ＣＲＴ画面帳票作成ツール

SimpleMaster

取り扱い説明書

株式会社 フジテクニカ

目次

ページ

1　はじめに ---------- 1

2　レイアウト作成

2－1　SimpleMaster の起動 ---------- 2

2－2　メニュー画面の作成 ---------- 4

2－3　売上げ帳画面の作成 ---------- 11

2－4　伝票入力画面の作成 ---------- 21

2－5　帳票レイアウトの作成 ---------- 27

2－6　制御用画面の作成 ---------- 32

3　ロジック作成

3－1　プログラムの構造 ---------- 36

3－2　共有データ領域 ---------- 37

3－3　最小限のアプリケーションプログラム ---------- 39

3－4　コンパイラの設定 ---------- 41

3－5　プログラムの実行 ---------- 47

3－6　売上げ帳画面処理プログラム ---------- 50

3－7　伝票入力画面処理プログラム ---------- 55

3－8　帳票印字処理プログラム ---------- 58

3－8　制御用データ処理プログラム ---------- 61

4　デモ用プログラム

4－1　ソースプログラムの修正 ---------- 63

4－2　コンパイラの設定 ---------- 64

4－3　デモ用プログラムの実行 ---------- 64

5　スプレッドシート

5－1　概要 ---------- 65

5－2　レイアウト作成 ---------- 67

5－3　スプレッドシートの表示 ---------- 72

5－4　表示データの設定 ---------- 74

5－5　サンプルプログラム ---------- 76

Windows 版 と Linux 版の相違点

レイアウト作成プログラム SimpleMaster ではダイアログボックスの表示形式などに相違点がありますが、設定する項目、内容は同一です。
作成したアプリケーションプログラムの実行内容は同一です。
その他、明確な相違点がある場合には説明書に記述しています。

改訂記録

2004/01/15　Ver1.01　新規作成
2004/06/01　Ver1.11　スプレッドシート機能追加 ／ Linux 対応

1 はじめに

SimpleMaster は CRT 画面、帳票印字用のレイアウト作成を行います。
アプリケーションプログラムでは SimpleMaster で作成したレイアウト情報を参照して CRT 画面処理、帳票印刷を簡単に行うことができます。

SimpleMaster は次のような機能を持っています。

(1) プロジェクト単位での管理
SimpleMaster を使用して作成するレイアウト情報とアプリケーションプログラムをプロジェクト単位で管理しています。
プロジェクトにはつぎのようなファイルが含まれます。
プロジェクト情報ファイル
レイアウト情報ファイル
ソースプログラムファイル
表示色データ、文字フォントデータなどレイアウト作成時、アプリケーションプログラム実行時に参照されるデータファイル。

(2) レイアウト作成
レイアウトは CRT 画面、帳票を模擬した作業領域に
直線、矩形などの基本図形
固定文字列、可変文字列、数値データ
スクロールバー、ボタンなどのコントロール
などを貼り付けることで作成します。
レイアウトの作成はマウス操作と簡単なキーボード入力で行います。

(3) スプレッドシート機能
簡単な設定で表を作成することができます。 キーボード入力、画面のスクロールなどが自動的に行われます。

(4) マン・マシン部を構築するスタティックライブラリ
SimpleMaster で作成したレイアウト情報はアプリケーションプログラムに組込まれたスタティックライブラリ SimplePack で処理します。
SimplePack には CRT 画面表示、帳票印刷以外に、マウス操作、キーボード入力などマン・マシン部を構築するための機能が含まれています。
アプリケーションプログラムでは簡単な関数を使用して SimplePack と情報の受渡しを行うだけで CRT 画面、帳票印刷の処理を行うことができます。

(5) ウィンドウアプリケーション開発に必要な機能を提供
アプリケーションプログラムの起動、停止処理
ディスクファイルの操作
CRT 画面のハードコピー
などウィンドウアプリケーションの開発に必要な機能が SimplePack に実装されています。

[取り扱い説明書]では簡単なアプリケーションを想定してレイアウト作成からロジック作成までの手順を説明します。
詳細な仕様はレファレンスマニュアルを参照してください。

2　レイアウトの作成

SimpleMaster を使用して簡単な メニュー選択画面 を作成してみます。

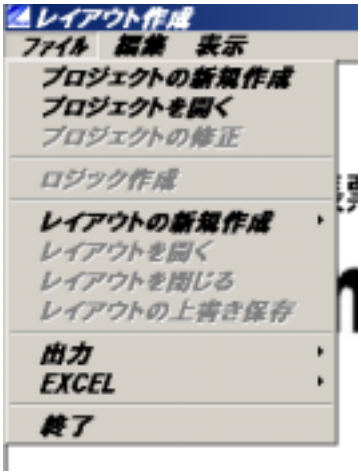
2－1　SimpleMaster の起動

SimpleMaster を起動するには次の3種類の方法があります。

(1) ［スタート］メニュー → ［プログラム］ → ［SimpleMaster］ → [レイアウト編集] の順にメニューを選択して起動します。
(2) デスクトップ上に配置された SimpleMaster のアイコンをダブルクリックして起動します。
(3) エクスプローラで プロジェクトファイル (拡張子:prj) または レイアウトファイル (拡張子:ptn) をダブルクリックすることで起動します。

2－1－1　SimpleMaster 起動時の画面

［ファイル］メニュー⇒ ［プロジェクトの新規作成］ を選択します。



SimpleMaster を2回目以降に起動した場合は、前回に開いたプロジェクトが選択された状態で起動します。

プロジェクトファイルを直接指定して起動した場合には、指定されたプロジェクトが選択された状態で起動します。

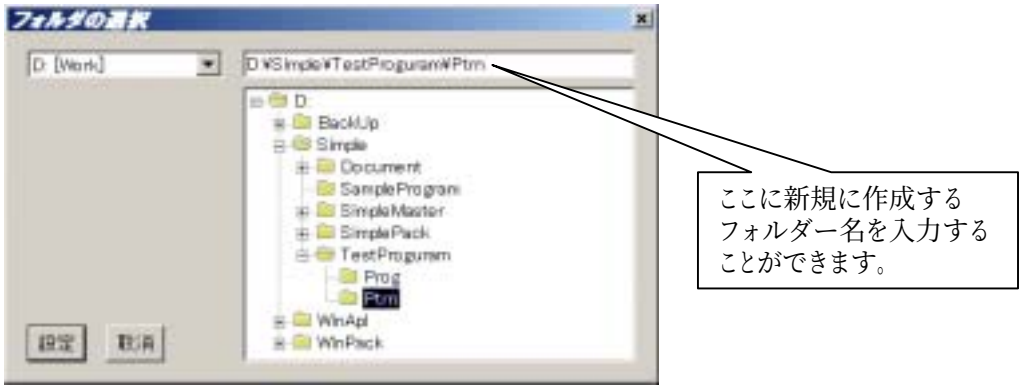
レイアウトファイルを直接指定して起動した場合には、レイアウトが属するプロジェクトが選択され、指定されたレイアウトが読込まれた状態で起動します。

2－1－2　プロジェクトの新規作成

SimpleMaster では、適応するアプリケーションごとにプロジェクトを作成し、レイアウト情報、ロジック情報を管理しています。
プロジェクトの内容は後で修正することができますので、ここでは メニュー画面 を作成するために最小限の設定のみを行います。

プロジェクト名	プロジェクトの名称を設定します。 ここでは 「SimpleDemo」 と設定します。
プロジェクト情報格納先フォルダー名	レイアウト情報など SimpleMaster で生成、編集するファイルを格納するフォルダーを指定します。
ロジック情報格納先フォルダー名	ロジック情報（アプリケーションプログラム）を格納するフォルダーを指定します。 SimpleMaster ではレイアウト情報とプログラムの間で情報を共有するためにヘッダーファイル、ソースプログラムファイル環境設定ファイルを作成し、ここに格納します。

フォルダーの ［変更］ボタン をクリックするとフォルダーの選択ダイアログボックスが開き、既存のドライブ、フォルダーより選択することができます。（新規作成もできます。）

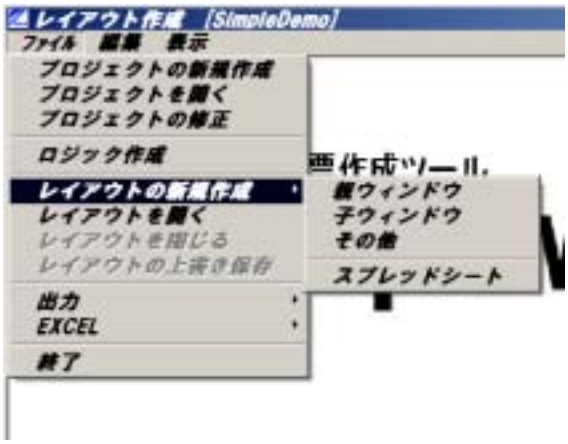


指定されたフォルダーが存在しない場合には生成されます。

2－2　メニュー画面の作成

2－2－1　レイアウトの新規作成

プロジェクト登録後、［ファイル］メニュー　→［レイアウトの新規作成］→［親ウィンドウ］を選択します。



親ウィンドウ / 子ウィンドウ / その他

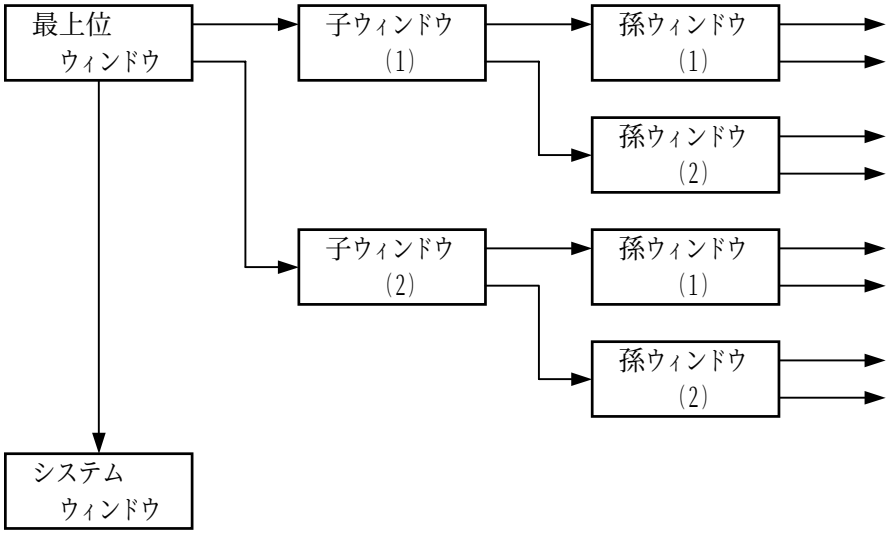
SimpleMaster ではフルスクリーンで表示される CRT 画面を 親ウィンドウ、フルスクリーンよりも小さい大きさで表示される CRT 画面を 子ウィンドウ と称しています。
［その他］は帳票用のレイアウトなどを作成するときに使用します。

スプレッドシート

表形式の画面を作成します。
（5 スプレッドシート を参照してください。）

ウィンドウの構成

SimpleMaster では次のようなウィンドウの構成を持つアプリケーションに適応することができます。



最上位ウィンドウはフルスクリーンで作成しなければなりません。
その他のウィンドウの大きさは自由に設定することができます。（フルスクリーンも可）
システムウィンドウは、バージョン情報画面、エラーメッセージ表示画面などに使用します。

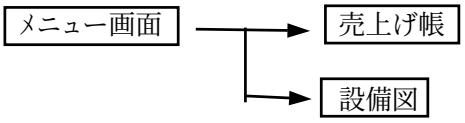
2－2－2　レイアウトの初期設定

［レイアウトの新規作成］ で 親ウィンドウ を選択すると、レイアウトの詳細設定ウィンドウが表示されます。　この時、背景にはプロジェクトで設定した親ウィンドウのテンプレートが表示されています。

レイアウトは後で簡単に修正できますので、ここでは タイトル名称 のみ設定します。
タイトル名称 はレイアウト情報を保存するときのファイル名(拡張子は ptn) に使用されます。
タイトルに 「メニュー画面」 と設定し、[起動時画面] をセット後 ［登録］ボタンを押します。

2－2－3　メニュー画面の編集

今回作成するデモ用アプリケーションの画面構成を次のようにします。

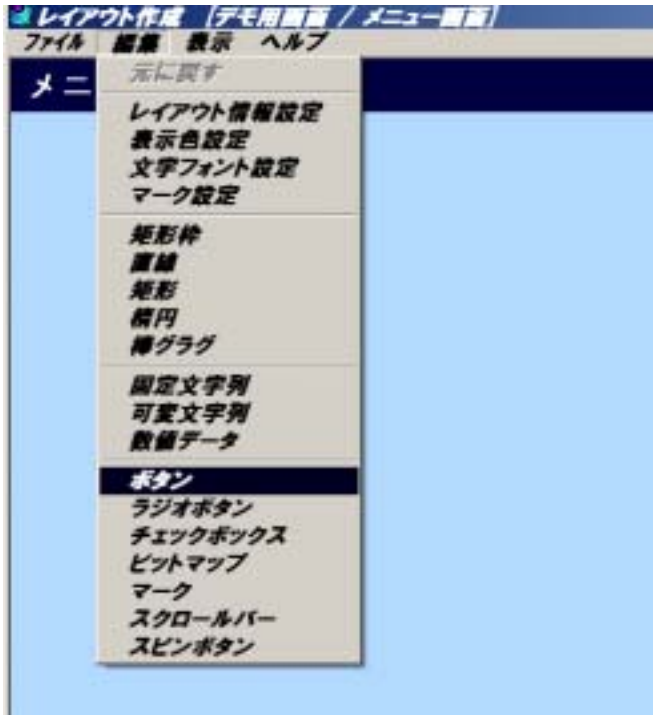


簡単な表形式の画面

制御システム用の画面

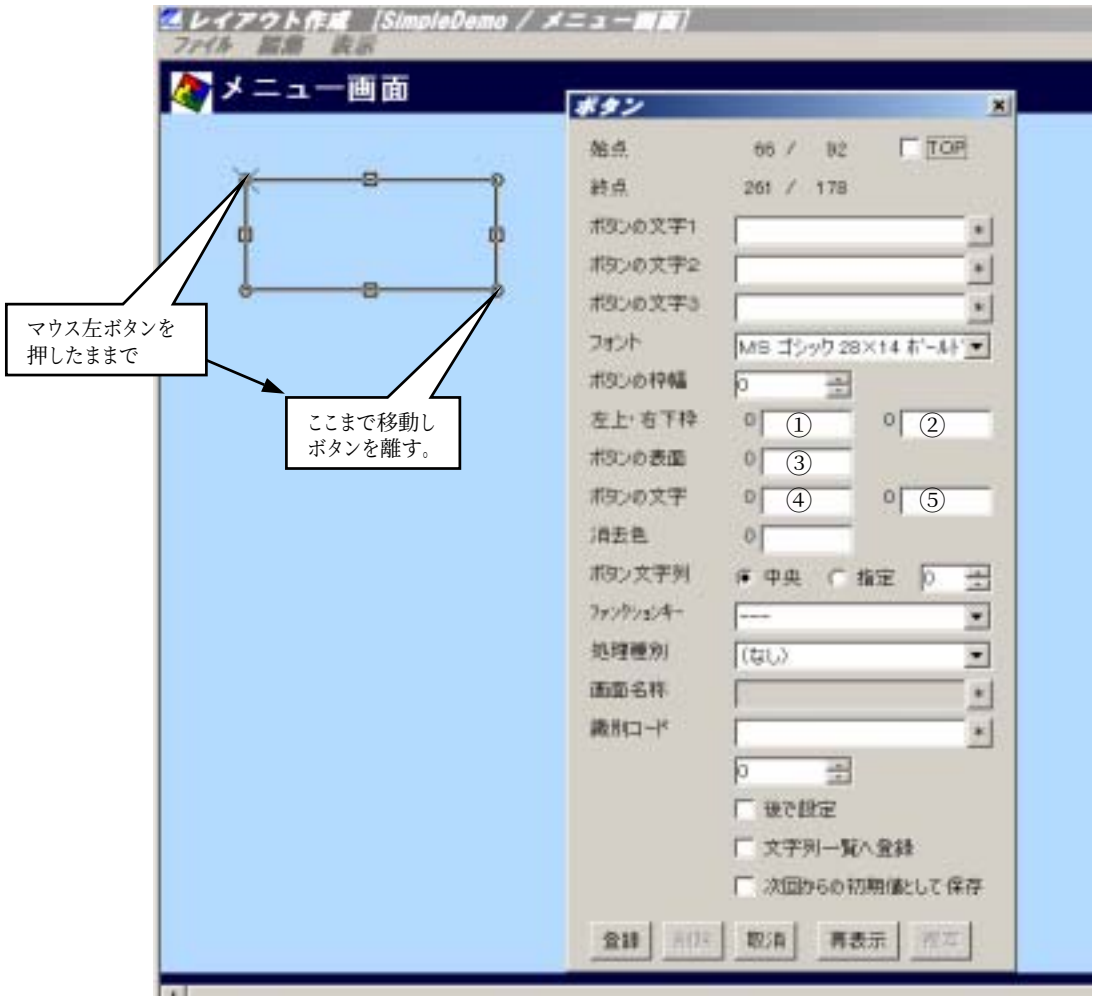
メニュー画面にはそれぞれの画面を開くためのボタンを配置します。

CRT 画面上にボタンを配置するために、［編集］メニュー → ［ボタン］ を選択します。



作業領域上でマウス右ボタンをクリックすることでも、［ボタン］を配置するためのメニューが表示されます。

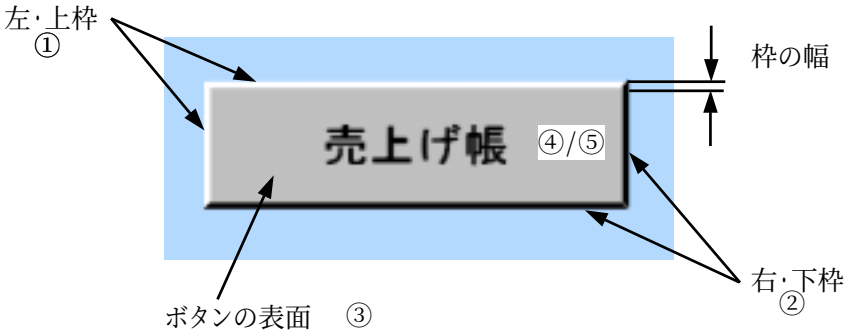
［ボタン］ を選択すると、ボタンの詳細設定画面が表示されます。



［ボタン］の表示位置と大きさを決めるため、始点でマウス左ボタンを押し、終点で離します。
表示位置や大きさは後で簡単に修正できますので、ここでは適当な位置と大きさで作成します。

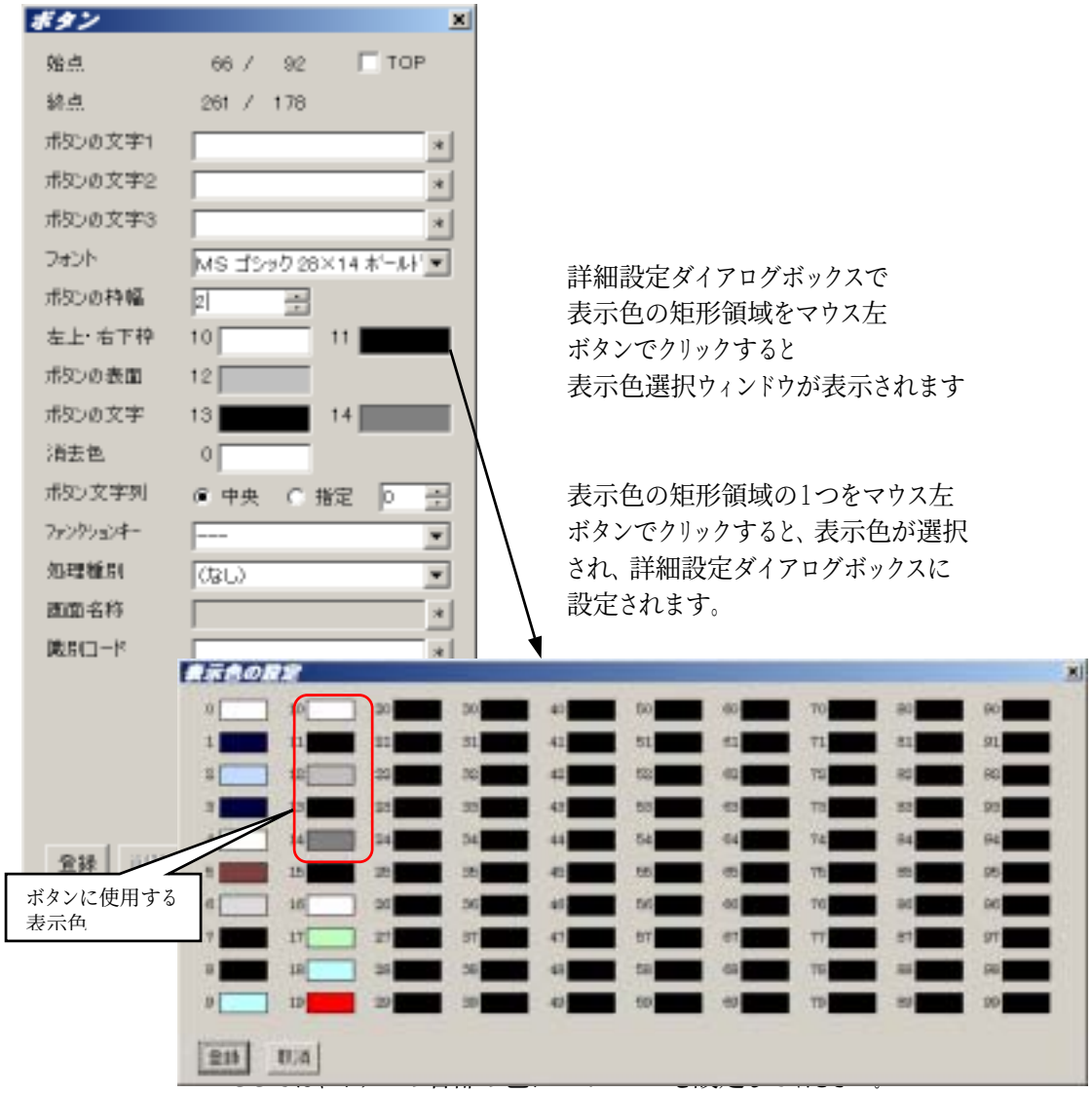
次に詳細設定ダイアログボックスへの入力を行います。
（ダイアログボックスの入力後にボタンの表示位置を設定することもできます。）

ボタンの文字に ［売上げ帳］ と入力します。
ボタンの領域の大きさに合わせてボタンの文字の表示に使用する文字フォントを選択します。
次にボタンの色などを設定します。



ボタン文字の表示色は、ボタンの操作可能時（④）、操作不可時（⑤）の2種類を設定することができます。

表示色の設定方法



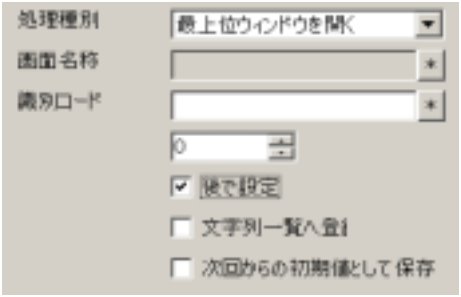
詳細設定ダイアログボックスで
表示色の矩形領域をマウス左
ボタンでクリックすると
表示色選択ウィンドウが表示されます

表示色の矩形領域の1つをマウス左
ボタンでクリックすると、表示色が選択
され、詳細設定ダイアログボックスに
設定されます。

処理内容の設定

つぎに、ボタンをマウス左ボタンでクリックした時の処理を設定します。

ここをマウス左ボタンで
クリックすると処理項目
の一覧が表示されます。



ボタンを押したとき、売上げ帳画面を開くために
［最上位ウィンドウを開く］ を選択します。

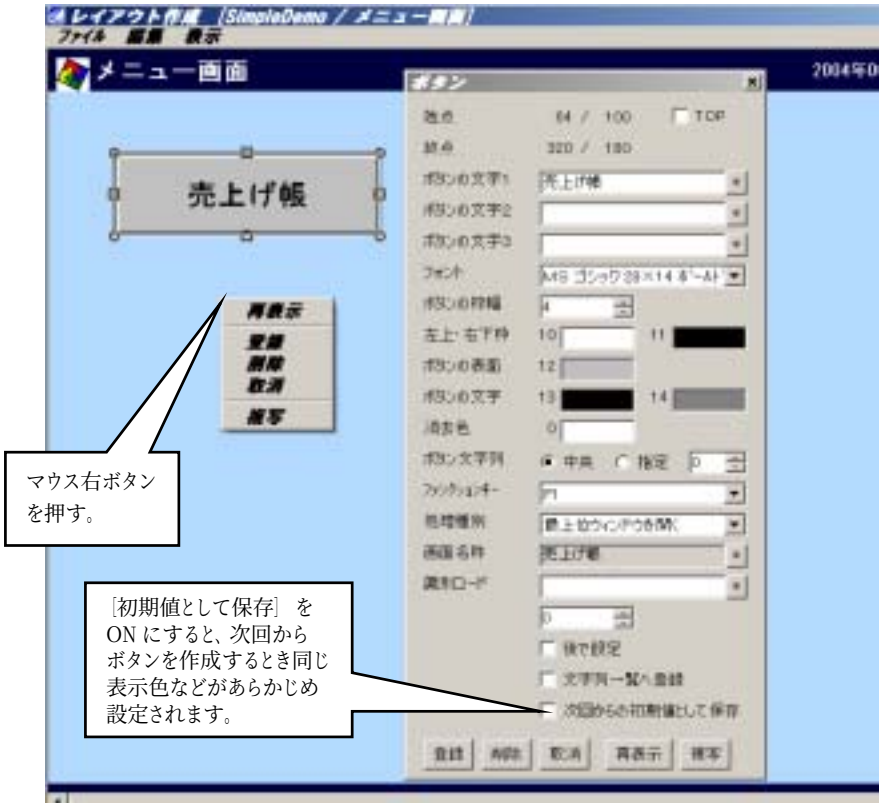
次に画面名称を設定しなければなりませんが
現時点では画面を作成していませんので
［後で設定］ を ON にしておきます。

設定内容の登録

作業領域上でマウス右ボタンを押すとポップアップメニューが表示されます。

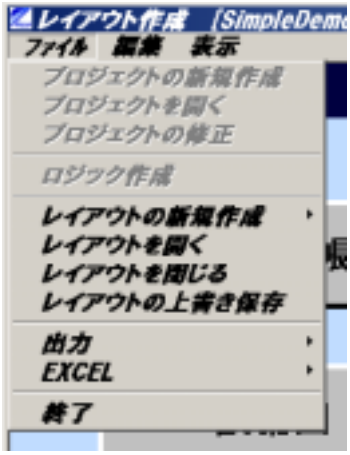
ここで、[登録]を選択すると作成したボタンが登録されます。

詳細ダイアログボックス下端の[登録]ボタンで登録することもできます。



[売上げ帳]ボタン と同じ手順で [設備図] のボタンを作成します。
また、アプリケーションプログラムを終了させるため、[終了]ボタンを作成し、処理種別に[システム 終了] を設定します。

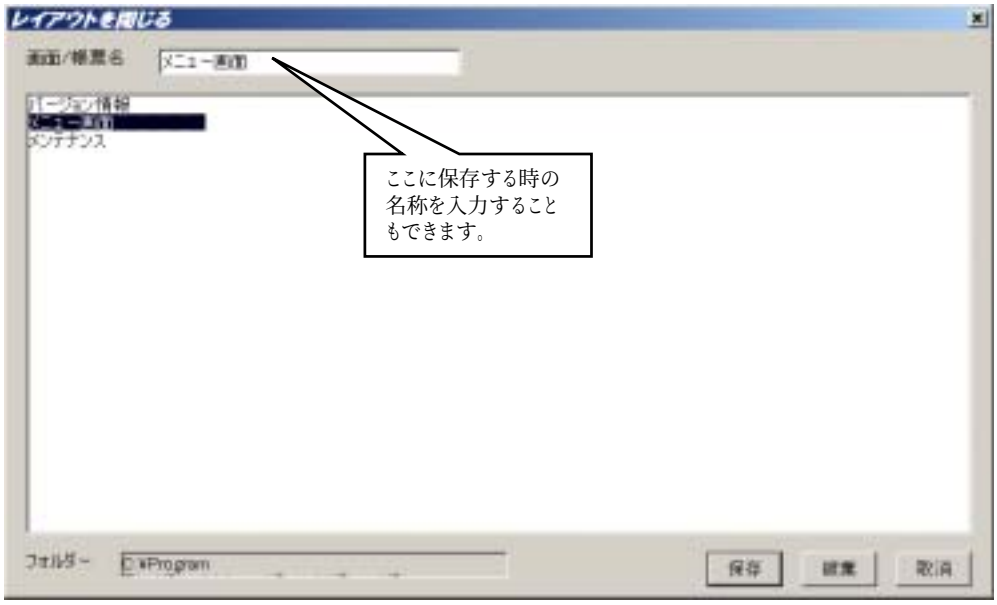
2－2－4　メニュー画面の登録



編集が終わったメニュー画面を登録します。

［ファイル］メニュー → ［レイアウトを閉じる］を選択します。

レイアウト保存用の画面が表示されます。
［保存］ボタンを押してレイアウト情報をハードディスクに
保存します。



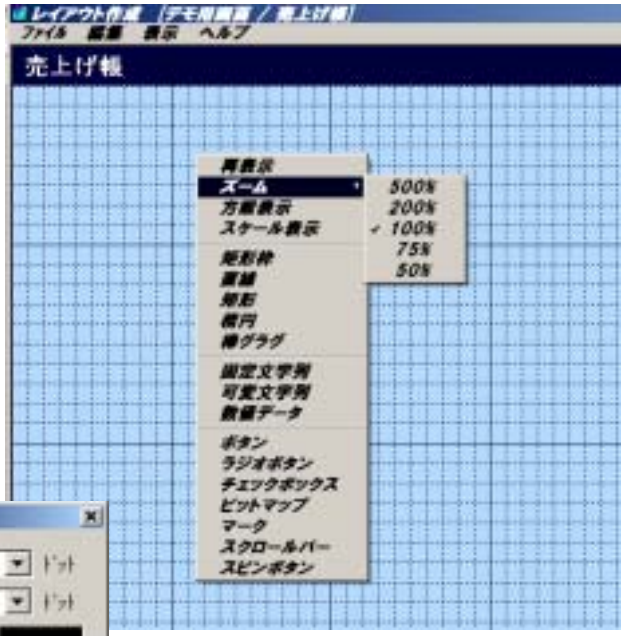
レイアウト情報を保存するとき、ファイル名として画面のタイトル、拡張子として ptn が使用されます。
同じタイトルの画面が複数存在する場合などでは、ファイル名を修正することもできます。
ファイルの保存先はプロジェクト設定時に指定されたフォルダーが使用されます。

2－3　売上げ帳画面の作成

［売上げ帳］ 画面を作成します。

メニュー画面の作成時と同様に ［ファイル］メニュー → ［新規作成］ → ［親ウィンドウ］ を選択し、タイトルに ［売上げ帳］ と設定します。

レイアウト編集時に、方眼を表示したり、拡大･縮小表示を行うことができます。
マウス右ボタンメニュー または ［表示］ メニューで設定してください。



［売上げ帳］ では表形式の画面を作成しますので、罫線やデータの表示位置を等間隔に配置することになります。
SimpleMaster では表示位置は方眼に合わせて調整されます。

方眼の間隔は ［表示］メニュー → ［方眼設定］ で任意の大きさに設定することができます。

2－3－1　表の作成

表の作成には 矩形枠 を使用します。
矩形枠 は塗りつぶされた矩形と上下左右の罫線で構成されます。
矩形枠 は方眼の内側に描画されるので上下左右に複数個配置することで表を作成することができます。
矩形枠の内部には、固定/可変文字列、数値データを配置します。

売上げ帳 の表示項目は次のとおりとします。

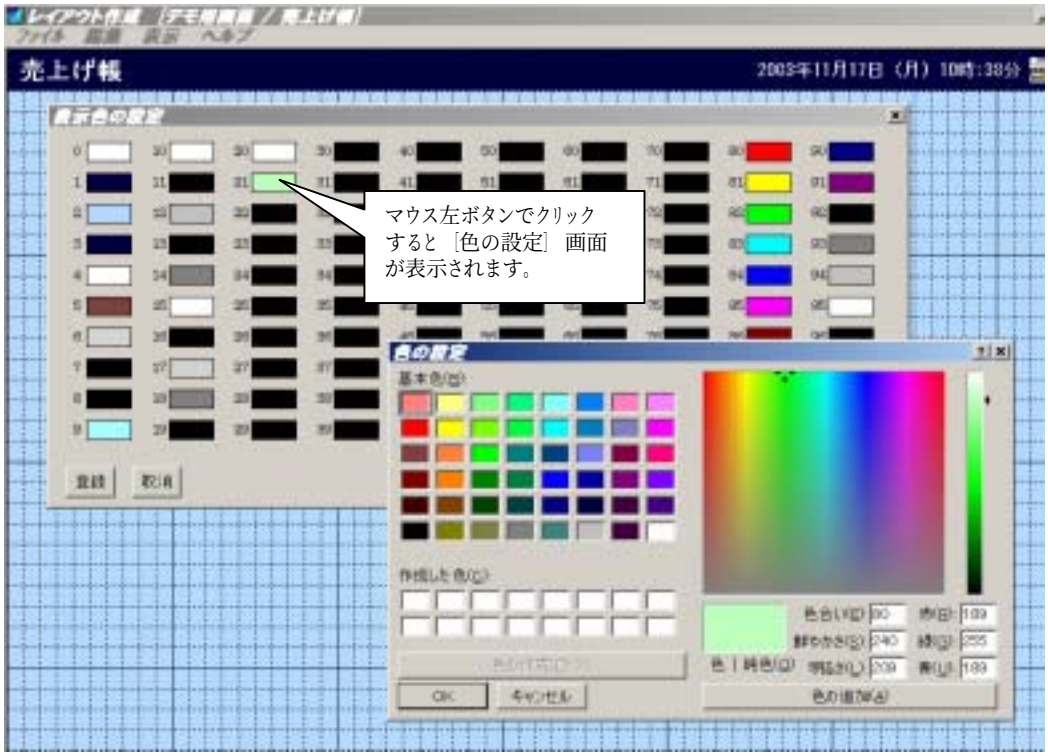
No.	項目名	データ形式	桁数
1	日付	可変文字列	7
2	品名	可変文字列	24
3	単価	数値データ	8
4	数量	数値データ	8
5	合計	数値データ	10

2－3－2　表示色 の設定

売上げ帳では次の表示色を使用します。

表示色番号	項目名
１０	ボタン 左・上枠 ／ スクロールバー 左・上枠
１１	ボタン 右・下枠 ／ スクロールバー 右・下枠
１２	ボタン 表面　　／ スクロールバー 内部
１３	ボタン 文字　　／ スクロールバー スライダー
１４	ボタン 文字（操作不可）
２０	項目名称 文字
２１	データ　文字
２２	項目名称 背景
２３	データ　背景

［編集］メニュー → ［表示色設定］ を選択し表示色を設定します。



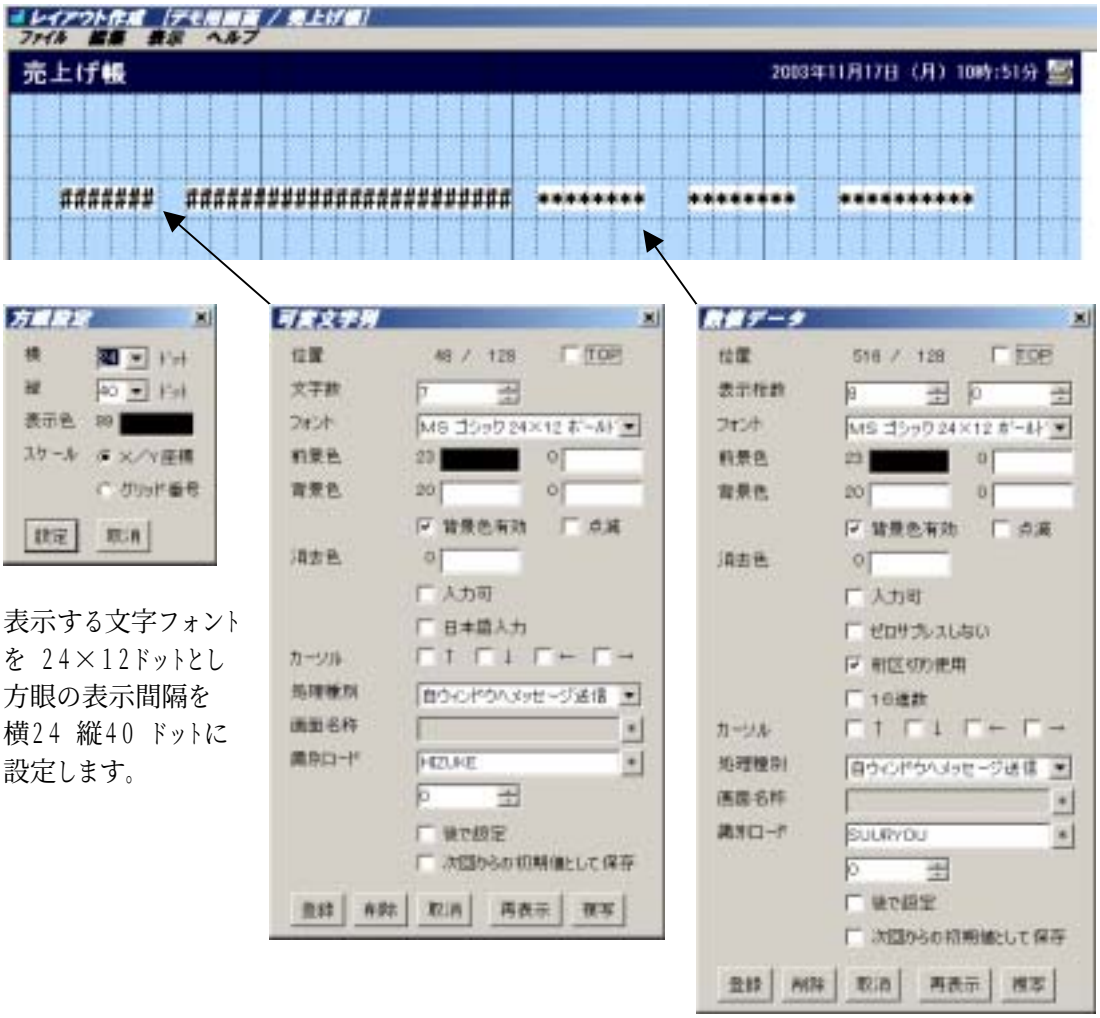
SimpleMasterをデフォルトの状態で起動すると、表示色番号 0～19には次の目的で使用する表示色が設定されています。

番号	項目名
0	親ウィンドウ タイトル領域 文字の表
1	親ウィンドウ タイトル領域 背景色
2	親ウィンドウ 作業領域　　背景色
3	親ウィンドウ ボタン領域　背景色
4	子ウィンドウ タイトル領域 文字の表
5	子ウィンドウ タイトル領域 背景色
6	子ウィンドウ 作業領域　　背景色
7	子ウィンドウ 枠の表示色
8	キーボード入力時 文字の表示色
9	キーボード入力時 文字の背景色

番号	項目名
１０	ボタン/スクロールバーで使用
１１	ボタン/スクロールバーで使用
１２	ボタン/スクロールバーで使用
１３	ボタン/スクロールバーで使用
１４	ボタン/スクロールバーで使用
１５	メンテナンス画面で使用
１６	メンテナンス画面で使用
１７	メンテナンス画面で使用
１８	メンテナンス画面で使用
１９	エラーメッセージで使用

2－3－3　表示位置の設定

1行分のデータを画面上に配置し、表示位置を決めます。



表示する文字フォントを 24×12ドットとし方眼の表示間隔を横24 縦40 ドットに設定します。

［編集］メニュー →［可変文字列］ または ［数値データ］を選択します。
表示開始位置 （文字列の左上位置を指定）をマウス左ボタンで指定します。
詳細設定ダイアログボックスに次の項目を設定します。

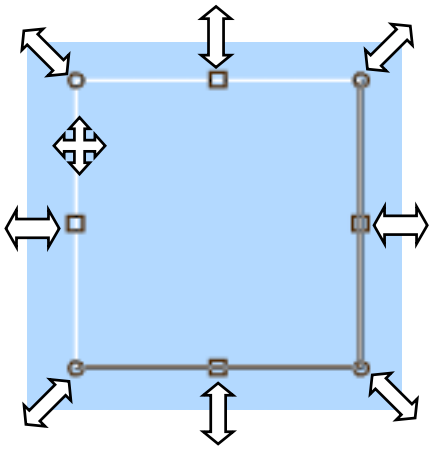
　　文字数 または 桁数
　　表示色 （前景色、 背景色、 背景色有効）

各項目の表示データを画面上に配置します。 表示位置は簡単に修正できますので、ここでは大まかな位置決めを行います。

表示位置は次のような手順で修正することができます。

① マウスを修正しようとするデータの境界部分に移動します。
② マウスカーソルの形状が ［矢印］ から ［十字］ に変化します。
③ マウスカーソルの形状が ［十字］ の時、マウス左ボタンをｸﾘｯｸします。
④ 修正しようとする項目が矩形で囲まれ、詳細設定ダイアログメッセージが表示されます。

選択された描画要素の境界付近にマウスカーソルを移動すると、マウスカーソル形状が十字から右図のように変化します。



マウスカーソル形状が右図のように変化したときマウス左ボタンを押したままでマウスを移動するとカーソルの方向に従って図形が拡大･縮小されます。
所定の大きさに達したときマウス左ボタンを離すと図形の大きさが確定します。

のカーソルが表示されているときは、大きさは変化せず表示位置のみを移動させることができます。

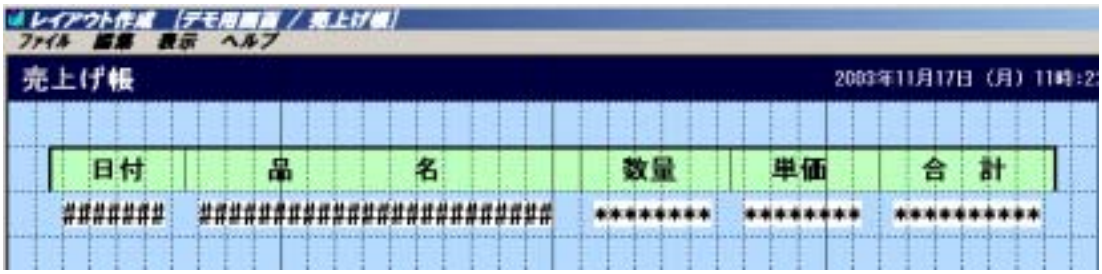
図形の表示位置と大きさは ｸﾞﾘｯﾄﾞ線の間隔の 1/2 で揃えられます。

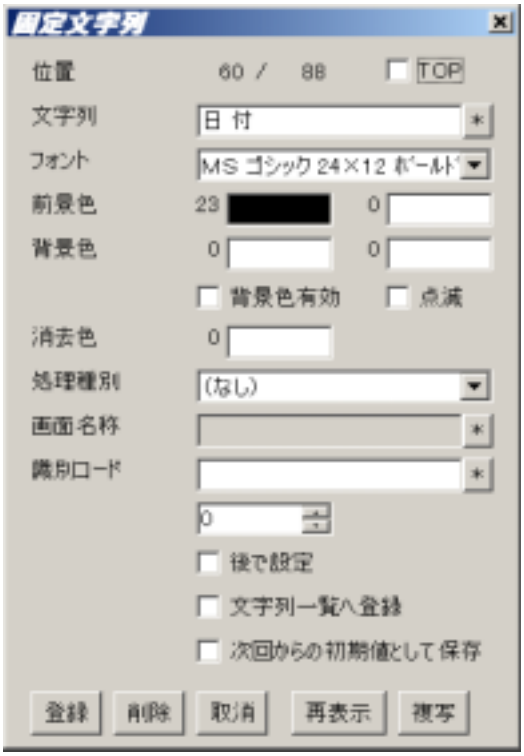
固定文字列などでは、 のカーソルが表示され、表示位置の変更のみが可能です。
表示位置はｸﾞﾘｯﾄﾞ線の間隔で揃えられます。
この状態で、移動、図形の大きさの変更、詳細設定の変更 などを行い、マウス右ボタンメニューを開いて登録します。
修正が不要な場合は、マウス右ボタンメニューで ［再表示］ または ［取消］ を選択します。
データが表示されていない位置でマウス左ボタンをｸﾘｯｸすることで取消すこともできます。

ＣＲＴ画面上に配置する図形や文字列は方眼に合わせて表示位置が調整されます。 微妙な位置合わせを行う場合には方眼の間隔を変更してください。 （表示メニュー → 方眼設定）

2－3－4　項目名称の配置

矩形枠と固定文字列を使用して項目名称をＣＲＴ画面上に配置します。

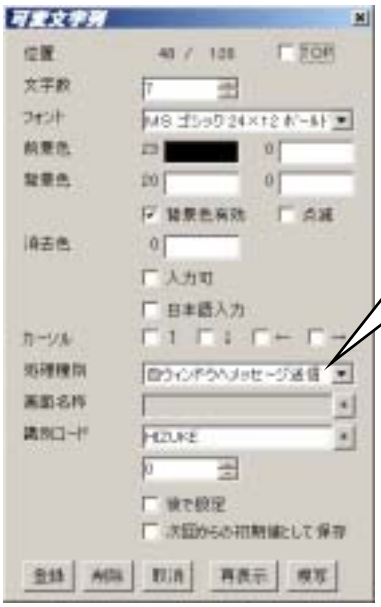




矩形枠の枠幅は 左端／右端／それ以外
で変えています。

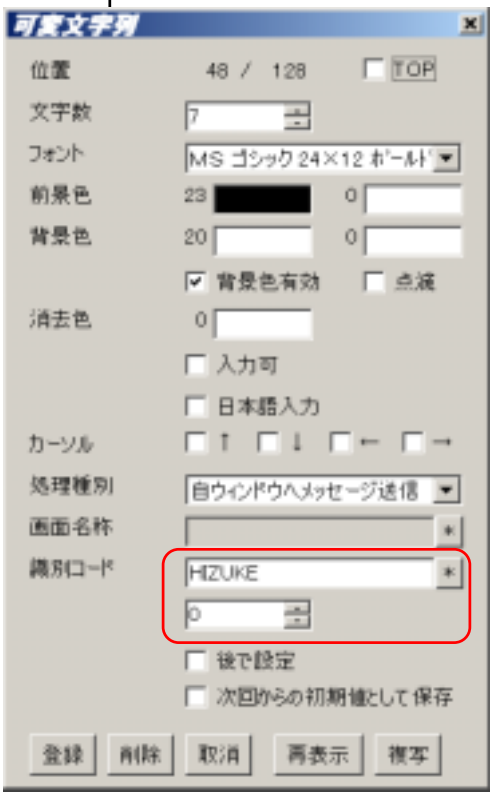
2－3－5　データ部の詳細設定

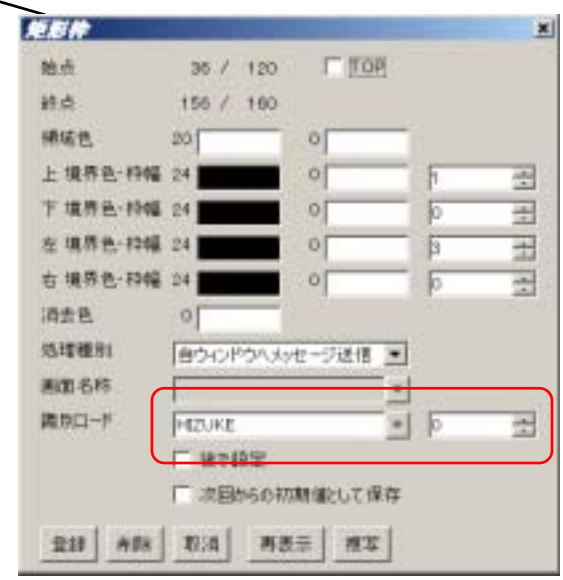
可変文字列、数値データは表示する内容を決めるために識別コードを設定します。
また、マウス左ボタンでクリックされたとき、ロジック部にメッセージを送ることにします。
ロジック部ではクリックされた項目を判定し入力用ウィンドウを開きます。



売上げ帳 の表示、入力処理を行うプログラム
へメッセージを送ります。

識別コード　ロジック部で変数名として使用します。
英数字、"_"(アンダースコア)で設定
してください。
先頭は英字、最大２０文字で設定して
ください

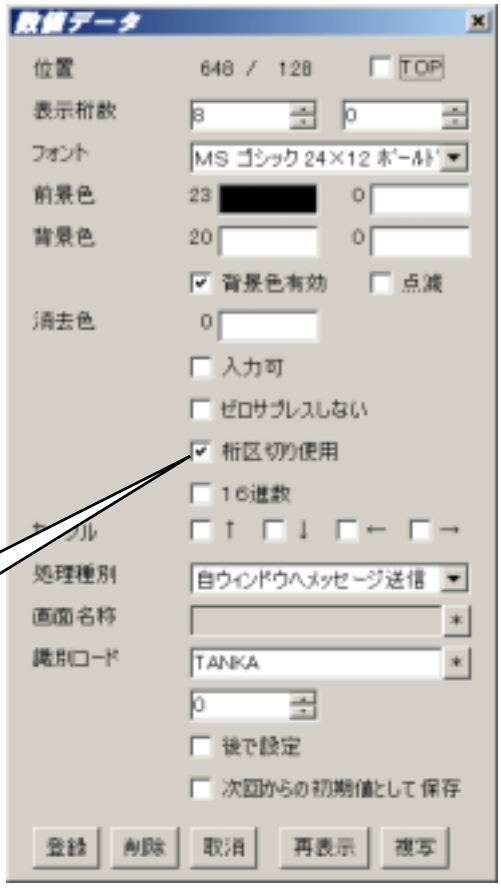




可変文字列と矩形枠の識別コード
には同一の文字列を設定します。

識別コードは次のように設定します。

項目名	識別コード
日付	HIZUKE
品名	HINMEI
単価	TANNKA
数量	SUURYOU
合計	GOUKEI



数値データは3桁ごとに
区切り(カンマ)を挿入します。
(カンマも文字数に加えます。)

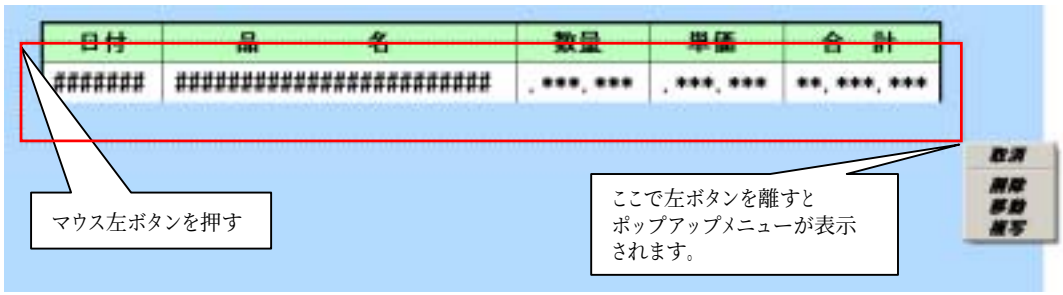
2－3－6　表の完成

データ部を下方向に複写して表を完成します。
複写には2種類の方法があります。

個別複写

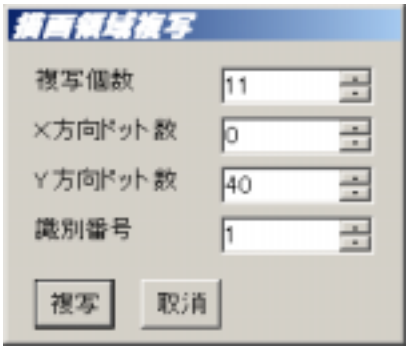
データの1項目のみを選択し
マウス右ボタンメニューで
［複写］を選択します。
詳細設定ダイアログボックス下端
の ［複写］ ボタンでも同様に個別
複写を行うことができます。

一括複写



描画要素を選択していない状態で、複写の対象となる項目を囲む矩形を選択します。
固定文字列 日付 のように一部のみしか矩形領域に含まれていない描画要素は複写の対象とはなりません。　複写以外に、移動、削除を行うことができます。

ここでは、一括複写を使用して表を完成させます。
ポップアップメニューで複写を選択すると、詳細設定ウィンドウが表示されます。



複写個数　　11行複写し、合計12行表示できるようにします。

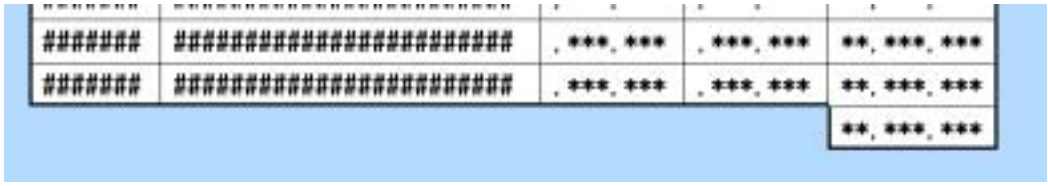
Y方向ドット数　　複写の間隔を40ドットとします。
－xx と設定すると、上方向へ複写します。

識別番号　　複写するとき、識別コードの番号部を ＋1 して行きます。
1行目のデータは　　HIZUKE+0
12行目のデータは　　HIZUKE+11 として識別できるようになります。

複写後

複写により表がほぼ完成しました。
最下行の罫線の太さなどを調整すれば
完成です。

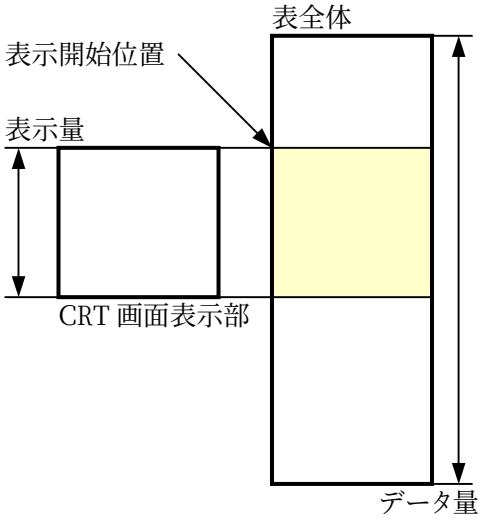
罫線の太さ調整、総合計の欄を追加して完成



2－3－7　スクロールバーの配置

SimpleMaster ではスクロールバーを部品として CRT 画面上に配置することができます。

スクロールバーでは、データの一部を CRT 画面上に表示するときに、表示位置を決める手段として使用します。



SimpleMaster ではスクロールバーを配置するとスクロールバーを操作されたとき、ロジック側へメッセージを送ります、ロジック側では表示開始位置を取得し再表示を行います。

ダブルボタン
シングルボタン
スライダー

表示量　　表を作成したときの値（12行）を設定します。

データ量　　データ全体の大きさが固定の場合には数値を設定できますが、実行時にプログラム側より設定することもできます。
今回はプログラムで設定します。

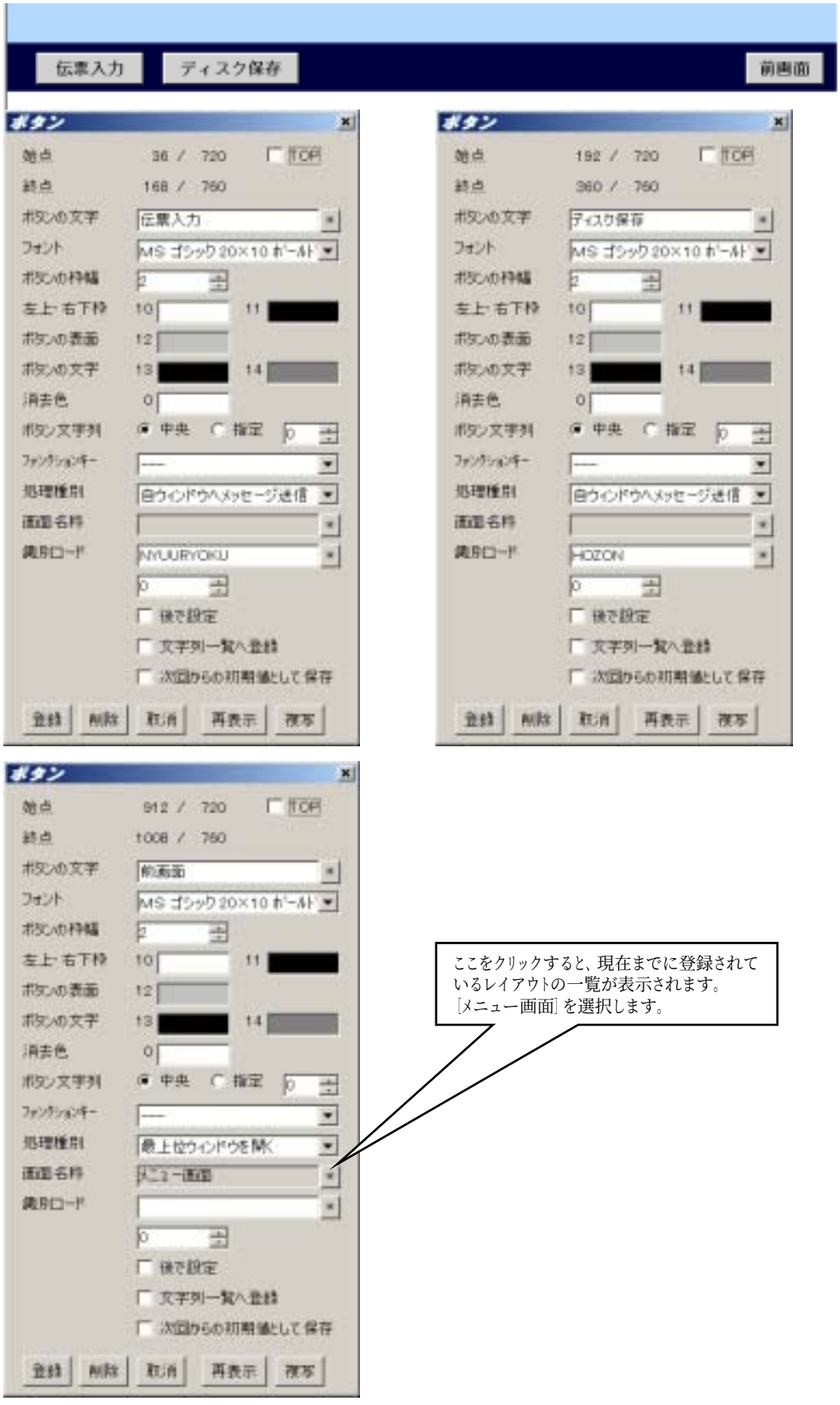
移動量　　左より シングルボタン / ダブルボタン / ボタン⇔スライダー間をマウス左ボタンでクリックした時の表示開始位置の移動量を設定します。

消去色　　表示量 ＞ データ量 の時スクロールバーは非表示となります。 この時に使用する消去色（作業領域の背景色）を設定します。
作業領域の背景色はレイアウトの初期設定で設定しています。

2－3－8　ボタンの配置

次のボタンを配置します。

［伝票入力］　　入力ウィンドウを開きます。
［ディスク登録］　　データをハードディスクに保存します。
［前画面］　　メニュー画面に戻ります。

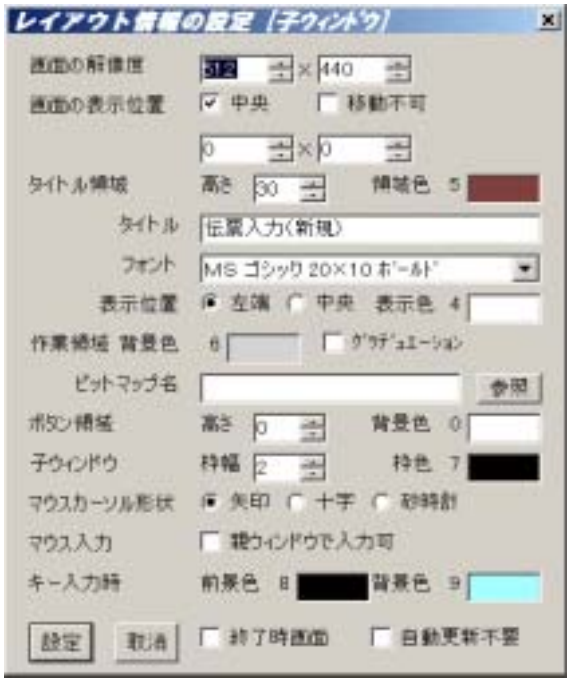


2－4　伝票入力画面の作成

［売上げ帳］ 画面での 入力画面 を作成します。

2－4－1　伝票入力画面の作成

メニュー画面の作成時と同様に ［ファイル］メニュー → ［新規作成］ → ［子ウィンドウ］ を選択し、タイトルに ［伝票入力(新規)］ と設定します。



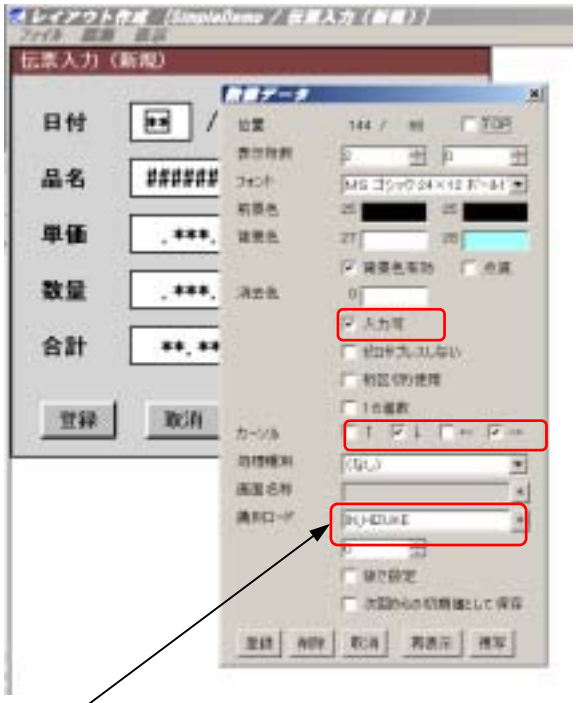
画面の大きさはレイアウト作成途中で調整しますのでここでは適当な値を設定します。

固定文字列、可変文字列、数値データ、矩形(キーボード入力域)、ボタン を使用してレイアウトを作成します。

左右、上下の余白を調整し画面の大きさを確定します。

2－4－2　キーボード入力の設定

可変文字列、数値データの詳細設定ダイアログボックスで［入力可］をセットするだけでその項目のキーボード入力が可能となります。



キーボード入力は、ロジック側で記述することにより、継続させることができます。
（例）月　の入力が終わると続けて
　　日　の入力を行う。

カーソルキー　をセットすれば、右図の場合［↓］または［→］方向のカーソルキーのみを受付け、プログラム側の記述により入力を継続させることができます。

キーボード入力可能な数値データを囲む矩形領域に、数値データと同一の識別コードを割当てることができます。
この場合、矩形領域の内部をマウス左ボタンでクリックすると同一の識別コードが設定された数値データの入力処理が開始されます。
（可変文字列も同様です。）

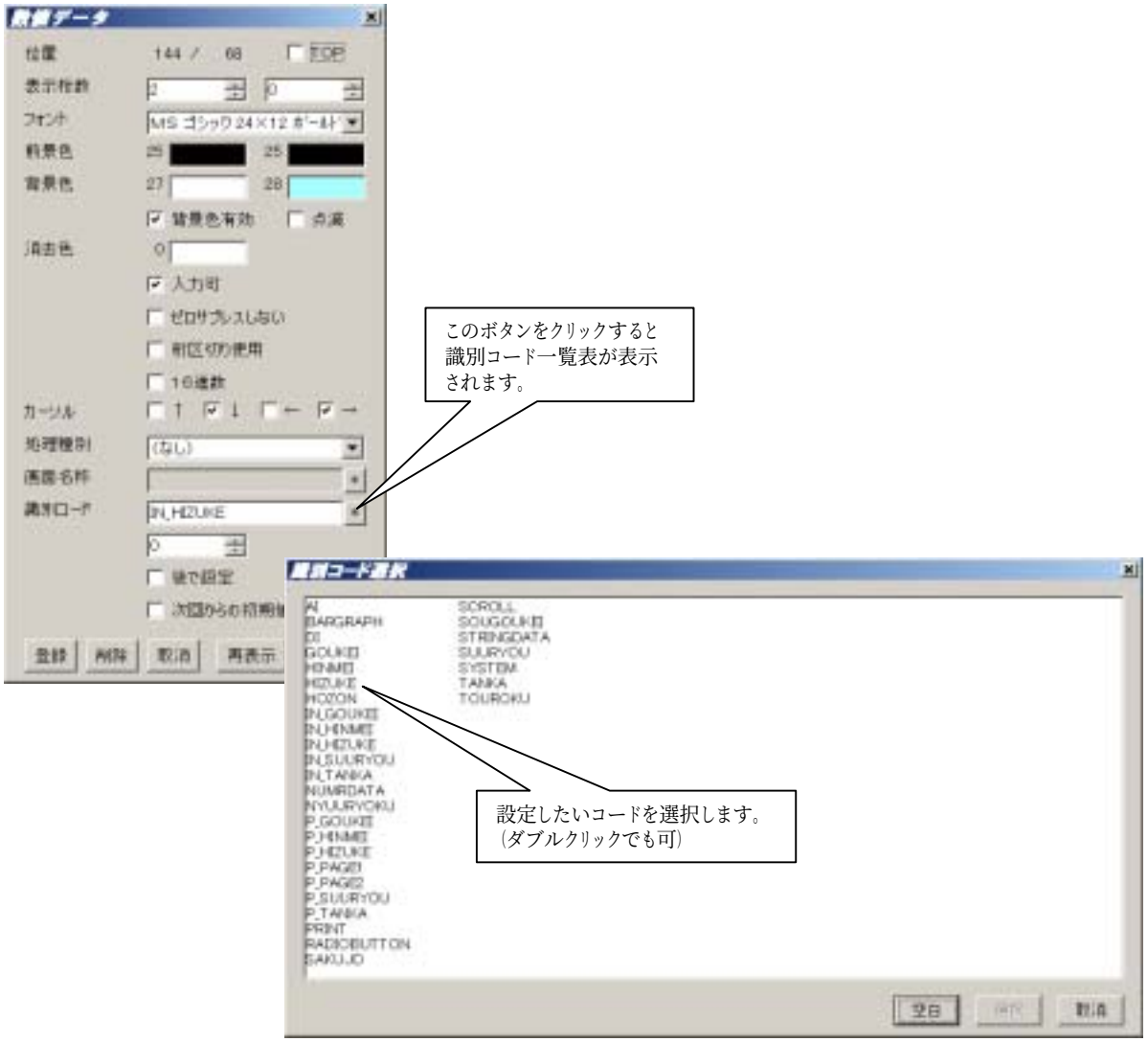
各項目には次の識別コードを設定します。

項目名称	識別コード	備考
日付（月）	IN_HIZUKE1	入力可
日付（日）	IN_HIZUKE2	
品名	IN_HINMEI	
単価	IN_TANKA	
数量	IN_SUURYOU	
合計	IN_GOUKEI	入力不可

（合計値はプログラムで計算します。）

識別コードの設定

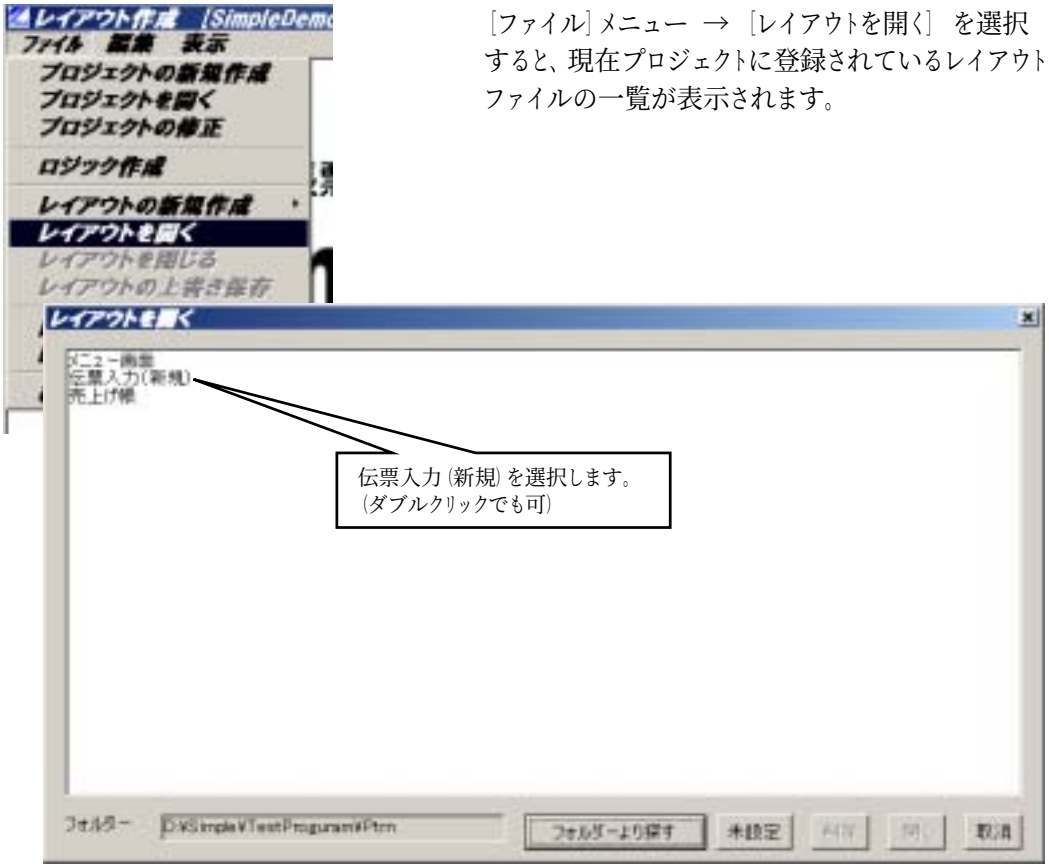
レイアウト作成時に設定された識別コードはプロジェクト単位で保存されています。
簡単なマウス操作で過去に設定された識別コードより選択することができます。



ボタンの詳細設定

2－4－3　伝票入力(修正)画面の作成

伝票入力(修正)画面 は 伝票入力(新規)画面 を複写して作成します。



［ファイル］メニュー → ［レイアウトを開く］ を選択すると、現在プロジェクトに登録されているレイアウトファイルの一覧が表示されます。

伝票入力(新規) で次の項目を変更します。

① 画面のタイトルを 伝票入力(修正) に変更します。 (レイアウト情報設定)

② ［取消］ボタンの位置を右に移動し、［削除］ボタンを追加します。

［ファイル］メニュー → ［閉じる］ または ［上書き保存］ でディスクに保存します。

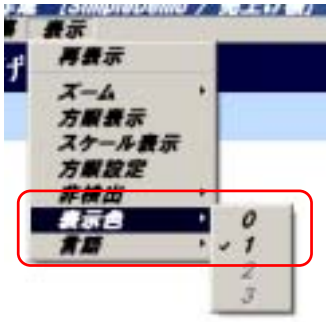
ファイル名はタイトルを修正したので、［伝票入力(修正)］ となり、［伝票入力(新規)］はそのまま保存されています。

2－4－4　エラーメッセージ表示

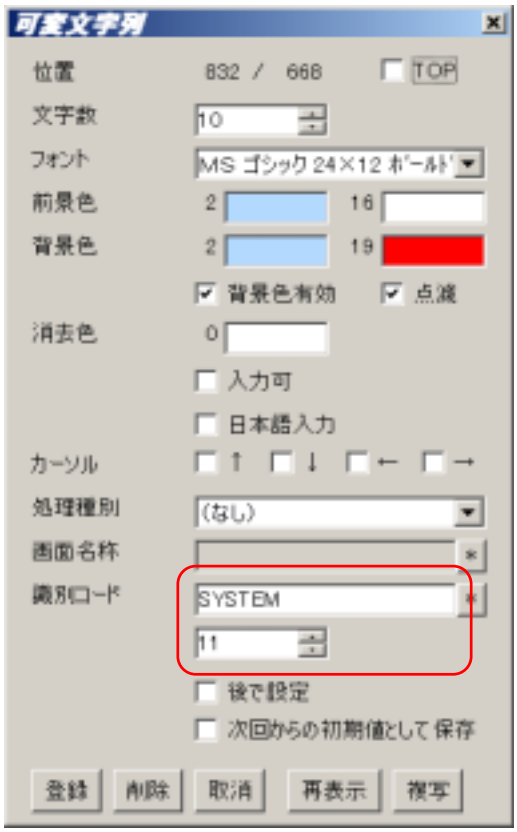
SimplePack では数値データ入力時に上下限値の検査を行い、上下限の範囲外の数値が入力された場合、エラーメッセージを表示することができます。

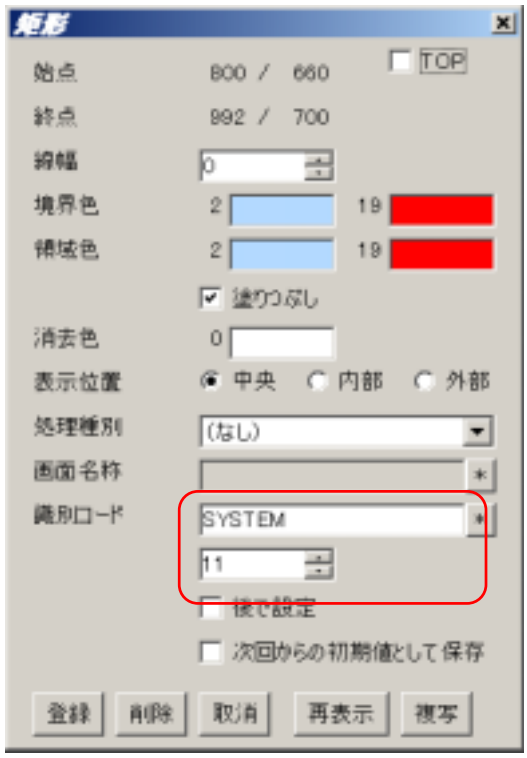
レイアウト側の準備

売上げ帳画面にエラーメッセージを表示する領域を設定します。



CRT 画面下部にエラーメッセージ表示領域を作成します。
上記の画面例は 表示色番号1 の場合です。





エラーメッセージ表示領域として、矩形と可変文字列を定義します。

表示色／背景色
　通常時は背景色で表示されるように設定します。
　エラー発生時には、エラーメッセージを白、背景を赤で表示します。

識別コード
　SYSTEM+11 を設定します。　この識別コードはエラーメッセージ表示用に予約されています。

可変文字列の文字数
　文字列設定ファイル（次ページ参照）で設定した文字列が表示できる文字数を指定します。

Microsoft Excel ファイルの設定

数値データ入力時の入力可能範囲、およびエラーメッセージとして CRT 画面上に表示する文字列は Microsoft Excel 形式のファイルで設定します。
雛形となる Excel 形式のファイルはプロジェクト作成時に次のフォルダーへ複写されています。
（プロジェクト情報格納先フォルダー）＋ \Excel

LimtData.xls　　上下限値設定ファイル

No	識別コード			下限値	上限値	エラー番号	
	名称	番号(1)	番号(2)			下限	上限
1	IN HIZUKE	0	0	1	12	1	2
2	IN HIZUKE	1	1	1	31	1	2
3	IN SUURYOU	0	0	1	9999	1	2
4	IN TANKA	0	0	1	99999	1	2
5							
6							
7							

識別コードに対応して入力範囲、入力エラー発生時のエラーメッセージ番号を設定します。
メッセージ番号は MesgData.xls 左端の番号に相当します。

MesgData.xls　　文字列設定ファイル

No	識別コード		共通文字列（言語１）	共通文字列（言語２）	共通文字列（言語３）
	名称	番号			
1			下限エラー		
2			上限エラー		
3					
4	SYSTEM	15	SimpleMaster V1.01		
5	SYSTEM	16	デモ用プログラム		
6	SYSTEM	17			
7	SYSTEM	18	㈱ フジテクニカ		
8					
9					

多国籍言語に対応して文字列を設定することができます。
番号 4～7 はバージョン情報画面で参照しています。

2－5　帳票レイアウトの作成

売上げ帳を印刷するためのレイアウトを作成します。

2－5－1　レイアウトの新規作成

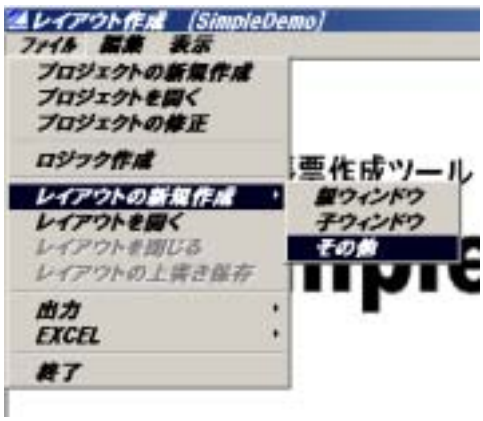
帳票用レイアウトはＣＲＴ画面用レイアウトとほぼ同じ手順で作成することができます。
ＣＲＴ画面用レイアウトと異なる点は

① 用紙の大きさ　画面の大きさとは異なり、任意の大きさを指定することができます。縦横比を実際の用紙に合わせておくことで良好な印字結果が得られます。

② 表示色　モノクロプリンターを使用する場合には、白･黒･灰色 のみを使用してください。 カラープリンターを使用する場合には、レイアウトで設定された色で印字されます。

レイアウトの新規作成

［ファイル］メニュー →［レイアウトの新規作成］
→［その他］ を選択します。



レイアウト情報の設定

［新規作成］ を選択するとレイアウト情報の詳細設定画面が開きます。

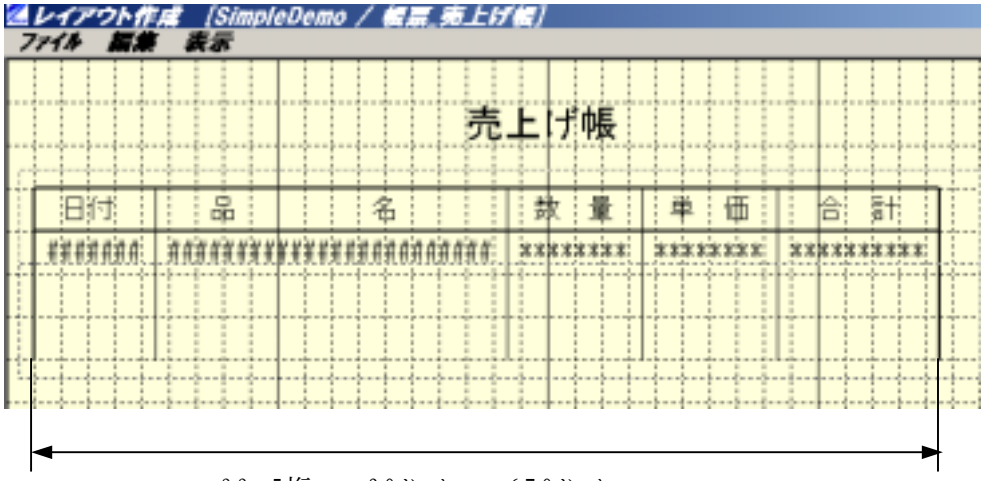
レイアウトの大きさに仮の値 800×600 を設定します。（適当な数値で可）

レイアウトのファイル名を設定し［登録］ します。

（注）　レイアウトの大きさをわかりやすくするため背景色を設定しています。

2－5－2　レイアウトの大きさの決定

横方向の大きさを決めるため、項目名称、1行分の印字データを作業領域に配置します。



33．5桁 × 20ドット ＝ 670ドット

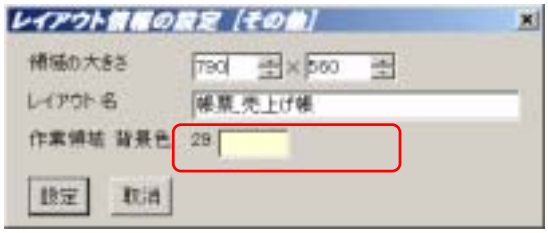
左右の余白を 3桁（60ドット） とすると
横方向の大きさは 39．5桁（790ドット） となります。

つぎに、縦方向の大きさを決めます。

実際の用紙の大きさ	A4 の場合	297× 210 mm
縦横比	210 ／ 297 ＝ 0．7070	
縦方向の大きさ	790 × 0．7070 ＝ 558．6 → 560ドット	

［編集］メニュー → ［レイアウト詳細設定］ により、
レイアウト情報の詳細設定画面を開きます。

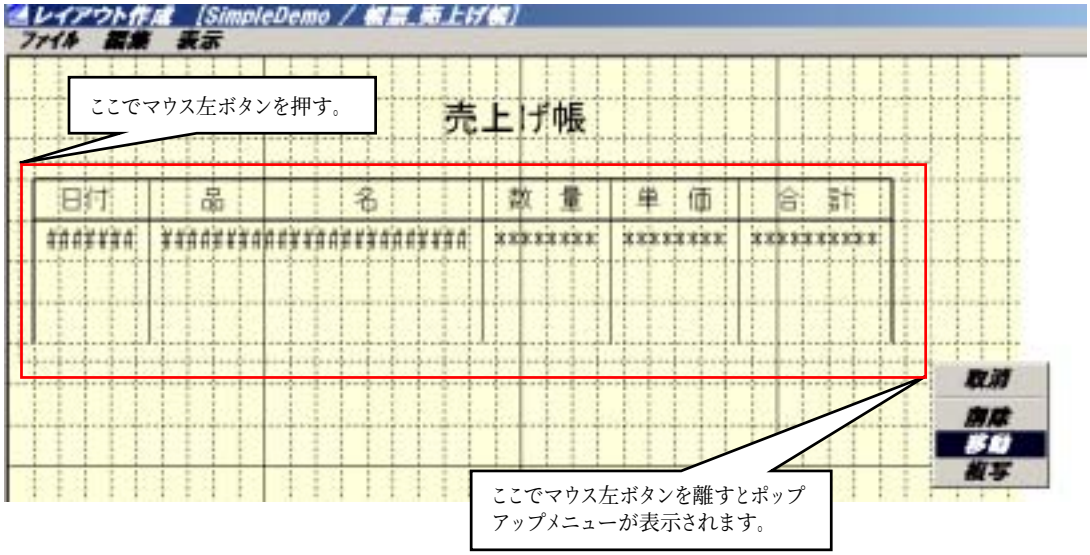
横方向の大きさを 800 → 790 に変更します。
縦方向の大きさを 600 → 560 に変更します。



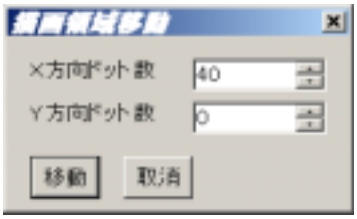
［注］　レイアウトの大きさを実際の用紙の縦横比に合わせることで
拡大印字した時良好な印字結果が得られます。

2－5－3　左右余白の調整

作業領域の大きさを設定し、描画要素の一括移動機能を使用して左右の余白を調整します。



詳細設定ダイアログボックスで 移動量 40ドット を設定し一括移動を行います。



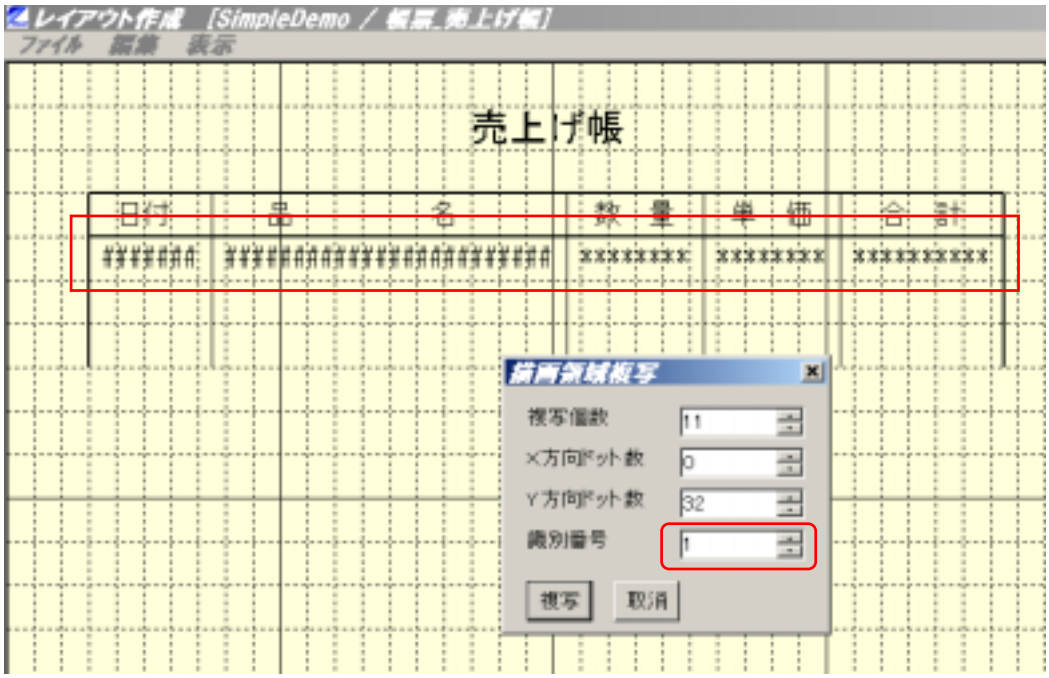
移動後の帳票レイアウト

2－5－4　識別コードの設定と複写

日付などの描画要素に識別コードを設定します。

日付	P_HIZUKE
品名	P_HINMEI
数量	P_SUURYOU
単価	P_TANKA
合計	P_GOUKEI

一括複写機能を使用して描画要素を複写します。

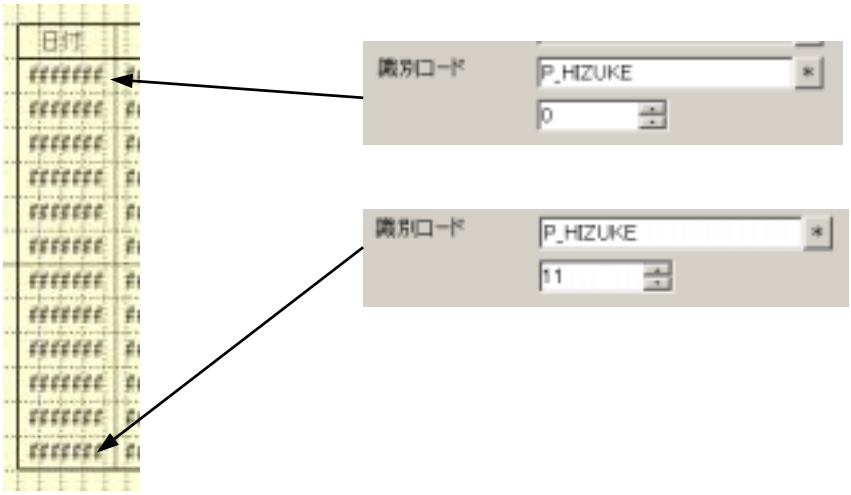


一括移動処理時と同様に、複写の対象となる描画要素を矩形で囲み、複写を選択します。
複写の対象となるのは、図形全体が矩形の内部となる描画要素のみで、上図のように、日付などの固定文字列や罫線は一部が矩形の外部となるため複写の対象とはなりません、

詳細設定ダイアログボックスで、複写個数、間隔を設定します。
識別番号に 1 を設定すると複写時に識別コードの番号部が ＋1 加算されます。

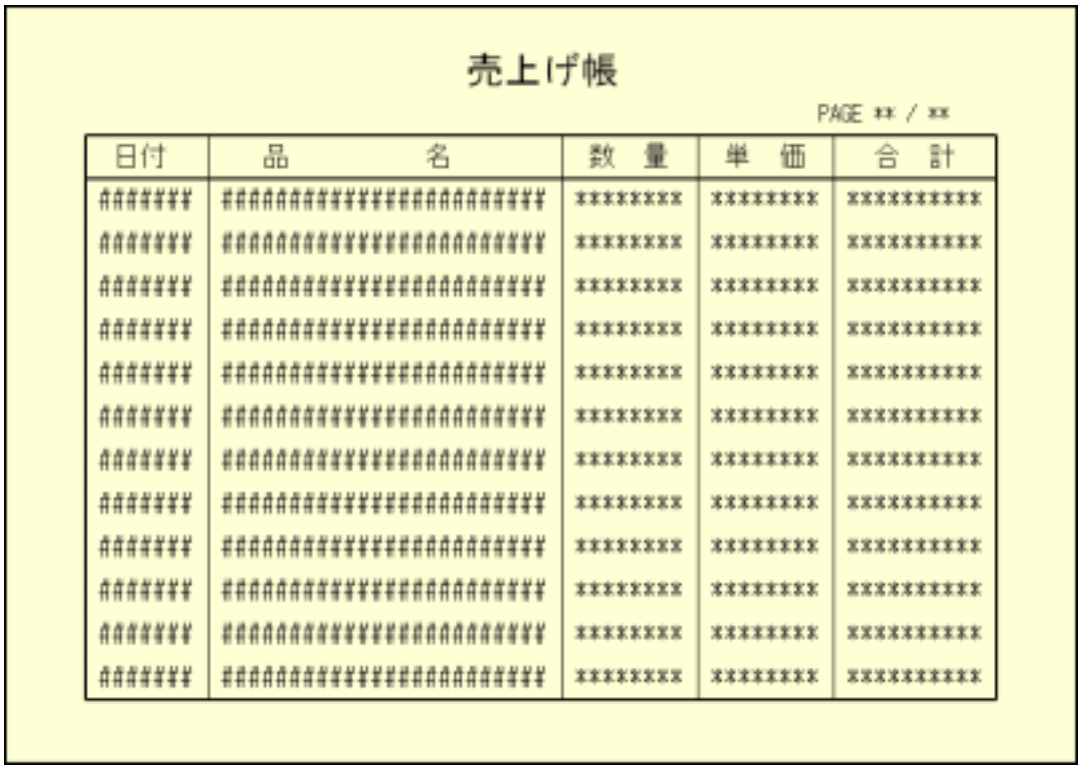
日付データの場合、

1行目の識別コードは　P_HIZUKE＋　0
最下行の識別コードは P_HIZUKE＋11 と設定されます。



最後に、罫線を調整しページ番号などを配置して帳票レイアウトを完成させます。

完成した帳票レイアウト


売上げ帳

PAGE ** / **

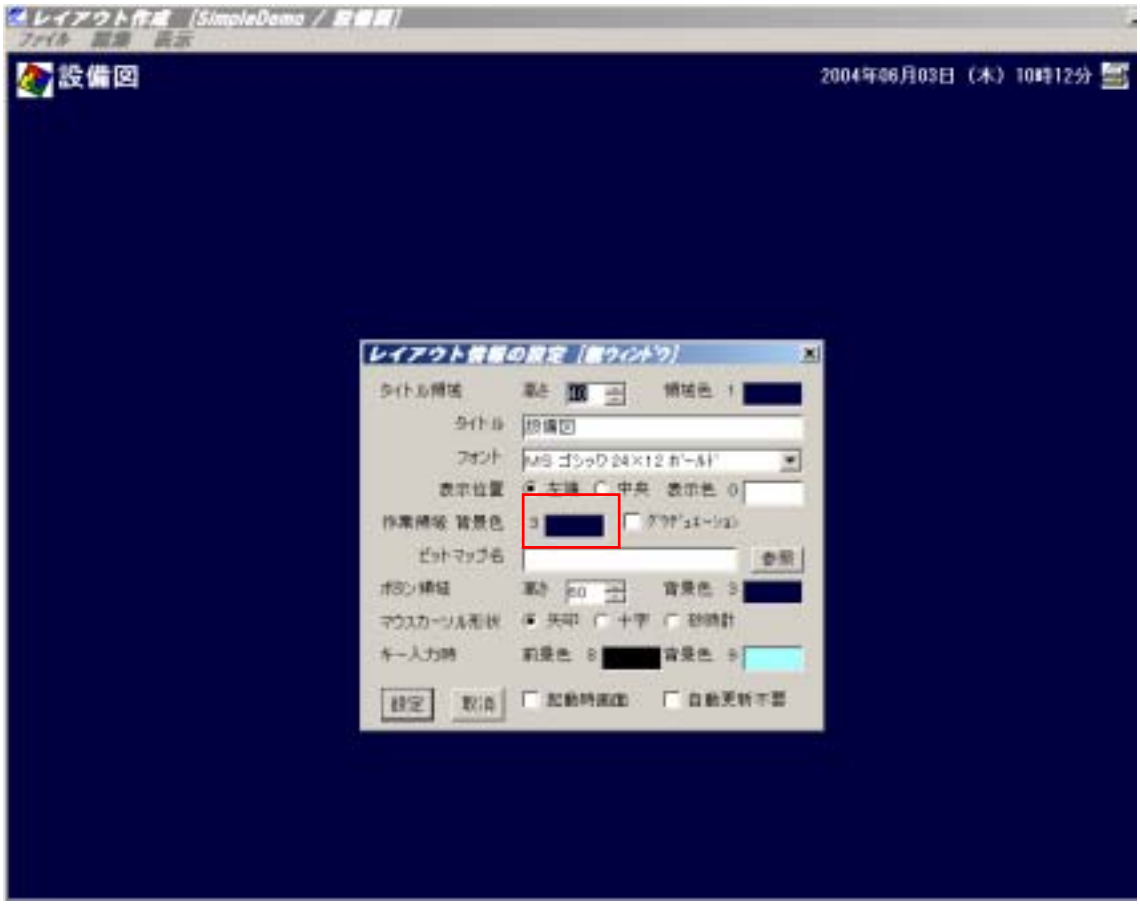
日付	品　　名	数　量	単　価	合　計
########	##########################	********	********	**********
########	##########################	********	********	**********
########	##########################	********	********	**********
########	##########################	********	********	**********
########	##########################	********	********	**********
########	##########################	********	********	**********
########	##########################	********	********	**********
########	##########################	********	********	**********
########	##########################	********	********	**********
########	##########################	********	********	**********
########	##########################	********	********	**********
########	##########################	********	********	**********

2－6　制御用画面の作成

制御システム用のレイアウト例として、簡易水道システムの監視画面を作成します。
（ただし、実際の画面からはかなり簡略化しています。）

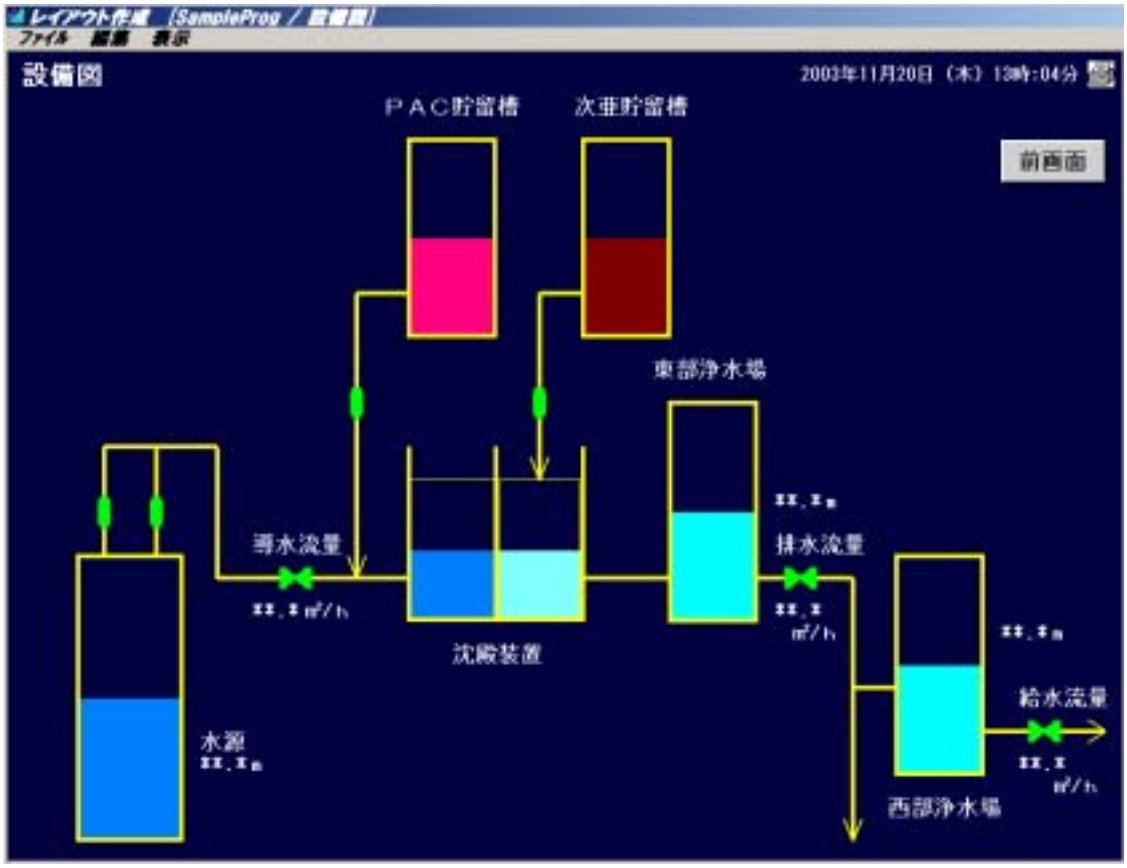
2－6－1　設備図画面の新規作成

メニュー画面の作成時と同様に［ファイル］メニュー → ［新規作成］ → ［親ウィンドウ］ を選択します。



設備図画面は他の画面とは異なり、背景色全体を暗くします。
また、この時方眼が見えなくなりますので方眼の表示色も白に変更します。
（表示メニュー → 方眼設定）

2－6－2　設備図画面の完成図



設備図には、直線、 マーク、 棒グラフ、 固定文字列、 数値データ を使用しています。

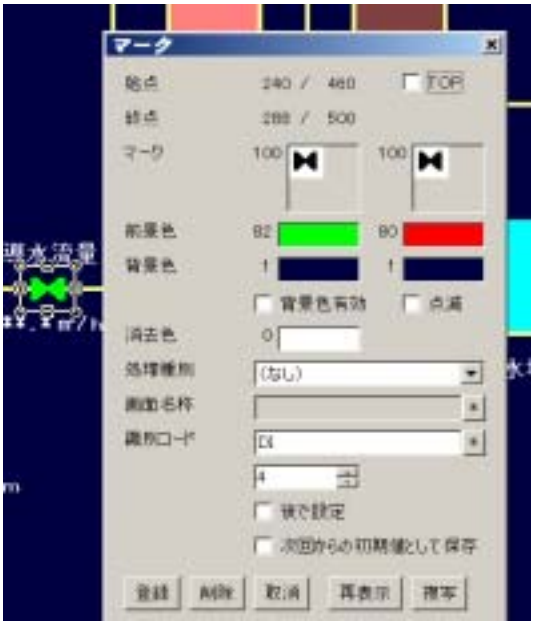
2－6－3　マーク

マーク は 2値のビットマップ で ON／OFF のピクセルに表示色を設定することができます、

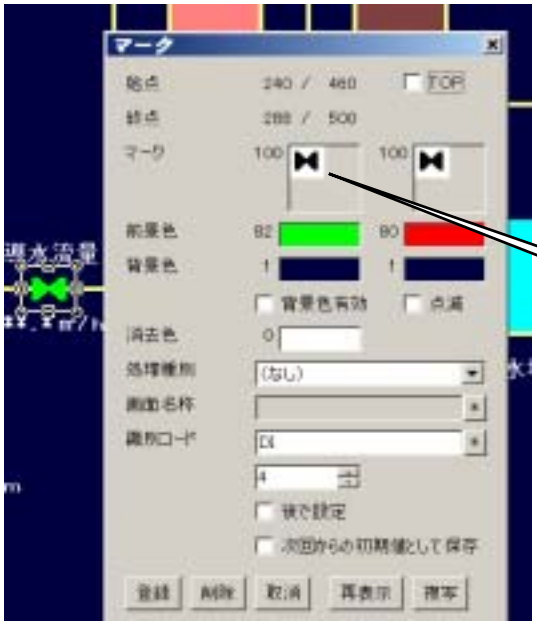
ON のビットに 前景色、 OFF のビットに 背景色 が割り付けられます。
設備図では［背景色有効］ が設定されていないので、OFF のビットは作業領域の色となります。

SimpleMaster で CRT 画面上に配置することができる図形や文字列は2系統の色を設定することができます。（プロジェクトの設定により4系統の設定も可能です。）

設備図では ポンプやバルブの運転状態（入/切）に合わせて表示色を切り替えます。



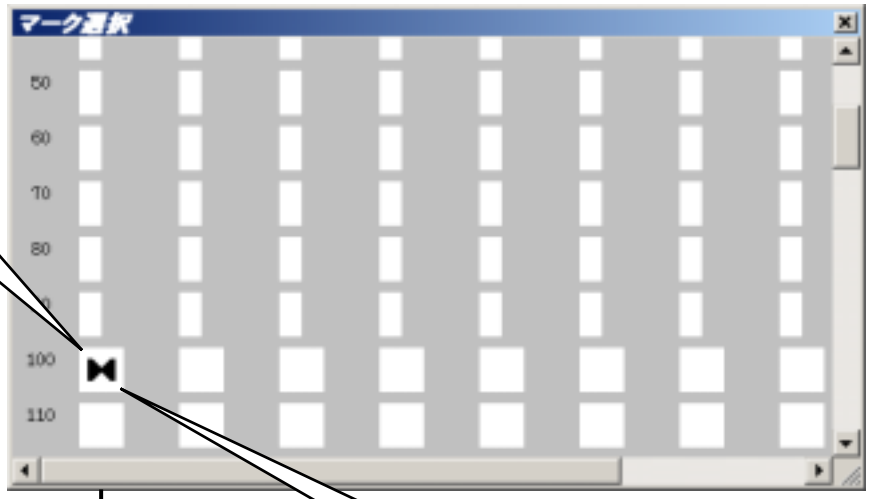
マークの編集



マークをマウス左ボタンで
クリックすると

マーク選択画面が
開きます。

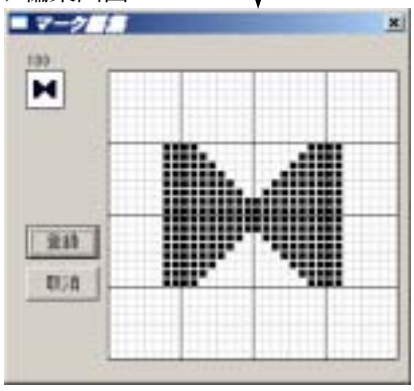
マーク選択画面



表示したいマークを
マウス左ボタンでクリック
して選択します。

マウス右ボタンでクリック
するとマーク編集画面が
開きます。

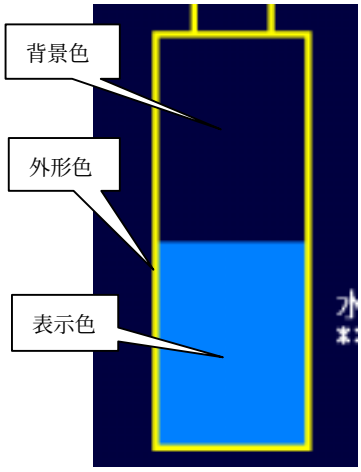
マーク編集画面



マウス左ボタンで ON のビット
マウス右ボタンで OFF のビットを設定します。
左右いずれかのボタンを押したままの状態で
編集画面上を移動することで連続した設定が
できます。
［登録］ボタンで選択画面に登録されます。

マークは ［編集］メニュー → ［マークの設定］ でも編集することができます。

2－6－4　棒グラフ

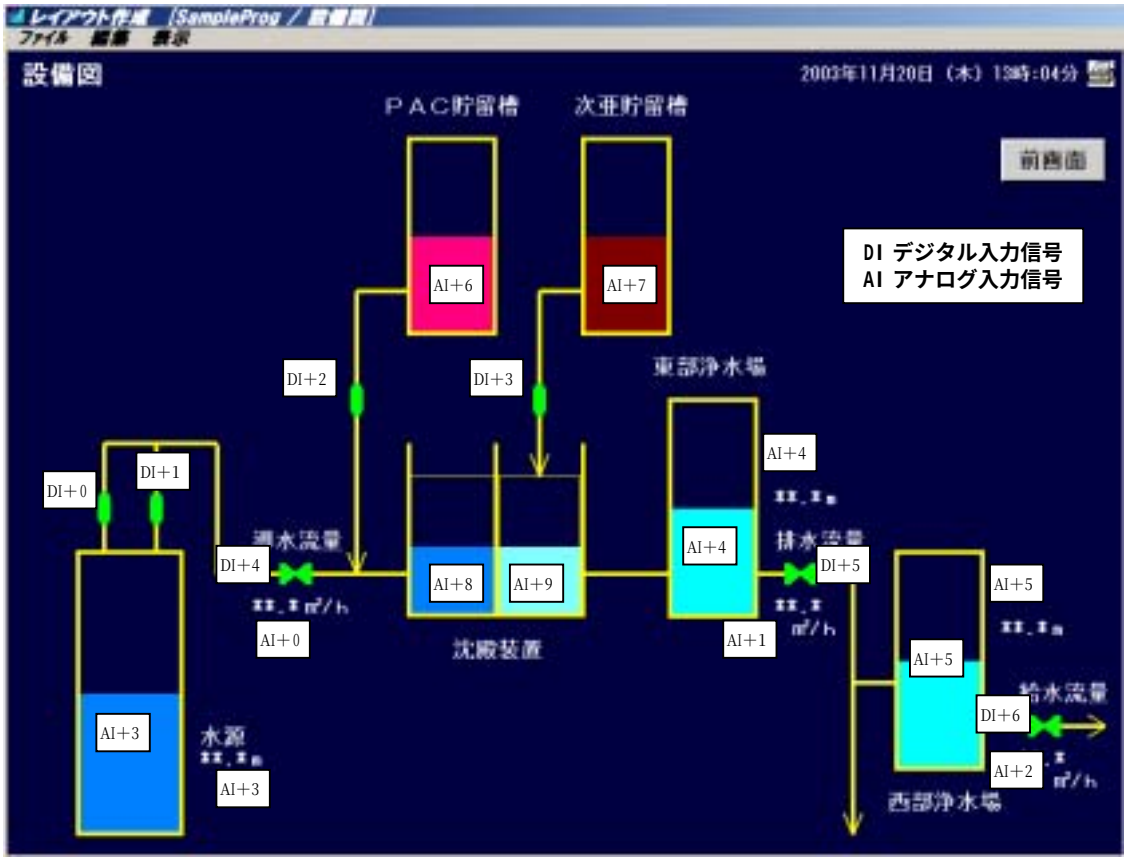


数値データを上下限値でスケーリングし
棒グラフとして CRT 画面上に表示します。

設備図では数値データが異常の時
棒グラフの表示色を切り替えるため
2系統の表示色を設定します。

2－6－5　識別コード設定

下図のように識別コードを設定します。



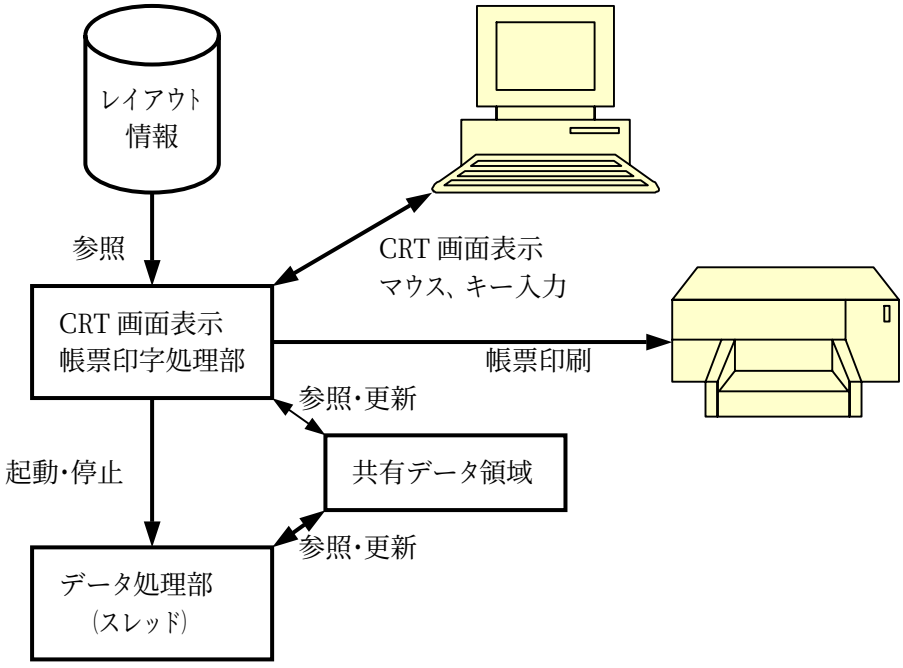
3　ロジック作成

SimpleMaster で作成したレイアウト情報に対してロジック（プログラム）を作成します。
ここで作成するプログラムは、シングルプロセスで構成される小規模なアプリケーションです。

ロジックを作成するためには次のコンパイラが必要です。
　Microsoft Visual C++ V6.0　または　 Microsoft Visual C++ .Net

〔注意〕 SimpleMaster にはコンパイラは含まれていません。 別途準備してください。
本書ではコンパイラの使用方法などの説明は含まれていません。 コンパイラに付属するマニュアルを参照してください。

3－1　プログラムの構造

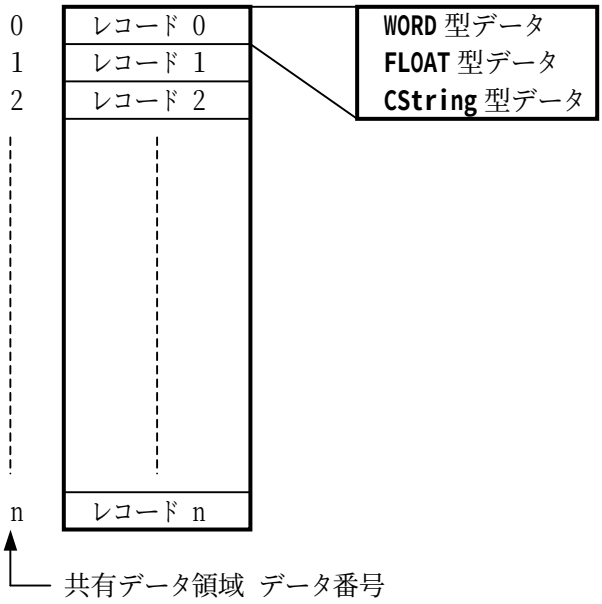


小規模なアプリケーションでは、シングルプロセスで構成し、共有データ領域はローカルメモリ上に確保されます。 データ処理部はスレッドで作成し、共有データ領域を介して CRT 画面処理、帳票印字処理部と情報の受渡しを行います。

中・大規模アプリケーションでは共有データ領域を 共有メモリ 上に確保し、複数のプロセスから参照・更新されます。 また、名前付きパイプを使用して複数のパソコンで共有メモリ上に確保されたデータ領域をアクセスする手法も準備しています。

3－2　共有データ領域

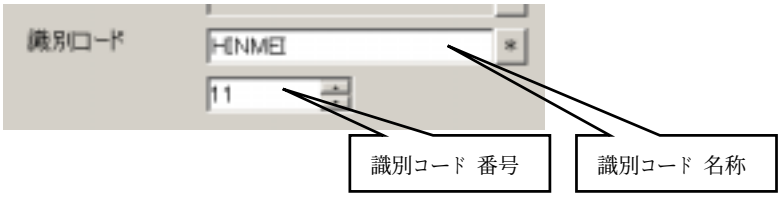
共有データ領域は WORD 型データ、数値データ、文字列データ が格納される配列で構成されます。



共有データ領域には一連の番号が割当てられ、ロジック部はこの番号で参照･更新を行います。

共有データ領域の確保

共有データ領域はレイアウト作成時にデータ数などが決まります。



レイアウト作成時、共有データ領域は 識別コード名称と番号で管理されます。
レイアウト作成プログラム SimpleMaster は ［ファイル］メニュー → ［ロジック作成］ を選択したとき、次の手順で共有メモリの一覧表を作成します。

プロジェクトに属するすべてのレイアウト情報を読込み、識別コード名称と同一名称で参照する番号の最大値の一覧表を作成します。　（作成例）

識別コード 名称	データ番号 最大値	データ番号 先頭値
SYSTEM	１００	０
AAAA	１０	１００
BBBB	２０	１１０
CCCC	５	１３０
DDDD	１０	１３５
	（合計） １４５	

このような表を作成し、共有データ領域のデータ量、各識別名称の先頭のデータ番号を決めます。
識別コード名称 SYSTEM（データ量 １００） は SimplePack で管理する領域で、レイアウト作成側で確保する必要はありません。

ヘッダーファイル　SimpleCode.h

レイアウト作成プログラム SimpleMaster で［ファイル］メニュー → ［ロジック作成］ を選択すると識別コードの一覧表を作成し、ヘッダーファイル SimpleCode.h をロジック領域に作成します。

```
#define     SYSTEM          0
#define     AAAA            100
#define     BBBB            110
#define     CCCC            130
#define     DDDD            135
```

ロジック部ではこのヘッダーファイルを参照してレイアウト部で定義された共有データ領域の識別コードに該当するデータへアクセスします。

ロジック部で右図のデータ領域をアクセスするためには次のように記述します。

```
#include “SimpleCode.h”

float  data = Get_fData (SUURYOU+10);
Set_fData (SUURYOU+10, data);
```

識別コードはこのようにコンパイラに渡される情報となりますのでコンパイラの「名前付け規則」に従わなければなりません。
レイアウト作成プログラムでは識別コードに次の制限を加えています。

使用できる文字	数字 (0～9)、英字 (大文字、小文字 共に可)、特殊記号 ("_" アンダースコアのみ)
先頭文字	英字のみ
文字数	最大 20文字

レイアウト作成プログラム SimpleMaster で識別コードの追加、データ量の変更を行った場合、［ロジック作成］メニューを使用してヘッダーファイル SimpleCode.h を作成しなおし、再コンパイルが必要となります。

3－3　最小限のアプリケーション

レイアウト作成プログラム SimpleMaster の［ファイル］メニュー →［ロジック作成］を選択すると次の4個のファイルがプロジェクトで設定されたロジック格納先フォルダーに作成されます。

SimpleCode.h	**識別コード一覧表**	**（詳細は前ページ参照）**
SampleDemo.h	**ヘッダーファイル**	**（プロジェクト名.h ）**
SampleDemo.cpp	**ソースプログラムファイル**	**（プロジェクト名.cpp）**
SampleDemo.ini	**環境設定ファイル**	**（プロジェクト名.ini）**

SimpleDemo.h

```
#include    <SimplePack.h>
#include    "SimpleCode.h"
#include    "Uriagecyo.h"
```

SimpleDemo.cpp

```
#include    "SampleDemo.h"

BOOL  UserInit()                                        // 起動時処理
{
     return TRUE;
}

void  UserExit ()                                       // 終了時処理
{
}

void  UserTimer ()                                      // 100msec タイマー処理
{
}

CPtrnDisp*  GetPtrnWnd (LPCTSTR name)                   // ウィンドウの生成
{
     if (lstrcmp(name, "メニュー画面")       == 0) return (new CPtrnDisp());
     if (lstrcmp(name, "売上げ帳")           == 0) return (new CPtrnDisp());
     if (lstrcmp(name, "伝票入力（新規）")   == 0) return (new CPtrnDisp());
     if (lstrcmp(name, "伝票入力（修正）")   == 0) return (new CPtrnDisp());
     if (lstrcmp(name, "設備図")             == 0) return (new CPtrnDisp());
(注) if (lstrcmp(name, "帳票_売上げ帳")      == 0) return (new CPtrnDisp());
     return NULL;
}
```

関数 UserInit　　アプリケーション固有の初期化処理を組込むために使用します。
自動生成された時点では **return TRUE;** が記述されます。
戻り値を **FALSE** とするとアプリケーションプログラムの起動は中止されます。

関数 UserExit　　アプリケーション固有の終了時処理を組込むために使用します。

関数 UserTimer　　100msec 周期で呼び出されます。
アプリケーションプログラム側で任意の機能を組込むことができます。

関数 GetPtrnWnd　　ＣＲＴ画面とその画面を処理するクラスを関連付けし、クラスを生成します。

```
if (lstrcmp(name, "伝票入力（修正）") == 0) return (new CPtrnDisp());
```

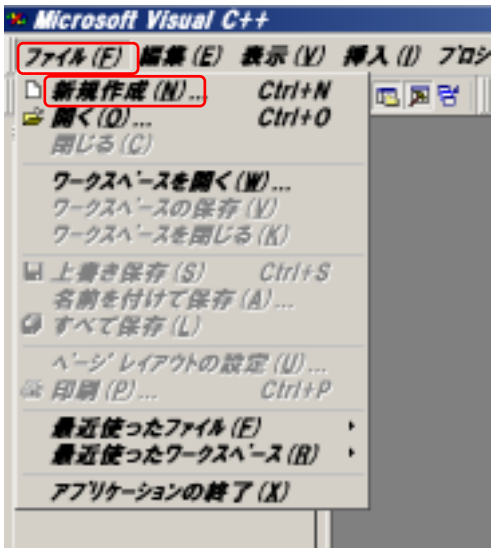
（注）帳票用レイアウトは関数 GetPtrnWnd で記述する必要はありませんが、自動生成では区別できないため作成されます。（削除可、 残しておいても影響はありません。）

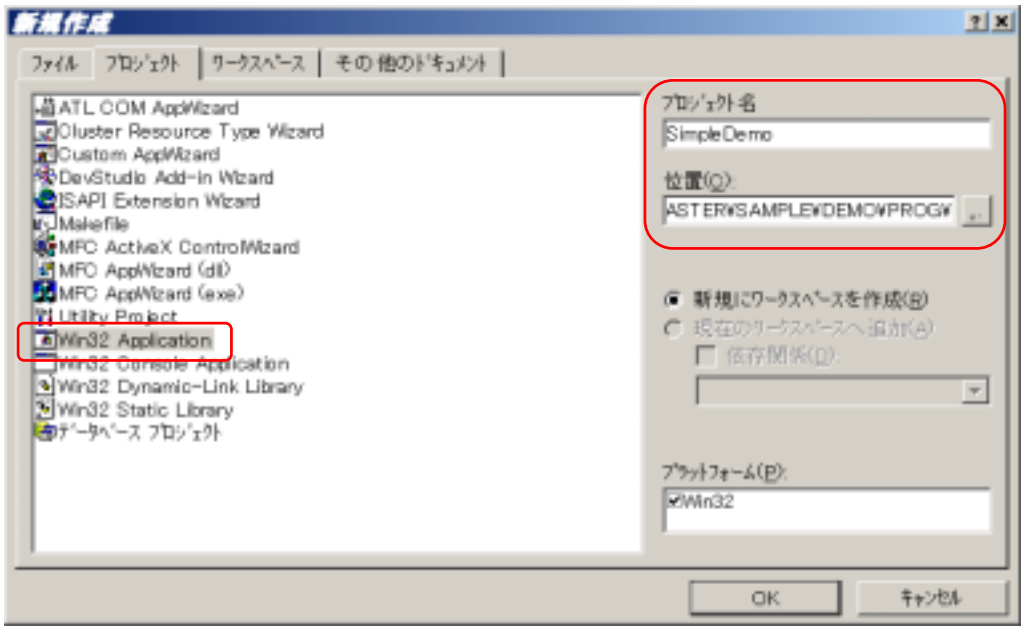
3－4　コンパイラの設定

最小限のアプリケーションプログラムを実行できるようにするため、コンパイル／リンクを行います。
こではコンパイラに Microsoft Visual C++ V6.0 を使用します。
SimpleMaster はデフォルトの設定でインストールされたものとして説明しています。
（インストール先フォルダー名　　**C:\Program Files\SimpleMaster)**

3－4－1　プロジェクトの新規作成

Microsoft Visual C++ を起動し、［ファイル］ → ［新規作成］ を選択します。





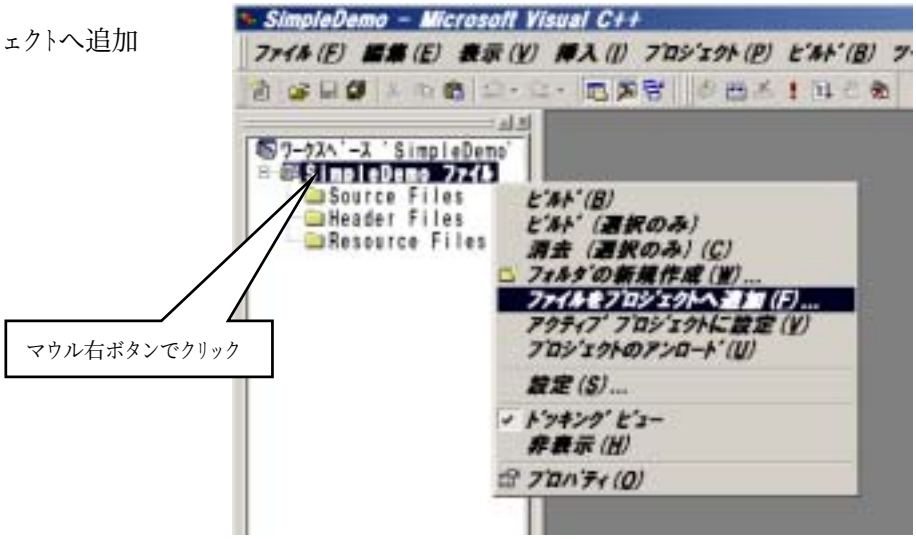
プロジェクトの種別　　Win32 Applocation
プロジェクト名　　SimpleDemo　　**SimplrMaster** でレイアウト作成時に設定したプロジェクト名
位置　　C:\Program Files\SimpleMaster\Sample\Demo\Prog　　**SimplrMaster** でレイアウト作成時に設定したロジック格納先フォルダー名

新規作成のダイアログボックスで [OK] を押します。

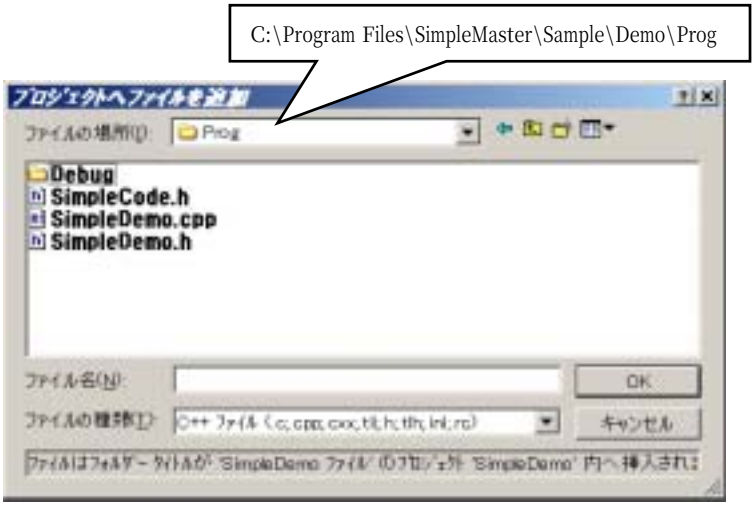
OK を押し空のプロジェクトを作成します。

3－4－2　プロジェクトを構成するファイルの設定

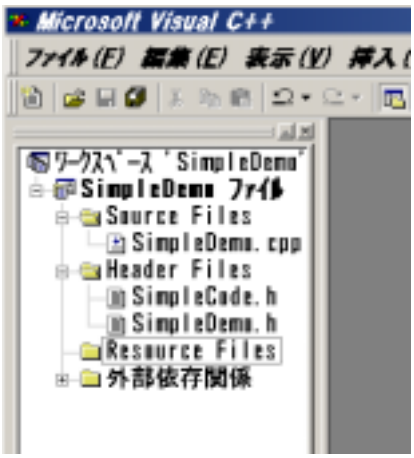
ファイルをプロジェクトへ追加します。



アプリケーションプログラム側のソースファイルを設定します。

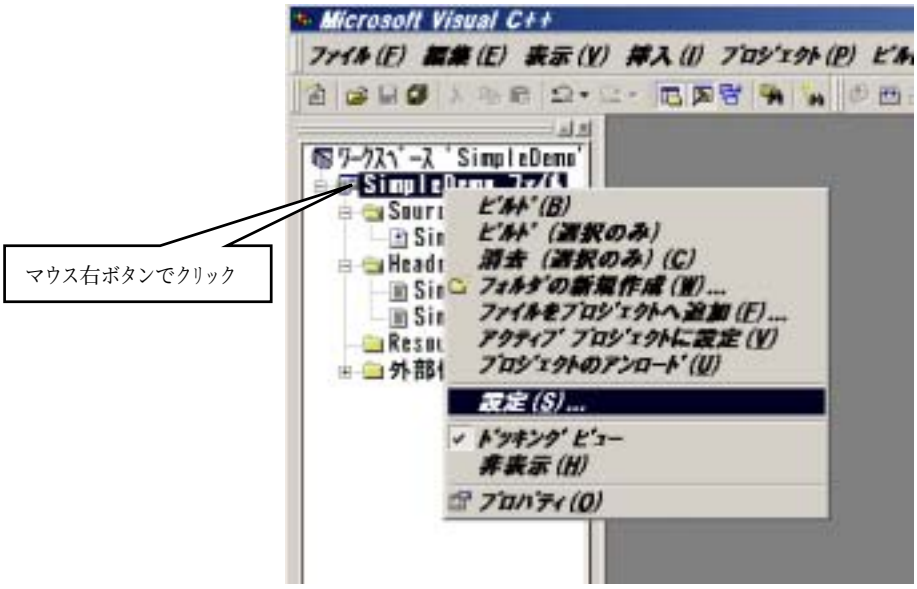


ソースファイルの設定完了時

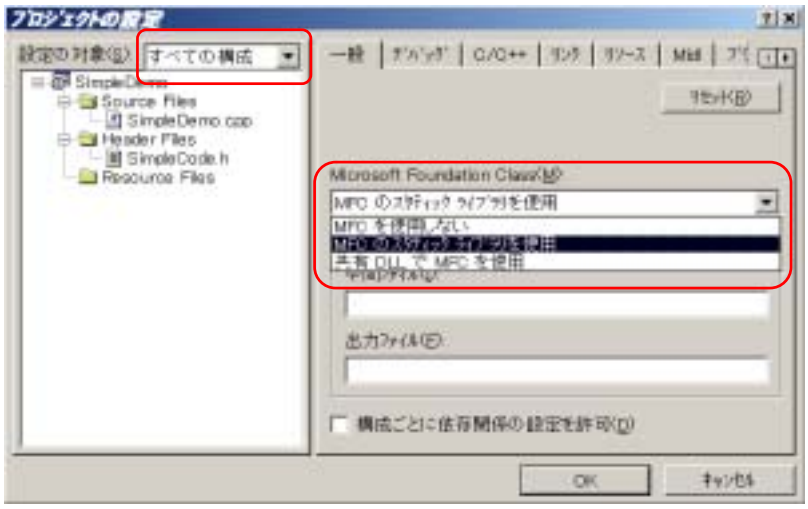


SimpleCode.h	レイアウト作成プログラム **SimpleMaster** で作成した識別コード定義表
SimpleDemo.h	最小限のアプリケーションが記述されたヘッダーファイル
SimpleDemo.cpp	最小限のアプリケーションが記述されたソースファイル

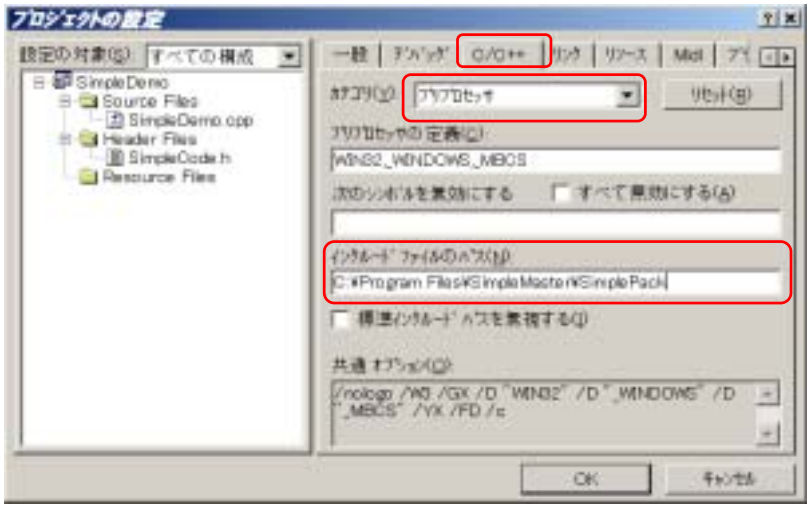
3－4－3　プロジェクトの設定



［すべての構成］ を選択し
ＭＦＣのスタティクライブラリ
を使用 を選択します。

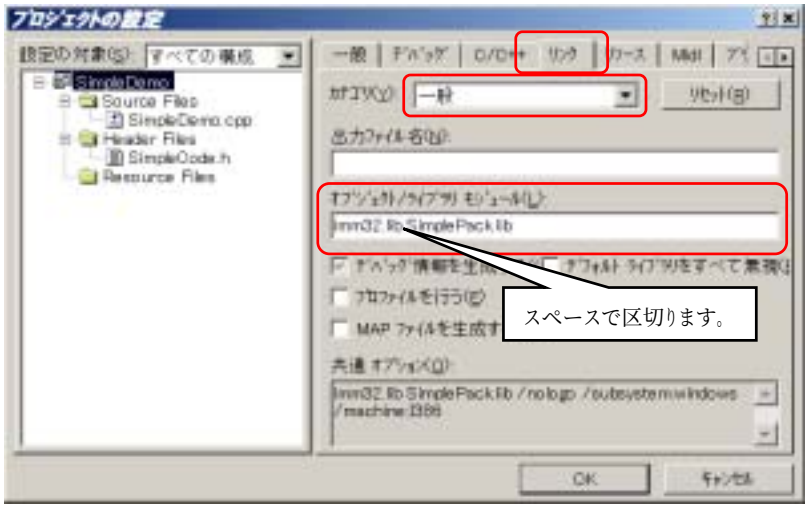


［C/C++］ を選択し
プリプロセッサ、
インクルードファイルの
パスを設定します。



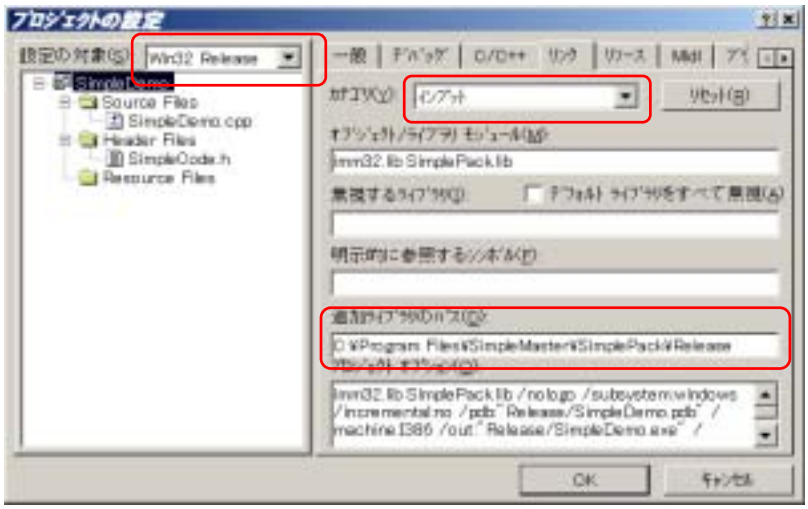
［リンク］を選択し
オブジェクト/ライブラリ
モジュールに
　　imm32.lib
　　SimplePack.lib
を設定します。

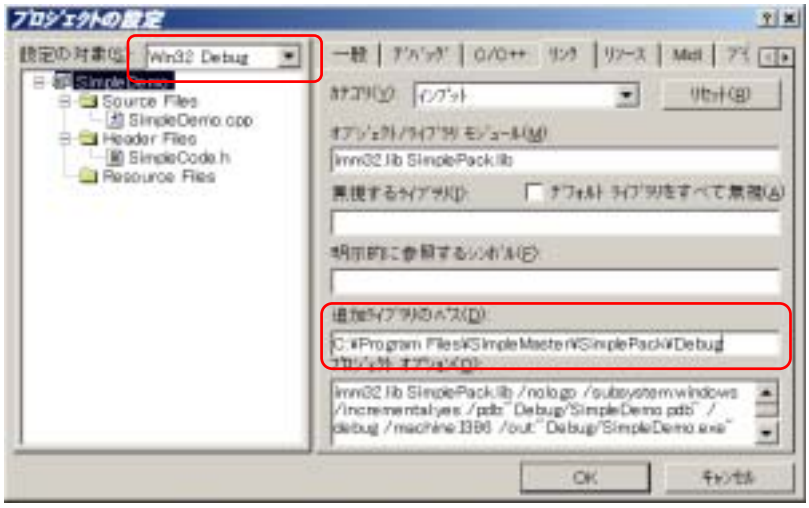
imm32.lib は日本語IMEを直接制御するために必要です。



SimplePack.lib へのパスを設定します。

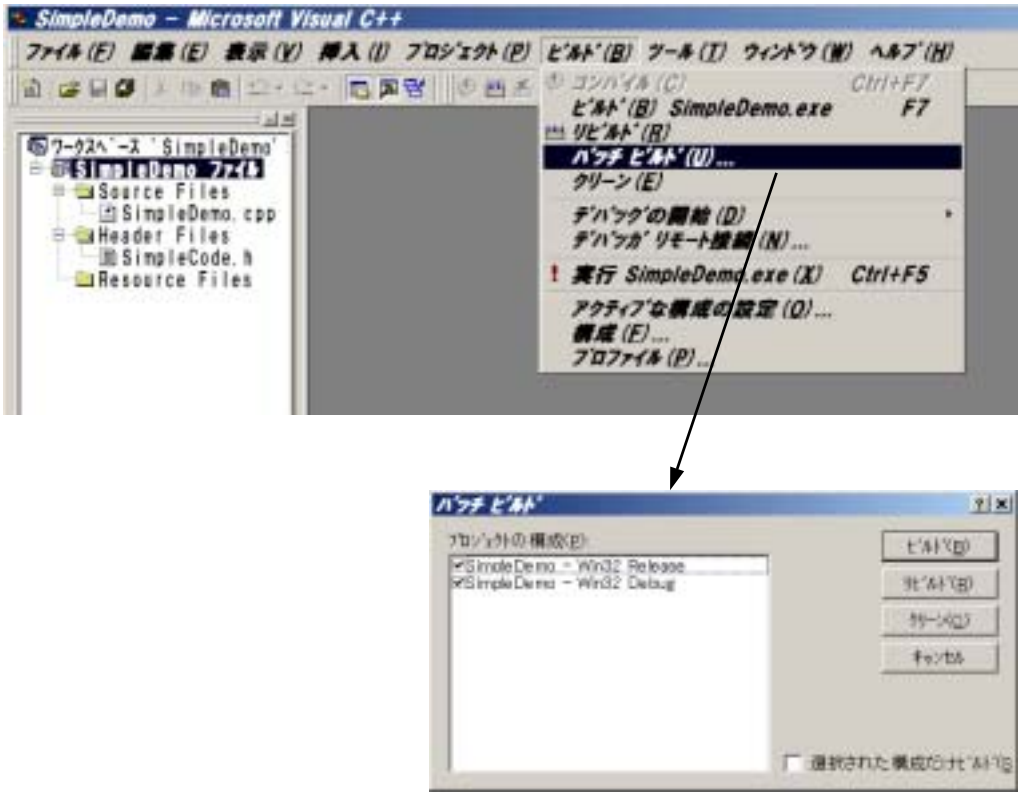
Release と Debug モードでパスの最後が異なります。





3－4－4　プロジェクトのビルド

［ビルドメニュー］、［リビルド］または［バッチビルド］でコンパイル、リンクを行います。



3－5　アプリケーションの実行

アプリケーションの実行には環境設定ファイル SimpleDemo.ini が必要となります。
このファイルは、レイアウト作成プログラム SimpleMaster で ［ファイル］メニュー → ［ロジック作成］を選択すると 自動生成されます。

コンパイラで実行プログラムは Debug および Release サブフォルダーに作成されます。
実行プログラム SimpleDemo.exe と環境設定ファイル SimpleDemo.ini は同一のフォルダーに格納する必要があるので、ここでは手作業で環境設定ファイルをそれぞれのサブフォルダーに複写してください。

以上で準備は終了し、プログラムを実行することができます。

3－5－1　メンテナンス画面の追加

最小限のアプリケーション では、メニュー選択、前画面、終了ボタンの操作を処理することができます、メンテナンス画面を追加すると、データを注入し画面に表示することができるようになります。
メンテナンス画面はプロジェクトを新規作成するとバージョン情報画面と共に自動的に複写されています。
メンテナンス画面を使用できるようにするためにプログラムの追加/修正は必要ありませんが、メンテナンス画面を呼び出すためにレイアウトの変更が必要です。

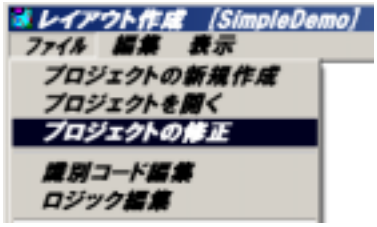
レイアウトの修正

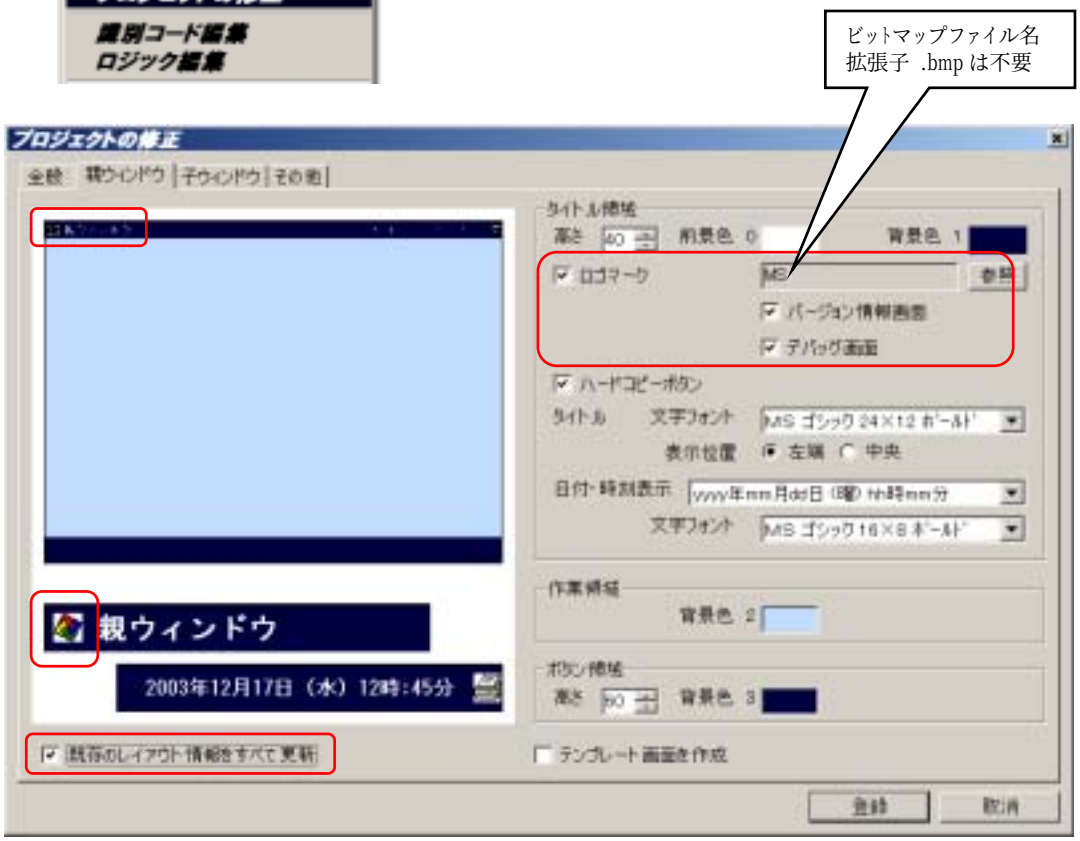
CRT 画面上の操作でメンテナンス画面、およびバージョン情報画面を使用できるようにするためレイアウトの修正を行います。

バージョン情報画面、メンテナンス画面は親ウィンドウ／タイトル領域 の左端に配置されたロゴマーク（ビットマップ）を

マウス左ボタンでクリック　　バージョン情報画面
マウス右ボタンでクリック　　メンテナンス画面　　を開くように設定することができます。

プロジェクトの修正





ロゴマークは 参照 ボタンを押し、ハードディスク上で探してください。 ここで選択されたビットマップファイルは、レイアウト情報が格納されたフォルダーへ複写されます。
［既存のレイアウト情報をすべて変更］ をセットして登録してください。

3－5－2　メンテナンス画面からのデータ注入

アプリケーションプログラム実行中にロゴマークをマウス右ボタンでクリックするとメンテナンス画面が開きます。

① 売上げ帳 日付の欄は 識別コード **HIZUKE+0,1,2 ～ 11** の文字列情報を表示するようにレイアウト情報で設定しています。

② メンテナンス画面で識別コード HIZUKE をマウス左ボタンで選択します。

③ STRING 欄でデータを注入したい行をマウス左ボタンでクリックします。
キーボードより注入したいデータを入力します。（Enter キーで入力終了）

① メンテナンス画面で設定されたデータは直ちに画面に表示されます。

3－6　売上げ帳画面処理プログラム

売上げ帳画面を処理するためのクラス CPtrnUriagecyo を作成します。
一般のクラスと同様に 定義部 と 実装部 に分けて作成します。

3－6－1　CPtrnUriagecyo クラスの定義部　（ヘッダーファイル：Uriagecyo.h）

```
class CPtrnUriagecyo : public CPtrnDisp    ①                    // 派生クラスの定義
{
     protected:    ②
          virtual void    PtrnInitFunc    ();                   // 表示開始処理
          virtual void    PtrnMesgFunc    (int);                // マウス入力時処理

     private:  int     DispLine;                                 // 表示開始行番号
               int     TotalLine;                                // 伝票設定済み行数
       ③      int     SelctLine;                                // 選択された行番号
               DENPYO  Denpyo[RECORD_DENPYO];                    // 伝票データ領域

               void    SetDispData     (int);                    // 伝票データ表示
               void    SetDenpyo       (int);                    // 入力データ設定
               void    GetDenpyo       (int);                    // 入力データ取得

               void    PrintUriagecyo  ();                       // 帳票印刷
};
```

① クラス CPtrnUriagecyo を宣言します。
SimpleMaster / SimplePack を使用するアプリケーションプログラムでは、CRT 画面ごとに基本クラス CPtrnDisp より派生したクラスを定義します。
メニュー画面のように基本クラス CPtrnDisp の機能で画面を処理できる場合は不要です。

② 売上げ帳画面 独自の処理を記述するために仮想関数をオーバーライドします。
PtrnInitFunc　ＣＲＴ画面が開かれる前に実行されます。 表示データの初期化、スクロールバーの初期化などの処理を行います。
PtrnMesgFunc　マウス左ボタン操作などで送られるメッセージに対応した処理を記述します。

③ 売上げ帳画面 のみで使用する作業領域、関数を定義します。

画面に表示する 伝票データ はディスクファイルに保存されており、売上げ帳画面を表示する時にのみ CPtrnUriagrcyo のデータメンバ領域 Denpyo に読込まれます。
伝票データの構造はこのヘッダーファイルで次のように定義します。

```
const   int     RECORD_DENPYO = 500;                              // データ容量

struct  DENPYO  {                                                 // 伝票フォーマット
     int     date1, date2;                                        // 月・日
     int     tanka;                                               // 単価
     int     suuryou;                                             // 数量
     int     goukei;                                              // 合計
     char    hinmei[22];                                          // 品名
}
```

3－6－2　CPtrnUriagecyo クラスの実装部　（ソースファイル ：Uriagecyo.cpp）

仮想関数　PtrnInitFunc

```
void  CPtrnUriagecyo :: PtrnInitFunc ()                          // 表示開始時処理
  {
   FileRead(ProjPath, "URIAGECYO", Denpyo, sizeof(Denpyo)); ①    // ファイル読込み

   TotalLine = SelctLine = 0;    ②                               // 行番号初期化
   for (int i=0; i<RECORD_DENPYO; ++i) {                         // 登録済みレコード数
        if (Denpyo[i].date1 != 0) ++TotalLine;
        else                      break;
   }
   PtrnSetScroll(SCROLL, TotalLine, 0, S_FILE, S_LINE); ③        // スクロールバー初期化

  SetDispData(0);   ④                                            // 初期画面表示
   ShowWindow(SW_SHOW);   ⑤
}
```

① 売上げ帳データが格納されたファイルを読込みます。
　関数 FileRead は SimplePack で定義されています。
② 作業領域の初期化と有効なデータ数を取得します。
　日付情報が設定されていれば有効なデータとみなします。
③ スクロールバーの初期設定を行います。
④ CRT 画面の表示情報を共有データ領域へ設定します。
⑤ CRT 画面を表示します。　SimplePack では CRT 画面を非表示の状態で作成します。
　仮想関数 PtrnInitFunc を実装した場合には必ずこの行の記述が必要です。

仮想関数　PtrnMesgFunc

```
void  CPtrnUriagecyo :: PtrnMesgFunc (int code)                  // マウス入力時処理
{
    if (code == NYUURYOKU) {        ①                            // 伝票入力（新規）
        SetDenpyo(TotalLine);
        PtrnDispChild(1, "伝票入力（新規）", S_FILE, S_LINE);
    }

    else if (code == HOZON ) FileWrite(ProjPath, "URIAGECYO", Denpyo, ② // ファイル書込み
                                                  sizeof(Denpyo));
    else if (code == PRINT ) PrintUriagecyo(); ③                 // 帳票印刷
    else if (code == SCROLL) SetDispData(Get_iData(SCROLL));  ④   // スクロールバー

    else if (code == TOUROKU) {   ⑤                              // 登録（新規）
        PtrnCloseWnd(1);                                         // 入力画面を閉じる
        GetDenpyo(TotalLine);                                    // 入力結果取得
        ++TotalLine;
        PtrnSetScroll(SCROLL, TotalLine, ++DispLine, S_FILE, S_LINE); // スクロールバー再設定
        SetDispData(Get_iData(SCROLL));                          // 再表示
    }

    else if (code == TOUROKU+1) {   ⑥                            // 登録（修正）
        PtrnCloseWnd(1);                                         // 入力画面を閉じる
        GetDenpyo(SelctLine);                                    // 入力結果取得
        SetDispData(DispLine);                                   // 再表示
    }

    else if (code == SAKUJO) {    ⑦                              // 削除
        PtrnCloseWnd(1);                                         // 入力画面を閉じる
        for (int i=SelctLine; i<=TotalLine; ++i)
             Denpyo[i] = Denpyo[i+1];
        --TotalLine;
        PtrnSetScroll(SCROLL, TotalLine, DispLine, S_FILE, S_LINE); // スクロールバー再設定
        SetDispData(Get_iData(SCROLL));                          // 再表示
    }
```

```
    else {                                                    // 行選択
        int  line = -1;
⑧       if      ((code >= HIZUKE ) && (code < (HIZUKE +12))) line = code - HIZUKE;
        else if ((code >= HINMEI ) && (code < (HINMEI +12))) line = code - HINMEI;
        else if ((code >= TANKA  ) && (code < (TANKA  +12))) line = code - TANKA;
        else if ((code >= SUURYOU) && (code < (SUURYOU+12))) line = code - SUURYOU;
        else if ((code >= GOUKEI ) && (code < (GOUKEI +12))) line = code - GOUKEI;

        if (line != -1) {
            if ((DispLine + line) < TotalLine) {              // 修正
                SetDenpyo(SelctLine = DispLine + line);
                PtrnDispChild(1, "伝票入力（修正）", S_FILE, S_LINE);
            }
        }
    }
}
```

仮想関数 PtrnMesgFunc は画面上に配置された表示要素をマウスでクリックした時、処理種別コードに［メッセージ送信］ が設定されている場合に呼び出されます。
引数には 識別コード が設定されます。

① 売上げ帳画面で ［伝票入力］ ボタンをマウス左ボタンでクリックされた時の処理を記述します。 入力用データ領域を初期化し、伝票入力画面［新規］ を子ウィンドウ1に表示します。

② 売上げ帳画面で ［登録］ ボタンをクリックされた時の処理で、売上げ帳データをディスクに保存します。

③ 帳票印刷を行います。 （3－7参照）

④ スクロールバーを操作された時の処理を記述します。
スクロールバーは SimplePack 側で処理されています。（マウスホイール、キーボードにも対応しています。）
スクロールバーが操作されると、操作された内容や設定値（表示量、データ量、移動量）に従って表示開始位置を計算し、識別コードで設定された共有データ領域に設定し仮想関数を呼び出します。
アプリケーションプログラム側では共有データ領域より表示開始位置を取得し、1ページ分の再表示を行います。

⑤、⑥、⑦ 伝票入力画面で 登録、削除 ボタンを操作された時、子ウィンドウからメッセージが送られます。 親ウィンドウ（売上げ帳画面）では、子ウィンドウの消去 （PtrnCloseWnd）、入力結果の取り込みやデータの削除を行い画面を再表示します。
子ウィンドウの 取消 ボタンは処理種別コードで ［自ウィンドウを閉じる］ と設定されているのでアプリケーションプログラムで処理する必要はありません。

⑧ 売上げ帳の表内部をマウス左ボタンでクリックされた時の処理を記述します。
引数で渡された識別コードより行番号を計算し、その行に有効なデータが表示されていれば修正画面を開きます。

ローカル関数　SetDispData

```
void  CPtrnUriagecyo :: SetDispData (int line)                    // 表示データ設定
{
   DispLine = line;

   Clear_AllData(HIZUKE );
   Clear_AllData(HINMEI );   ①
   Clear_AllData(TANKA  );
   Clear_AllData(SUURYOU);
   Clear_AllData(GOUKEI );
   for (int i=0; i<12; ++i) {          ②
        int  datano = DispLine + i;
        if (datano < TotalLine) {
             CString  date;
             date.Format("%02d / %02d", Denpyo[datano].date1,      // 日付変換
                                        Denpyo[datano].date2);

        ③   Set_sData(HIZUKE +i, date, S_FILE, S_LINE);           // 表示データ設定
             Set_sData(HINMEI +i, Denpyo[datano].hinmei,  S_FILE, S_LINE);
             Set_iData(TANKA  +i, Denpyo[datano].tanka,   S_FILE, S_LINE);
             Set_iData(SUURYOU+i, Denpyo[datano].suuryou, S_FILE, S_LINE);
             Set_iData(GOUKEI +i, Denpyo[datano].goukei,  S_FILE, S_LINE);
        }
        else {
        ④   PtrnSetDisable(TANKA  +i, TRUE, FALSE);               // 非表示設定
             PtrnSetDisable(SUURYOU+i, TRUE, FALSE);
             PtrnSetDisable(GOUKEI +i, TRUE, FALSE);
        }
   }

   int  goukei = 0;          ⑤                                    // 総合計
   for (i=0; i<TotalLine; ++i)
        goukei += Denpyo[i].goukei;
   Set_iData(SOUGOUKEI, goukei, S_FILE, S_LINE);
}
```

1ページ分のＣＲＴ画面表示情報を共有データ領域へ設定します。
引数には表示開始行番号が渡されます。
SimpleMaster を使用したアプリケーションでは CRT 画面への表示は専用の関数を使用した共有データ領域へのデータ設定で行われます。

① CRT 画面表示に使用する共有メモリ領域を初期化します。
WORD 型データ領域には 0x0000, 数値データ領域には 0.0、文字列データ領域には NULL 文字が設定されます。

② 1ページの行数分繰り返します。

③ 関数 **Set_xData** を使用してクラス内の作業領域より CRT 画面表示用の共有データ領域へ表示データを設定します。
関数 **Set_xData** の引数 S_FILE, S_LINE はコンパイル時にソースファイル名と行番号が展開されます。共有データ領域への設定エラーが発生した場合やメンテンンス画面でモニタが指定されているとログファイルに設定値とともに設定元のソースファイル名／行番号が出力されます。

④ CRT 画面の途中で有効なデータの表示が終了した場合の処理です。
文字列データの場合は NULL 文字で初期化しますので (①) 画面上には何も表示されませんが、数値データは 0 が表示されます。 関数 Set_Disable を使用して表示禁止に設定します。

⑤ 総合計値を計算し共有データ領域へ設定します。

ローカル関数　SetDenpyo, GetDenpyo

```
void  CPtrnUriagecyo :: SetDenpyo (int line)                          // 伝票入力データ設定
{
     Set_iData(IN_HIZUKE,    Denpyo[line].date1,   S_FILE, S_LINE);       // 月
     Set_iData(IN_HIZUKE+1,  Denpyo[line].date2,   S_FILE, S_LINE);       // 日
     Set_sData(IN_HINMEI,    Denpyo[line].hinmei,  S_FILE, S_LINE);       // 品名
     Set_iData(IN_TANKA,     Denpyo[line].tanka,   S_FILE, S_LINE);       // 単価
     Set_iData(IN_SUURYOU,   Denpyo[line].suuryou, S_FILE, S_LINE);       // 数量
     Set_iData(IN_GOUKEI,    Denpyo[line].goukei,  S_FILE, S_LINE);       // 合計
}


void  CPtrnUriagecyo :: GetDenpyo (int line)                          // 伝票入力データ取得
{
     Denpyo[line].date1   = Get_iData(IN_HIZUKE,    S_FILE, S_LINE);      // 月
     Denpyo[line].date2   = Get_iData(IN_HIZUKE+1,  S_FILE, S_LINE);      // 日
     Denpyo[line].tanka   = Get_iData(IN_TANKA,     S_FILE, S_LINE);      // 単価
     Denpyo[line].suuryou = Get_iData(IN_SUURYOU,   S_FILE, S_LINE);      // 数量
     Denpyo[line].goukei  = Get_iData(IN_GOUKEI,    S_FILE, S_LINE);      // 合計
     lstrcpy(Denpyo[line].hinmei, Get_sData(IN_HINMEI, S_FILE, S_LINE));  // 品名
}
```

関数 SetDenpyo　伝票入力画面（新規／修正）を開く前に表示データを設定します。
関数 GetDenpyo　伝票入力画面（新規／修正）で 登録 ボタンが押され親ウィンドウ（売上げ帳画面）でメッセージを受け取った時、修正結果を取得します。

3－7　伝票入力画面処理プログラム

売上げ帳1行分のデータをキーボードより入力するための画面処理プログラムを作成します。
ここで作成するプログラムでは 新規入力画面 と 修正入力画面 の処理を行います。

3－7－1　派生クラスの定義部　（ヘッダーファイル ：Uriagecyo.h）

```
class CPtrnDenpyo : public CPtrnDisp                          // 伝票入力画面
{
     protected:
     virtual   void    PtrnInitFunc    ();                    // 表示開始処理
     virtual   void    PtrnKeyFunc     (int, int, PTRNDATA*); // キー入力処理

     private:
               void    SetButton       ();                    // ボタン操作可否設定
};
```

派生クラス CPtrnDenpyo を定義します。
仮想関数 PtrnKeyFunc はキーボード入力が終了した時に呼び出され、入力を継続するための処理と、入力完了項目を判定し、 登録 ボタンの操作の可否の設定、合計値の計算を行っています。

3－7－2　派生クラスの実装部　（ソースファイル ：Uriagecyo.cpp）

仮想関数 PtrnInitFunc は画面を表示する前に呼び出されます。

```
void  CPtrnDenpyo :: PtrnInitFunc ()                          // 表示開始時処理
{
     SetButton();
     ShowWindow(SW_SHOW);
}
```

ここに KeyInputStart(IN_HIZUKE); と記述すれば画面を開いた時点でキーボード入力を開始することができます。

この仮想関数の最後に CRT 画面を表示するために次の行が必要です。

ShowWindow(SW_SHOW);

仮想関数 PtrnKeyFunc はキーボード入力が終了した時に呼び出されます。
SimplePack ではキーボードの入力は次の手順で行います。

① 入力項目の選択
［入力可］が設定された表示項目をマウスでクリックするか、プログラム側より KeyInputStart 関数を実行することでキーボード入力処理を開始します。

② 入力開始の通知
仮想関数 PtrnKeyStart が呼び出されます。
ガイダンスメッセージの表示などを行う場合に実装します。

③ キーボード入力処理
入力種別（文字列または数値データ）に対応して入力処理を行います。
入力終了キーが入力されエラーチェックが終わるまでの処理は SimplePack 側で処理されます。

④ キーボード入力終了通知
仮想関数 PtrnKeyFunc を呼び出します。

```
void  CPtrnDenpyo :: PtrnKeyFunc (int Key, int Shift, PTRNDATA* Ptrn)   // キー入力終了時処理
{
     SetButton();
     int  code = Ptrn->DataNo;
     if ((Key == VK_RETURN) || (Key == VK_TAB)) {
          if      (code == IN_HIZUKE+0) code = IN_HIZUKE+1;
          else if (code == IN_HIZUKE+1) code = IN_HINMEI;
          else if (code == IN_HINMEI  ) code = IN_TANKA;
          else if (code == IN_TANKA   ) code = IN_SUURYOU;
          else if (code == IN_SUURYOU ) code = IN_HIZUKE+0;
          KeyInputStart(code);
     }

     else if (Key == VK_DOWN) {
          if      (code == IN_HIZUKE+0) code = IN_HINMEI;
          else if (code == IN_HIZUKE+1) code = IN_HINMEI;
          else if (code == IN_HINMEI  ) code = IN_TANKA;
          else if (code == IN_TANKA   ) code = IN_SUURYOU;
          KeyInputStart(code);
     }

     以下省略
```

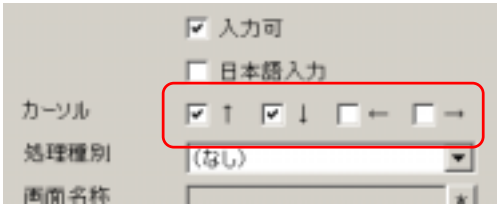
キーボード入力が行われた表示項目の識別コード

仮想関数 **PtrnKeyFunc** では入力終了時のキーコードと入力された表示項目より次に入力を継続すべき項目を選択します。

入力終了時のキーコードには次の7種類が渡されます。

No.	キーコード	ﾌﾟﾛｸﾞﾗﾑでのｺｰﾄﾞ	備考
1	Enter	VK_RETURN	
2	Tab	VK_TAB	
3	↑	VK_UP	有効なキーを個別に指定することができます。
4	↓	VK_DOWN	
5	←	VK_LEFT	
6	→	VK_RIGHT	
7	Esc	VK_ESCAPE	入力処理の終了とします。

カーソル移動キーはレイアウト作成時に描画要素の詳細設定画面で個別に指定することができます。



入力終了キーコードと入力処理が行われた表示項目の識別コードより次に入力する項目を選びその識別コードを引数として **KeyInputStart** を実行すればキーボード入力処理を再開することができます。

TAB キーが入力された場合、引数で渡される シフトキーの状態を判定して次のキー入力を行う識別コードを選択することができます。 (Shift = 1 : Shift キー押下中)

登録 ボタンの操作の可否と合計値の計算

```
void  CPtrnDenpyo :: SetButton ()                          // 登録ボタン
{
     BOOL  flag = TRUE;
     if (Get_iData(IN_HIZUKE+0) == 0) flag = FALSE;        // 月
     if (Get_iData(IN_HIZUKE+1) == 0) flag = FALSE;        // 日
     if (Get_iData(IN_TANKA   ) == 0) flag = FALSE;        // 単価
     if (Get_iData(IN_SUURYOU ) == 0) flag = FALSE;        // 数量
     CString  hinmei = Get_sData(IN_HINMEI);               // 品名
     if (hinmei.GetLength() == 0) flag = FALSE;

     if (flag == TRUE) OFF_wData(TOUROKU, 0);              // 操作可
     else              ON_wData (TOUROKU, 0);              // 操作不可

     int  tanka   = Get_iData(IN_TANKA  );                 // 合計 算出
     int  suuryou = Get_iData(IN_SUURYOU);
     int  goukei  = tanka * suuryou;
     Set_iData(IN_GOUKEI, goukei);
}
```

すべての入力項目が設定されている場合に 登録 ボタンの操作を可能とします。
ボタンに設定された識別コードの WORD 型データ、最下位ビットを ON に設定すればボタンの入力は不可となります。 ボタンの文字も灰色表示となります。

3－8　帳票印字処理プログラム

レイアウト情報を使用した帳票印刷は、CRT 画面表示と同様に共有データ領域へ印字データを設定することで行います。

売上げ帳 の印刷は 関数 **PrintUriagecyo** で行います。 この関数は売上げ帳画面に配置されたボタンをマウス左ボタンでクリックされた時に実行されます。

3－8－1　帳票印刷プログラムの構造

帳票印刷プログラムは 売上げ帳 以外の帳票の場合でも同一の構造で作成することができます。

印刷用デバイスコンテキストの生成	**CPrintDraw** クラスのインスタンスを生成します。
レイアウトファイルの読込み	**CPtrnData** クラスのインスタンスを生成します。
印字開始	関数 PrintOpen を実行します。
	1ページ目の印字データを共有メモリへ設定します。
デバイスコンテキストへ出力	関数 PtrnDispData を実行します。
改ページ	関数 PrintNewePage を実行します。
	2ページ目の印字データを共有メモリへ設定します。
デバイスコンテキストへ出力	関数 PtrnDispData を実行します。
改ページ	関数 PrintNewePage を実行します。
	最終ページの印字データを共有メモリへ設定します。
デバイスコンテキストへ出力	関数 PtrnDispData を実行します。
印字終了	関数 PrintClose を実行します。

関数 **PrintOpen, PrintClose, PrintNewPage**
は印刷用デバイスコンテキスト **CPrintDraw** のメンバ関数です。
関数 **PtrnDispData** はレイアウトデータ管理クラス **CPtrnData** のメンバ関数です。

3－8－2　売上げ帳 帳票印刷プログラム

```
const   char*   PRINT_PTRN = "帳票_売上げ帳";                              // 帳票名称
const   int     PRINT_LINE =  12;                                          // 印字行数

void  CPtrnUriagecyo :: PrintUriagecyo ()                                  // 帳票印刷
{
     CPrintDraw  DC(this, "URIAGECYO");  ①                                 // プリンターデバイス
     CPtrnData   PtrnData(PRINT_PTRN);   ②                                 // レイアウト情報読込み

     if (DC.PrintOpen("売上げ帳", PtrnData.PtrnHedr.Parm1[1],   ③          // 印字開始処理
                                  PtrnData.PtrnHedr.Parm1[2]) == TRUE) {
          int  Page = 1 + (TotalLine-1) / PRINT_LINE;                      // 総ページ数
          for (int page=0; page<Page; ++page) {      ④
               Set_iData(P_PAGE1, page+1);                                 // ページ番号
               Set_iData(P_PAGE2, Page  );

               Clear_AllData(P_HIZUKE );     ⑤                            // 共有データ領域消去
               Clear_AllData(P_HINMEI );
               Clear_AllData(P_TANKA  );
               Clear_AllData(P_SUURYOU);
               Clear_AllData(P_GOUKEI );

               for (int line=0; line<PRINT_LINE; ++line) {      ⑥
                    int  datano = PRINT_LINE * page + line;
                    if (datano < TotalLine) {
                         CString  date;
                         date.Format("%02d / %02d", Denpyo[datano].date1,  // 日付変換
                                                    Denpyo[datano].date2);
                                                                      ⑦
                         Set_sData(P_HIZUKE +line, date);                  // 印字データ設定
                         Set_sData(P_HINMEI +line, Denpyo[datano].hinmei );
                         Set_iData(P_TANKA  +line, Denpyo[datano].tanka  );
                         Set_iData(P_SUURYOU+line, Denpyo[datano].suuryou);
                         Set_iData(P_GOUKEI +line, Denpyo[datano].goukei );
                    }
                    else {
                         Set_Disable(P_TANKA  +line, TRUE, FALSE);  ⑧     // 非印字設定
                         Set_Disable(P_SUURYOU+line, TRUE, FALSE);
                         Set_Disable(P_GOUKEI +line, TRUE, FALSE);
                    }
               }
               PtrnData.DispPtrnData(&DC);  ⑨                            // １ページ印刷

               if (Page == (page+1)) break;                              // 最終ページ
               else DC.PrintNewPage();          ⑩                        // 改ページ
          }
          DC.PrintClose();     ⑪                                         // 印刷終了
     }
}
```

① 印字用デバイスコンテキストを作成します。
引数“URIAGECYO”より環境設定ファイルを参照し印字に使用する用紙などの情報を取得します。

```
[PRINTER]
URIAGECYO=2,1,0,10,10                    ; 売上げ帳 印字
```

設定情報の数値は左側より次の内容を設定します。

①	用紙サイズ	1:A3 / 2:A4 / 3:A5 / 4:B4 / 5:B5
②	印字方向	0:縦 / 1:横
③	給紙装置	0:自動選択 / 1:上用紙ﾄﾚｲ / 2:中用紙ﾄﾚｲ / 3:下用紙ﾄﾚｲ 4:用紙カセット / 5:手差しﾌｨｰﾀﾞ / 6:ﾌﾛﾝﾄﾄﾚｲ
④	左右方向余白	印刷する用紙の大きさに対する比率(%)で設定します。
⑤	上下方向余白	

用紙サイズに 0 を設定すると、ﾌﾟﾘﾝﾀｰ使用開始時に［ﾌﾟﾘﾝﾀｰ設定］のダイアログボックスが表示され、ﾌﾟﾘﾝﾀｰの選択、用紙、印字方向などを設定することができます。

② ﾚｲｱｳﾄ処理ｸﾗｽ CPtrnData を作成します。 引数で指定されたﾚｲｱｳﾄﾌｧｲﾙが読込まれます。

③ 印字開始処理を行います。
1番目の引数は帳票名称でﾌﾟﾘﾝﾀｰｽﾌﾟｰﾗｰに渡されます。
2、3番目の引数でﾚｲｱｳﾄの大きさ（横×縦 単位:ﾄﾞｯﾄ数）を設定します。
印字用デバイスコンテキストでは用紙の大きさとﾚｲｱｳﾄの大きさを比較し拡大／縮小印字を行います。

④ ページ数分繰り返します。

⑤ 1ページ分の印字データ領域を初期化します。
WORD、数値データ領域には 0、文字列データ領域には NULL 文字が設定されます。

⑥ 1ページの行数分繰り返します。

⑦ 共有データ領域へ印字データを設定します。

⑧ 印字するデータが存在しない行では、文字列は空白で印字されますが、数値データは〝0〟が印字されます。 数値データも空白で印字するために非印字の設定を行います。

⑨ 印字用デバイスコンテキストへ出力を行います。

⑩ 改ページ処理を行います。

⑪ 印字終了処理を行います。 この時最終ページが印字されます。

3－9　制御用データ処理プログラム

制御データは画面処理部の外で作成します。（ソースファイル名 ： SampleDemo.cpp）

3－9－1　スレッドの起動

制御データを作成するためスレッドを起動します。
スレッドの起動はアプリケーションプログラムの起動時に SimplePack より呼ばれる 起動時処理関数 UserInit で行います。

```
static  UINT   SetsubiDataThread   (LPVOID);          // 設備データ入力スレッド

BOOL  UserInit ()                                     // 初期化処理
{
     AfxBeginThread(SetsubiDataThread, 0);            // データ処理スレッド起動
      return TRUE;
}
```

3－9－2　データ作成スレッド

データとして　デジタル入力信号 7点、アナログ入力信号 10点 を模擬的に作成します。

```
static  UINT  SetsubiDataThread (LPVOID)                    // 設備データ入力
  {
    int  timr = 0;      ①
    int  AI_Data[10];                                       // ＡＩ初期値設定
    for (int i=0; i<10; ++i) AI_Data[i] = 10 * i;

    Sleep(1000);    ②                                       // 起動待ち（１秒）
    for (;;) {
         Sleep(100);                                        // 100msec 周期
         if (SystemStop != 0) break;  ③                     // スレッド停止要求

         if (++timr >= 10) {                                // １秒周期
              timr = 0;
              int  dich = rand() % 7;   ④                   // ＤＩ擬似信号設定
              EOR_wData(DI+dich, 0x0001);

              for (int j=0; j<10; ++j) {    ⑤              // ＡＩ擬似信号設定
                  if (++AI_Data[j] >= 100) AI_Data[j] = 0;
                  Set_iData(AI+j, AI_Data[j]);

                  if (AI_Data[j] >= 80) Set_Color(AI+j, 1);  ⑥
                  else                  Set_Color(AI+j, 0);
              }
         }
    }
    return (0);
}
```

① 作業領域の初期化
② 画面処理が開始するまで待ちます。（特別な意味はありません。）
③ 100msec周期でシステム停止要求フラグを判定します。
CRT画面上で 終了 ボタンが押されると、この変数 SystemStop に１が設定されます。
④ 1秒周期でデジタル、アナログ信号の作成を行います。
デジタル信号は乱数を使用して 0～6 の数値を作成し、1秒毎に1個の信号を反転させます。 WORD 型データの下位2ビットは表示色を決めるために使用されています。
制御用画面ではデジタル信号を表示するマークに2系統の表示色を設定していますから最下位ビットを反転することで表示色が切り替ります。
⑤ アナログ信号10点の数値を 0～99 の範囲で設定します。
⑥ アナログ信号が 80～99 の時 WORD 型データ最下位ビットを ON に設定することで表示色を切り替えています。

3－9－3　設備図画面

設備図画面で表示するデータはすべてデータ処理部（制御データ作成スレッド）で作成しています。したがって、設備図画面を処理するクラスは作成せずに、基本クラス CPtrnDisp で画面処理を行うことができます。

```
CPtrnDisp*  GetPtrnWnd (LPCTSTR name)                          // ウィンドウの生成
{
   CPtrnDisp*  PtrnWnd = NULL;
   if      (lstrcmp(name, "メニュー画面"    ) == 0) PtrnWnd = new CPtrnDisp();
   else if (lstrcmp(name, "売上げ帳"        ) == 0) PtrnWnd = new CPtrnUriagecyo();
   else if (lstrcmp(name, "伝票入力（新規）") == 0) PtrnWnd = new CPtrnDenpyo();
   else if (lstrcmp(name, "伝票入力（修正）") == 0) PtrnWnd = new CPtrnDenpyo();
   else if (lstrcmp(name, "設備図"          ) == 0) PtrnWnd = new CPtrnDisp();
   return PtrnWnd;
}
```

メニュー画面、設備図画面は画面処理用のクラスを派生させずに基本クラスで処理することができます。　派生クラスの作成が必要かどうかは次の要因で決まります。

① 処理種別コード

CRT 画面上に配置された表示項目に設定される処理種別コードにより派生クラスが必要になります。

No.	処理種別コード	派生クラス	備考
1	自ウィンドウへメッセージ送信	必要	
2	子ウィンドウ1へメッセージ送信	不要	他のウィンドウより
3	子ウィンドウ2へメッセージ送信	不要	メッセージが送られる
4	親ウィンドウへメッセージ送信	不要	場合には必要
5	子ウィンドウ1を開く	不要	
6	子ウィンドウ2を開く	不要	
7	システムウィンドウを開く	不要	
8	最上位ウィンドウを開く	不要	
9	自ウィンドウを閉じる	不要	
10	子ウィンドウ1を閉じる	不要	
11	子ウィンドウ2を閉じる	不要	
12	システムウィンドウを閉じる	不要	
13	バージョン情報画面を開く	不要	
14	デバッグ画面を開く	不要	
15	ハードコピー	不要	
16	システム終了	不要	

② CRT 画面へ表示するデータの作成が必要な場合

売上げ帳画面のようにスクロールバーに連動して表示情報を共有データ領域に設定する必要がある場合には派生クラスの作成が必要です。

④ キーボード入力を継続する場合

伝票入力画面ではキーボード入力を継続することができるようにするために派生クラスを作成しています。 基本クラスでもキーボード入力処理、入力範囲のエラーチェック、ボタン操作の親ウィンドウへの通知は可能です。

4. デモ用プログラム

デモ用プログラムを完成させるためにソースプログラムの修正、再コンパイルを行います。

4－1　ソースプログラム修正

売上げ帳画面、伝票入力画面を追加します。

ヘッダーファイル名 ： **SimpleDemo.h**

```
#include    <SimplePack.h>
#include    "SimpleCode.h"
#include    "Uriagecyo.h"    ①
```

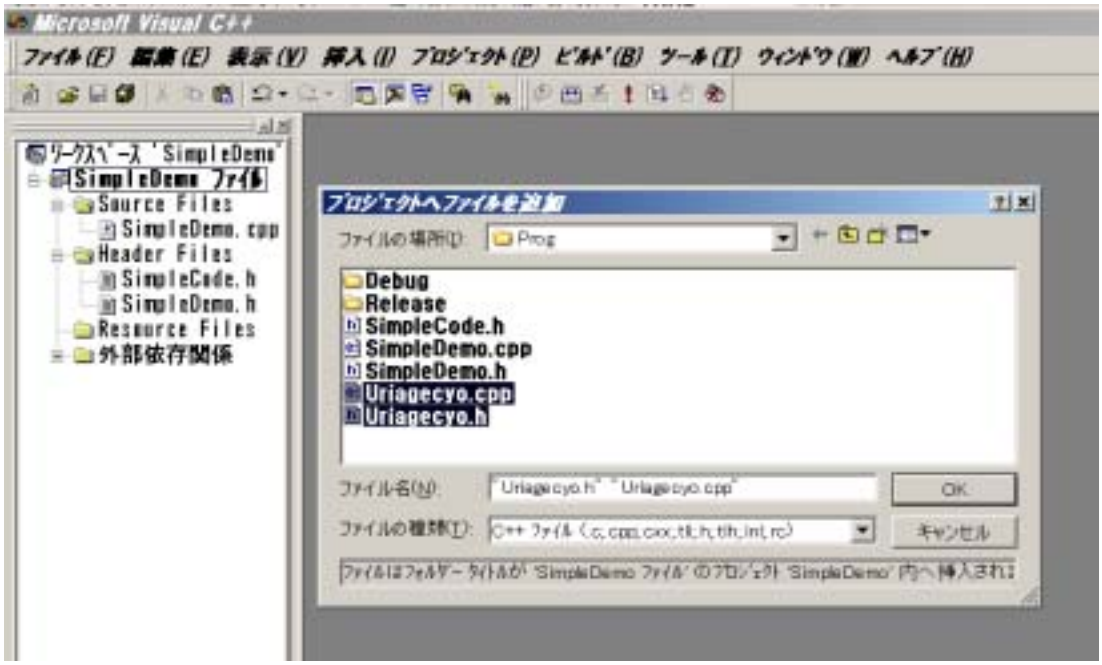
ソースファイル名 ： **SimpleDemo.cpp**

```
static  UINT    SetsubiDataThread   (LPVOID);   ②          // 設備データ入力スレッド

BOOL  UserInit ()                                          // 初期化処理
{
     AfxBeginThread(SetsubiDataThread, 0);    ③           // データ処理スレッド起動
     return TRUE;
}

void  UserExit ()   { }                                    // 終了時処理

void  UserTimer ()   { }                                   // 100msec タイマー処理


CPtrnDisp*  GetPtrnWnd (LPCTSTR name)                      // ウィンドウの生成
{
     CPtrnDisp*  PtrnWnd = NULL;
     if      (lstrcmp(name, "メニュー画面"    ) == 0) PtrnWnd = new CPtrnDisp();
     else if (lstrcmp(name, "売上げ帳"        ) == 0) PtrnWnd = new CPtrnUriagecyo();
     else if (lstrcmp(name, "伝票入力（新規）") == 0) PtrnWnd = new CPtrnDenpyo();   ④
     else if (lstrcmp(name, "伝票入力（修正）") == 0) PtrnWnd = new CPtrnDenpyo();
     else if (lstrcmp(name, "設備図"          ) == 0) PtrnWnd = new CPtrnDisp();
     return PtrnWnd;
}
```

① 売上げ帳、伝票入力画面処理プログラムの宣言部
② 設備画面用データ作成スレッドのプロトタイプ宣言
③ 設備画面用データ作成スレッドの起動
④ 売上げ帳、伝票入力画面処理クラスの関連付け

4－2　コンパイラの設定

プロジェクト SimpleDemo へ2本のソースプログラム Uriagecyo.h, Uriagecyo.cpp を追加します。



4－3　デモ用プログラムの実行

ソースプログラムを追加し再コンパイルを行うとデモ用プログラムが完成します。
実行前に環境設定ファイルに帳票印刷用の設定が必要です。

SimpleDemo.ini

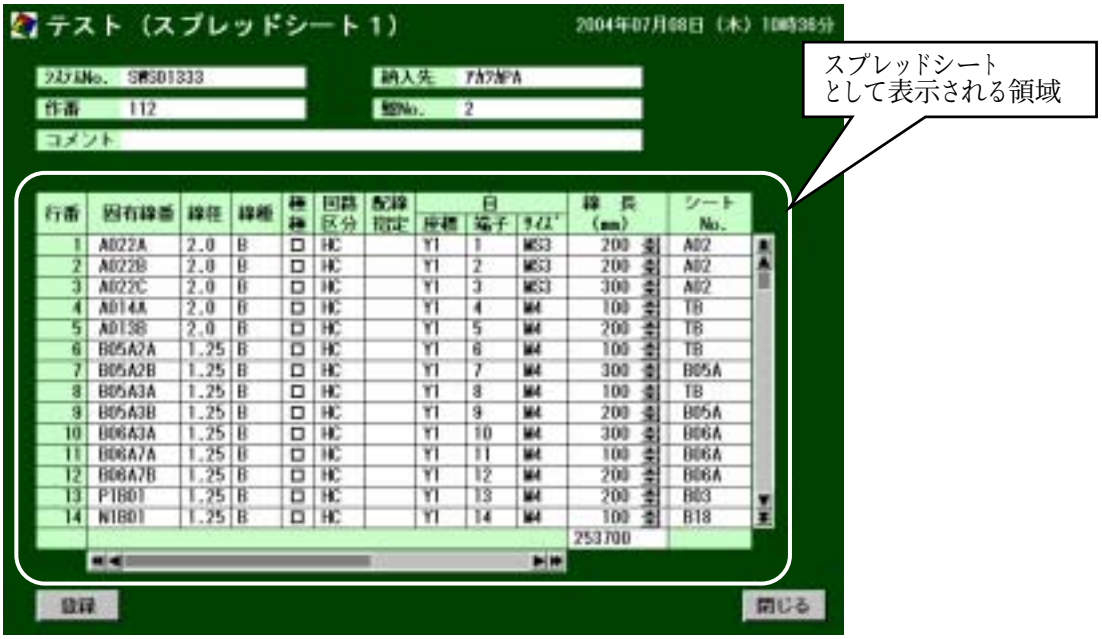
```
[PRINTER]
URIAGECYO=2,1,0,0,0                    ; 売上げ帳
```

（設定値の詳細は 3－7－2 を参照してください。）

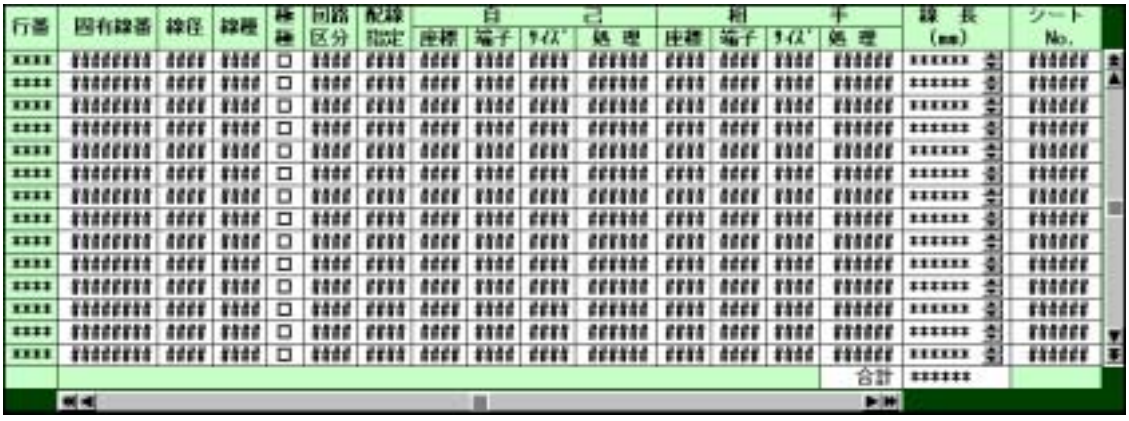
5　スプレッドシート

5－1　概要

SimpleMaster では次のようなスプレッドシートを作成し、親ウィンドウの指定された位置に表示することができます。



スプレッドシート用レイアウト



スプレッドシートには次の情報を表示することができます。

- 可変文字列　　キーボード入力可
- 数値データ　　キーボード入力可、スピンボタンの付加が可能
- チェックボックス
- マーク（2値のビットマップ）

スプレッドシート上には通常のレイアウトを配置することができます。
上記の例では、1／2行目のタイトル、下部の合計値、罫線は通常のレイアウト情報で描画しています。

スプレッドシートでは表の一部をスクロール禁止に設定することができます。
上記の例では、上部2行、下部1行、左側1列、右側2列 をスクロール禁止に設定しています。

スプレッドシート上では次の処理を行うことができます。
（アプリケーションプログラム側での記述は不要です。）

1. 可変文字列、数値データなどの表示、定周期での表示更新
2. 可変文字列、数値データに対するキーボード入力
   マウスで入力を行うセルを選択（画面を開いた時に自動起動することも可能です。）
   入力終了キー（Enter / TAB / Shift+TAB / カーソルキー）に従って次に入力を行うセルを選択します。 この時、セルは非表示の状態であれば自動スクロールします。
   入力を自動継続しない、または自動で選択するセルとは別のセルで入力を継続することも可能です。（アプリケーションプログラム側での記述が必要）
3. 上下･左右方向へのスクロール（マウスホィールも使用可能）

アプリケーションプログラム側では次の処理の記述が必要となります。

1. 親ウィンドウ側で表示位置を指定してスプレッドシートを子ウィンドウとして表示します。
2. 実際に表示を行うデータ行数を指定します。
3. スプレッドシートに表示するデータを共有データ領域に書込み。
4. キーボード入力結果を共有メモリ領域から読込み。

スプレッドシートからは、マウス操作時、キーボード入力完了時にメッセージを出すことができます。
前ページの例では 線長の合計値の算出はキーボード入力完了、およびスピンボタンの操作をメッセージとして受け取り、アプリケーションプログラム側で再計算しています。

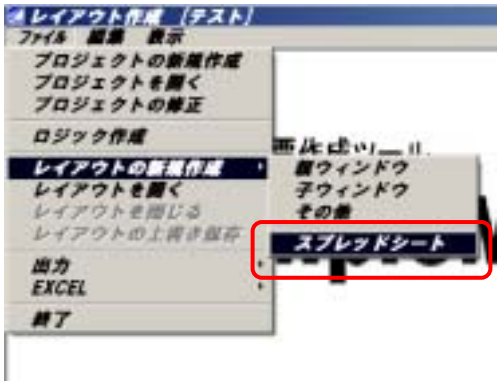
5－2　レイアウト作成

スプレッドシートのレイアウトは、レイアウト作成プログラム SimpleMaster で作成します。

5－2－1　レイアウトを開く

新規作成時

［ファイル］メニュー → ［新規作成］
→ ［スプレッドシート］
と選択することで詳細設定ダイアログボックスが開きます。



修正時

通常のレイアウトと同様に
［ファイルメニュー］ → ［レイアウトを開く］
によりスプレッドシート用のレイアウトを開きます。
Microsoft Explorer 上でレイアウトファイルをダブルクリックすることで開くこともできます。

［注意］　スプレッドシートのレイアウト情報は通常のレイアウト情報と同一種類のファイル（ファイル名拡張子 ptn）に格納されます。

5－2－2　詳細設定ダイアログボックス

スプレッドシートはダイアログボックスへの設定でレイアウトが自動生成されます。

詳細設定ボックスの各項目に値を設定することで表の大きさ、色などが決まります。

［データ列設定］ ボタンを押すと、個々の列に表示する情報を指定することができます。

［スクロールバー］ ボタンを押すことでスクロールバーの大きさや色などを設定することができます。

行の設定

1行の高さ	1行の高さをドット数で指定します。
データ行数	表として表示を行う行数を設定します。 （上部／下部の固定行数も含みます。）
上部／下部固定行数	表 j の上部、下部にスクロールを行わない行を指定します。 行数（不要の場合は 0 を設定）、および背景色を指定することができます。

列の設定

1桁の幅	1桁の幅をドット数で指定します。 表の大きさの管理は 行／桁数で行います。 データを表示する列はそれぞれ異なる桁数を設定することができます。
データ列数	データとして表示する行数を指定します。
左／右側固定行数	表の左右にスクロールを行わない列を指定することができます。

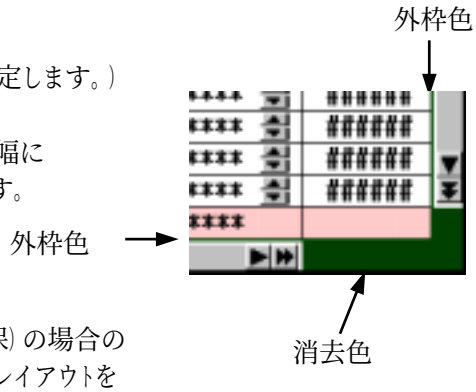
識別コード　データ列に表示する可変文字列などに割り当てる識別コードの総数を指定します。
表示データ設定方式により設定内容が異なります。（4 表示データ設定 を参照）

表示桁数　スプレッドシートを表示する時、親ウィンドウ上の座標を指定する方法と、矩形領域を領域を指定する方法があります。 開始座標のみを指定する場合に表示桁数が有効となります。（5－3 スプレッドシート表示 を参照）

消去色 ／ 外枠表示色

スクロールバーの非表示領域を塗りつぶす色を指定します。（通常は親ウィンドウの背景色を設定します。）

表の周囲には外枠を表示することができます、 枠幅に 0 以外の値が設定されている場合に有効となります。



未確定領域背景色　データ列未設定（桁数のみ確保）の場合の背景色を指定します。 画面のレイアウトを試作する場合などに使用します。

罫線表示色／罫線幅　表の内部に罫線を表示する場合に設定します。 罫線幅に 0 以外の数値が設定されている場合に有効となります。

文字フォント　データ列に表示される可変文字列、数値データに使用する文字フォントを指定します。（文字フォントは個別に指定することはできません。）

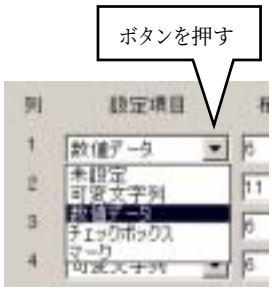
表示データ設定方式　表示データ設定方式 一括 または 個別 を設定します。
（5－4 表示データ設定 を参照）

5－2－3　　データ列設定

レイアウト詳細設定ダイアログボックスで［データ列設定］ボタンを押すと右の画面が表示されます。

データ列は最大 100項目まで設定することができますが、詳細設定ダイアログボックス / データ列数 で指定された列数のみが有効となります。

設定項目　　右端のボタンをマウス左ボタンでクリックすると選択画面が開きます。
可変文字列、数値データ、チェックボックスマーク より選択することができます。
未設定を選択した場合、桁数が設定されていれば表として表示することができます。



桁数　　データ列の桁数を設定します。
最大 99桁 まで指定することができます。

識別コード　　データとして表示する可変文字列などには識別コードを割当てる必要があります。
識別コード文字列と番号を指定します。
通常のレイアウト作成時と同様に右端のボタンをクリックすると登録済みの識別コード一覧表が表示され、一覧表より選択することができます。

詳細 ボタン　　可変文字列、数値データなどの設定項目に対応した詳細設定画面が開きます。
(次ページ参照)

挿入 ボタン　　選択されている列の直前に 未設定のデータ列 を挿入します。

削除 ボタン　　選択されているデータ列を削除します。

可変文字列 詳細設定

文字数(最大99文字)、表示色などを設定します。

表示色　　プロジェクトで 4系統 が指定されている場合、4系統となります。

数値データ 詳細設定

総桁数、小数点以下桁数、表示色などを設定します。

スピンボタン付き　数値データの右側にスピンボタンを表示します。
スピンボタンの大きさ(横×縦ドット数)を設定します。
数値データは入力不可でもスピンボタンから入力することは可能です。

メッセージ出力　スピンボタンが操作されたとき、メッセージを出力することができます。

チェックボックス 詳細設定

チェックボックスの大きさ、色を設定します。

マーク 詳細設定

マークの大きさ、種類、色を設定します。

5－2－4　　スクロールバー 詳細設定

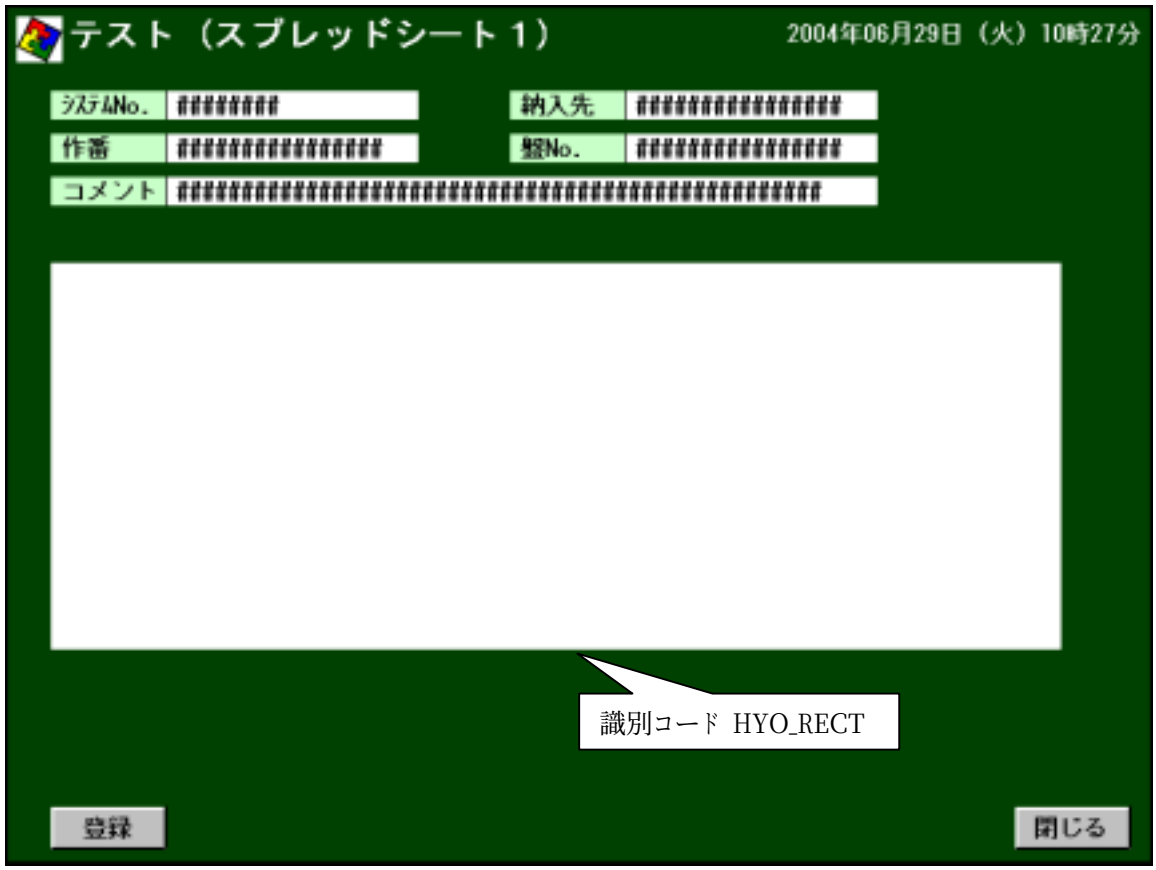
レイアウト詳細設定ダイアログボックスで［スクロールバー］ボタンを押すと右の画面が表示されます。

水平、垂直方向スクロールバーの表示方式、制御方式などを設定します。

5－3　スプレッドシートの表示

スプレッドシートは親ウィンドウ上で指定された位置に表示されます。

親ウィンドウ



スプレッドシートを表示するためにはアプリケーションプログラム側で次のようなロジックの記述が必要です。
この処理は 親ウィンドウ側 に記述します。

例 1

```
void  CSpreadSheet1 :: PtrnInitFunc ()                  // ウィンドウの初期化
{
    PTRNDATA  Ptrn = {32, 180};                         // 表示開始位置
    PtrnDispGrid(1, "スプレッドシート１", Point);        // スプレッドシート表示
```

例1 ではスプレッドシートの左上座標を指定して表示します。
スプレッドシートの大きさは次のようになります。

縦　　(1行の高さ) × (データ行数)
横　　(1桁の幅)　× (表示桁数)

データ行数、表示桁数はレイアウト詳細設定ダイアログボックスで設定します。

例1 の場合、上図で矩形領域を準備する必要はありませんが、スプレッドシートの表示開始位置を矩形領域の左上座標から取得することもできます。

例 2

```
void  CSpreadSheet2 :: PtrnInitFunc ()                    // ウィンドウの初期化
{
    PTRNDATA  Ptrn;                                       // 矩形領域表示情報
    PtrnReadData(HYO_RECT, PTRN_RECT, &Ptrn);
    PtrnDispGrid(1, "スプレッドシート２", Ptrn.Rect);
```

例2 ではスプレッドシートの左上座標を指定して表示します。

スプレッドシートの大きさは次のようになります。

縦　　(1行の高さ) × (データ行数)

横　　(1桁の幅)　× (表示可能桁数)

表示可能桁数 ＝ (矩形領域の幅) / (1桁の幅)

(余りは切上げ)

この方式では表の大きさは指定された矩形領域に収まるように自動的に調整されます。

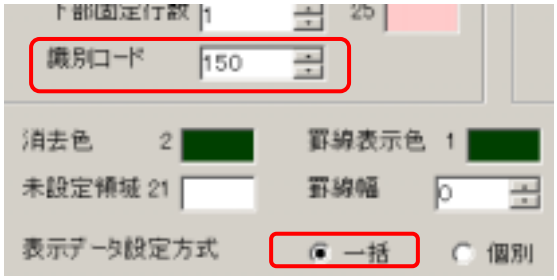
(1桁分大きくなる場合があります。)

5－4　表示データの設定

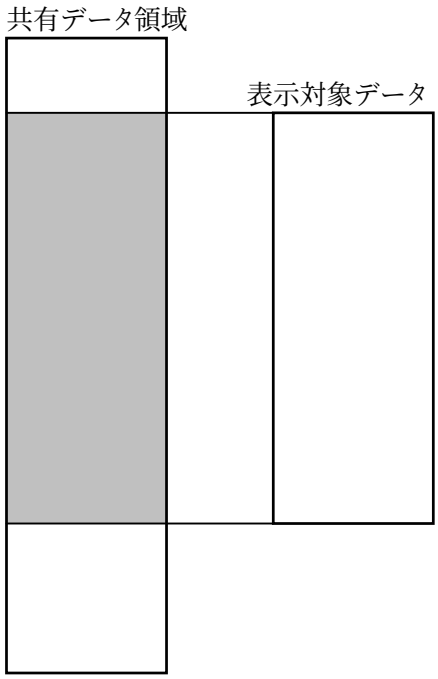
スプレッドシートに表示するデータの設定方式には、 **一括** と **個別** の2種類の方式が準備されています。

5－4－1　一括設定方式

一括設定方式では、スプレッドシートの表示開始前に表示するデータをすべて共有データ領域に設定しておきます。



上記の例では 識別コード として 150個 確保されます。 表示可能なデータ列は 150行以下 となります。

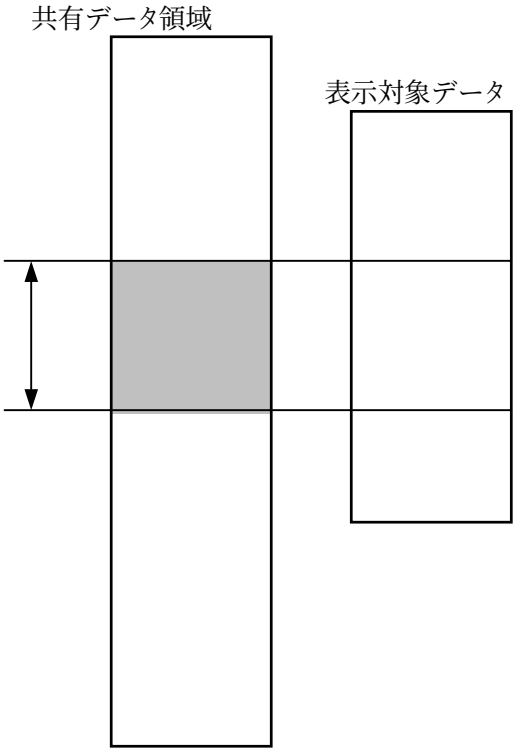


アプリケーションプログラムからは 次のプログラムで実際に表示する行数を指定することができます。

```
void  CHaisenhyo1 :: PtrnInitFunc ()              // ウィンドウの初期化
{
    GridSetScroll(100, 1);
}
```

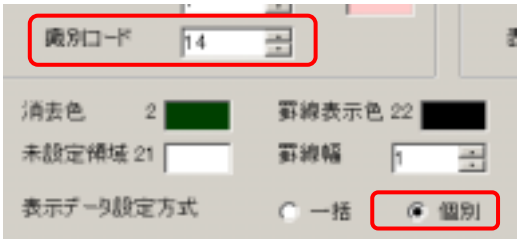
上記の場合、データ行数は 100行 となります。 また、2番目の引数に 1 が指定されているのでスプレッドシート表示後、ただちにキーボード入力が開始されます。
データ行数を設定しなければ、レイアウト詳細設定画面で指定された 識別コード定義数（上記の場合 150）が表示対象データ行数となります。
識別コード定義数よりも大きい値を指定しようとするとエラーとなり、ログへエラーメッセージが出力されます。 スプレッドシートは識別コード定義数をデータ行数に設定して処理を行います。

5－4－2　個別設定方式



個別設定方式では、スプレッドシートのスクロールに対応して表示データの読書きを行います。
表示対象データ数が不定の場合、または膨大で共有データ領域に一括して格納できない場合に使用します。

表示可能行数
（データ行数 － 固定行数）



個別設定方式を指定した場合、[登録] 時に識別コード定義数に次の値が設定されます。

（識別コード定義数） ＝ （データ列数） － （上部／下部固定行数）

アプリケーションプログラムからは 次のプログラムで実際に表示する行数を指定します。

```
void  CHaisenhyo1 :: PtrnInitFunc ()              // ウィンドウの初期化
{
    GridSetScroll(100, 1);
}
```

上記の場合、データ行数は 100行 となります。

個別設定方式では垂直方向スクロールバーの変化を検出して表示データを入れ替える必要があります。

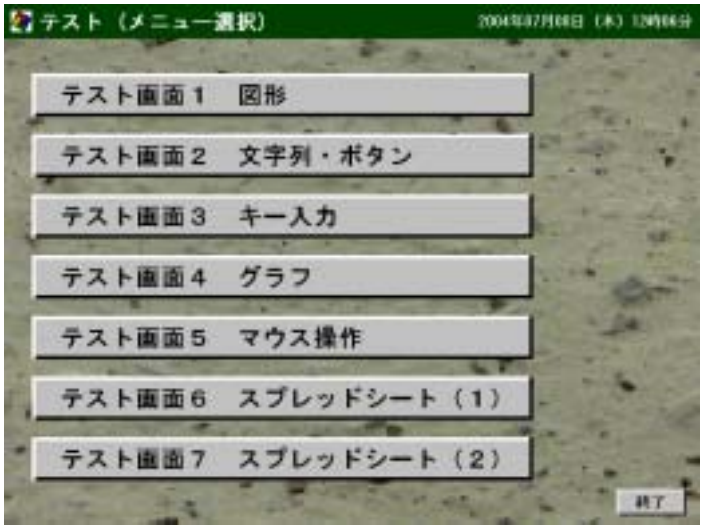
```
void  CHaisenhyo1 :: PtrnMesgFunc (int code)      // マウス操作による通知
{
    if (code == V_SCROLL) {                       // スクロールバー（縦）
        DispDataGet();                  ①
        DispDataSet(GridGetScroll());   ②
    }
```

① 現在表示中のデータを共有データ領域より取り出し表示対象データ領域に保存します。
キーボード入力可能なデータがある場合にのみ必要です。

② スクロールバーの表示開始行番号を取得し、1ページ分の表示対象データを共有データ領域へ設定します。

5－5　サンプルプログラム

テストプログラムには2種類のスプレッドシートの例が準備されています。



テスト画面6 スプレッドシート(1)
表示位置は 左上座標 で指定します。
表示データの設定は 個別方式で行います。
右側2列をスクロール禁止領域に設定しています。

テスト画面7 スプレッドシート(2)
表示位置は 親ウィンドウの矩形領域で指定します。
表示データの設定は 一括方式で行います。

ソースプログラムは haisenhyo.h / haisenhyo.cpp に格納されています。

スプレッドシートを処理するために次のようは専用の関数が準備されています。

No.	クラス名	関数名	処理内容	備考
1	CPtrnDisp	PtrnDispGrid	スプレッドシートの表示	親ウィンドウで実行
2		GeidSetScroll	実際に表示する行数の設定、キー入力の自動開始	スプレッドシートを処理する子ウィンドウで実行
3		GridGetScroll	表示開始行番号の取得	
4		GridReadParm	スプレッドシートの表示処理パラメータを取得　(下記参照)	
5		GridReadData	スプレッドシート データ列設定情報の取得	

キーボード入力開始 / 強制終了 などは通常のレイアウト処理関数を使用することができます。

関数　**GridReadOarm**　スプレッドシートの表示処理パラメータを取得

レイアウト設定画面で設定した内容などを取得することができます。

```
Parm = GridReadParm (data);
    int     Parm;       取得したパラメータ情報
    int     data;       取得したい情報
```

引数 data には次の定数を指定します。 (ヘッダーファイル Ptrndata.h に定義されています。)

No.	定数	取得できるパラメータ情報	備考
1	MODE	表示データ設定方式	0:一括 / 1:個別
1	HEIGHT	1行の高さ	ドット数
2	WIDTH	1桁の幅	ドット数

No.	定数	取得できるパラメータ情報	備考
3	DATALINE	データ行数	固定行を含む
4	FIXLINE1	上部固定行数	
5	FIXLINE2	下部固定行数	
6	DATACULM	データ列数	固定列を含む
7	FIXCULE1	左側固定列数	
8	FIXCULM2	右側固定列数	
9	TOTALSIZE	総桁数	
10	DISPLINE	表示データ行数	固定行を含まない
11	DISPSIZE	表示桁数	固定列を含む
12	DATACOLOR	未設定データ領域 背景色	
13	FIXCOLOR1	上部固定行 背景色	
14	FIXCOLOR2	下部固定行 背景色	
15	CLEARCOLOR	消去色番号	
16	GRIDCOLOR	罫線表示色番号	
17	BODERCOLOR	外枠表示色番号	
18	GRIDWIDTH	罫線 線幅	
19	BODERWIDTH	外枠 線幅	
20	DATACOUNT	表示対象データ数	
21	FONT	文字フォント番号	
22	HSCROLL	水平スクロールバー	0:表示 / 1:非表示
23	VSCROLL	垂直スクロールバー	0:表示 / 1:非表示

5－5－1　テスト画面6 スプレッドシート(1)

親ウィンドウ側

画面を開く時、閉じる時、マウス入力時の処理を記述します。

```
void  CSpreadSheet1 :: PtrnInitFunc ()                          // ウィンドウの初期化
{
①  CFileRead  HaisenHyo(ProjPath, "Haisenhyo");                 // 配線表ファイル読込み
   HaisenHyo.ReadBlock(&HaisenHedr, sizeof(HAISENHEDR));        // ヘッダー部

②  if (HaisenData != NULL) delete [] HaisenData;                // 作業領域破棄
   HaisenData = new HAISENDATA [HaisenHedr.datacount];          // データ部
   for (int i=0; i<HaisenHedr.datacount; ++i)
        HaisenHyo.ReadBlock(&HaisenData[i], sizeof(HAISENDATA));

③  Set_sData(SYSTEMNO,    HaisenHedr.systemno);                // 初期表示
   Set_sData(NOUNYUUSAKI, HaisenHedr.nousaki );
   Set_sData(SAKUBAN,     HaisenHedr.sakuban );
   Set_sData(BANNO,       HaisenHedr.banno   );
   Set_sData(COMMENT,     HaisenHedr.comment );

④  POINT  Point = {32, 180};                                    // 表示開始位置
   PtrnDispGrid(1, "スプレッドシート１", Point);
 }
```

①/②　配線表データが格納されたファイルを開きデータを読込みます。
③　　共通表示項目を共有データ領域に設定します。
③　　ウインドウ左上座標を指定してスプレッドシートを子ウィンドウとして表示します。

```
void  CSpreadSheet1 :: PtrnExitFunc ()                          // ウィンドウの終了処理
{
   if (HaisenData != NULL) delete [] HaisenData;                // 作業領域破棄
   HaisenData = NULL;
}
```

画面と開く時に確保していた領域を開放します。

```
void  CSpreadSheet1 :: PtrnMesgFunc (int code)                  // マウス操作による通知
{
①  if (code == TOUROKU) {                                       // 登録 ボタン
②       lstrcpy(HaisenHedr.systemno, (LPCTSTR)Get_sData(SYSTEMNO   ));   // 入力結果の取得
        lstrcpy(HaisenHedr.nousaki,  (LPCTSTR)Get_sData(NOUNYUUSAKI));
        lstrcpy(HaisenHedr.sakuban,  (LPCTSTR)Get_sData(SAKUBAN    ));
        lstrcpy(HaisenHedr.banno,    (LPCTSTR)Get_sData(BANNO      ));
        lstrcpy(HaisenHedr.comment,  (LPCTSTR)Get_sData(COMMENT    ));

③       CFileWrite  HaisenHyo(ProjPath, "Haisenhyo");           // 配線表ファイル書込み
        HaisenHyo.WriteBlock(&HaisenHedr, sizeof(HAISENHEDR));
        for (int i=0; i<HaisenHedr.datacount; ++i)
             HaisenHyo.WriteBlock(&HaisenData[i], sizeof(HAISENDATA));
        }
 }
```

①　登録 ボタンを押された時の処理を記述します。
②　共通表示項目を共有データ領域より読込みます。
③　配線表データをディスクファイルに保存します。

スプレッドシート(子ウィンドウ側)

画面を開く時、マウス入力時、キーボード入力終了時の処理を記述します。

```
void  CHaisenhyo1 :: PtrnInitFunc ()                    // ウィンドウの初期化
{
①  DispDataSet(0);                                          // 先頭ページ 表示データ設定
②  GridSetScroll(HaisenHedr.datacount, 1);             // 表示対象データ数設定

③  DispDataSum();                                       // 合計値計算}
```

① 先頭ページの表示データを設定します。 引数で表示開始行番号を指定します。
② スプレッドシートの表示対象データ数を設定します。 2番目の引数に 1 を指定するとスプレッドシート先頭部の入力可能セルよりキーボード入力処理を開始します。
③ 線長データの合計値を計算します。

```
void  CHaisenhyo1 :: PtrnMesgFunc (int code)            // マウス操作による通知
{
① DispDataGet();                                        // 表示データ取得

② int  cont = GridReadParm(DISPLINE);                   // 表示行数
③ if (code == V_SCROLL)                                 // スクロールバー（縦）
        DispDataSet(GridGetScroll());

④ else if ((code >= HYO_DATA15)                         // スピンボタン
        && (code < (HYO_DATA15+cont))) DispDataSum();    // 合計値計算
}
```

① 表示中の情報を共有データ領域から配線表データ領域へ読込みます。
② スプレッドシート1ページ当たりの表示行数を取得します。
③ 垂直方向スクロールバーが操作された時、スクロールバーの現在位置情報(表示開始行番号)を引数に設定してスプレッドシート1ページ分の表示データを共有データ領域に設定します。
④ 線長データ部のスピンボタンが操作された時、合計値の再計算を行います。

```
BOOL  CHaisenhyo1 :: PtrnKeyFunc (int code)             // キー入力終了処理
{
① DispDataGet();                                        // 表示データ取得

② int  cont = GridReadParm(DISPLINE);                   // 表示行数
  if ((code >= HYO_DATA15)
   && (code < (HYO_DATA15+cont))) DispDataSum();

③  return TRUE;
}
```

① キーボード入力終了時、表示中データを共有データ領域より配線表データ領域へ読込みます。 次項の線長合計値再計算、親ウィンドウで 登録 ボタンがいつ押されても良いように、表示中データに変化があれば読込みを行います。
② 線長データに対しキーボード入力が行われた時、合計値の再計算を行います。
③ TRUE を返すとパッケージプログラム SimpleMaster 側でキーボード入力の継続を行います。

スプレッドシート(子ウィンドウ)側では、表示データの設定／取得、線長合計値算出のため関数を定義しています。

```
void  CHaisenhyo1 :: DispDataSet (int dispno)              // 表示データ設定
{
①  DispDtno = dispno;

②  int  cont = GridReadParm(DISPLINE);                   // 表示行数
③  for (int i=0; i<cont; ++i) {
      int  j = DispDtno + i;
      Set_iData(LINENO    +i, DispDtno+i+1);
      Set_sData(HYO_DATA1 +i, HaisenData[j].senban);
        (一部省略)
   }
}
```

① 引数で渡された表示開始行番号を保存します。
② スプレッドシート1ページ当たりの表示行数を取得し、
③ 表示開始行番号より1ページ分の配線表データを共有データ領域に設定します。

```
void  CHaisenhyo1 :: DispDataGet ()                        // 表示データ取得
{
①  int  cont = GridReadParm(DISPLINE);                   // 表示行数
②  for (int i=0; i<cont; ++i) {
      int  j = DispDtno + i;
      HaisenData[j].length = Get_iData(HYO_DATA15+i);
      lstrcpy(HaisenData[j].senban, (LPCTSTR)Get_sData(HYO_DATA1 +i));
      lstrcpy(HaisenData[j].senkei, (LPCTSTR)Get_sData(HYO_DATA2 +i));
      lstrcpy(HaisenData[j].sensyu, (LPCTSTR)Get_sData(HYO_DATA3 +i));
        (一部省略)
   }
 }
```

① スプレッドシート1ページ当たりの表示行数を取得し、
② 表示開始行番号より1ページ分の配線表データを共有データ領域より取得します。

```
void  CHaisenhyo1 :: DispDataSum ()                        // 線長データ 合計値再計算
{
   int  goukei = 0;
   for (int i=0; i<HaisenHedr.datacount; ++i) goukei += HaisenData[i].length;
   Set_iData(GOUKEI, goukei);
   }
```

配線表 線長データの合計値を計算し、共有データ領域へ設定します。

5－5－2　テスト画面7 スプレッドシート(2)

親ウィンドウ側

画面を開く時、閉じる時、マウス入力時の処理を記述します。

```
void  CSpreadSheet2 :: PtrnInitFunc ()                          // ウィンドウの初期化
{
①  CFileRead  HaisenHyo(ProjPath, "Haisenhyo");                 // 配線表ファイル読込み
   HaisenHyo.ReadBlock(&HaisenHedr, sizeof(HAISENHEDR));        // ヘッダー部

②  if (HaisenData != NULL) delete [] HaisenData;               // 作業領域破棄
   HaisenData = new HAISENDATA [HaisenHedr.datacount];          // データ部
③  for (int i=0; i<HaisenHedr.datacount; ++i) {
        HaisenHyo.ReadBlock(&HaisenData[i], sizeof(HAISENDATA));

        Set_iData(LINENO    +i, i+1);                           // 表データ設定
        Set_sData(HYO_DATA1 +i, HaisenData[i].senban);          //  （一括方式）
        Set_sData(HYO_DATA2 +i, HaisenData[i].senkei);
         （一部省略）
   }

④  Set_sData(SYSTEMNO,    HaisenHedr.systemno);                // 初期表示
   Set_sData(NOUNYUUSAKI, HaisenHedr.nousaki );
   Set_sData(SAKUBAN,     HaisenHedr.sakuban );
   Set_sData(BANNO,       HaisenHedr.banno   );
   Set_sData(COMMENT,     HaisenHedr.comment );
   ShowWindow(SW_SHOW);

⑤  PTRNDATA  Ptrn;                                              // 矩形領域表示情報
   PtrnReadData(HYO_RECT, PTRN_RECT, &Ptrn);
   PtrnDispGrid(1, "スプレッドシート２", Ptrn.Rect);
}
```

① 配線表データが格納されたファイルを開きデータを読込みます。
② 配線表データを読込む領域を確保します。
③ 配線表データを読込みながら、共有データ領域に設定します。
④ 共通表示項目を共有データ領域に設定します。
⑤ 親ウインドウ上の矩形領域の座標を指定してスプレッドシートを子ウィンドウとして表示します。

```
void  CSpreadSheet2 :: PtrnExitFunc ()                          // ウィンドウの終了処理
{
   if (HaisenData != NULL) delete [] HaisenData;                // 作業領域破棄
   HaisenData = NULL;
}
```

画面と開く時に確保していた領域を開放します。

```
void  CSpreadSheet2 :: PtrnMesgFunc (int code)                    // マウス操作による通知
{
① if (code == TOUROKU) {                                          // 登録 ボタン
②      lstrcpy(HaisenHedr.systemno, (LPCTSTR)Get_sData(SYSTEMNO   ));
       lstrcpy(HaisenHedr.nousaki,  (LPCTSTR)Get_sData(NOUNYUUSAKI));
       lstrcpy(HaisenHedr.sakuban,  (LPCTSTR)Get_sData(SAKUBAN    ));
       lstrcpy(HaisenHedr.banno,    (LPCTSTR)Get_sData(BANNO      ));
       lstrcpy(HaisenHedr.comment,  (LPCTSTR)Get_sData(COMMENT    ));

③      CFileWrite  HaisenHyo(ProjPath, "Haisenhyo");              // 配線表ファイル書込み
       HaisenHyo.WriteBlock(&HaisenHedr, sizeof(HAISENHEDR));
       for (int i=0; i<HaisenHedr.datacount; ++i) {
            HaisenData[i].length = Get_iData(HYO_DATA15+i);       // 表データ取得
④           lstrcpy(HaisenData[i].senban, (LPCTSTR)Get_sData(HYO_DATA1 +i)); //  (一括方式)
            lstrcpy(HaisenData[i].senkei, (LPCTSTR)Get_sData(HYO_DATA2 +i));
            lstrcpy(HaisenData[i].sensyu, (LPCTSTR)Get_sData(HYO_DATA3 +i));
             (一部 省略)

            if (Get_iData(HYO_DATA4+i) == 0) lstrcpy(HaisenData[i].kyoku, "0");
            else                             lstrcpy(HaisenData[i].kyoku, "1");

⑤           HaisenHyo.WriteBlock(&HaisenData[i], sizeof(HAISENDATA));
       }
   }
}
```

① 登録 ボタンを押された時の処理を記述します。
② 共通表示項目を共有データ領域より読込みます。
③ 配線表ファイルをオープンし
④ 共有データ領域より配線表データを読込みながら
⑤ ファイルへ書込みます。

スプレッドシート(子ウィンドウ側)

画面を開く時、マウス入力時、キーボード入力終了時の処理を記述します。

```
void  CHaisenhyo1 :: PtrnInitFunc ()                              // ウィンドウの初期化
{
①  GridSetScroll(HaisenHedr.datacount, 0);                        // 表示対象データ数設定
②  DispDataSum();                                                 // 合計値計算
}
```

① スプレッドシートの表示対象データ数を設定します。 2番目の引数に 0 を指定するとキーボード入力処理は開始されません。
③ 線長データの合計値を計算します。

```
void  CHaisenhyo1 :: PtrnMesgFunc (int code)                      // マウス操作による通知
{
   if ((code >= HYO_DATA15)                                       // スピンボタン
    && (code < (HYO_DATA15+ HaisenHedr.datacount))) DispDataSum();  // 合計値計算
}
```

線長データ部のスピンボタンが操作された時、合計値の再計算を行います。

```
BOOL  CHaisenhyo1 :: PtrnKeyFunc (int code)                       // キー入力終了処理
{
   if ((code >= HYO_DATA15)                                       // 線長入力
    && (code < (HYO_DATA15+ HaisenHedr.datacount))) DispDataSum();  // 合計値計算
}
```

線長データに対しキーボード入力が行われた時、合計値の再計算を行います。

スプレッドシート(子ウィンドウ)側では、線長合計値算出のため関数を定義しています。

```
void  CHaisenhyo1 :: DispDataSum ()                    // 線長データ 合計値再計算
{
   int  goukei = 0;
   for (int i=0; i<HaisenHedr.datacount; ++i) goukei += Get_iData(HYO_DATA15+i);
   Set_iData(GOUKEI, goukei);
}
```

共有データ領域の線長データの合計値を計算し、共有データ領域へ設定します。

SimpleMaster へのお問合せは下記へお願いします。

株式会社	フジテクニカ
住所	〒８１２－００１３
	福岡市博多区博多駅東２丁目９番２５号
TEL	092-431-0800
FAX	092-431-2018
URL	http://www.fuji-technica.jp/
E-Mail	master@fuji-technica.jp