Algo Smart Panel シリーズ

マニュアル

Algo Smart Panel 用Linuxディストリビューション

『AlgonomixDFB 2』について

ＡＰ－１０００
ＡＰ－２０００
ＡＰ－３１００
ＡＰ－３１０１
ＡＰ－３１０２

目　次

はじめに

第１章　概要

第２章　システム構成

第３章　開発環境

第４章　Algo Smart Panelについて

付録

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 Panel 用Linuxディストリビューション（以降 **AlgonomixDFB 2**）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など多数ございます。これらの書籍等と併せて本書をお読みください。

２）　保証について

AlgonomixDFB 2の動作は組み込みパッケージのバージョンでのみ動作確認しております。詳細は「表2-1-2. 組み込まれているパッケージとそのバージョン」を参照してください。AlgonomixDFB 2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。

３）　Algo Smart Panel型番について

本書では、AP-1000/AP-2000/AP-3100/AP-3101/AP-3102 の Algo Smart Panel について説明しています。

第１章　概要

本章では、**AlgonomixDFB 2** の具体的な内容を説明する前に、AlgonomixDFB 2 の概要について説明します。

１－１　　AlgonomixDFB 2 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‥‥）
- DirectFB

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。
AlgonomixDFB 2 は弊社が、Algo Smart Panel 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。
AlgonomixDFB 2 用の開発環境イメージを図 1-1-1 に示します。AlgonomixDFB 2 用の開発環境を動作させる Linux OS として「Ubuntu」と「Red Hat Enterprise Linux WS Ver4」の 2 通りを推奨しています。

- Ubuntu ： Ubuntu 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。

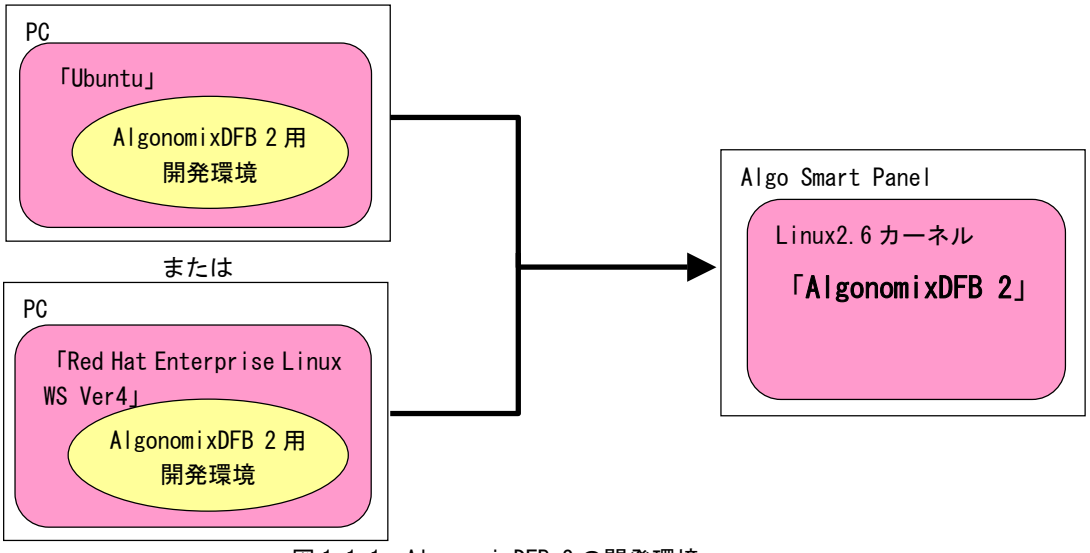


図 1-1-1. AlgonomixDFB 2 の開発環境

このマニュアルではUbuntuにAlgonomixDFB２用開発環境が組み込こまれたPCLinux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のホームページに記述されています。日本語に訳された定義の原本が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 DFB2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 Panel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OSのほかに、それぞれのデバイスドライバ、およびファイルシステム等が必要です。

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生産されていて欲しいものです。しかし、ハードは供給されていても、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 1-3-1 に示します。

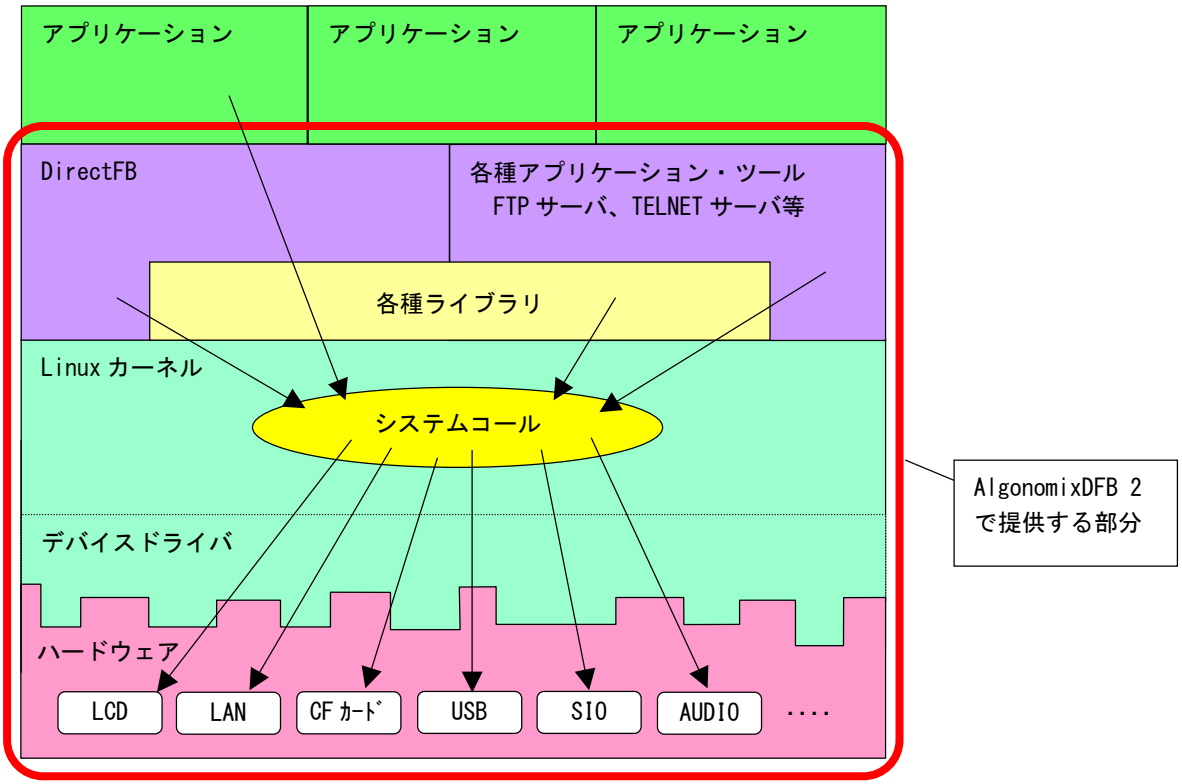


図 1-3-1. Linuxソフトウェア構成図

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 DirectFB 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 Panel 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 1-3-1 にある DirectFB 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　AlgonomixDFB 2 用パッケージDVD-ROMについて

Windows 起動後に、DVD-ROM の中身を開くと以下のようなフォルダ構成になっています。
※DVD-ROM ドライブを右クリックし、開くボタンで DVD-ROM フォルダをオープンしてください。
DVD-ROM は表 2-1-1 の構成になっています。

表 2-1-1. AlgonomixDFB 2 用パッケージDVD-ROMの構成

DVD-ROM のディレクトリ	内容
¥doc	Algo Smart Panel の取扱説明書が格納されています。 ・Software Users Manual (g1a2).pdf 本書です。
¥development	Ubuntu 以外の Linux ディストリビューションに AlgonomixDFB 2 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-dev2-src-g1a2-X.XX.tgz AlgonomixDFB 2 用開発環境のソースコードがパッケージされています。 **※AlgonomixDFB 2 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-wsproject2-g1a2-X.XX.tgz Algo Smart Panel のサンプルソースがパッケージされています。
¥Packages	標準のルートファイルシステムを作成する際に必要なパッケージが格納されています。詳細は「２－２　ルートファイルシステム作成用パッケージのディレクトリ構造」を確認してください。
¥CF-Image	CF カードルートファイルイメージが格納されています。 CF カードに DD コマンドでコピーすることで、標準のルートファイルシステムを作成することが出来ます。
¥kernel	Flash ROM に書き込むためのバイナリデータが格納されています。

表 2-1-2 に組み込みパッケージのバージョンを示します。

表 2-1-2. 組み込み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	2.6.22.10
alsalib	ALSA サウンドライブラリ	1.0.10
alsautils	ALSA サウンドユーティリティ	1.0.10
bash	bash シェル	3.0
busybox	組み込み向け基本コマンド環境	1.4.1
db	バークレーDB ライブラリ（Perl に必要なライブラリ）	2.7.7
directfb	DirectFB	1.1.1
e2fsprogs	EXT ファイルシステムユーティリティ	1.40.2
expat	XML パーサ	2.0.0
fbshot	フレームバッファスクリーンショット作成ソフト	0.3
fontconfig	フォント管理ライブラリ	2.3.94
freetype	TrueType フォント レタリング用ライブラリ	2.3.5
gdb	GNU デバッガ（gdbserver）	6.8
gdbm	GNU dbm データベース	1.8.3
ifupdown	ネットワーク設定ツール	0.6.8
jpeg	JPEG ライブラリ	6b
lesstif	Motif 互換用ライブラリ	0.94.4
libpng	PNG 画像の圧縮/展開処理用ライブラリ	1.2.8
libtiff	TIFF ライブラリ	3.8.2
libxml2	XML ライブラリ	2.6.9
mtd-utils	MTD ユーティリティ	20050619
murasaki	PnP マネージャ	0.9.1
ncurses	ターミナル用ライブラリ	5.5
netkit-base	INETD スーパーサーバ	0.16
netkit-ftp	FTP クライアント	0.17
netkit-telnet	TELNET サーバ	0.10
ntp	NTP クライアント	4.2.4
openssl	SSL と TLS を実装したライブラリ	0.9.8a
perl	perl 言語環境	5.8.5
plaympeg-dfb	MPEG プレーヤ（DirectFB 用）	0.4.4
portmap	portmap サービス	5
proftpd	FTP サーバ	1.3.0rc3
RICOH Font	RICOH TrueType フォント	－
samba	samba サービス（samba マウントのみ実装）	2.2.11-ja-1.0
sysfsutils	sysfs ユーティリティ	2.1.0
tcp_wrappers	TCP Wrapper サービス	7.6
termcap	端末機能データベース	2.0.8
widestudio	Widestudio ライブラリ	3.97-4
z lib	圧縮・解凍ライブラリ	1.2.3

２－２　　ルートファイルシステム作成用パッケージのディレクトリ構造

Algo Smart Panel では、Linux カーネル本体は NOR 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を作成しており、Linux システムに必要なファイルやディレクトリを保存しています。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。
以下にルートファイルシステムを作成する際に必要な Algonomix DFB2 用パッケージ DVD「Packages」フォルダ内のパッケージ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.XX.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。
リスト 2-2-1 にディレクトリ構成を、表 2-2-1 にディレクトリ内容を示します。

リスト 2-2-1. 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 2-2-1. 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 **※/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。 ・CF カード：/dev/sda1 CF カードをアクセスするためのデバイスファイルです。 ・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：ttySC0、ttySC1、ttySC2 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。 ・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。FlashROM ブロックデバイスの詳細については『２－５　プラットフォームの FlashROM イメージについて』を参照してください。 **※　/dev/mtdblock にアクセスする際は、IPL やカーネルイメージ等を間違って消さないように十分注意が必要です。** ・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。DirectFB 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：sdb1、sdb2 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。
/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 Panel 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。

＜AP-1000 用の追加用パッケージファイル（ap110-X.XX.tgz）＞

AP-1000 専用の追加用パッケージです。
AP-1000 にはパネルスイッチが 5 つ実装されています。
この追加パッケージにはパネルスイッチの制御用ファイルや追加のカーネルモジュールを含んでいます。
リスト 2-2-2 にディレクトリ構成を示します。

リスト 2-2-2. AP1000 用追加パッケージのディレクトリ構成

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

＜AP-2000/3100 用の追加用パッケージファイル（ap310-X.XX.tgz）＞

AP-2000/3100 専用の追加用パッケージです。
AP-2000/3100 にはネットワークポートおよび、汎用入出力を標準で実装されています。
この追加パッケージにはネットワークポートの設定ファイルおよび、汎用入出力（IN6、OUT4）の制御用ファイルや追加のカーネルモジュールを含んでいます。
リスト 2-2-3 にディレクトリ構成を示します。

リスト 2-2-3. AP2000/AP3100 用追加ファイルシステムのディレクトリ構成

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

＜AP-3101 用の追加用パッケージファイル（ap315-X.XX.tgz）＞

AP-3101 専用の追加用パッケージです。
AP-3101 にはネットワークポートおよび、汎用入出力を標準で実装されています。
この追加パッケージにはネットワークポートの設定ファイルおよび、汎用入出力（IN6、OUT4）の制御用ファイル、RAS 機能や追加のカーネルモジュールを含んでいます。
リスト 2-2-4 にディレクトリ構成を示します。

リスト 2-4-4. AP3101 用追加ファイルシステムのディレクトリ構成

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

＜AP-3102 用の追加用パッケージファイル（ap315a-X.XX.tgz）＞

AP-3102 専用の追加用パッケージです。
AP-3102 にはネットワークポートおよび、汎用入出力を標準で実装されています。
この追加パッケージにはネットワークポートの設定ファイルおよび、汎用入出力（IN6、OUT4）の制御用ファイル、RAS 機能や追加のカーネルモジュールを含んでいます。
リスト 2-2-5 にディレクトリ構成を示します。

リスト 2-4-5. AP-3102 用追加ファイルシステムのディレクトリ構成

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

＜DirectFB 追加用パッケージファイル（directfb-X.XX.tgz）＞

DirectFB を実装するために追加パッケージです。

※　DirectFB と X Window System の共存はできませんので注意してください。

リスト 2-2-6 にディレクトリ構成を、表 2-2-2 にディレクトリ内容を示します。

リスト 2-2-6. DirectFB追加用パッケージのディレクトリ構成

```
+--etc
+--lib
+--usr--+--bin
        +--lib
        +--local--+--bin
        |         +--share
        +--share
```

表 2-2-2. DirectFB追加用パッケージのディレクトリ内容（抜粋）

ディレクトリ名	内容
/etc	WideStudio のフォント情報が格納されています。
/usr/bin	DirectFB のコマンドが格納されています
/usr/lib	DirectFB 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/local/bin	DirectFB のサンプルが格納されています。

＜DirectFB 用 ASD Config 追加用パッケージ（asdconfig-dfb.X.XX.tgz）＞

DirectFB 用の ASD Config や専用のプログラムです。

※　X Window System 上では動作しませんので注意してください。

リスト 2-2-7 にディレクトリ構成を、表 2-2-3 にディレクトリ内容を示します。

リスト 2-2-7. DirectFB用ASD Config追加用パッケージのディレクトリ構成

```
+--etc
+--usr--+--bin
        +--share--+--asd--+--image
                          +--sound
```

表 2-2-3. DirectFB用ASD Config追加用パッケージのディレクトリ内容（抜粋）

ディレクトリ名	内容
/usr/bin	DirectFB 用の ASD Config が格納されています。
/usr/share/asd	ASD Config で使用するイメージデータおよび音声データが格納されています。

＜Perl 言語の追加用パッケージ（perl-X.XX.tgz）＞

Algo Smart Panel で Perl 言語を実装するための追加用パッケージです。

リスト 2-2-8 にディレクトリ構成を、表 2-2-4 にディレクトリ内容を示します。

リスト 2-2-8. Perl言語追加用パッケージのディレクトリ構成

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

表 2-2-4. Perl言語追加用パッケージのディレクトリ内容（抜粋）

ディレクトリ名	内容
/usr/bin	Algo Smart Panel に必要な Perl 言語の実行ファイルが格納されています。
/usr/lib	Algo Smart Panel に必要な Perl 言語のライブラリが格納されています。

＜TrueType フォントの実装（sharefonts-X.XX.tgz）＞

Algo Smart Panel で TrueType フォントを実装するための追加パッケージです。
リスト 2-2-9 にディレクトリ構成を、表 2-2-5 にディレクトリ内容を示します。

リスト 2-2-9. sharefonts追加用パッケージのディレクトリ構成

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

表 2-2-5. sharefonts追加用パッケージのディレクトリ内容（抜粋）

ディレクトリ名	内容
/usr/share	リコーフォントと Type1 フォントが格納されています。 リコーフォント：HG ゴシック M, HGP ゴシック M

＜MPEG プレーヤ追加用パッケージ（plaympeg-dfb.X.XX.tgz）＞

Algo Smart Panel で MPEG プレーヤを実装するための追加パッケージです。
※　X Window System 上では動作しませんので注意してください。
リスト 2-2-10 にディレクトリ構成を、表 2-2-6 にディレクトリ内容を示します。

リスト 2-2-10. MPEGプレーヤ追加用パッケージのディレクトリ構成

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

表 2-2-6. MPEGプレーヤ追加用パッケージのディレクトリ内容（抜粋）

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

＜MPEG プレーヤ(フレームバッファ直接描画版)の追加用パッケージ（plaympeg-hi-X.XX.tgz）＞

Algo Smart Panel で MPEG プレーヤを実装するための追加パッケージです。動画の描画をフレームバッファに直接書込みます
リスト 2-2-11 にディレクトリ構成を、表 2-2-7 にディレクトリ内容を示します。

リスト 2-2-11. MPEGプレーヤ(フレームバッファ直接描画版)追加用パッケージのディレクトリ構成

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

表 2-2-7. MPEGプレーヤ(フレームバッファ直接描画版)追加用パッケージのディレクトリ内容（抜粋）

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

２－３　　ルートファイルシステムの作成方法

Algo Smart Panel で使用する CF カード用のルートファイルシステムを作成する方法を以下に示します。

①USB 接続の CF カードリーダを開発環境の入ったパソコンに接続します。

※　本書では、CF カードが/dev/sdb と認識された場合を示しています。

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。

以下のコマンドを実行して、ルート権限になります。

パスワード:asdusr

```
$ sudo su
# /sbin/fdisk /dev/sdb
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/sdb: 513 MB, 513277952 bytes
9 heads, 40 sectors/track, 2784 cylinders
Units = シリンダ数 of 360 * 512 = 184320 bytes

 デバイス Boot      Start         End      Blocks   Id  System
/dev/sdb1               1        2785      501131+   6  FAT16

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/sdb: 513 MB, 513277952 bytes
9 heads, 40 sectors/track, 2784 cylinders
Units = シリンダ数 of 360 * 512 = 184320 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のまま次へいきます。パーティション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 (1-2784, 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 (1-2784, default 2784):
Using default value 2784

コマンド (m でヘルプ): p

Disk /dev/sdb: 513 MB, 513277952 bytes
9 heads, 40 sectors/track, 2784 cylinders
Units = シリンダ数 of 360 * 512 = 184320 bytes

 デバイス Boot      Start         End      Blocks   Id  System
/dev/sdb1               1        2784      501100+  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。

※　Ubuntu では、以下のコマンドを実行する前にデスクトップ上から CF カードをアンマウントする必要があります。

```
# /sbin/mkfs.ext3 /dev/sdb1
mke2fs 1.38 (30-Jun-2005)
Filesystem label=
OS type: Linux
～～～～～省略～～～～～
Writing inode tables: done
Creating journal (8192 blocks): done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 28 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/sdb1 /mnt
```

⑧CF カードにパッケージをインストールする場合は下記の順序で行ってください。

1. ルートファイルファイルシステム（base-x.xx.tgz）
↓
2. 追加用パッケージ(ap110-x.xx.tgz / ap310-x.xx.tgz / ap315-x.xx.tgz / ap315a-x.xx.tgz)
↓
3. DirectFB 追加用パッケージ(directfb-X.XX.tgz)
↓
4. 他のパッケージ(asdconfig-x.X.XX.tgz, sharefonts-X.XX.tgz, plaympeg-x.X.XX.tgz 等)

⑨CF カードにルートファイルシステム（base パッケージ）をインストールします。

```
# tar zxvf <DVD ドライブ>/Packages/base-X.XX.tgz -C /mnt
```

⑩CF カードに固有モジュールである追加用パッケージをインストールします。

```
# tar zxvf <DVD ドライブ>/Packages/apXXX-X.XX.tgz –C /mnt
```

apXXX-X.XX.tgz ： ap110-X.XX.tgz / ap310-X.XX.tgz / ap315-X.XX.tgz / ap315a-x.xx.tgz

⑪CF カードに DirectFB 追加用パッケージをインストールします。

```
# tar zxvf <DVD ドライブ>/Packages/directfb-X.XX.tgz –C /mnt
```

⑫CF カードに perl の追加用パッケージをインストールします。

```
# tar zxvf <DVD ドライブ>/Packages/perl-X.XX.tgz –C /mnt
```

⑬CF カードに TrueType フォント追加用パッケージをインストールします。

```
# tar zxvf <DVD ドライブ>/Packages/sharefonts-X.XX.tgz –C /mnt
```

⑭CF カードに DirectFB 用 ASD Config 追加用パッケージをインストールします。

```
# tar zxvf <DVD ドライブ>/Packages/asdconfig-dfb-X.XX.tgz –C /mnt
```

⑮ここまでが、標準で CF カードに実装するべきものです。
MPEG プレーヤなどの他のパッケージをインストールしたい場合は、⑨～⑭を参考にや「plaympeg-dfb-X.XX.tgz」などのイメージファイルを CF カードにインストールします。

⑯CF カードをアンマウントします。

```
# umount /mnt
```

以上で CF カード作成が完了しました。
Algo Smart Panel に CF カードを挿入し起動後、動作確認をしてください。
また、お客様で作成されたプログラムとそのプログラムに必要なライブラリや設定ファイル等を tar 形式で圧縮しておくと⑨～⑭のように、新規作成した CF カードにインストールすることができます。
※　tar 形式で圧縮または展開するときは、ディレクトリの位置に注意してください。

２－４　　Algo Smart Panelコンフィグプログラムについて

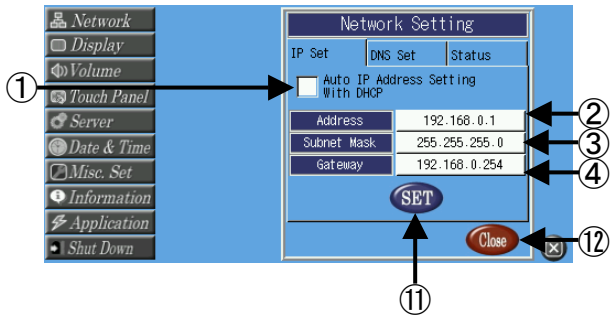
出荷状態では Algo Smart Panel を起動した時、下図のような Algo Smart Developer Config プログラム(以下 ASD Config と称す)のメイン画面が起動されます。
この ASD Config 画面のボタンより、Algo Smart Panel の各種設定を行うことができます。

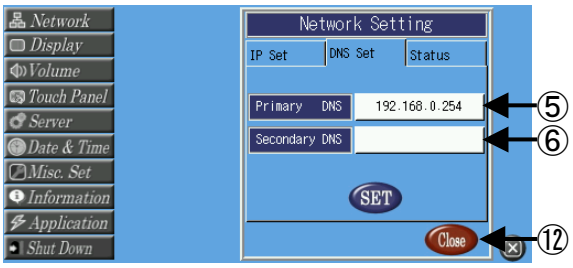
メイン画面ボタン一覧表で、それぞれのボタンの意味について説明します。

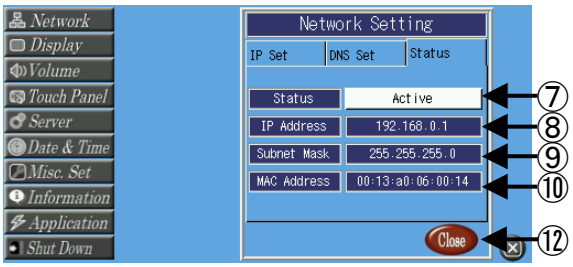
表 2-4-1. ASD Config一覧

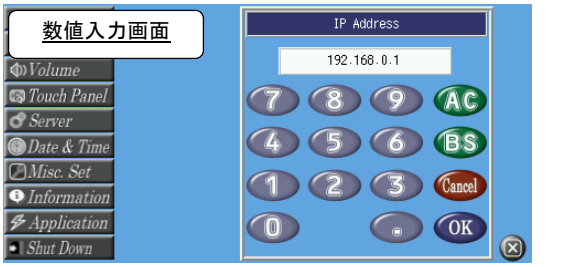
ボタン	名称	内容
	Network Setting	IP アドレスと DNS アドレスを設定します。
	Display Setting	液晶パネルのバックライトの設定ができます。
	Volume Setting	ステレオオーディオ出力端子の音量を調節できます。
	Touch Panel Calibration	タッチパネルのキャリブレーションを実行できます。
	Server Setting	TELNET サーバと FTP サーバを実行するかしないかを設定できます。 設定は Algo Smart Panel の再起動後に反映されます。
	Date & Time	Algo Smart Panel の現在日時と NTP サーバ、ローカル情報を設定できます。
	Misc. Setting	その他の設定を行います。
	Hardwear Information	Algo Smart Panel ハードウェア情報を表示します。 カーネル、システムのネットワークアップデートを実行できます。
	User Application	ユーザーアプリケーションを実行できます。
	AlgonomixDFB 2 Shutdown	AlgonomixDFB 2 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。

２－４－１　Network　Network Setting









①DHCP 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。

②IP アドレスの設定
　クリックすると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。

③サブネットマスクの設定
　クリックすると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。

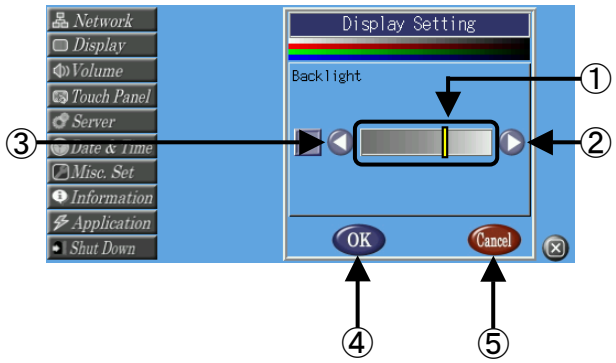
⑨取得サブネットマスク
　現在、取得しているサブネットマスクを表示しています。

⑩MAC アドレスの表示
　固有の MAC アドレスを表示しています。
　変更は不可です。

⑪SET ボタン
　現在の設定を反映し、保存します。

⑫Close ボタン
　メイン画面に戻ります。

２－４－２　Display　Display Setting



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。
②③のボタンをクリックすることで上下します。
8 段階で調節可能です。

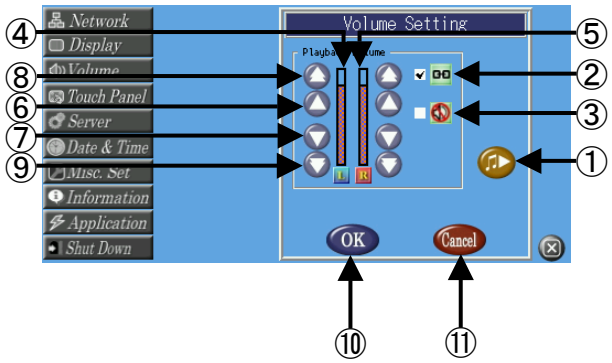
②UP ボタン
クリックすると液晶のバックライトが 1 段階明るくなります。

③DOWN 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。

④OK ボタン
クリックすると現在のディスプレイの設定を保存してメイン画面へ戻ります。
次回起動時でも、設定したディスプレイの設定が保持されます。

⑤Cancel ボタン
クリックすると設定前の状態に戻した上で、メイン画面へ戻ります。

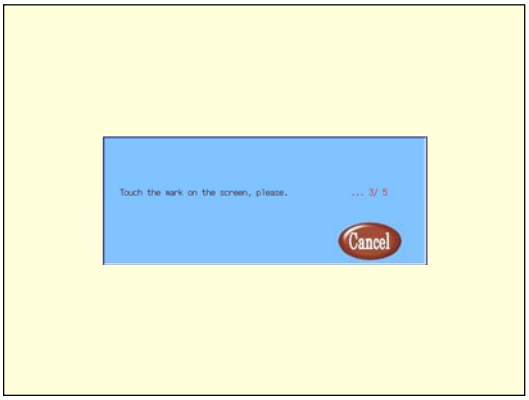
２－４－３ Volume Setting

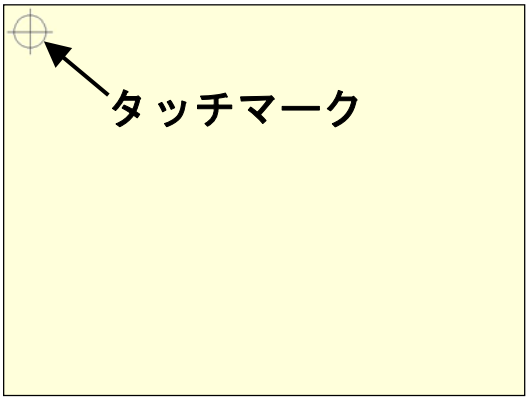


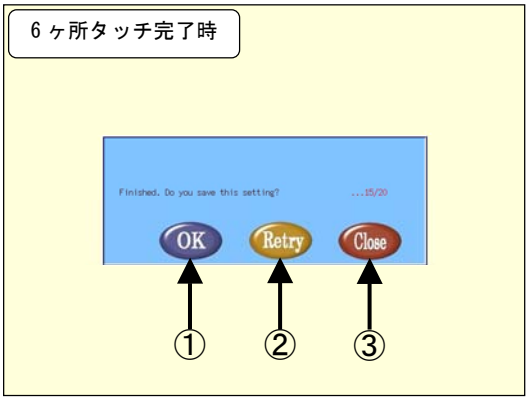
Playback Volume	
Playback	オーディオ出力レベル設定

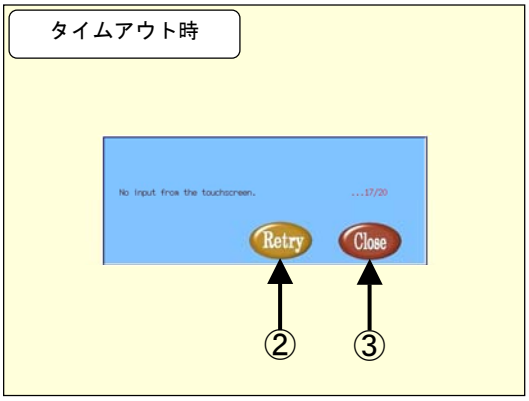
①サンプル音再生
　クリックするとサンプル音を再生します。
②シンクロ設定
　チェックを入れると､LとRで音量を同じにします。
　⑥～⑨のボタンをクリックすると、両方のメータが連動して上下します。
③ミュート設定
　チェックを入れると、ミュートにします。
④左音量メータ
　ステレオ左の音量を表示しています。⑥～⑨のボタンをクリックすることで上下します。
⑤右音量メータ
　ステレオ右の音量を表示しています。
　⑥～⑨のボタンをクリックすることで上下します。
⑥微 UP ボタン
　クリックすると音量が 1 段階大きくなります。
⑦微 DOWN ボタン
　クリックすると音量が 1 段階小さくなります。
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。
⑩OK 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。
　次回起動時も、設定が保持されます。
⑪Cancel ボタン
　クリックすると変更前の設定に戻したうえで、メイン画面へ戻ります。

２－４－４ Touch Panel Touch Panel Calibration









メイン画面で Touch Panel ボタンをクリックした後、

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。

準備が整ったら、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。

※　なるべく中心を正確にタッチしてください。キャリブ結果に影響されます。

※　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､6 ヶ所タッチ完了時の画面が表示されます。

①OK 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。

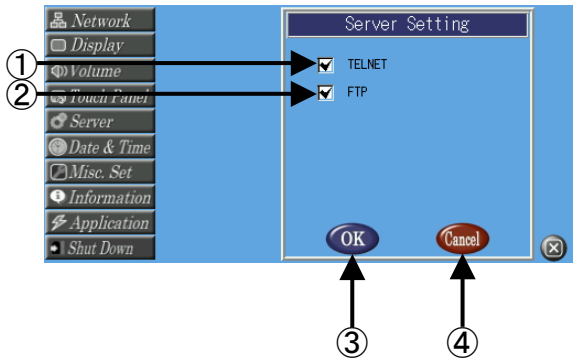
②Retry ボタン
クリックすると再度タッチパネルキャリブレーションを実行します。

③Close ボタン
キャリブレーション結果をキャリブレーション実行前の状態に戻して終了します。

※　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。

※　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設定前の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。

２－４－５　Server　Server Setting



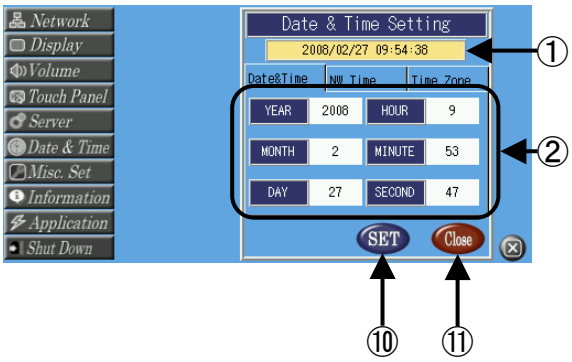
①TELNET サーバ自動起動設定
　チェックすると、次回 Algo Smart Panel を起動したとき、TELNET サーバを自動的に起動します。
　チェックをはずすと、次回 Algo Smart Panel を起動したとき、TELNET サーバは起動されません。
※　結果が反映されるのは、次回 Algo Smart Panel を起動したときです。

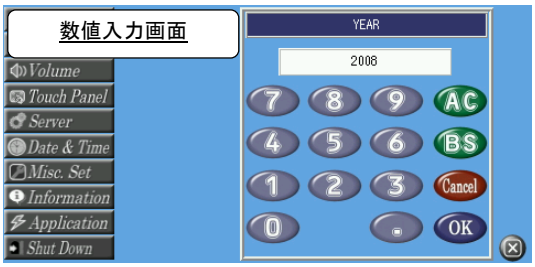
②FTP サーバ自動起動設定
　チェックすると、次回 Algo Smart Panel を起動したとき、FTP サーバを自動的に起動します。
　チェックをはずすと、次回 Algo Smart Panel を起動したとき、FTP サーバは起動されません。
※　結果が反映されるのは、次回 Algo Smart Panel を起動したときです。

③OK 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。

④Cancel 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。

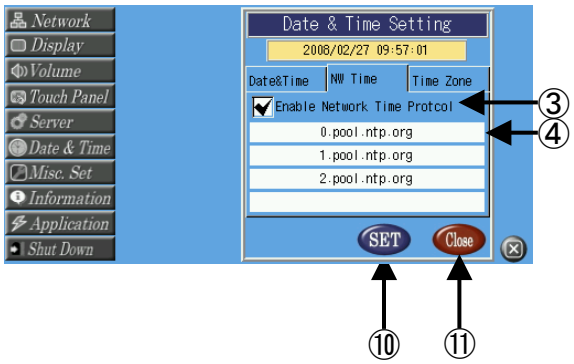
２－４－６　Date & Time　Date & Time

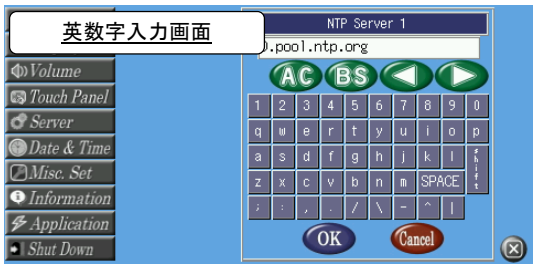


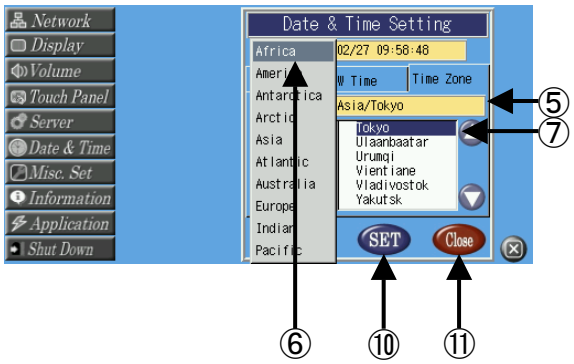


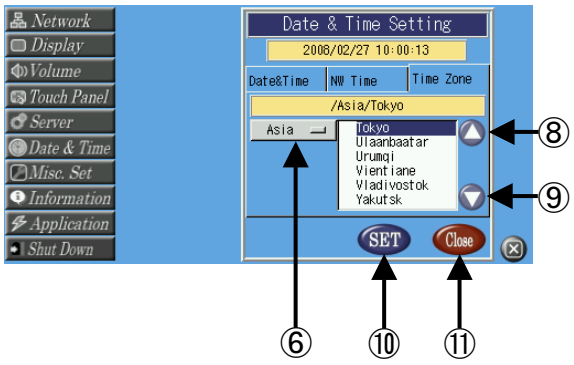
①現在時刻の表示
　現在のシステムクロックを表示しています。
②年、月、日、時、分、秒の設定
　クリックすると、数値入力画面が表示されます。
　正しい日付や時間を入力してください。
　⑩の SET ボタンが押されるまで設定は反映されません。秒の設定をするときはタイムラグを計算した上で設定してください。

※　③の Network Time Protcol(NTP)サーバが無効の時のみ設定することができます。









③NTP サーバを使用するか使用しないかを設定できます。チェックを入れると NTP サーバは有効です。

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

④NTP サーバの設定

ここで使用する NTP サーバを設定します。4 つまで登録することができます。

クリックすると、英数字入力画面が表示されます。NTP サーバ名を入力してください。

※　③の Network Time Protcol(NTP)サーバが有効の時のみ設定することができます。

⑤選択しているタイムゾーンの地域・都市を表示します。

⑩の SET ボタンを押すことで設定が反映されます。

⑥使用するタイムゾーンの地域の選択を行います。選択することで⑦に選択できる都市のリストが表示されます。

⑦使用するタイムゾーンの都市の選択を行います。

⑧表示されるリストが上方向にスクロールされます。

⑨表示されるリストが下方向にスクロールされます。

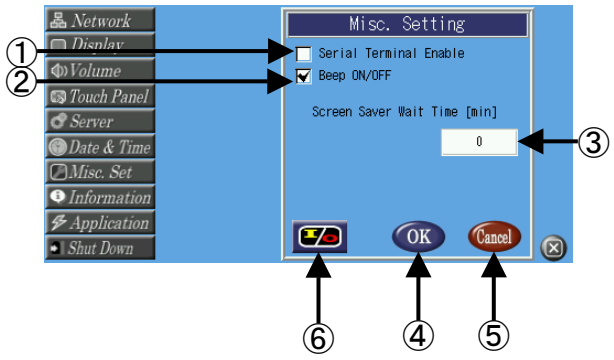
⑩SET ボタン

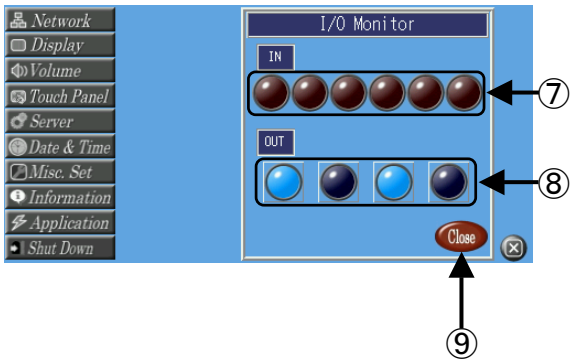
クリックすると設定を保存します。

⑪Close ボタン

クリックするとメイン画面へ戻ります。

２－４－７　Misc. Set　Misc. Setting





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 Panel 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場合はチェックをはずしてください。

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

②Beep 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に 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。

③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。
スクリーンセーバが起動すると画面が真っ黒になります。画面を元に戻したい時は、画面にタッチしてください。

④OK 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。

⑤Cancel 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。

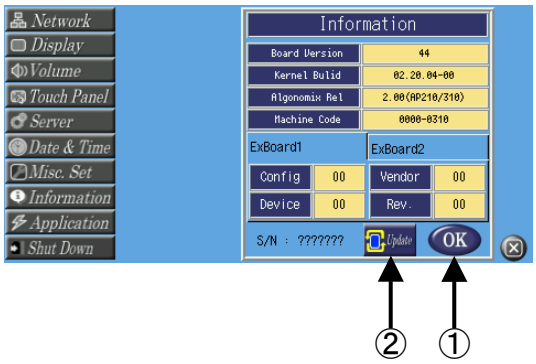
⑥I/O モニタボタン
汎用入出力のモニタができます。
ボタンを押すと I/O Monitor 画面に移行します。

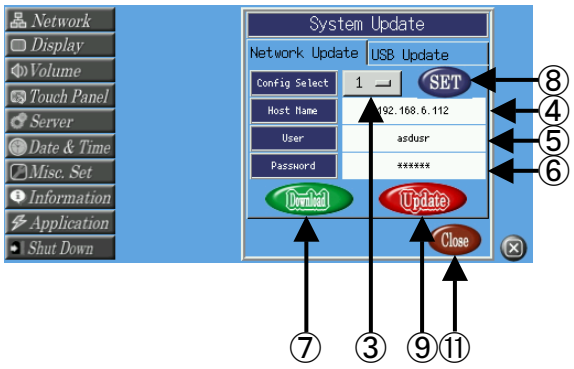
⑦IN モニタ
現在の汎用入力状態を表示します。

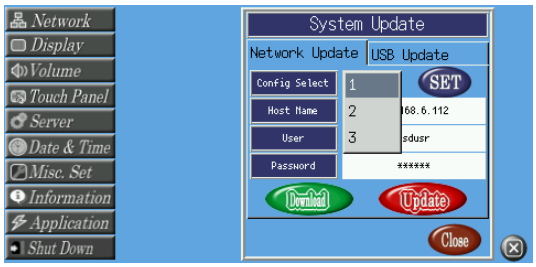
⑧OUT ボタン
クリックすることで汎用出力を ON/OFF することができます。

⑨Close ボタン
クリックすると、Misc. Setting 画面へ戻ります。

２－４－８ Information Hardwear Information



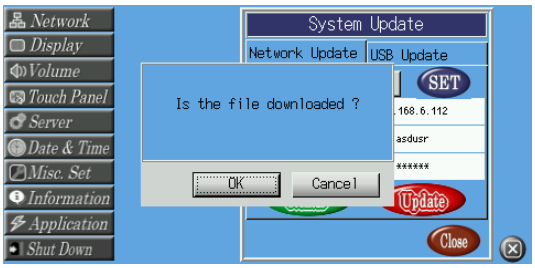


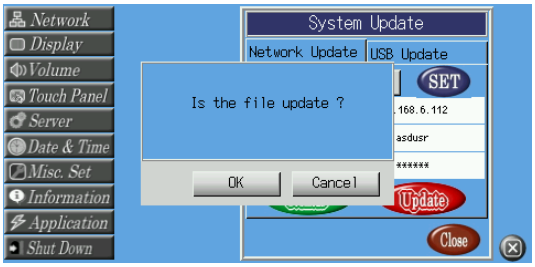


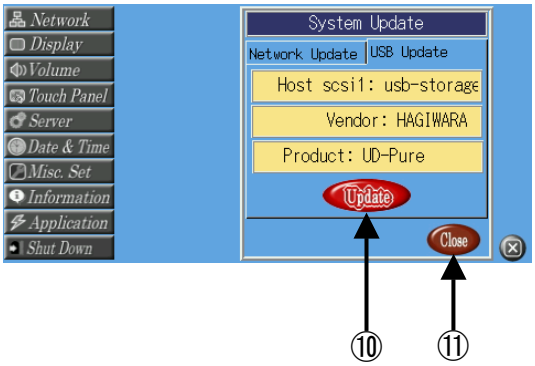
ここで、Algo Smart Panel のハードウェア情報、ファームウェアバージョンを確認することができます。下記に各項目の意味について示します。

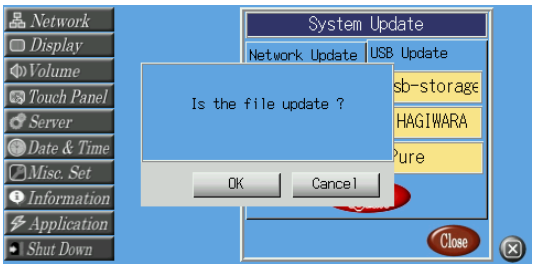
項目名	内容
Board Version	ハードウェアバージョン
Kernel Build Version	Linux カーネルのビルドバージョン
Algonomix Rel Version	AlgonomixDFB 2 構成のリリースバージョン
Machine Code	Algo Smart Panel を区別するための型式です。 ・0000-0110:AP1000 ・0000-0210:AP2000 ・0000-0310:AP3100 ・0000-0315:AP3101/AP3102
ExBoard1	拡張ボード 1 台目の情報
ExBoard2	拡張ボード 2 台目の情報
Config	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

①OK ボタン
　クリックすると、メイン画面へ戻ります。
②Update ボタン
　ネットワーク上のアップデートスクリプトを使用してカーネル及びシステムのアップデートを行うことができます。ボタンを押すと System Update 画面に移行します。
③コンフィグ選択
　ホスト設定番号です。3 種類のホスト設定を保存できます。
④ホスト名（アドレス）
　ホスト名（アドレス）を設定します。
　クリックすると、英数字入力画面が表示されます。ホスト名（アドレス）を入力してください。入力後、OK ボタンでホスト名（アドレス）を確定します。
⑤ユーザー名
　ユーザー名を設定します。
　クリックすると、英数字入力画面が表示されます。ユーザー名を入力してください。入力後、OK ボタンでユーザー名を確定します。









⑥パスワード

パスワードを設定します。

クリックすると、英数字入力画面が表示されます。パスワードを入力してください。入力後、OK ボタンでパスワードを確定します。

※　パスワードの入力はアスタリスクになります。

⑦ダウンロードボタン

④～⑥に設定されたホストに対してファイルのダウンロードを実行します。ポップアップ画面が表示され「Is the file downloaded ?」のメッセージが表示されます。ダウンロードを行う場合は OK ボタンを押してください。行わない場合は Cancel ボタンを押してください。

※　ダウンロードするファイル名は『download.sh』『 asd-update.tgz 』『 sumcheck_download 』『sumcheck_update』になります。それ以外のファイルはダウンロードできません。ダウンロード終了後、ダウンロードファイルの CRC チェックを実行します。CRC チェックに失敗した場合、ダウンロードファイルは削除されます。

※　『sumcheck_download』ファイル内に『download.sh』の CRC 値が入っています。『sumcheck_download』はユーザーが作成する必要があります。「2-7 CRC ファイルチェックについて」を参照してください。

※　『sumcheck_update』ファイル内に『asd-update.tgz』の CRC 値が入っています。『sumcheck_update』はユーザが作成する必要があります。「2-7 CRC ファイルチェックについて」を参照してください。

※　ダウンロードしたファイルは「/home/asdusr/」に保存されます。ダウンロードが完了したら「/home/asdusr/」にファイルがダウンロードできているかを確認してください。

⑧SET ボタン

クリックすると、ホスト設定番号にホスト名(アドレス)、ユーザー名、パスワードを保存します。

⑨ネットワークのアップデートボタン

⑦でダウンロードしたファイルをファイルシステムに組み込みます。ポップアップ画面が表示され「Is the file update ?」のメッセージが表示されます。アップデートを行う場合は OK ボタンを押してください。行わない場合はキャンセルボタンを押してください。

※　ダウンロードしたファイルのアップデートを行います。それ以外のファイルはアップデートできません。

※　アップデートはダウンロードした『download.sh』スクリプトファイルを実行します。ユーザーが『download.sh』を作成することによって自由にシステムのアップデートをすることができます。

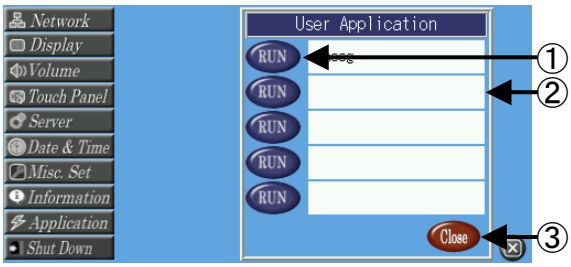
⑩USB のアップデートボタン

USB メモリを挿入した場合は USB メモリの情報が表示されます。アップデートは USB メモリにある『download.sh』スクリプトファイルを実行します。ユーザーが『download.sh』を作成することによって、自由にシステムのアップデートをすることができます。

⑪Close ボタン

クリックすると、設定を保存せずに Information 画面へ戻ります。

２－４－９ Application User Application



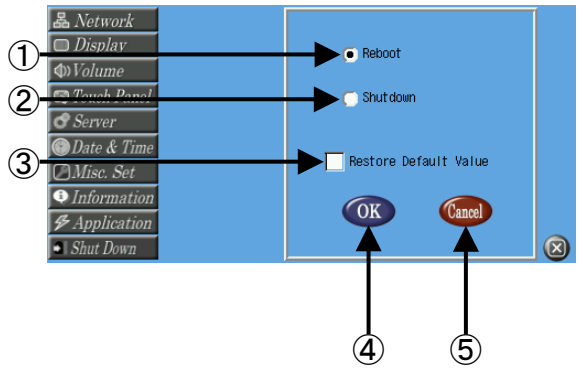
①RUN ボタン
クリックすると、RUN ボタンの右に指定したアプリケーションを実行します。

②アプリケーション名
クリックすると、英数字入力画面が表示されます。アプリケーション名をフルパスで入力してください。入力後、OK ボタンでアプリケーション名を確定します。

例 1)ユーザーのホームディレクトリにある Sample という実行ファイルを実行する場合は、アプリケーション名に「/home/asdusr/Sample」と入力します。

③Close ボタン
クリックすると、アプリケーション名を保存し、メイン画面へ戻ります。

２－４－１０ Shut Down Shutdown



①再起動
　こちら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 AlgonomixDFB 2 を再起動します。
②シャットダウン
　こちら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 AlgonomixDFB 2 をシャットダウンします。
　※電源は落ちません。
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、下図の画面で確認を促します。
④OK ボタン
　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。
⑤Cancel ボタン
　クリックすると、メイン画面へ戻ります。

＜初期設定＞

設定	
バックライト	16
ヘッドホン音量(左/右)	31/31
IP	192.168.0.1
サブネットマスク	255.255.255.0
TELNET 自動起動	有効
FTP 自動起動	有効
Beep 音	ON
スクリーンセーバ	OFF
NTP サーバ	無効
タイムゾーン	Asia/Tokyo

２－５　　プラットフォームのFlashROMイメージについて

Algo Smart Panel には IPL、Linux カーネル等が保存される 8Mbyte(AP-3101 のみ 32Mbyte)の NOR Flash ROM(以下、NOR フラッシュ)が実装されています。NOR フラッシュは IPL や、Linux カーネル本体や、システムに必要不可欠なデータが保存されておりユーザが自由にアクセスすることはできないません。

IPL、カーネル等は、アップデートする場合や、ユーザでカーネルを変更しカーネルを書き換えたい場合のみアクセスするようにしてください。

以下に NOR フラッシュメモリマップを示します。

表 2-5-1. MTD ブロック機能一覧

デバイス名	機能	サイズ	書き込みファイル名
/dev/mtdblock0　※1	IPL	32KByte	-
/dev/mtdblock1　※1	Config	32KByte	-
/dev/mtdblock2　※2	Logoimage	256KByte	chbmp で変更した bmp
/dev/mtdblock3　※2	Normal Kernel	3MByte	vmlinux-XXXXXXXX.bin
/dev/mtdblock4　※1	Safety Kernel	3MByte	-
/dev/mtdblock5　※1	Reserve	56Kbyte	-
/dev/mtdblock6　※1	Serial No.	8KByte	-

※1　このデバイスに対しては書き込みを行わないでください。OS が起動しなくなります。
※2　書き込むデータはそれぞれの機能毎に用意されたものを書いてください。

２－５－１　　NORフラッシュの書き換え

カーネルや起動時の画面イメージ等を書き換えることができます。

＜Normal Kernel の書き換え＞

```
# cp /home/asdusr/vmlinux-XXXXXXXX.bin /dev/mtdblock3
# sync
#
```

＜起動画面イメージの書き換え＞

```
# cp /home/asdusr/<chbmp で変更した bmp ファイル> /dev/mtdblock2
# sync
#
```

２－６　　Linuxカーネルの復旧

Algo Smart Panel では、NOR フラッシュ内の 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。ASD Config 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。

※　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。

①Algo Smart Panel 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 2-6-1 のようにオープニング画面の色が青色になります。画面が青色になるまで押し続けてください。

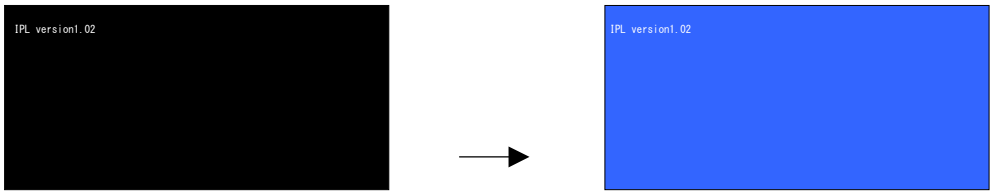


図 2-6-1.　セーフモードへの移行画面

②オープニング画面の色が青色になりましたら、10 秒以内に図 2-6-2 のように画面をなぞってください。

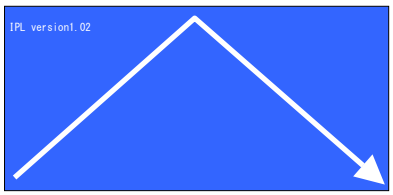


図 2-6-2.　セーフモードへの移行手順

③認識されたら図 2-6-3 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。

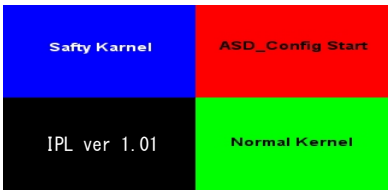


図 2-6-3.　起動メニュー画面

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。

※　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、正常なカーネルイメージを書き込んでください。書き込み方法は『2-5-1 NOR フラッシュの書き換え』を参照してください。

２－７　　sysfsファイルシステム

Linux2.6カーネルは sysfs ファイルシステムを導入しました。sysfs ファイルシステムは、proc、devfs、devpty のファイルシステムの統合だと言えます。sysfs ファイルシステムは、システムに接続されているデバイスとバスを、ユーザースペースからアクセスできるファイルシステム内に階層式に 列記します。以前は「/proc」内に存在していたデバイスとドライバー特定のオプションを処理したり、以前に devfs で提供されていた動的デバイス追加を担当するように設計されています。

sysfs ファイルシステムは /sys/ でマウントされ、いくつか異なる方法でシステムに接続されたデバイスを構成する複数のディレクトリを含んでいます。

AP-1000/2000/3100/3101/3102 では以下のデバイスの制御が可能です。

２－７－１　バックライト

AP-1000/2000/3100/3101/3102 のバックライトを制御します。

表 2-7-1-1. AP-1000 の Sysfs のバックライトを制御する項目

sysfs ファイル名	データ	内容
/sys/class/backlight/ap110-bl/brightness	0～15	0～15 の数値を Write することで、バックライトの光量を調節します。 Read すると現在の光量設定値が読み出せます。
/sys/class/backlight/ap110-bl/power	0 or 1	0 を Write することでバックライトを ON します。 1 を Write することでバックライトを OFF します。 Read することで現在のバックライトの ON/OFF 状況が読み出せます。

表 2-7-1-2. AP-2000/3100 の Sysfs のバックライトを制御する項目

sysfs ファイル名	データ	内容
/sys/class/backlight/ap310-bl/brightness	0～255	0～255 の数値を Write することで、バックライトの光量を調節します。 Read すると現在の光量設定値が読み出せます。
/sys/class/backlight/ap310-bl/power	0 or 1	0 を Write することでバックライトを ON します。 1 を Write することでバックライトを OFF します。 Read することで現在のバックライトの ON/OFF 状況が読み出せます。

表 2-7-1-31. AP-3101/3102 の Sysfs のバックライトを制御する項目

sysfs ファイル名	データ	内容
/sys/class/backlight/ap315-bl/brightness	0～255	0～255 の数値を Write することで、バックライトの光量を調節します。 Read すると現在の光量設定値が読み出せます。
/sys/class/backlight/ap315-bl/power	0 or 1	0 を Write することでバックライトを ON します。 1 を Write することでバックライトを OFF します。 Read することで現在のバックライトの ON/OFF 状況が読み出せます。

２－７－２　ブザー

AP-1000/2000/3100/3101/3102 のブザーを制御します。

表 2-7-2. Sysfs のブザーを制御する項目

sysfs ファイル名	データ	内容
/sys/devices/platform/buzz/autobuzz	0 or 1	1 を Write するとタッチパネルがタッチされたときにブザーを鳴らします。 0 を Write するとタッチパネルがタッチされたときにブザーを鳴らしません。 Read することで現在の状態が読み出せます。
/sys/devices/platform/buzz/status	0 or 1	1 を Write するとブザーが ON します。 0 を Write するとブザーが OFF します。 Read することで現在の状態が読み出せます。

２－７－３　タッチパネル

AP-1000/2000/3100/3101/3102 のタッチパネルのタッチ範囲を管理します。

表 2-7-3. Sysfs のタッチパネルを制御する項目

sysfs ファイル名	データ	内容
/sys/devices/platform/asdadc_ts.0/xmax	任意	任意のデータを Write することでタッチパネル X 方向の最大値を設定します。 Read することで現在の設定値が読み出せます。
/sys/devices/platform/asdadc_ts.0/xmin	任意	任意のデータを Write することでタッチパネル X 方向の最小値を設定します。 Read することで現在の設定値が読み出せます。
/sys/devices/platform/asdadc_ts.0/ymax	任意	任意のデータを Write することでタッチパネル Y 方向の最大値を設定します。 Read することで現在の設定値が読み出せます。
/sys/devices/platform/asdadc_ts.0/ymin	任意	任意のデータを Write することでタッチパネル Y 方向の最小値を設定します。 Read することで現在の設定値が読み出せます。

２－７－４　基板情報

AP-1000/2000/3100/3101/3102 の基板情報を管理します。

表 2-7-4. Sysfs の基板情報を制御する項目

sysfs ファイル名	データ	内容
/sys/devices/platform/mainboard/boardversion	0x00XX	Read することで FPGA バージョンが読み出せます。
/sys/devices/platform/mainboard/buildversion	0xXXXXXXXX	Read することでカーネルのビルドバージョンが読み出せます。
/sys/devices/platform/mainboard/machcode	0x00000XXX	Read することでマシンコードが読み出せます。
/sys/devices/platform/mainboard/exb0_asdxreg	“ASD”	拡張ボードが１枚挿入されている状態で Read すると”ASD”という文字列が読み出せます。
/sys/devices/platform/mainboard/exb0_config	0xXX	拡張ボードが１枚挿入されている状態で Read or Write することで、拡張ボードのコンフィグレーションレジスタを読み書きします。書き込むには、exb0_resetenable に 0xAA を書き込む必要があります。拡張ボードのリセットを行うことができます。
/sys/devices/platform/mainboard/exb0_product	0xXX	拡張ボードが１枚挿入されている状態で Read することで、拡張ボードの製品コードを読み出すことができます。
/sys/devices/platform/mainboard/exb0_resetenable	0x00 or 0xAA	拡張ボードが 1 枚挿入されている状態で、0xAA を Write すると、exb0_config に対して書き込みを行うことができます。通常は 0x00 にしておいてください。
/sys/devices/platform/mainboard/exb0_revision	0xXX	拡張ボードが１枚挿入されている状態で Read することで、拡張ボードのリビジョンコードが読み出せます。
/sys/devices/platform/mainboard/exb0_vendor	0xXX	拡張ボードが１枚挿入されている状態で Read することで、拡張ボードのベンダコードが読み出せます。
/sys/devices/platform/mainboard/exb1_asdxreg	“ASD”	拡張ボードが２枚挿入されている状態で Read すると”ASD”という文字列が読み出せます。
/sys/devices/platform/mainboard/exb1_config	0xXX	拡張ボードが２枚挿入されている状態で Read or Write することで、拡張ボードのコンフィグレーションレジスタを読み書きします。書き込むには、exb0_resetenable に 0xAA を書き込む必要があります。拡張ボードのリセットを行うことができます。
/sys/devices/platform/mainboard/exb1_product	0xXX	拡張ボードが２枚挿入されている状態で Read することで、拡張ボードの製品コードを読み出すことができます。
/sys/devices/platform/mainboard/exb1_resetenable	0x00 or 0xAA	拡張ボードが 2 枚挿入されている状態で、0xAA を Write すると、exb0_config に対して書き込みを行うことができます。通常は 0x00 にしておいてください。

/sys/devices/platform/mainboard/exb1_revision	0xXX	拡張ボードが2枚挿入されている状態でReadすることで、拡張ボードのリビジョンコードが読み出せます。
/sys/devices/platform/mainboard/exb1_vendor	0xXX	拡張ボードが2枚挿入されている状態でReadすることで、拡張ボードのベンダコードが読み出せます。

２－８　　データの保護について

Linux では NOR FLash や CF カード、USB メモリ等のストレージに、あるファイルを Write したとき、一端書き込みデータをキャッシュ領域に保存して、Write 処理を完了し、OS のアイドル時間に実際のストレージへ書き込むことで CPU の有効活用を行っています。

そのため、キャッシュ領域にデータがあり、実際のストレージに書き込まれる前に電源を落としたりした場合、そのファイルまたは書き込み途中のセクタが破損する可能性があります。これを防ぐために、sync というコマンドがあります。このコマンドを実行することで、キャッシュ内にたまっているデータを実際のストレージにすべて書き出すことができます。ストレージにファイルを書き込んだ際は電源を落とす前に sync コマンドを実行してください。

また、USB メモリは抜き差しが可能なデバイスであるため取り扱いには注意が必要となります。使用中に USB メモリが抜かれた場合などは、メモリ内のファイルが破損してしまう場合があります。USB メモリにファイルを書き込んだ場合は、sync コマンドなどを使用して書き込んだ内容が確実に USB メモリまで書き込まれるようにしてください。また、USB メモリを抜く場合には、必ず USB メモリをアンマウントしてから抜くようにしてください。

● sync コマンドの使用

Linux でファイル操作を行った場合、ファイルデータがファイルキャッシュとしてシステムメモリに保存され、実際の CF カードなどのデバイスには反映されていないことがあります。CF カードのファイル操作後、変更されたデータが CF カードに確実に反映されるようにするには、sync コマンドを使用してキャッシュと CF カードのデータの同期をとるようにしてください。

CF カードにファイルコピー後、sync コマンドで同期

```
# cp /home/asdusr/FILE.dat /mnt
# sync
```

● USB メモリの書き込み不可/可能の切り替え

USB メモリを書き込み不可状態でマウントし、書き込み不可/可能の切り替えを行います。書き込み不可状態では、ファイルの書き込みを行えませんがファイルの読み出しは行えます。

USB メモリを書き換え不可でマウントします。

```
# mount –o ro /dev/sdb1 /mnt
```

書き換え不可でマウントされている CF カードを書き換え可能にします。

```
# mount –o rw /dev/sdb1 /mnt
```

書き換え可能でマウントされている CF カードを書き換え不可にします。

```
# mount –o ro /dev/sdb1 /mnt
```

第３章　開発環境

本章では、AlgonomixDFB 2 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場合に必要となるのが、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。Linux マシン上で動作するエディタでソースを書き、gcc でコンパイル後に生成され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パソコン環境上で行うことができます。このような開発方式をセルフ開発といいます。

Algo Smar Panel では、ルネサステクノロジ社製の SuperH RISC engine SH4 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パソコン上で行い、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 Panel 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。

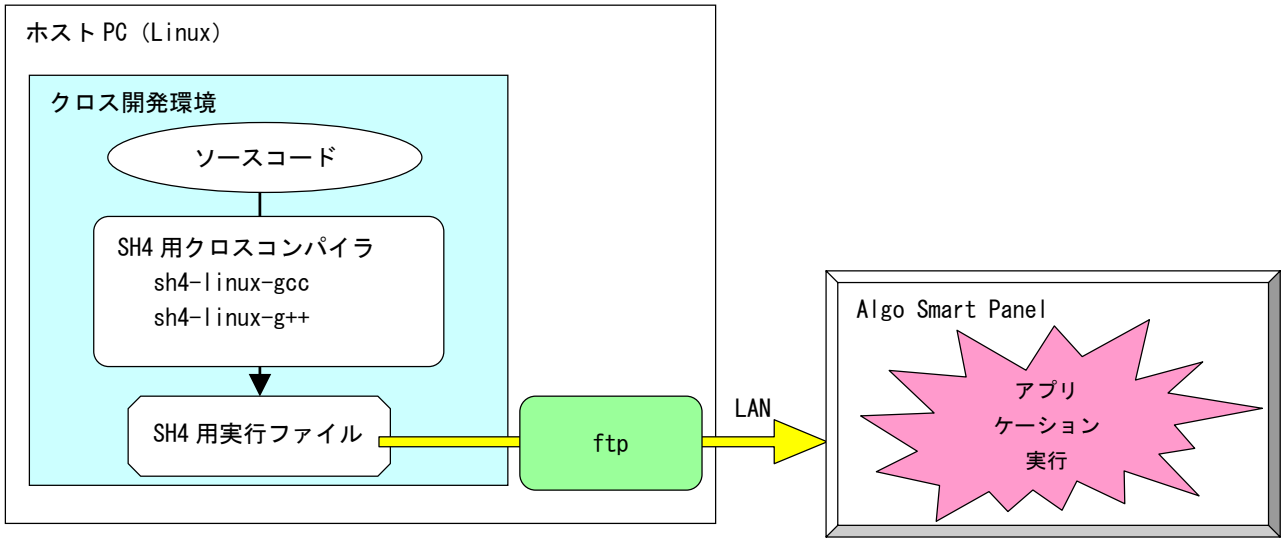


図 3-1-1. クロス開発方式イメージ図

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、順次インストールを行います。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必要となります。

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

AlgonomixDFB 2 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 Panel 用のアプリケーションを開発することが可能です。

３－２　　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。
AlgonomixDFB 2 用開発環境に組み込まれている WideStudio/MWT は V3.97-4 をベースに Algo Smart Panel 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。

３－２－１　　WideStudioの起動

のアイコンをクリックすると WideStudio が起動します。

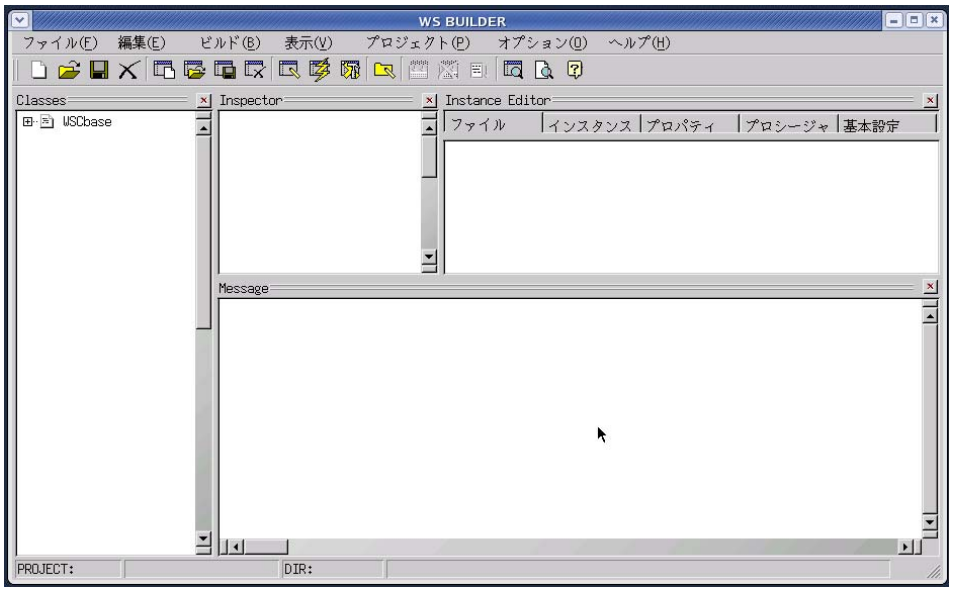


図 3-2-1-1. WideStudio メイン画面

３－２－２　　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。

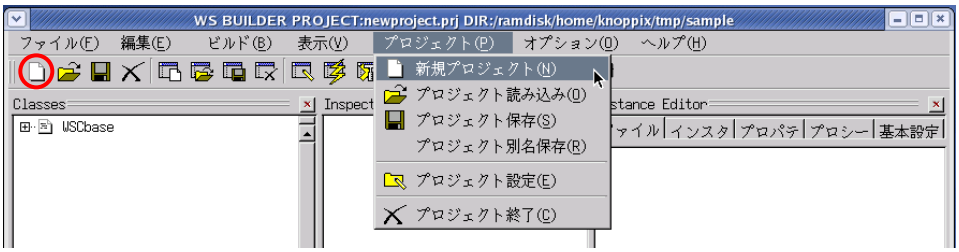


図 3-2-2-1. 新規プロジェクト作成


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 3-2-2-2 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。

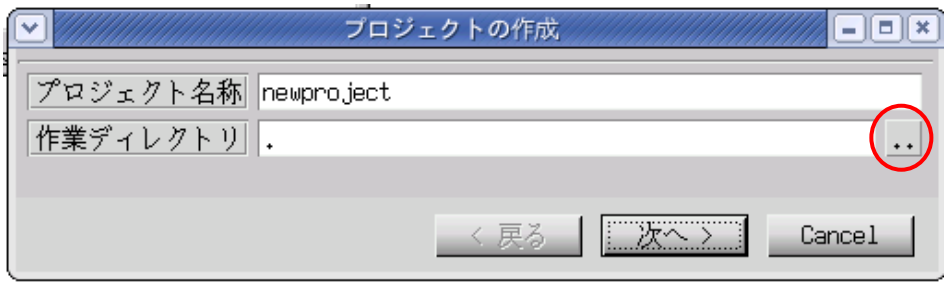


図 3-2-2-2. プロジェクト作成画面

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

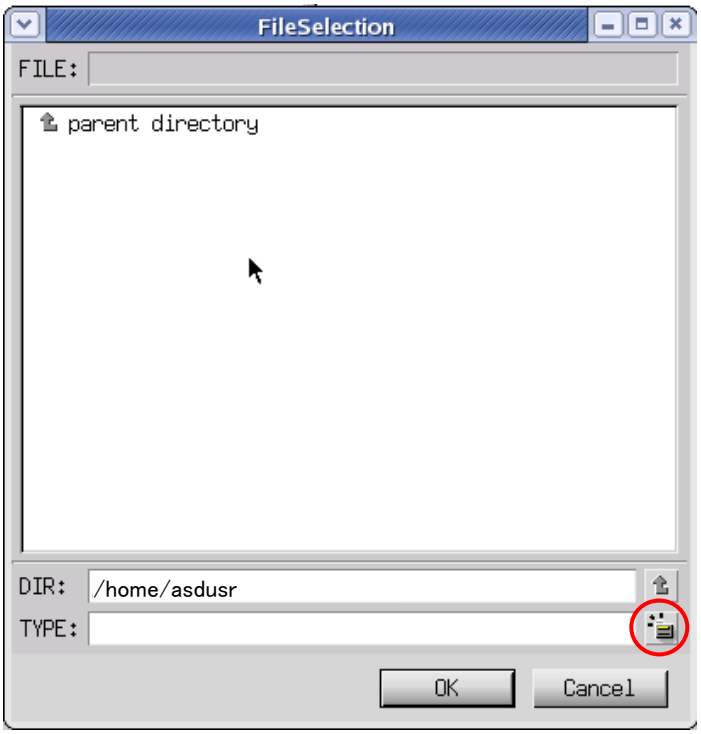


図 3-2-2-3. ファイル選択画面

DIR に「/home/asdusr」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。

※　文字入力はマウスカーソルを入力ダイヤログ上に持っていった上で行ってください。

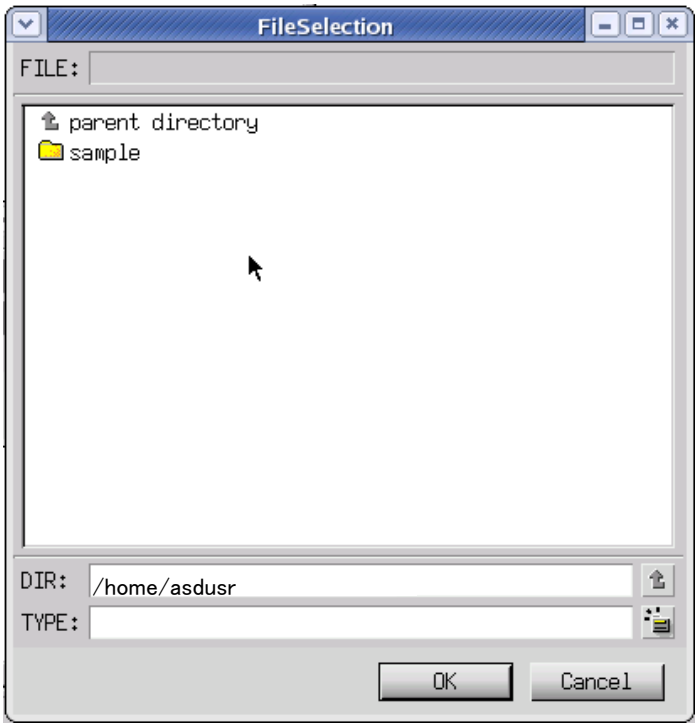


図 3-2-2-4. sample ディレクトリの作成

sample ディレクトリをダブルクリックし、DIR が「/home/asdusr/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。

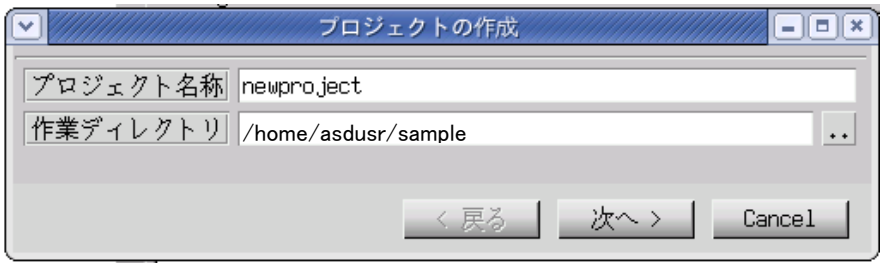


図 3-2-2-5. プロジェクト名と作業ディレクトリの設定完了

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。

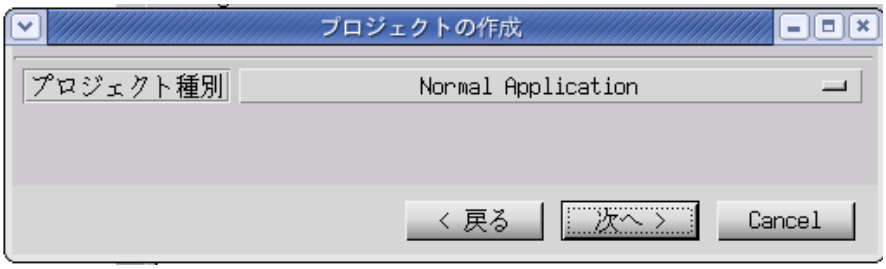


図 3-2-2-6. プロジェクト種別の選択

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。

※　Algo Smart Panel 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

※　言語種別はC/C++のみサポートしています。

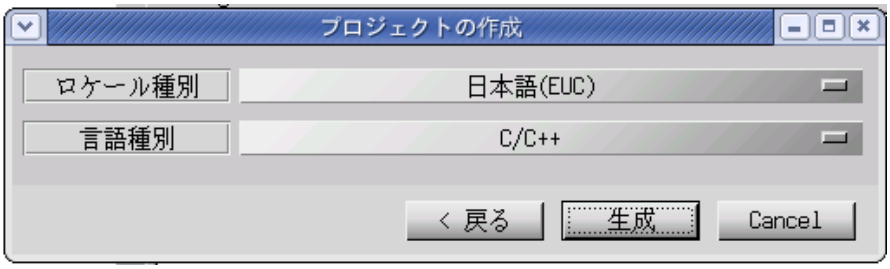


図 3-2-2-7.　ロケール種別と言語種別を選択

以上でプロジェクトの作成は完了です。

３－２－３　　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。

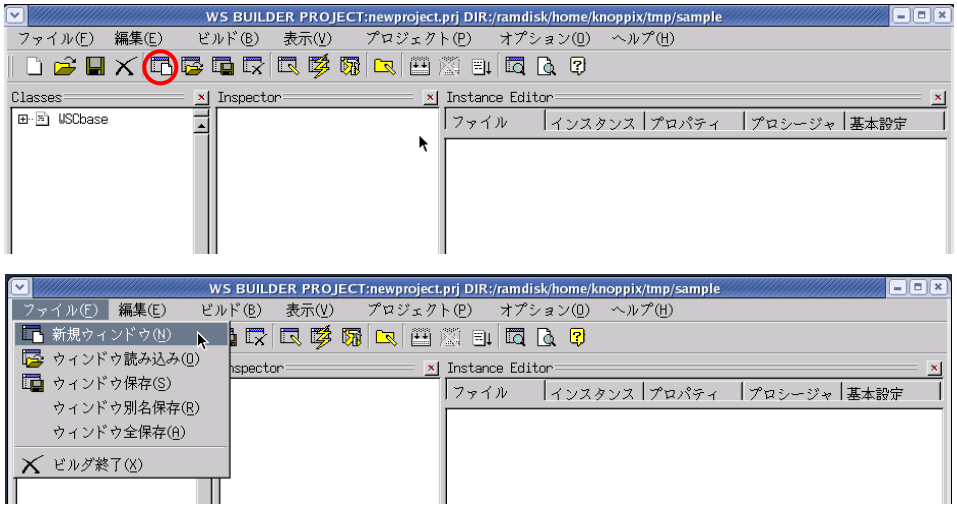


図 3-2-3-1.　新規ウィンドウ作成

 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 3-2-3-2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。

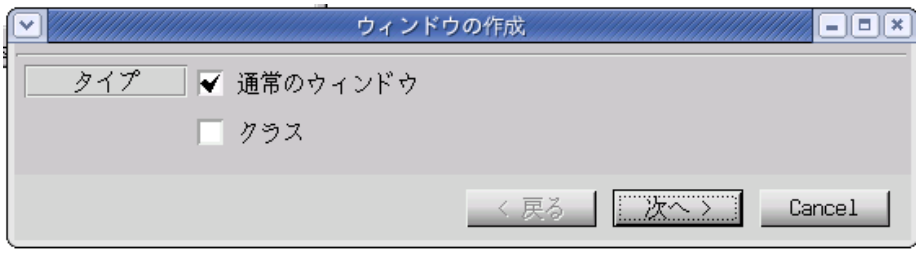


図 3-2-3-2.　アプリケーションウィンドウタイプ選択

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でいいと思います。設定を変更せずに「次へ」をクリックします。

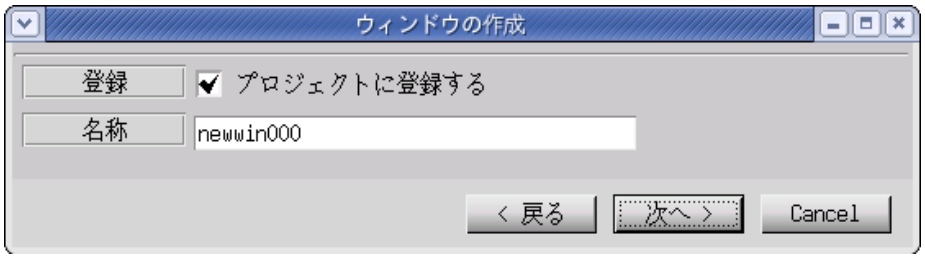


図 3-2-3-3. アプリケーションウィンドウ名称設定

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。

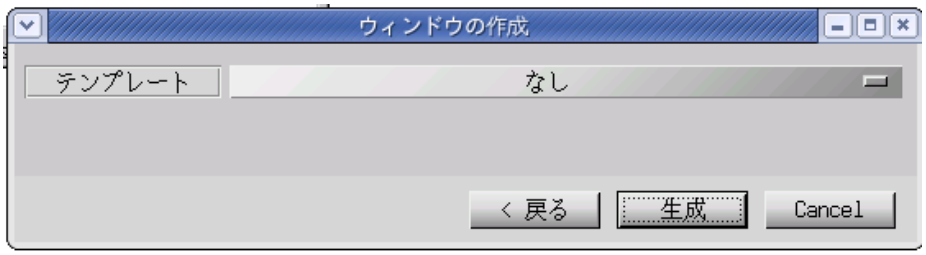


図 3-2-3-4. テンプレートの選択

これで、アプリケーションウィンドウが作成されたと思います。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 3-2-3-1 を参考にプロパティを変更してください。

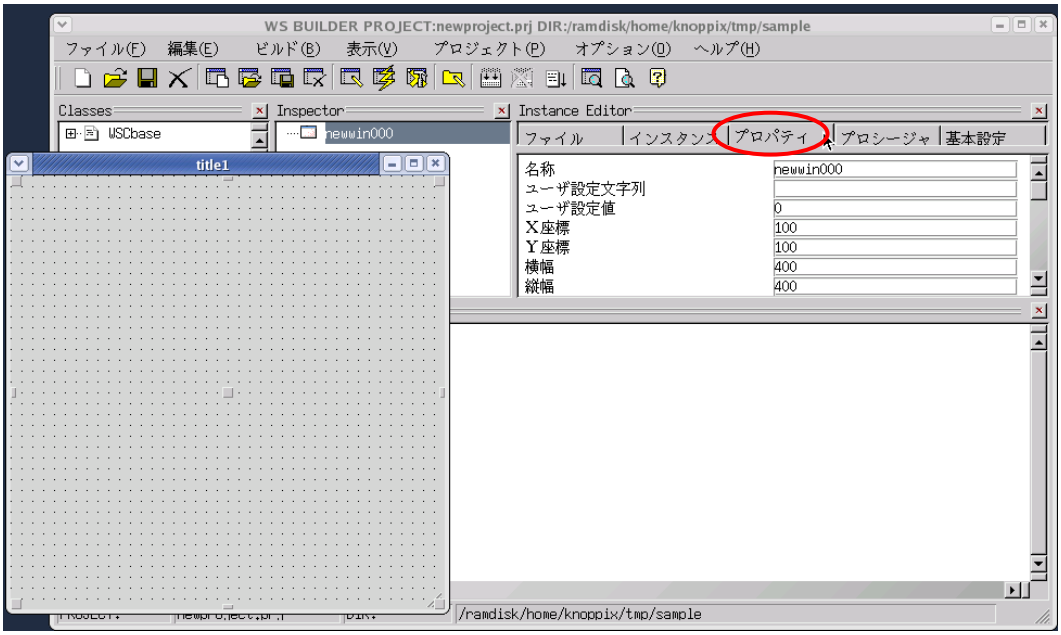


図 3-2-3-5. アプリケーションウィンドウの生成

表 3-2-3-1. 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。

３－２－４　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。

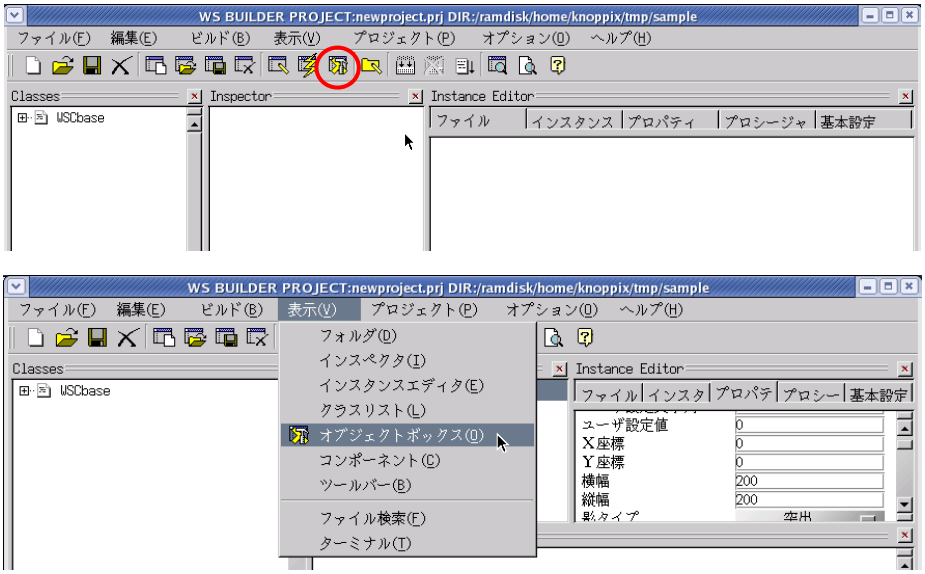


図 3-2-4-1. オブジェクトボックスの表示

 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。

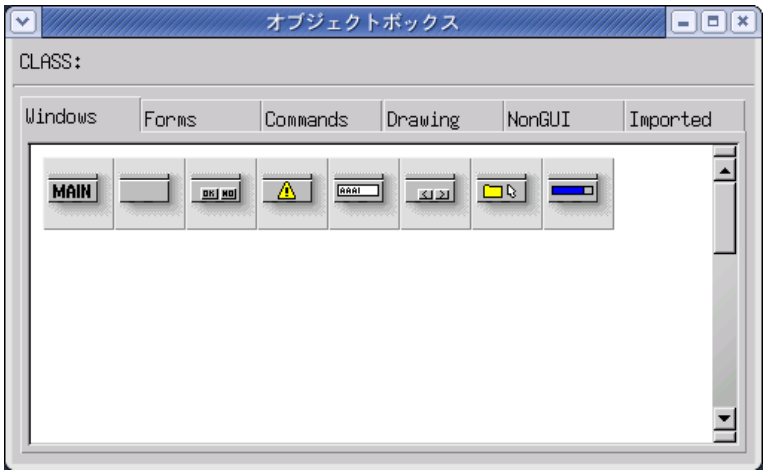


図 3-2-4-2. オブジェクトボックス（Windows タブ）

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 3-2-4-3. ボタンの配置

図 3-2-4-4 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。

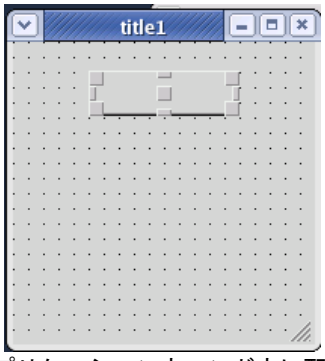


図 3-2-4-4. アプリケーションウィンドウに配置されたボタン

ここでは、例としてボタンオブジェクトを貼り付けただけのテストサンプルを作成します。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。

※　Algo Smart Panel 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。

※　AlgonomixDFB 2 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 3-2-4-1 を参考に値を変更してください。

表 3-2-4-1. 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

３－２－５　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 3-2-5-1 のような画面になります。

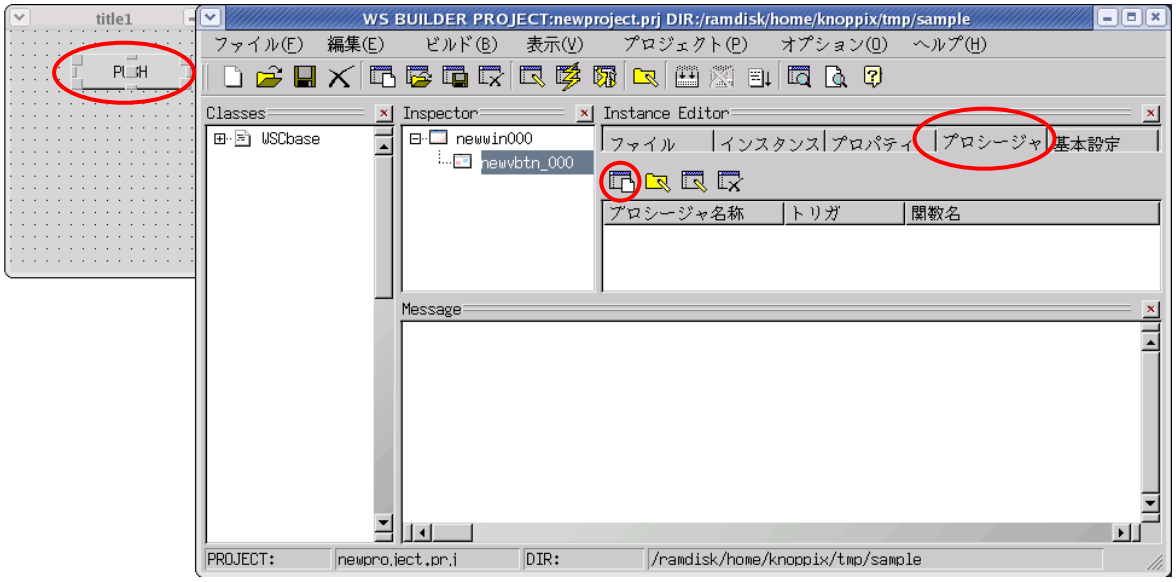


図 3-2-5-1. イベントプロシージャの作成

 をクリックすると、図 3-2-5-2 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。

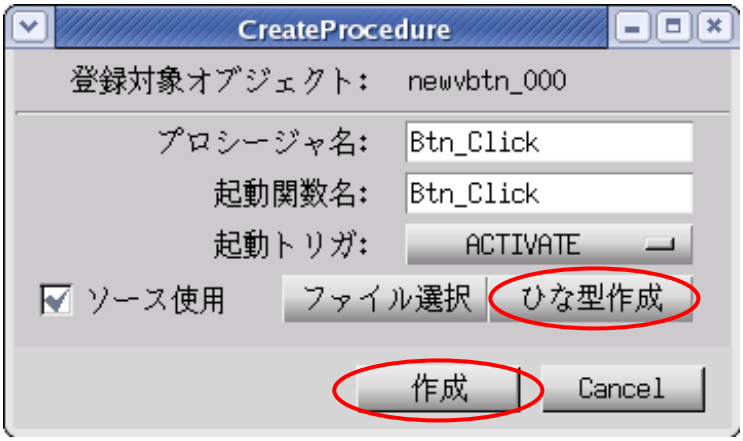


図 3-2-5-2. イベントプロシージャ作成ダイアログ

「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 3-2-5-3 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。

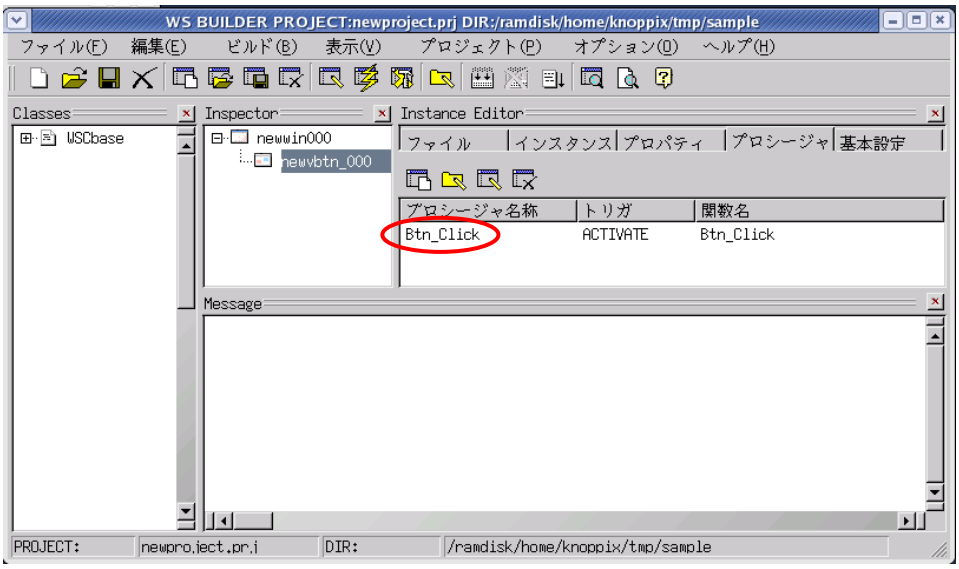


図 3-2-5-3. 作成されたイベントプロシージャ

３－２－６　イベントプロシージャの編集

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 3-2-6-1 のようになっています。

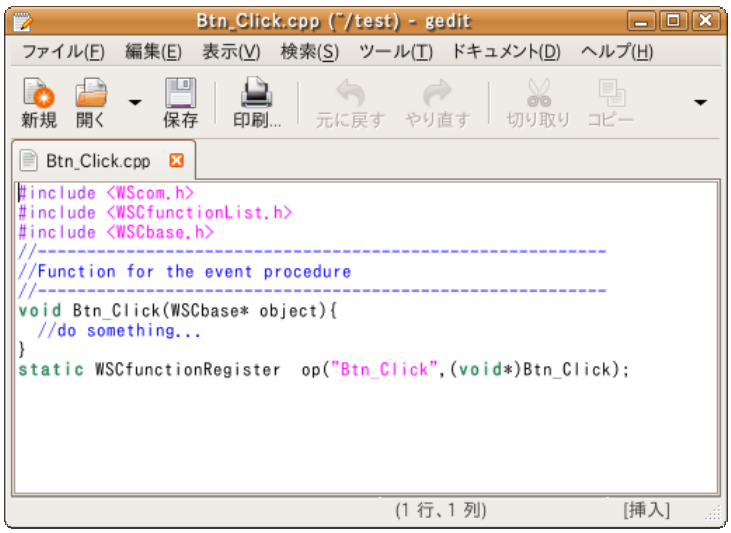


図 3-2-6-1. イベントプロシージャ初期ソースコードをエディタで開いた所

リスト 3-2-6-1 のようにコードを変更します。

リスト 3-2-6-1. 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。

３－２－７　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。

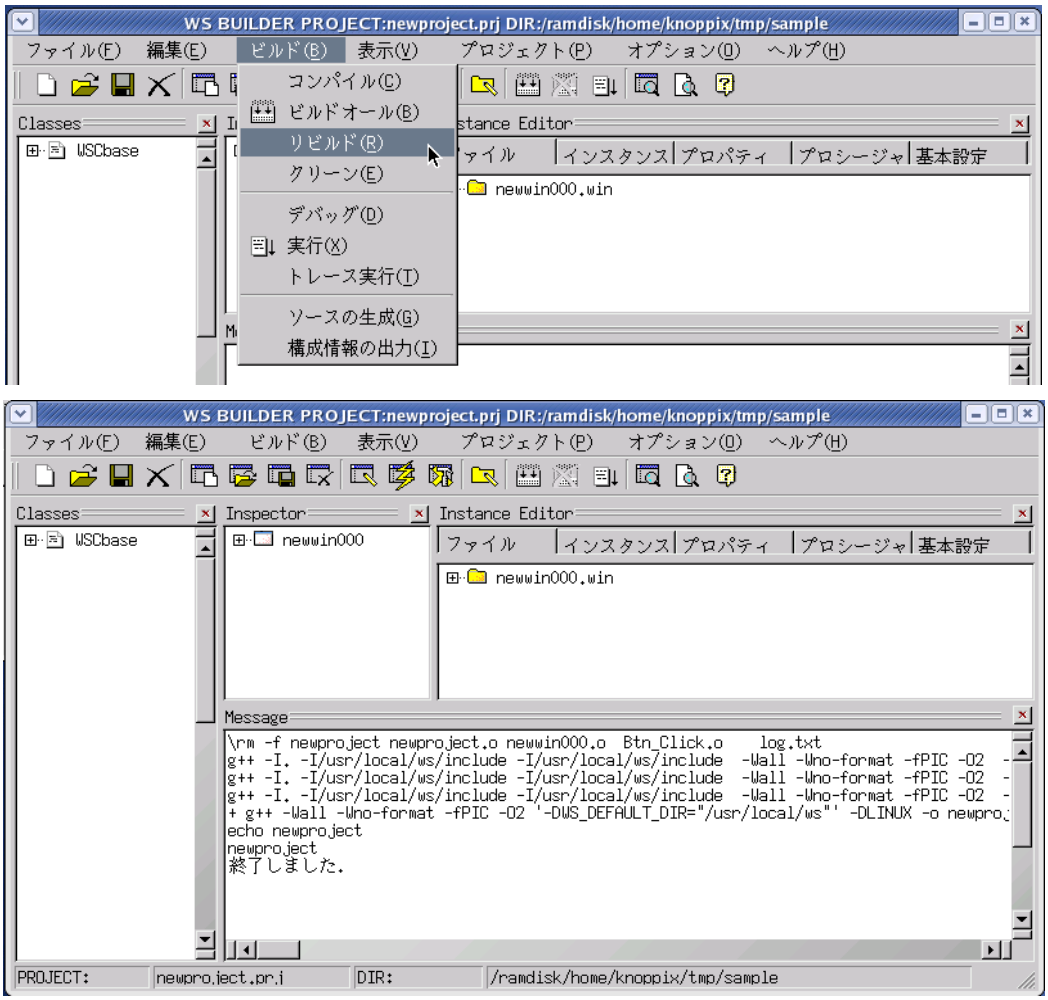


図 3-2-7-1. サンプルプロジェクトのコンパイル

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 Ubuntu 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。

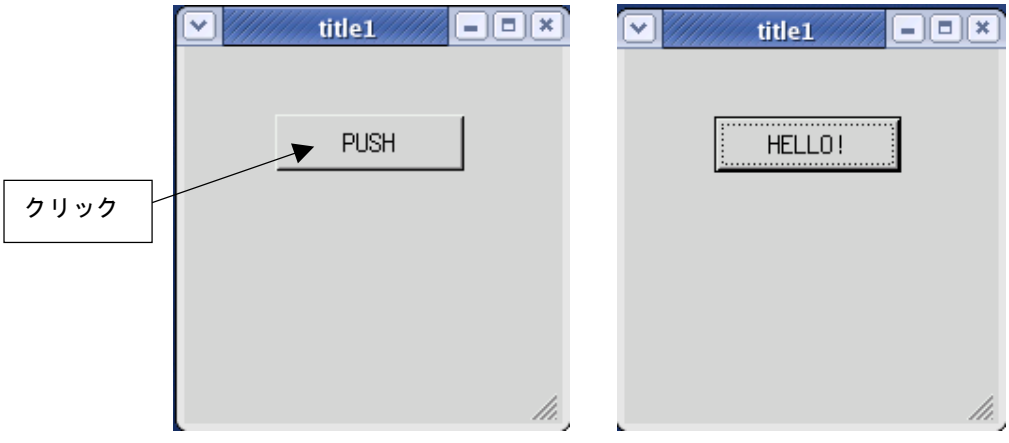


図 3-2-7-2. プログラム実行画面

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。

３－２－８　クロスコンパイル

Algo Smart Panel 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 3-2-8-1 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が、設定を変更していない場合は「Native」となっています。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 3-2-8-2 にターゲット一覧を示します。これは「Native」の部分をクリックすることで表示されます。

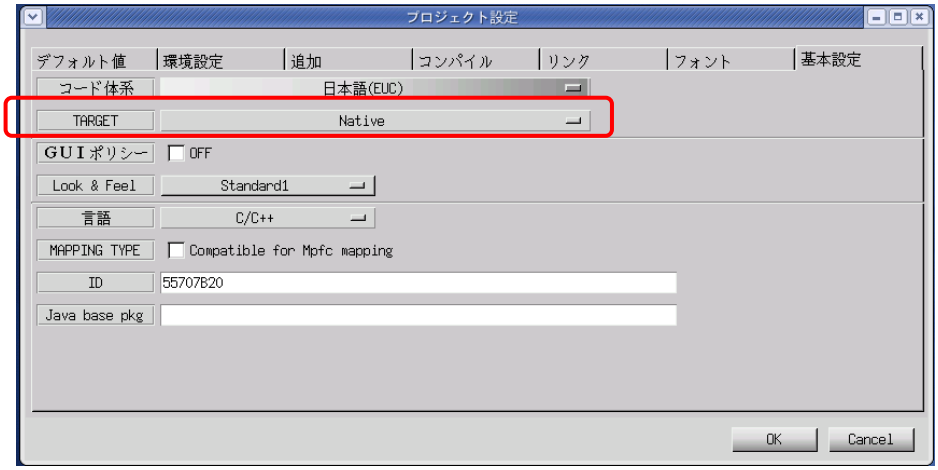


図 3-2-8-1. プロジェクト設定画面（基本設定タブ）

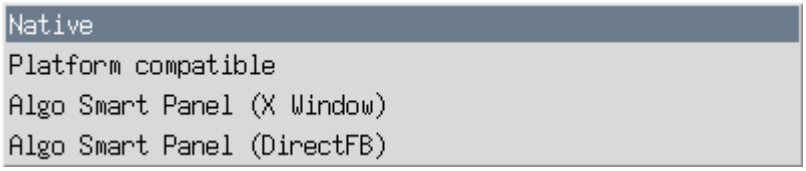


図 3-2-8-2. ターゲット一覧

AlgonomixDFB 2 で選択できるターゲットは表 3-2-8-1 に書かれている 3 種類のみです。
Platform compatible は現在使用しません。

表 3-2-8-1. AlgonomixDFB 2 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 Panel (X Window)	Algo Smart Panel 上の X Windows 上で動作させる場合の設定
Algo Smart Panel (DirectFB)	Algo Smart Panel 上の DirectFB 上で動作させる場合の設定

※　これら以外のターゲットではコンパイルできません
※　GUI に DirectFB を使用している場合は「TARGET」を「Algo Smart Panel (DirectFB)」にして「OK」ボタンをクリックしてください。GUI に DirectFB を使用している場合に「TARGET」を「Algo Smart Panel (X Window)」にしてコンパイルを行った場合、プロジェクトは正常に動作しません。

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 3-2-7-1 のコンパイルログと図 3-2-8-3 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++となっていますが、これがクロスコンパイルです。

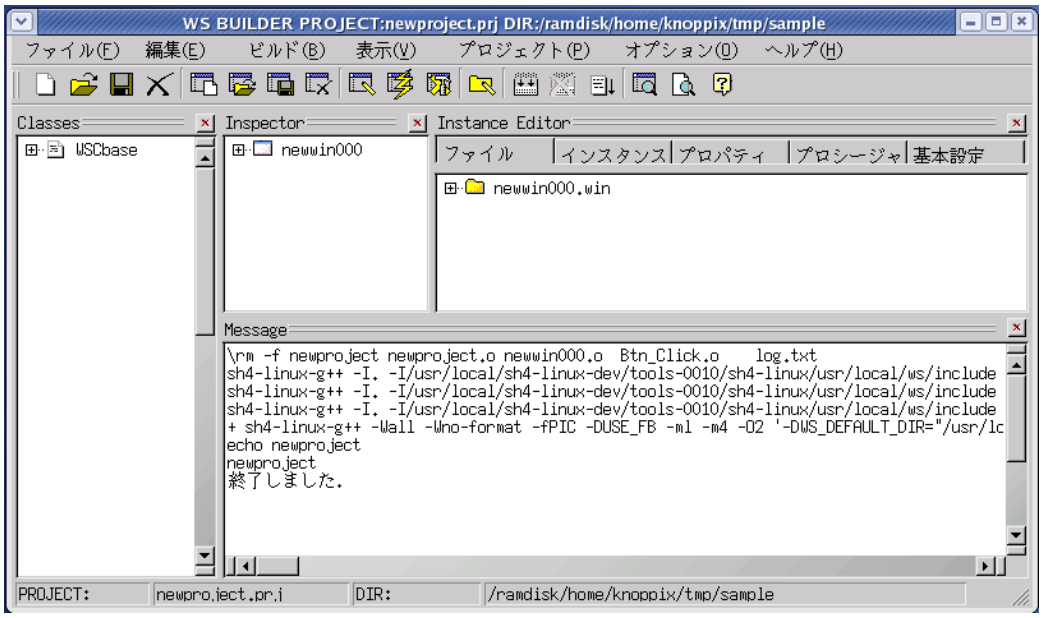


図 3-2-8-3. サンプルプロジェクトのクロスコンパイル

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いと思います。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。

３－２－９　ファイルの転送

作成した実行プログラムを Algo Smart Panel 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のデータを Algo Smart Panel に転送するには表 3-2-9-1 の方法があります。

表 3-2-9-1. Algo Smart Panel への転送方法

方法	AP-1000	AP-2000	AP-3100	AP-3101	AP-3102
USB メモリを使って転送	○	○	○	○	○
LAN 経由で ftp 転送	×※	○	○	○	○

※　Algo Smart Panel 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 メモリでの転送

①パソコンに USB メモリを挿入し、転送するデータ（サンプルプログラム）を USB メモリにコピーします。

②クロス LAN ケーブルで PC と Algo Smart Panel を接続します。

例）Algo Smart Panel の IP 設定：「192.168.0.1」

③デスクトップ上の「アプリケーション」ツールバーから「アクセサリ」→「端末」をクリックするとコンソールが立ち上がります。telnet を使用し Algo Smart Panel 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
```

④転送するデータが保存された USB メモリを Algo Smart Panel に挿入します。

⑤次に、USB メモリのマウントを行います。

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

⑥転送するデータを、Algo Smart Panel の「/home/asdusr/」にコピーします。

例）転送するデータ　　：newproject

　　転送先ディレクトリ：/home/asdusr

以下のコマンドを入力することで、Algo Smart Panel 上に転送するデータが保存されます。

```
# cp /mnt/newproject /home/asdusr
```

⑦マウントを解除して USB メモリを抜きます。

```
# umount /mnt
```

以上で USB メモリから Algo Smart Panel へのコピーが完了しました。

⑧Algo Smart Panel 上で起動している「ASD Config」を閉じます。

⑨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。

●LAN 経由で ftp 転送

①クロス LAN ケーブルで PC と Algo Smart Panel を接続します。

例）Algo Smart Panel の IP 設定：「192.168.0.1」

②デスクトップ上の「アプリケーション」ツールバーから「アクセサリ」→「端末」をクリックするとコンソールが立ち上がります。telnet を使い、Algo Smart Panel 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
```

③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
```

④「アプリケーション」ツールバーの「アクセサリ」→「端末」から、別にコンソールを立ち上げます。
以下のコマンドで Algo Smart Panel 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:asdusr): 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 /home/asdusr/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>
```

⑤④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
```

⑥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。

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。

３－２－１０ 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。

●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。

■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 Panel の/etc/xunicoderc にて詳細なフォントを定義

■T-Engine 系の設定

FONT0:サイズ フォント ID

[例] FONT0:10 60c6
　　FONT1:12 60c6

■DirectFB 系 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

■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、「pr」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。

※　Algo Smart Panel 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。

●DirectFB の場合

Linux/DirectFB 版を使用するには、wsfonts ファイルによるフォントの設定を行う必要があります。現在のところ pcf フォント、bdf フォントが使用可能です。

・wsfonts ファイル配置場所：/etc/ wsfonts

・pcf フォントを使用する場合（/usr/font に配置した場合）

/usr/font/7x14.pcf

/usr/font/k14.pcf

wsfonts の内容：

default、PCF、/usr/font/7x14.pcf、ISO8859_1、/usr/font/k14.pcf、JIS0208

３－３　　DDDについて

AlgonomixDFB 2 開発環境では、Wide Studio 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 3-3-1 のようなプロジェクト設定画面が表示されます。

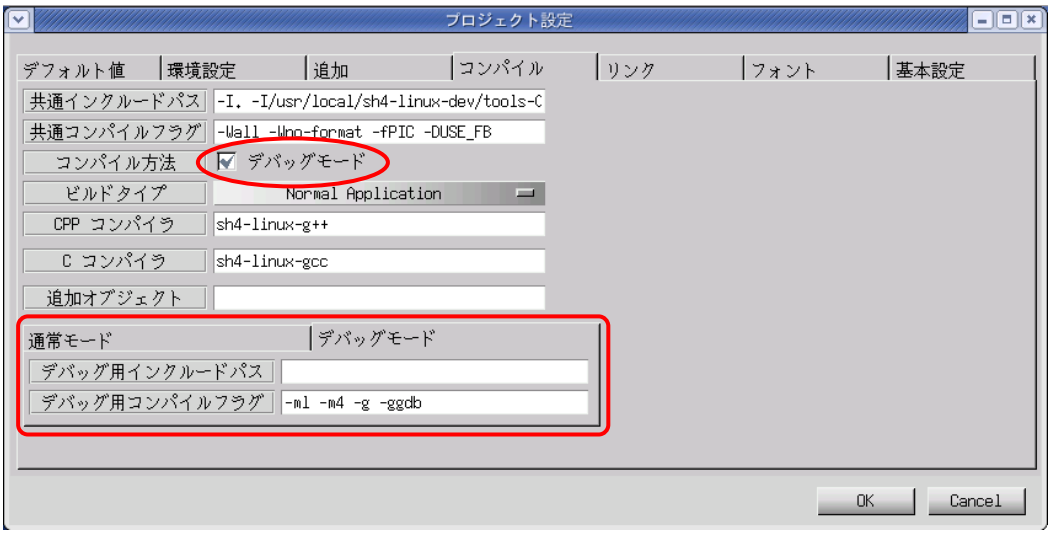


図 3-3-1. プロジェクト設定画面（コンパイルタブ）

「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。

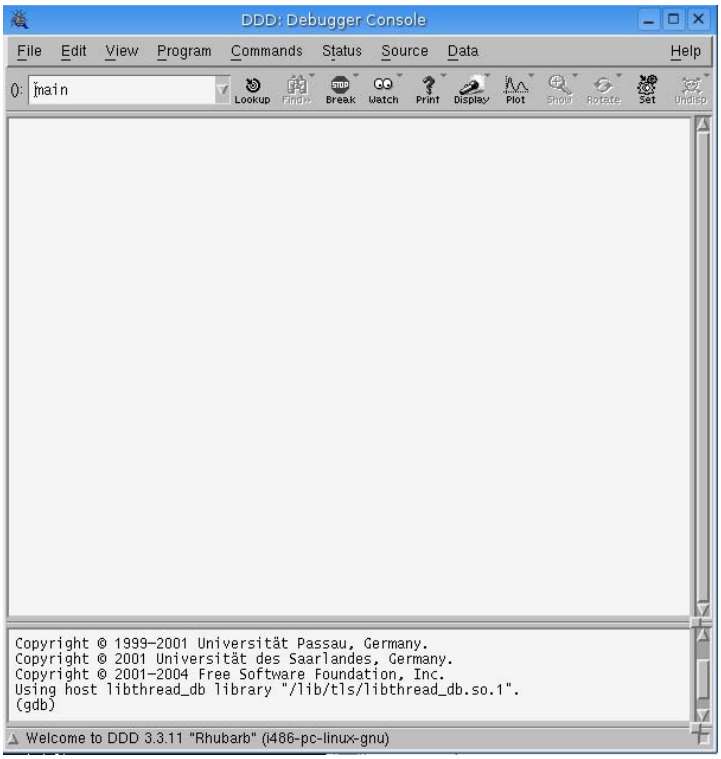


図 3-3-2. DDD 起動画面

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。

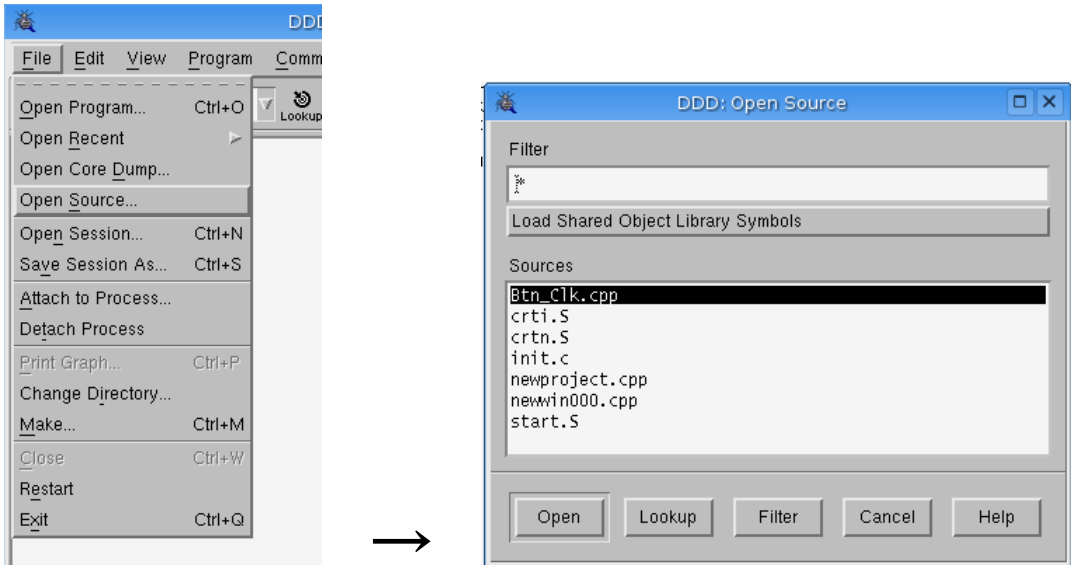


図 3-3-3. ソースの選択

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。

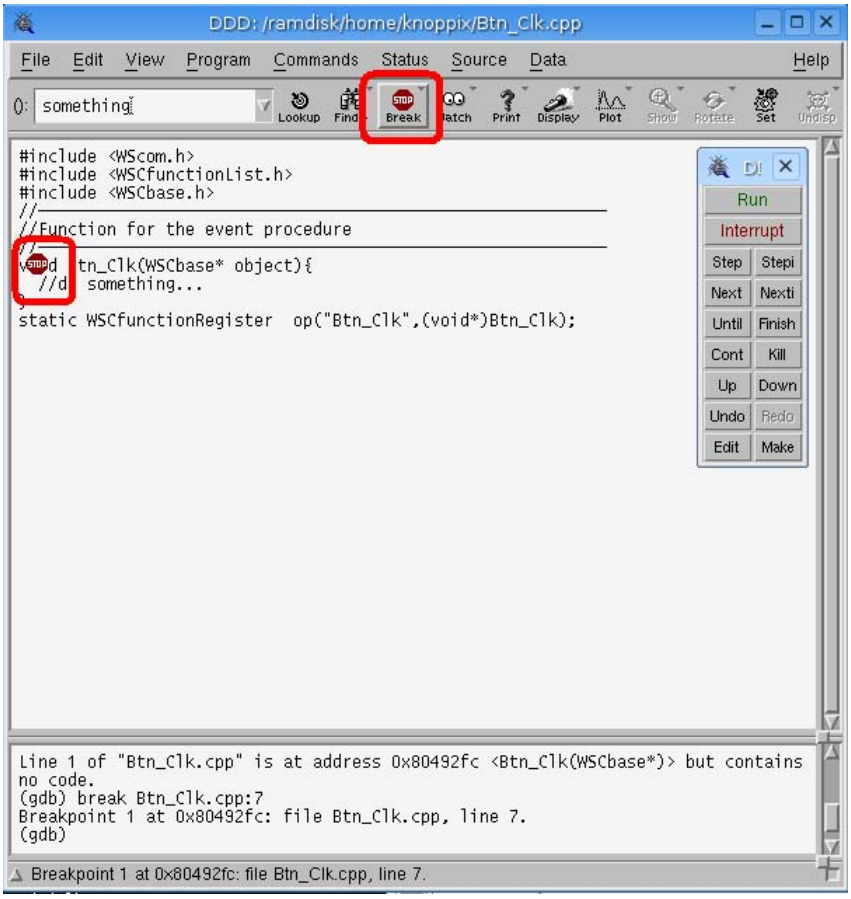


図 3-3-4.　ブレーク設定

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。

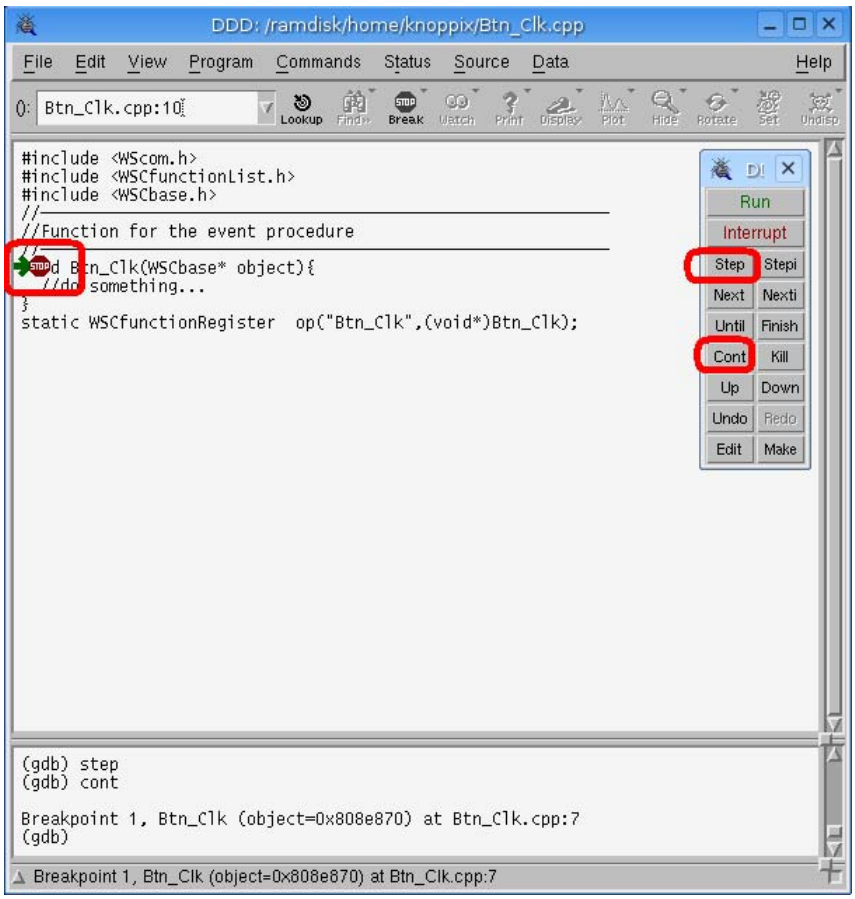


図 3-3-5. ブレークによる実行停止

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、break point 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 break point 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 breakpoint 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 breakpoint を再開するには、Enable Breakpoint を選択します。

３－４　　GDBによるデバッグ方法

Algo Smart Panel の AlgonomixDFB 2 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 Panel 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 Panel 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。
以下より『3－2 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 3-4-1 のように変更します。

リスト 3-4-1. 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、またはメニューの「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。

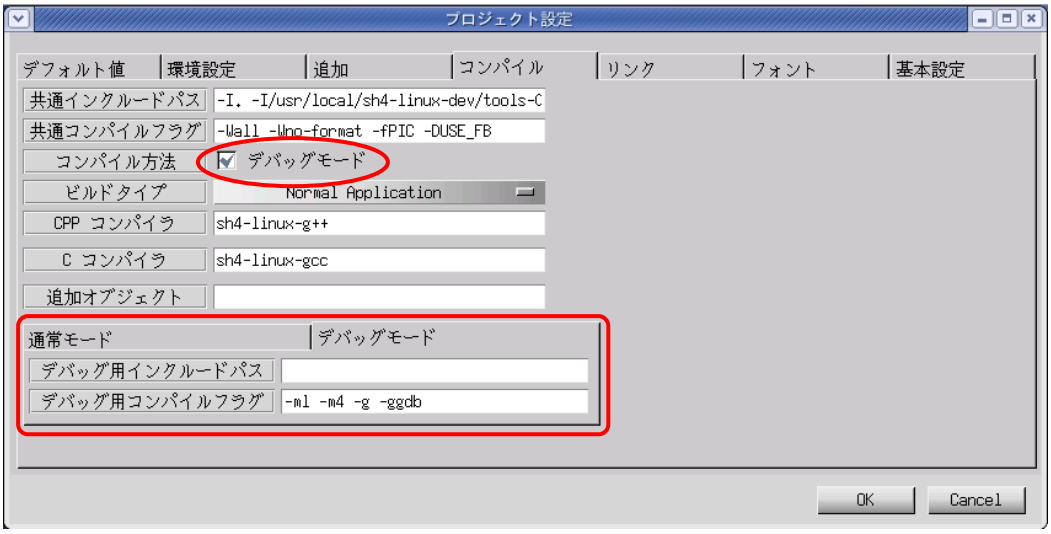


図 3-4-1. プロジェクト設定画面（コンパイルタブ）

「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」)。

②GDB サーバの起動

Algo Smart Panel で「gdbserver」を起動します。まず、「3-2-9　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 Panel 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
```

③GDB の起動

デスクトップ上の「アプリケーション」ツールバーから「アクセサリ」→「端末」をクリックし、別の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 /home/asdusr/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-0040/bin:$PATH
```

GDB を起動し、Algo Smart Panel 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 Panel 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 3-4-1 に示します。

表 3-4-1. 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。 **※　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。

３－５　　シリアルコンソールについて

Windows のハイパーターミナルを用いて Algo Smart Panel とパソコンをシリアルコンソール接続する方法を以下に示します。

①シリアルケーブルを Algo Smart Panel に接続します。

②パソコン上で「スタート」→「アクセサリ」→「通信」→「ハイパーターミナル」を選択し、プログラムを実行します。

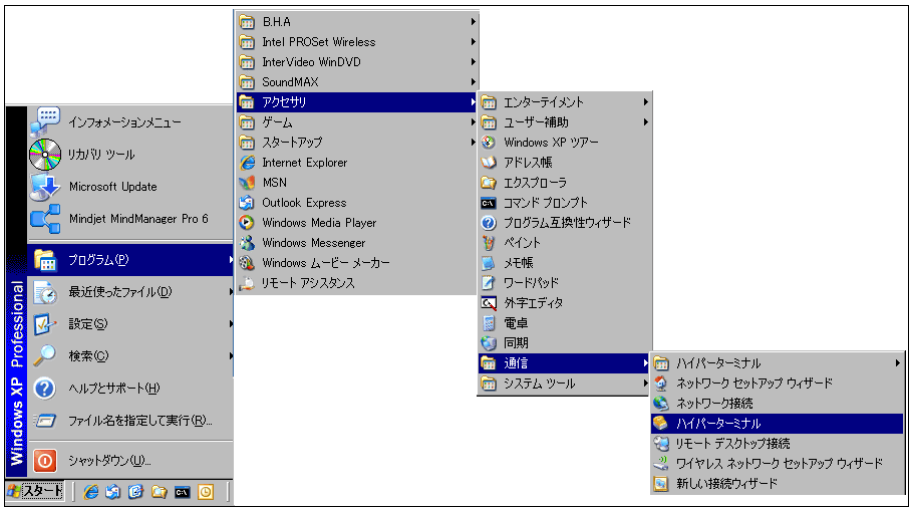


図 3-5-1. ハイパーターミナル起動

③「接続の設定」ウィンドウが表示されますので、名前を入力し「OK」ボタンを押下します。
図 3-5-2 では名前を「asd」と入力します。

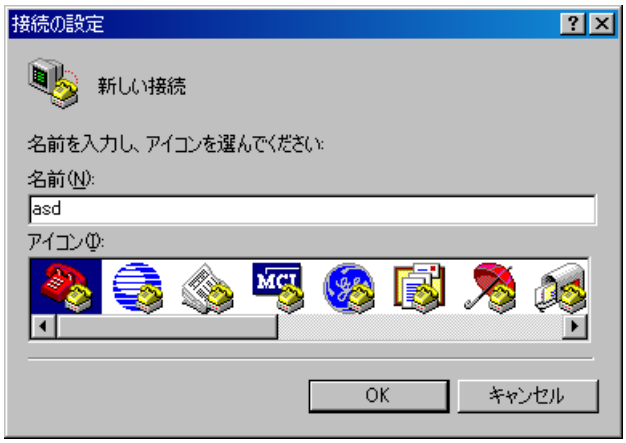


図 3-5-2. 接続の設定(名前入力)

④接続方法等の情報を入力します。
「接続方法」のプルダウンメニューより接続するシリアルポートを選択し「OK」ボタンを押下します。
ここでは、COM1 を選択します。

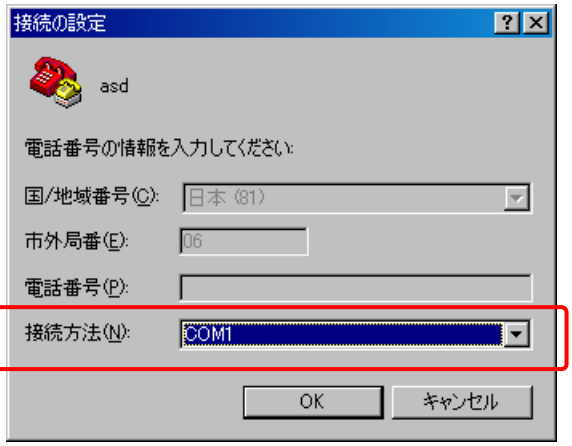


図 3-5-3. 接続の設定(接続方法)

⑤シリアルポートの設定を行います。下記の通り設定後、「OK」ボタンを押下してください。

ビット/秒(B)	: 38400
データビット(D)	: 8
パリティ(P)	: なし
ストップビット(S)	: 1
フロー制御(F)	: なし

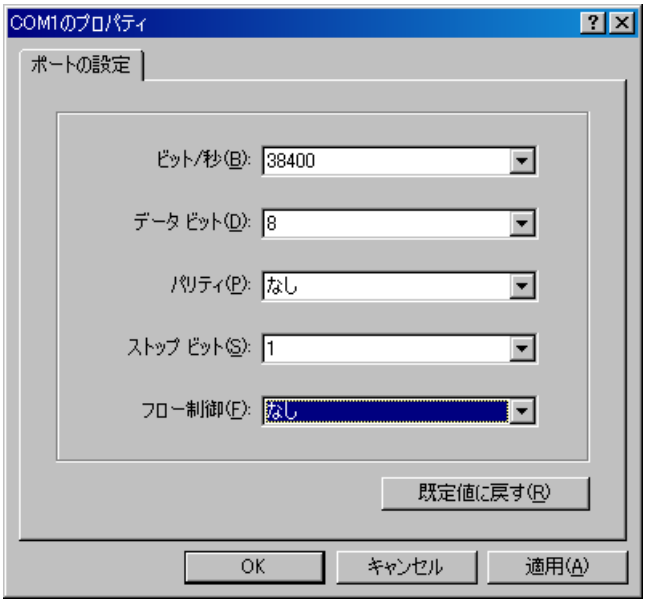


図 3-5-4. シリアルポートのプロパティ

⑥ハイパーターミナルの設定は完了です。これでシリアルコンソールを使用することができます。

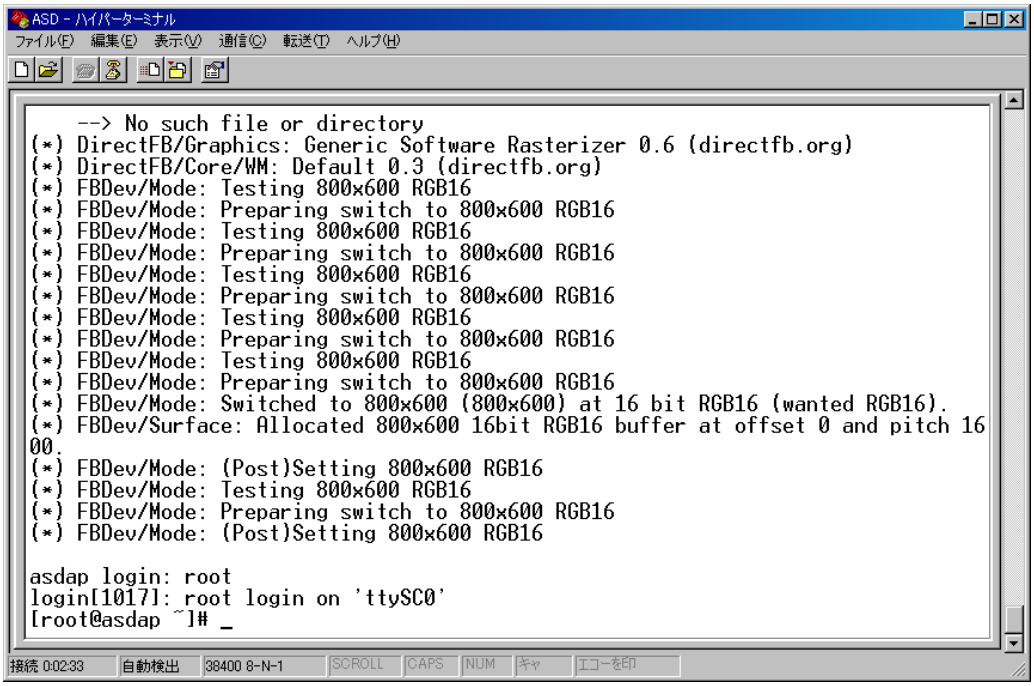


図 3-5-5. ハイパーターミナル設定完了

３－６　　画面なしアプリケーション開発について

画面なしアプリケーション開発について説明します。
画面なしアプリケーション開発として、Eclipse（CDT(C/C++ Development Tooling)プラグインを追加）を紹介しますが、Eclipse を使用しなくても独自で makefile を作成して gcc コンパイラを使用してアプリケーションを開発することも可能です。
Eclipse の詳細な使用方法に関しては、インターネットまたは書籍を参照下さい。

３－６－１　Eclipseの起動

開発環境が入った Linux パソコンのディスクトップ上で「Eclipse」のアイコンをクリックすると Eclipse が起動します。

図 3-6-1-1. Eclipse の起動

Eclipse が起動すると、プロジェクトを保存するワークスペースのフォルダ指定のウィンドウが開きます。
ここでは、「/home/asdusr/workspace」を指定します。

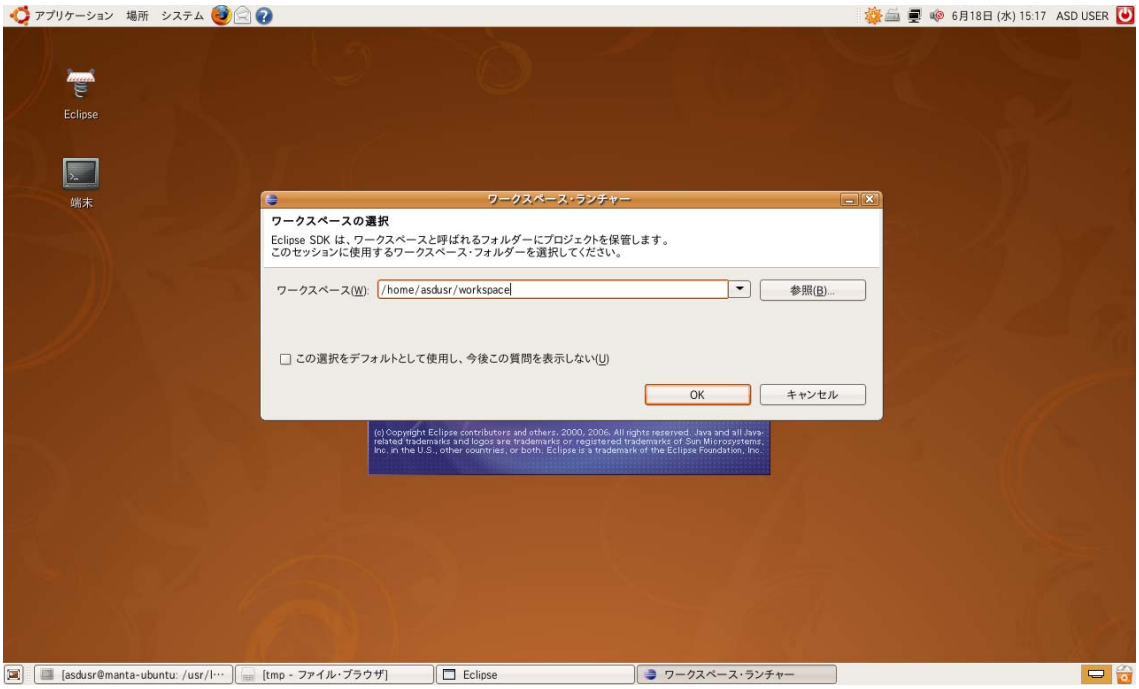


図 3-6-1-2.　ワークスペースフォルダ指定画面

Eclipse が起動後、「ようこそ」の×印をクリックし、図 3-6-1-3 の画面を終了させます。

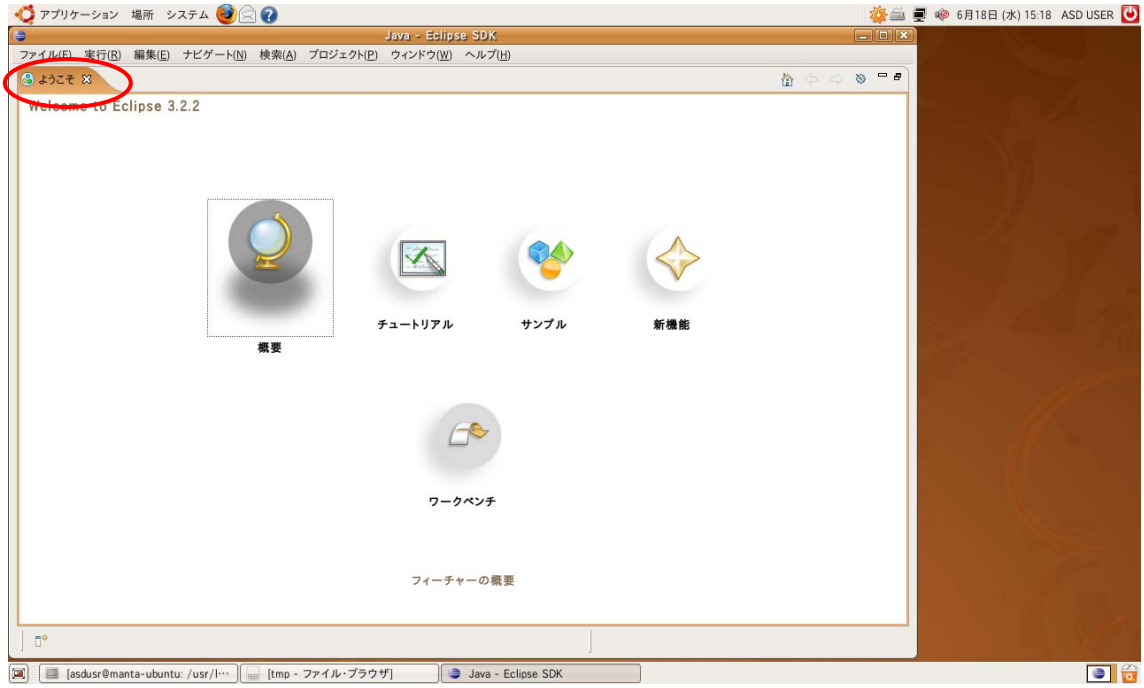


図 3-6-1-3.　ようこそ画面

図 3-6-1-4 のメイン画面が表示されると、アプリケーション作成できます。

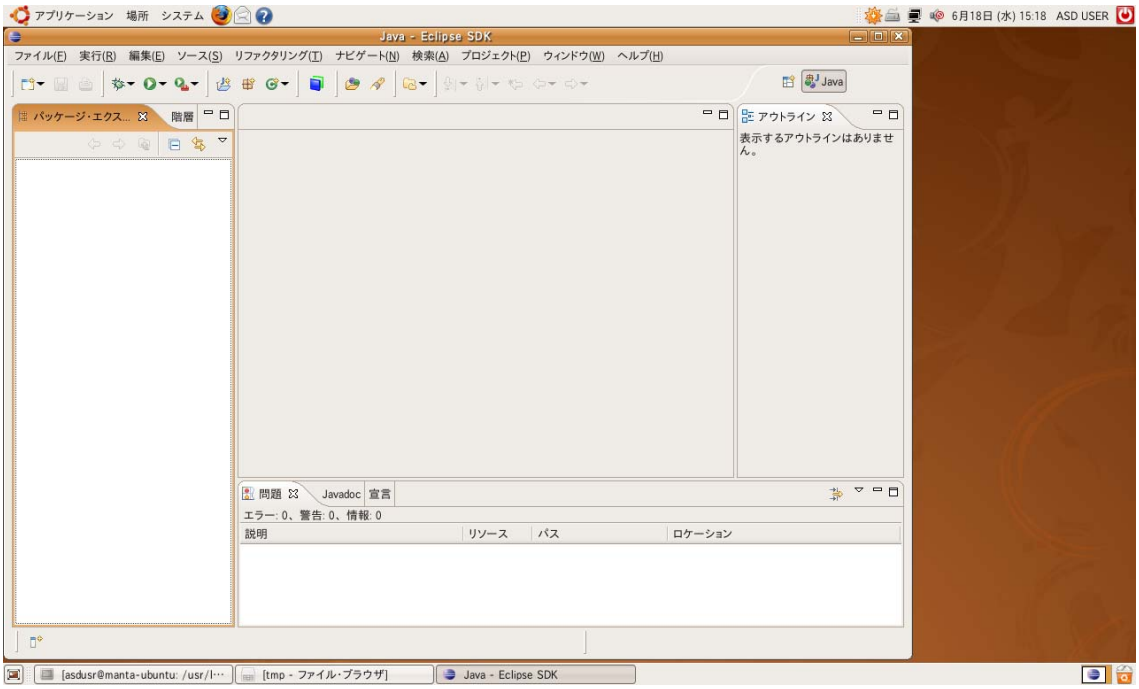


図 3-6-1-4. Eclipse メイン画面

３－６－２　　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。

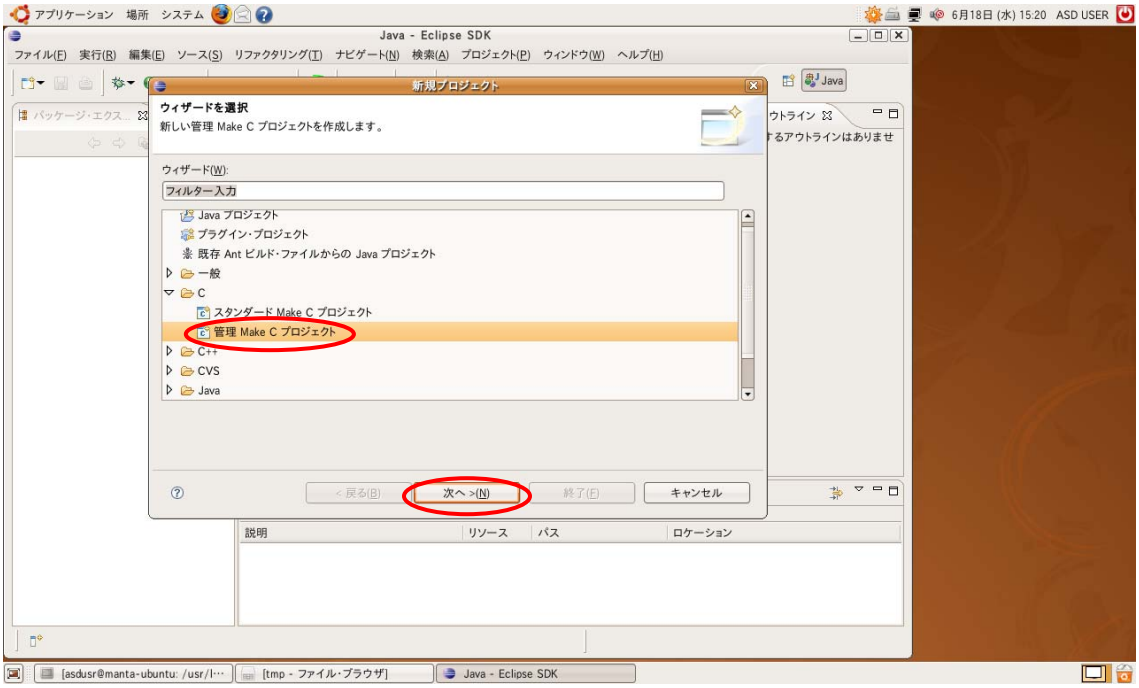


図 3-6-2-1. プロジェクト作成画面

画面左上メニューの「ファイル(F)」→「新規(N)」→「プロジェクト(R)」を選択します。
「新規プロジェクト」のウィンドウが表示されますので、「C」→「管理 Make C プロジェクト」を選択し、「次へ」をクリックしてください。

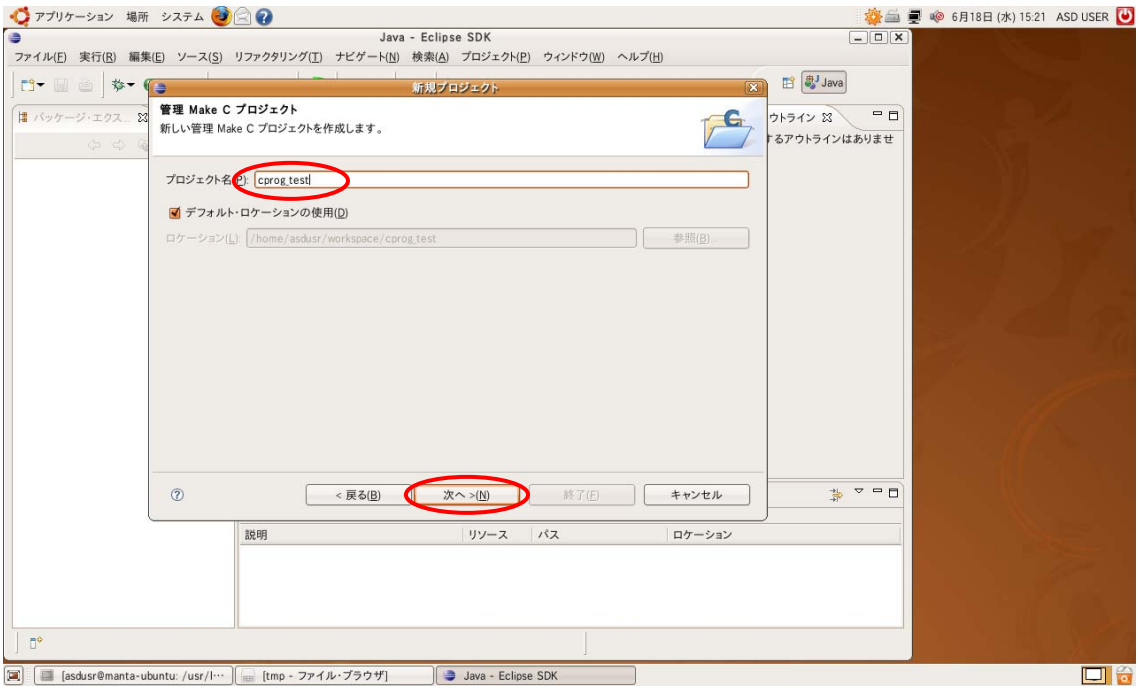


図 3-6-2-2.　プロジェクト名入力画面

図 3-6-2-2 のプロジェクト名入力画面が表示されますので、プロジェクト名を入力します。
ここでは、例として「cproc_test」と入力しています。
入力完了後、「次へ」をクリックしてください。

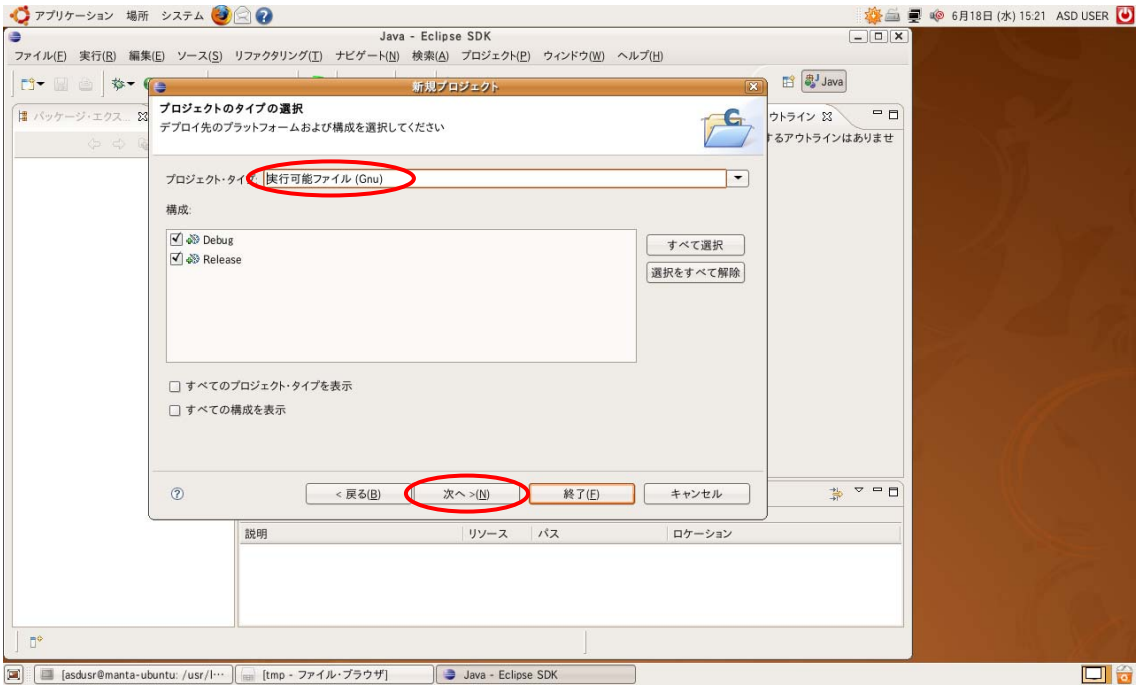


図 3-6-2-3.　プロジェクトタイプ選択画面

図 3-6-2-3 のプロジェクトタイプ選択画面が表示されますので、プロジェクト・タイプが「実行可能ファイル（Gnu)」であることを確認し、「次へ」をクリックしてください。

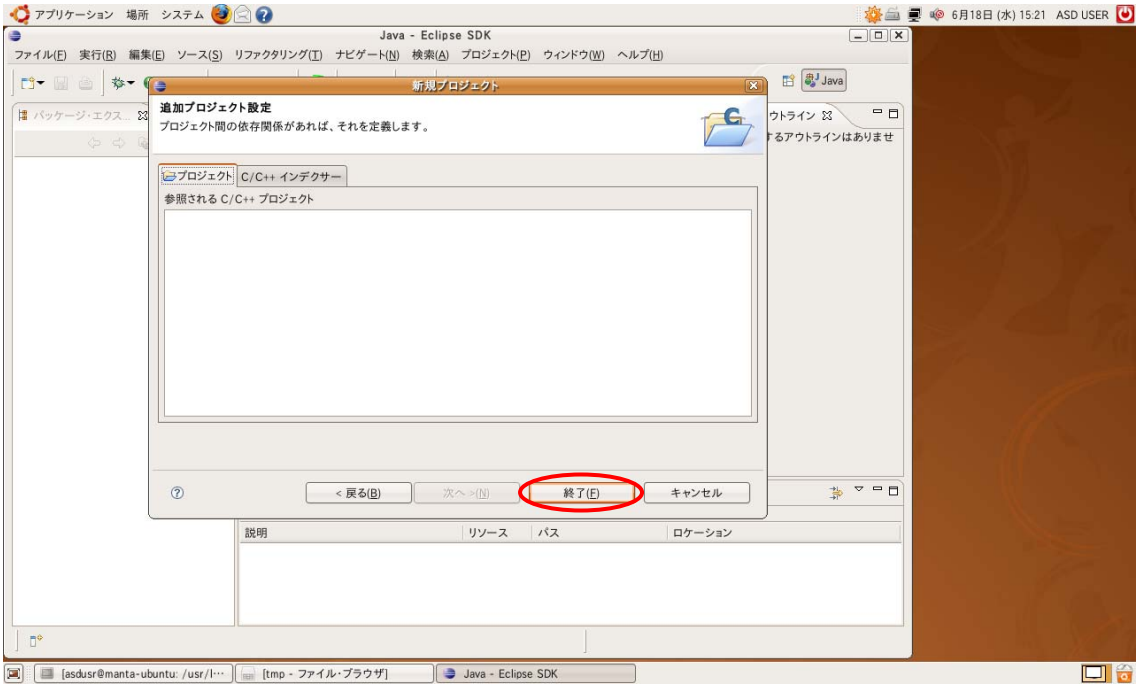


図 3-6-2-4. 追加プロジェクト設定画面

図 3-6-2-4 の追加プロジェクト設定画面が表示後、「終了」をクリックしてください。

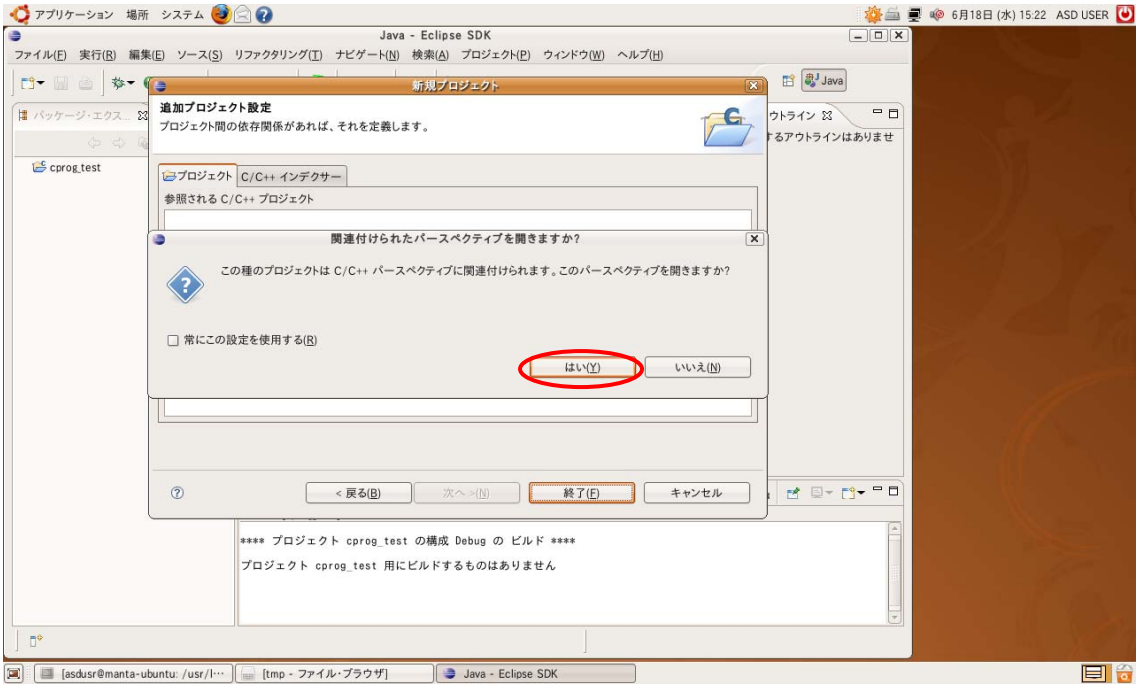


図 3-6-2-5. パースペクティブ確認画面

図 3-6-2-5 の関連付けられたパースペクティブを開きますか？の画面が表示されたら「はい」をクリックしてください。これで、プロジェクトの作成は終了です。

３－６－３　プロジェクトの開発環境設定

Eclipse は新規で作成したプロジェクトの開発環境設定をする必要がありますので、その環境開発の設定方法を説明します。

「3-6-2 プロジェクトの新規作成」で作成したプロジェクト（例では「cproc_test」）をクリックし、メニューから「プロジェクト（P)」→「プロパティー（P)」をクリックすると、図 3-6-3-1 の画面が表示されます。

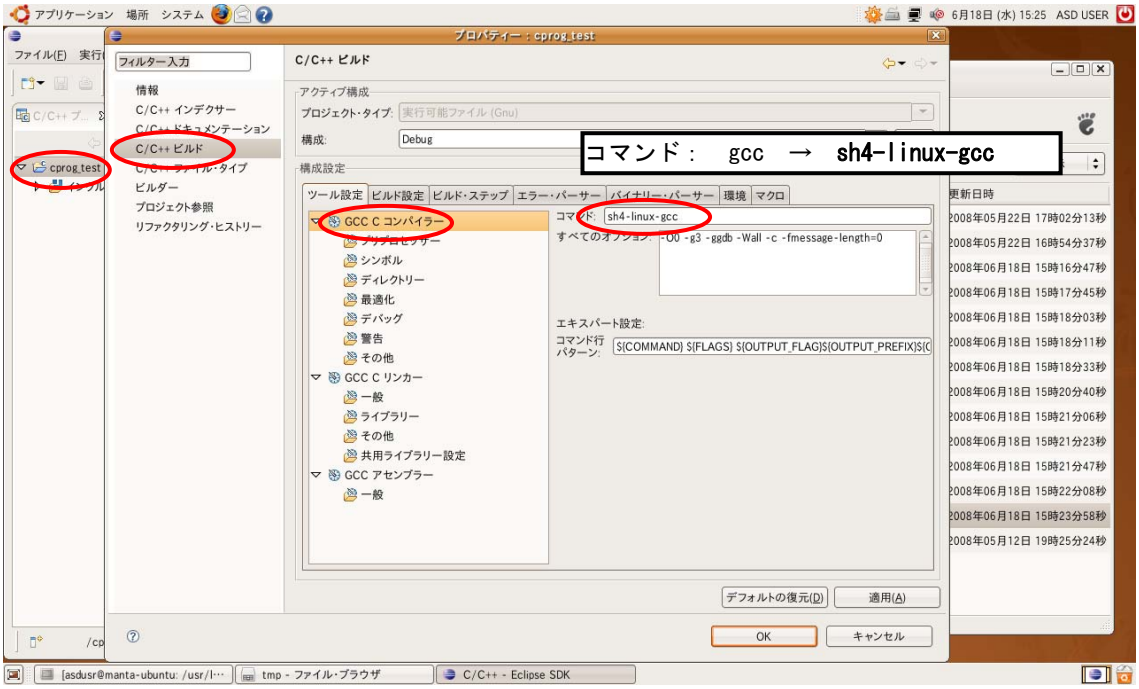


図 3-6-3-1.　プロパティー画面(GCC C コンパイラー)

ここで、図 3-6-3-1 の画面の左側、情報として「C/C++ビルド」を選択してください。
次に、画面中央ツール設定タブを選択し、「GCC　C コンパイラー」をクリックします。
右側のコマンド欄には「gcc」と記載されていますが、「**sh4-linux-gcc**」と変更してください。

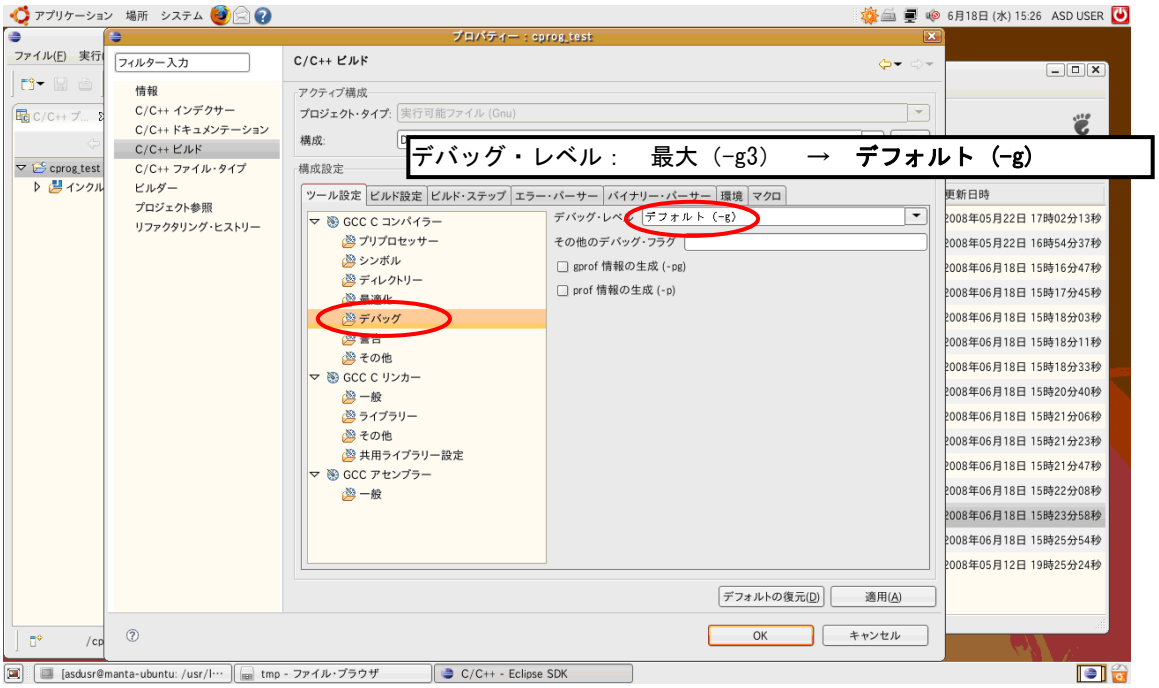


図 3-6-3-2.　プロパティー画面(GCC C コンパイラー　デバッグ)

次に、図 3-6-3-2 の画面中央「GCC　C コンパイラー」の「デバッグ」をクリックします。
右側の「デバッグ・レベル」は「最大（-g3)」が選択されていますが、「**デフォルト(-g)**」に変更してください。

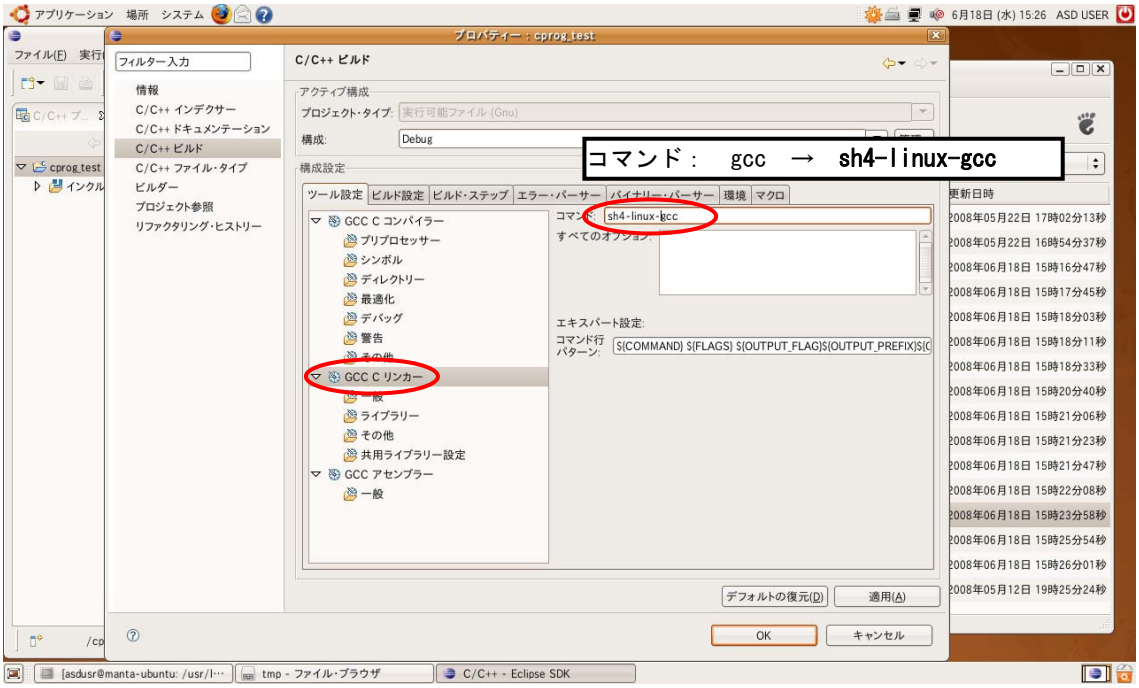


図 3-6-3-3.　プロパティー画面(GCC　C リンカー)

次に、図 3-6-3-3 の画面中央「GCC　C リンカー」をクリックします。
右側のコマンド欄には「gcc」と記載されていますが、「**sh4-linux-gcc**」と変更してください。

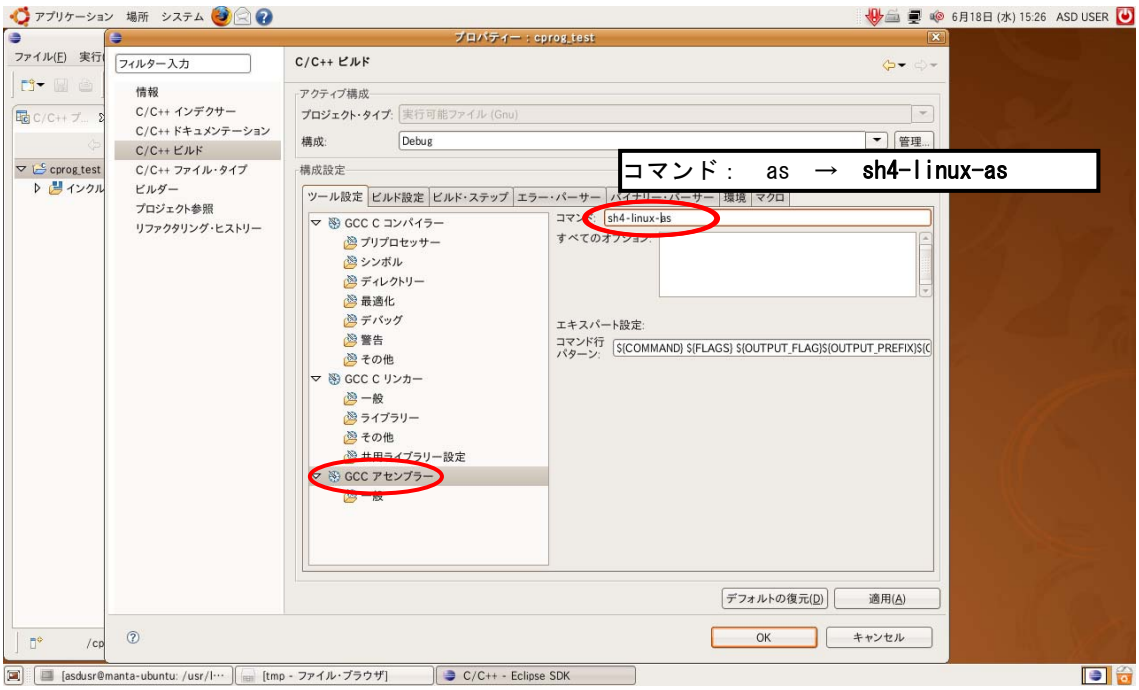


図 3-6-3-4.　プロパティー画面(GCC　アセンブラー)

次に、図 3-6-3-4 の画面中央「GCC　C アセンブラー」をクリックします。
右側のコマンド欄には「as」と記載されていますが、「**sh4-linux-as**」と変更してください。

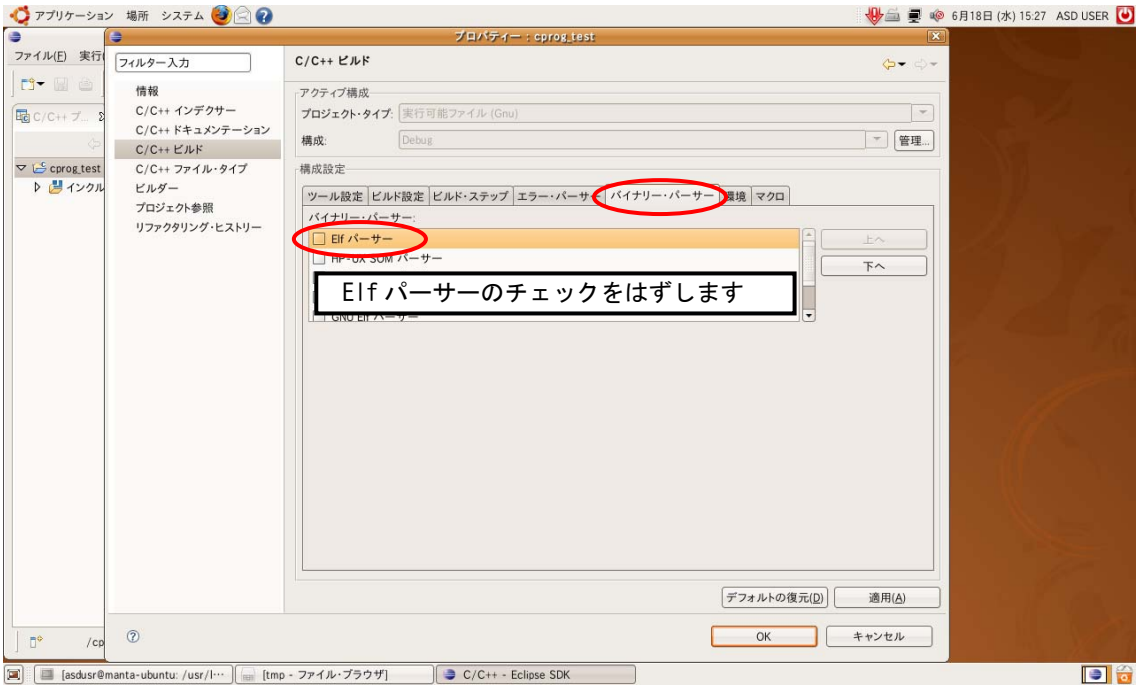


図 3-6-3-5.　プロパティー画面(バイナリー・パーサー1)

次に、「バイナリー・パーサー」タブをクリックし、「Elf パーサー」のチェックをはずします。

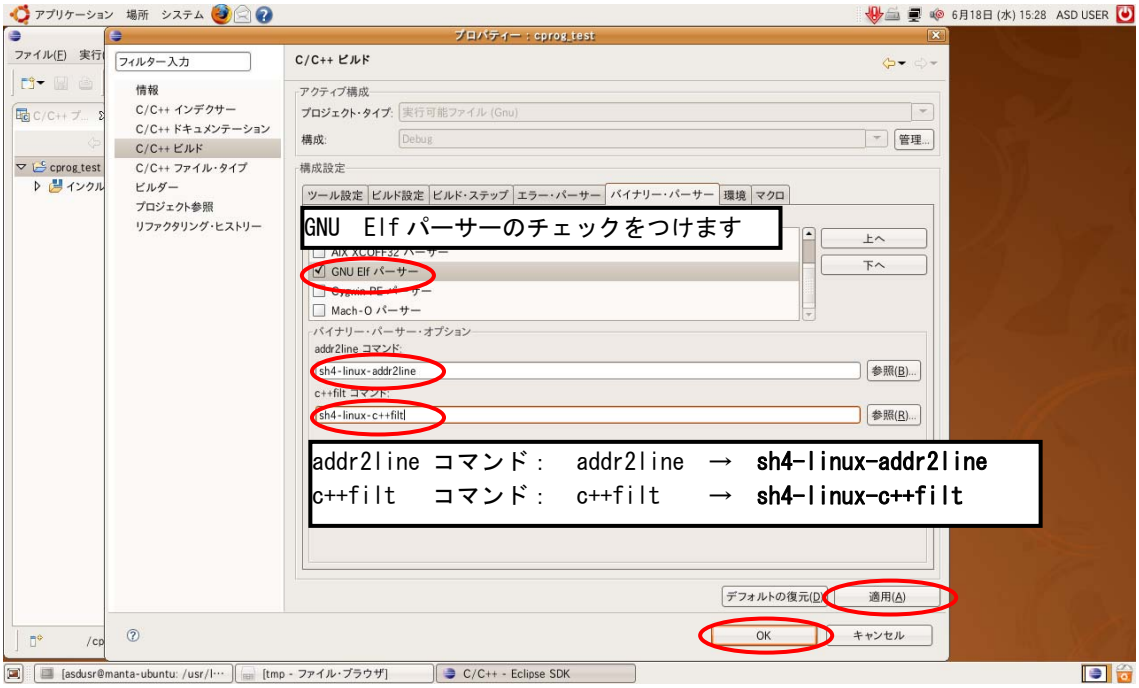


図 3-6-3-6.　プロパティー画面(バイナリー・パーサー2)

「GNU Elf パーサー」にチェックをいれ、図 3-6-3-6 の addr2line コマンドと c++filt コマンドを下記のように設定します。

addr2line コマンド :　addr2line　→　**sh4-linux-addr2line**
c++filt　コマンド :　c++filt　→　**sh4-linux-c++filt**

入力完了後、「適用(A)」をクリックし、「OK」をクリックしてください。

３－６－４　ソースファイル作成とコンパイル

ソースファイルを作成する場合は、「ファイル（F)」→「新規（N)」→「ソース・ファイル」をクリックすると、図 3-6-4-1 の画面が表示されます。

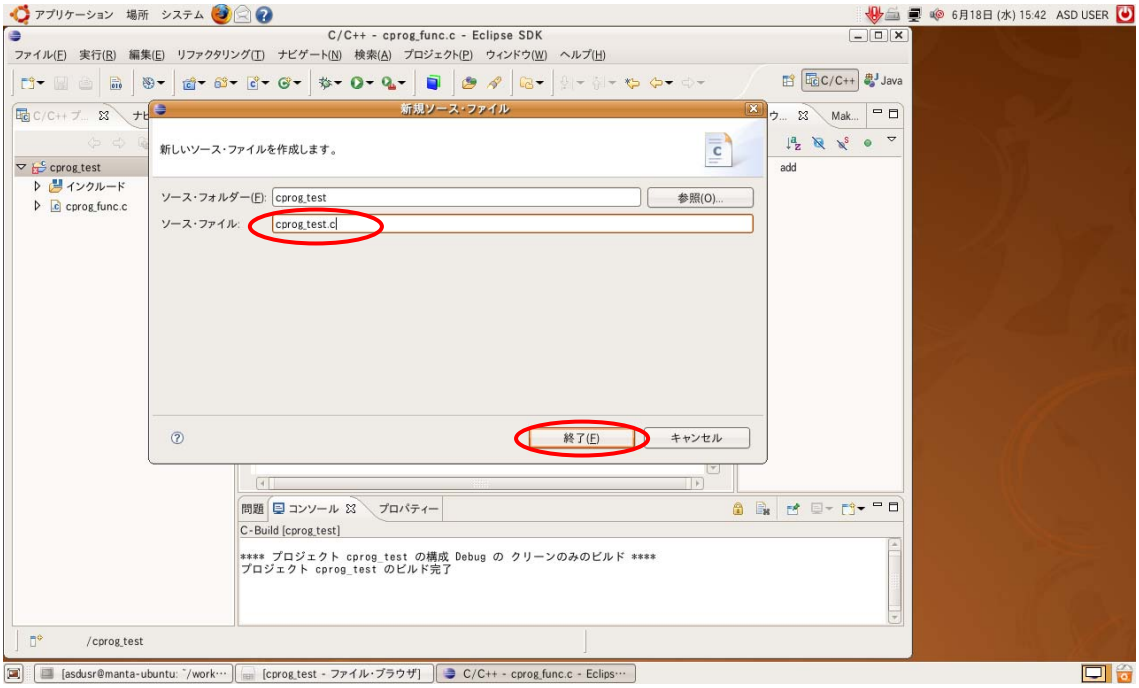


図 3-6-4-1. 新規ソースファイル

ソース・ファイル名を入力し「終了(f)」をクリックしてください。メイン画面のプロジェクトフォルダにソースファイルが追加され、中央のテキストエディタでソースが作成できます。

初期設定では、ソースファイルを保存するたびにビルドするという「自動的にビルド」の設定になっています。
手動でビルドする場合は、メニューの「プロジェクト（P)」→「自動的にビルド」のチェックをはずし、メニューの「プロジェクト（P)」内にある、それぞれのビルド項目を選択してビルドを行ってください。

３－６－５　Eclipseを用いたリモートデバッグ方法

まず、アプリケーションのビルドを行い、ftp で Algo Smart Panel へ転送します。
ビルドされたアプリケーションは、workspace フォルダの Debug フォルダに保存されますので、そのアプリケーションを ftp で Algo Smart Panel へ転送してください。

次に、Algo Smart Panel へ telnet で接続し、下記コマンドを実行し gdbserver を起動します。
（ここでは、例として cprog_test のデバッグを行います）

```
# chmod 755 cprog_test
# gdbserver host1:10000 ./cprog_test
Process ./ cprog_test created; pid = 21507
Listening on port 10000
```

次に、デバッガーの設定を行います。
メニューの「実行（R)」→「構成およびデバッグ（B)」をクリックすると、図 3-6-5-1 の Debug 画面が表示されます。

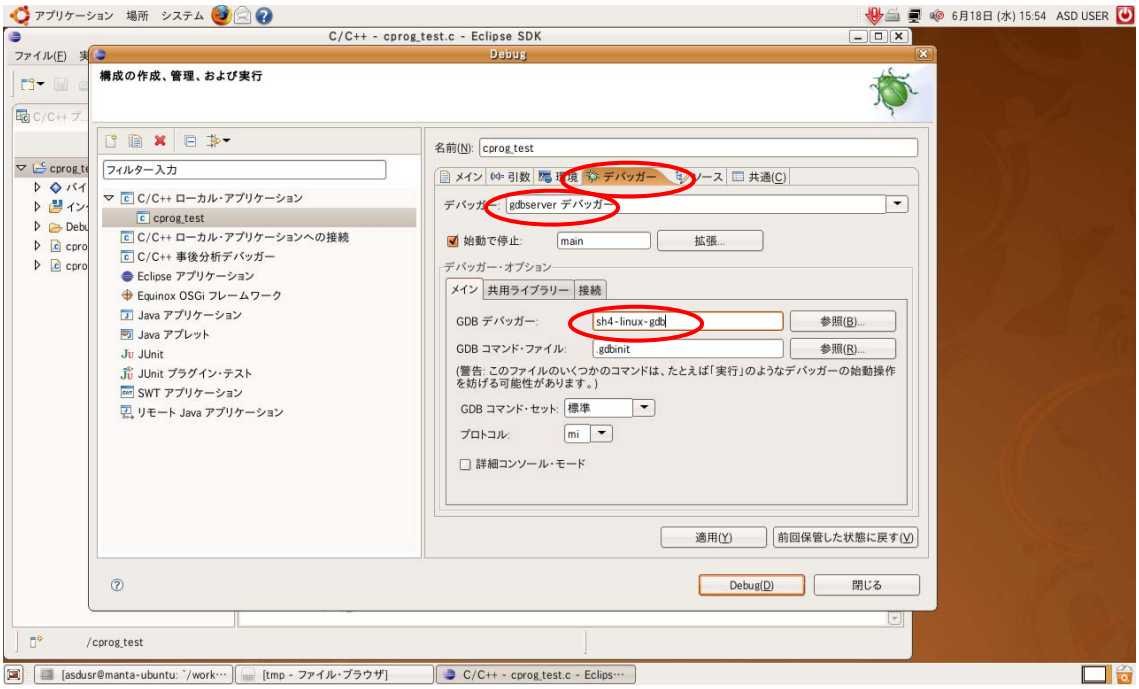


図 3-6-5-1. Debug 画面(デバッガー1)

図 3-6-5-1 左側の「C/C++ローカルアプリケーション」をダブルクリックし、その下のプロジェクト名をクリックしてください。（ここでは例として、cprog_test をクリックしています。）

図 3-6-5-1 右側の「デバッガー」タブをクリックし、デバッガーやデバッガーオプションを下記のように設定します。

デバッガー　　：　gdb/mi　→　**gdbserver　デバッガー**
GDB デバッガー　：　gdb　　→　sh4-linux-gdb

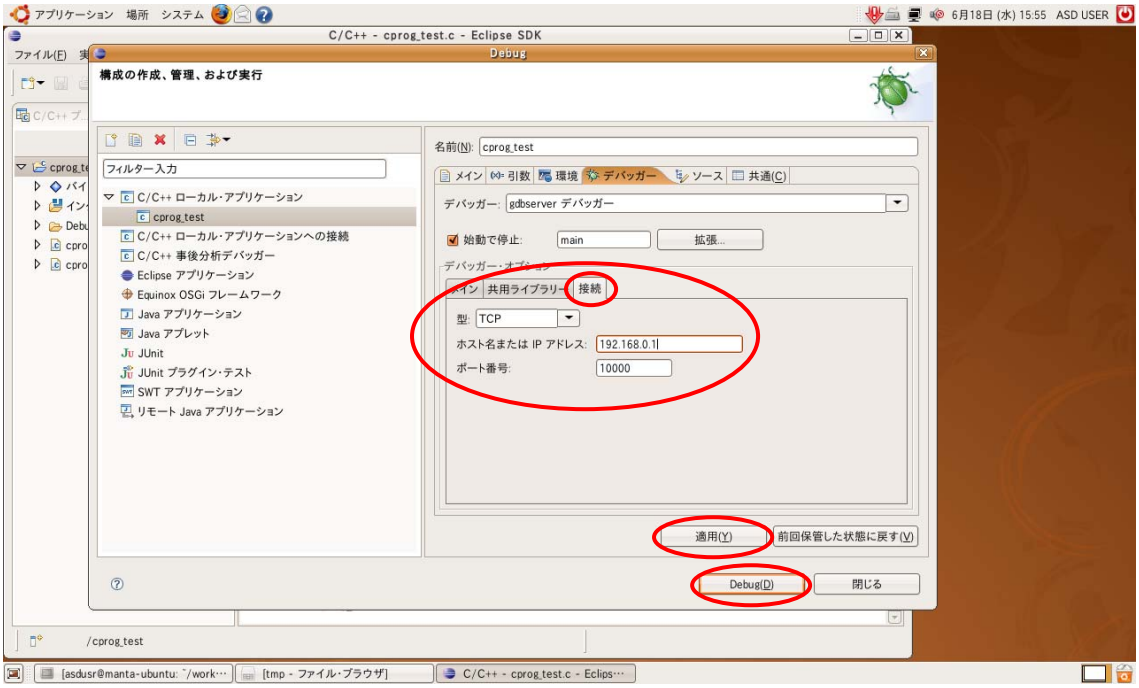


図 3-6-5-2.　debug 画面(デバッガー2)

次に、リモートデバッグを行うホスト（接続先）の設定を行います。
デバッガーオプションの「接続」タブをクリックし、下記のように設定します。

型	:	シリアル　→　**TCP**
ホスト名または IP アドレス	:	AlgoSmartPanel の IP アドレスを設定
ポート番号	:	gdbserver のポートアドレスを設定

設定後、「適用(Y)」ボタンをクリックし、「Debug(D)」ボタンをクリックします。

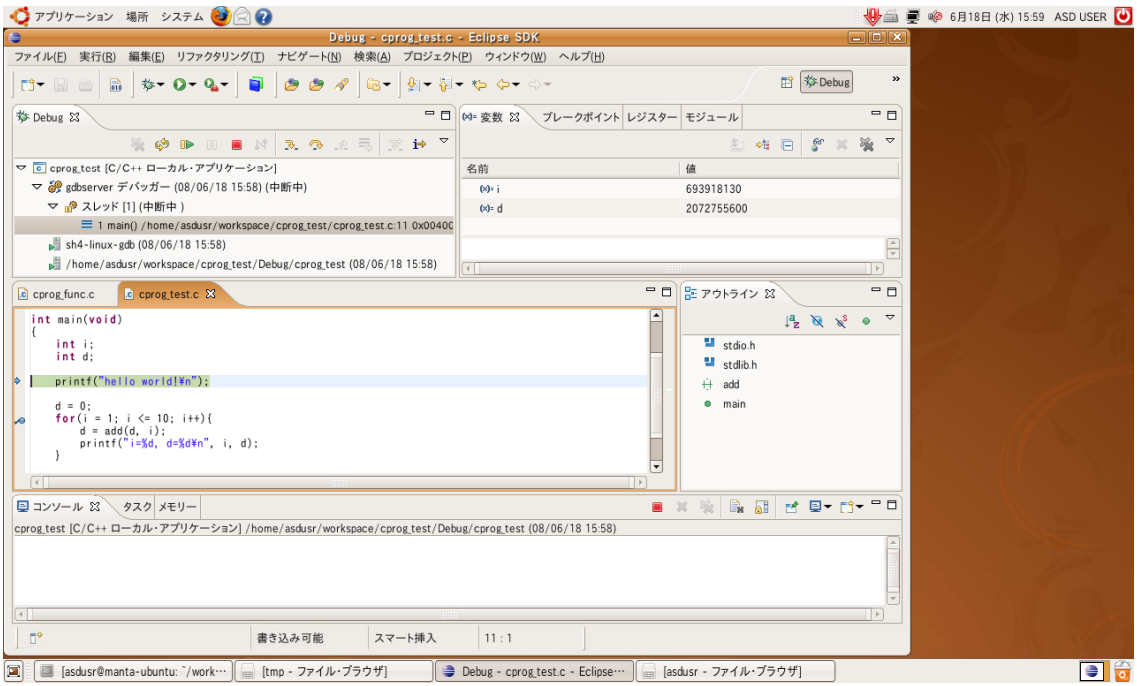


図 3-6-5-3.　リモート接続画面

図 3-6-5-3 のように変更すると、リモート接続ができている状態です。
ブレークポイントは、ブレークポイントを付加する行の左側をダブルクリックすることで作成できます。
再度、ブレークポイントを消去するには、再度、行の左側をダブルクリックします。

デバッグの実行やステップ実行等は、制御パネルを用いて行います。制御パネルについては主なボタンのみを記載します。その他のボタンについては、インターネット及び書籍などを参照してください。

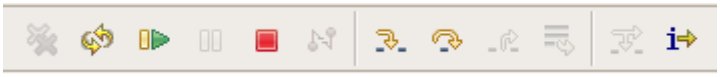


図 3-6-5-4.　制御パネル

・再開ボタン（  ）　：プログラムの実行を再開します。プログラムを明示的に停止するイベント（ブレークポイントなど）があるまでプログラムを実行します。

・中断ボタン（  ）　：実行中のスレッドを中断します。

・停止ボタン（  ）　：デバッグ中のプログラムを強制的に終了します。デバッグ作業が終了となります

・ステップインボタン（  ）
：プログラムを 1 行実行します。もしその行が関数であればその関数の先頭行を実行します。関数呼び出し先の中をデバッグする場合に使用します

・ステップオーバーボタン（  ）
：プログラムを 1 行実行します。もしその行が関数であればその関数全体を実行します。関数呼び出し先の中をデバッグしない場合に使用します

・ステップリターンボタン（  ）
：現在の関数からリターンするまでプログラムを実行します。直前の呼び出し元関数の呼び出し直後に戻ります

第４章　Algo Smart Panelについて

本章では、Algo Smart Panel 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。

※　サンプルソースは、AlgonomixDFB 2 用パッケージ DVD-ROM にある「asd-wsproject2-g1a2-X.XX.tgz」を任意のディレクトリに展開してご使用ください。

表 4-1. サンプルソース一覧

フォルダ名	内容	AP-1000	AP-2000 AP-3100	AP-3101	AP-3102
sample1	AP1000 パネルスイッチインプットデバイス制御方法	○	－	－	－
sample2	AP1000 パネルスイッチキャラクタデバイス制御方法	○	－	－	－
sample3	汎用入出力制御方法	－	○	○	○
sample4	シリアルポート制御方法	○	○	○	○
sample5	ネットワークポート制御方法　※1	△	○	○	○
sample6	オーディオデバイス制御方法	－	○	○	○
sample7	起動ランチャーサンプルプログラム　※1	○	○	○	○
sample8	多言語表示サンプルプログラム	○	○	○	○
Sample9	ウォッチドッグタイマ制御方法	－	○	○	○
Sample10	汎用入力 IN0 リセット制御方法	－	－	○	○
Sample11	汎用入力 IN1 割込み制御	－	－	○	○
Sample12	バックアップ SRAM 制御方法（read/write）	－	－	○	○
Sample13	バックアップ SRAM 制御方法（mmap）	－	－	○	○

※1　AlgonomixDFB 2 では実行できません。

４－１　Algo Smart Panelのデバイスについて

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 Panel 用に作成したデバイスの仕様について説明します。

４－１－１　AP-1000 のパネルスイッチ

AP-1000 に実装されている 5 つのパネルスイッチは、下記の 2 通りの方法でアクセスできます。

・「/dev/input/event0」を使用します。
　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。

・「/dev/pnlsw」を使用します。
　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/AP110/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 4-1-1-1 のような画面が作成されています。

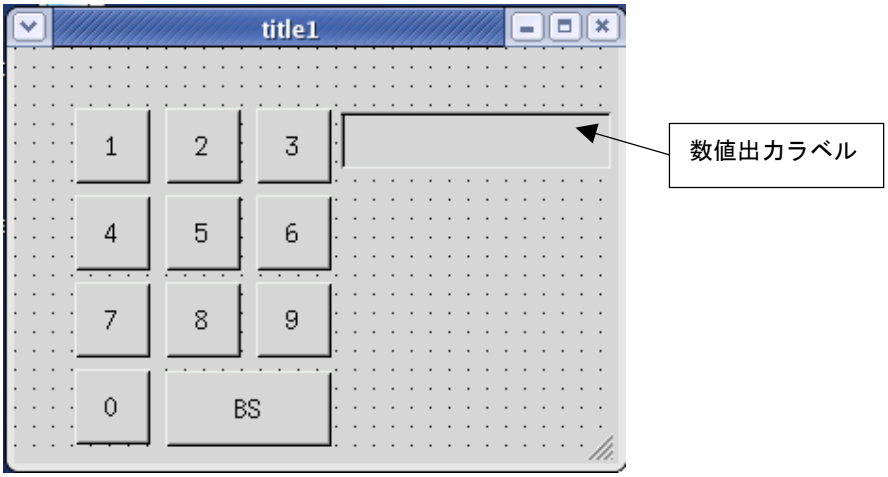


図 4-1-1-1. パネルスイッチインプットデバイスアクセスサンプル画面

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 4-1-1-2 参照）

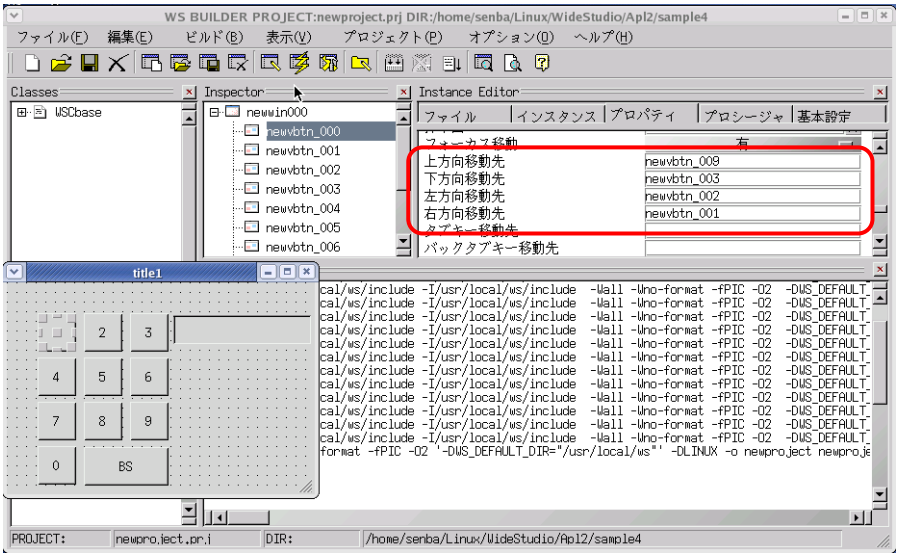


図 4-1-1-2. ボタンプロパティ設定

「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 4-1-1-1. 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ます。AP-1000 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/AP110/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。

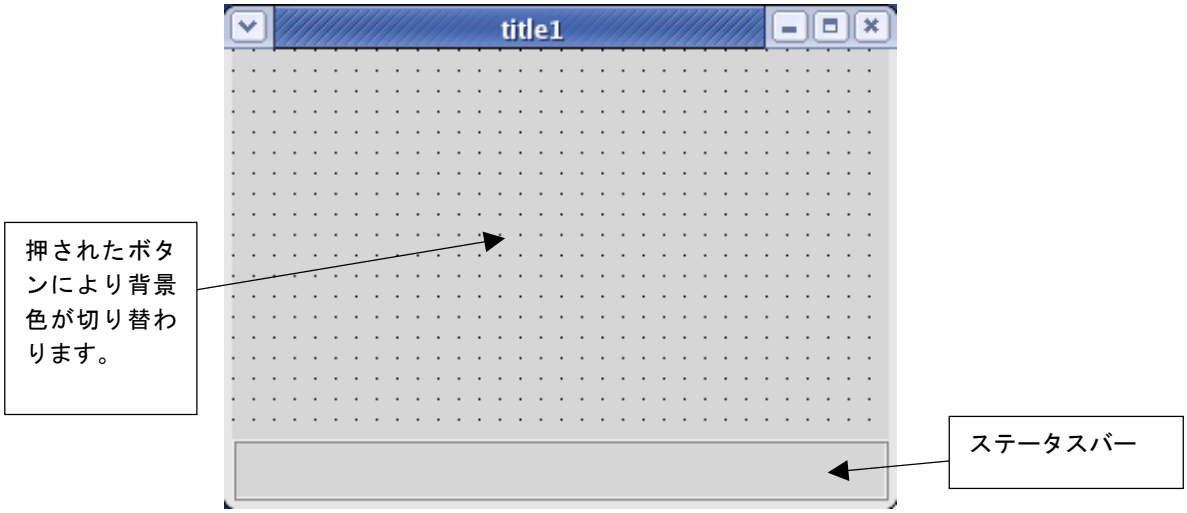


図 4-1-1-3. パネルスイッチキャラクタデバイスアクセスサンプル画面

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 4-1-1-2 に示します。

リスト 4-1-1-2. 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;

        /*パネルスイッチデバイスのオープン*/
    in_fd = open("/dev/pnlsw", O_RDWR);          //パネルスイッチデバイスオープン
    if(in_fd < 0){                               //エラー処理
```

```
        Status_Bar->setProperty(WSNlabelString, "Pnlsw Open Error");
        return;
    }
    mode = PNLSW_RD_ON;          //動作モード変更（ON トリガで READ 関数を抜けるように設定）
    stat = ioctl(in_fd, PNLSW_IOCSMODE, &mode); //動作モード設定
    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();          //スレッドインスタンス取得
    ioctrl_thr->setFunction(Ioctrl_Thread);            //スレッド本体関数を設定
    ioctrl_thr->setCallbackFunction(Io_callback_func); //コールバック関数を設定
    ioctrl_thr->createThread((void*)0);                //スレッド生成
}
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 4-1-1-1 に示します。

表 4-1-1-1. 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2Byteのデータがリターンされます。
1Byte目は、現状のスイッチ状態です。1でON、0でOFFです。
2Byte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 4-1-1-3 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 4-1-1-3. 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));    //コールバック関数実行
            }
        }
    }
    return(NULL);
}
/*スレッドから通知され、メインスレッドで実行されるコールバック関数*/
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");    //ピンク
        break;
    }
}
```

パネルスイッチデバイスを読み出すとき 2Byte 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 4-1-1-4 参照）。

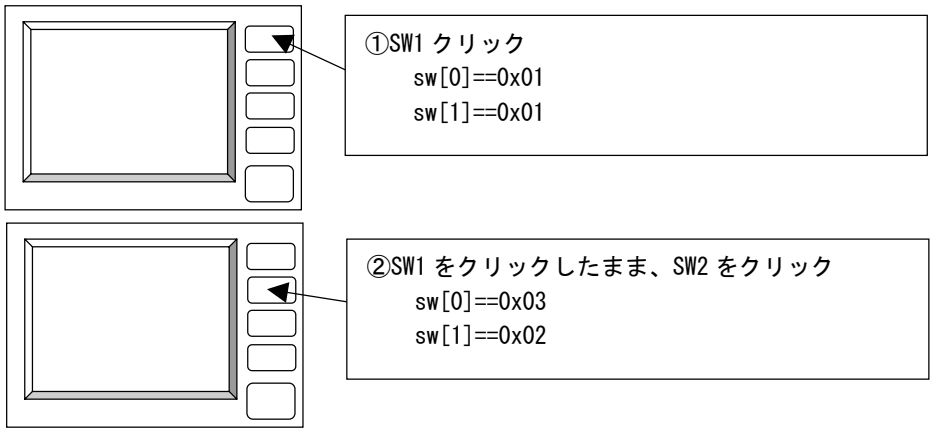


図 4-1-1-4. パネルスイッチ読み出しデータ

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。

※　このプログラムを AP1000 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 AP1000 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。

４－１－２　汎用入出力

Algo Smart Panel 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/AP210_310/sample3」「/wsproject/AP315/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。

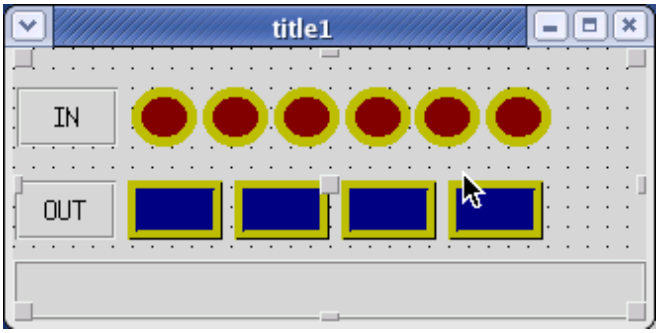


図 4-1-2-1. 汎用入出力デバイス制御サンプル画面

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 4-1-2-1 に示します。

リスト 4-1-2-1. 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;

        /*汎用出力デバイスのオープン*/
    out_fd = open("/dev/genout", O_RDWR);       //汎用出力デバイスオープン
    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

        /*汎用入力デバイスのオープン*/
    in_fd = open("/dev/genin",  O_RDWR);            //汎用入力デバイスオープン
    if(in_fd < 0){                                  //エラー処理
        Status_Bar->setProperty(WSNlabelString, "IN Device Open Error");
        close(out_fd);
        return;
    }

        /*スレッドの生成*/
      ioctrl_thr = 0;
      ioctrl_thr = WSDthread::getNewInstance();         //スレッドインスタンス取得
    ioctrl_thr->setFunction(Ioctrl_Thread);             //スレッド本体関数を設定
    ioctrl_thr->setCallbackFunction(Io_callback_func);  //コールバック関数を設定
    ioctrl_thr->createThread((void*)0);                 //スレッドを生成
}
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 4-1-2-2 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 4-1-2-2. 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);
}
/*スレッドから通知され、メインスレッドで実行されるコールバック関数*/
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、1Byte 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 4-1-2-3 に汎用出力の制御ソースコードを示します。

リスト 4-1-2-3. 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 4-1-2-1 に汎用入出力のリファレンスを示します。

表 4-1-2-1. 汎用入出力デバイスリファレンス

GENIN,GENOUT

名前

genin ー汎用入力6点
genoutー汎用出力4点

説明

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1Byte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1Byte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 Panel 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

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。
以下より、シリアル通信のサンプルについて説明します。

●通信制御サンプルプログラム

「/wsproject/AP110/sample4」「/wsproject/AP310/sample4」「/wsproject/AP315/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;

        /*シリアルポートオープン*/
    comm_fd = open("/dev/ttySC4", O_RDWR | O_NOCTTY);          //シリアルポートオープン
    if(comm_fd < 0){
        Status_Bar->setPropertyV(WSNlabelString, "ttySC4 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 （データ長 8bit ストップビット 1bit パリティ無し 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、パリティエラーなし）
    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 50ms
    tio.c_cc[VMIN] = 1;                 //1 文字取得するまでブロック
    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 （ボーレート 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;
    }

        /*スレッドの生成*/
      comctrl_thr = 0;
      comctrl_thr = WSDthread::getNewInstance();        //スレッドインスタンス取得
    comctrl_thr->setFunction(Comctrl_Thread);           //スレッド本体関数を設定
    comctrl_thr->setCallbackFunction(Com_callback_func);  //コールバック関数を設定
    comctrl_thr->createThread((void*)0);                //スレッドを生成
}
static WSCfunctionRegister  op("Main_Init", (void*)Main_Init);
```

「open」関数で通信を行いたいポートのデバイスをオープンします。「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、ならびにシリアルデバイスの使用方法等の詳細については、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);              /*1Byte 受信*/
        if(len == 1){
            write(comm_fd, &data, 1);               /*1Byte ト送信*/
            obj->execCallback((void*)&data);
        }
    }
    return(NULL);
}
/*スレッドから通知され、メインスレッドで実行されるコールバック関数*/
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 1Bype 受信するまで待ちます。1Byte 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/AP110/sample5」「/wsproject/AP210_310/sample5」「/wsproject/AP315/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。

※　AP-1000 は Algo Smart Panel で動作確認済みの USB LAN アダプタを使用すれば LAN 通信が可能です。

●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;
      }

        /* ポート番号指定 */
      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);              //ポート番号指定
    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;
      }
        /*スレッドの生成*/
      srvctrl_thr = 0;
      srvctrl_thr = WSDthread::getNewInstance();            //スレッドインスタンス取得
    srvctrl_thr->setFunction(Srvctrl_Thread);               //スレッド本体関数を設定
    srvctrl_thr->setCallbackFunction(Srv_callback_func);    //コールバック関数を設定
    srvctrl_thr->createThread((void*)0);                    //スレッドを生成
}
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 flg;
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;
            }
            flg = 0;

            for(;;){
                  /* データ受信待ち */
                  if(flg == 0){ /*exeCallback 関数が実行されるまで処理しない*/
                        len = recv(cli_sock, tmp, BUFFER, 0);
                        if (len > 0){
                            flg = 1;
                            obj->execCallback((void *)len); /*正常受信完了*/
                        }else{
                            close(cli_sock);                /*通信断*/
                            break;
                        }
                  }
                  usleep(10*1000);  /*10mswait*/
        }
    }
    return(NULL);
}
/*スレッドから通知され、メインスレッドで実行されるコールバック関数*/
void Srv_callback_func(WSDthread *, void *val)
{
    int len;

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

「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 4-2-2-2 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 アドレス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;

    /* プログラム起動時の引数でサーバの IP アドレスを取得 */
    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;
    }

    /* サーバに接続 */
    memset(&srv_addr, 0, sizeof(srv_addr));
    srv_addr.sin_family = AF_INET;
    srv_addr.sin_addr.s_addr = inet_addr(srv_ip);       //ターゲットサーバ IP アドレス
    srv_addr.sin_port = htons(8900);                    //ターゲットサーバポート番号
    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;
    /* データ送信 */
    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 台の Algo Smart Panel 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

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/AP310/sample6」「/wsproject/AP315/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");  /*オーディオデバイスオープンエラー*/
        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.data", O_RDONLY)) < 0) {
        fprintf(stderr, "sound data open() failed¥n");   /*ファイルオープンエラー*/
        close(dsp);
        return;
    }

    // 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

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/AP310/sample9」「/wsproject/AP315/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

RAS 機能として以下の様な機能を実装しています。

- 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。

注　RAS 機能は AP3101/AP3102 のみのサポートしております。

●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/AP315/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/AP315/sample11」に汎用入力 IN1 割込み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/AP315/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。バックアップ SRAM として 512KByte の SRAM を実装しています。4KByte はシステム領域として使用しているため、ユーザー領域は 508KByte を使用することができます。

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 0x7F000
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/AP315/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 0x7F000

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/AP110/sample7」「/wsproject/AP310/sample7」「/wsproject/AP315/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。

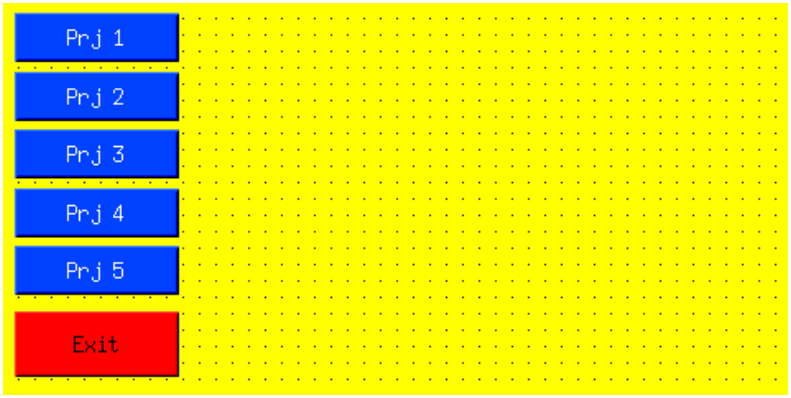


図 4-3-1-1. ランチャーサンプルプログラムメイン画面

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);/* クリックされたボタンを特定する */
    switch(btn){
    case 1:
        newwin000->setVisible(False);       /* メインウィンドウを非表示 */
            system("xeyes");                /* xeyes プログラムを起動 */
```

```
        newwin000->setVisible(True);        /* メインウィンドウを表示 */
        break;
    case 2:
        newwin000->setVisible(False);
            system("xfontsel");             /* xclock プログラムを起動 */
        newwin000->setVisible(True);
            break;
    case 3:
        newwin000->setVisible(False);
            system("xcalc");                /* xcalc プログラムを起動 */
        newwin000->setVisible(True);
            break;
    case 4:
        newwin000->setVisible(False);
            system("xlogo");                /* xlogo プログラムを起動 */
        newwin000->setVisible(True);
            break;
    case 5:
        newwin000->setVisible(False);
            system("asd_config");           /* ASD Config プログラムを起動 */
        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、ASD Config 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/AP110/sample8」「/wsproject/AP310/sample8」「/wsproject/AP315/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 UTF-8 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『3-2-10 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 Panel 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　AlgonomixDFB 2 設定ファイルについて

AlgonomixDFB 2 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。

４－４－１　/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。

※　自動起動の場合は、ルート権限で実行されます。

４－４－２　/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
```

①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。
②eth0 のネットワークアドレスを設定します。

４－４－３　/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）
①asdap 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。

４－４－４　/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 アドレスを検索して見つからなかったとき 2 番目の IP アドレスを検索します。

４－４－５　/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/local/bin:/usr/local/sbin:/usr/bin:/usr/sbin:/bin:/sbin"

if [ "`id -gn`" = "`id -un`" -a `id -u` -gt 99 ]; then
        umask 002
else
        umask 022
fi

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

①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。

４－４－６　/etc/ftp/ftp_download.sh

リスト 4-4-6-1. FTP ダウンロード設定ファイル

```
!/bin/sh
#
# /etc/ftp/ftp_download.sh
#

cd /home/asdusr

ftp -n ${host_name} << _EOD
user ${user_name} ${password}
passive
get download.sh
binary
get asd-update.tgz
bye
_EOD

exit 0
```

①ASD Config でデータのダウンロードを行う場合に、ダウンロード対象ファイルを指定できます。

※　「2-4-8 Hardware Information」参照してください。

４－５　動作確認済みUSB機器一覧

Algo Smart Panel で対応したデバイス（USB 1.1 / USB 2.0）を使用できる 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、弊社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。

※　AP-3101/3102 のみ USB 2.0 に対応しています。

※　USB HUB を使用して、複数の USB 機器を同時に接続する場合は、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、弊社ホームページにてご確認ください。

４－６　起動画面の変更について

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。

※　起動画面の Algo Smart Panel 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 4-6-1-1 のようになります。

表 4-6-1-1. 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	240 ピクセル
色数	24 ビットカラー

画像ファイルは AP-1000/2000/3100/3101/3102 のどの機種も表 4-6-1-1 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-1000/2000 と AP-3100/3101/3102 で異なります。

- AP-1000/2000 の場合
  画像ファイルの画像中央部 320x240 のエリアが表示されます。
- AP-3100/3101/3102 の場合
  画像ファイルの画像が表示されます。

各機種での表示イメージを図 4-6-1-1 に示します。

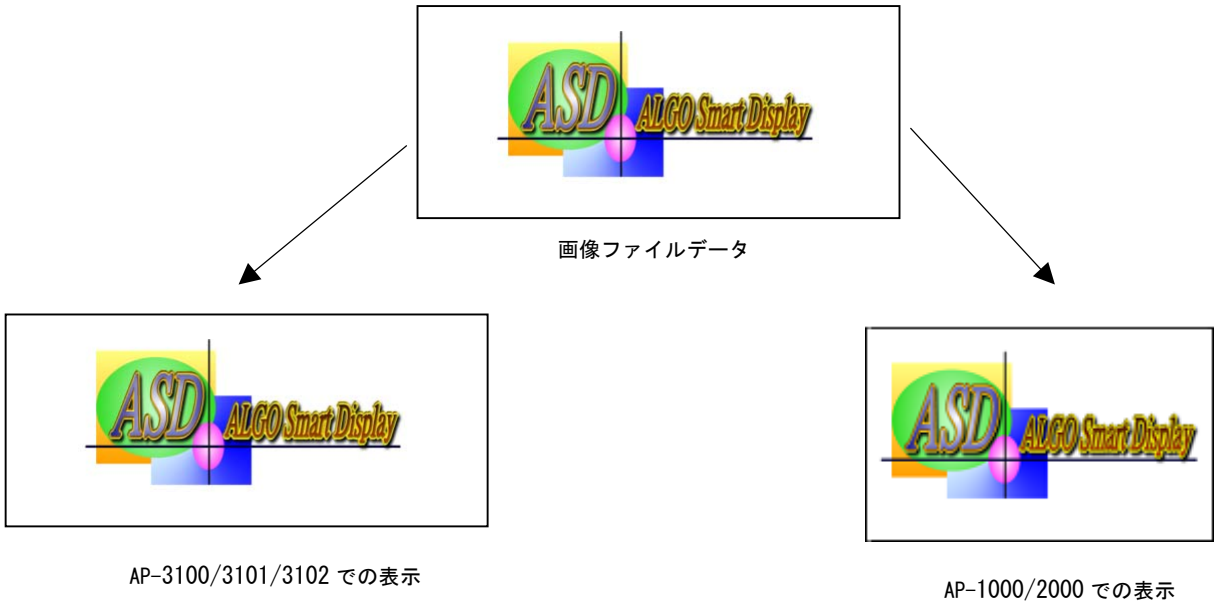


図 4-6-1-1. 起動画面表示イメージ

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 Panel 本体のフラッシュメモリに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から画像バイナリデータへの変換方法を説明します。

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-dev/tools-0040/bin/chbmp sample.bmp
```

③sample.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 Panel 本体のフラッシュメモリに書き込みます。

※　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。

●USB メモリを使って転送

①USB 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 4-6-3-1 のように記述します。

リスト 4-6-3-1.「download.sh」の例

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

②USB メモリを Algo Smart Panel に挿入し、USB メモリ自動スクリプトを実行します。
③USB 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。

●ftp を使って転送

①「3-2-9 ファイルの転送」を参考に ftp で Algo Smart Panel 本体に画像バイナリデータ(sample.bmp.bin)を転送します。
②telnet またはシリアルコンソールで Algo Smart Panel 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。

４－７　CRCファイルチェックについて

ここでは、CRC ファイルチェックの作成方法について説明します。

チェックプログラムは、『/usr/local/sh4-linux-dev/tools-0040/bin/checksum』に格納されています。
本プログラムの使用法を以下に記述します。

『download.sh』の CRC チェック用ファイルを作成する場合は、ターミナルプログラムを起動し、以下のコマンドを実行します。コマンドを実行すると、「chksum: XXXXXXX」と画面に出力されます。sumcheck_download ファイルを作成し、XXXXXXX の値を１行記述します。
/usr/local/sh4-linux-dev/tools-0040/bin/checksum 0 download.sh

４－８　付属ツールについて

ここでは、付属ツールについて説明します。

４－８－１　list_out

スクリプトファイル等でコンソールに出力する文字列を Algo Smart Panel の画面上で表示したい場合には本プログラムを使用します。
本プログラムは標準出力(echo、printf)を表示します。

スクリプトファイルを実行している箇所の後ろに「スペース、パイプ(|)、スペース、list_out」と入力します。
下記例では、「dmesg」コマンドを画面に表示します。

```
# dmesg | list_out
```

①実行完了後、画面上に結果が表示されます。

図 4-8-1-1. list_out 実行結果

②確認後、Close ボタンを押します。

※　DirectFB は、マルチウインドウが未対応になります。従って他のプログラム(ASD コンフィグ等)でウインドウを使用している場合は、ウインドウを表示することができませんので注意してください。

４－８－２　ConsoleMesg、SendMesg

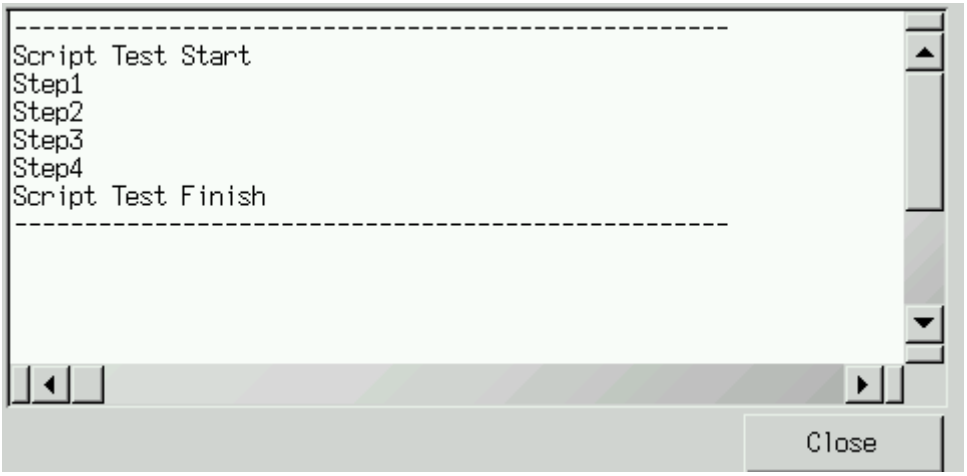
スクリプトファイル等でコンソールに出力する文字列を Algo Smart Panel の画面上で表示したい場合には本プログラムを使用します。
本プログラムは、ConsoleMesg で画面の表示を行い、SendMesg で表示したい文字列を送信します。

①スクリプトファイル内で文字列を送信したい箇所に「SendMesg　＜文字列＞」を入力します。

```
#!/bin/sh
   SendMesg "--------------------------------------------------"
   SendMesg "Script Test Start"
   SendMesg "Step1"
   SendMesg "Step2"
   SendMesg "Step3"
   SendMesg "Step4"
   SendMesg "Script Test Finish"
   SendMesg "--------------------------------------------------"
```

②ConsoleMesg プログラムを起動します。
（スクリプトファイル内で ConsoleMesg を起動することもできます。）

③ConsoleMesg プログラムが正常に起動しているのを確認した上で、作成したスクリプトファイルを実行します。実行後、画面に文字列が表示されます。



④確認後、Close ボタンを押します。
（スクリプトファイル内で ConsoleMesg を終了させることもできます。）

※　DirectFB は、マルチウインドウが未対応になります。従って他のプログラム(ASD Config 等)でウインドウを使用している場合は、ウインドウを表示することができませんので注意してください。

４－９　マウスカーソルを非表示にする方法について

DirectFB ではカーソルデータを透明なカーソルデータに変更することでカーソルを非表示にすることができます。ここでは、カーソルデータファイルを置き換える方法を紹介します。

DVD に添付している、<DVD>/development/cursor/cursorTOUKA.dat を ASP の /usr/share/directfb-1.1.1/cursor.dat と置き換えます。
ここでは、cursorTOUKA.dat を/home/asdusr にアップロードしたと仮定します。
ASP へのファイルのアップロード方法については「3-2-9 ファイルの転送」を参照ください。

```
# cp /home/asdusr/cursorTOUKA.dat  /usr/share/directfb-1.1.1/cursor.dat
```

その後再起動することでカーソルを非表示にすることができます。

また、カーソルを標準状態に戻したい場合は、<DVD>/development/cursor/cursorDEFAULT.dat を ASP へアップロードし、先程と同様に置き換えます。

```
# cp /home/asdusr/cursorDEFAULT.dat  /usr/share/directfb-1.1.1/cursor.dat
```

この場合も同様に再起動をしてください。

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

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。

７７Ｇ０１００２５Ｇ　　２００９年　１０月　第７版
７７Ｇ０１００２５Ａ　　２００８年　　３月　　初版

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　　ＴＥＬ（０３）３５４１－７１７０
銀座堀ビル２Ｆ　　ＦＡＸ（０３）３５４１－７１７５

大阪支社
〒５４２－００８１　大阪府大阪市中央区南船場１－１２－３　　ＴＥＬ（０６）６２６３－９５７５
船場グランドビル３Ｆ　　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　　ＴＥＬ（０５２）９３９－５３３３
ふぁみーゆ葵ビル５０３　　ＦＡＸ（０５２）９３９－５３３０

ホームページ　http://www.algosystem.co.jp