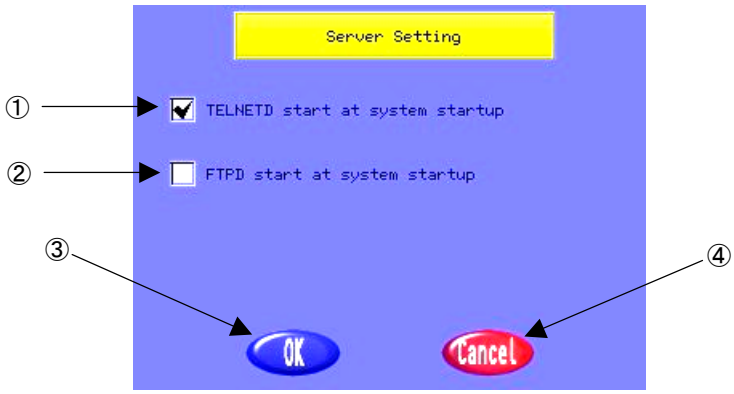

図 2-6-5-1. 各種サーバ自動起動設定画面

① telnet サーバ自動起動設定

チェックすると、次回 Algo Smart Display を起動したとき、telnet サーバを自動的に起動します。

チェックをはずすと、次回 Algo Smart Display を起動したとき、telnet サーバは起動されません。

※注：結果が反映されるのは、次回 Algo Smart Display を起動したときです。

② ftp サーバ自動起動設定

チェックすると、次回 Algo Smart Display を起動したとき、ftp サーバを自動的に起動します。

チェックをはずすと、次回 Algo Smart Display を起動したとき、ftp サーバは起動されません。

※注：結果が反映されるのは、次回 Algo Smart Display を起動したときです。

③ OK ボタン

クリックすると現在の設定を保存してメイン画面へ戻ります。

④ キャンセルボタン

クリックすると設定を保存せずにメイン画面へ戻ります。

２－６－６　  Setting At Date

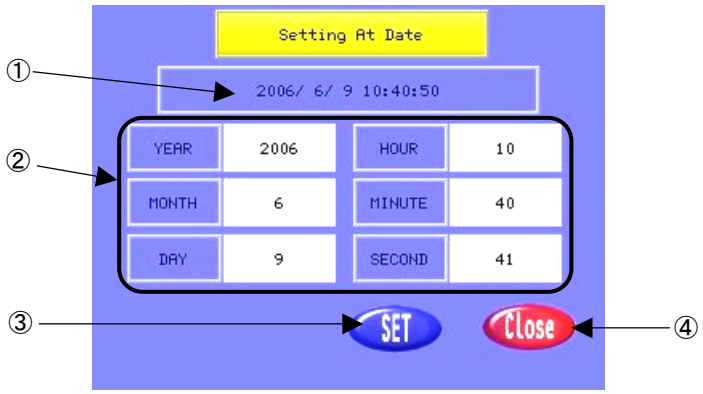

図 2-6-6-1. 現在日時設定画面

① 現在時刻の表示
　現在のシステムクロックを表示しています。

② 年、月、日、時、分、秒の設定
　クリックすると、図 2-6-6-2 のような数値入力画面が表示されます。正しい日付や時間を入力してください。③の SET ボタンが押されるまで設定は反映されません。秒の設定をするときはタイムラグを計算した上で設定してください。

図 2-6-6-2. 数値入力画面

③ SET ボタン
　クリックすると②に設定された日時にシステムクロックを書き換えます。
　①の現在時刻の表示で設定された日時が正しいか確認してください。

④ Close ボタン
　クリックするとメイン画面へ戻ります。

２－６－７　Kernel Update

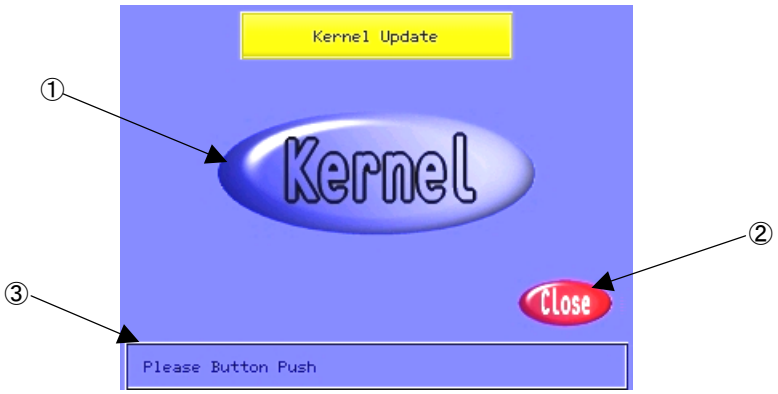

図 2-6-7-1. カーネル書き換え画面

① Linux カーネル書き換えボタン

「/home/asdusr」ディレクトリにある、「vmlinux.bin」というカーネルバイナリファイルをフラッシュ ROM にコピーします。

書き込み中は図 2-6-7-2 のような画面が表示されます。

図 2-6-7-2. カーネル書き込み中

※注：カーネル書き込み中は Algo Smart Display の電源を切らないでください。

※注：不正なカーネルバイナリファイル「vmlinux.bin」を書き込まれた場合は、『２－８ Linux カーネルの復旧』を参考に復旧させてください。

書き込みが完了したら、図 2-6-7-1 の画面に戻ります。このとき、③に書き込み結果を表示します。

② Close ボタン

クリックすると、メイン画面に戻ります。

③ ステータス表示

書き込みが正常に完了すれば、「Write OK」と表示されます。

「vmlinux.bin」が開けない場合は、「File Open Error.」と表示されます。

書き込みに失敗すれば、「Write Error! Please Retry.」と表示されます。再度、①のボタンをクリックし再書き込みを行ってください。

２－６－８  Misc. Settings

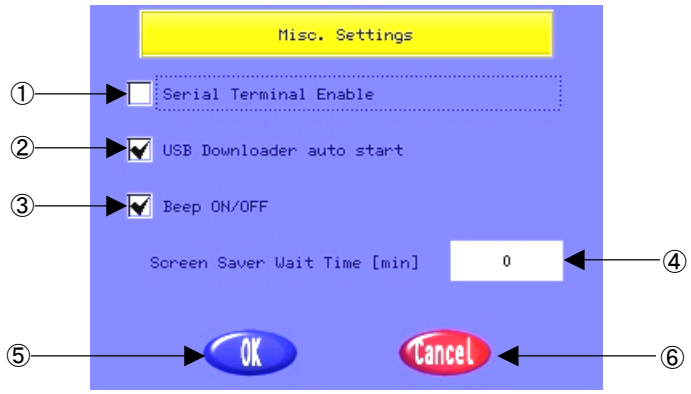

図 2-6-8-1. その他の設定画面

① シリアルターミナルの有効／無効

チェックをつけると、Algo Smart Display のシリアルポートをシリアルコンソールとして使用します。ユーザプログラムで、シリアルポートを使用するときは、チェックをはずしてください。

※注：結果が反映されるのは、次回 Algo Smart Display を起動したときです。

② USB メモリ 自動ダウンローダ起動の ON／OFF

チェックをつけると、USB メモリを挿入したとき、自動的にスクリプトを実行するプログラムを起動します。このプログラムの詳細は「Information 1」を参照してください。

チェックをはずすと、USB メモリを挿入してもこのプログラムは起動されません。

③ Beep ON／OFF

チェックをつけると、タッチパネルをタッチしたとき、または、AP110 のパネルスイッチを押したとき Beep 音を鳴らします。

チェックをはずすと、Beep 音は鳴りません。

④ スクリーンセーバ起動時間設定

スクリーンセーバ起動時間を分単位で指定します。0 を設定するとスクリーンセーバは起動しません。

⑤ OK ボタン

クリックすると、設定を保存してメイン画面へ戻ります。

⑥ キャンセルボタン

クリックすると、設定を保存せずにメイン画面へ戻ります。

２－６－９　Hardware Information

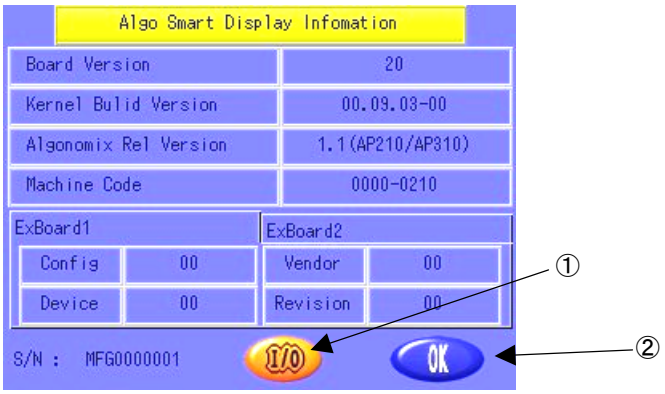

図 2-6-9-1. Algo Smart Display ハードウェア情報

ここで、Algo Smart Display のハードウェア情報とファームウェアバージョンを確認することができます。表 2-6-9-1 に各項目の意味について示します。

表 2-6-9-1. ハードウェア情報

項目名	内容
Board Version	ハードウェアバージョン
Kernel Build Version	Linux カーネル 2.6.15 のビルドバージョン
Algonomix Rel Version	Algonomix 構成のリリースバージョン
Machine Code	Algo Smart Display を区別するための型式です。 ・0000-0110:AP110 ・0000-0210:AP210 ・0000-0310:AP310 ・0000-0315:AP315
ExBoard1	拡張ボード 1 枚目の情報（AP210/310/315 のみ）
ExBoard2	拡張ボード 2 枚目の情報（AP210/310/315 のみ）
Config	拡張ボードコンフィグ情報
Vender	拡張ボードのベンダーコード
Device	拡張ボードのデバイスコード
Revision	拡張ボードのファームウェアバージョン
S/N	シリアルナンバー

① I/O モニタボタン（AP210/310/315 のみ）

クリックすると、図 2-6-9-2 のような I/O モニタが画面になります。

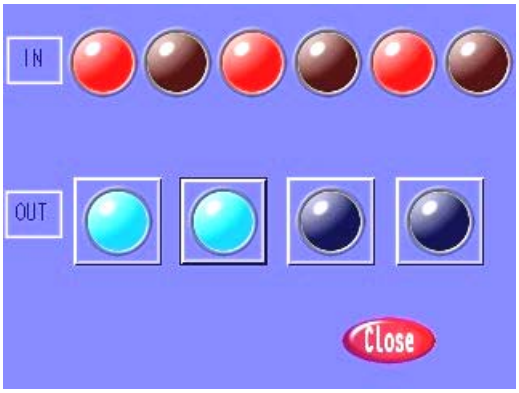

図 2-6-9-2. I/O モニタ画面

OUT のボタンをクリックすることで汎用出力を ON/OFF することができます。また、汎用入力の状態を IN に表示しています。
「Close」ボタンでハードウェア情報モニタ画面に戻ります。

② OK ボタン
クリックすると、メイン画面へ戻ります。

２－６－１０  Algonomix Shutdown

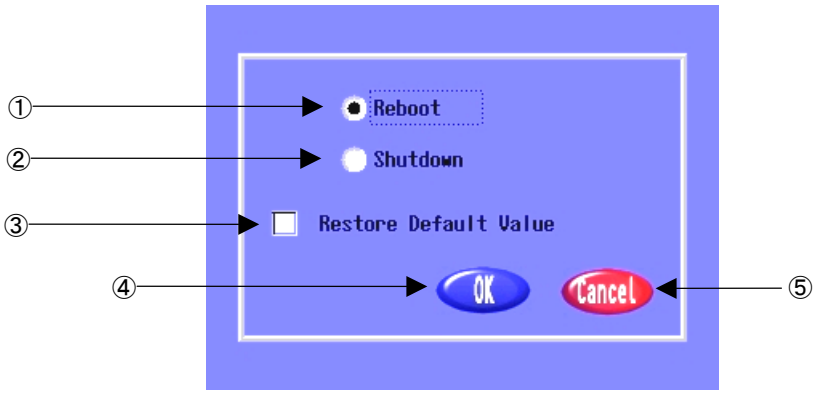

図 2-6-10-1. Algonomix シャットダウン画面

① 再起動
ここにチェックが入った状態で④の「OK」ボタンをクリックすると Algonomix を再起動します。

② シャットダウン
ここにチェックが入った状態で④の「OK」ボタンをクリックすると Algonomix をシャットダウンします。
※注：画面が暗くなりましたら電源を OFF にしてください。

③ 初期設定値復帰
チェックが入った状態で、④の「OK」ボタンをクリックすると、図 2-6-10-2 の画面で確認を促します。

図 2-6-10-1. Algonomix 設定初期化確認画面

「OK」ボタンをクリックすると、ネットワーク設定とバックライト設定、ボリューム設定、サーバ設定、misc 設定を初期状態に戻してから、再起動またはシャットダウンします。

表 2-6-10-1 に各初期設定値を示します。

表 2-6-10-1. 初期設定値

設定	AP110	AP210/310/315
バックライト	2	168
音量（左,右）	ー	80,80
IP	192.168.0.1	192.168.0.1
サブネットマスク	255.255.255.0	255.255.255.0
ゲートウェイ	192.168.0.254	192.168.0.254
telnet 自動起動	有効	有効
ftp 自動起動	有効	有効
シリアルターミナル	無効	無効
USB メモリ 自動ダウンローダ起動	有効	有効
Beep 音	ON	ON

２－７　　Linuxカーネルの復旧

Algo Smart Display では、FlashROM 内の Linux カーネル保護のため、２重にカーネルを保持しています。Algo Smart Display コンフィグプログラムで不正なカーネルを書き込んだ場合や、書き込み途中に電源が OFF された場合でも、出荷時の Linux カーネルに復旧することが可能です。以下に復旧手順について説明します。

※注：復旧されるのは Linux カーネルのみです。CF カードの中身は保護されます。

２－７－１　AP110 の場合

① 「↑」キーと「Enter」キーを同時に押しながら電源を ON します。しばらくすると図 2-8-1-1 のようにオープニング画像の色が反転します。反転するまでボタンを押し続けてください。

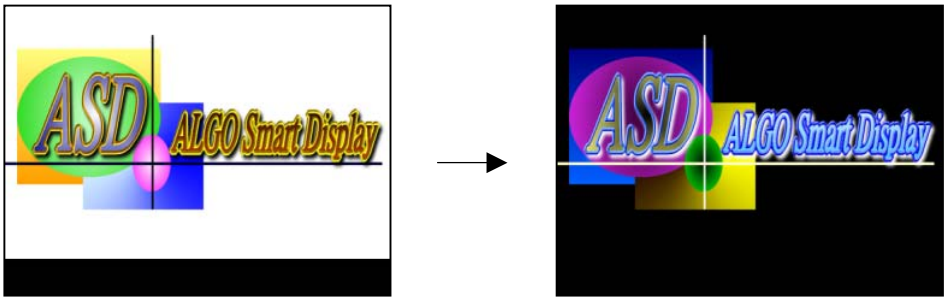

図 2-8-1-1. セーフモードへの移行画面

③ オープニング画面の色が反転したら、10 秒以内に「↑」、「↓」、「←」、「→」、「Enter」の順にボタンを押してください。認識されたら図 2-8-1-2 のような起動メニューが表示されます。10 秒経過した場合は、通常カーネルで起動します。

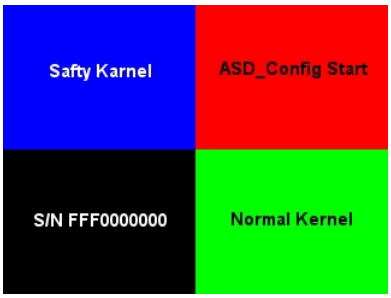

図 2-8-1-2. 起動メニュー画面

④ 起動メニューを選択してください。

「↑」キー：セーフティカーネルで起動します。起動後、「asd_config」が起動されます。
「↓」キー：通常カーネルで起動します。起動後、「/home/asdusr/autostart」の設定を無視して「asd_config」が起動されます。
「←」キー：通常カーネルで起動します。起動後、「/home/asdusr/autostart」に設定されているアプリケーションを起動します。

④ 「↑」キーを押して、セーフティカーネルを起動し、「asd_config」の「Kernel Update」で正常な Linux カーネルを書き込んでください。

２－７－２　AP210/310/315 の場合

① Algo Smart Display の画面をどこでもいいので押しながら電源を ON します。しばらくすると図 2-8-2-1 のようにオープニング画像の色が反転します。反転するまで画面を押し続けてください。

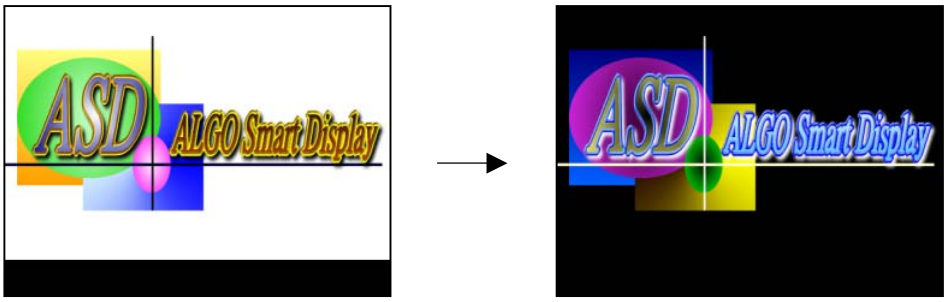

図 2-8-2-1. セーフモードへの移行画面

② オープニング画面の色が反転したら、10 秒以内に図 2-8-2-2 のように画面をなぞってください。

図 2-8-2-2. セーフモードへの移行手順

③ 認識されたら図 2-8-2-3 のような起動メニューが表示されます。②の状態で 10 秒経過した場合は、通常カーネルで起動します。

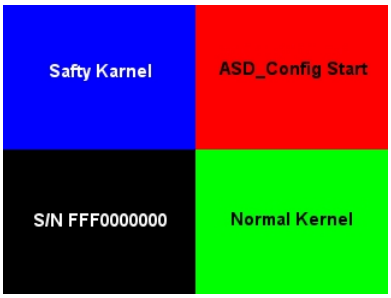

図 2-8-2-3. 起動メニュー画面

④ 起動メニューを選択してください。

Safty Kernel :　セーフティカーネルで起動します。起動後、「asd_config」が起動されます。

Asd_Config Start :　通常カーネルで起動します。起動後、「/home/asdusr/autostart」の設定を無視して「asd_config」が起動されます。

Normal Karnel :　通常カーネルで起動します。起動後、「/home/asdusr/autostart」に設定されているアプリケーションを起動します。

※注：それぞれのエリアの中心をクリックしてください。

⑤ 「Safty Kernel」を選択し、セーフティカーネルを起動し、「asd_config」の「Kernel Update」で正常な Linux カーネルを書き込んでください。

第３章　開発環境

本章では、Algonomix の開発環境について説明します。

３－１　クロス開発環境

プログラムを開発するとき、必要となるのがソースコードを記述するエディタ、ソースコードをコンパイルするためのコンパイラ、コンパイルされたプログラムを実行するための実行環境です。

例えば、Microsoft 社の Windows 上で動作するアプリケーションを開発する場合、エディタでソースを書き、Visual Studio 等のコンパイラでコンパイルを行い、作成された exe ファイルを実行します。これで作成したアプリケーションが Windows 上で実行されます。

Linux の場合でも同じです。PC 上で動作する Linux マシン上で動作するエディタでソースを書き、gcc でコンパイル、できた実行ファイルを実行します。

両者とも、コンパイルと実行を同じ PC 環境上で行えます。このような開発方式をセルフ開発といいます。

ALGO Smart Display（AP110/210/310/315）では、ルネサス テクノロジ社製の SuperH RISC engine SH4（SH7760）という CPU を採用しています。つまり、SH4 で動作する形式でのコンパイルが必要になります。

そこで登場するのがクロス開発方式です。クロス開発とは、コンパイルする環境と実行する環境が異なる方式です。ソースコードの記述や、コンパイルは、PC 上で行い。LAN 等で実行ファイルをターゲットに送って実行することになります。（図 3-1-1 参照）。

ALGO Smart Display はターゲットマシンとして開発された商品であるため、セルフコンパイル環境は用意していません。

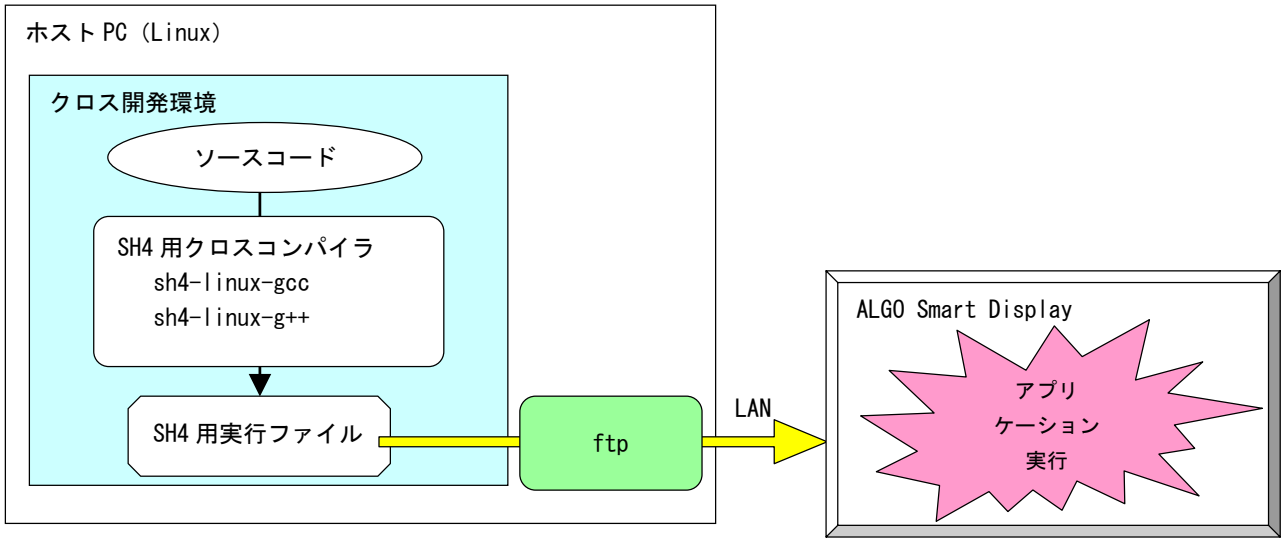

図 3-1-1. クロス開発方式イメージ図

このような開発環境を構築するには、表 3-1-1 に示したようなツールをターゲット用にコンパイルし、順次インストールしていく必要があります。これらのツールにはそれぞれのバージョンの相性があり、相応の経験や知識がないと難しくなります。

表 3-1-1. クロス開発に必要なツール

ツール名	説明
GCC	GNU C コンパイラ
binutils	リンカ、アセンブラ等のソフトウェア開発ツール
gdb	デバッガ
glibc	GNU C ライブラリ

Algonomix の開発環境には上記のようなクロス開発ツールがすでに組み込まれています。PC を起動してすぐに、ALGO Smart Display 用のアプリケーションを開発することが可能です。

３－２　Algonomix開発環境の起動

本書に同梱されてある DVD-ROM は、Knoppix-4.02 日本語版をベースに ALGO Smart Display 用の開発環境を追加し不要なプログラムを削除したものです。

①DVD-ROM を PC にセットして、電源を入れます。
DVD-ROM が読み込まれて、図 3-2-1 のようなオープニング画面が表示されます。

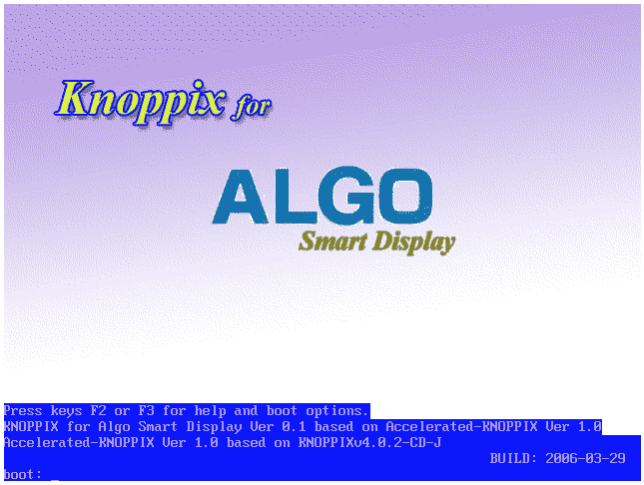

図 3-2-1. オープニング画面

②boot：で入力待ちになるので、通常は何も入力せずに Enter キーを押します。
Linux のカーネルがロードされて、Knoppix が起動します。

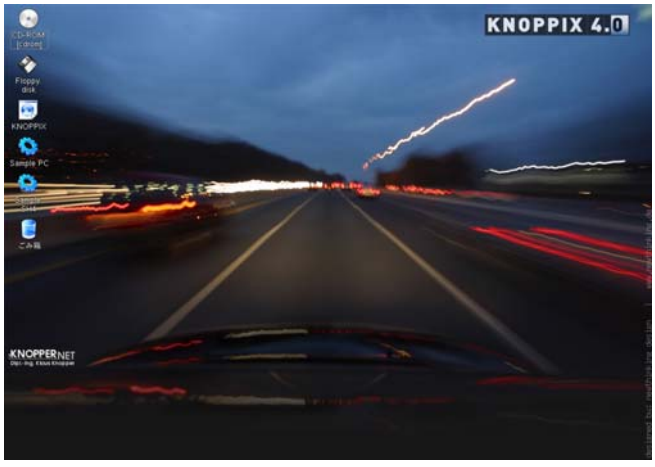

図 3-2-2. Knoppix 画面

! 起動画面が表示されずに、Windows が立ち上がってしまう場合は、BIOS 画面を参照して、PC の BOOT 設定で、ハードディスクよりも DVD-ROM をが優先になっていることを確認してください。

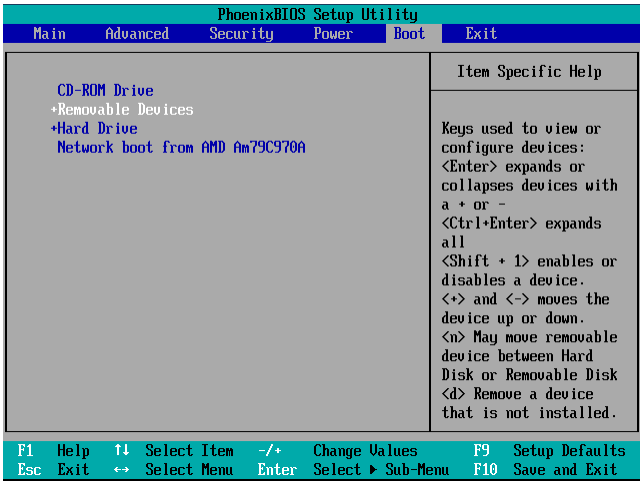

図 3-2-3. BIOS 画面

DVD-ROM の起動を優先したら、設定を保存して再起動します。

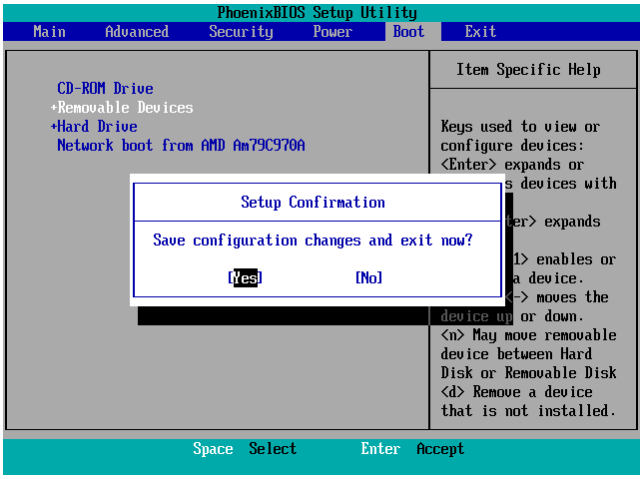

図 3-2-4. BIOS 保存画面

３－３　WideStudio/MWTによるアプリケーション開発

WideStudio/MWTはデスクトップアプリケーションを迅速に作成することのできる統合開発環境です。詳細はWideStudioホームページ（http://www.widestudio.org/index.html）を参照してください。

Algonomix 用開発環境に組み込まれている WideStudio/MWT は V3.92-1 をベースに ALGO Smart Display 用の環境設定を加えてコンパイルしたものです。ここで、WideStudio/MWT で簡単なプログラムをコンパイルして実際に動作させてみましょう。

３－３－１　WideStudioの起動

デスクトップ上の 

のアイコンをクリックすると Wide Studio が起動します。

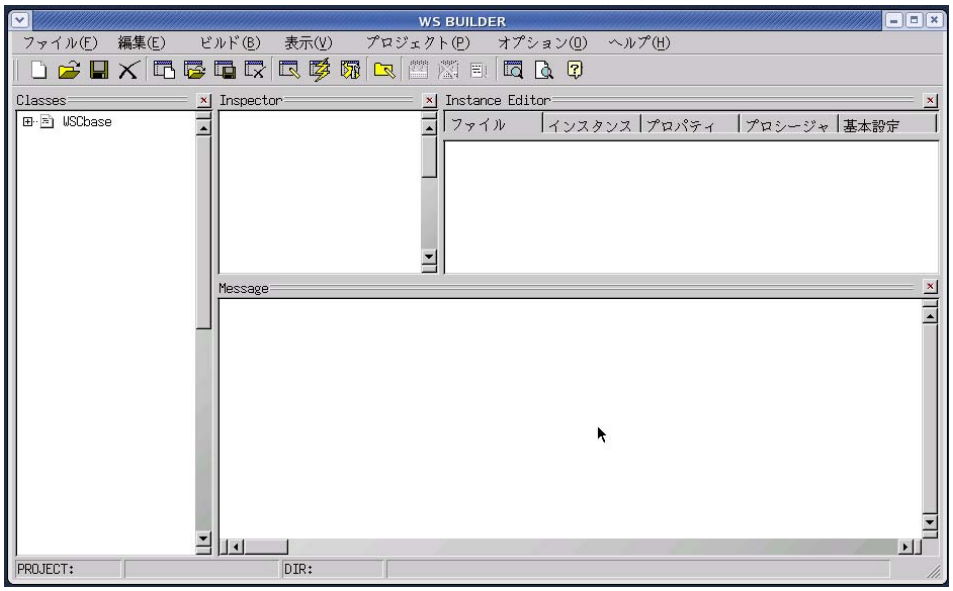

図 3-3-1-1. WideStudio メイン画面

３－３－２　プロジェクトの新規作成

はじめに、アプリケーションを作成するためのプロジェクトを以下の手順で作成します。

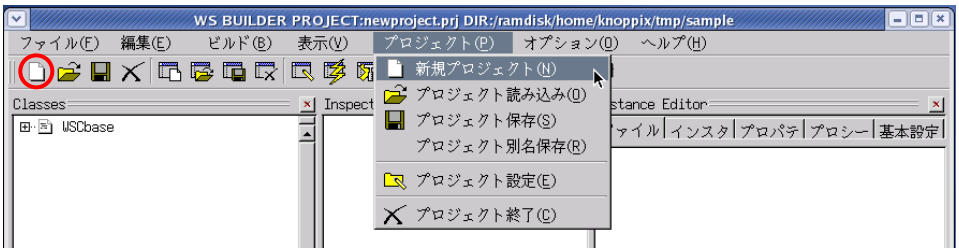

図 3-3-2-1. 新規プロジェクト作成

 をクリックするか、メニューの「プロジェクト(P)」→「新規プロジェクト(N)」をクリックすると、図 3-3-2-2 のような画面が表示されます。ここでプロジェクト名称を記入してください。今回はデフォルトの newproject とします。

図 3-3-2-2. プロジェクト作成画面

 をクリックすると図 3-3-2-3 のようなファイル選択ダイアログが表示されます。

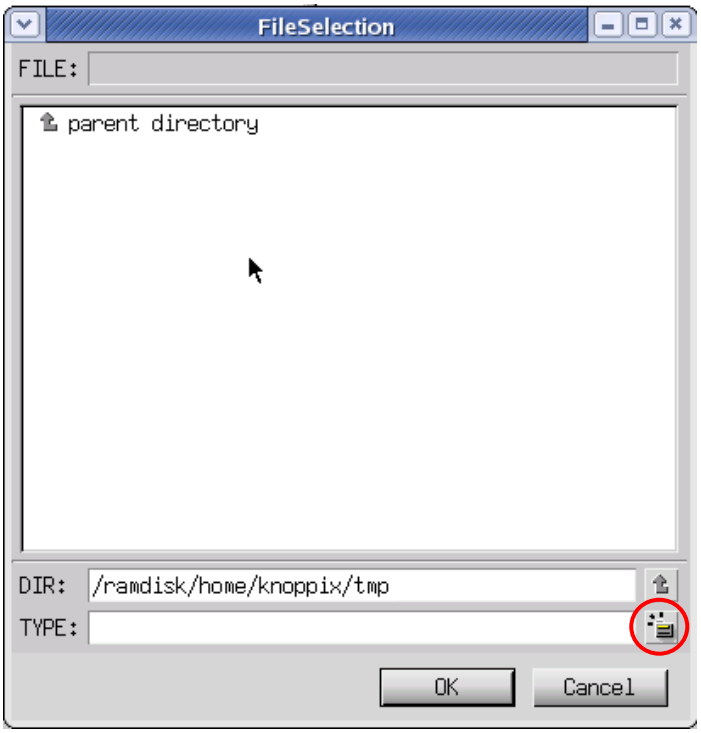

図 3-3-2-3. ファイル選択画面

DIR に「ramdisk/home/knoppix/tmp」を指定します。sample というディレクトリを作成するために、  をクリックし、「sample」と入力し「OK」ボタンをクリックします。

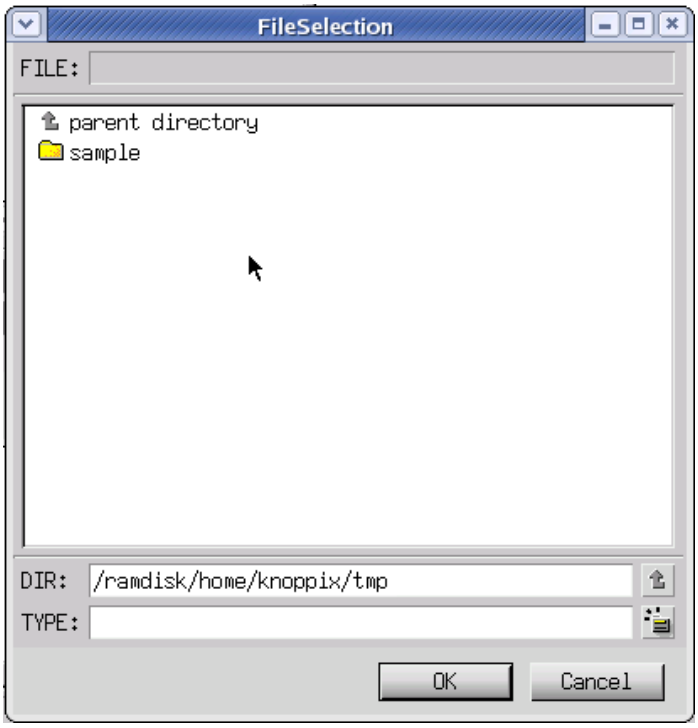

図 3-3-2-4. sample ディレクトリの作成

sample ディレクトリをダブルクリックし、DIR が「ramdisk/home/knoppix/tmp/sample」と指定されたことを確認して「OK」ボタンをクリックします。

プロジェクト名と作業ディレクトリの指定が完了しましたので「次へ」ボタンをクリックします。

図 3-3-2-5. プロジェクト名と作業ディレクトリの設定完了

プロジェクトの種別を選択します。通常のアプリケーションを作成するので、そのまま「次へ」をクリックします。

図 3-3-2-6. プロジェクト種別の選択

アプリケーションで使用する文字コード（ロケール種別）、プログラミング言語を選択します。言語種別はC/C++を選択してください。ロケール種別については注意が必要です。例えば、シリアル通信を通して、SJISの文字列が送られてきてそれを表示する場合、ここのロケール種別はSJISにするべきです。今回はLinux標準コードである日本語(EUC)を選択します。「生成」ボタンをクリックします。

※注：ALGO Smart Display で動作確認したロケール種別は、ユニコード(UTF8)と日本語(EUC)と日本語(SJIS)のみです

※注：言語種別はC/C++のみサポートしています。

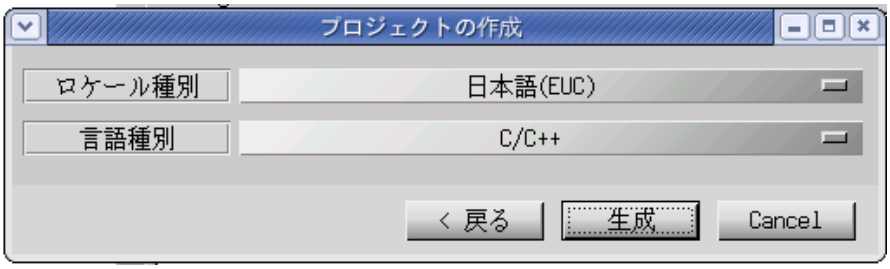

図 3-3-2-7. ロケール種別と言語種別を選択

以上でプロジェクトの作成は完了です。

３－３－３　アプリケーションウィンドウの作成

前項でプロジェクトの作成が完了しましたので、次にアプリケーションウィンドウを作成します。

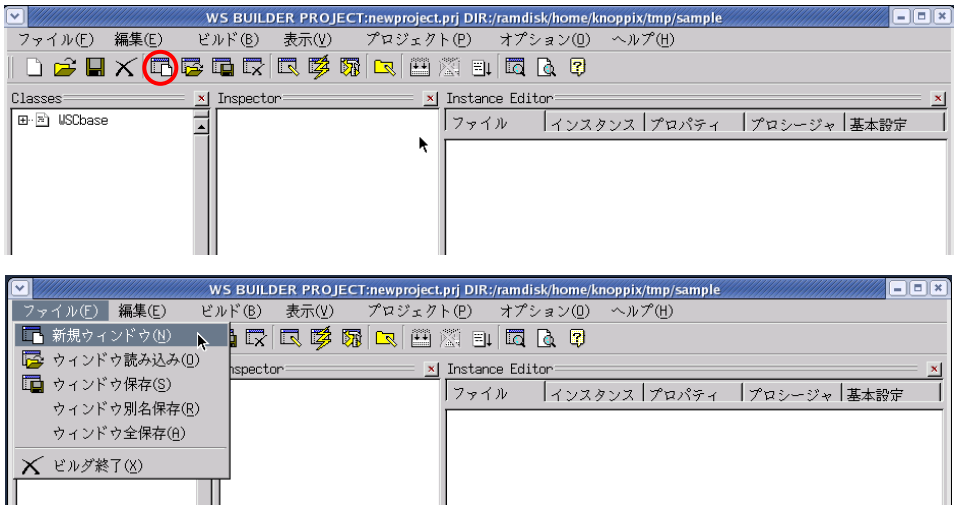

図 3-3-3-1. 新規ウィンドウ作成

 をクリックするか、メニューの「ファイル(F)」→「新規ウィンドウ(N)」をクリックすると、図 3-3-3-2 の画面が表示されます。「通常のウィンドウ」を選択し、「次へ」ボタンをクリックします。

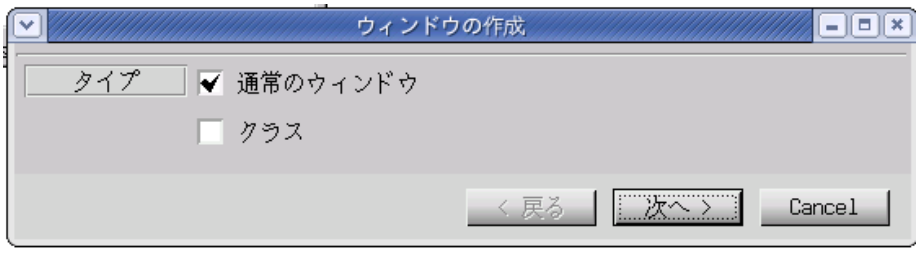

図 3-3-3-2. アプリケーションウィンドウタイプ選択

アプリケーションウィンドウの名称とプロジェクトに登録するかを選択します。名称は変数として使われるので空白のない英数字のみ有効です。基本的にメインのアプリケーションウィンドウはデフォルト設定のままにします。設定を変更せずに「次へ」をクリックします。

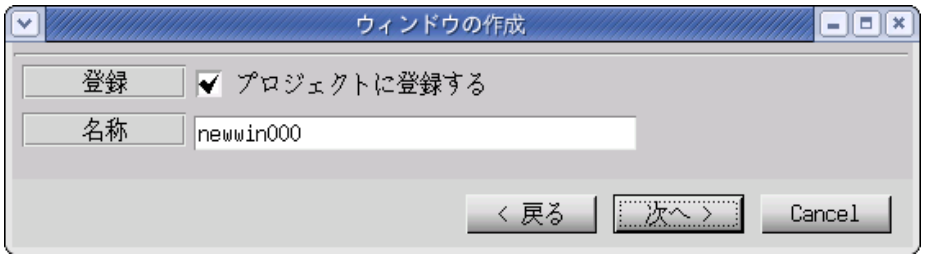

図 3-3-3-3. アプリケーションウィンドウ名称設定

テンプレートの選択を行います。あらかじめ用意されたテンプレートを選ぶことで、標準的なメニューやツールバーを持つアプリケーションウィンドウを自動的に作成することができます。今回はメニューを持たないウィンドウを作成するので、「なし」を選択して「生成」ボタンをクリックします。

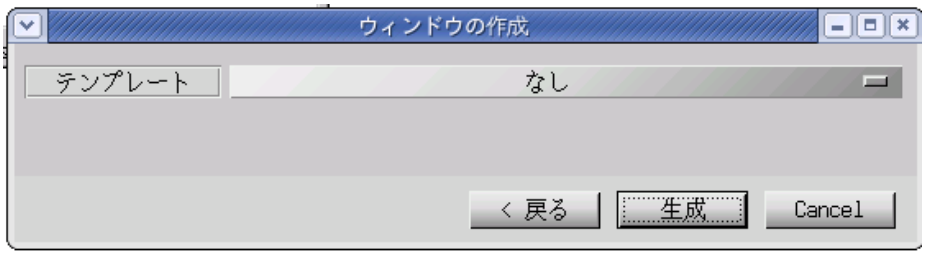

図 3-3-3-4. テンプレートの選択

これで、アプリケーションウィンドウが作成されます。作成されたアプリケーションウィンドウをクリックし、「Instance Editor」の「プロパティ」タブをクリックすると、アプリケーションウィンドウのプロパティを設定できるようになります。表 3-3-3-1 に変更する設定のみが書かれていますので変更してください。

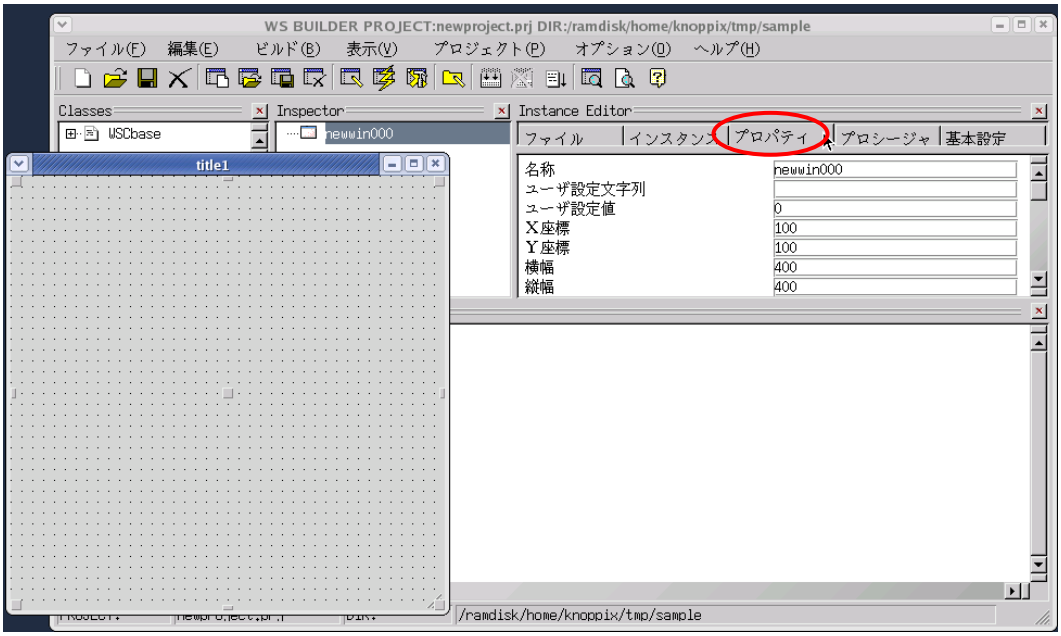

図 3-3-3-5. アプリケーションウィンドウの生成

表 3-3-3-1. アプリケーションウィンドウのプロパティ変更

プロパティ名	説明	設定値
X 座標	ウィンドウの初期起動 X 位置	0
Y 座標	ウィンドウの初期起動 Y 位置	0
横幅	ウィンドウの横幅	200
縦幅	ウィンドウの縦幅	200

以上で、アプリケーションウィンドウの作成は完了です。

３－３－４　部品の配置

アプリケーションウィンドウの上に部品を配置します。

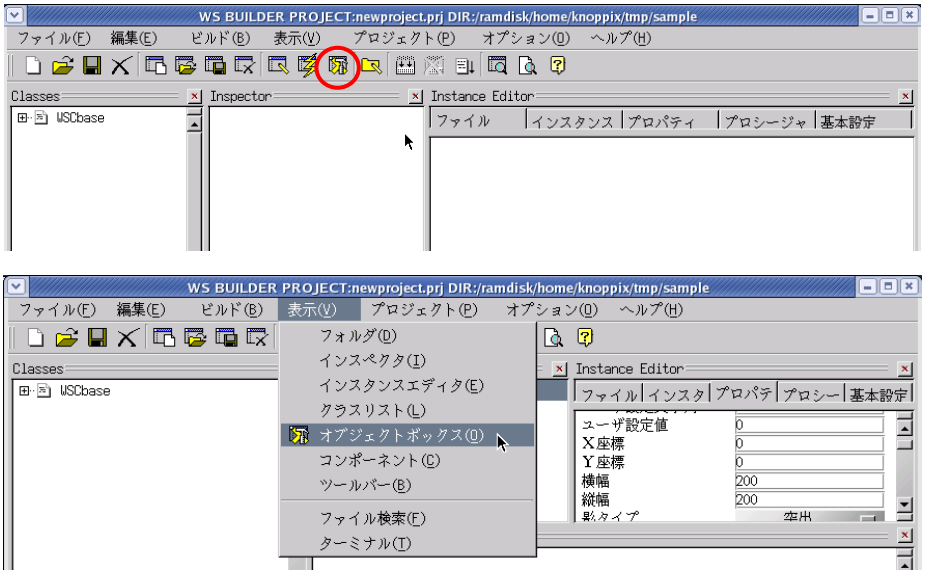

図 3-3-4-1. オブジェクトボックスの表示

 をクリックするか、メニューの「表示(V)」→「オブジェクトボックス(O)」をクリックすると図 3-3-4-2 の画面が表示されます。

図 3-3-4-2. オブジェクトボックス（Windows タブ）

ボタンを配置したいので、「Commands」タブをクリックし、ボタンオブジェクトをアプリケーションウィンドウにドラッグ＆ドロップで配置させます。

図 3-3-4-3. ボタンの配置

図 3-3-4-4 にボタンが配置された様子をしめします。他の部品も同じようにオブジェクトボックスから目的のアイコンを選び出し、ドラッグ＆ドロップして、ウィンドウ上に配置することができます。

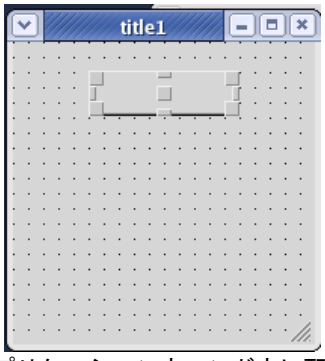

図 3-3-4-4. アプリケーションウィンドウに配置されたボタン

次に、テストサンプル作成しますのでボタンオブジェクトのみにしておきます。他の部品の詳細については、WideStudio ホームページや WideStudio の書籍等を参照してください。

※注：ALGO Smart Display は組み込み用の機器のため、CPU やメモリリソース等の関係上動作しない部品があります。

※注：Algonomix に組み込まれていないライブラリを使用している部品についてはコンパイルできません。

ボタンのプロパティを変更します。アプリケーションウィンドウ上のボタンをクリックし、「Instance Editor」の「プロパティ」タブをクリックすると、ボタンのプロパティを設定できるようになります。表 3-3-4-1 に変更する設定のみが書かれていますので変更してください。

表 3-3-4-1. ボタンのプロパティ変更

プロパティ名	説明	設定値
表示文字列	ボタンに表示される文字列の設定	PUSH

３－３－５　イベントプロシージャの設定

ボタンのクリックという動作で、なんらかのプログラムを実行したいとき、プロシージャと呼ばれるプログラムを貼り付けることで実現することができます。

アプリケーションウィンドウ上のボタンをクリックし、「Instance Editor」の「プロシージャ」タブをクリックすると、図 3-3-5-1 のような画面になります。

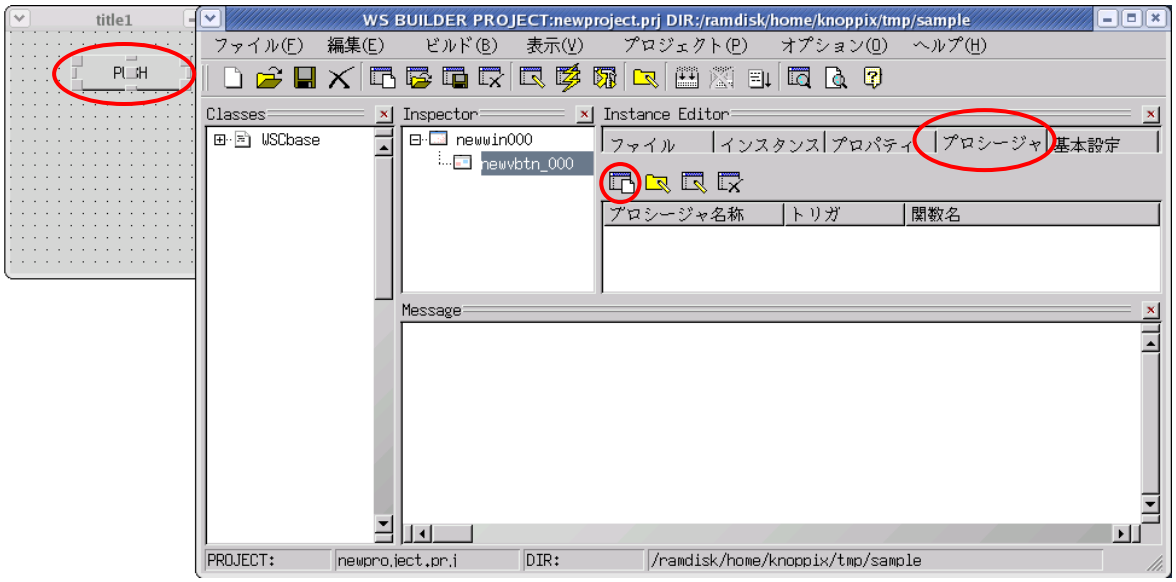

図 3-3-5-1. イベントプロシージャの作成

 をクリックすると、図 3-3-5-2 のようなイベントプロシージャ作成ダイアログが表示されます。

プロシージャ名は、イベントプロシージャを識別するための名前です。今回は、「Btn_Click」と入力します。起動関数名は、イベント発生時に起動される C/C++の関数名です。この関数に処理を記述します。今回はプロシージャ名と同じ「Btn_Click」と入力します。

起動トリガは、イベントの発生条件を選択します。今回は、ボタンを押して、離されたときに発生するイベントとして「ACTIVATE」を選択します。

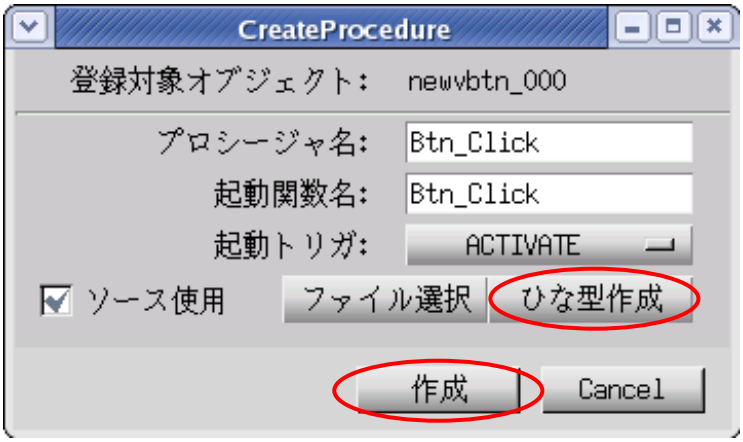

図 3-3-5-2. イベントプロシージャ作成ダイアログ

「ひな型作成」ボタンをクリックすると、確認ダイアログが表示されるので「OK」ボタンをクリックします。これで、イベントプロシージャのソースコードが自動的に生成されます。「作成」ボタンをクリックします。この操作でイベントプロシージャを作成します。

図 3-3-5-3 のように「Instance Editor」の「プロシージャ」画面に、作成したイベントプロシージャが表示されます。プロシージャ名をダブルクリックすることで、エディタを起動し、処理を記述することができます。

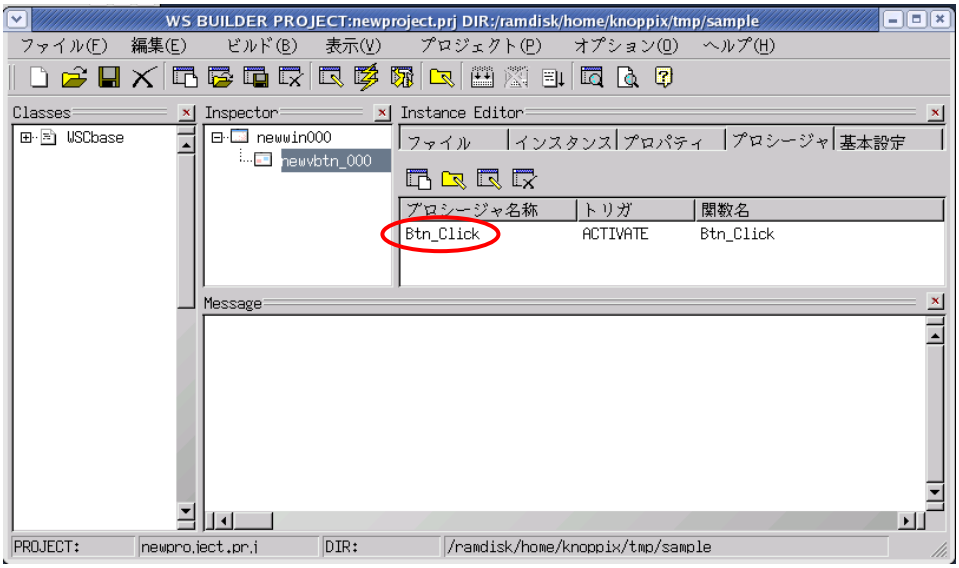

図 3-3-5-2.　作成されたイベントプロシージャ

３－３－６　　イベントプロシージャの編集

WideStudio のデフォルト設定で、起動されるエディタは「vi」となっています。「vi」とは Linux はもちろん、UNIX 系 OS で著名なテキストエディタです。しかし、Windows 上の通常のエディタとは少し変わった操作方法になっているため、Windows 上で動作するエディタ（メモ帳、秀丸等）を使いなれた方には、「vi」でエディットするのは難しいかもしれません。「vi」の使い方については、書籍やインターネットでお調べください。

Knoppix にあらかじめインストールされているエディタに、「kwrite」や「xedit」というエディタがあります。これらのエディタは、Windows 上のエディタと使い方に違いは無いと思います。「vi」では使いにくいという方にはこれらのエディタをお使いください。エディタの指定方法を以下に示します。「vi」がすでに起動されている方は終了させてください。「×」をクリックすることで終了できます。

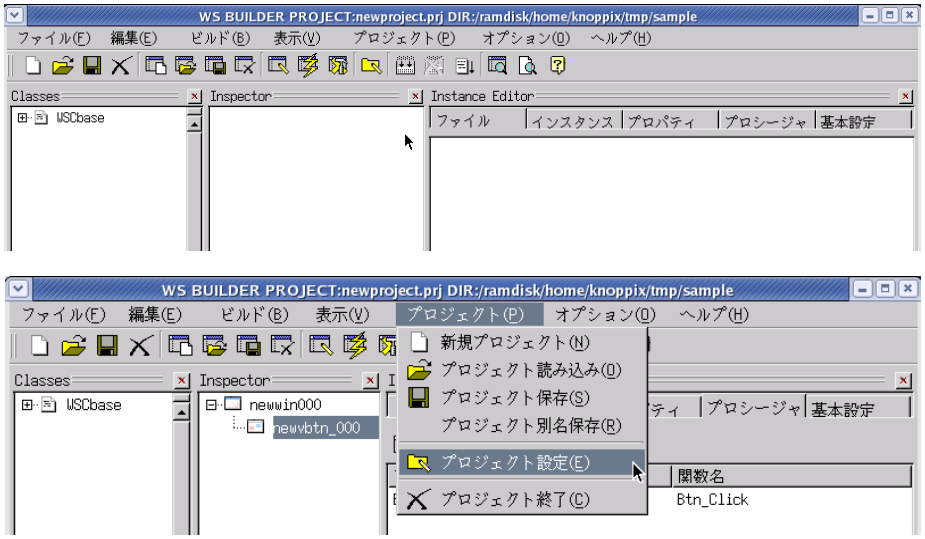

図 3-3-6-1.　プロジェクト設定画面の表示

 をクリックするか、メニューの「プロジェクト(P)」→「プロジェクト設定(E)」をクリックしてください。「環境設定」タブをクリックすると、図 3-3-6-2 のようなプロジェクト設定画面が表示されます。

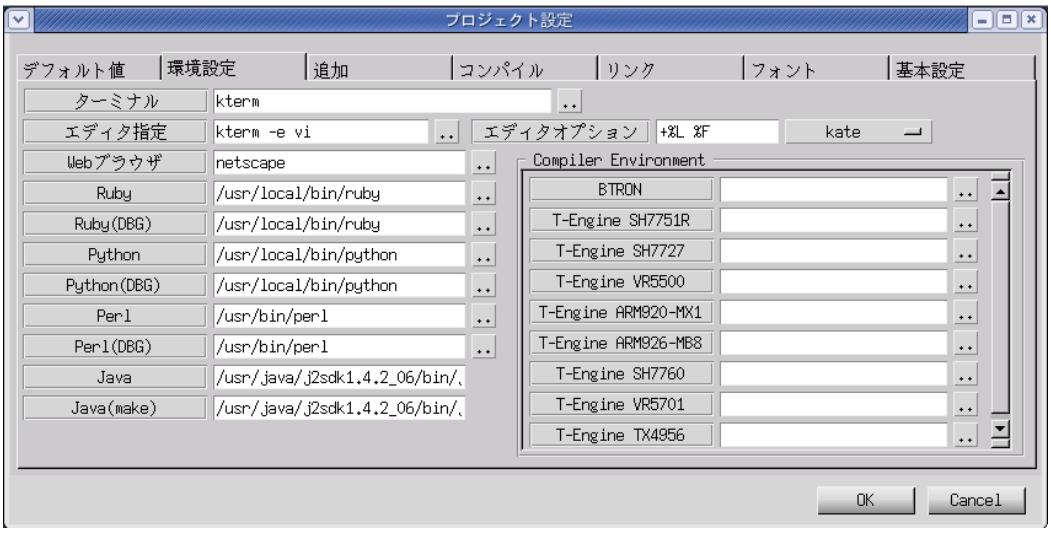

図 3-3-6-2. プロジェクト設定画面（環境設定タブ）

「エディタ指定」の項目を見てください。デフォルトでは「kterm –e vi」となっていると思います。これは、kterm というターミナルエミュレータを起動し、コンソール上で「vi」を実行するという意味です。この項目に「kwrite」または「xedit」と入力して「OK」ボタンをクリックしてください。これで、プロシージャ名をダブルクリックされたときに起動するエディタが切り替わります。

図 3-3-6-3. エディタを変更したところ

ボタンをクリックしたときに、ボタンの表示文字列を変更するコードを記述します。イベントプロシージャを作成したとき、初期状態は図 3-3-6-4 のようになっています。

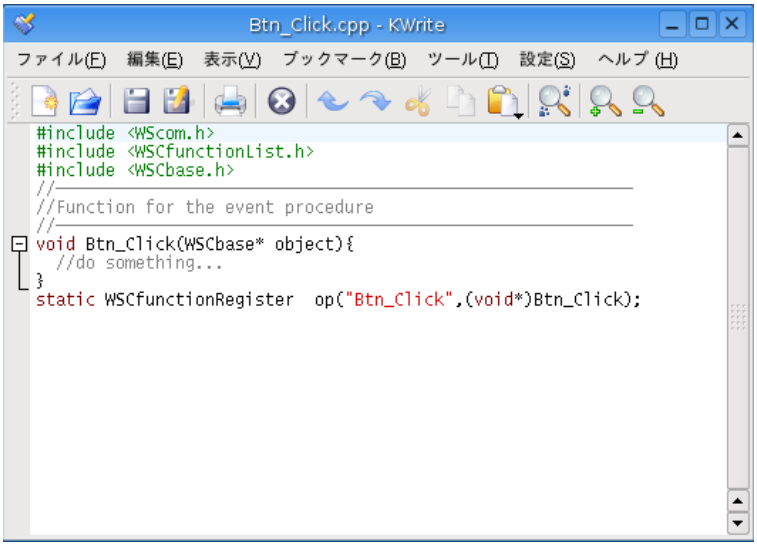

図 3-3-6-4. イベントプロシージャ初期ソースコードをエディタで開いた所

リスト 3-3-6-1 のようにコードを変更します。

リスト 3-3-6-1. 表示文字列を変更するコード

```
#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
//---------------------------------------------------------
//Function for the event procedure
//---------------------------------------------------------
void Btn_Click(WSCbase* object){
    object->setProperty(WSNlabelString,"HELLO!");        //表示文字列変更
}
static WSCfunctionRegister  op("Btn_Click",(void*)Btn_Click);
```

これで、ボタンをクリックしたら、「PUSH」が「HELLO!」となるプログラムができます。保存してエディタを終了します。以上でコーディングは完了です。

３－３－７　セルフコンパイル

サンプルプログラムのビルドを行います。メニューの「ビルド(B)→リビルド(R)」をクリックしてください。プログラムのコンパイルが始まります。

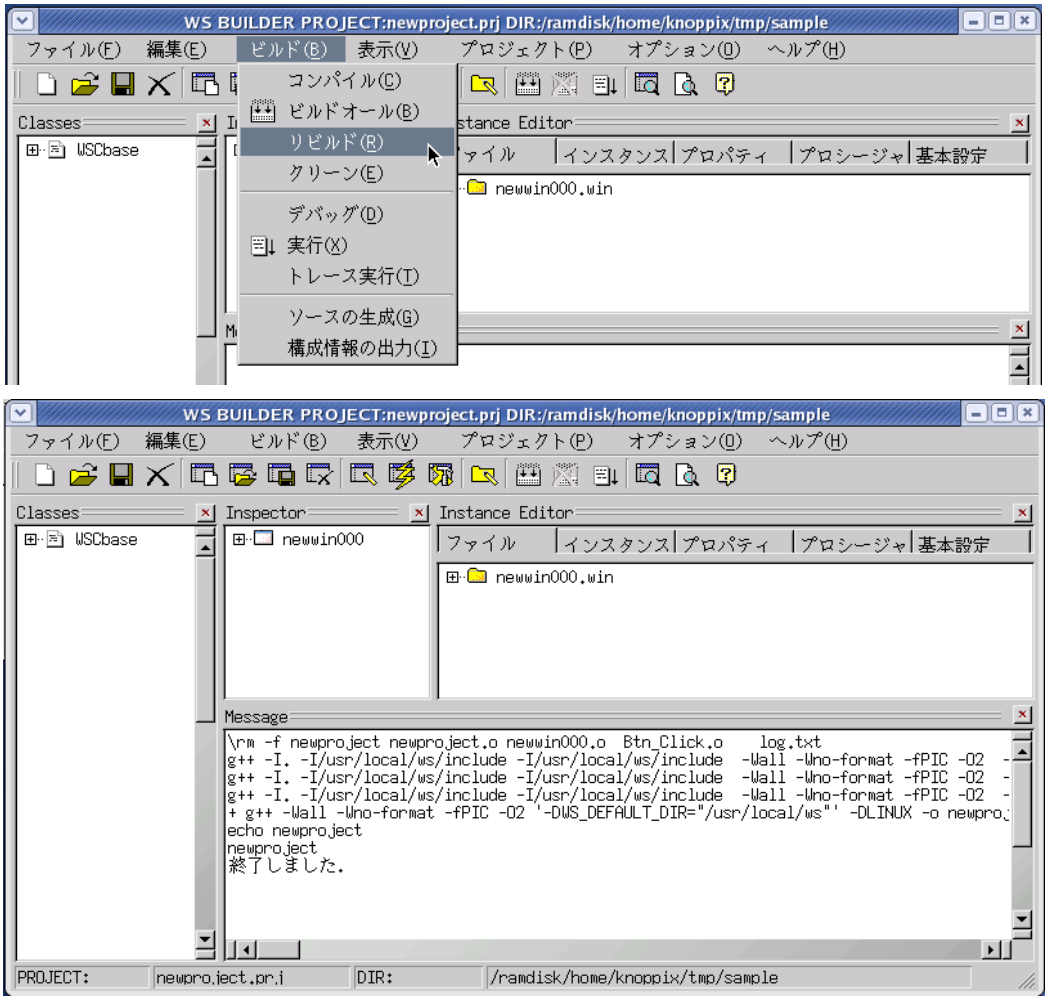

図 3-3-7-1. サンプルプロジェクトのコンパイル

コンパイルが完了したら、メニューの「ビルド(B)→実行(X)」をクリックしてください。サンプルプログラムが Knoppix 上で実行されます。「PUSH」ボタンをクリックして、「HELLO!」と表示されることを確認してください。

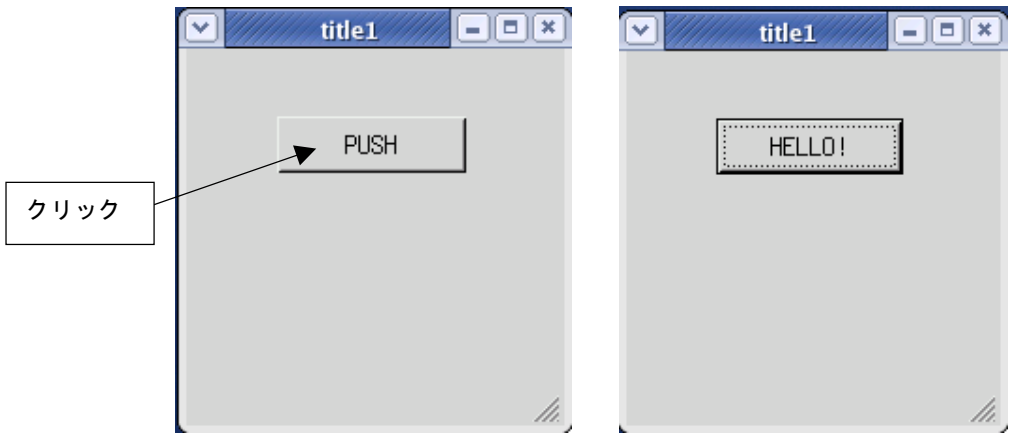

図 3-3-7-2. プログラム実行画面

動作を確認できたら、プログラムを終了させてください。メニューの「ビルド(B)→実行中止(X)」で終了できます。「終了」または「X」をクリックした場合でも、メニューの「ビルド(B)→実行中止(X)」をクリックしてください。

今行ったのは、PC 上でコンパイルして PC 上で実行したのでセルフコンパイルです。

３－３－８　クロスコンパイル

ALGO Smart Display 上で動作させるためにクロスコンパイルしてみましょう。  をクリックするか、メニューの「プロジェクト(P)」→「プロジェクト設定(E)」をクリックしてください。「基本設定」タブをクリックすると、図 3-3-8-1 のようなプロジェクト設定画面が表示されます。

ここに「TARGET」という項目があります。設定を変更していないのならここは「Native」となっていると思います。この設定でクロスコンパイルのターゲットを選択することができます。図 3-3-8-2 にターゲット一覧を示します。これは「Native」と書かれている部分をクリックすることで表示されます。

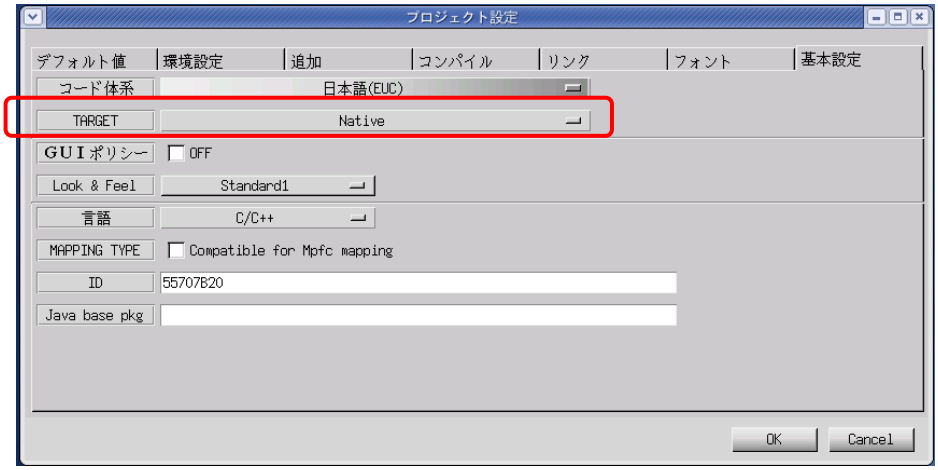

図 3-3-8-1. プロジェクト設定画面（基本設定タブ）

図 2-3-8-2. ターゲット一覧

Algonomix で選択できるターゲットは表 3-3-8-1 に書かれている 2 種類のみです。

表 3-3-8-1. Algonomix で選択できるターゲット

ターゲット名	説明
Native	Linux PC 上で動作させるための設定
Algo Smart Display (X11)	ALGO Smart Display 上の、X Windows System 上で動作させる場合の設定

※注：これら以外のターゲットではコンパイルできません

ALGO Smart Display は出荷時設定で X Window が起動されるようになっています。「TARGET」を「ALGO Smart Display(X11)」にして、「OK」をクリックしてください。

メニューの「ビルド(B)→リビルド(R)」をクリックしてください。プログラムのクロスコンパイルが始まります。図 3-3-7-1 のコンパイルログと図 3-3-8-3 のコンパイルログを見比べてください。G++となっていた部分が sh4-linux-G++となっていると思います。これがクロスコンパイルです。

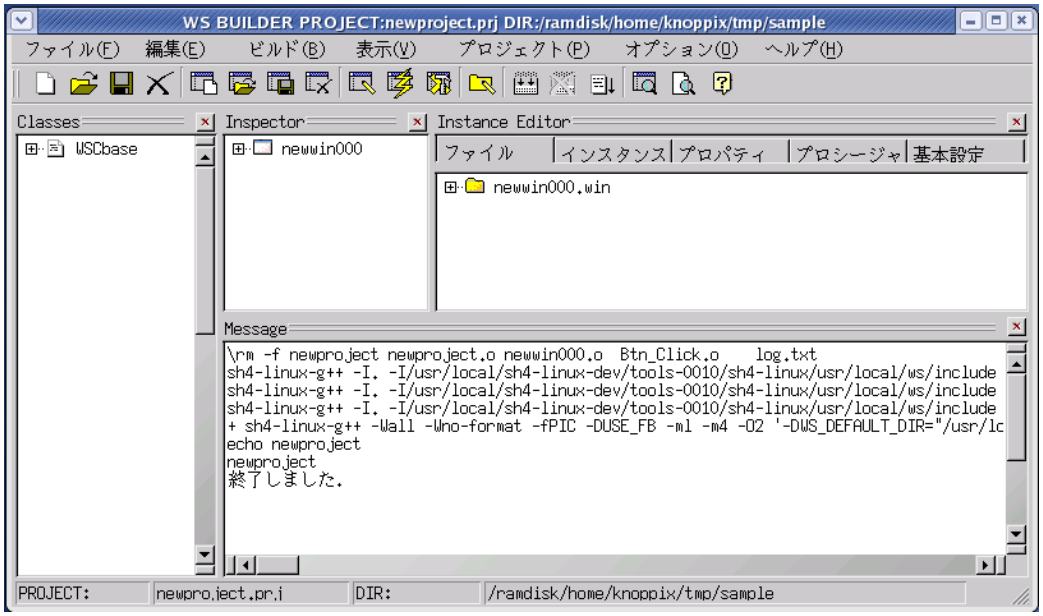

図 3-3-8-3. サンプルプロジェクトのクロスコンパイル

メニューの「ビルド(B)→実行(X)」をクリックして見てください。今度は実行することができなくなります。
今回作成された実行プログラムは SH4 用にコンパイルされているからです。

３－３－９　ファイルの転送

作成した実行プログラムを ALGO Smart Display に移して実行してみましょう。実行プログラムだけでなく、設定ファイルや画像データ等、ALGO Smart Display にデータを転送するには表 3-3-9-1 の方法があります。

表 3-3-9-1. ALGO Smart Display への転送方法

方法	AP-110	AP-210	AP-310	AP-315
USB メモリを使って転送	○	○	○	○
LAN 経由で ftp 転送	×※	○	○	○

※注：ALGO Smart Display で動作確認済みの USB LAN アダプタを使用すれば可能です

● USB メモリを使って転送

Information 1. USB メモリ自動スクリプト実行プログラムについて

USB メモリ自動スクリプト実行プログラムについて

Algo Smart Display では USB メモリを挿入したときに、自動的にスクリプトファイルを実行するプログラムが起動されます。USB メモリのなかにスクリプトファイルを入れておけば、コピーやプログラム実行など基本的なことが実行することが可能です。

スクリプトファイルとは、一連のコマンドをファイルに書き込んだもので、シリアルコンソール等で入力されたコマンドと同じように実行することができます。

スクリプトファイルはここで説明するような単純にコマンドを実行する他に、「if」等による条件判断やループを使用することができ、より複雑なスクリプトファイルを作成することができます。スクリプトファイル作成方法の詳細については、インターネットや書籍を参照してください。

① スクリプトファイルの作成

USB メモリ自動スクリプト実行プログラムで実行されるスクリプトファイルのファイル名は「download.sh」固定となっています。テキストエディタを開いてリスト 3-3-9-1 のように記述し、「download.sh」というファイル名で保存します。

保存場所は任意の場所で構いませんが、ここでは「/ramdisk/home/knoppix/tmp」というディレクトリに保存します。

リスト 3-3-9-1.「download.sh」の例

```
#!/bin/sh
cp -a /media/sda1/newproject /home/asdusr
sync
chmod 755 /home/asdusr/newproject
export DISPLAY=localhost:0.0
/home/asdusr/newproject &
```

1 行目：　この文はコメントですが、/bin/sh プログラムがスクリプトファイルのコマンドを実行する必要があるということを示すためのもので、スクリプトファイルの先頭に必ずつける必要があります。

2 行目：　USB メモリ自動スクリプト実行プログラムでは自動的に「/media/sda1」というディレクトリに USB メモリをマウントします。つまり、この行では USB メモリの中の「newproject」というファイルを「/home/asdusr」というディレクトリにコピーしています。

3 行目：　メモリにバッファされたデータをすべてディスクに書き込みます。

4 行目：　コピーしたアプリケーションに実行アクセス権を付加します。

5 行目：　X Window System で、ウィンドウを表示させたい X サーバのアドレスとディスプレイ番号(とスクリーン番号)を環境変数に登録しています。

6 行目：　「newproject」というアプリケーションをバックグランドで実行します。

② PC に USB メモリを挿入します。

③ デスクトップ上の  をクリックするとコンソールが立ち上がります。以下のコマンドを入力するこ

とで、USB メモリをマウントします。

```
$ su
# mount –t vfat /dev/sdb1 /mnt/sdb1 –o rw
```

④　実行ファイルとスクリプトファイルを、USB メモリにコピーします。

```
# cp /ramdisk/home/knoppix/tmp/sample/newproject /mnt/sdb1
# cp /ramdisk/home/knoppix/tmp/download.sh /mnt/sdb1
```

⑤　マウントを解除して USB メモリを抜きます。

```
# umount /mnt/sdb1
```

以上で USB メモリへのコピーが完了しました。

⑥　USB メモリを Algo Smart Display に挿入します。

⑦　図 3-3-9-1 のような画面が表示されます。

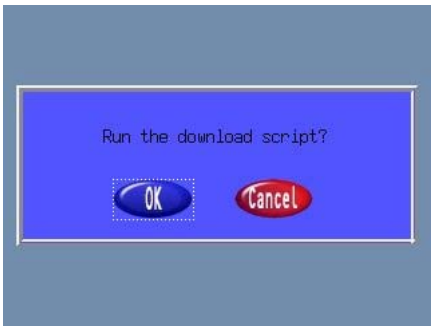

図 3-3-9-1. USB メモリ自動スクリプト実行プログラム実行画面

⑧　「OK」ボタンをクリックすると、自動的に USB メモリを「/media/sda1」にマウントし、スクリプトファイル「download.sh」を実行します。「Cancel」ボタンをクリックすると何も行いません。
AP110 の場合は「←」「→」キーでボタンを選択し、「Enter」キーで実行します。

⑨　スクリプトファイルの実行完了後、USB メモリは自動的にアンマウントされ、図 3-3-9-1 の画面が閉じられます。この状態で USB メモリを抜いてください。このときには、すでにサンプルプログラムが起動していると思います。

※注：図 3-3-9-1 の画面が表示されている間は、USB メモリを抜かないでください。ファイルが壊れる可能性があります。

⑩　サンプルプログラム実行確認

「PUSH」ボタンをクリックしたら、「HELLO!」となることを確認してください。

※注:AP110 の場合は、タッチパネルではないので USB マウスを接続して操作してください。

● LAN 経由で ftp 転送

① デスクトップ上の  をクリックするとコンソールが立ち上がります。

② クロス LAN ケーブルで PC と ALGO Smart Display を接続します。ALGO Smart Display のデフォルト IP 設定は「192.168.0.1」となっています。

③ telnet を使い、ALGO Smart Display と接続します。以下のコマンドで接続します。

```
# telnet 192.168.0.1
login:asdusr
Passwd:asdusr
$ su
#
```

※注：Passwd は入力文字を表示しません。

④ 以下のコマンドで「proftpd」という ftp プログラムを起動します。
「ASD_config」の「Server Setting」で FTP を自動的に起動する設定を行っていれば以下のコマンドは必要ありません。

```
# /etc/init.d/proftpd start
```

⑤  をクリックし、もう一つコンソールを立ち上げます。

⑥ 以下のコマンドで ALGO Smart Display の ftp と接続し、サンプルプログラムをコピーします。

```
# ftp 192.168.0.1
Connected to 192.168.0.1
220 ProftpD 1.3.0rc3 Server (ProftpD) [192.168.0.1]
Name (192.168.0.1:knoppix): asdusr
331 Password required for asdusr
Password: asdusr
230 User asdusr logged in
Remote system type is UNIX
Using binary mode to transfer files
ftp> lcd /ramdisk/home/knoppix/tmp/sample
ftp> put newproject
local newproject remote newproject
200 PORT command successful
150 Opening BINARY mode data connection for newproject
226 Transfer complete
30304 bytes sent in 0.02 secs(1185.6 kB/s)
ftp>
```

⑦ ④のコンソールの続きで以下のコマンドを実行し、サンプルプログラムを実行できるモードに変更します。

```
# chmod 755 newproject
```

⑧ サンプルプログラムを実行します。

```
# ./newproject
```

「PUSH」ボタンをクリックしたら、「HELLO!」となることを確認してください。

※注：AP110 の場合は、タッチパネルではないので USB マウスを接続して操作してください。

以上で WideStudio/MWT による、アプリケーション開発の説明は終了です。

３－３－１０ WideStudio/MWTの開発例

ここでは、今まで開発したアプリケーションの中で使った方法について列挙していきます。アプリケーション開発の参考資料としてお使いください。

● X Window 上で全画面表示させる方法

X Window 上でアプリケーションを実行したら必ず、タスクバーがついてきます。これは、X Window 上で Window Manager が動作しており、Window Manager がタスクバーをつけ、複数のアプリケーションを管理しています。組み込み用途で使用する場合、メイン画面は全画面表示して欲しい場合があります。表 3-3-10-1 のプロパティを変更することで、タスクバーのついていないアプリケーションを開発することができます。この場合、起動したプログラムが常に前に表示されるので、アプリケーションウィンドウの表示、非表示を切り替えて使用する必要があります。

表 3-3-10-1. アプリケーションウィンドウのプロパティ変更

プロパティ名	説明	設定値
タイトル属性	ウィンドウのタスクバー属性の設定	WM 管理外
終了	ウィンドウを非表示にしたとき終了するかしないかの設定	オフ

サンプルとして作成した、HELLO プログラムを修正して、試してみましょう。アプリケーションウィンドウのプロパティを表 3-3-10-1 に書かれているように変更してください。Btn_Click のイベントプロシージャをリスト 3-3-10-1 のように記述してください。

リスト 3-3-10-1. 別アプリケーションの起動サンプル

```
#include "stdlib.h"
#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
#include "newwin000.h"
//--------------------------------------------------------
//Function for the event procedure
//--------------------------------------------------------
void Btn_Click(WSCbase* object){
    newwin000->setVisible(False);                    //アプリケーションウィンドウの非表示
    system("xeyes");                                 //xeyes というプログラムを起動する
    newwin000->setVisible(True);                     //アプリケーションウィンドウの表示
}
static WSCfunctionRegister  op("Btn_Click",(void*)Btn_Click);
```

このプログラムをコンパイルして実行してみてください。実行したアプリケーションにはタスクバーがついていないと思います。ボタンをクリックしてみてください、メイン画面が消えて、マウスポイントを追いかける目玉アプリケーションが起動すると思います。目玉アプリケーションを終了すると、再度メイン画面が表示されます。

● WideStudio のフォント設定について

WideStudio には次の４系統のフォント設定があります。

1. X11 系の設定
   FONT0:サイズ フォント名 Weight(0:無し 1:bold) Slant(0:無し 1:イタリック)
   [例] FONT0:10 * 0 0
   FONT1:12 * 0 0
   Algo Smart Display の/etc/xunicoderc にて詳細なフォントを定義
2. T-Engine 系の設定
   FONT0:サイズ フォント ID
   [例] FONT0:10 60c6
   FONT1:12 60c6
3. Linux フレームバッファ系、T-Engine フレームバッファ系
   FONT0:font0
   [例] FONT0:font0
   FONT1:font1
   /etc/wsfonts にてフォントファイルを定義
4. Windows 系の設定
   Windows フォントパラメータをカンマで列挙

WideStudio のフォント設定は、本来ならプロジェクト設定のフォント設定タブで設定します。しかし、現状の WideStudio では「TARGET」を「Native」以外にしたとき、かならず、上記の 2 の設定を行うようになってしまいます。このため、フォント設定は、「prj ファイル」に記録されているので、この値を直接変更することでフォント設定を行います。

※注： Algo Smart Display 上のフォントと開発環境上のフォントはインストールされているものが違うため、開発時に見えているフォントと実際に動作させたときのフォントが違います。

Algo Smart Display 上で WideStudio アプリケーションを動作させる時のフォント設定方法を以下に示します。

リスト 3-3-10-2 に prj ファイルに記述する X11 用のフォント設定例を、リスト 3-3-10-3 に Algo Smart Display 上にある「/etc/xunicoderc」を示します。

リスト 3-3-10-2. X11 用のフォント設定例（prj ファイル）

```
#FONT0
12 * 0 0        /* サイズ 12 フォント名 *       Weight 無し  slant 無し */
#FONT1
8 * 0 0         /* サイズ 8  フォント名 *       Weight 無し  slant 無し */
#FONT2
10 * 0 0        /* サイズ 10 フォント名 *       Weight 無し  slant 無し */
#FONT3
14 * 0 0        /* サイズ 14 フォント名 *       Weight 無し  slant 無し */
#FONT4
18 * 1 0        /* サイズ 18 フォント名 *       Weight bold  slant 無し */
#FONT5
24 * 0 1        /* サイズ 24 フォント名 *       Weight 無し  slant イタリック */
#FONT6
30 fixed 1 1    /* サイズ 30 フォント名 fixed   Weight bold  slant イタリック */
#FONT7
34 * 0 0        /* サイズ 34 フォント名 *       Weight 無し  slant 無し */
```

リスト 3-3-10-3. Algo Smart Display の/etc/xunicoderc(出荷時)

```
fixed     -misc-fixed-medium-r-*-*-%d-*-*-*-*-*-jisx0201.1976-0, ¥
          -misc-fixed-medium-r-*-*-%d-*-*-*-*-*-jisx0208.1983-0, ¥
          -daewoo-gothic-medium-r-*-*-%d-*-*-*-*-*-ksc5601.1987-0, ¥
          -isas-fangsong ti-medium-r-*-*-%d-*-*-*-*-*-gb2312.1980-0, ¥
          -misc-fixed-medium-r-*-*-%d-*-*-*-*-*-iso8859-1, ¥
          -misc-fixed-medium-r-*-*-%d-*-*-*-*-*-iso10646-1
courier   -*-gothic-medium-r-*-*-%d-*-*-*-*-*-jisx0201.1976-0, ¥
          -*-gothic-medium-r-*-*-%d-*-*-*-*-*-jisx0208.1983-0, ¥
          -daewoo-gothic-medium-r-*-*-%d-*-*-*-*-*-ksc5601.1987-0, ¥
          -isas-fangsong ti-medium-r-*-*-%d-*-*-*-*-*-gb2312.1980-0, ¥
          -ibm-courier-medium-r-*-*-%d-*-*-*-*-*-iso8859-1, ¥
          -ibm-courier-medium-r-*-*-%d-*-*-*-*-*-iso10646-1
helvetica -*-gothic-medium-r-*-*-%d-*-*-*-*-*-jisx0201.1976-0, ¥
          -*-gothic-medium-r-*-*-%d-*-*-*-*-*-jisx0208.1983-0, ¥
          -daewoo-gothic-medium-r-*-*-%d-*-*-*-*-*-ksc5601.1987-0, ¥
          -isas-fangsong ti-medium-r-*-*-%d-*-*-*-*-*-gb2312.1980-0, ¥
          -adobe-helvetica-medium-r-*-*-%d-*-*-*-*-*-iso8859-1, ¥
          -adobe-helvetica-medium-r-*-*-%d-*-*-*-*-*-iso10646-1
*         -*-gothic-medium-r-*-*-%d-*-*-*-*-*-jisx0201.1976-0,¥
          -*-gothic-medium-r-*-*-%d-*-*-*-*-*-jisx0208.1983-0, ¥
          -daewoo-gothic-medium-r-*-*-%d-*-*-*-*-*-ksc5601.1987-0, ¥
          -isas-fangsong ti-medium-r-*-*-%d-*-*-*-*-*-gb2312.1980-0, ¥
          -*-gothic-medium-r-*-*-%d-*-*-*-*-*-iso8859-1,¥
          -*-gothic-medium-r-*-*-%d-*-*-*-*-*-iso10646-1
fixed.b   -*-gothic-bold-r-*-*-%d-*-*-*-*-*-jisx0201.1976-0,¥
          -*-gothic-bold-r-*-*-%d-*-*-*-*-*-jisx0208.1983-0, ¥
          -daewoo-gothic-medium-r-*-*-%d-*-*-*-*-*-ksc5601.1987-0, ¥
          -isas-fangsong ti-medium-r-*-*-%d-*-*-*-*-*-gb2312.1980-0, ¥
          -misc-fixed-bold-r-*-*-%d-*-*-*-*-*-iso8859-1, ¥
          -misc-fixed-bold-r-*-*-%d-*-*-*-*-*-iso10646-1
courier.b -*-gothic-bold-r-*-*-%d-*-*-*-*-*-jisx0201.1976-0, ¥
          -*-gothic-bold-r-*-*-%d-*-*-*-*-*-jisx0208.1983-0, ¥
          -daewoo-gothic-medium-r-*-*-%d-*-*-*-*-*-ksc5601.1987-0, ¥
          -isas-fangsong ti-medium-r-*-*-%d-*-*-*-*-*-gb2312.1980-0, ¥
          -ibm-courier-bold-r-*-*-%d-*-*-*-*-*-iso8859-1, ¥
          -ibm-courier-bold-r-*-*-%d-*-*-*-*-*-iso10646-1
helvetica.b -*-gothic-bold-r-*-*-%d-*-*-*-*-*-jisx0201.1976-0, ¥
            -*-gothic-bold-r-*-*-%d-*-*-*-*-*-jisx0208.1983-0, ¥
            -daewoo-gothic-medium-r-*-*-%d-*-*-*-*-*-ksc5601.1987-0, ¥
            -isas-fangsong ti-medium-r-*-*-%d-*-*-*-*-*-gb2312.1980-0, ¥
            -adobe-helvetica-bold-r-*-*-%d-*-*-*-*-*-iso8859-1, ¥
            -adobe-helvetica-bold-r-*-*-%d-*-*-*-*-*-iso10646-1
*.b       -*-gothic-bold-r-*-*-%d-*-*-*-*-*-jisx0201.1976-0,¥
          -*-gothic-bold-r-*-*-%d-*-*-*-*-*-jisx0208.1983-0, ¥
          -daewoo-gothic-medium-r-*-*-%d-*-*-*-*-*-ksc5601.1987-0, ¥
          -isas-fangsong ti-medium-r-*-*-%d-*-*-*-*-*-gb2312.1980-0, ¥
          -*-gothic-bold-r-*-*-%d-*-*-*-*-*-iso8859-1,¥
          -*-gothic-bold-r-*-*-%d-*-*-*-*-*-iso10646-1
fixed.i   -*-gothic-medium-i-*-*-%d-*-*-*-*-*-jisx0201.1976-0,¥
          -*-gothic-medium-i-*-*-%d-*-*-*-*-*-jisx0208.1983-0, ¥
```

```
          -daewoo-gothic-medium-r-*-*-%d-*-*-*-*-*-ksc5601.1987-0, ¥
          -isas-fangsong ti-medium-r-*-*-%d-*-*-*-*-*-gb2312.1980-0, ¥
          -*-gothic-medium-i-*-*-%d-*-*-*-*-*-iso8859-1,¥
          -*-gothic-medium-i-*-*-%d-*-*-*-*-*-iso10646-1
courier.i -*-gothic-medium-i-*-*-%d-*-*-*-*-*-jisx0201.1976-0, ¥
          -*-gothic-medium-i-*-*-%d-*-*-*-*-*-jisx0208.1983-0, ¥
          -daewoo-gothic-medium-r-*-*-%d-*-*-*-*-*-ksc5601.1987-0, ¥
          -isas-fangsong ti-medium-r-*-*-%d-*-*-*-*-*-gb2312.1980-0, ¥
          -ibm-courier-medium-i-*-*-%d-*-*-*-*-*-iso8859-1, ¥
          -ibm-courier-medium-i-*-*-%d-*-*-*-*-*-iso10646-1
helvetica.i -*-gothic-medium-i-*-*-%d-*-*-*-*-*-jisx0201.1976-0,¥
            -*-gothic-medium-i-*-*-%d-*-*-*-*-*-jisx0208.1983-0, ¥
            -daewoo-gothic-medium-r-*-*-%d-*-*-*-*-*-ksc5601.1987-0, ¥
            -isas-fangsong ti-medium-r-*-*-%d-*-*-*-*-*-gb2312.1980-0, ¥
            -*-gothic-medium-i-*-*-%d-*-*-*-*-*-iso8859-1,¥
            -*-gothic-medium-i-*-*-%d-*-*-*-*-*-iso10646-1
*.i       -*-gothic-medium-i-*-*-%d-*-*-*-*-*-jisx0201.1976-0,¥
          -*-gothic-medium-i-*-*-%d-*-*-*-*-*-jisx0208.1983-0, ¥
          -daewoo-gothic-medium-r-*-*-%d-*-*-*-*-*-ksc5601.1987-0, ¥
          -isas-fangsong ti-medium-r-*-*-%d-*-*-*-*-*-gb2312.1980-0, ¥
          -*-gothic-medium-i-*-*-%d-*-*-*-*-*-iso8859-1,¥
          -*-gothic-medium-i-*-*-%d-*-*-*-*-*-iso10646-1
fixed.b.i -*-gothic-bold-i-*-*-%d-*-*-*-*-*-jisx0201.1976-0,¥
          -*-gothic-bold-i-*-*-%d-*-*-*-*-*-jisx0208.1983-0, ¥
          -daewoo-gothic-medium-r-*-*-%d-*-*-*-*-*-ksc5601.1987-0, ¥
          -isas-fangsong ti-medium-r-*-*-%d-*-*-*-*-*-gb2312.1980-0, ¥
          -*-gothic-bold-i-*-*-%d-*-*-*-*-*-iso8859-1, ¥
          -*-gothic-bold-i-*-*-%d-*-*-*-*-*-iso10646-1
courier.b.i -*-gothic-bold-i-*-*-%d-*-*-*-*-*-jisx0201.1976-0, ¥
            -*-gothic-bold-i-*-*-%d-*-*-*-*-*-jisx0208.1983-0, ¥
            -daewoo-gothic-medium-r-*-*-%d-*-*-*-*-*-ksc5601.1987-0, ¥
            -isas-fangsong ti-medium-r-*-*-%d-*-*-*-*-*-gb2312.1980-0, ¥
            -ibm-courier-bold-i-*-*-%d-*-*-*-*-*-iso8859-1, ¥
            -ibm-courier-bold-i-*-*-%d-*-*-*-*-*-iso10646-1
helvetica.b.i -*-gothic-bold-i-*-*-%d-*-*-*-*-*-jisx0201.1976-0, ¥
              -*-gothic-bold-i-*-*-%d-*-*-*-*-*-jisx0208.1983-0, ¥
              -daewoo-gothic-medium-r-*-*-%d-*-*-*-*-*-ksc5601.1987-0, ¥
              -isas-fangsong ti-medium-r-*-*-%d-*-*-*-*-*-gb2312.1980-0, ¥
              -*-gothic-bold-i-*-*-%d-*-*-*-*-*-iso8859-1,¥
              -*-gothic-bold-i-*-*-%d-*-*-*-*-*-iso10646-1
*.b.i     -*-gothic-bold-i-*-*-%d-*-*-*-*-*-jisx0201.1976-0,¥
          -*-gothic-bold-i-*-*-%d-*-*-*-*-*-jisx0208.1983-0, ¥
          -daewoo-gothic-medium-r-*-*-%d-*-*-*-*-*-ksc5601.1987-0, ¥
          -isas-fangsong ti-medium-r-*-*-%d-*-*-*-*-*-gb2312.1980-0, ¥
          -*-gothic-bold-i-*-*-%d-*-*-*-*-*-iso8859-1,¥
          -*-gothic-bold-i-*-*-%d-*-*-*-*-*-iso10646-1
```

図 3-3-10-1 に「xfontsel」というプログラムの起動画面を示します。X Window System では、図のようなフォーマットでフォントを指定しています。

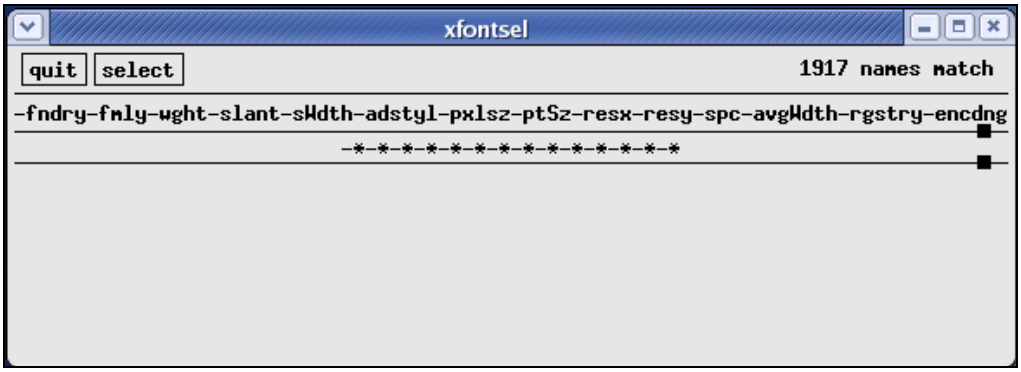

図 3-3-10-1. xfontsel 起動画面

X Window System のフォントフォーマットの詳細を表 3-3-10-2 に示します。

表 3-3-10-2. X Window System フォントフォーマットについて

項目名	意味	補記
fndry	フォントのベンダ名	この２つの項目でフォントを特定しています。
fmly	フォントの種類	
wght	フォントの太さ	フォントの太さを指定します。 bold：太字 medium：普通
slant	フォントの傾き	フォントの傾きを指定します。 r：ローマン体 i：イタリック体 o：オブリーク体
sWdth	フォントの幅	
adstyl	追加スタイル	
pxlsz	フォントの大きさ （ピクセル単位）	フォントサイズを指定します。 WideStudio ではここを%d とし、prj ファイルでサイズを指定できます。
ptSz	フォントの大きさ （ポイント単位）	
resx	フォントの X 方向の解像度	
resy	フォントの Y 方向の解像度	
spc	フォント幅の方式	
avgWdth	フォントの平均幅	
rgstry	文字集合名 （文字コードセット）	登録された組織もしくは標準名を指定します。 iso8859　　：英数半角文字 iso10646　　：2 バイト文字 jisx0201.1976：英数カナ半角文字 jisx0208.1983：日本語文字コード ksc5601.1987　：韓国語文字コード gb2312.1980　：中国語文字コード
encdng	エンコード方式	

このプログラムで表示できるフォントフォーマットが、「xunicoderc」ファイルに記述されています。prj ファイルの設定値と「xunicoderc」ファイルの対応図を図 3-3-10-2 に示します。

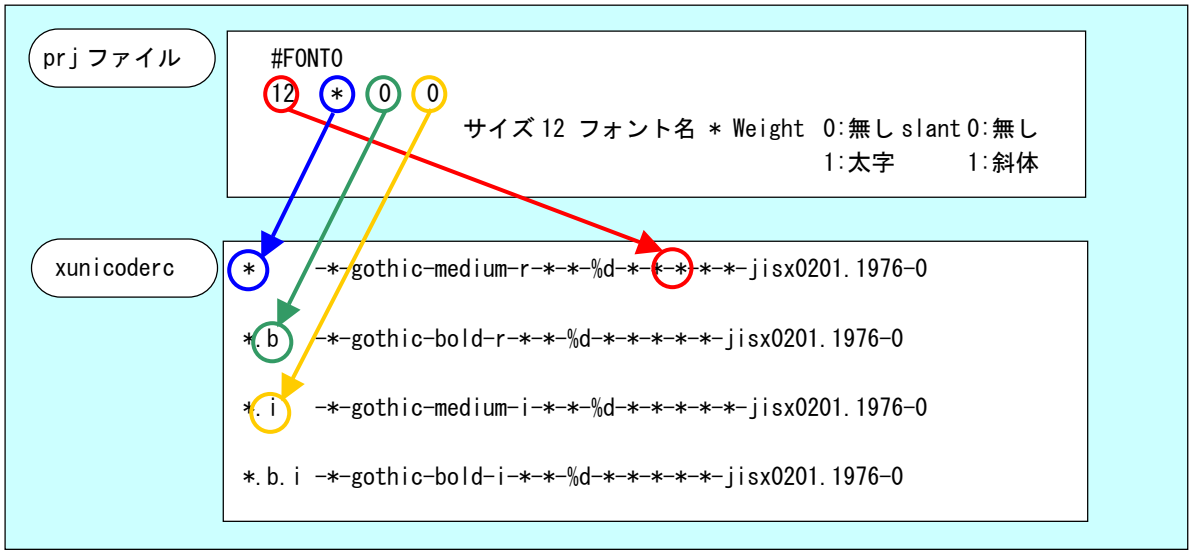

図 3-3-10-1. xfontsel 起動画面

「*」の部分を自由に指定することができます。「xunicoderc」に新しいフォントタイプを設定するときは、「太字、斜体無し」、「太字のみ」、「斜体のみ」、「太字、斜体」の 4 つで一つの組み合わせとします。

WideStudio では prj ファイルで 8 個までのフォントを指定することができます。それぞれのオブジェクトの「フォント番号」プロパティに設定した番号のフォントが表示されます。「フォント番号」のデフォルト設定である「8」という設定は「#FONT0」の設定が使用されます。

使用するフォントがビットマップフォントの場合、対応するサイズのフォントがないときは、WideStudio で存在するサイズのビットマップフォントを拡大、縮小して使用します。

※注：表示するときに拡大、縮小を行うため、表示に時間がかかる場合があります。

Algo Smart Display に実装されているフォント以外でも、新しいフォントを X Window System に登録することで、使用することが可能です。

３－４　DDDについて

Algonomix 開発環境では、WidoStudio の標準デバッグツールとして DDD を採用しています。DDD とは後述する GDB というコンソール用デバッグツールに GUI の殻をかぶせたものです。

GDB を使用する場合、コンパイルオプションとして「-g」や「-ggdb」をつけてコンパイルする必要があります。

WideStudio の場合、  をクリックするか、メニューの「プロジェクト(P)」→「プロジェクト設定(E)」をクリックしてください。「コンパイル」タブをクリックすると、図 3-4-1 のようなプロジェクト設定画面が表示されます。

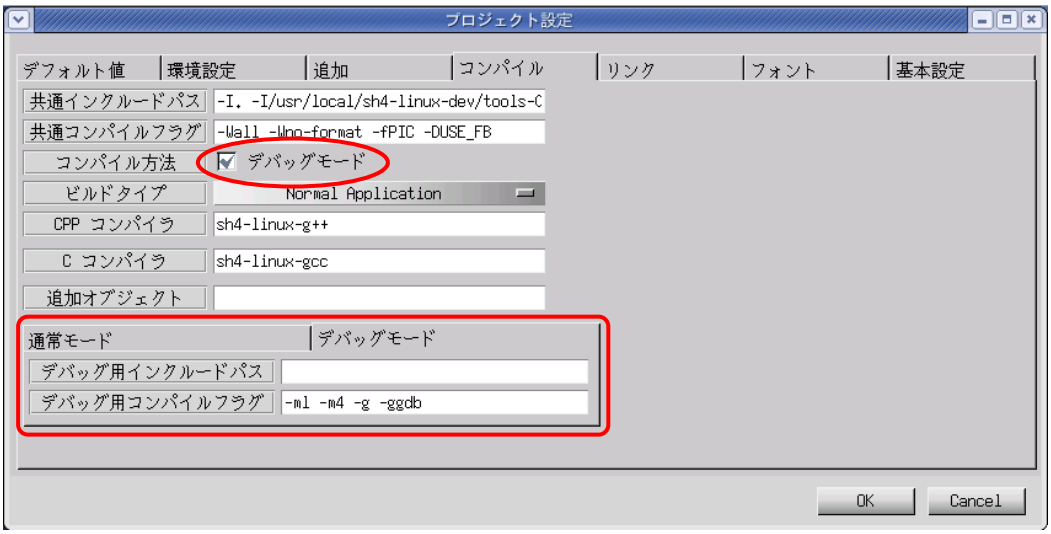

図 3-4-1. プロジェクト設定画面（コンパイルタブ）

「コンパイル方法」という項目の、「デバッグモード」というチェックボックスにチェックを入れます。これでデバッグモードのコンパイルフラグが使用されるようになります。デバッグモードのタブに「-ml -m4 -g -ggdb」というデバッグ用コンパイルフラグが設定されていますので確認してください。

「OK」ボタンをクリックし、プロジェクト全体をコンパイルし直します。通常モードで作成される実行プログラム名に「d」が付いた実行プログラムが作成されます。(「newproject」→「newprojectd」)。

①DDD の起動

リビルド実行後、ビルド→デバッグを実行すると、DDD が起動します。

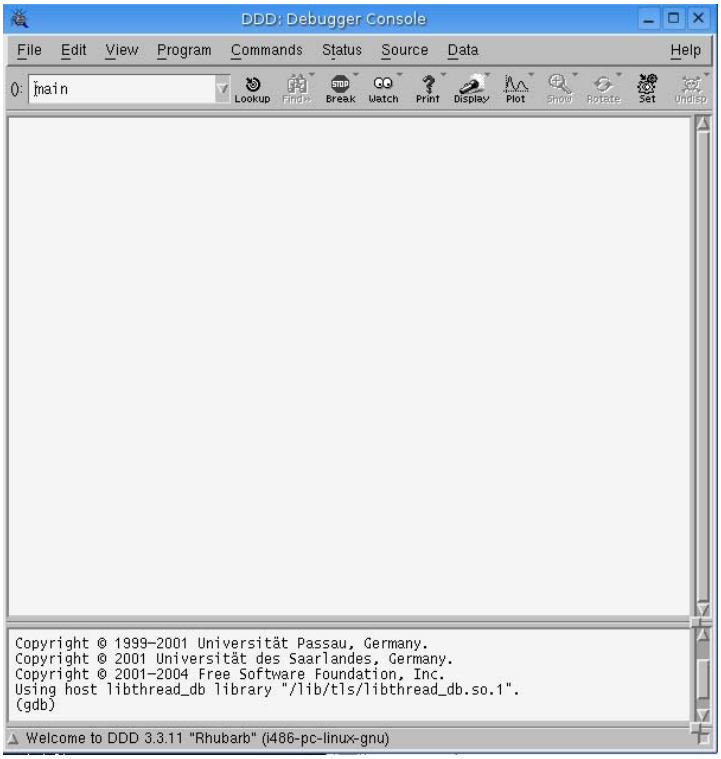

図 3-4-2. DDD 起動画面

DDD でファイルメニューから OpenSource を選択してソースを選択します。

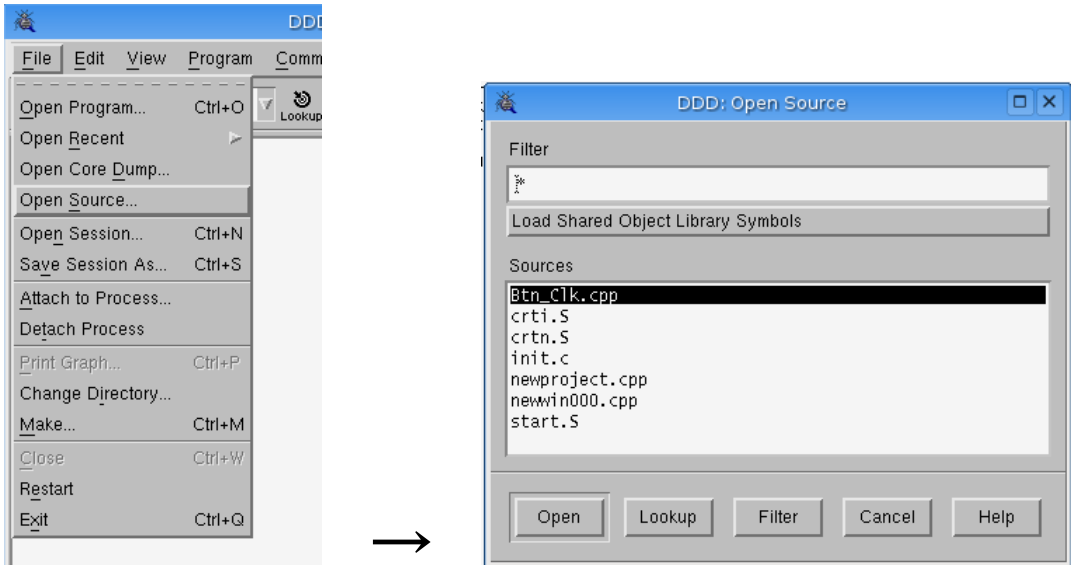

図 3-4-3. ソースの選択

②ブレークポイントの設定

ソースの中から、実行を一時的に止めたい場所にブレークポイントを設定します。

先ほど作成したサンプルプログラムで、ボタンをクリックされた時の処理をデバッグしたいなら、Btn_clk 関数のまえにカーソルを置いてブレークボタンをクリックします。

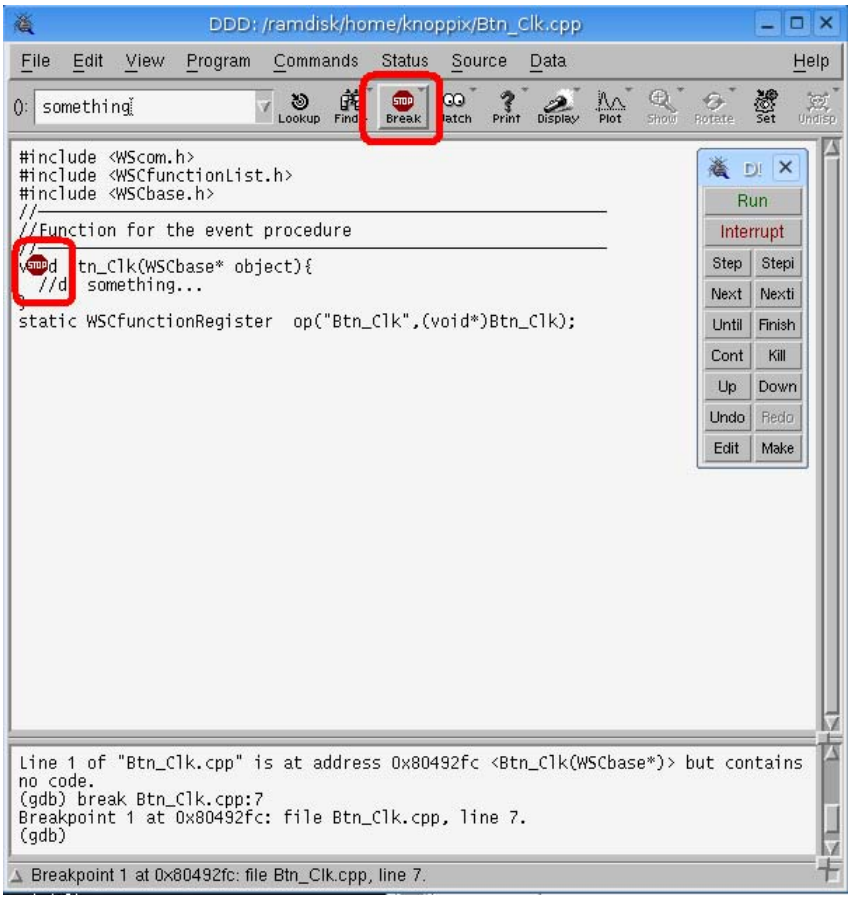

図 3-4-4.　ブレーク設定

ソース上に  アイコンがセットされます。

コマンドボタンウインドウ上の RUN ボタンをクリックするとプログラムが実行され、プログラムのボタンをクリックすると、  アイコンのところで実行が止まり、現在実行中の行が矢印で表示されます。

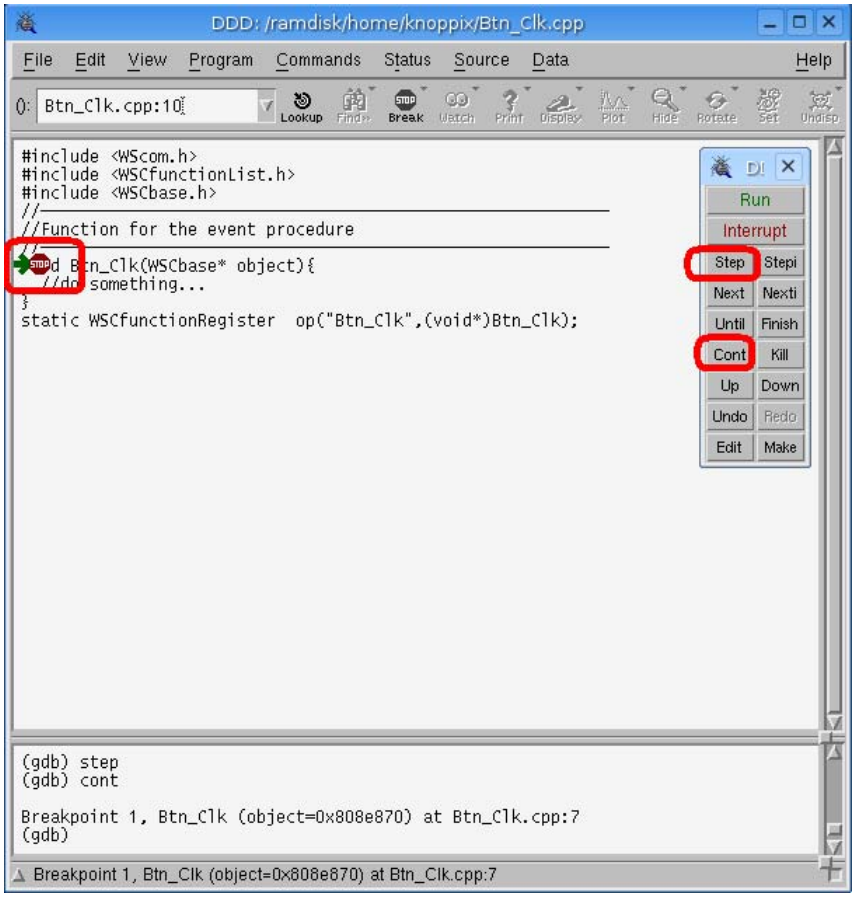

図 3-4-4. ブレークによる実行停止

③ブレークポイントでの変数の内容表示

ソースウインドウで、表示させたい変数名をドラッグして選択反転させ、そこで右クリックして、"Print 変数名"アイテムを選ぶとコマンドラインに変数の中身が表示さます。

"Display 変数名"アイテムを選択すると、変数表示領域に変数の内容が表示され、ブレークポイントで停止した時点での変数の内容を常に表示してくれます。

また，ポインタ変数の場合、"Print *変数名"及び"Display *変数名"のアイテムを選択すると、ポインタ変数が示す先（メモリ番地）の内容を表示してくれます。

④一旦停止時からの操作

cont ボタン：　Continue 操作を意味し，次にブレークポイントに到達するまで実行続行します。

next ボタン：　一行ずつ実行します。ただし関数呼出し時は，関数内部のコードは表示せずに関数の実行を完了させて、その次の行に制御が移ります。(コマンドラインから next 数値とすると、その数値の行数分をまとめて実行します)。

step ボタン：　一行ずつ実行します。関数呼出し時には関数内のコードも一行ずつ実行します（コマンドラインから step 数値とすると，その数値の行数分をまとめて実行します)。

⑤ブレークポイントの解除

ブレークポイントを表す  アイコンを右クリックして Delete Breakpoint を選択すると、そのブレークポイントは解除されます。一時的に解除したい場合は、Disable Breakpoint を選択し、一時解除していたブレークポイントを再開するには、Enable Breakpoint を選択します。

３－５　GDBによるデバッグ方法

Algo Smart Display の Algonomix 上で実行されるプログラムをデバッグするのに GDB や GDB サーバを使用します。Algo Smart Display 上では GDB サーバを実行させ、PC 側では GDB を実行します。PC 側で GDB 用のコマンドを実行することによりブレークやステップ実行等を行うことができます。

ここでは簡単な使い方について説明します。が、さらに高度に使いこなしたい場合は、GDB のマニュアルや解説書などを参照ください。

Algo Smart Display では、ネットワークポートを使用したデバッグ方法を標準としています。AP110 の場合は、USB-LAN アダプタをご購入していただくか、シリアルポートを使用する方法もあります。

以下より『３－３ WideStudio/MWT によるアプリケーション開発』でサンプルプログラムとして作成したプログラムを使用して、GDB にてデバッグする方法を示します。

① デバッグモードでのコンパイル

デバッグ過程がよくわかるように、Hello!と表示するボタンのプロシージャ関数のコードをリスト 3-5-1 のように変更します。

リスト 3-5-1. 表示文字列を変更するコード（デバッグ確認用）

```
#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
//---------------------------------------------------------
//Function for the event procedure
//---------------------------------------------------------
void Btn_Click(WSCbase* object){
    int i,j,k;

    k=0;
    for(i=0;i<10;i++){
        for(j=0;j<10;j++){
            k++;
        }
    }
    object->setProperty(WSNlabelString, k);          //表示文字列変更（k の値を表示）
}
static WSCfunctionRegister  op("Btn_Click",(void*)Btn_Click);
```

GDB を使用する場合、コンパイルオプションとして「-g」や「-ggdb」をつけてコンパイルする必要があります。

WideStudio の場合、  をクリックするか、メニューの「プロジェクト(P)」→「プロジェクト設定(E)」をクリックしてください。「コンパイル」タブをクリックすると、図 3-5-1 のようなプロジェクト設定画面が表示されます。

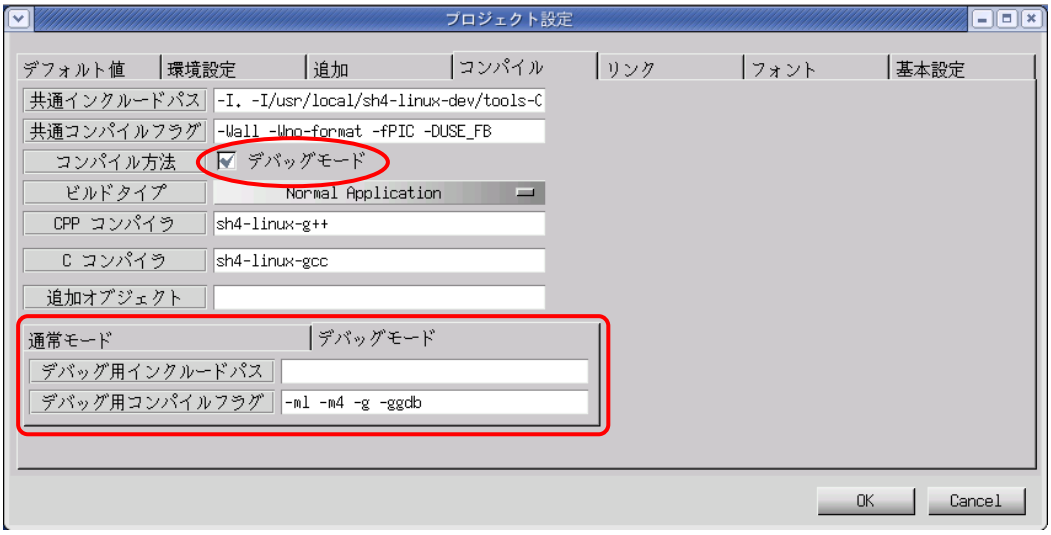

図 3-5-1. プロジェクト設定画面（コンパイルタブ）

「コンパイル方法」という項目の、「デバッグモード」というチェックボックスにチェックを入れます。これでデバッグモードのコンパイルフラグが使用されるようになります。デバッグモードのタブに「-ml -m4 -g -ggdb」というデバッグ用コンパイルフラグが設定されています。ので確認してください。

「OK」ボタンをクリックし、プロジェクト全体をコンパイルし直します。通常モードで作成される実行プログラム名に「d」が付いた実行プログラムが作成されます。(「newproject」→「newprojectd」)。

② GDB サーバの起動

Algo Smart Display で「gdbserver」を起動します。まず、『3-3-9　ファイル転送』の「LAN 経由による ftp 転送」の項目を参考に、①でコンパイルした「newprojectd」を Algo Smart Display に転送します。以下のコマンドを実行して「gdbserver」を起動します。

```
# chmod 755 newprojectd
# gdbserver host1:2345 ./newprojectd
Process ./newprojectd created; pid = 21507
Listening on port 2345
```

③ GDB の起動

 をクリックし、もう一つコンソールを立ち上げます。デバッグするプログラムのソースディレクトリへ移動します。

```
# cd /ramdisk/home/knoppix/tmp/sample
```

「sh4-linux-gdb」があるディレクトリに PATH を通します。

```
# export PATH=/usr/local/sh4-linux-dev/tools-0010/bin:$PATH
```

GDB を起動し、Algo Smart Display の GDB サーバと接続します。

```
# sh4-linux-gdb ./newprojectd
GNU gdb 6.3
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "--host=i686-pc-linux-gnu --target=sh4-linux"...
(gdb) target remote 192.168.0.1:2345
Remote debugging using 192.168.0.1:2345
0x295568c0 in ?? ()
(gdb)
```

ここまでで、GDB の起動は完了しました。

④　デバッグ

ブレークポイントを指定します。ここではボタンのプロシージャ関数でブレークするようにします。

```
(gdb) b Btn_Click(WSCbase*)
Breakpoint 1 at 0x401904: file Btn_Click.cpp, line 10.
(gdb)
```

コンティニュー実行します。Algo Smart Display 上にサンプルプログラムの画面が表示されます。

```
(gdb) c
Continuing.
```

「PUSH」ボタンをクリックすると、以下のメッセージがでてブレークされます。

```
Breakpoint 1, Btn_Click (object=0x452d08) at Btn_Click.cpp:10
10              k=0;
Current language:  auto; currently c++
(gdb)
```

監視する変数を指定します。

```
(gdb) display k
1: k = 0
(gdb) display i
2: i = 0
(gdb) display j
3: j = 0
(gdb)
```

ステップ実行します。

```
(gdb) n
11              for(i=0;i<10;i++){
3: j = 0
2: i = 0
1: k = 0
(gdb) n
12                      for(j=0;j<10;j++){
3: j = 0
2: i = 0
1: k = 0
(gdb) n
13                              k++;
3: j = 0
2: i = 0
1: k = 0
```

```
(gdb) n
12                      for(j=0;j<10;j++){
3: j = 0
2: i = 0
1: k = 1
(gdb)
```

リストを表示します。入力された数値を中心に 10 行表示されます。

```
(gdb) list 13
8               int i,j,k;
9
10              k=0;
11              for(i=0;i<10;i++){
12                      for(j=0;j<10;j++){
13                              k++;
14                      }
15              }
16              object->setProperty(WSNlabelString, k);
17      }
(gdb)
```

16 行目にブレークを張り、コンティニュー実行します。

```
(gdb) b 16
Breakpoint 2 at 0x40193a: file Btn_Click.cpp, line 16.
(gdb) c
Continuing.

Breakpoint 2, Btn_Click (object=0x452d08) at Btn_Click.cpp:16
16              object->setProperty(WSNlabelString, k);
3: j = 10
2: i = 10
1: k = 100
(gdb)
```

この時点で、ボタンの表示が変更される直前まで実行しました。さらにコンティニュー実行を行うとボタンに 100 と表示されると思います。サンプルプログラムを終了し、GDB を終了します。

```
(gdb) c
Continuing.

Program exited normally.
(gdb) q
#
```

以上で GDB によるデバッグ方法について簡単に説明しました。ここで紹介したコマンドの他にもいろいろなコマンドが用意されています。比較的よく使うと思われるコマンドについて表 3-5-1 に示します。

表 3-5-1. GDB コマンド（抜粋）

コマンド（省略）		書式	説明
実行	continue (c)	c	停止中のプログラムを再開します。
	next (n)	n	実行する行が関数の場合、関数の中へ入らずに次の行まで実行します。
	step (s)	s	実行する行が関数の場合、関数の中に入って実行します。 **※注:実行する関数が WideStudio の関数の場合はライブラリがデバッグ対応でないためステップ実行することができません。**
中断	break (b)	b <関数名> b <行番号> b <ファイル名>:<行番号>	ブレークポイントを設定します。
変数	display (disp)	disp <変数名>	監視する変数を設定します。 プログラムが停止するたびに値が表示されます。
	print (p)	p <変数名>	変数の値をモニタします。
	set	set <変数名>=<値>	変数の値を変更します。
その他	list (l)	l <行番号> l <関数名>	指定した箇所のソースを 10 行表示します。
	delete (d)	d <ブレークポイント> d <監視中の変数>	指定した番号のブレークポイントや監視中の変数を削除できます。
	info (i)	i breakpoints i local i func	デバッグ中の情報を表示します。他のサブコマンドについては「help info」を参照してください。 breakpoints：現在、張っているブレークポイントの表示 local　　：ローカル変数の表示 func　　：関数の表示
	quit (q)	q	デバッガを終了します。

３－６　シリアルコンソールについて

Algo Smart Display でシリアルコンソールを使用する方法を以下に示します。ここで記述している方法は、あくまで Knoppix 上で行う場合の方法です。

初期設定では、ALGO Smart Display のシリアルコンソールは使用しないという設定になっているので、「ASD_config」の「misc.Setting」で「Serial Terminal Enable」の項目にチェックを入れ、再起動してください。

① シリアルケーブルを ALGO Smart Display に接続します。

② PC 上で「minicom」というプログラムを実行します。コンソールで以下のコマンドを実行することで起動できます。

```
# minicom
minicom へようこそ 2.1

オプション:History Buffer, F-key Macros, Search History Buffer, I18n
コンパイルされた日時は: Mar 29 2005, 09:39:09.

CTRL-A Z を押すと、説明画面になります。
```

③ 「minicom」の通信設定を行います。「CTRL-A」を押して、「Z」キーを押してください。図 2-3-9-1 のような画面になります。

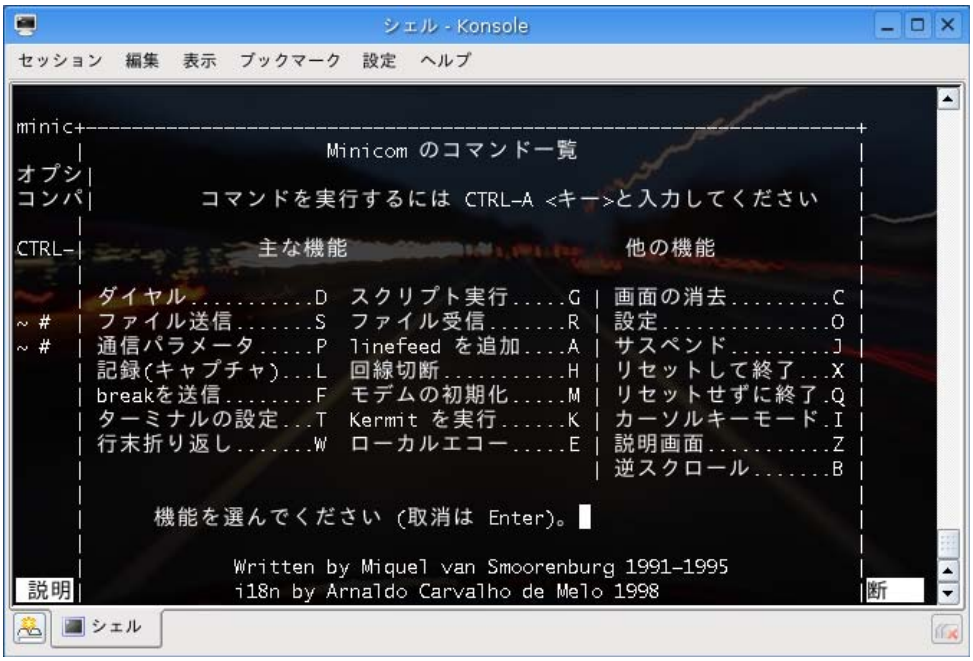

図 2-3-9-1. Minicom コマンド一覧

⑨ 「O」キーを押して、設定メニューを表示させます。

```
+---------[設定]----------+
| ファイル名とパス名      |
| ファイル転送プロトコル  |
| シリアルポート          |
| モデムとダイヤル        |
| 画面とキーボード        |
| "dfl" に設定を保存      |
| 新規に設定を保存        |
| 終了                    |
+-------------------------+
```

⑩　シリアルポートを選択してシリアルポートの設定を行います。シリアルデバイスは通常のパソコンの場合、COM1 が「/dev/ttyS0」となっています。USB 接続のシリアルデバイスを使う場合は、「/dev/ttyUSB0」となります。通信速度等は以下の通りに設定してください。設定が完了したら「Enter」キーで⑨の画面に戻ります。

```
+-----------------------------------------------------------------+
| A -    シリアルデバイス    : /dev/ttyS0                          |
| B - ロックファイルの位置   : /var/lock                           |
| C -   Callin Program       :                                    |
| D -  Callout Program       :                                    |
| E - 速度/パリティ/ビット   : 38400 8N1                          |
| F - ハードウェア流れ制御   : いいえ                             |
| G - ソフトウェア流れ制御   : いいえ                             |
|                                                                 |
|    どの設定を変更しますか?                                      |
+-----------------------------------------------------------------+
```

⑪　次に、モデムとダイヤルの設定を行います。標準設定でモデムとダイヤル設定の初期化文字列が設定されているので以下のようにすべて消去します。

```
+---------------------[モデムとダイヤルの設定]---------------------+
|                                                                 |
| A - 初期化文字列.........                                       |
| B - リセット文字列.......                                       |
| C - ダイヤル前置文字列 1..                                      |
| D - ダイヤル後置文字列 1..                                      |
| E - ダイヤル前置文字列 2..                                      |
| F - ダイヤル後置文字列 2..                                      |
| G - ダイヤル前置文字列 3..                                      |
| H - ダイヤル後置文字列 3..                                      |
| I - 接続時の文字列....... CONNECT                              |
| J - 非接続時の文字列..... NO CARRIER          BUSY             |
|                           NO DIALTONE         VOICE            |
| K - ハングアップ文字列... ~~+++~~ATH^M                         |
| L - ダイヤル中止文字列... ^M                                    |
|                                                                 |
| M - ダイヤル時間......... 45    Q - 自動 bps 検出........ いいえ |
| N - リダイヤル待ち時間... 2     R - Modem has DCD line .. はい  |
| O - リダイヤル回数....... 10    S - ステータスライン表示. DTE 速度 |
| P - DTR drop 時間 (0=no). 1     T - Multi-line untag .... いいえ |
|                                                                 |
| どの設定を変更しますか?       (Return か Esc で終了)            |
+-----------------------------------------------------------------+
```

⑫　これで、「minicom」の設定は完了です。ALGO Smart Display の電源を入れてください。以下のようなメッセージがでてきます。「root」と入力しログインします。

```
SH IPL+g version 0.9, Copyright (C) 2000 Free Software Foundation, Inc.



This software comes with ABSOLUTELY NO WARRANTY; for details type `w'.
```

```
This is free software, and you are welcome to redistribute it under

certain conditions; type `l' for details.



RAM CHECK....... OK

Loading linux.....

Jump to kernel entry point



asdap login:root

BusyBox v1.1.0 (2006.03.09-13:30+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

#
```

これでシリアルコンソールを使用することができます。

第４章　Algo　Smart　Displayについて

本章では、Algo Smart Display に実装されているデバイスの使用方法およびアプリケーション作成のテクニックについて説明しています。

DVD-ROM に格納されているサンプルソースの一覧を表 4-1 に示します。

表 4-1. サンプルソース一覧

フォルダ名	内容	参照先
sample1	AP110 パネルスイッチインプットデバイス制御方法	4-1-2
sample2	AP110 パネルスイッチキャラクタデバイス制御方法	4-1-2
sample3	AP210/310/315 汎用入出力制御方法	4-1-3
sample4	シリアルポート制御方法	4-2-1
sample5	ネットワークポート制御方法	4-2-2
sample6	オーディオデバイス制御方法	4-2-3
sample7	起動ランチャーサンプルプログラム	4-3-1
sample8	多言語表示サンプルプログラム	4-3-2
Sample9	ウォッチドッグタイマ制御方法	4-2-4
Sample10	汎用入力 IN0 リセット制御方法	4-2-5
Sample11	汎用入力 IN1 割込み制御	4-2-5
Sample12	バックアップ SRAM 制御方法 (read/write)	4-2-5
Sample13	バックアップ SRAM 制御方法 (mmap)	4-2-5

４－１　ALGO Smart Displayのデバイスについて

AP110 には 5 つのパネルスイッチが、AP210/310/315 には汎用入出力が実装されています。これらのデバイスをアクセスする方法を以下に説明します。

４－１－１　デバイスドライバアクセスについて

固有デバイスの説明の前に、一般的なデバイスドライバアクセスについて説明します。前述したとおり、デバイスのアクセスにはデバイスファイルをシステムコール（open,close,read,wirte,ioctl 等）でアクセスすることで行います。簡単に説明すると、デバイスファイルをオープンし、リード／ライトすることで制御するということになります。が、実際にはデバイス毎に動作方法を設定する必要があります。例えば、「read」というシステムコールを実行したとき、遅延を行わずにデータがある場合はそのデータをリターンし、無ければエラーをリターンする場合と、なんらかのデータが入ってくるまで遅延し、イベントが発生したらリターンする場合、イベントが発生するまでの遅延をタイムアウトで抜ける場合等があるので、どのモードで動作するかを指定する必要があります。

デバイス毎にどのような設定があるかは、インターネットや書籍で確認してください。ここでは、ALGO Smart Display 用に作成したデバイスの仕様について説明します。

４－１－２　AP110 のパネルスイッチ

AP110 に実装されている、５つのパネルスイッチは、２通りの方法でアクセスすることができます。１つは「/dev/input/event0」を使い、５つのスイッチをそれぞれキーボードの「↑」「↓」「←」「→」「Enter」に割り当て、インプットデバイスとしてアクセスする方法です。もう一つは、「/dev/pnlsw」を使って、スイッチの状態をバイトコードで読み出す、キャラクタデバイスとしてアクセスする方法です。それぞれの使い方のサンプルコードを以下に示します。

●インプットデバイスとしてアクセスする場合

この場合、Linux はキーボードがつながっていると判断して動作します。「/wsproject/sample1」にインプットデバイスとして扱う際のサンプルコードが入っています。プロジェクトを開いてみてください。図 4-1-2-1 のような画面が作成されています。

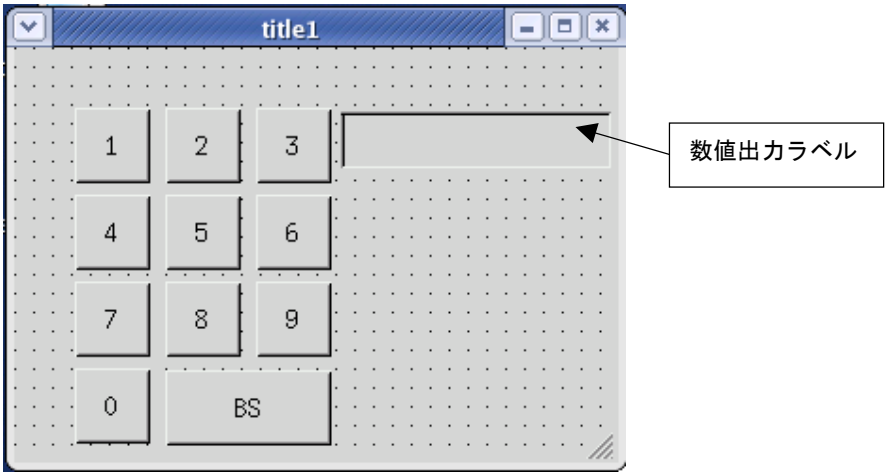

図 4-1-2-1. パネルスイッチインプットデバイスアクセスサンプル画面

ボタンのプロパティに、フォーカスの移動先を設定するプロパティがあります。(図 4-1-2-2 参照)

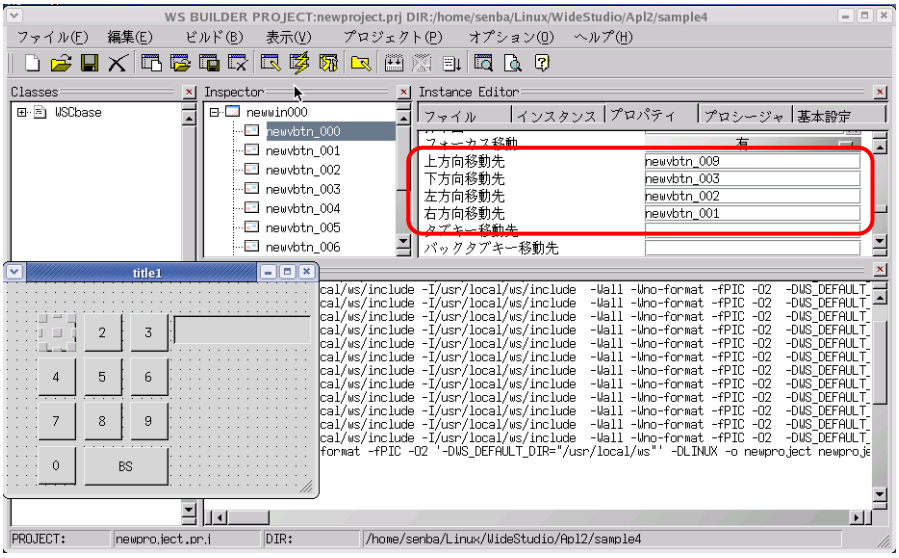

図 4-1-2-2. ボタンプロパティ設定

「上方向移動先」、「下方向移動先」、「左方向移動先」、「右方向移動先」というプロパティはそれぞれパネルスイッチの「↑」「↓」「←」「→」が押されたとき、フォーカスをどこに移すかを設定しています。

パネルスイッチの「Enter」が押されたときは、マウスのクリックと同じ意味なので、プロシージャの「ACTIVATE」イベントが実行されます。それぞれのボタンに、「ACTIVATE」のプロシージャを作成し、リスト 4-1-2-1 に書かれている関数を呼ぶようにしています。この関数は引数でもらった数値を ASCII に変換して「数値出力ラベル」に表示する関数です。10 以上の数値が引数に渡された場合は、バックスペースとして機能します。

リスト 4-1-2-1. ACTIVATE イベントで呼ばれる関数

```
#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
#include "newwin000.h"

char buf[20];
int n = 0;
void Set_Code(int dat)
{
    if(dat < 10){
        buf[n]='0'+ dat;
        buf[n+1] = 0;
        n++;
    }else{
        if(n){
            n--;
            buf[n] = 0;
        }
    }
    newvlab_013->setProperty(WSNlabelString, buf);
}
```

このプログラムを一度、PC 上で動作させてみてください。キーボードの「↑」「↓」「←」「→」でフォーカスが移動され、「Enter」で数値がラベルに出力されると思います。AP110 でこのプログラムを動作させると、パネルスイッチの入力で PC 上と同じような動作ができると思います。

●キャラクタデバイスとしてアクセスする場合

インプットデバイスとしてアクセスすると、あくまでキーボードの入力というイベントでしか使用することができません。パネルスイッチ毎にある特定の機能を持たせたい場合は、単純にボタンの入力状態をモニタしたいものです。このようなときは「/dev/pnlsw」というデバイスファイルをオープンし、リードすることで、パネルスイッチの状態を読み出すことができます。「/wsproject/sample2」に、キャラクタデバイスとしてパネルスイッチ入力を読み出した場合のサンプルコードが入っています。

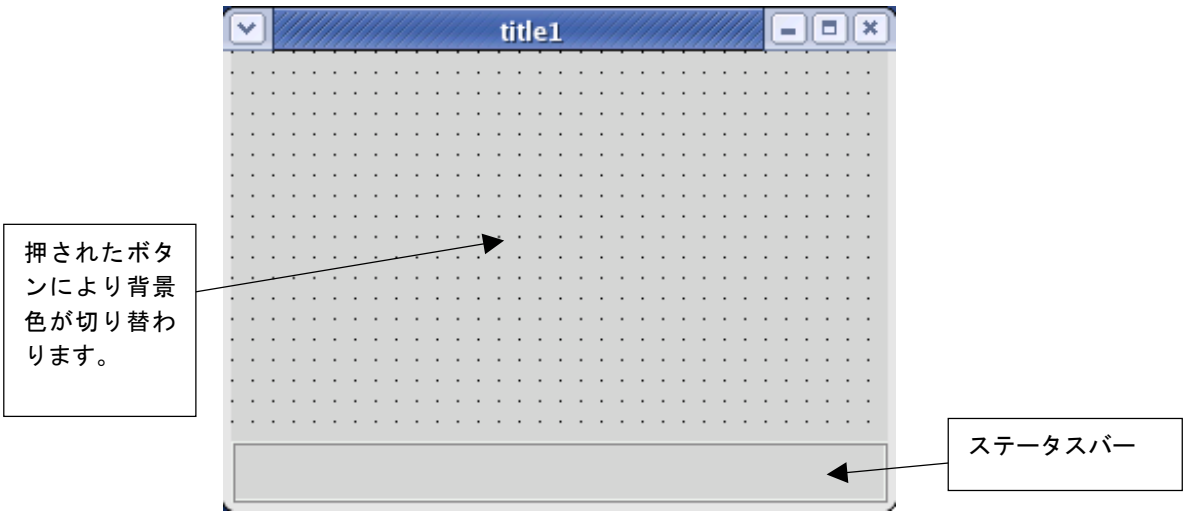

図 4-1-2-3. パネルスイッチキャラクタデバイスアクセスサンプル画面

このサンプルは、スレッド内でパネルスイッチの入力状態を監視し、押されたスイッチによって画面の色を変化させています。パネルスイッチデバイスのオープンとスレッドの生成を記したコードをリスト 3-3-2-2 に示します。

リスト 4-1-2-2. パネルスイッチのオープンとスレッドの生成

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <termios.h>
#include <pthread.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>

#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>

#include "newwin000.h"
#include "IOThread.h"
//-------------------------------------------------------
//Function for the event procedure
//-------------------------------------------------------
void Main_Init(WSCbase* object){
int mode;
int stat;

        /*ﾊﾟﾈﾙｽｲｯﾁﾃﾞﾊﾞｲｽのｵｰﾌﾟﾝ*/
    in_fd = open("/dev/pnlsw", O_RDWR);          //ﾊﾟﾈﾙｽｲｯﾁﾃﾞﾊﾞｲｽｵｰﾌﾟﾝ
    if(in_fd < 0){                               //ｴﾗｰ処理
        Status_Bar->setProperty(WSNlabelString,"Pnlsw Open Error");
        return;
    }
    mode = PNLSW_RD_ON;         //動作ﾓｰﾄﾞ変更 (ONﾄﾘｶﾞでREAD関数を抜けるように設定)
    stat = ioctl(in_fd, PNLSW_IOCSMODE, &mode); //動作ﾓｰﾄﾞ設定
    if(stat < 0){                                //ｴﾗｰ処理
        Status_Bar->setProperty(WSNlabelString,"Pnlsw ModeSet Error");
        close(in_fd);
        return;
    }
        /*ｽﾚｯﾄﾞの生成*/
    ioctrl_thr = 0;
    ioctrl_thr = WSDthread::getNewInstance();           //ｽﾚｯﾄﾞｲﾝｽﾀﾝｽ取得
    ioctrl_thr->setFunction(Ioctrl_Thread);             //ｽﾚｯﾄﾞ本体関数を設定
    ioctrl_thr->setCallbackFunction(Io_callback_func);  //ｺｰﾙﾊﾞｯｸ関数を設定
    ioctrl_thr->createThread((void*)0);                 //ｽﾚｯﾄﾞ生成
}
static WSCfunctionRegister  op("Main_Init",(void*)Main_Init);
```

まず、「open」関数で「/dev/pnlsw」をリードライトモードで開きます。つぎに、パネルスイッチの動作モードを設定します。「PNLSW_RD_ON」はパネルスイッチの ON トリガ検出したときに、「read」システムコールをリターンするモードです。パネルスイッチが押されるまで、「read」関数はリターンされません。パネルスイッチデバイスのリファレンスを表 4-1-2-1 に示します。

表 4-1-2-1. パネルスイッチデバイスリファレンス

PNLSW

名前

pnlsw－AP110のスイッチ入力5点のキャラクタデバイス

ヘッダ

```
#include "pnlsw.h"
```

説明

AP110に装備されているスイッチ入力5点の入力状態をモニタすることができます。
モニタタイプを**ioctl**関数を使用することで設定することができます。

OPEN

スイッチ入力のデバイスファイル（/dev/pnlsw）を**open**関数でオープンします。

```
fd = open("/dev/pnlsw", O_RDWR);
```

IOCTL

以下に示す動作モードを設定するには**ioctl**関数を用いてアクセスすることができます。

```
error = ioctl(fd,  ioctl_type, mode);
```

● ioctl_type

PNLSW_IOCSMODE
　　スイッチ入力のモードを設定します。
PNLSW_IOCGMODE
　　スイッチ入力のモードを取得します。

● mode

PNLSW_RD_SW
　　read関数を呼び出したとき、現在のスイッチ状態を取得して、すぐにリターンされます。
PNLSW_RD_ON
　　read関数を呼び出したとき、いずれかのスイッチのONトリガでリターンされます。ONトリガを検出するまではread関数でウエイトされます。
PNLSW_RD_OFF
　　read関数を呼び出したとき、いずれかのスイッチのOFFトリガでリターンされます。OFFトリガを検出するまではread関数でウエイトされます。
PNLSW_RD_ON|PNLSW_RD_OFF
　　read関数を呼び出したとき、いずれかのスイッチのONトリガまたはOFFトリガでリターンされます。ONトリガまたはOFFトリガを検出するまではread関数でウエイトされます。

READ

read関数を用いて、スイッチの入力状態をモニタすることができます。

```
char sw[2];
len = read(fd, &sw[0], 2);
```

AP110のスイッチデバイスをリードすると2バイトのデータがリターンされます。
1バイト目は、現状のスイッチ状態です。1でON、0でOFFです。
2バイト目は、変化があったスイッチビットのみがONされます。

bit	7	6	5	4	3	2	1	0
SW	＼	＼	＼	Ent	→	←	↓	↑

次にパネルスイッチの状態を見るために、スレッドを生成します。「WSDthread」は WideStudio で使用する汎用スレッドクラスです。リスト 4-1-2-3 にスレッド本体とコールバック関数のソースコードを示します。

リスト 4-1-2-3. パネルスイッチリードスレッドのソースコード

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <termios.h>
#include <pthread.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>

#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>

#include "newwin000.h"
#include "IOThread.h"

int in_fd;
WSDthread* ioctrl_thr;

void *Ioctrl_Thread(WSDthread* obj, void *arg)
{
    int len;
    unsigned char sw[2];
    unsigned char swd;

    for(;;){
        len = read(in_fd, &sw[0], 2);        //SW 状態の読み出し
        if(len == 2){
            swd = 0;
            if(sw[1] & 0x01){                //SW1 が押された
                swd = 1;
            }
            if(sw[1] & 0x02){                //SW2 が押された
                swd = 2;
            }
            if(sw[1] & 0x04){                //SW3 が押された
                swd = 3;
            }
            if(sw[1] & 0x08){                //SW4 が押された
                swd = 4;
            }
            if(sw[1] & 0x10){                //SW5 が押された
                swd = 5;
            }
```

```
            if(swd){
                obj->execCallback((void*)swd);  //ｺｰﾙﾊﾞｯｸ関数実行
            }
        }
    }
    return(NULL);
}
/*ｽﾚｯﾄﾞから通知され、ﾒｲﾝｽﾚｯﾄﾞで実行されるｺｰﾙﾊﾞｯｸ関数*/
void Io_callback_func(WSDthread *, void *val)
{
unsigned char swd;

    swd = (unsigned char)val;                                   //押されたｽｲｯﾁを取得
    switch(swd){
        case 1:
            newwin000->setProperty(WSNbackColor, "#FF0000");    //赤
            break;
        case 2:
            newwin000->setProperty(WSNbackColor, "#00FF00");    //緑
            break;
        case 3:
            newwin000->setProperty(WSNbackColor, "#0000FF");    //青
            break;
        case 4:
            newwin000->setProperty(WSNbackColor, "#FFFF00");    //黄
            break;
        case 5:
            newwin000->setProperty(WSNbackColor, "#FF00FF");    //ﾋﾟﾝｸ
            break;
    }
}
```

パネルスイッチデバイスを読み出すとき２バイト読み出すことができます。１バイト目にはスイッチデータの生データが入っています。現在押されているスイッチのビットが ON しています。２バイト目にはオープンした後、設定したトリガで検出したスイッチのビットのみが ON しています。（図 4-1-2-4 参照）。

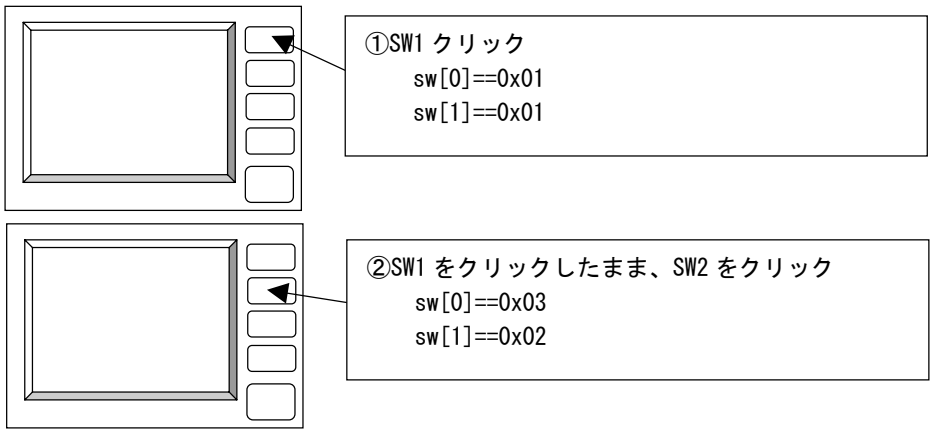

図 4-1-2-4. パネルスイッチ読み出しデータ

押された、SW 番号をコールバック関数に引き渡し、その SW 番号に対応した色をメイン画面の背景に設定しています。このプログラムを AP110 で実行すると、押されたボタンにより色が変わるのがわかると思います。

※注：このプログラムを AP110 以外で実行した場合、ステータスバーに「Pnlsw Open Error」と表示されます。これは「/dev/pnlsw」というデバイスファイルが AP110 以外に存在しないからです。

! WideStudio で用意されている「WSDthread」はスレッド本体と、コールバック関数に分かれています。これは、スレッド本体で、WideStudio のオブジェクトを操作すると、メインスレッドとの間で不整合が発生する場合がありためです。オブジェクトを操作したいときに、スレッド本体で「obj->execCallback()」を呼ぶことで、メインスレッドにコールバック関数の実行を依頼します。メインスレッドでオブジェクト操作を含むコールバック関数を実行することで、不整合の発生を防いでいます。スレッド本体で WideStudio のオブジェクト操作は行わないでください。

４－１－３　AP210/310/315 の汎用入出力

AP210/310/315 では出力 4 点、入力 6 点を使用することができます。これらの入出力は、入力用のデバイスファイルと出力用のデバイスファイルに分かれています。アクセス方法はキャラクタデバイス方式で入出力の状態を読み書きします。「/wsproject/sample3」に、汎用入出力を使ったサンプルコードが入っています。

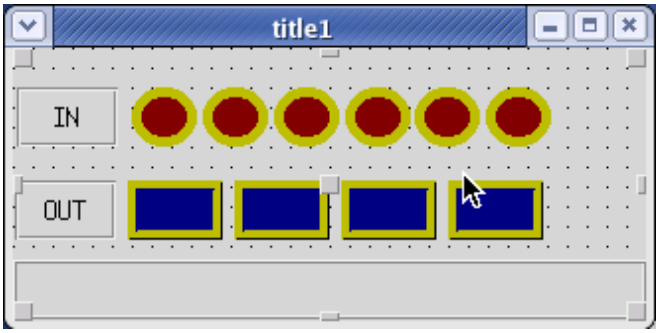

図 4-1-3-1. 汎用入出力デバイス制御サンプル画面

このサンプルは、スレッド内で汎用入力状態を監視し、入力状態によって IN の色を変化させています。また、OUT のボタンをクリックすると汎用出力が ON/OFF します。汎用入出力デバイスのオープンとスレッドの生成を記したコードをリスト 4-1-3-1 に示します。

リスト 4-1-3-1. 汎用入出力のオープンとスレッドの生成

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <termios.h>
#include <pthread.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>

#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>

#include "newwin000.h"
```

```
#include "IOThread.h"

//-------------------------------------------------------
//Function for the event procedure
//-------------------------------------------------------
void Main_Init(WSCbase* object){
unsigned char data;

        /*汎用出力ﾃﾞﾊﾞｲｽのｵｰﾌﾟﾝ*/
    out_fd = open("/dev/genout", O_RDWR);       //汎用出力ﾃﾞﾊﾞｲｽｵｰﾌﾟﾝ
    if(out_fd < 0){                             //ｴﾗｰ処理
        Status_Bar->setProperty(WSNlabelString,"OUT Device Open Error");
        return;
    }
    data = 0;
    write(out_fd, &data, 1);                    //初期０出力

        /*汎用入力ﾃﾞﾊﾞｲｽのｵｰﾌﾟﾝ*/
    in_fd = open("/dev/genin", O_RDWR);         //汎用入力ﾃﾞﾊﾞｲｽｵｰﾌﾟﾝ
    if(in_fd < 0){                              //ｴﾗｰ処理
        Status_Bar->setProperty(WSNlabelString,"IN Device Open Error");
        close(out_fd);
        return;
    }

        /*ｽﾚｯﾄﾞの生成*/
    ioctrl_thr = 0;
    ioctrl_thr = WSDthread::getNewInstance();           //ｽﾚｯﾄﾞｲﾝｽﾀﾝｽ取得
    ioctrl_thr->setFunction(Ioctrl_Thread);             //ｽﾚｯﾄﾞ本体関数を設定
    ioctrl_thr->setCallbackFunction(Io_callback_func);  //ｺｰﾙﾊﾞｯｸ関数を設定
    ioctrl_thr->createThread((void*)0);                 //ｽﾚｯﾄﾞを生成
}
static WSCfunctionRegister  op("Main_Init",(void*)Main_Init);
```

まず、「open」関数で「/dev/genout」と「/dev/genin」をリードライトモードで開きます。汎用入出力には設定すべきモードは存在しないため、これでリード／ライトすることができます。

汎用入力の状態を見るために、スレッドを生成します。リスト 4-1-3-2 にスレッド本体とコールバック関数のソースコードを示します。

リスト 4-1-3-2. 汎用入力リードスレッドのソースコード

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <termios.h>
#include <pthread.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include <math.h>
```

```
#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>

#include "newwin000.h"
#include "IOThread.h"

int in_fd;
int out_fd;
WSDthread* ioctrl_thr;
unsigned char o_data;

void *Ioctrl_Thread(WSDthread* obj, void *arg)
{
    int len;
    unsigned char data;
    o_data = data = 0;
    int i;
    i = 0;
    for(;;){
        len = read(in_fd, &data, 1);                /*汎用入力読みだし*/
        data &= 0x3F;
        if(len==1){
            if(o_data != data){
                o_data = data;
                obj->execCallback((void*)data);
            }
        }
    }
    return(NULL);
}
/*ｽﾚｯﾄﾞから通知され、ﾒｲﾝｽﾚｯﾄﾞで実行されるｺｰﾙﾊﾞｯｸ関数*/
void Io_callback_func(WSDthread *, void *val)
{
unsigned char data;

    data = (unsigned char)val;
    if(data & 0x01){                    /*入力 1*/
        newvarc_000->setPropertyV(WSNhatchColor,"#FF0000");
    }
    else{
        newvarc_000->setPropertyV(WSNhatchColor,"#800000");
    }
    if(data & 0x02){                    /*入力 2*/
        newvarc_001->setPropertyV(WSNhatchColor,"#FF0000");
    }
    else{
        newvarc_001->setPropertyV(WSNhatchColor,"#800000");
    }
    if(data & 0x04){                    /*入力 3*/
```

```
        newvarc_002->setPropertyV(WSNhatchColor,"#FF0000");
    }
    else{
        newvarc_002->setPropertyV(WSNhatchColor,"#800000");
    }
    if(data & 0x08){                    /*入力 4*/
        newvarc_003->setPropertyV(WSNhatchColor,"#FF0000");
    }
    else{
        newvarc_003->setPropertyV(WSNhatchColor,"#800000");
    }
    if(data & 0x10){                    /*入力 5*/
        newvarc_004->setPropertyV(WSNhatchColor,"#FF0000");
    }
    else{
        newvarc_004->setPropertyV(WSNhatchColor,"#800000");
    }
    if(data & 0x20){                    /*入力 6*/
        newvarc_005->setPropertyV(WSNhatchColor,"#FF0000");
    }
    else{
        newvarc_005->setPropertyV(WSNhatchColor,"#800000");
    }
}
```

汎用入力デバイスを読み出すとき、１バイト読み出すことができます。汎用入力の「read」関数は遅延せずに、汎用入力の ON/OFF 状態を即リターンします。サンプルソースでは、前回値と比較して変化があったときのみコールバック関数を実行しています。

ボタンの「ACTIVATE」プロシージャで、汎用出力の ON/OFF を行っています。リスト 4-1-3-3 に汎用出力の制御ソースコードを示します。

リスト 4-1-3-3. 汎用出力 ON/OFF のソースコード

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <termios.h>
#include <pthread.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>

#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>

#include "newwin000.h"
#include "IOThread.h"
```

```
int btn[4]={0,0,0,0};
//-------------------------------------------------------
//Function for the event procedure
//-------------------------------------------------------
void Btn_Click(WSCbase* object){
    int len;
    unsigned char data;
    long code;

    code = object->getProperty(WSNuserValue);              //押されたボタン番号を取得
    len = read(out_fd, &data, 1);                          //汎用出力の出力値取得
    if(len==1){
        if(btn[code]){
            data &= (~(0x0001 << code));                   //出力 OFF
            object->setProperty(WSNbackColor,"#000080");
            btn[code] = 0;
        }
        else{
            data |= (0x0001 << code);                      //出力 ON
            object->setProperty(WSNbackColor,"#0000FF");
            btn[code] = 1;
        }
        len = write(out_fd, &data, 1);                     //汎用出力更新
    }
}
static WSCfunctionRegister  op("Btn_Click",(void*)Btn_Click);
```

ボタンが押されたら、現在の状態から ON/OFF を切り替えます。「read」関数で現在の状態を読み込み、新しい状態に変更し、「write」関数でデータを更新します。

表 4-1-3-1 に汎用入出力のリファレンスを示します。

表 4-1-3-1. 汎用入出力デバイスリファレンス

GENIN,GENOUT

名前

genin－AP210,AP310に実装されている汎用入力6点
genout－AP210,AP310に実装されている汎用出力4点

説明

AP210、AP310に装備されている汎用入力6点の状態読み出しと、汎用出力4点の制御を行います。

OPEN

汎用入出力デバイス(/dev/genin,/dev/genout)を**open**関数でオープンします。

```
infd = open("/dev/genin", O_RDWR);
outfd = open("/dev/genout", O_RDWR);
```

READ

read関数を用いて汎用入出力の状態をモニタすることができます。

```
char in;
char out;
len = read(infd, &in, 1);
len = read(outfd, &out, 1);
```

汎用入出力デバイスをリードすると1バイトのデータが即リターンされます。
リターンされた値は現在の入出力状態です。

bit	7	6	5	4	3	2	1	0
IN			IN6	IN5	IN4	IN3	IN2	IN1

bit	7	6	5	4	3	2	1	0
OUT					OT4	OT3	OT2	OT1

WRITE

write関数を用いて汎用出力を制御することができます。

```
char out;
out = 1;
len = write(outfd, &out, 1);
```

汎用出力データを1バイトライトすることで対応するビットの出力をON/OFFすることができます。

bit	7	6	5	4	3	2	1	0
OUT					OT4	OT3	OT2	OT1

４－２　その他のデバイスについて

ALGO Smart Display に実装されているシリアルポートとネットワークポートの使用例について記述します。これらのポートは、一般的な Linux の標準デバイスに準拠しています。詳細な使用方法はインターネットや書籍等を参照してください。

４－２－１　シリアルポート

Algo Smart Display でケース外にでているシリアルポートは「/dev/ttySC2」となっています。アプリケーションでシリアルポートを使用するには、「/dev/ttySC2」をオープンし、リード／ライトすることで制御します。

「/wsproject/sample4」に、シリアルポートを使用したサンプルコードが入っています。リスト 4-2-1-1 にシリアルポートのオープンと通信設定を行うソースコードを示します。

リスト 4-2-1-1. シリアルポートのオープンと通信設定

```
#include <termios.h>
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/signal.h>
#include <sys/types.h>

#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>

#include "newwin000.h"
#include "ComThread.h"
//---------------------------------------------------------
//Function for the event procedure
//---------------------------------------------------------
void Main_Init(WSCbase* object){
    struct termios tio;
    int stat;

        /*ｼﾘｱﾙﾎﾟｰﾄｵｰﾌﾟﾝ*/
    comm_fd = open("/dev/ttySC2", O_RDWR | O_NOCTTY);          //ｼﾘｱﾙﾎﾟｰﾄｵｰﾌﾟﾝ
    if(comm_fd < 0){
        Status_Bar->setPropertyV(WSNlabelString,"ttySC2 Open Error");
        return;
    }

    stat = tcgetattr(comm_fd, &tio);                           //現在の通信設定を待避
    if(stat < 0){
        Status_Bar->setPropertyV(WSNlabelString,"Terminal Attribute Get Error");
        close(comm_fd);
        return;
    }
    //通信設定　(ﾃﾞｰﾀ長 8bit ｽﾄｯﾌﾟﾋﾞｯﾄ 1bit ﾊﾟﾘﾃｨ無し 制御線無視)
    tio.c_cflag &= ~(CSIZE | CSTOPB | PARENB | PARODD | HUPCL);
    tio.c_cflag |= CS8 | CLOCAL | CREAD;
    //通信設定　(ﾌﾚｰﾑｴﾗｰ、ﾊﾟﾘﾃｨｴﾗｰなし)
```

```
    tio.c_iflag = IGNPAR;
    tio.c_oflag = 0;
    tio.c_lflag = 0;
    tio.c_cc[VINTR] = 0;
    tio.c_cc[VQUIT] = 0;
    tio.c_cc[VERASE] = 0;
    tio.c_cc[VKILL] = 0;
    tio.c_cc[VEOF] = 0;
    tio.c_cc[VTIME] = 50;                //ｷｬﾗｸﾀ間ﾀｲﾑｱｳﾄ時間 5Oms
    tio.c_cc[VMIN] = 1;                  //1 文字取得するまでﾌﾞﾛｯｸ
    tio.c_cc[VSWTC] = 0;
    tio.c_cc[VSTART] = 0;
    tio.c_cc[VSTOP] = 0;
    tio.c_cc[VSUSP] = 0;
    tio.c_cc[VEOL] = 0;
    tio.c_cc[VREPRINT] = 0;
    tio.c_cc[VDISCARD] = 0;
    tio.c_cc[VWERASE] = 0;
    tio.c_cc[VLNEXT] = 0;
    tio.c_cc[VEOL2] = 0;

    //通信設定　(ﾎﾞｰﾚｰﾄ38400)
    cfsetospeed(&tio, B38400);
    cfsetispeed(&tio, B38400);
    stat=tcsetattr(comm_fd, TCSAFLUSH, &tio);          //変更した通信設定の反映
    if(stat < 0){
        Status_Bar->setPropertyV(WSNlabelString,"Terminal Attribute Set Error");
        close(comm_fd);
        return;
    }

        /*ｽﾚｯﾄﾞの生成*/
    comctrl_thr = 0;
    comctrl_thr = WSDthread::getNewInstance();          //ｽﾚｯﾄﾞｲﾝｽﾀﾝｽ取得
    comctrl_thr->setFunction(Comctrl_Thread);           //ｽﾚｯﾄﾞ本体関数を設定
    comctrl_thr->setCallbackFunction(Com_callback_func);  //ｺｰﾙﾊﾞｯｸ関数を設定
    comctrl_thr->createThread((void*)0);                //ｽﾚｯﾄﾞを生成
}
static WSCfunctionRegister  op("Main_Init",(void*)Main_Init);
```

「open」関数で「/dev/ttySC2」をオープンします。「O_NOCTTY」は制御端末として使用しないモードでオープンします。「tcgetattr」関数で現状の通信設定（「termios」構造体）を取得します。「termios」構造体のメンバの値を変更することで通信設定を変更します。「tcsetattr」関数で変更した通信設定を反映します。これで通信できる状態になります。「termios」構造体の詳細については、インターネットや書籍を参照してください。

リスト 4-2-1-2 にシリアル通信スレッドのソースコードを示します。

リスト 4-2-1-2. シリアル通信スレッド

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
```

```
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <termios.h>
#include <pthread.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include <math.h>

#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>

#include "newwin000.h"
#include "ComThread.h"

int comm_fd;
WSDthread* comctrl_thr;

void *Comctrl_Thread(WSDthread* obj, void *arg)
{
    int len;
    char data;
    int i;
    i = 0;
    for(;;){
       len = read(comm_fd, &data, 1);              /*1ﾊﾞｲﾄ受信*/
        if(len==1){
           write(comm_fd, &data, 1);               /*1ﾊﾞｲﾄ送信*/
           obj->execCallback((void*)data);
        }
    }
    return(NULL);
}
/*ｽﾚｯﾄﾞから通知され、ﾒｲﾝｽﾚｯﾄﾞで実行されるｺｰﾙﾊﾞｯｸ関数*/
void Com_callback_func(WSDthread *, void *val)
{
char data[2];

    data[0] = (char)val;
    data[1] = 0;
    newtext_001->addString(data);                  /*文字列追加表示*/
}
```

「read」関数で１バイト受信するまで待ちます。１バイト受信したら、受信した文字を「write」関数で送信します。また、同時にアプリケーションのテキストフィールドに受信した文字を表示します。

４－２－２　ネットワークポート

いままでのデバイスファイルでは「open」関数でデバイスをオープンし、制御してきました。ネットワーク通信ではソケットと呼ばれる概念で通信します。ソケットには接続を待つサーバと、サーバに接続にいくクライアントがあります。サーバプログラムがまず起動され、接続を待ちます。次にクライアントプログラムを起動してサーバに接続にいきます。これでネットワーク通信が確立します。

WideStudio にはネットワーク用のオブジェクトがあります。が、ここではネットワーク用のシステムコールを使用したサンプルプログラムについて説明します。「/wsproject/sample5」に、ネットワークポートを使用したサンプルコードが入っています。

●サーバ側のサンプルプログラム

サーバ側のプログラムは、表 4-2-2-1 に書かれているシステムコールを実行して、接続します。

表 4-2-2-1. サーバ側ソケットシステムコール

関数名	説明
socket	ソケットを作成し、対応するファイルディスクリプタを返します。
bind	待ちポート番号を指定します。
listen	カーネルにサーバソケットであることを伝えます。
accept	クライアントが接続してくるまで待ちます。通信が確立したら、接続済みのファイルディスクリプタを返します。

リスト 4-2-2-1 にサーバソケット作成を行うソースコードを示します。

リスト 4-2-2-1. サーバソケット生成

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <bits/local_lim.h>

#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>

#include "newwin000.h"
#include "SrvThread.h"

//--------------------------------------------------------
//Function for the event procedure
//--------------------------------------------------------
void Main_Init(WSCbase* object){
        /* ｿｹｯﾄ生成 */
    if ((srv_sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0) {
        Status_Bar->setProperty(WSNlabelString,"socket() failed");
```

```
        return;
    }

        /* ﾎﾟｰﾄ番号指定 */
    memset(&srv_addr, 0, sizeof(srv_addr));
    srv_addr.sin_family      = AF_INET;
    srv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    srv_addr.sin_port        = htons(8900);                  //ﾎﾟｰﾄ番号指定
    if (bind(srv_sock, (struct sockaddr *) &srv_addr, sizeof(srv_addr)) < 0) {
        Status_Bar->setProperty(WSNlabelString,"bind() failed");
        close(srv_sock);
        return;
    }
        /* ｶｰﾈﾙ通知 */
    if (listen(srv_sock, 1) < 0) {
        Status_Bar->setProperty(WSNlabelString,"listen() failed");
        close(srv_sock);
        return;
    }
        /*ｽﾚｯﾄﾞの生成*/
    srvctrl_thr = 0;
    srvctrl_thr = WSDthread::getNewInstance();              //ｽﾚｯﾄﾞｲﾝｽﾀﾝｽ取得
    srvctrl_thr->setFunction(Srvctrl_Thread);               //ｽﾚｯﾄﾞ本体関数を設定
    srvctrl_thr->setCallbackFunction(Srv_callback_func);    //ｺｰﾙﾊﾞｯｸ関数を設定
    srvctrl_thr->createThread((void*)0);                    //ｽﾚｯﾄﾞを生成
}
static WSCfunctionRegister  op("Main_Init",(void*)Main_Init);
```

接続待ちを行う、サーバソケットを作成し、実際に待ちを行うためのスレッドを作成します。各関数の引数の意味については、インターネットや書籍を参照してください。

リスト 4-2-2-2 に接続待ちとクライアントからのデータ受信待ちを行うスレッドのソースコードを示します。

リスト 4-2-2-2. 接続待ちおよびデータ受信待ちスレッド

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <bits/local_lim.h>

#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>

#include "newwin000.h"
#include "SrvThread.h"
```

```
char  tmp[BUFFER];
int srv_sock;
int cli_sock;
struct sockaddr_in srv_addr;
struct sockaddr_in cli_addr;
WSDthread* srvctrl_thr;

void *Srvctrl_Thread(WSDthread* obj, void *arg)
{
    int len;

    for(;;){
        /* ｸﾗｲｱﾝﾄ接続待ち */
        len = sizeof(cli_addr);
        if ((cli_sock = accept(srv_sock, (struct sockaddr *)&cli_addr, (socklen_t *)&len)) < 0) {
            continue;
        }

        for(;;){
            /* ﾃﾞｰﾀ受信待ち */
            len = recv(cli_sock, tmp, BUFFER, 0);
            if (len > 0){
                obj->execCallback((void *)len); //正常受信完了
            }else{
                close(cli_sock);
                break;                          //通信断
            }
        }

    }
    return(NULL);
}
/*ｽﾚｯﾄﾞから通知され、ﾒｲﾝｽﾚｯﾄﾞで実行されるｺｰﾙﾊﾞｯｸ関数*/
void Srv_callback_func(WSDthread *, void *val)
{
int len;

    len = (int)val;
    tmp[len] = 0;
    newvlab_000->setProperty(WSNlabelString, tmp);
}
```

「accept」関数でクライアントプログラムが接続してくるのを待ちます。正常に通信確立すれば、通信確立済みのファイルディスクリプタがリターンされます。通信確立済みのファイルディスクリプタを使用してデータの送受信を行います。送信は「send」関数を、受信は「recv」関数を使用します。

このプログラムでは、受信した文字列を画面に表示します。

●クライアント側のサンプルプログラム

クライアント側のプログラムは、表 3-4-2-2 に書かれているシステムコールを実行して、接続待ちしているサーバに接続します。

表 4-2-2-2. クライアント側ソケットシステムコール

関数名	説明
socket	ソケットを作成し、対応するファイルディスクリプタを返します。
connect	指定された IP とポート番号のサーバに接続にいきます。

リスト 4-2-2-3 にクライアントソケット作成を行うソースコードを示します。

リスト 4-2-2-3. クライアントソケット生成

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <bits/local_lim.h>

#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
#include <WSDappDev.h>

#include "newwin000.h"

int sock;
struct sockaddr_in srv_addr;
//-------------------------------------------------------
//Function for the event procedure
//-------------------------------------------------------
void Main_Init(WSCbase* object){
    char **argv;
    char *srv_ip;

    /* ﾌﾟﾛｸﾞﾗﾑ起動時の引数でｻｰﾊﾞの IP ｱﾄﾞﾚｽを取得 */
    if(WSGIappDev()->getArgc() < 2){
        Status_Bar->setProperty(WSNlabelString,"No IP Address");
        return;
    }
    argv = WSGIappDev()->getArgv();
    srv_ip = *(argv + 1);
```

```
    /* ｸﾗｲｱﾝﾄｿｹｯﾄ作成*/
    if ((sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0) {
        Status_Bar->setProperty(WSNlabelString,"socket() failed");
        return;
    }

    /* ｻｰﾊﾞに接続 */
    memset(&srv_addr, 0, sizeof(srv_addr));
    srv_addr.sin_family      = AF_INET;
    srv_addr.sin_addr.s_addr = inet_addr(srv_ip);       //ﾀｰｹﾞｯﾄｻｰﾊﾞ IP ｱﾄﾞﾚｽ
    srv_addr.sin_port        = htons(8900);             //ﾀｰｹﾞｯﾄｻｰﾊﾞﾎﾟｰﾄ番号
    if (connect(sock, (struct sockaddr *) &srv_addr, sizeof(srv_addr)) < 0) {
        Status_Bar->setProperty(WSNlabelString,"connect failed");
        close(sock);
        return;
    }

    /* list 初期化 */
    newlist_000->delAll();
    newlist_000->addItem("ABCDEFG...");
    newlist_000->addItem("1234567...");
    newlist_000->addItem("abcdefg...");
    newlist_000->updateList();

}
static WSCfunctionRegister  op("Main_Init",(void*)Main_Init);
```

「socket」関数でソケットを生成します。プログラム起動時に引数として、サーバの IP アドレスを指定します。指定した IP アドレスとサーバプログラムで指定したポート番号に「connect」関数で接続します。通信確立したら 0 が、エラーなら-1 がリターンされます。これで、通信できる状態です。

ボタンの「ACTIVATE」プロシージャで、リストから選んだ文字列を送信します。リスト 4-2-2-4 に文字列送信のソースコードを示します。

リスト 4-2-2-4. クライアントプログラムボタン「ACTIVATE」プロシージャ

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <bits/local_lim.h>

#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>

#include "newwin000.h"
```

```
#define  BUFFER  2048

extern int sock;
extern struct sockaddr_in srv_addr;
const char dat[][21]={
    {"ABCDEFGHIJKLMNOPQRST"},
    {"12345678901234567890"},
    {"abcdefghijklmnopqrst"}
};
//-------------------------------------------------------
//Function for the event procedure
//-------------------------------------------------------
void Btn_Click(WSCbase* object){
long d;

    d = newlist_000->getSelectedPos();
    if((d < 0)||(d > 2)) return;
    /* ﾃﾞｰﾀ送信 */
    if (send(sock, &dat[d][0], 20, 0) != 20) {
        Status_Bar->setProperty(WSNlabelString,"send() failed");
    }
}
static WSCfunctionRegister  op("Btn_Click",(void*)Btn_Click);
```

リストで送信する文字列を選択し、ボタンをクリックします。「send」関数で該当する文字列を送信します。

サーバプログラムとクライアントプログラムの起動手順は以下の通りです。

```
# ./server_spl &
# ./client_spl 127.0.0.1 &
```

この場合、１個の ALGO Smart Display 上でサーバとクライアントのプログラムが動作し、お互いに通信を行います。

４－２－３　オーディオ出力

AP210/310/315 に実装されているオーディオデバイスの制御方法について以下に示します。
オーディオデバイスは「/dev/dsp0」というデバイスファイルに音声ファイルのデータを書き込むことで再生させることができます。

「/wsproject/sample6」に、オーディオデバイスを使用したサンプルコードが入っています。リスト 4-2-3-1 にオーディオデバイスのオープンから音声の再生まで記述したソースコードを示します。

リスト 4-2-3-1. オーディオ再生

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include <sys/ioctl.h>
#include <sys/soundcard.h>

#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>

#define  BUFFER  1024
static unsigned char sound_data[BUFFER];

//-------------------------------------------------------
//Function for the event procedure
//-------------------------------------------------------
void btnPlayOnClick(WSCbase* object)
{
    int dsp;
    int fd;
    int len;
    int format;

    // サウンドデバイスの open
    if ((dsp = open("/dev/dsp0", O_WRONLY)) < 0) {
        fprintf(stderr, "sound device open() failed¥n");  /*ｵｰﾃﾞｨｵﾃﾞﾊﾞｲｽｵｰﾌﾟﾝｴﾗｰ*/
        return;
    }

    // 音声データフォーマット指定
    // 16bit Little Endian, 16kHz, Stereo
    format = AFMT_S16_LE;
    ioctl(dsp, SNDCTL_DSP_SETFMT, &format);     /*16bit Little Endian*/
    format = 1;
    ioctl(dsp, SNDCTL_DSP_STEREO, &format);     /*ｽﾃﾚｵ*/
    format = 16000;
    ioctl(dsp, SNDCTL_DSP_SPEED, &format);      /*16KHz*/
```

```
    // 音声データファイルの open
    if ((fd = open("sound.wav", O_RDONLY)) < 0) {
        fprintf(stderr, "sound data open() failed¥n");   /*ﾌｧｲﾙｵｰﾌﾟﾝｴﾗｰ*/
        close(dsp);
        return;
    }
    l seek(fd,44.SEEK_SET);
    // 音声データ読込みと再生
    while((len = read(fd, sound_data, BUFFER)) > 0) {
        write(dsp, sound_data, len);
    }

    close(fd);
    close(dsp);
}
static WSCfunctionRegister  op("btnPlayOnClick",(void*)btnPlayOnClick);
```

このサンプルプログラムでは、ボタンをクリックすると、「open」関数で「/dev/dsp0」をオープンし、音声データのフォーマットを「ioctl」関数で指定します。再生する音声ファイルをオープンして、読み出したデータをデバイスファイルに「write」することでオーディオ出力端子から音声が出力されます。音声ファイルの EOF を検出したらデバイスファイルと音声ファイルをクローズして終了です。

４－２－４　ウォッチドッグタイマ

AP210/310/315 に実装されているウォッチドッグタイマデバイスの制御方法について以下に示します。

ウォッチドッグタイマデバイスを利用するには「/dev/watchdog」というデバイスファイルを操作します。タイムアウトは 100～25500msec の範囲で設定することが出来ます。

「/wsproject/sample9」にウォッチドッグタイマデバイスを使用したサンプルコードが入っています。リスト 4-2-4-1 にウォッチドッグタイマデバイスのオープン、タイマクリア処理を記述したソースコードを示します。

リスト 4-2-4-1. ウォッチドッグタイマ

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <linux/watchdog.h>

int main(void)
{
    int fd;
    int ret;
    int timeout;
    int c;

    /*
     * watchdog デバイスのオープン
     */
    fd = open("/dev/watchdog", O_WRONLY);
    if (fd == -1) {
        printf("/dev/watchdog open faild¥n");
        return -1;
    }

    /*
     * タイムアウトの設定 (100～25500msec)
     */
    timeout = 5000;
    ret = ioctl(fd, WDIOC_SETTIMEOUT, &timeout);
    if (ret < 0) {
        printf("ioctl WDIOC_SETTIMEOUT faild: err=%d¥n", ret);
    }

    /*
     * ウォッチドッグクリア処理
     */
    while(1){
        c = fgetc(stdin);
        if(c == 'e' || c == 'E'){   /* WDT 無効で終了 */
            write(fd, "V", 1);
            break;
        }
        if(c == 'q' || c == 'Q'){   /* WDT 有効のまま終了 */
            break;
```

```
        }


        write(fd, "¥0", 1);          /* クリア */
        sleep(1);                    /* 1sec */
    }

    close(fd);
    return 0;
}
```

このサンプルプログラムはコンソールアプリケーションとして作成されています。シリアルターミナル、TELNET などで動作させることが出来ます。「e」、[E]で終了した場合はウォッチドッグを無効にして終了します。「q」、「Q」で終了した場合、または強制終了などの場合はウォッチドッグを有効にした状態で終了します。ウォッチドッグを有効にした状態で終了した場合は、ウォッチドッグタイマのタイムアウトと共にハードウェアリセットが入ります。

４－２－５　RAS機能

AP315 には RAS 機能として以下の様な機能を実装してます。

- 汎用入力 IN0 リセット
- 汎用入力 IN1 割込み
- バックアップ SRAM

これらの RAS 機能デバイスの制御方法について説明します。

● 汎用入力 IN0 リセット

汎用入力 IN0 リセットは、汎用入力の IN0 が ON した場合に本体をリセットする機能です。デバイスドライバからこの機能の有効・無効を切り替えることができます。

「/wsproject/sample10」に汎用入力 IN0 リセット機能を操作したサンプルコードが入っています。このサンプルプログラムはコンソールアプリケーションとして作成されています。シリアルターミナル、TELNET などで動作させることが出来ます。リスト 4-2-5-1 にソースコードを示します。

リスト 4-2-5-1. 汎用入力 IN0 リセット

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include "rasin.h"

int main(void)
{
    int desc;
    int len;
    int onoff;

    /*
     * RAS/Input デバイスのオープン
     */
    desc = open("/dev/rasin", O_RDWR);
    if(desc < 0){
        printf("open faild: err=%d¥n", errno);
        return -1;
    }

    /*
     * 現在の設定を取得
     *
     * onoff: 1    有効
     *      : 0    無効
     */
    len = ioctl(desc, RASIN_IOCGINORST, &onoff);
    if (len < 0) {
        printf("ioctl get INORST faild: err=%d¥n", len);
    }
    else {
```

```
        printf("ioctl get INORST: %d¥n", onoff);
    }

    /*
     * IN0 リセットを有効にする
     *
     * onoff: 1    有効
     *      : 0    無効
     */
    onoff = 1;
    len = ioctl(desc, RASIN_IOCSINORST, &onoff);
    if (len < 0) {
        printf("ioctl set INORST faild: err=%d¥n", len);
    }
    else {
        printf("ioctl set INORST: %d¥n", onoff);
    }

    close(desc);
    return 0;
}
```

● 汎用入力 IN1 割込み

汎用入力 IN1 割込みは、汎用入力の IN1 が ON した場合に割込みを発生させる機能です。デバイスドライバからこの機能の有効・無効を切り替えることができます。ユーザーアプリケーションでは SIGIO シグナルとして通知を受けることができます。

「/wsproject/sample11」に汎用入力 IN1 リセット機能を操作したサンプルコードが入っています。このサンプルプログラムはコンソールアプリケーションとして作成されています。シリアルターミナル、TELNET などで動作させることが出来ます。リスト 4-2-5-2 にソースコードを示します。

リスト 4-2-5-2. 汎用入力 IN1 割込み

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include <signal.h>
#include "rasin.h"

void sighandler(int signo)
{
    /*
     * SIGIO シグナルなら終了
     */
    if(signo == SIGIO){
```

```
        printf("IN1 INTERRUPT¥n");
        exit(0);
    }
}

int main(void)
{
    int desc;
    int len;
    int onoff;
    struct sigaction action;

    /*
     * SIGIO シグナルのハンドラを登録
     */
    memset(&action, 0, sizeof(action));
    action.sa_handler = sighandler;
    action.sa_flags = 0;
    sigaction(SIGIO, &action, NULL);

    /*
     * RAS/Input デバイスのオープン
     */
    desc = open("/dev/rasin", O_RDWR);
    if(desc < 0){
        printf("open faild: err=%d¥n", errno);
        return -1;
    }

    /*
     * プロセスが SIGIO を受け取れるようにする
     */
    fcntl(desc, F_SETOWN, getpid());
    fcntl(desc, F_SETFL, fcntl(desc, F_GETFL) | FASYNC);

    /*
     * 現在の設定を取得
     *
     * onoff: 1    有効
     *      : 0    無効
     */
    len = ioctl(desc, RASIN_IOCGIN1INT, &onoff);
    if (len < 0) {
        printf("ioctl get IN1INT faild: err=%d¥n", len);
    }
    else {
        printf("ioctl get IN1INT: %d¥n", onoff);
    }

    /*
     * IN1 割り込みを有効にする
```

```
     *
     * onoff: 1    有効
     *      : 0    無効
     */
    onoff = 1;
    len = ioctl(desc, RASIN_IOCSIN1INT, &onoff);
    if (len < 0) {
        printf("ioctl set IN1INT faild: err=%d¥n", len);
    }
    else {
        printf("ioctl set IN1INT: %d¥n", onoff);
    }

    while(1){
        sleep(1);
    }
    close(desc);
    return 0;
}
```

● バックアップ SRAM

バックアップ SRAM は、バックアップバッテリー付きの SRAM です。デバイスドライバから SRAM のデータを読み書きできます。

「/wsproject/sample12」にバックアップ付き SRAM を read/write システムコールで操作したサンプルコードが入っています。このサンプルプログラムはコンソールアプリケーションとして作成されています。シリアルターミナル、TELNET などで動作させることが出来ます。リスト 4-2-5-3 ソースコードを示します。

リスト 4-2-5-3. バックアップ付き SRAM (read/write)

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>

#define RASRAM_SIZE 0x80000
unsigned char rasram[RASRAM_SIZE];

int main(void)
{
    int desc;
    int i;
    int len;
    unsigned char d;

    /*
     * RAS/SRAM デバイスのオープン
     */
```

```
    desc = open("/dev/rasram", O_RDWR);
    if(desc < 0){
        printf("open faild: err=%d¥n", errno);
        return -1;
    }

    /*
     * 書込みデータの作成
     */
    for(i = 0, d = 1; i < RASRAM_SIZE; i++, d++){
        rasram[i] = d;
    }

    /*
     * データの書込み
     */
    lseek(desc, 0, SEEK_SET);
    len = write(desc, &rasram[0], RASRAM_SIZE);
    if(len != RASRAM_SIZE){
        printf("data write faild: err=%d¥n", errno);
        close(desc);
        return -1;
    }

    /*
     * データの読込み
     */
    memset(&rasram[0], 0x00, RASRAM_SIZE);
    lseek(desc, 0, SEEK_SET);
    len = read(desc, &rasram[0], RASRAM_SIZE);
    if(len != RASRAM_SIZE){
        printf("data read faild: err=%d¥n", errno);
        close(desc);
        return -1;
    }

    /*
     * データのチェック
     */
    for(i = 0, d = 1; i < RASRAM_SIZE; i++, d++){
        if(rasram[i] != d){
            printf("data compare faild¥n");
            close(desc);
            return -1;
        }
    }

    close(desc);
    printf("read/write check ok¥n");
    return 0;
}
```

「/wsproject/sample13 にバックアップ付き SRAM を mmap システムコールで操作したサンプルコードが入っています。このサンプルプログラムはコンソールアプリケーションとして作成されています。シリアルターミナル、TELNET などで動作させることが出来ます。リスト 4-2-5-4 ソースコードを示します。

リスト 4-2-5-4. バックアップ付き SRAM (mmap)

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>

#define RASRAM_SIZE 0x80000

int main(void)
{
    int desc;
    int i;
    void *memmap = NULL;

    unsigned char d;
    volatile unsigned char *m;

    /*
     * RAS/SRAM デバイスのオープン
     */
    desc = open("/dev/rasram", O_RDWR);
    if(desc < 0){
        printf("open faild: err=%d¥n", errno);
        return -1;
    }

    /*
     * RAS/SRAM デバイスをメモリにマッピング
     */
    memmap = mmap(0, RASRAM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, desc, 0);
    if (!memmap) {
        printf("mmap faild¥n");
                close(desc);
        return -1;
    }
    printf("mmap: %08x¥n", (unsigned long)memmap);

    /*
     * データ書き込み
     */
    d = 1;
    m = (volatile unsigned char *)memmap;
```

```
    for(i = 0; i < RASRAM_SIZE; i++){
        *m++ = d++;
    }

    /*
     * データ読込みとチェック
     */
    d = 1;
    m = (volatile unsigned char *)memmap;
    for(i = 0; i < RASRAM_SIZE; i++){
        if(*m++ != d++){
            printf("byte check1 compare faild¥n");
            close(desc);
            return -1;
        }
    }

    close(desc);
    printf("mmap check ok¥n");
    return 0;
}
```

４－３　サンプルプログラム

これまでは、デバイスドライバを使用したサンプルプログラムの説明をしてきました。ここでは、その他のサンプルプログラムについて説明します。

４－３－１　起動ランチャー

「/wsproject/sample7」に、別のアプリケーションを起動するランチャープログラムのサンプルソースが入っています。

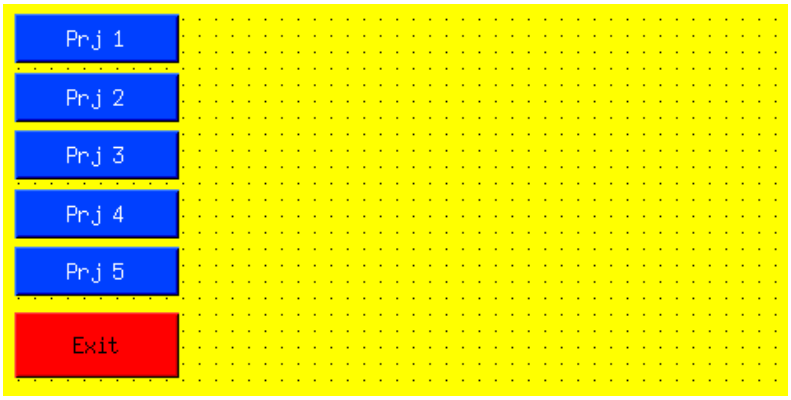

図 4-3-1-1. ランチャーサンプルプログラムメイン画面

このサンプルソースでは、ウインドウマネージャを使用しない設定にしたため、メインウインドウのプロパティを表 4-3-1-1 の用に変更します。

表 4-3-1-1. メインウィンドウのプロパティ変更

プロパティ名	説明	設定値
タイトル属性	ウィンドウのタスクバー属性の設定	WM 管理外
終了	ウィンドウを非表示にしたとき終了するかしないかの設定	オフ

各ボタンのプロパティの「ユーザ設定値」の項目を各ボタンでユニークな番号を設定します。これで、ボタンクリックしたときのプロシージャイベント関数を同じにすることができます。リスト 4-3-1-1 にボタンクリックプロシージャ関数のソースコードを示します。

リスト 4-3-1-1. ランチャーボタンソースコード

```
#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
#include "newwin000.h"
//-------------------------------------------------------
//Function for the event procedure
//-------------------------------------------------------
void Btn_Click(WSCbase* object){
long btn;

    btn = object->getProperty(WSNuserValue);     /* ｸﾘｯｸされたﾎﾞﾀﾝを特定する */
    switch(btn){
        case 1:
            newwin000->setVisible(False);        /* ﾒｲﾝｳｲﾝﾄﾞｳを非表示 */
            system("xeyes");                     /* xeyes ﾌﾟﾛｸﾞﾗﾑを起動 */
            newwin000->setVisible(True);         /* ﾒｲﾝｳｲﾝﾄﾞｳを表示 */
```

```
            break;
        case 2:
            newwin000->setVisible(False);
            system("xclock");                   /* xclock ﾌﾟﾛｸﾞﾗﾑを起動 */
            newwin000->setVisible(True);
            break;
        case 3:
            newwin000->setVisible(False);
            system("xcalc");                    /* xcalc ﾌﾟﾛｸﾞﾗﾑを起動 */
            newwin000->setVisible(True);
            break;
        case 4:
            newwin000->setVisible(False);
            system("xlogo");                    /* xlogo ﾌﾟﾛｸﾞﾗﾑを起動 */
            newwin000->setVisible(True);
            break;
        case 5:
            newwin000->setVisible(False);
            system("asd_config");               /* asd_config ﾌﾟﾛｸﾞﾗﾑを起動 */
            newwin000->setVisible(True);
            break;
        case 0:
            exit(0);                            /* 終了 */
            break;
    }
}
static WSCfunctionRegister  op("Btn_Click",(void*)Btn_Click);
```

「system」関数の引数で指定した文字列のコマンドを実行します。コマンドが終了するまで、「system」関数から戻って来ません。WideStudio で作成したメイン画面のプロパティの「タイトル属性」を「WM 管理外」としたとき、そのメイン画面が常に最前面で表示されるため、メイン画面を非表示にしないと起動するプログラムが隠れてしまいます。また、メインプログラムを非表示にするとき、メイン画面のプロパティの「終了」が「オン」になっていると、プログラムが終了してしまうので、「オフ」にしています。

「Prj 1」をクリックすると、マウスを追いかける目玉プログラムが起動します。
「Prj 2」をクリックすると、時計が起動します。
「Prj 3」をクリックすると、電卓が起動します。
「Prj 4」をクリックすると、X ロゴが起動します。
「Prj 5」をクリックすると、コンフィグプログラムが起動します。
「Exit」をクリックすると、ランチャーを終了します。

４－３－２　多言語表示

「/wsproject/sample8」に、日英中韓の文字列を同時に表示するプログラムのサンプルソースが入っています。多言語を同時に表示するためには、文字コードを UTF-8 にする必要があります。そのため、サンプルソースのファイルもユニコード形式でないと正常に読み出すことができないのでご注意ください。ユニコードで書かれた文字列を表示するようにして、対応するフォントが存在すれば、多言語プログラムが実現します。フォント設定の仕方については『3-3-10 WideStudio/MWT の開発例』を参照してください。

多言語表示のサンプルプログラムを実行した画面を図 4-3-2-1 に示します。

図 4-3-2-1. 多言語表示実行画面

Algo Smart Display 標準では、中国語と韓国語のフォントがビットマップフォントしか実装されていないため最初の表示にかなりの時間を要します。TrueType の中国語、韓国語のフォントを実装すれば改善されます。

４－４　Algonomix設定ファイルについて

Algonomix にある各種設定ファイルについて代表的なものの設定例について説明します。

４－４－３　/home/asdusr/autostart

リスト 4-4-1-1.アプリケーション自動起動設定ファイル

```
#! /bin/sh
asd_config &  ◀―――――――― ①
```

①：自動起動するプログラム名を設定します。設定するプログラムは基本的に 1 つです。プログラムがウィンドウタイプのアプリケーションなら複数起動することができます。

４－４－４　/etc/network/interface

リスト 4-4-2-1. ネットワーク IP 設定ファイル

```
auto lo eth0 ◀―――――――― ①

iface lo inet loopback

iface eth0 inet static
  address 192.168.0.1
  netmask 255.255.255.0  ◀―――――――― ②
  gateway 192.168.0.254
```

①：AP210/310/315 では標準でネットワークポートが実装されているので、eth0 を自動的に組み込むように設定しています。AP110 では記述はありませんが、AP110 で USB-LAN アダプタを接続した場合、ここに eth0 を記述すれば、AP110 起動時に自動的に eth0 が読み込まれるようになります。
②：eth0 のネットワークアドレスを設定します。

４－４－５　/etc/hosts

リスト 4-4-3-1. ホスト名設定ファイル

```
127.0.0.1       asdap localhost  ◀―――――――― ①
```

ホスト名と対応する IP アドレスを設定します。
【書式】　IP アドレス ホスト名（複数指定有り）
①：asdap と localhost の IP アドレスがループバックアドレスと指定しています。

４－４－６　/etc/resolv.conf

リスト 4-4-4-1. DNS 設定ファイル

```
#search
nameserver 192.168.0.1  } ①
nameserver 192.168.6.1  }
```

①：利用する DNS サーバのアドレスを指定します。最大２つまで指定可能です。1 番目の IP を検索して見つからなかったとき 2 番目の IP を検索します。

４－４－７　/etc/profile

リスト 4-4-5-1. プロファイル設定ファイル

```
# /etc/profile
#

export PATH="$PATH:/usr/bin:/usr/sbin:/bin:/sbin:/usr/X11R6/bin"   } ①
export DISPLAY=localhost:0.0

#umask 022

# This fixes the backspace when telnetting in.
if [ "$TERM" != "linux" ]; then
        stty erase ^H
fi
```

①：環境変数の設定です。PATH は実行ファイルがあるディレクトリを指定することでフルパス指定せずにファイルを実行することができます。DISPLAY は X Window System が使用するディスプレイ番号とスクリーン番号を指定します。
お客様で環境変数を設定したい場合は、ここに記述します。設定が有効になるのは次回起動時です。

４－４－８　/etc/inittab

リスト 4-4-6-1. 初期設定ファイル

```
# /etc/inittab
#
::sysinit:/etc/init.d/rcS  <-------------------- ①

# Terminal
ttySC1::respawn:/sbin/getty -L 38400 ttySC1 vt100  <---------- ②
#tty1::respawn:/sbin/getty 38400 tty1

# Stuff to do before rebooting
::ctrlaltdel:/sbin/reboot  <-------------------- ③
::shutdown:/bin/umount -a -r  <-------------------- ④
```

このファイルで init プロセスが行うべき処理の定義を行います。書式と意味は以下の通りです。

※ Algonomix では busybox を採用しているため、一般的な inittab の書式とは文法が異なります。

【書式】　id:runlevel:action:process

【意味】

id	:エントリの識別子。ユニークな文字列。
runlevel	:ランレベルの指定（1～6）。busybox ではランレベルの概念はありません。
action	:プロセスの起動や終了時の動作を指定。代表的な物を以下に示します。
sysinit	:ブート時に実行するプロセス。
respawn	:process で指定したプロセスを起動し、終了されたら再起動します。
shutdown	:終了時に実行するプロセス。
ctrlaltdel	:「Ctrl」＋「Alt」＋「Delete」キーが押されたときの処理。
process	:起動するプログラム。

①：ブート時に「/etc/init.d/rcS」というシェルスクリプトを実行します。
②：Algo Smart Display のシリアルポートをシリアルコンソールとして使用する設定です。
この行の先頭に「#」をつけ、コメントアウトすることで、お客様の作成されたプログラムでシリアルポートを使用することができます。
③：「Ctrl」＋「Alt」＋「Delete」キーが押されたとき再起動します。
④：終了時にすべてのファイルシステムをアンマウントします。

４－５　動作確認済みUSB機器一覧

ALGO Smart Display で使用できる USB 機器の一覧を表 4-5-1 に示します。
その他のデバイスにつきましては、ホームページを参照してください。

表 4-5-1. 使用できる USB 機器一覧

種別	型名	メーカ	備考
マウス	―	―	通常の Linux 対応の USB マウスは問題なく使用できます。
キーボード	―	―	通常の Linux 対応の USB キーボードは問題なく使用できます。
USB メモリ	―	―	通常の Linux 対応の USB メモリは問題なく使用できます。
USB HUB	―	―	通常の Linux 対応の USB HUB は問題なく使用できます。

※注：USB HUB を使用して、複数の USB 機器を同時に接続する場合、単体では動作するが、同時に接続するものによっては動作しない可能性があります。詳細はホームページを参照してください。

４－６　起動画面の変更について

ALGO Smart Display の ver1.50 以上では起動画面を変更することができます。ここでは、起動画面の変更方法について説明します。

注：起動画面の ALGO Smart Display 本体のフラッシュメモリへの操作を含みます。操作を間違うと起動しなくなるおそれがあります。作業をする場合は十分注意してください。

４－６－１　起動画面用の画像について

起動画面を変更するには起動画面用の画像ファイルを用意する必要があります。
起動画面として使用できる画像ファイルは表 4-6-1 のようになります。

表 4-6-1. 起動画面画像ファイル

フォーマット	Windows ビットマップ
幅	480 ピクセル
高さ	210 ピクセル
色数	24 ビットカラー

画像ファイルは AP-110/210/310/315 のどの機種も表 4-6-1 の画像ファイルとなりますが、実際に表示される部分は AP-110/210 と AP-310/315 で異なります。

- AP-110/210 の場合
  上部に画像ファイルの画像中央部 320x210 のエリアが表示されます。
  下部にメッセージ表示が表示されます。
- AP-310/315 の場合
  上部に画像ファイルの画像が表示されます。
  下部にメッセージ表示が表示されます。

各機種での表示イメージを図 4-6-1 に示します。

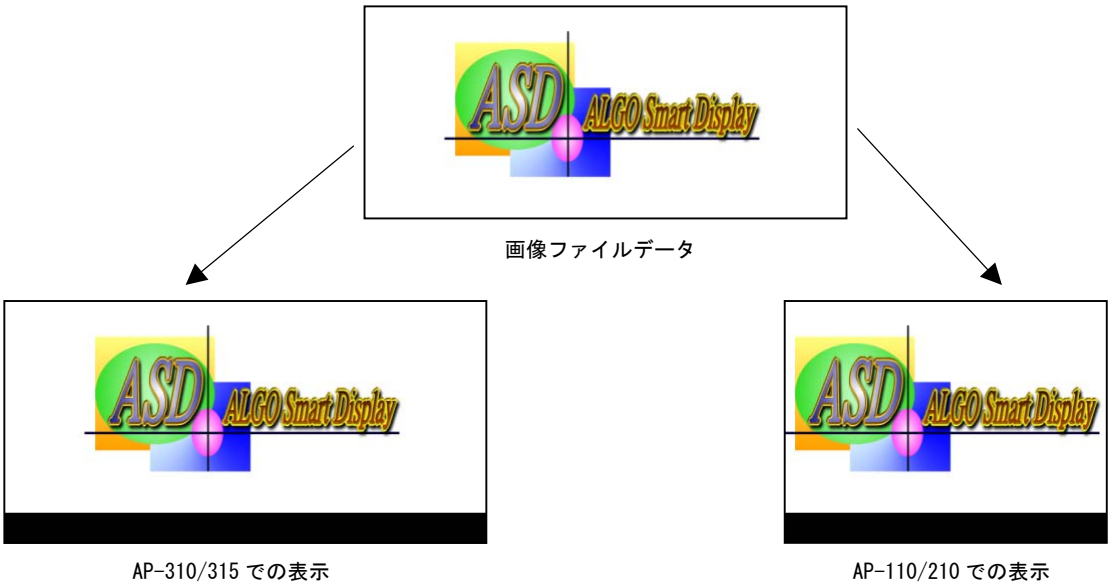

図 4-6-1. 起動画面表示イメージ

４－６－２　画像バイナリデータの作成

起動画面を変更するには ALGO Smart Display 本体のフラッシュメモリ保存するための画像バイナリデータが必要です。4-6-1 で説明した画像ファイルを画像バイナリデータの作成方法を説明します。

① 開発環境に画像フォーマットに従った画像ファイルを用意します。(sample.bmp)
② コマンドを実行し画像バイナリデータを作成します。

```
$ /usr/local/sh4-linux/bin/chbmp sample.bmp
```

③ sample.bmp.bin が作成されます。

４－６－３　画像バイナリデータの書き込み

4-6-2 で作成した画像バイナリデータを ALGO Smart Display 本体のフラッシュメモリに書き込みます。

注：操作を間違うと起動しなくなるおそれがあります。作業をする場合は十分注意してください。

● USB メモリを使って転送

① USB メモリ自動スクリプト（download.sh）と画像バイナリデータ（sample.bmp.bin）を USB メモリに格納します。USB メモリ自動スクリプトはリスト 4-6-3 のように記述します。

リスト 4-6-3.「download.sh」の例

```
#!/bin/sh
cp sample.bmp.bin /dev/mtdblock2
sync
```

② USB メモリを Algo Smart Display に挿入し、USB メモリ自動スクリプトを実行します。
③ USB メモリ自動スクリプトが終了（実行画面が閉じます）したら USB メモリを抜きます。
④ 再起動して画像を確認します。

注:USB メモリ自動スクリプト、USB メモリ自動スクリプトの動作については 3-3-9 を参照してください。

● ftp を使って転送

① 3-3-9 を参考に ftp で ALGO Smart Display 本体に画像バイナリデータ（sample.bmp.bin）を転送します。
② telnet またはシリアルコンソールで ALGO Smart Display 本体にログインします。
③ コマンドを実行してフラッシュメモリに書き込みを行います。

```
$ su
# cp sample.bmp.bin /dev/mtdblock2
# sync
```

④ 再起動して画像を確認します。

付録

Ａ－１　参考文献

●「ふつうのLinuxプログラミング Linuxの仕組みから学べるGCCプログラミングの王道」
著者　青木 峰郎
発行所　ソフトバンク パブリッシング
発行年　2005年

●「How Linux Works Linuxの仕組み」
著者　Brian Ward
訳　吉川 典秀
発行所　毎日コミュニケーションズ
発行年　2006年

●「WideStudio 徹底ガイドブック」
監修　坂村 健
編著　平林 俊一
共著　後藤 渉
末竹 弘之
川上 正平
平林 洋介
発行所　パーソナルメディア
発行年　2004年

●「TECHI Vol.16 組み込みLinux入門」
編集　インターフェース編集部
発行所　CQ出版社
発行年　2003年

●「Linuxデバイスドライバ 第３版」
著者　Jonathan Corbet
Alessandro Rubini
Greg Kroah-hartman
訳　山崎 康宏
山崎 邦子
長原 宏治
長原 陽子
発行所　オライリー・ジャパン
発行年　2005年

●「組み込みLinuxシステム構築」
著者　Karim Yaghmour
訳　林 秀幸
発行所　オライリー・ジャパン
発行年　2003年

MEMO

このユーザーズマニュアルについて

（1）本書の内容の一部又は全部を当社からの事前の承諾を得ることなく、無断で複写、複製、掲載することは固くお断りします。
（2）本書の内容に関しては、製品改良のためお断りなく、仕様などを変更することがありますのでご了承下さい。
（3）本書の内容に関しては万全を期しておりますが、万一ご不審な点や誤りなどお気付きのことがございましたらお手数ですが巻末記載の弊社もしくは、営業所までご連絡下さい。その際、巻末記載の書籍番号も併せてお知らせ下さい。

７７ＡＰ１０００１Ｇ　　２００８年　６月　３版

株式会社アルゴシステム

本社
〒５８７－００２１　大阪府堺市美原区小平尾６５６番地　　ＴＥＬ（０７２）３６２－５０６７
ＦＡＸ（０７２）３６２－４８５６

東京支社
〒１０４－００６１　東京都中央区銀座７－１５－８　銀座堀ビル２Ｆ　　ＴＥＬ（０３）３５４１－７１７０
ＦＡＸ（０３）３５４１－７１７５

大阪支社
〒５４２－００８１　大阪市中央区南船場１－１２－３　船場グランドビル３Ｆ　　ＴＥＬ（０６）６２６３－９５７５
ＦＡＸ（０６）６２６３－９５７６

名古屋営業所
〒４６１－０００４　愛知県名古屋市東区葵２－３－１５　ふぁみーゆ葵ビル５０３　　ＴＥＬ（０５２）９３９－５３３３
ＦＡＸ（０５２）９３９－５３３０

ホームページ　http://www.algosystem.co.jp