DENSO

デンソーロボット

RC7コントローラ用

操作盤機能説明書

取扱説明書（追補版）

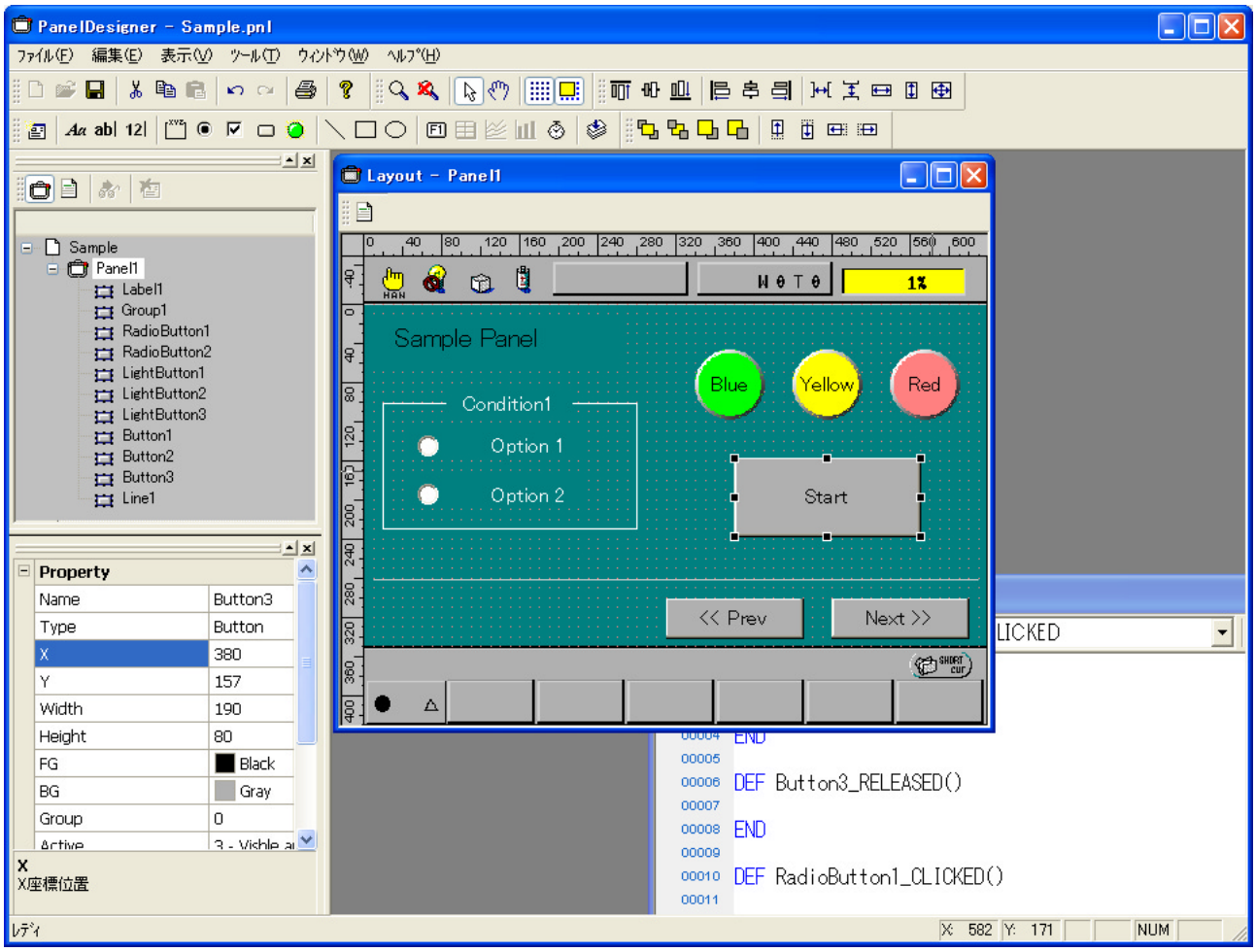
はじめに

バージョン Ver.2.2*から WINCAPSⅡに操作盤エディタが組み込まれました。操作盤エディタは、ティーチングペンダントの操作盤画面をパソコン上で作成するソフトウェアです。部品を画面上に配置し、各部品に行わせる動作を記述するだけで簡単に操作盤データを作成することができます。本書はこの操作盤機能について説明します。

目次

第1章 操作盤エディタ

操作盤エディタは、バージョン2.2からWINCAPSⅡに装備されたティーチングペンダントの操作盤画面をパソコン上で作成するソフトウェアです。部品を画面上に配置し、各部品に行わせる動作を記述するだけで簡単に操作盤データを作成することができます。ここではその操作方法について説明します。



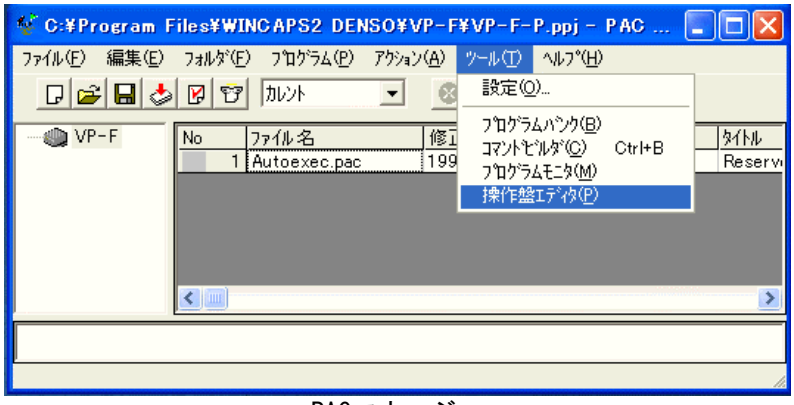
操作盤エディタで作成の操作盤画面例

1.1　操作盤データ作成手順の概要

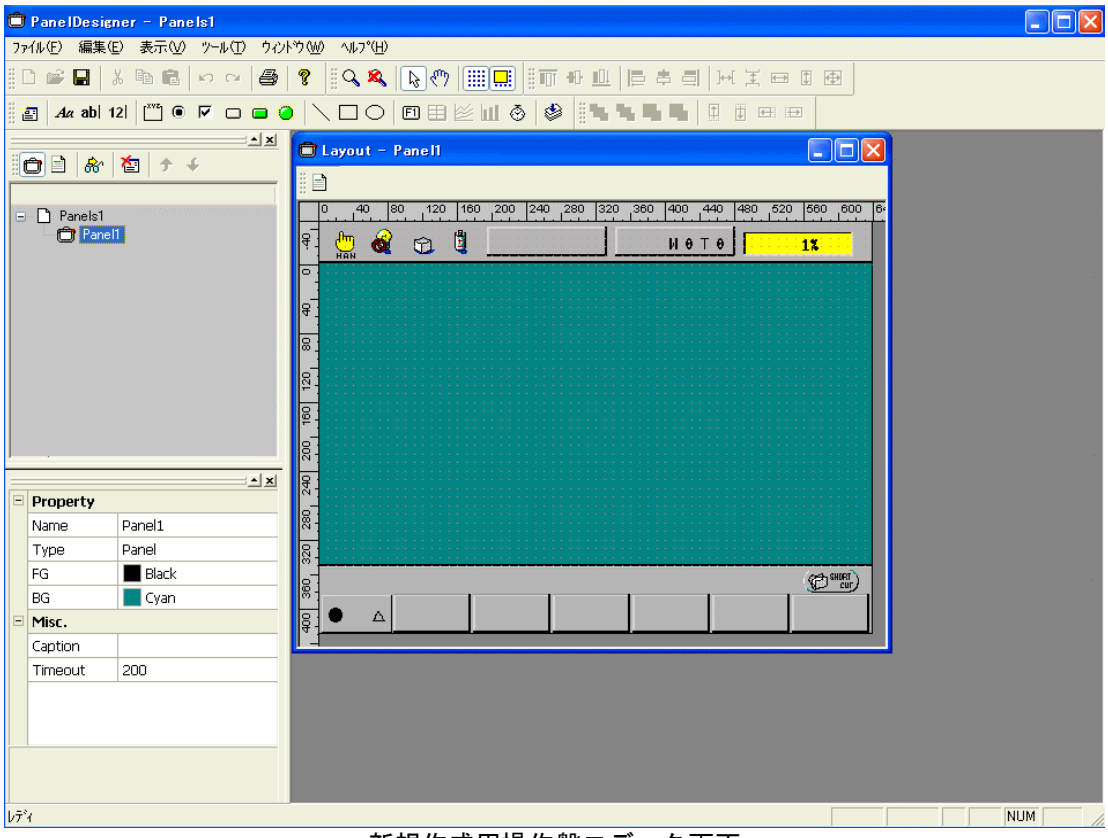
操作盤データを作成する手順の概要を(1)～(5)に示します。

(1)　操作盤エディタの起動

①WINCAPS Ⅱ のPACマネージャを開きます。
②[ツール]→[操作盤エディタ]を選択し、操作盤エディタを起動します。
③[ファイル]→[新規作成]を選択すると、ティーチングペンダント用の
操作盤エディタ画面(Layout Panel)を表示します。
注1：この画面は、[PACマネージャ]→[プログラム]→[新規作成]→[操作盤]からも表示ができます。
注2：**既存の操作盤データを開く場合は、**[ファイル]→[開く]を選択します。



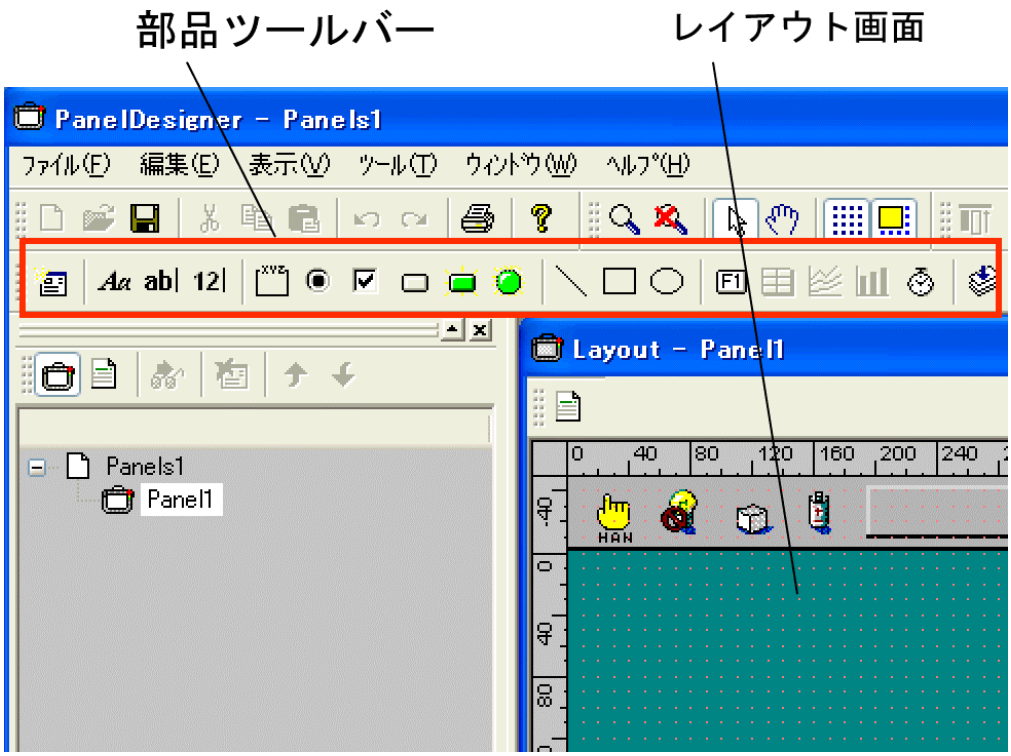
PACマネージャ



新規作成用操作盤エディタ画面

(2)　操作盤画面の作成

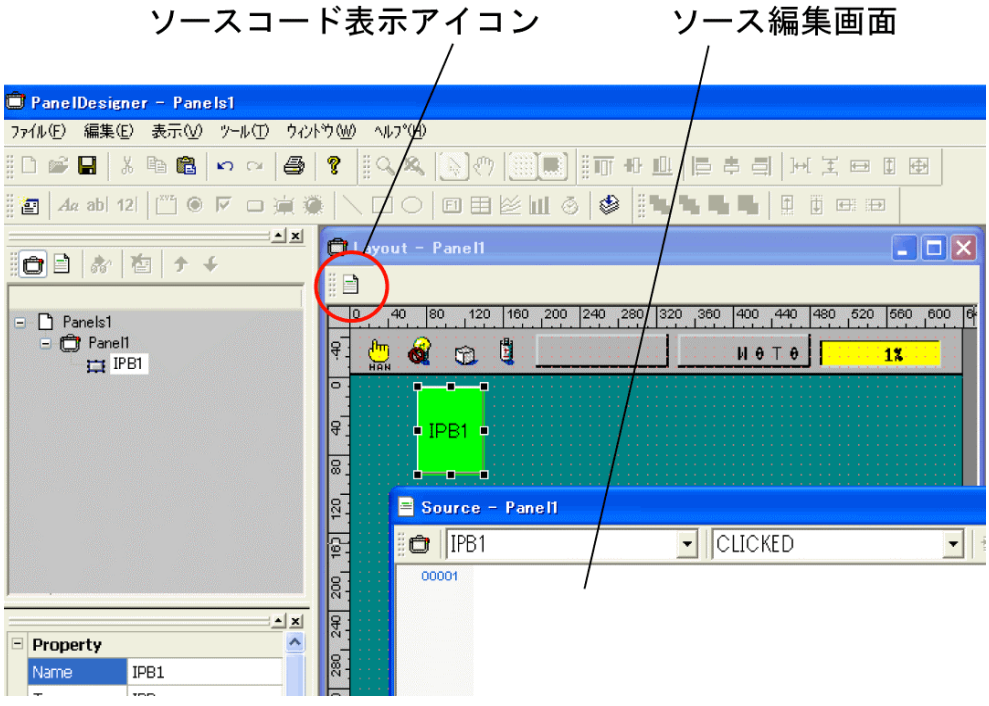
操作盤エディタに準備されている[部品ツールバー]から必要な部品を指定し、レイアウト画面上に配置して操作盤の画面を作成します。 詳細は「第2章　操作盤エディタで操作盤をつくる」を参照してください。



(3)　動作コードの編集

①レイアウト画面のソースコード表示アイコンをクリックし、ソース編集画面を開きます。

②部品が押された時に行わせる動作コードを、ソース編集画面に記述します。詳細は「2.2.2　部品のアクションの指定」を参照してください。



(4)　コンパイル

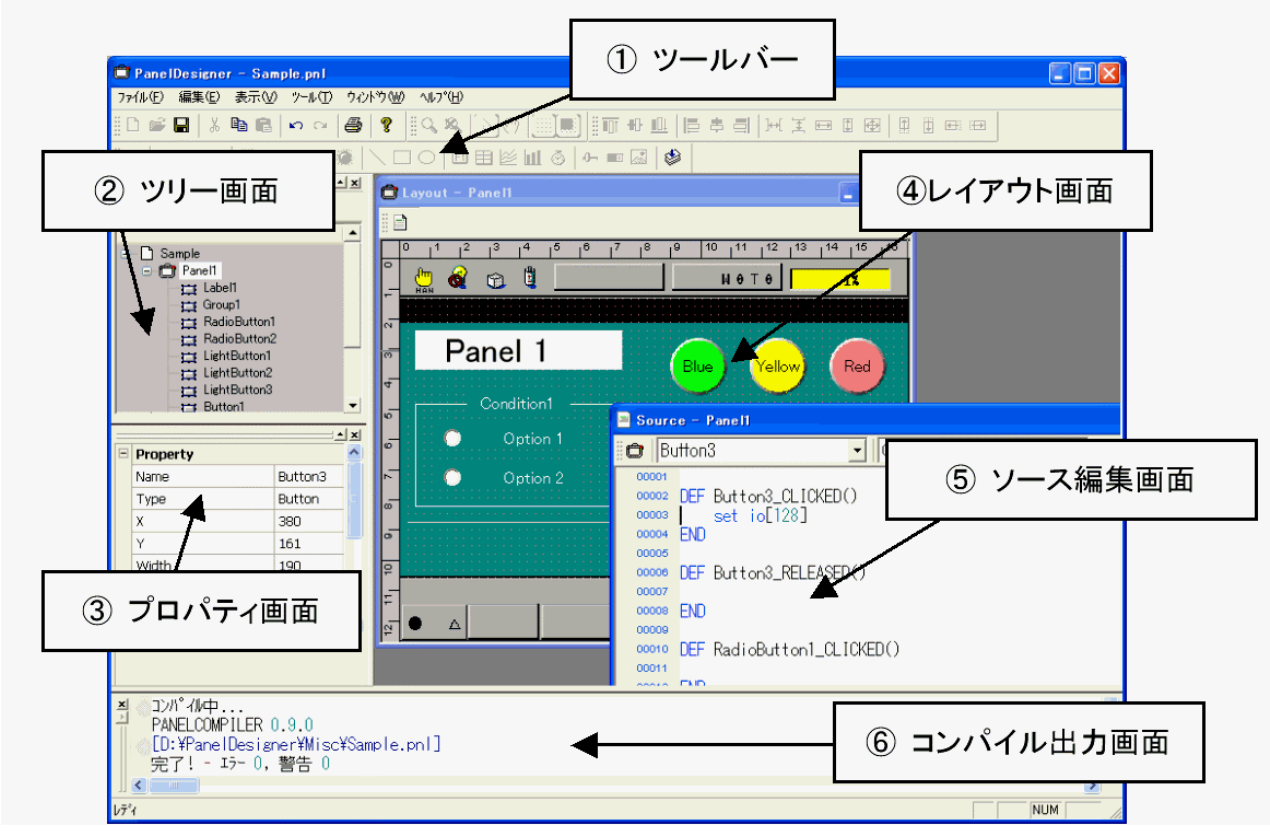
記述した動作コードに誤りがないかをコンパイルして確認します。
コンパイル結果は、操作盤エディタ画面下部の出力ウインドウに表示されます。

(5)　コントローラへデータ送信

作成した操作盤ファイルをコントローラに送信します。　これにより、ティーチングペンダントを操作盤として使用することが可能(**注**)になります。
送信は、WINCAPS Ⅱから他のプログラムと合わせて送信します。
注：操作盤機能有効化のティーチングペンダントの設定が必要です。

1.2　操作盤エディタ画面の機能説明

下図に操作盤エディタのレイアウトを示します。各画面の機能について以下に記します。



操作盤エディタのレイアウト

1.2.1 ツールバー

操作盤エディタ画面には、操作盤データ作成のために便利な各種のツールバーを準備しています。

(1) 標準ツールバー

操作盤エディタの標準ツールバーの機能を以下に示します。

	名称	説明
	新規作成	新しい操作盤ファイルを作成します。
	開く	既存の操作盤ファイルを開きます。
	保存	編集中のファイルを上書き保存します。
	切り取り	選択範囲を切り取り、クリップボードに移動します。
	コピー	選択範囲をクリップボードにコピーします.
	貼り付け	クリップボードの内容を現在のカーソル位置に挿入します。
	元に戻す	直前に行った操作を元に戻します。
	やり直し	元に戻した操作を再度、実行します。
	印刷	現在の画面を印刷します。
	バージョン情報	操作盤エディタのバージョン情報を表示します。

(2) 「ズーム・グリッド」ツールバー

レイアウト画面のサイズ変更、グリッド表示のON/OFFを切り替えます。

	名称	説明
	ズーム	選択領域を拡大・縮小表示します。
	標準倍率	ズーミングを取り消し、標準表示（100%）に戻します。
	パン	表示画面を指定の方向に動かします。
	グリッド ON/OFF	グリッドの表示／非表示を切り替えます。
	グリッド位置合わせ	部品をグリッド上に位置合わせします。

(3) 「レイアウト」ツールバー

選択されている部品の位置やサイズを合わせます。

	名称	説明
	上端合わせ	選択した部品の上端を統一します。
	中心合わせ － 垂直	選択した部品の上下中心を統一します。
	下端合わせ	選択した部品の下端を統一します。
	左端合わせ	選択した部品の左端を統一します。
	中心合わせ － 水平	選択した部品の左右中心を統一します。
	右端合わせ	選択した部品の右端を統一します。
	水平間隔	選択した部品の横間隔を統一します。
	垂直間隔	選択した部品の縦間隔を統一します。
	幅調整	選択した部品の幅を統一します。
	高さ調整	選択した部品の高さを統一します。
	サイズ調整	選択した部品のサイズを統一します。

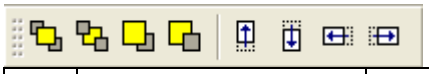
(4)　部品ツールバー

レイアウト画面に配置する部品を選択します。

	名称	説明
	新規パネル	新規レイアウトページを作成します。
	部品の選択	レイアウト画面を選択モードに切り替えます。
	ラベル	部品ボタン：ラベル
	テキストボックス	部品ボタン：テキストボックス
	数値入力ボックス	部品ボタン：数値入力ボックス
	グループボックス	部品ボタン：グループボックス
	ラジオボタン	部品ボタン：ラジオボタン
	チェックボックス	部品ボタン：チェックボックス
	プッシュボタン	部品ボタン：プッシュボタン
	照光式ボタン	部品ボタン：照光式ボタン
	パイロットランプ	部品ボタン：パイロットランプ
	直線	部品ボタン：直線
	四角形	部品ボタン：長方形
	楕円	部品ボタン：楕円
	ファンクションキー	部品ボタン：ファンクションキー
	タイマー	部品ボタン：タイマー
	コンパイル	作成した操作盤データをコンパイルします。

(5)　移動ツールバー

レイアウト画面に配置されている部品の位置や上下関係を変更します。



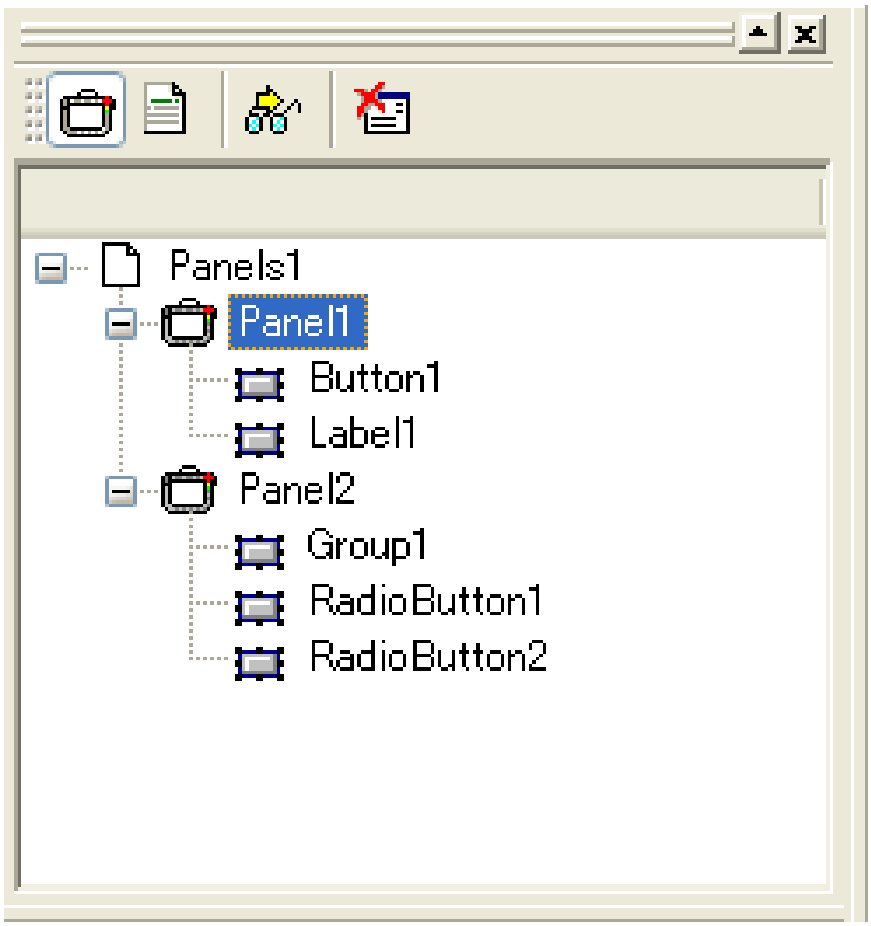
	名称	説明
	最前面へ	選択した部品を最前面に配置します。
	最背面へ	選択した部品を最背面に配置します。
	前面へ	選択した部品を１つ前面に移動します。
	背面へ	選択した部品を１つ背面に移動します.
	上へ	選択した部品を、上方向に移動させます。 [Shift]キーが押されていると 5 ポイントづつ移動します。
	下へ	選択した部品を、上方向に移動させます。
	左へ	選択した部品を、上方向に移動させます。
	右へ	選択した部品を、上方向に移動させます。

1.2.2　ツリー画面

レイアウト画面に配置された部品をツリー表示します。

(1)　ツリー画面

ツリー画面を下図に示します。
部品をダブルクリックすると、対応した画面が表示されます。



(2)　ツリー画面用ツールバー

ツリー画面用ツールバーと機能を下表に示します。

	名称	説明
	レイアウトフォーム	レイアウト画面を選択
	ソースフォーム	ソース編集画面を指定
	パネル表示	上記選択画面を表示します.
	パネル削除	レイアウト画面を削除します.

1.2.3　プロパティ画面

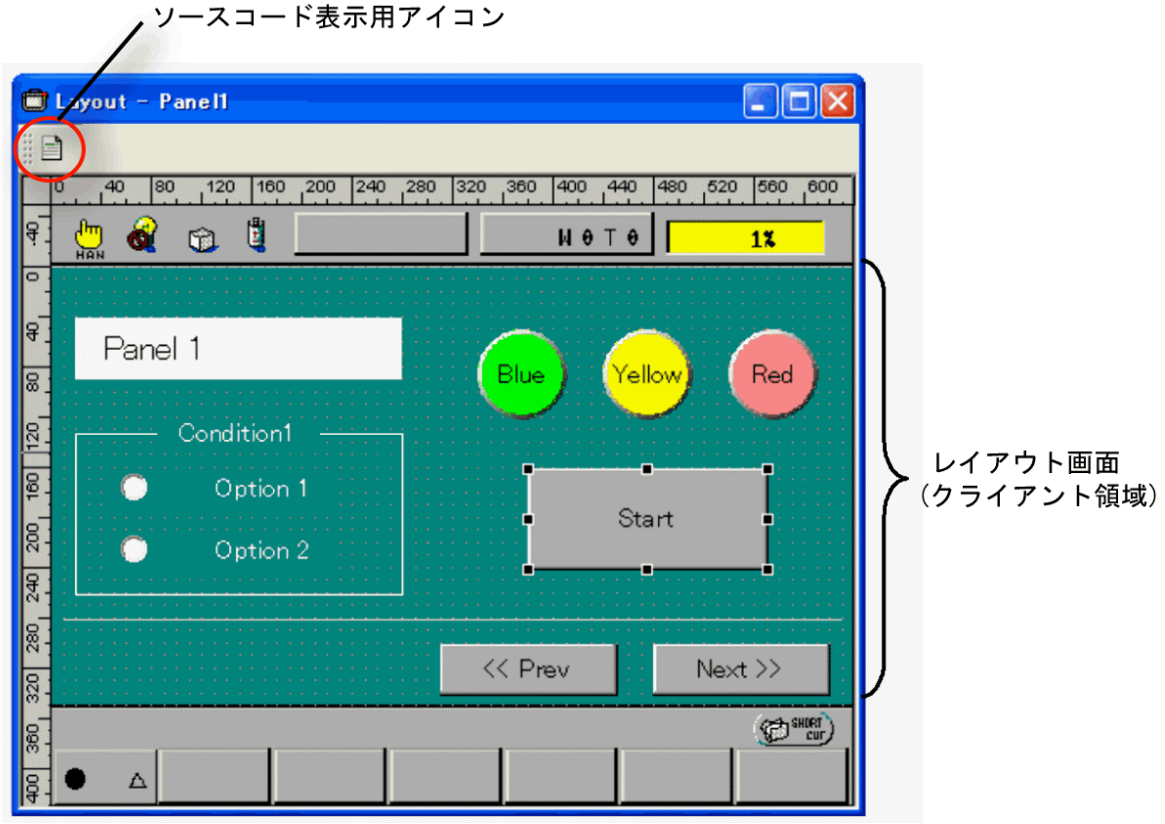
部品の位置やサイズ等の属性を指定します。表示される項目は部品の種類によって異なります。詳細は「1.5.1　プロパティ一覧」を参照してください。

プロパティ画面

1.2.4　レイアウト画面

(1)　ペンダントに表示される「操作盤画面」をデザインする画面です。
この画面に選択した部品を配置します。配置した部品は、カーソルキーやラバーバンドのドラッグによって位置やサイズの調整ができます。

(2)　ソースコード表示用アイコンをクリックすると、画面に対応したソース編集画面を表示します。



レイアウト画面

1.2.5　ソース編集画面

レイアウト画面に配置した部品に行わせる動作を記述する画面です。

ソース編集画

(1)　ソース編集画面のツールバー

	名称	説明
	レイアウト画面	画面に対応した レイアウト画面 を表示します。
	右インデント	選択行を 1 タブ分右インデントします。
	左インデント	選択行を 1 タブ分左インデントします。
	コメントアウト	選択行をコメントアウトします。
	非コメントブロック	選択行のコメントアウトを取り消します。
	ブックマーク	ブックマークの設定と解除を行います。
	次のブックマーク	カーソルを次のブックマークに移動します。
	前のブックマーク	カーソルを前のブックマークに移動します。
	ブックマーク解除	すべてのブックマークを解除します。
	検索・置換	指定された文字列を検索または置換します。

注：ブックマークを設定すると、コード行の左に が表示されます。

(2)　部品選択コンボボックス

コンボボックスから動作を記述する部品を選択します。

(3)　アクション選択コンボボックス

［(2)部品選択コンボボックス］の部品で指定可能なアクション一覧が表示されます。選択すると、エディタ画面に［(4)動作記述ブロック］の枠組みが自動生成されます。

［例］：挿入される動作記述ブロックの枠組み(Button1 がクリックされた場合)

```
DEF Button1_CLICKED()

END
```

(4)　動作記述ブロック

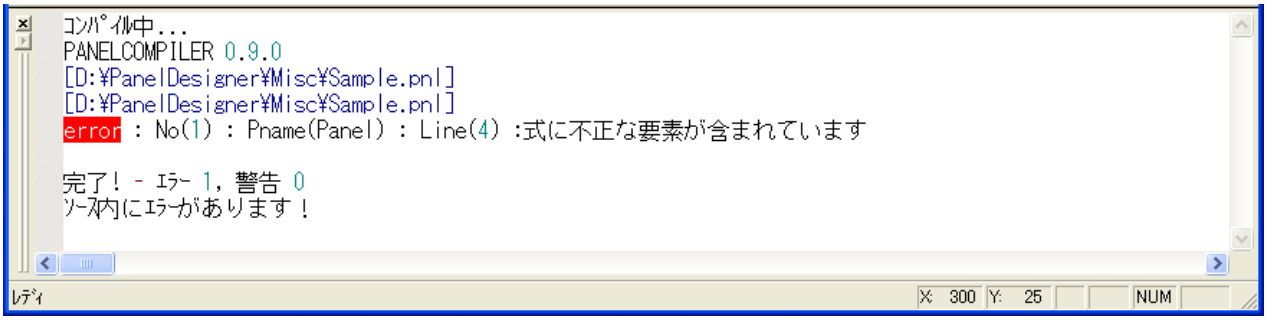
［(3)アクション選択コンボボックス］で生成された動作記述ブロック内に、行わせたコードを記述します。

［例］：挿入される動作記述ブロックの枠組み（Button1 がクリックされた場合）

```
DEF Button1_CLICKED()
   Set IO[128]        'IO 128 番を ON
   Run PRO100         'PRO100 を起動
END
```

1.2.6　コンパイル出力画面

作成した操作盤データのコンパイル結果を表示します。
エラー行をダブルクリックすることにより［ソース編集画面］の対応するエラー箇所へジャンプします。



コンパイル出力画面

1.2.7 メニュー説明

ここでは、操作盤エディタ画面のメニューについて説明します。

(1) ファイル（F）

メニュー		内容
	新規作成(N)	新しい操作盤ファイルを作成します。
	開く(O)...	既存の操作盤ファイルを開きます。
	閉じる(C)	編集中のファイルを閉じます。ファイルが保存されていない場合は保存用ダイアログが表示されます。
	上書き保存(S)	編集した内容をファイルに保存します。新規作成の場合は、保存用ダイアログが表示されます。
	名前を付けて保存(A) ...	編集した内容を別のファイルとして保存します。
	印刷(P)...	選択されている「レイアウト画面」または「ソース編集画面」の内容を印刷します。
	印刷プレビュー(V)...	印刷したときのイメージを表示します。
	プリンタの設定(R) ...	プリンタの設定ダイアログを表示します。
	インポート(I) ...	他の操作盤ファイルから「レイアウト画面」を取り込みます。
	最近使ったファイル	過去に保存した操作盤ファイルの一覧を表示します。
	アプリケーションの終了(X)	操作盤エディタを終了します。

(2) 編集（E）

メニュー		内容
	元に戻す(U)	直前の操作を元に戻します。
	やり直し(R)	元に戻した操作をもう一度行ないます。
	切り取り(T)	選択範囲を切り取ってシステムのクリップボードに移動します。
	コピー(C)	選択した部品や文字列をシステムのクリップボードにコピーします。
	貼り付け(P)	クリップボードの内容を現在のカーソル位置に挿入します。
	削除(D)	選択した部品や文字列を削除します。
	検索・置換(F)	指定した文字列の検索や置換を行うダイアログを表示します。

(3) 表示（V）

メニュー		内容
	ツールバー	各ツールバーの表示／非表示を切り替えます。
	ステータスバー(S)	ステータスバーの表示／非表示を切り替えます。
	ツリーバー(T)	ツリー画面の表示／非表示を切り替えます。
	プロパティバー(P)	プロパティ画面の表示／非表示を切り替えます。
	パネルレイアウト(A)	選択したレイアウト画面を表示します。
	グリッド(G)	レイアウトグリッドの表示／非表示を切り替えます。
	グリッド位置合わせ(S)	グリッド自動位置合わせ機能のオン／オフを切り替えます。
	ズーム － 標準に戻す(Z)	レイアウト画面を標準倍率で表示します。
	ズーム － 拡大率の指定(O)	レイアウト画面の表示倍率を指定します。

(4)　ツール（T）

メニュー		内容
	設定(O)	コンパイラの出力バージョンを選択します。
	コンパイル(C)	作成した操作盤データを実行形式に翻訳します。

(5)　ウインドウ（W）

メニュー		内容
	閉じる(O)	選択されているウインドウを閉じます。
	すべてを閉じる(L)	表示されている全ウインドウを閉じます。
	重ねて表示(C)	開いている全ウインドウを同じ大きさでタイトルバーが見えるように重ねて表示します。
	並べて表示(T)	開いているウインドウを並べて表示します。
	アイコンの整列(A)	最小化されているウインドウをメインウインドウ左下に整列します。
	表示画面一覧	表示されているウインドウ一覧を表示します。

(6)　ヘルプ（H）

メニュー		内容
	ヘルプ	操作盤エディタのヘルプを表示します。
	バージョン情報	操作盤エディタのバージョン画面を表示します。

1.3　レイアウトを作成・変更する

1.3.1　部品を追加する

レイアウト画面に新たに、部品を追加する手順を(1)～(3)に示します。

(1) レイアウト画面を表示する

作成、変更するレイアウト画面を表示させます。

①規作成時

［ファイル］メニューの［新規作成］、または［：新規パネル］ツールバーを選択します。

②存画面の変更時

ツリー画面で、［：レイアウトフォーム］を選択後、表示させる［Panel1：レイアウトツリー］をダブルクリック、または［：パネル表示］ボタンをクリックします。

(2) 部品を選択する

部品ツールバーから追加する部品を選択します．選択後はレイアウト画面上のカーソルに部品マーク［］が表示されます。

(3) 部品を追加する

レイアウト画面上で部品を配置する箇所をクリックすると、デフォルトのサイズで部品が追加されます。

注：追加の際にドラッグすると、その時点でサイズ調整ができます。

1.3.2 部品のレイアウトを変更する

レイアウト画面に配置した部品の位置やサイズを下記の何れかの方法で変更します。

(1)移動する

①マウスによるドラッグ（移動カーソル 表示中）
②カーソルキー；l
③移動ツールバー（ ）
④プロパティのX、Y項目の値を変更

(2)サイズを変更する

①部品の外枠のラバーバンドをドラッグ
②プロパティのWidth、Height項目の値を変更
③複数部品選択時は、レイアウトツールバーにより間隔、サイズの一括調整が可能（)

(3)位置合わせ

複数部品選択時は、レイアウトツールバーによりセンタリングや端の一括調整が可能 （)

注：[ファンクション]部品は、[Index]プロパティで位置、サイズが決定します。

(4)部品の順序（重なり合い）を変更する

レイアウト画面上で変更する部品を選択後、右クリックメニューの[移動]または[：順序ツールバー]から操作を選択します。

注： 順序を変更すると、それに応じてツリー画面の部品の並びが変更されます。

1.3.3 部品の属性を変更する

プロパティ画面の項目で、部品名称や色等の属性を変更することができます。

1.3.4 レイアウト画面を削除する

ツリー画面で削除する[Panel1：レイアウトツリー]を選択後、[：パネル削除]ボタンをクリックします。

1.3.5 他の操作盤ファイルからレイアウト画面をインポートする

他の操作盤ファイル(拡張子=pnl)から、レイアウト画面単位でデータをインポートすることができます。

(1)インポートする操作盤ファイルの選択

[ファイル]メニューの[インポート]から、インポート元のファイルを選択します。

(2)レイアウト画面の選択

ファイルに含まれるレイアウト一覧から、インポートするレイアウトを選択し、[Import]ボタンをクリックします．部品ツリーにインポートしたレイアウトが追加されます。

1.4　動作コードを記述する

操作盤画面上の部品に、「押された時」や「離された時」などの状態が変化した時に行わせる処理を、ソース編集画面で記述します。

1.4.1　動作コードの記述方法

(1) ソース編集画面の表示

下記の何れかの方法で、動作を記述する部品に対応した、[ソース編集画面]を表示します。

①レイアウト画面から部品をダブルクリックします.

②レイアウト画面で部品を選択後、[：レイアウト表示]ボタンをクリックします。

③部品ツリー画面で部品ツリーを選択後、[：ソースフォーム]ボタンが押されていることを確認し、[：パネル表示] ボタンをクリックします。

(2) 部品の選択

ソース編集画面の、部品選択コンボに動作を記述する部品が表示されていることを確認します. 表示されていない場合、コンボボックスから選択します。

(3) アクションの選択

アクション選択コンボボックスには、使用可能なアクション一覧が表示されます。動作を記述するアクションを選択すると、エディタ画面に動作記述ブロックの２行が自動挿入されます。

```
例:Button1 の CLICKED を選択した場合.
    DEF Button1_CLICKED()

    END
```

(4) 動作コードの記述

(3)で挿入されたブロック間に、行わせる動作コマンドを記述します。

```
例:Button1 の CLICKED を選択した場合.
    DEF Button1_CLICKED()
        Set IO[128]         'IO128 番を ON
        Run PRO100          'PRO100 を起動
        Run PRO200          'PRO200 を起動
    END
```

1.4.2　記述したコードの確認（コンパイル）

記述した動作コードの文法をチェックし、結果を[コンパイル出力画面]に表示します。エラーがある場合は、ウインドウ内にエラー内容が表示されます。
エラー行をダブルクリックすることにより、[ソース編集画面]の対応する箇所が表示されます。

1.5 その他

1.5.1 プロパティー覧

プロパティ画面に表示される位置、サイズ等のプロパティ一覧を下表に示します。

注：プロパティ画面に表示される属性は、部品の種類によって異なります。

名 称	内 容	備 考
Name	名称	部品の識別名称
Type	部品の種類	部品毎に固定です。
X	X 座標	ティーチングペンダント描画範囲内における X、Y 軸の基準位置です。
Y	Y 座標	
Widht	幅	基準位置からのサイズをピクセル単位で設定します。
Height	高さ	
FG	前景色	コンボボックスから色を選択します。
BG	背景色	
Group	グループ番号	部品が属するグループの番号です。
Active	有効／無効設定	コンボボックスから選択します。
Style	表示スタイル	コンボボックスから表示形式を選択します。
Caption	表示文字列	部品の表面に表示される文字です。 注：複数行対応の場合は[Ctrl+Ret]で改行
FSize	文字サイズ	0：SS、1：S、2：M、3：L
Justify	表示文字位置	0：中央、1：右寄せ、2：左寄せ
Thickness	線幅	線の太さをピクセル単位で設定します。 注：0 の場合は、塗りつぶし
MyGroup	管理グループ番号	グループボックス固有
State	状態	ON/OFF 等の状態をコンボボックスから選択します。
Value	入力値	数値入力ボックス固有
Text	入力テキスト	テキストボックス固有
Index	ファンクション番号	ファンクションキー固有
Interval	インターバル時間	タイマー固有
Timeout	タイムアウト時間	背景固有

1.5.2 アクション一覧

アクションとは、ボタンが「押された」「離された」等の動作を示します。
下表にアクションの一覧を示します。

注：使用可能なアクションは、部品の種類によって異なります。

アクション名	動作内容
CLICKED	部品が押された
RELEASED	部品が離された
TIMER	インターバル時間が経過した
REFRESH	画面リフレッシュが行われた

1.5.3 動作コマンドの書式

動作コマンドは、「**部品の状態取得／変更**」と「**操作盤制御コマンド**」の組み合わせにより記述します。

注：使用可能なアクションは、部品の種類によって異なります。

(1)部品の状態取得／変更

部品の参照は、[部品名称. プロパティ] の書式で記述します。

例 1 ： ラジオボタン（RadioBtn）の ON/OFF 状態を取得する。

```
DEFINT iState
IState = RadioBtn.State
```

例 2 ： ボタン（Button）幅を 200 に設定する。

```
Button.Width = 200
```

(2)操作盤制御コマンド

操作盤制御言語の文法は「第4章」、コマンド一覧は「5.1項」を参照してください。

1.5.4 コントローラへのデータ送信

作成した「操作盤データ」は、WINCAPSⅡを使ってコントローラへ送信します。WINCAPSⅡのプロジェクト内に「操作盤データ」があれば、PACプログラム送信時に「操作盤データも合わせてコントローラに送信されます。

注：操作盤エディタでデータを変更した場合は、データ送信前に必ず「保存」を行ってください。WINCAPSⅡはデータ送信時に、最後に保存された状態でコンパイルを行うため、保存されていないと修正前のデータがコントローラに送信されます。

1.5.5 制限事項

ラジオボタンの状態(State)は、デフォルトボタンに対してのみONしてください。操作盤エディタでは、複数のラジオボタンをONさせることが可能ですが、その状態のままコントローラへ送信すると「ペンダント操作盤画面」も複数ONした表示になります。

第2章 操作盤エディタで操作盤を作る

第1章ではWINCAPSⅡに組み込まれている操作盤エディタの機能について説明しました。本章ではこの操作盤エディタを使用して、実際にティーチングペンダントに操作盤を作成するための詳細を説明します。

ロボットのティーチングペンダントで任意の大きさ･位置･色を指定できる種々の部品を持つ汎用操作盤を実現します。画面設計はパソコン上でマウス操作により行います。
設計される画面データ（画面数は複数）は，1フォルダに1つのみ持つことが可能です。
作成された画面データには，複数の画面情報とそれぞれの画面に貼り付けられる部品（オブジェクト）情報が格納されています。
画面設計機能・部品属性（大きさ・位置・色など）設定は「第1章　操作盤エディタ」を参照してください。

2.1　ティーチングペンダントの設定

2.1.1　操作盤機能の有効化

操作盤機能はオプション機能ですので，事前にティーチングペンダントの基本画面から以下の手順で機能を有効化する必要があります。

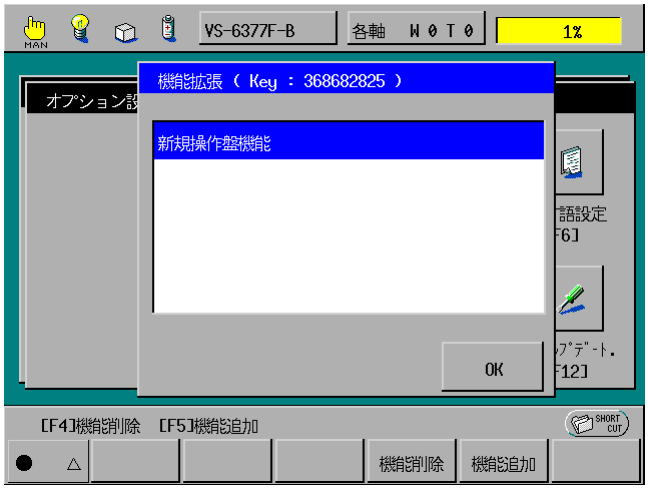
ステップ 1

［F6 設定］―［F7 オプション］―［F8 機能拡張］―［F5 機能追加］を選択し，機能拡張画面を表示させます。

ステップ 2

暗証番号「1453」を入力してください。

[OK]を押すと、操作盤機能が追加されます。



ステップ 3

基本画面でF5「各個」が「操作盤」に切り替わっていることが確認できます。

これで、[F5：操作盤]を押すと操作盤画面が起動できます。また，操作盤画面から抜けるにはペンダント画面外の「SHIFT」キーを押しながら「Cancel」キーを押してください。

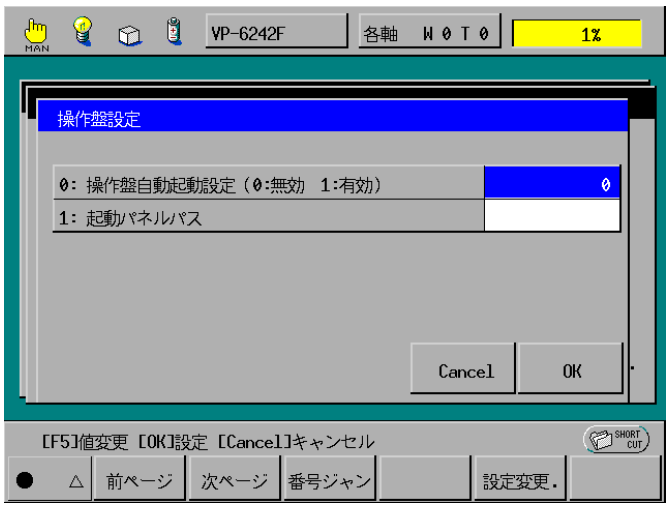
注：操作盤機能を選択した場合は，F9に割り当てられたTP簡易操作盤機能（RC5互換の操作盤）は使用できなくなります。

2.1.2 操作盤画面の自動表示設定

操作盤画面をコントローラ立ち上げ時に自動的に表示するためには、ティーチングペンダントで以下の設定を行ってください。

ステップ 1

[F6:設定]→[F7:オプション]→[F9:操作盤]を押すと、以下の画面が表示されます。



ステップ 2

操作盤自動起動設定を[有効:1]に設定してください。
さらに起動したい画面のあるフォルダを起動パネルパスに設定してください。
以上を行うことで次回起動時に自動表示を行います。

注意：エラーがペンダントに出力されている場合はこの機能が有効となりませんのでご注意ください。

2.2　操作盤エディタの各部品の使い方

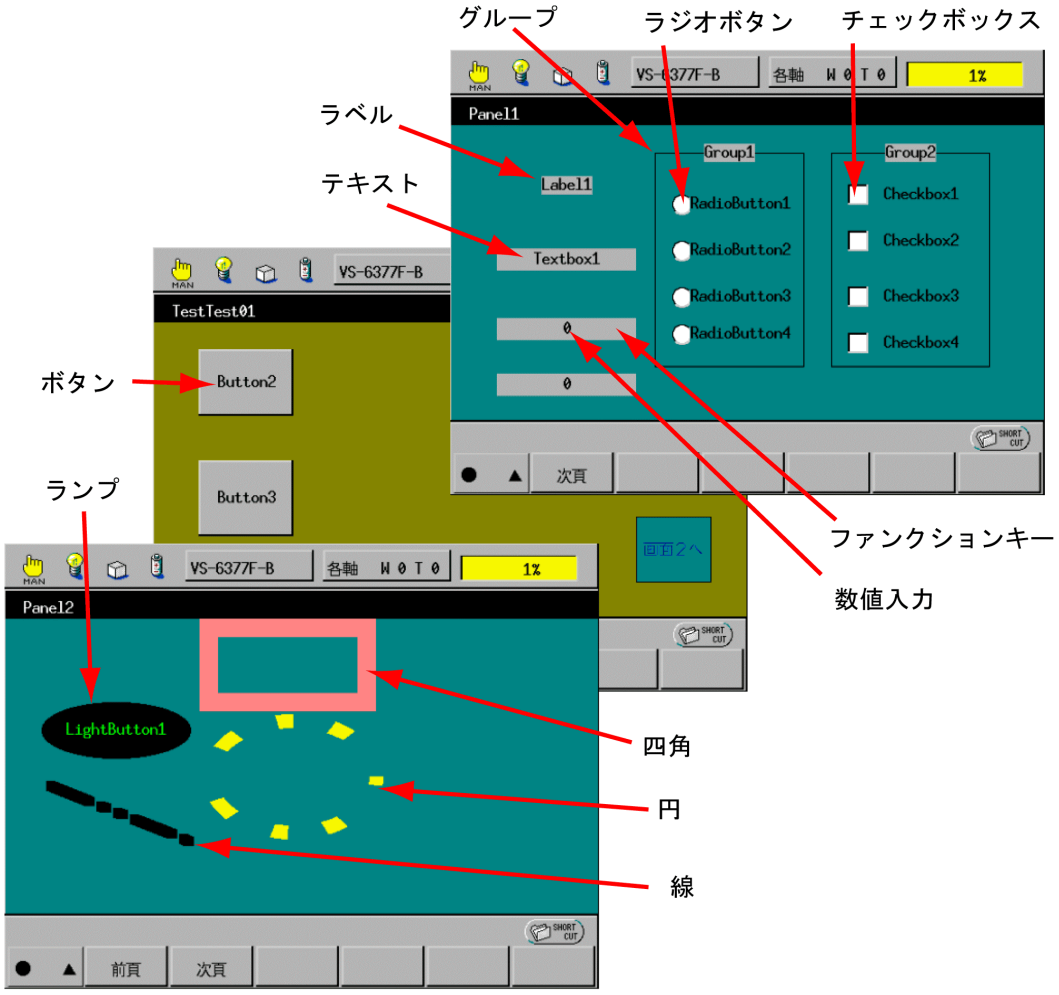
2.2.1　操作盤エディタの部品と機能の一覧

画面に配置できる部品は下表の１４種類です。

操作盤エディタの部品

	部品	機能
1	ボタン	押しボタンとして動作します。
2	ラベル	文字列を表示します。
3	数値入力	テンキーによる数値入力を行います。
4	テキスト	キーボードによる文字列入力を行います。
5	グループ	ラジオボタンの排他を行うグループとして指定します。
6	ラジオボタン	グループによる排他選択を行います。
7	チェックボックス	項目選択を行います。
8	ランプ	ON/OFFするランプとして動作します。
9	線	直線を配置します。
10	四角	正方形，長方形を配置します。
11	円	円，楕円を配置します。
12	ファンクションキー	ファンクションキー(F1～F12)を押しボタンとして使用します。
13	タイマ	一定時間ごとに操作を指定できる部品です。
14	照光式ボタン	動作は押しボタン，表示はランプとして動作するボタンです。

画面例



2.2.2 部品のアクションの指定

操作盤画面の部品のアクション（押されたときなどの動作）を指定します。
操作盤エディタ画面でボタンが押された場合の動作記述，属性変更・取得を行います。

(1)動作記述形式

動作記述は以下の形式で、行わせたい動作のみ記述します。

```
DEF　オブジェクト名__アクション
 行わせたい動作
END
```

操作盤エディタにおいてオブジェクト名とアクションを選択すると，
「DEF　オブジェクト名__アクション」と「END」の部分が自動生成されますので，内部のみ記述を行ってください。
アクションには下表のものがあり、使用可能なアクションは、部品の種類によって異なります。

アクション名	動作内容
CLICKED	部品が押された
RELEASED	部品が離された
TIMER	インターバル時間が経過した
REFRESH	画面リフレッシュが行われた

(2)動作記述内容

行わせたい動作は操作盤制御コマンドとそれぞれのオブジェクトの属性変更・取得により記述されます。オブジェクトの状態は「オブジェクト．属性」の形式で変更・取得ができます。

変更可能な属性については、「3.3.4　操作盤部品オブジェクト変数」を参照してください。
使用できる変数はI，F，D，S型グローバル変数とローカル変数，フォルダ変数です。

2.2.3 各部品の使い方

(1) ボタン

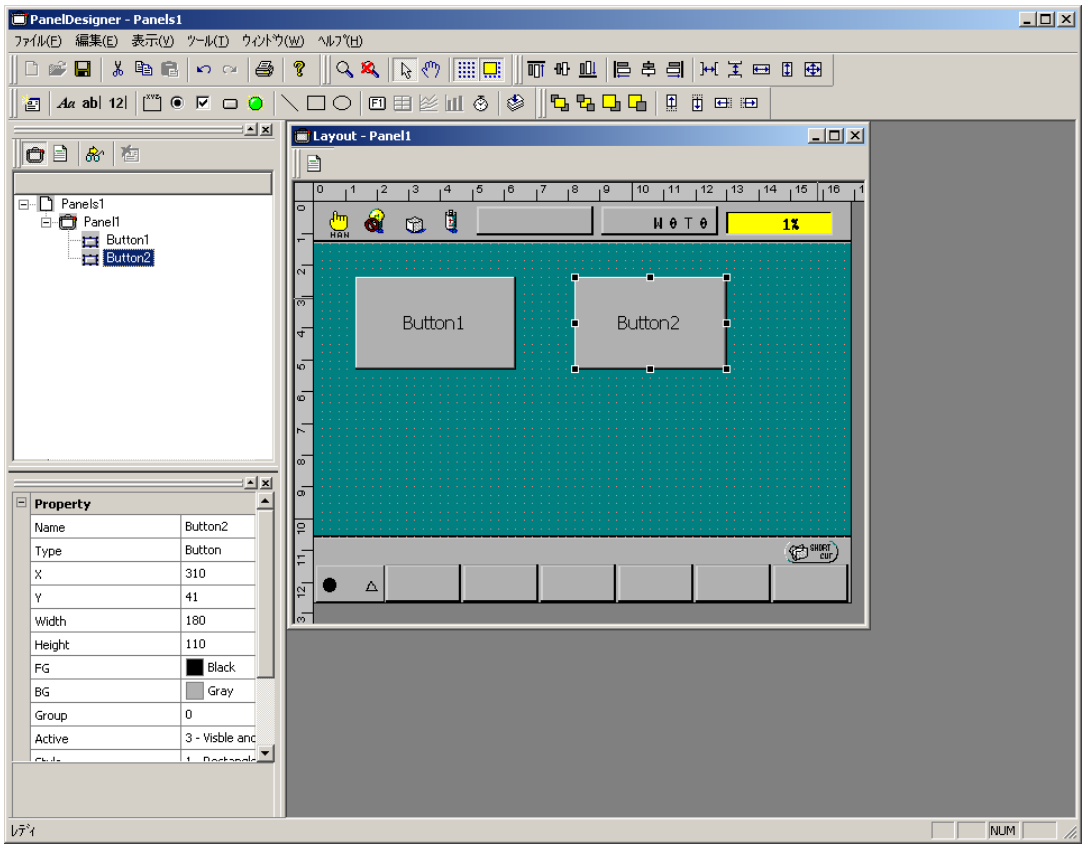
ボタンは押す/離すというアクションに対して動作を行う部品です。
ここでは押されたときにIO24番をONし，離されたときにOFFするボタンと，押されたときにプログラム（Sample pro）を起動するボタンを作成する方法を説明します。

■ボタン作成の例

STEP 1

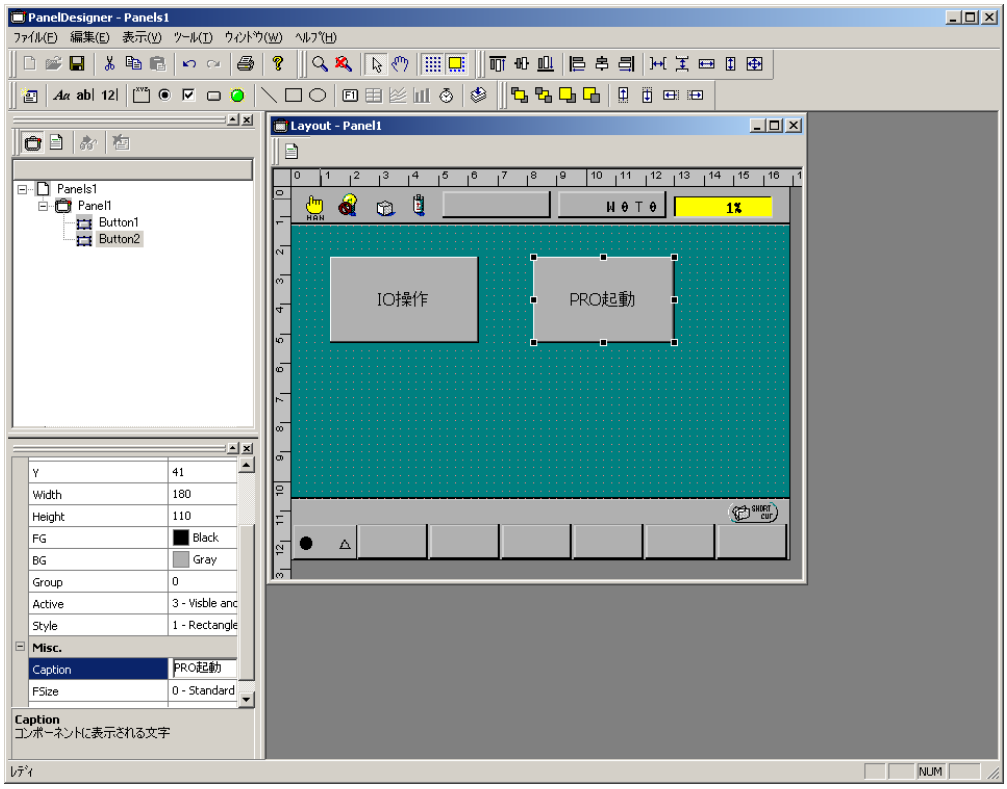
操作盤エディタでボタンを2個配置します。

配置の場所はペンダント画面上であれはどこでも自由におくことができます。ボタン上の表示文字はそれぞれ"IO操作"，"PRO起動"とします。
部品はボタンに限らず全部品とも，パラメータ変更・状態取得を自部品，同一パネル上の他部品から行うため部品名により識別されます。
ボタンの場合"Button"+"数字"がデフォルト値として与えられますが，自由に変更することが可能です。
ここではデフォルト（"Button1","Button2"）のまま作業を進めます。



STEP 2

ボタン上の表示文字はそれぞれのボタンのパラメータ”Caption”で変更してください。

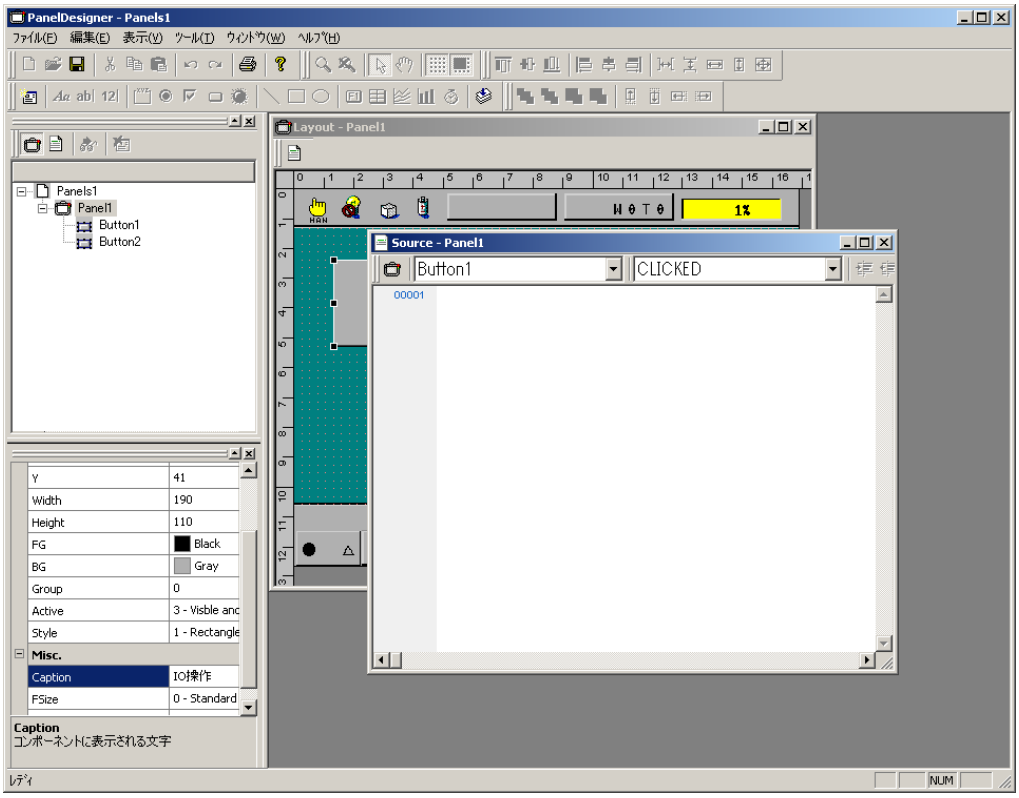


STEP 3

＜ボタンアクション記述＞

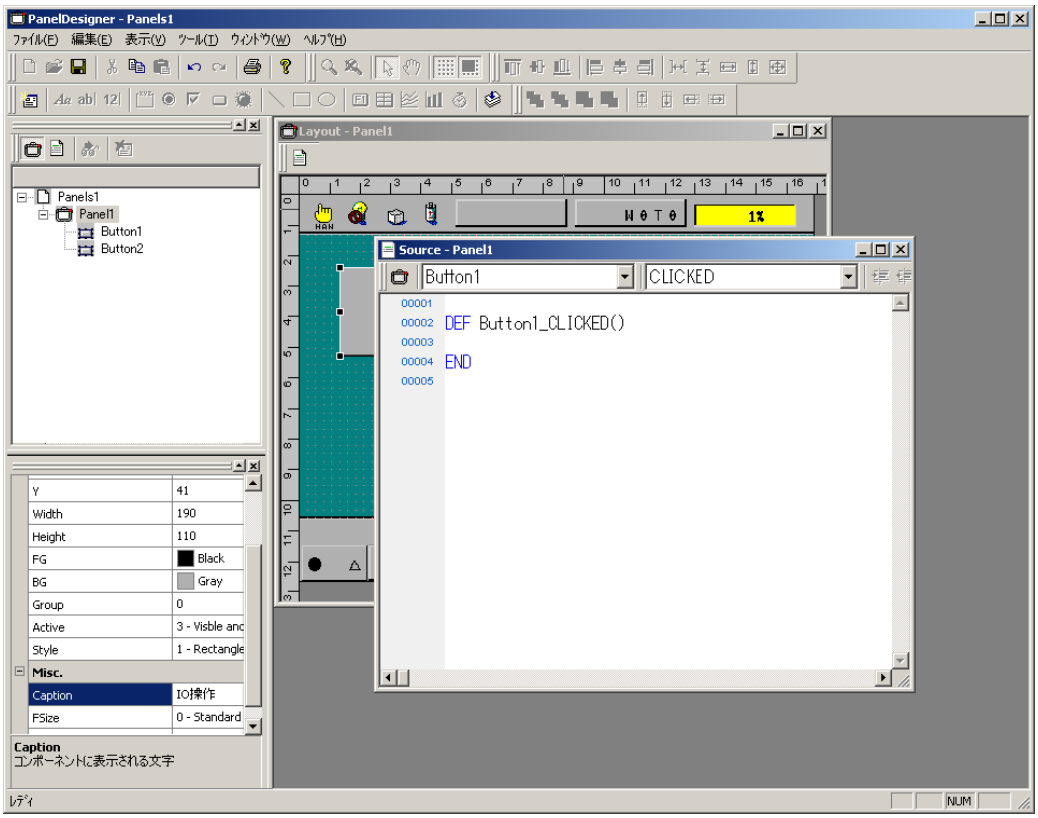
ボタンにはそれぞれ押されたとき(Clicked)/離されたとき(Released)の動作を自由に指定することができます。以下にボタンが押されたとき，離されたときの動作を記述する例を示します。

操作盤エディタで”IO 操作”ボタンをダブルクリックします。するとソース編集画面が開かれます。



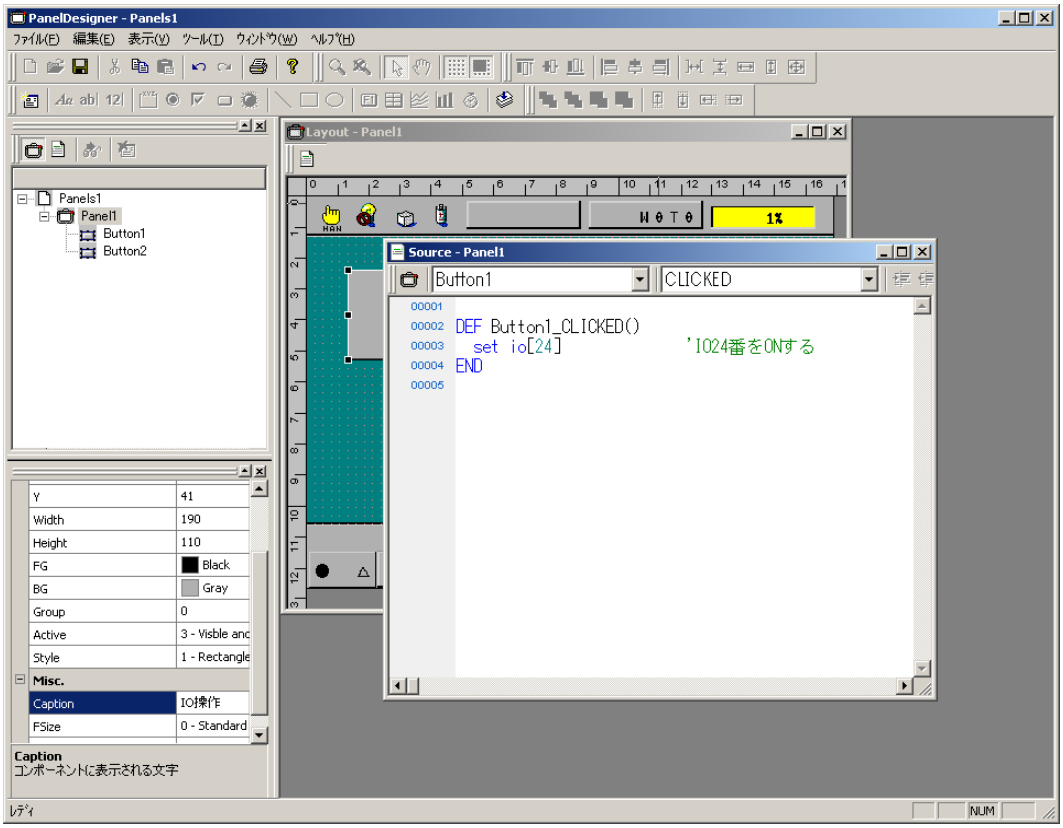
STEP 4

まず押されたときにIO24番をONするプログラムを記述します。ソース編集画面上部のコンボボックスから部品名"Button1"，アクション"Clicked"（押されたとき）を選択すると，自動的にプログラムの外側が生成されます。



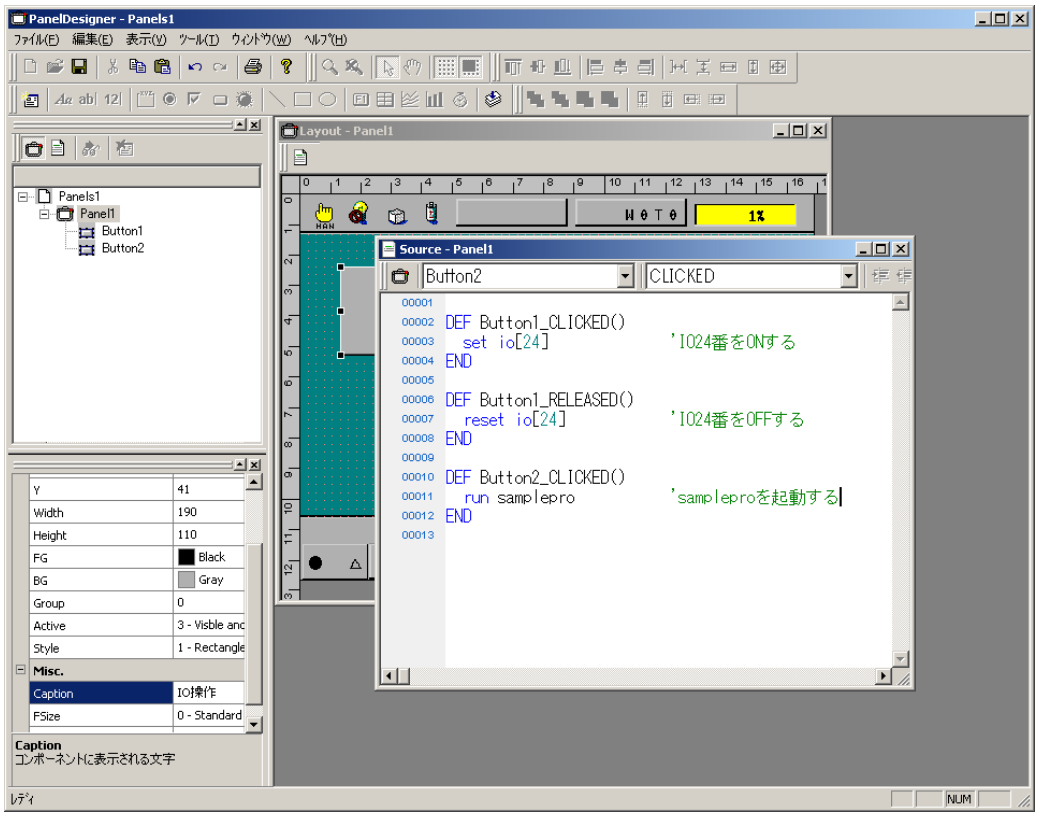
STEP 5

その中に動作を記述します。



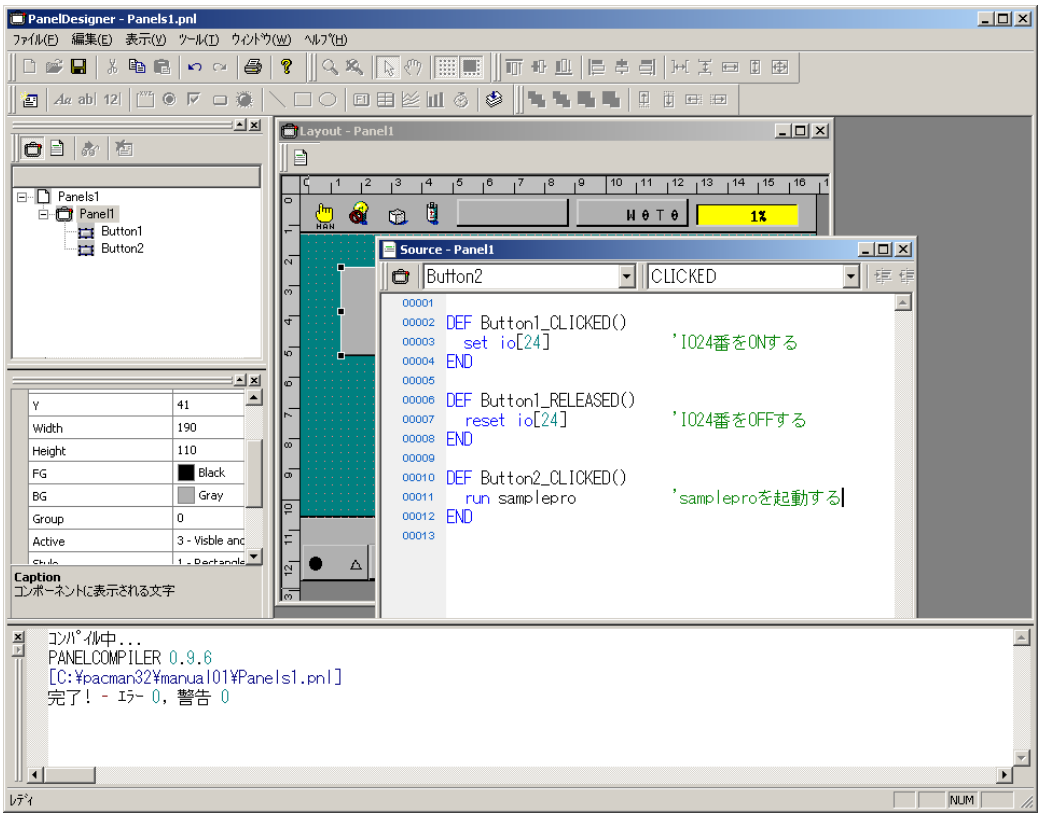
STEP 6

同様に離されたときにIO24番をOFFするプログラム（部品名"Button1"，アクション"Released"：離されたとき），押されたときにプログラム（samplepro）を起動するプログラム（部品名"Button2"，アクション" Clicked"）を記述します。



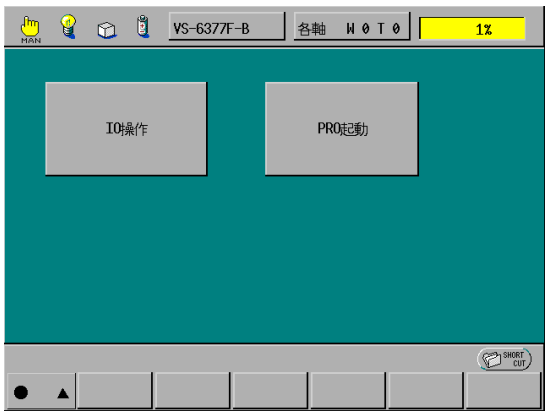
STEP 7

終了したら，プログラムをセーブして，コンパイルを実施し文法チェックを行ってください。



STEP 8

コンパイルが成功したら，PACManager でコントローラへの転送を行ってください。



STEP 9

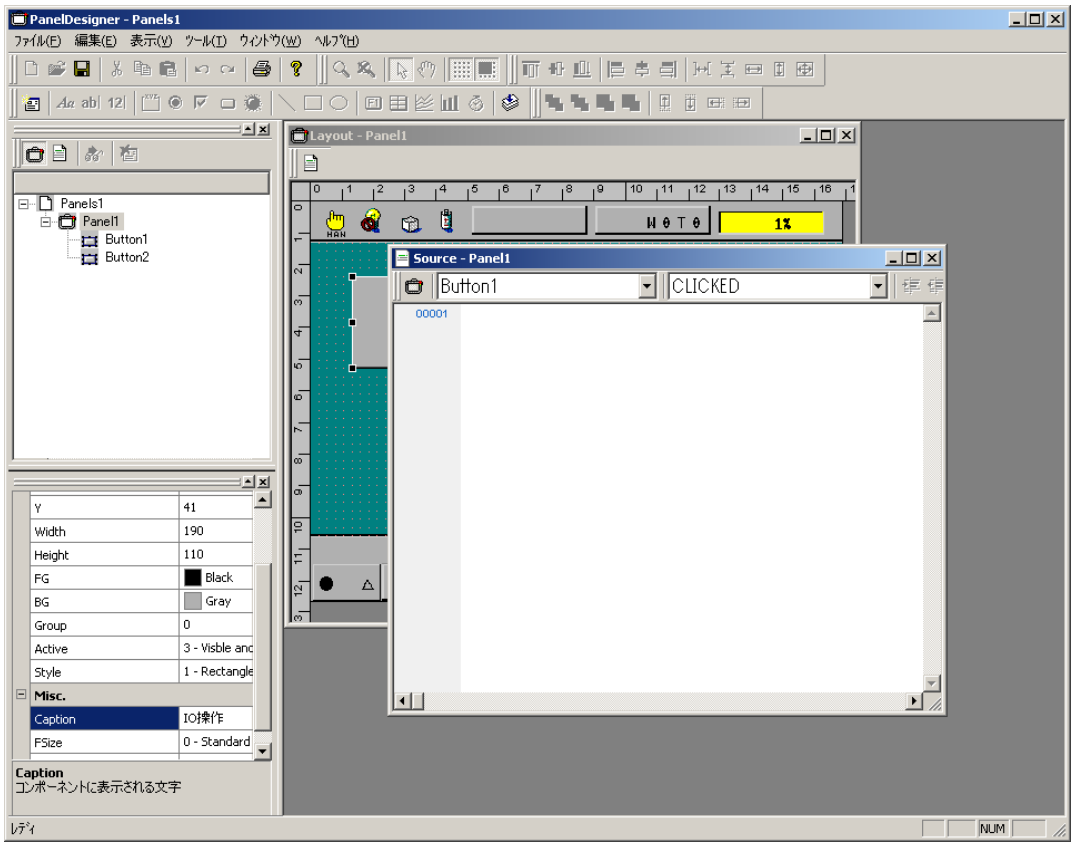
＜ボタンパラメータ変更＞

色，位置などのボタンパラメータは自部品（ボタン），同一パネル内の他部品から部品名を用いて変更/取得が行えます。

ボタンパラメータは”部品名．パラメータ”の形式でアクセス可能となっています。各部品のパラメータ，とりうる値の詳細は「3.3.4　操作盤部品オブジェクト変数」を参考にしてください。

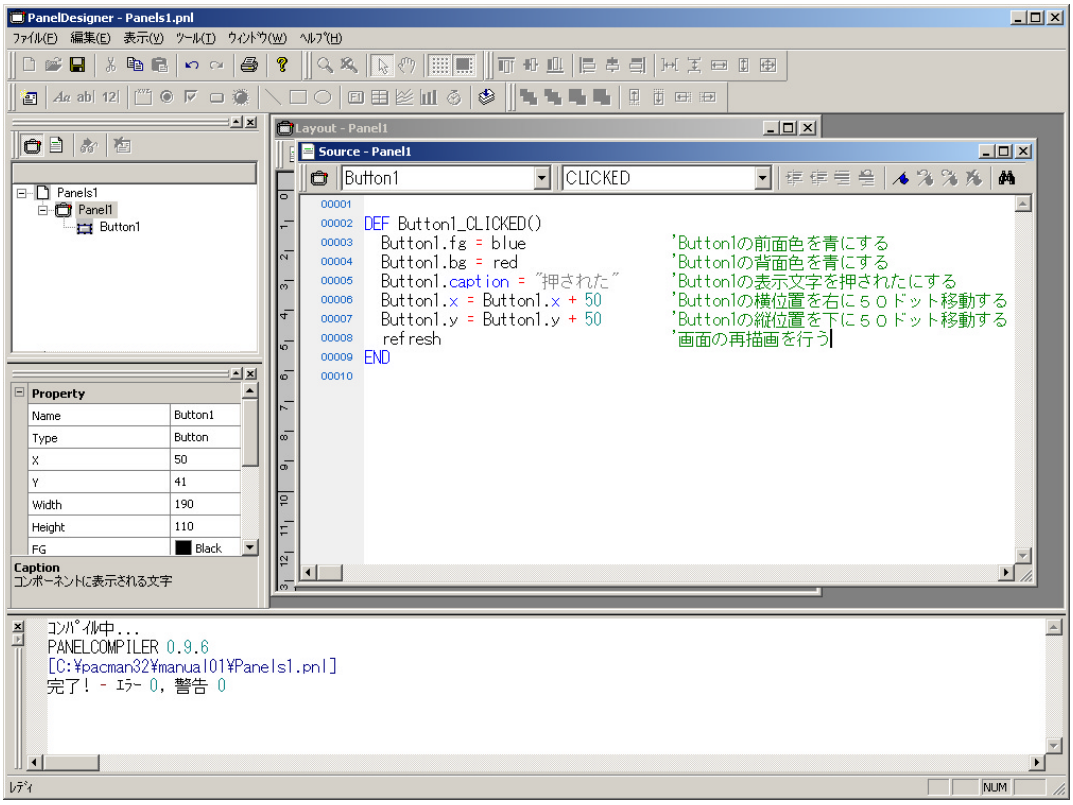
ここでは部品の色（Fg:前面色,Bg:背面色），表示文字（Caption），位置（X:横位置,Y:縦位置）を変更する例を示します。

先ほどのように操作盤エディタを起動し，ボタンの配置とソース編集画面の立上げを行ってください。



STEP 10

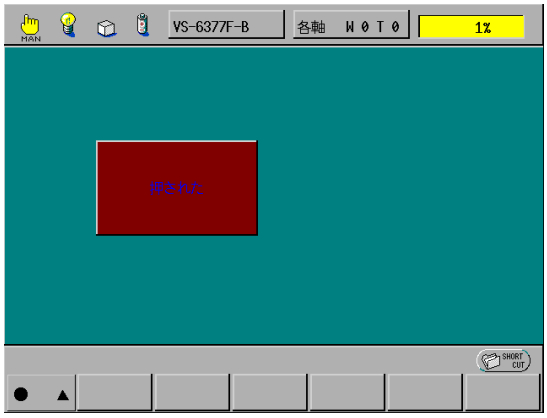
プログラムは次のように記述します。



STEP 11

先ほどと同様にセーブ，コンパイル，コントローラへの転送を行ってください。

ボタン押した後のペンダント図

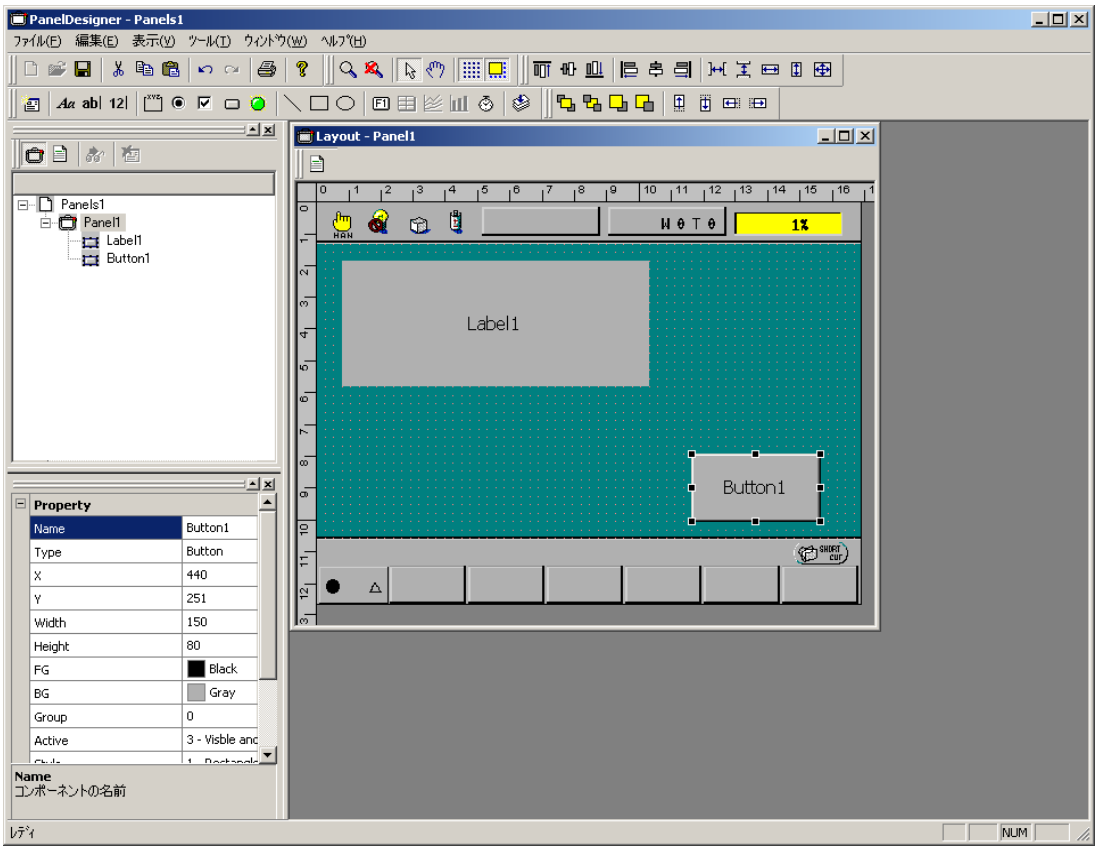


(2) ラベル

ラベルは文字列を表示する部品で，イベントは発生しないため動作記述は行えません。ここではラベルを作成し，同一画面中のボタンが押されたときにそのラベルのパラメータを変更する例を示します。

■ラベル作成の例

STEP 1 ボタンと同様に操作盤エディタを起動し，画面にラベル１個とボタン１個を配置します。



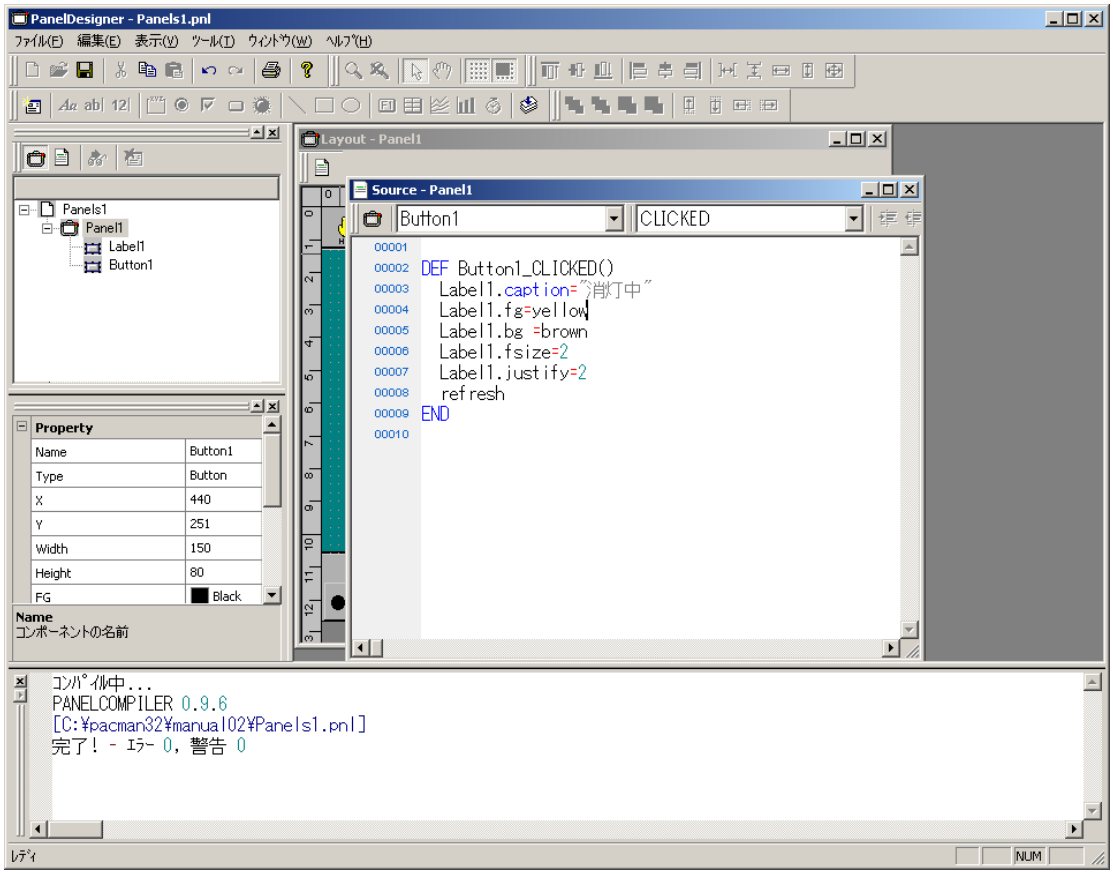
STEP 2　＜ラベルパラメータ変更＞

ラベルそれぞれの表示文字列，色，フォントサイズ，文字位置を変更します。パラメータへのアクセスはボタンと同様に部品名．パラメータで行います。

部品名Label1の表示文字列を”消灯中”に変更するには「Label1.caption=”消灯中”」と記述します。同様にラベルの前面色を黄色，背面色を茶色，フォントサイズを大，文字位置を左詰にするには以下のように記述を行います。

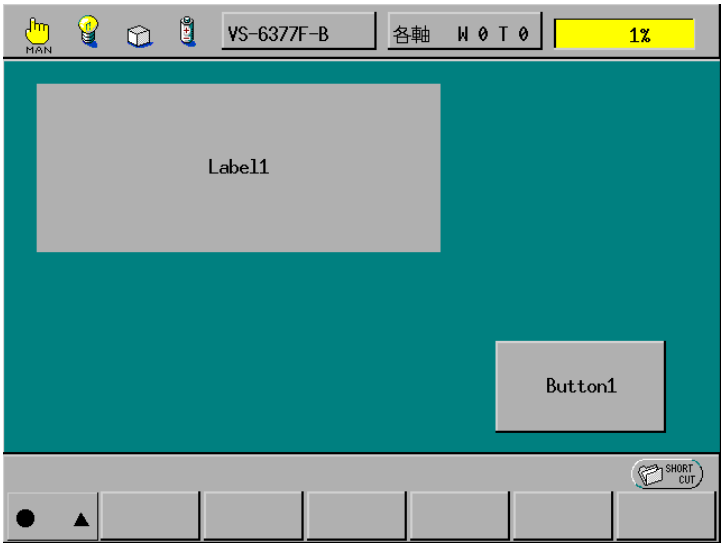
```
Label1.fg=yellow        ：前面色を黄色
Label1.bg =brown        ：背面色を茶色
Label1.fsize=2          ：フォントサイズを大
Label1.justify=2        ：文字位置を左詰
```

以上をソース編集画面でButton1のClickedで作成されたプログラムの中に記述します。



STEP 3 これをコントローラに送りボタンを押すと以下のように表示されます。

ボタンを押す前



ボタンを押した後

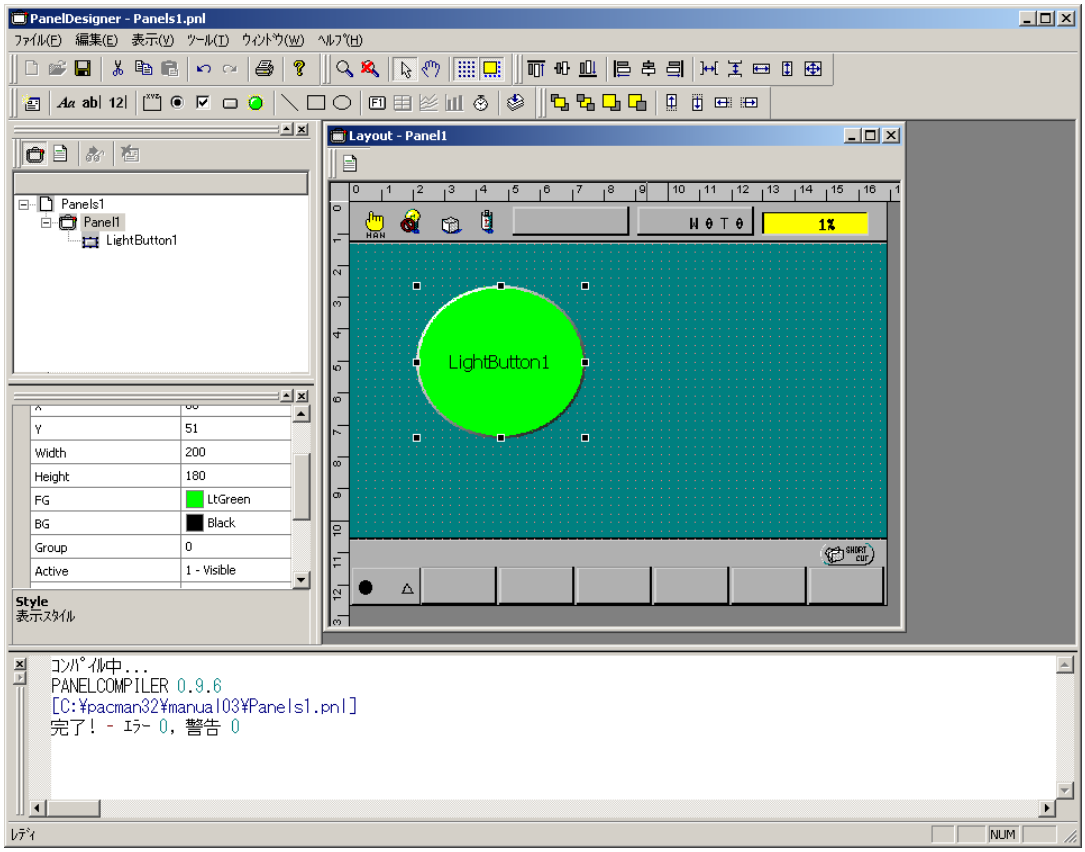
(3) ランプ

ランプはON/OFF状態を表示する部品で，定期的にアクション(refresh)を発生します。このアクションを用いて状態監視，表示を行います。
ここではI/O状態を監視しその状態を表示する例を示します。

■ランプ作成の例

STEP 1

ボタンと同様に操作盤エディタを起動し，画面にランプ１個を配置します。

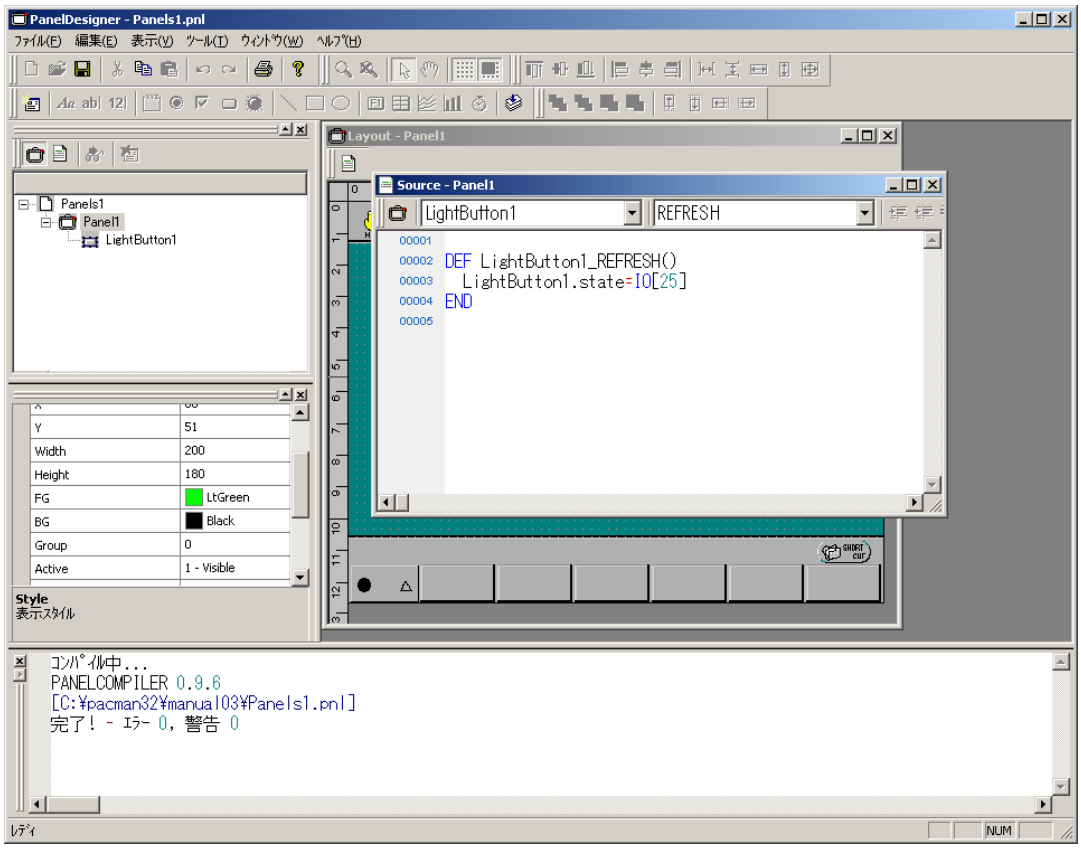


STEP 2　＜ランプアクション記述＞

ランプは定期的動作を行うrefreshアクションを持っています。このアクションを使いI/O 25番の状態をランプ自身の状態として表示するためには，ソース編集画面でランプのrefreshアクションを選択して作成されたプログラム内に以下のように記述します。

```
LightButton1.state=IO[25]
```

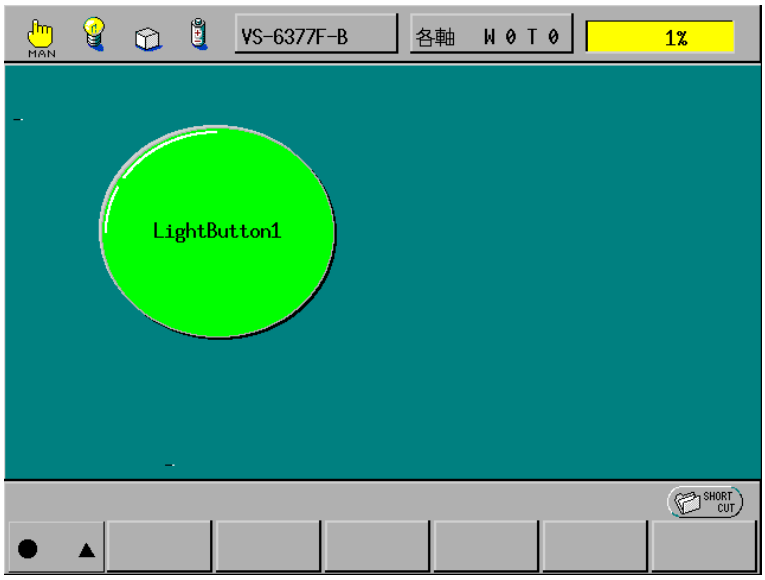
これはランプの状態をIO[25]の状態と同じにするという意味です。



STEP 3

これをコントローラに送信すると以下のように表示されます。

点灯時



消灯時

STEP 4

＜ランプパラメータ変更＞

ランプのパラメータもボタンと同様にアクセスが可能です。

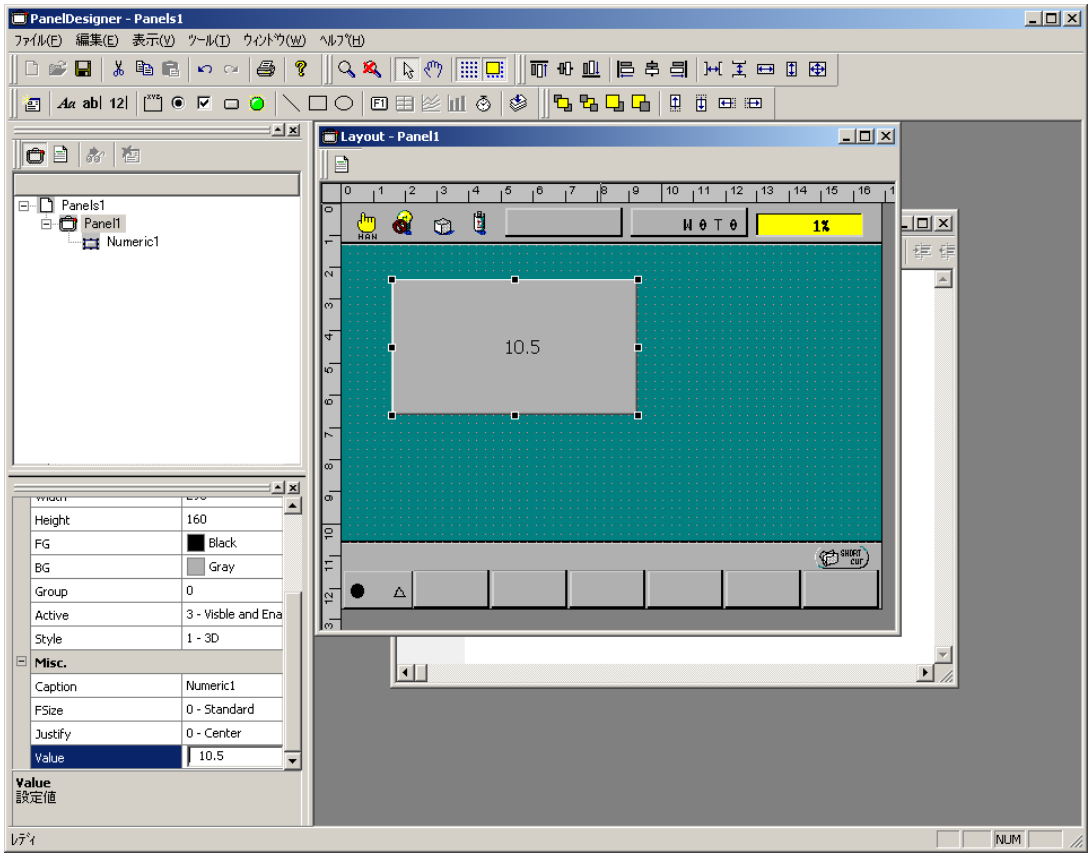
(4) 数値入力ボックス

数値入力ボックスはボタン自身に実数値を持ち，その値を表示する部品です。ボタンは押されることで数値入力テンキーを表示しペンダント操作画面上からも数値を直接入力することができます。またボタンと同様に押されたアクション，離されたアクションそれぞれに対して動作を記述することができます。

■数値入力ボックス作成の例

STEP 1

ボタンと同様に操作盤エディタを起動し，画面に数値入力ボックス１個を配置します。次に数値入力ボックスの持つ値の初期値を設定します（省略可）。



STEP 2

＜数値入力ボックスアクション記述＞

数値入力ボックスの持つアクションはボタンと同様に記述することができます。

STEP 3　＜数値入力ボックスパラメータ変更＞

数値入力ボックスはボタンと同様に色，位置等を持ちますが，この部品独特のパラメータとして数値（value：実数値），スタイル（style：表示形式10進，16進など）を持ちます。

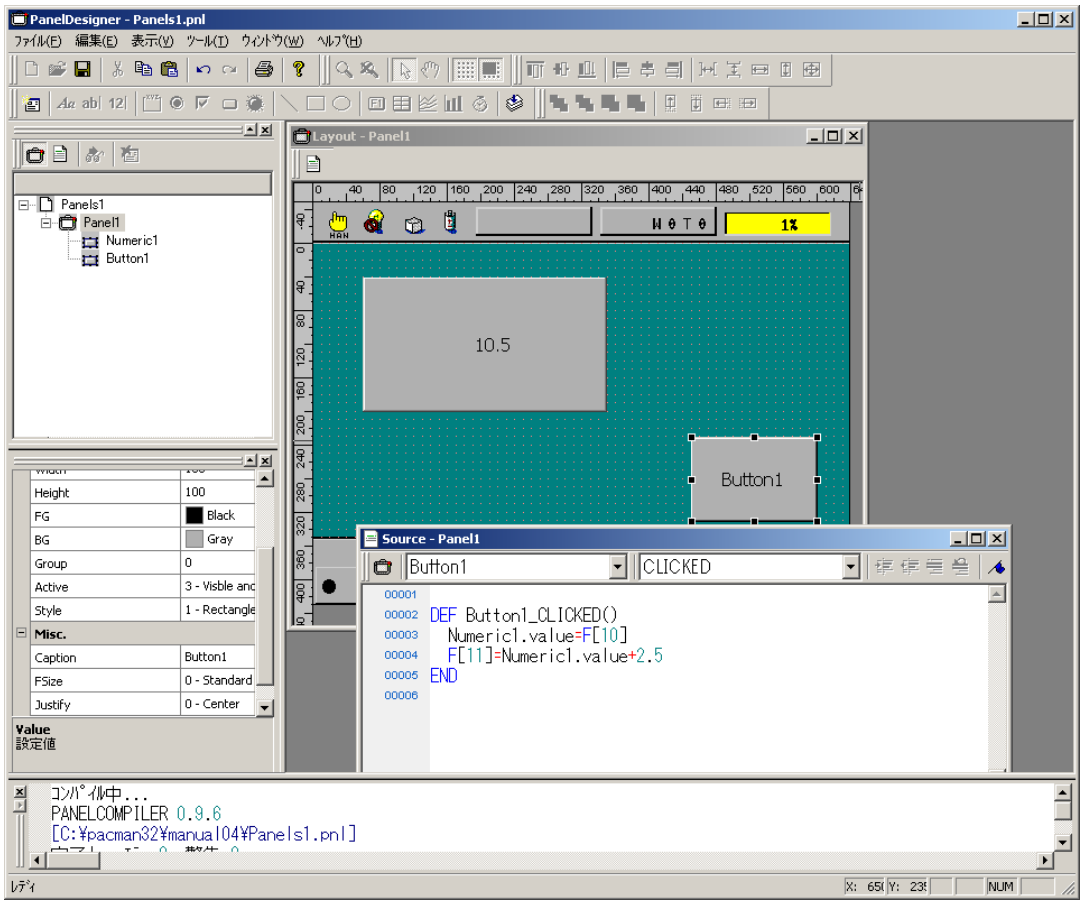
ここでは,同一画面上のボタンが押されたらF型グローバル変数10番の値を数値入力ボックスの数値へ取り込み,その値に2.5足してF型グローバル変数11番に格納する例を示します。

操作盤エディタで数値入力ボックスとボタンを作成し，ソース編集画面を開きボタンが押されたときの処理内容を記述します．パラメータへのアクセスは他部品と同様です。

Button1,Clickedを選択しプログラムの外側を作成し以下のように記述します。

```
Numeric1.value=F[10]

F[11]=Numeric1.value+2.5
```



(5) テキストボックス

テキストボックスはボタン自身に文字列を持ち，その値を表示する部品です。ボタンは押されることでキーボードを表示しペンダント操作画面上からも文字列を直接入力することができます。またボタンと同様に押されたアクション，離されたアクションそれぞれに対して動作を記述することができます。

■テキストボックス作成の例

STEP 1

ボタンと同様に操作盤エディタを起動し，画面にテキストボックスを配置します．次にテキストボックスの持つ文字列の初期値を設定します（省略可）。

STEP 2

＜テキストボックスアクション記述＞

テキストボックスの持つアクションはボタンと同様にClicked，Releasedを持ちますのでそれぞれについて処理を記述することができます。

STEP 3

＜テキストボックスパラメータ変更＞

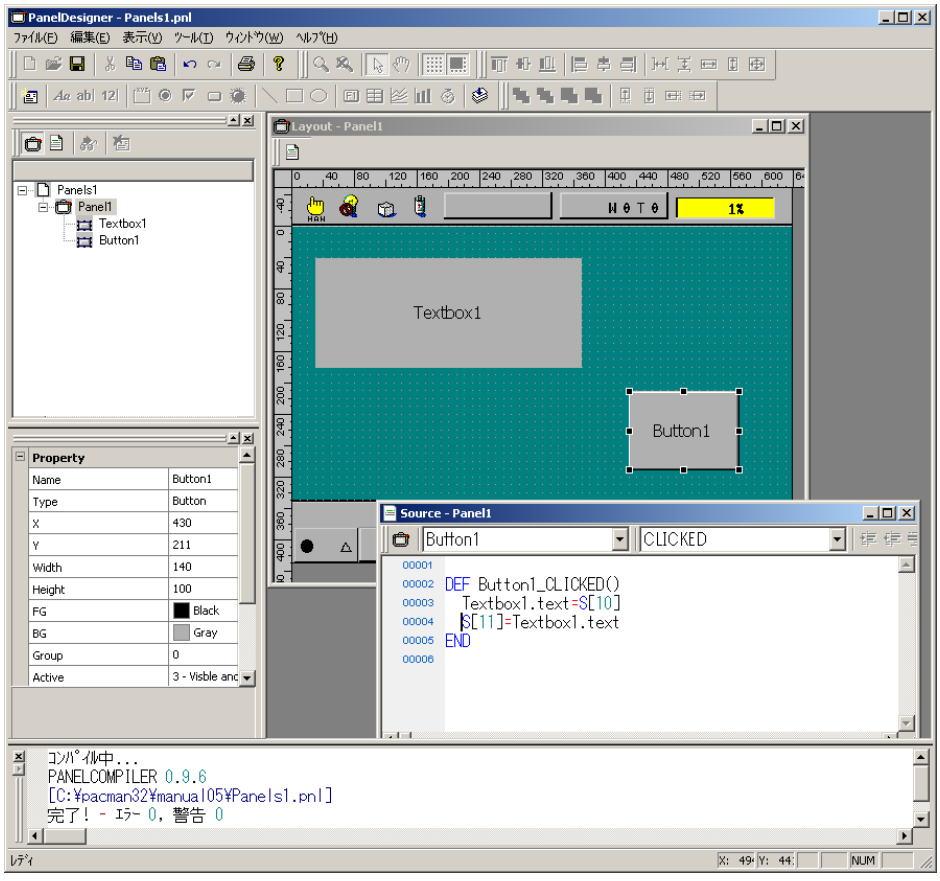
テキストボックスはボタンと同様に色，位置等を持ちますが，この部品独特のパラメータとして文字列（text：文字列値）を持ちます。ここでは，同一画面上のボタンが押されたらS型グローバル変数10番の値をテキストボックスへ取り込み，その値を再びS型グローバル変数11番に格納する例を示します。

操作盤エディタでテキストボックスとボタンを作成し，ソース編集画面を開きボタンが押されたときの処理内容を記述します．パラメータへのアクセスは他部品と同様です。

Button1，Clickedを選択しプログラムの外側を作成し以下のように記述します。

```
Textbox1.text=S[10]

S[11]=Textbox1.text
```



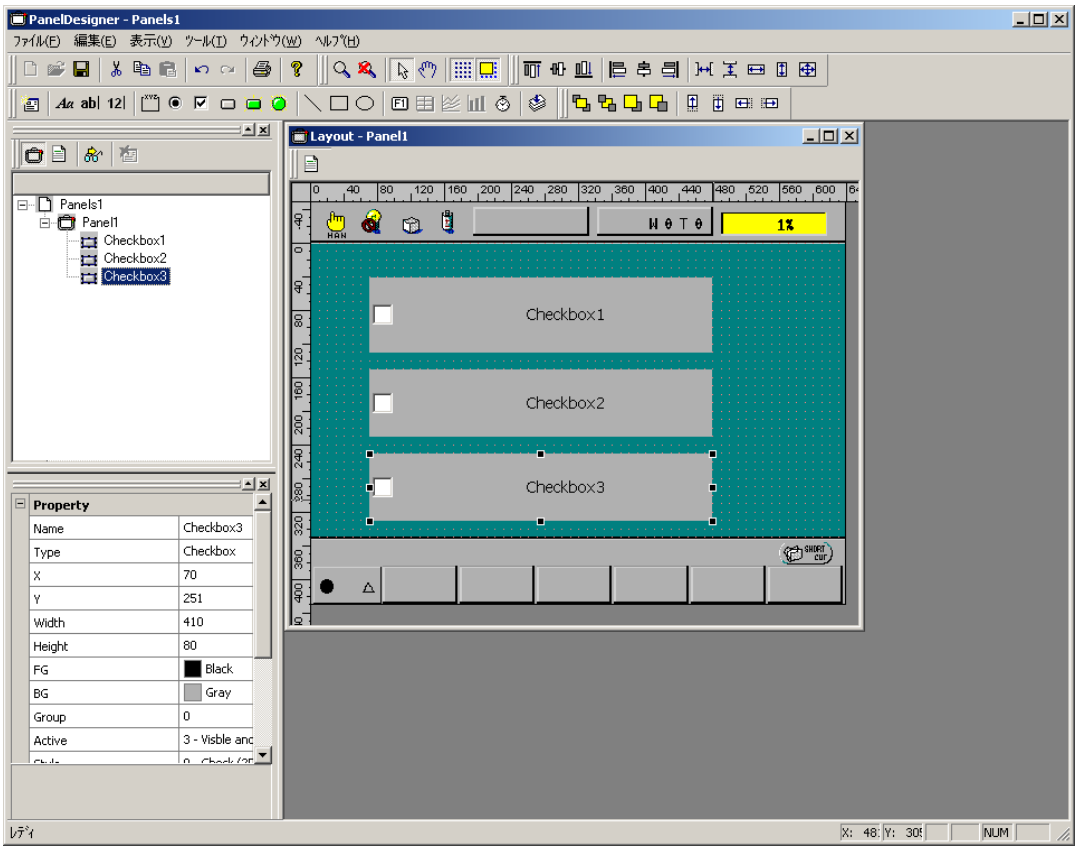
(6) チェックボックス

チェックボックスは押されることでON/OFF状態を切り替える部品です。トグルボタンとしても利用できます。またボタンと同様に押されたアクション，離されたアクションそれぞれに対して動作を記述することができます。ON/OFF状態はパラメータ(state)でアクセスすることが可能です。

■チェックボックス作成の例

STEP 1

ボタンと同様に操作盤エディタを起動し，画面にチェックボックスを配置します。



STEP 2

＜チェックボックスアクション記述＞

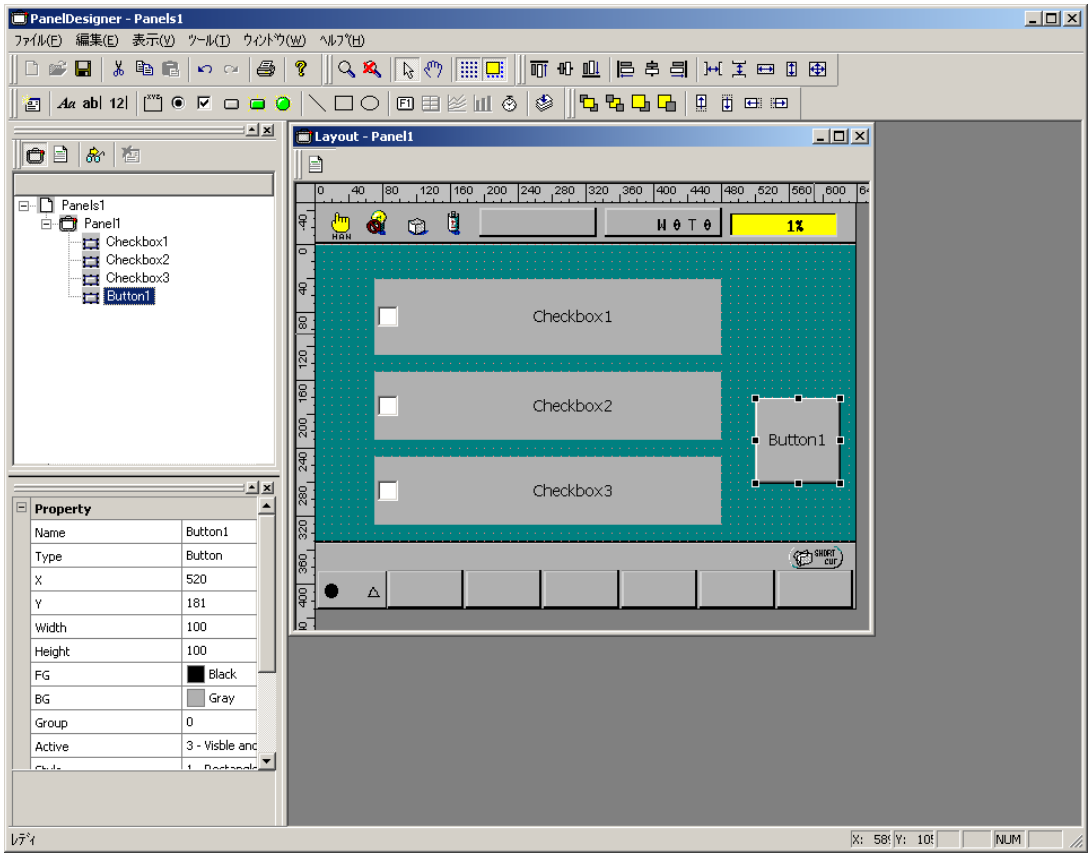
チェックボックスの持つアクションはボタンと同様にClicked,Releasedを持ちますのでそれぞれについて処理を記述することができます。

STEP 3　＜チェックボックスパラメータ取得・変更＞

チェックボックスはボタン，ラベルと同様のパラメータを持ちます。

ここでは同一画面上のボタンが押されたときに，チェックボックスの状態（ON/OFF）をIO[24]～IO[26]に出力する例を記述します。

まずそれぞれの部品を配置します。

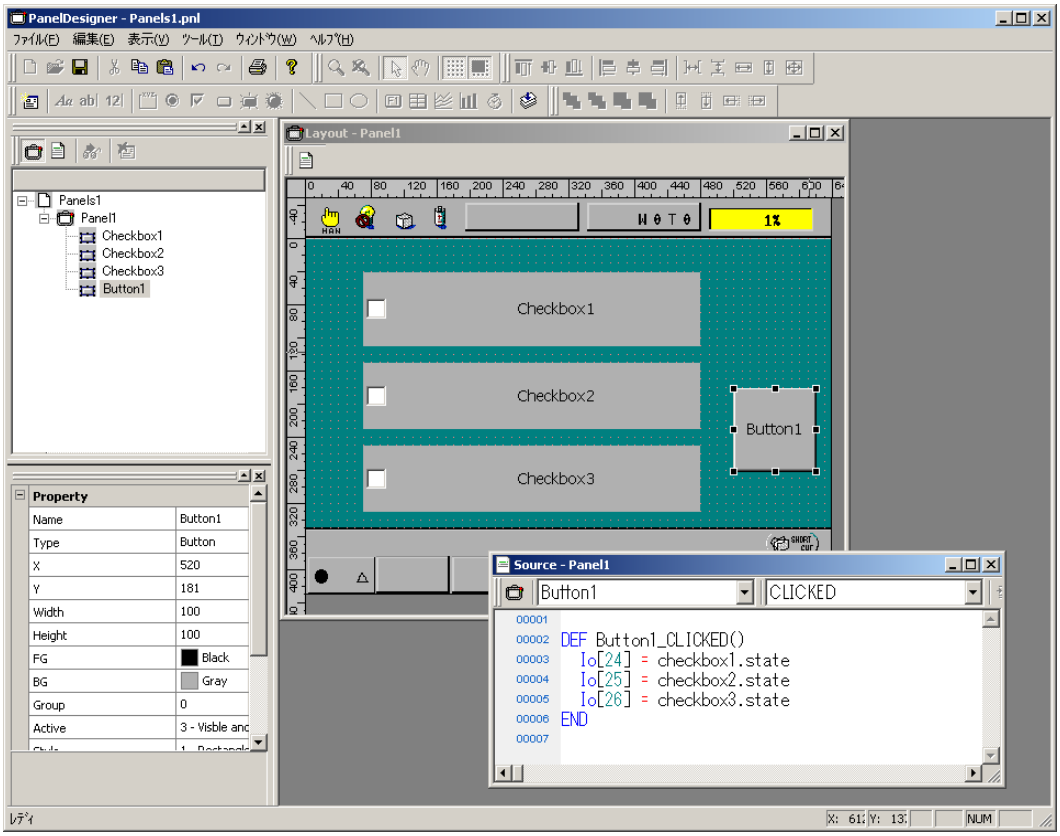


STEP 4

次に，ボタンが押されたときの処理（Button1,Clickedを選択）を記述します。
チェックボックスの状態をstateパラメータで取得する処理は以下のとおりです。

```
IO[24] = checkbox1.state
IO[25] = checkbox2.state
IO[26] = checkbox3.state
```

これをコントローラに転送することで上記の機能を実現できます。



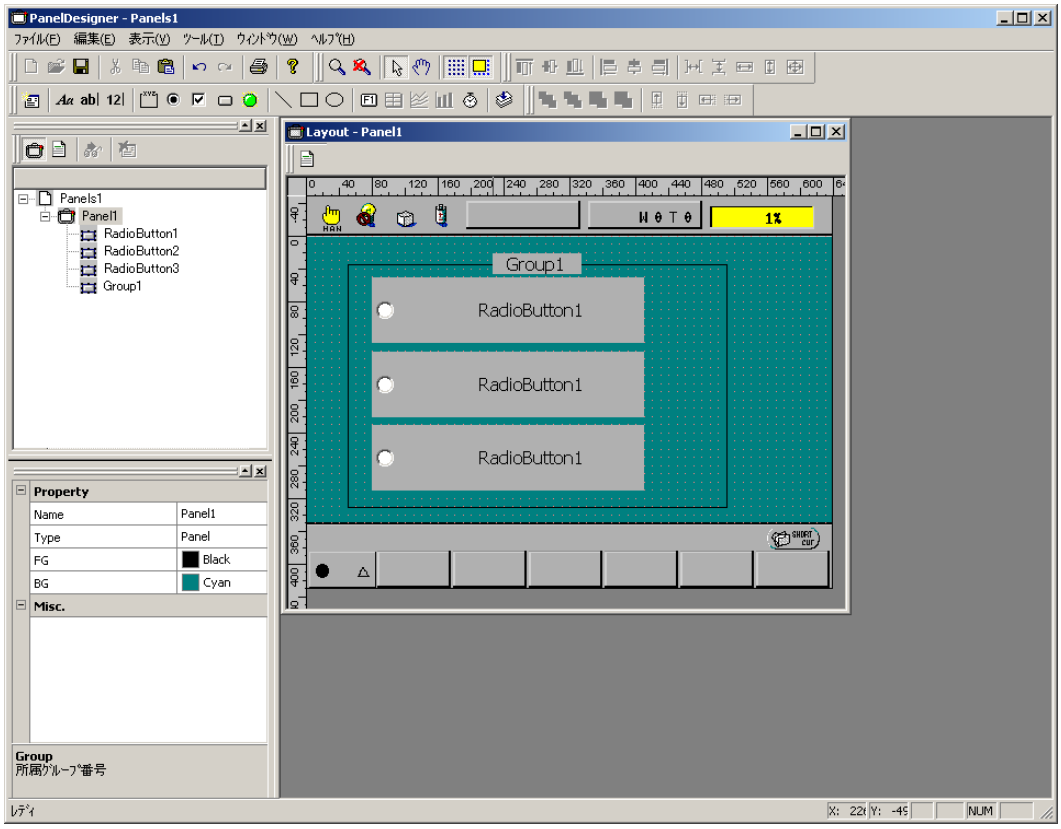
(7) ラジオボタン

ラジオボタンは後述の部品グループでグループ化することで同一グループ内のラジオボタン間での排他を実現する部品です。ボタンと同様に押されたアクション，離されたアクションそれぞれに対して動作を記述することができます。ランプ，チェックボックスと同様にON/OFF状態はパラメータ(state)でアクセスすることが可能です。

■ラジオボタン作成の例

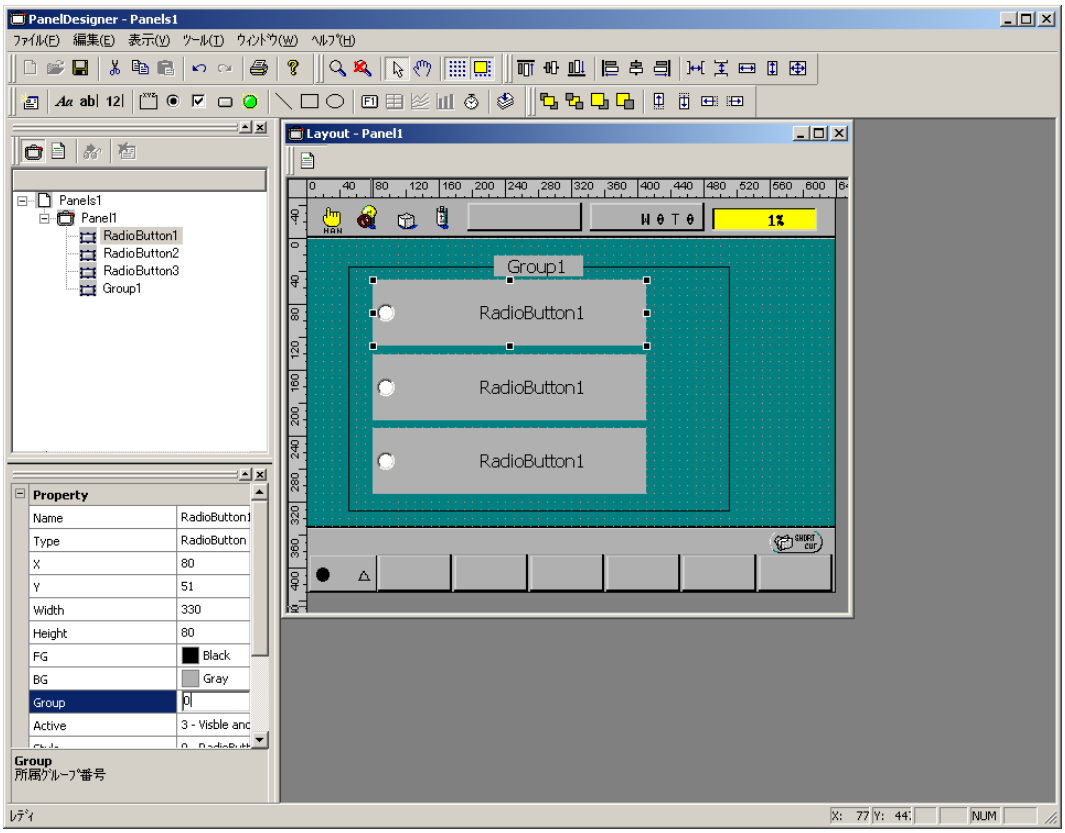
STEP 1

ラジオボタン3個で排他処理を行う例を示します。
操作盤エディタでラジオボタン3個とグループ1個を配置します。



STEP 2

すべてのラジオボタンのパラメータ(group)を配置した部品グループのグループ番号(group)に設定してください。(今回は0に設定)
これでラジオボタンの排他が実現できます。



STEP 3

＜ラジオボタンアクション記述＞

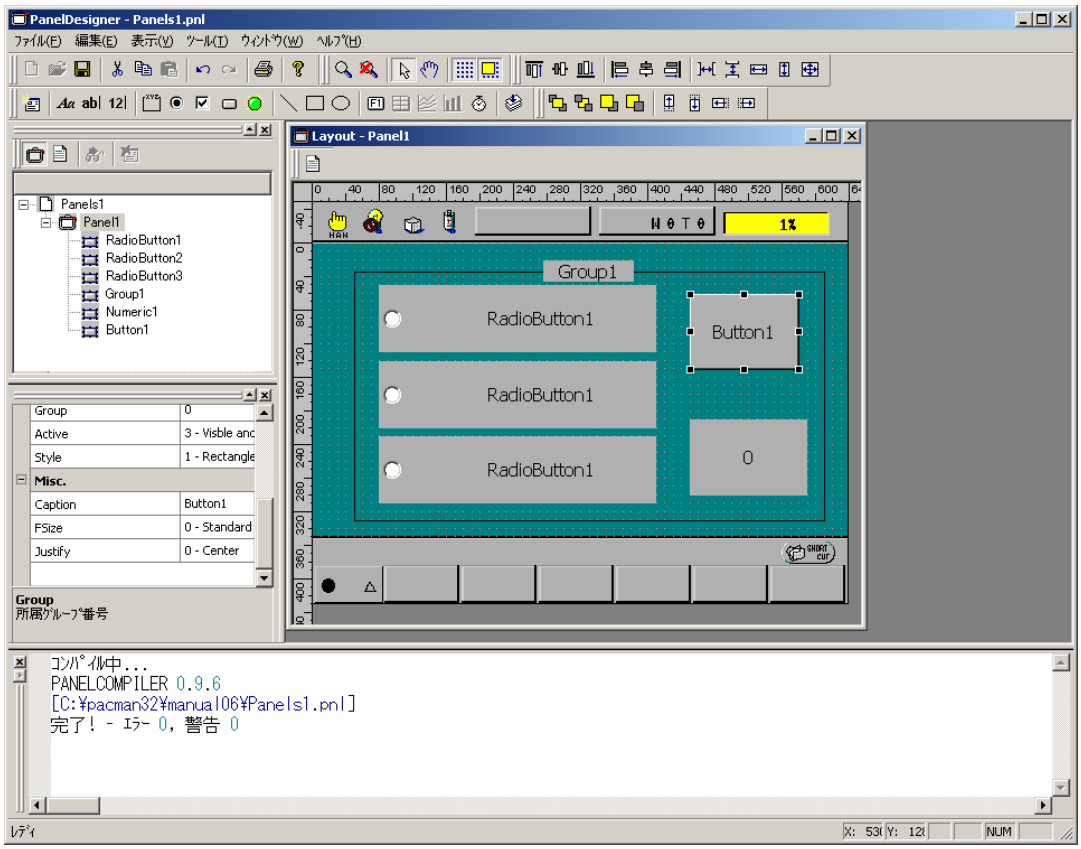
ラジオボタンの持つアクションはボタンと同様にClicked, Releasedを持ちますのでそれぞれについて処理を記述することができます。

STEP 4　＜ラジオボタンパラメータ変更＞

ラジオボタンはボタン，ラベルと同様のパラメータを持ちます。

ここでは同一画面上のボタンが押されたときに，ラジオボタンの状態（ON/OFF）をIO[24]～IO[26]に出力し，RadioButton1が選択されていたらF[10]の値を，RadioButton2が選択されていたらF[11]の値を，RadioButton3が選択されていたらF[12]の値を同一画面上の数値入力ボックスに格納する例を記述します。

まずそれぞれの部品を配置します。

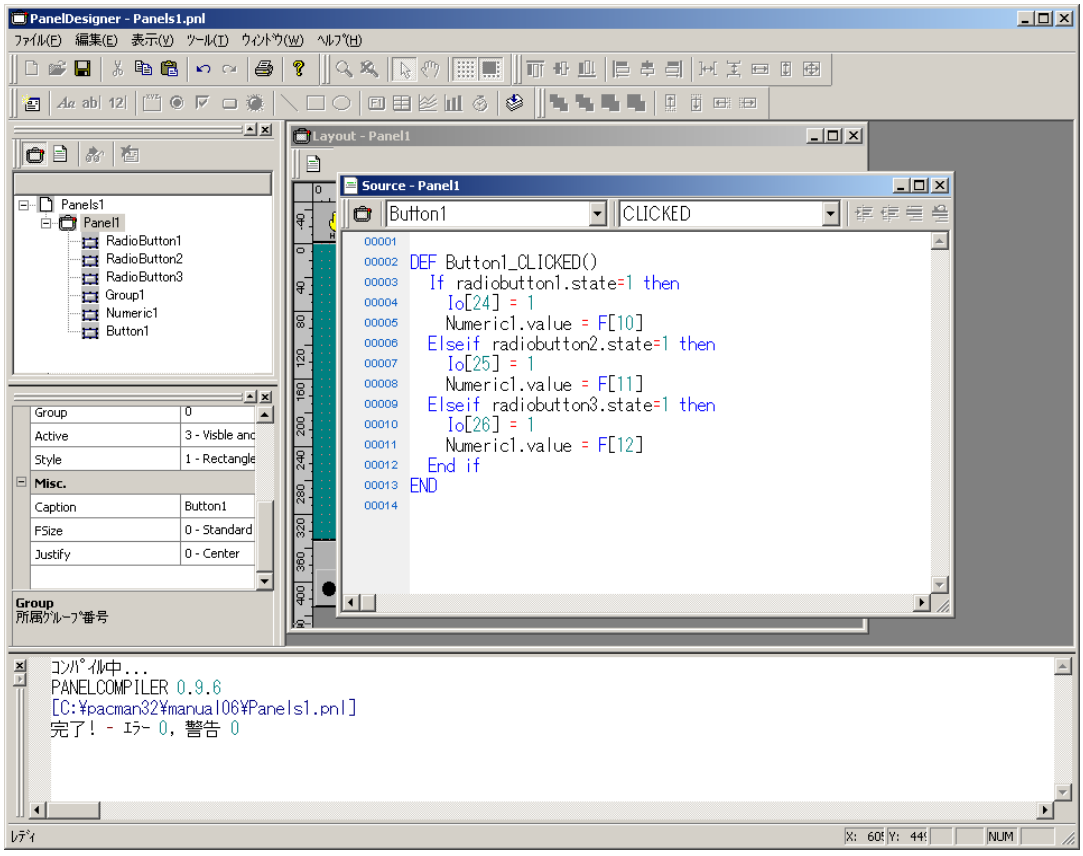


STEP 5

次に，ボタンが押されたときの処理（Button1,Clickedを選択）を記述します。

ラジオボタンの状態はstateパラメータで取得し，IF文で分岐をする処理は以下のとおりです。

```
If radiobutton1.state=1 then
  Io[24] = 1
  Numeric1.value = F[10]
Elseif radiobutton2.state=1 then
  Io[25] = 1
  Numeric2.value = F[11]
Elseif radiobutton3.state=1 then
  Io[26] = 1
  Numeric1.value = F[12]
End if
```



これをコントローラに転送することで上記の機能を実現できます。

(8) グループ

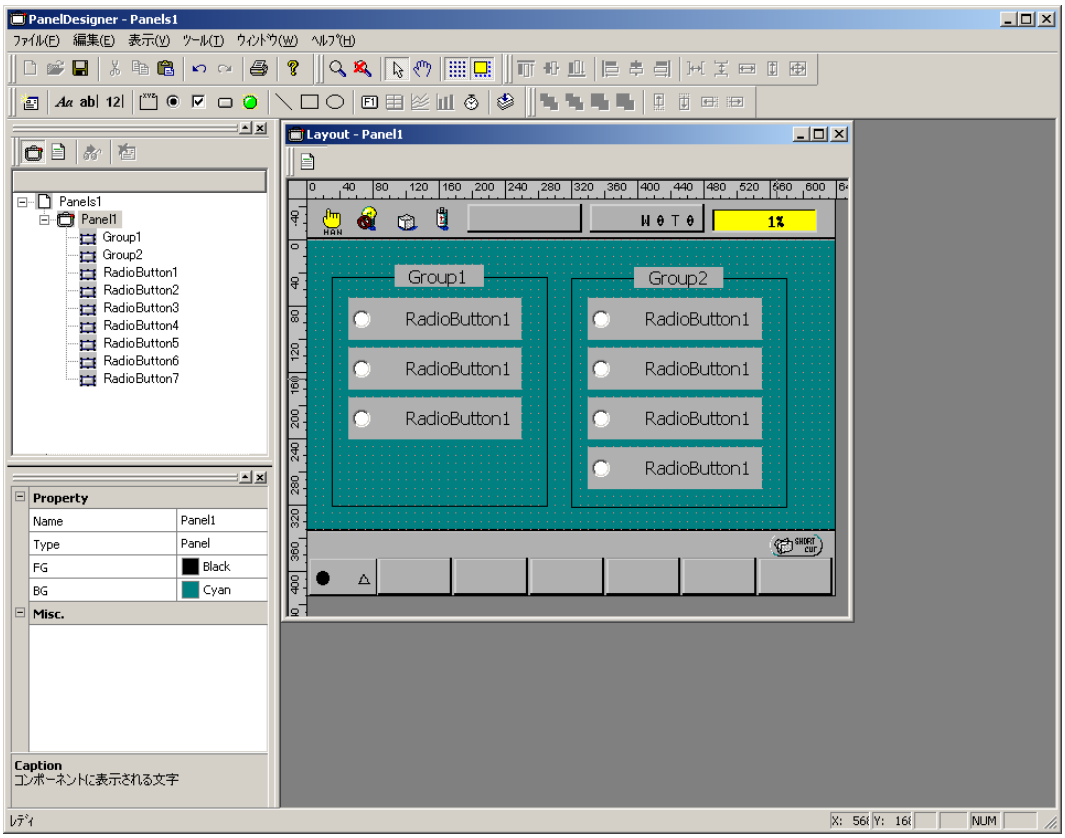
グループはラジオボタンの排他を実現するために，ラジオボタンのグループ化を行う部品です。

■グループ作成の例

STEP 1

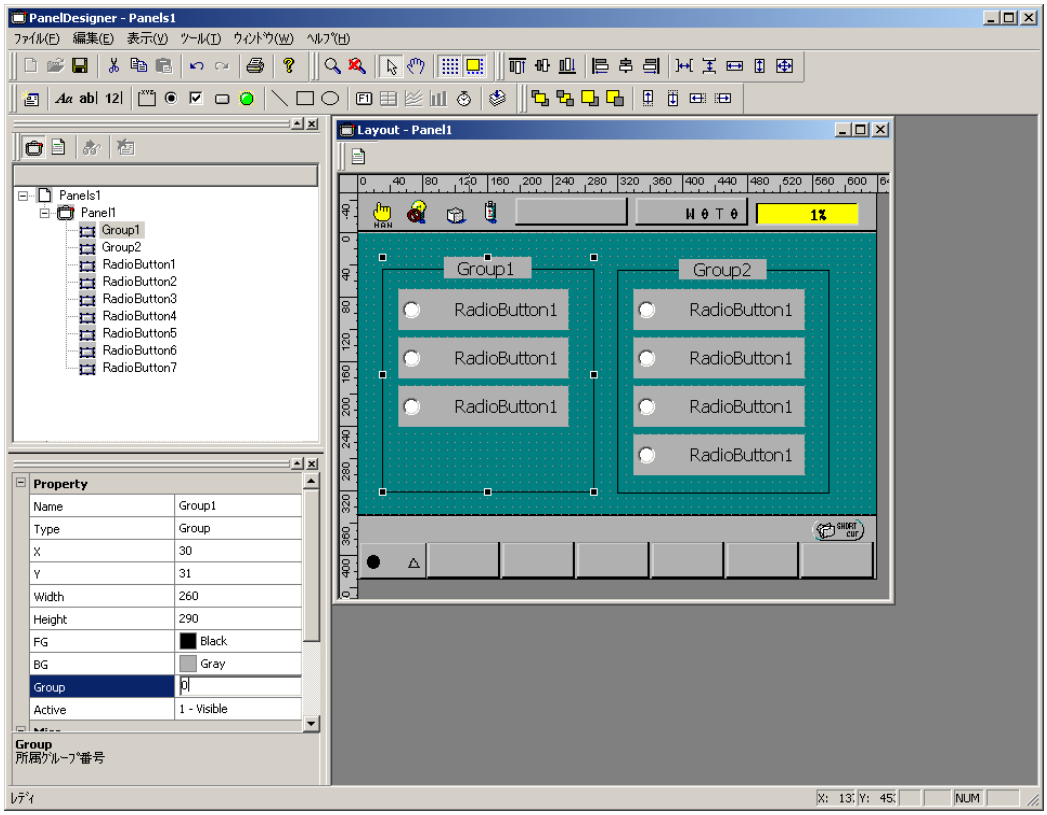
ここでは2つのグループを作成しそれぞれにラジオボタンを持たせそれぞれのグループで排他処理を行うための画面を作成します。

操作盤エディタで２つのグループとそれぞれに３つ，４つのラジオボタンを配置します。



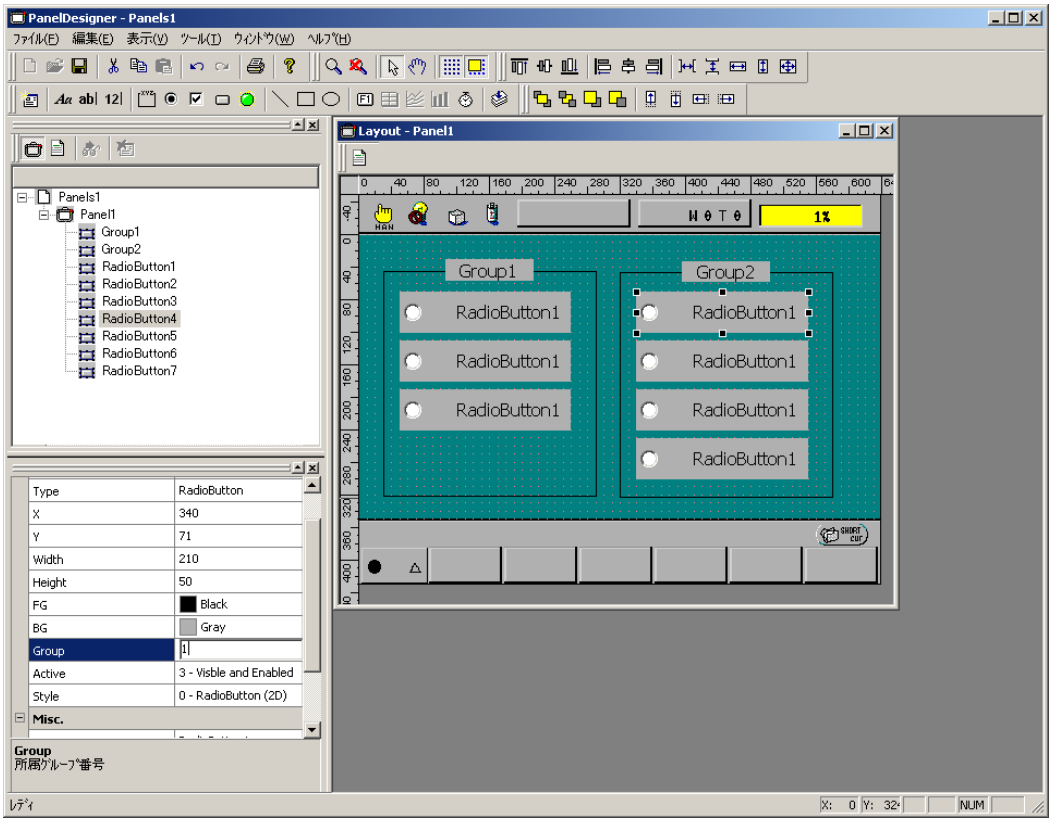
STEP 2

Group1 の番号を 0 とし Group2 の番号を 1 とします。



STEP 3

それぞれに含まれるラジオボタンのパラメータGroupを含まれるグループの番号に設定します。



これで，ラジオボタンのグループ別排他が実現できます。

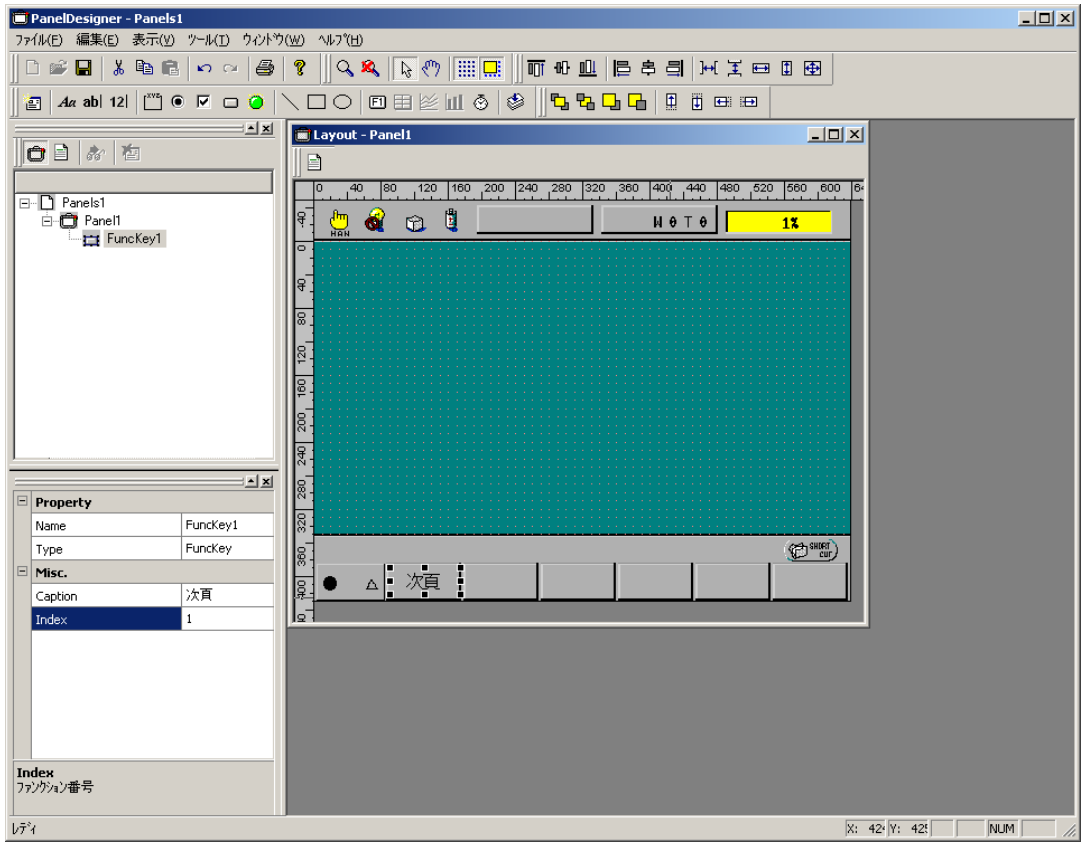
(9) ファンクションキー

ファクションキーはペンダントのファンクションキーをボタンと同様に押されたときの処理が記述でき，文字列を貼り付けることのできる部品です。ただしファンクションキー上に配置されるため他部品とは異なり位置の指定はできません。

■ファンクションキー作成の例

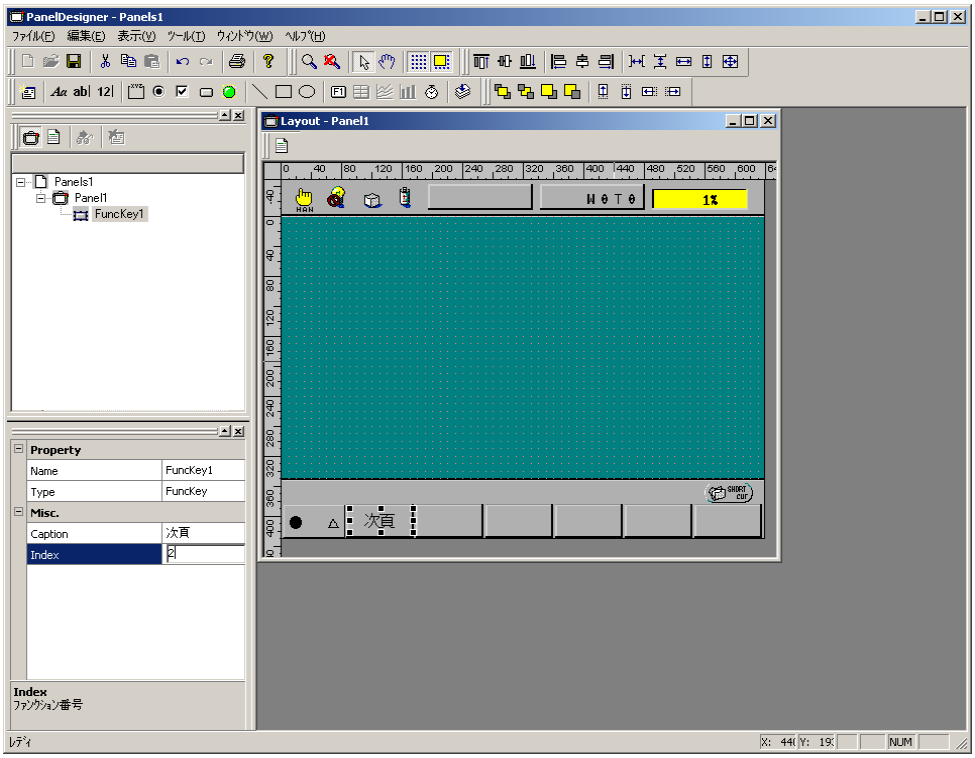
STEP 1

ボタンと同様に操作盤エディタを起動し，画面にファンクションキーを配置します。ファンクションキーは操作盤エディタ上では任意の場所に配置できますが，実際に配置されるのはファンクションキー上です。ファンクションキーの表示文字列を“次頁”とするため，Captionを“次頁”に変更します。



STEP 2

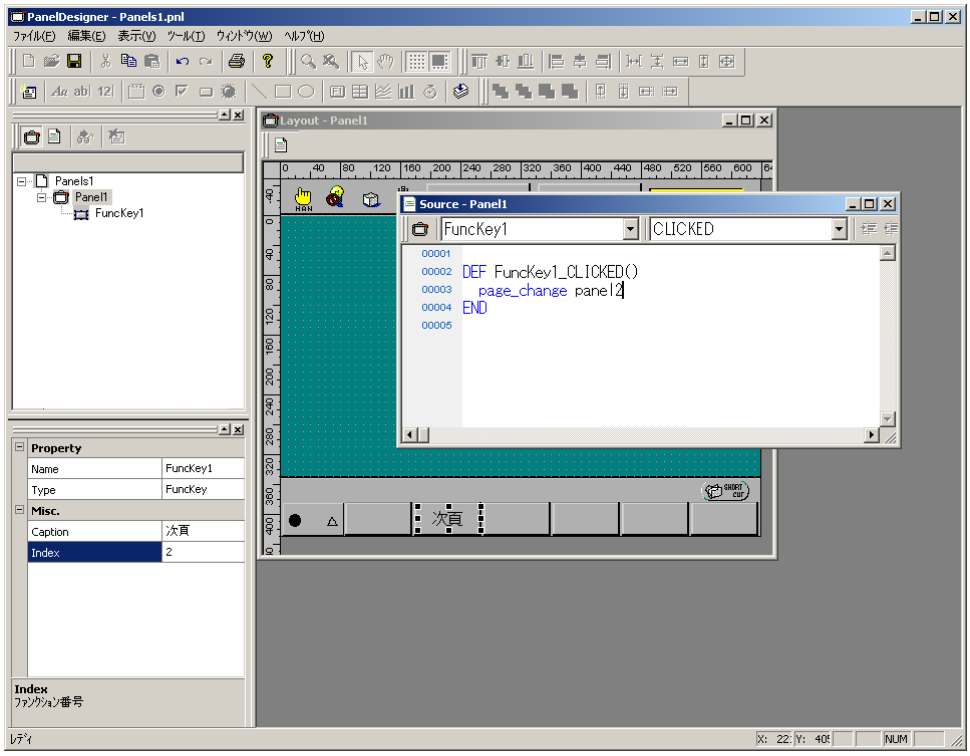
割り付けたいファンクションキー番号(0～9)を設定します．今回は2番に設定します。



STEP 3

＜ファンクションキーアクション記述＞

ファンクションキーはボタン等とは異なり押されたアクションに対する処理のみ記述できます。ここでは押されたときにパネル名”Panel2”のパネルへ移動するよう記述します。



STEP 4

＜ファンクションキーパラメータ変更＞

ファンクションキーのパラメータは他部品よりも少なくアクセスするパラメータはcaptionのみですがアクセス方法は他部品と同様です。

(10)　タイマー

タイマーは指定時間経過ごとにアクションを発生しそれに対する処理を記述できる部品です。処理が記述できるアクションはタイマー(TIMER)で間隔(interval)をパラメータで指定できます。

■タイマー作成の例

STEP 1

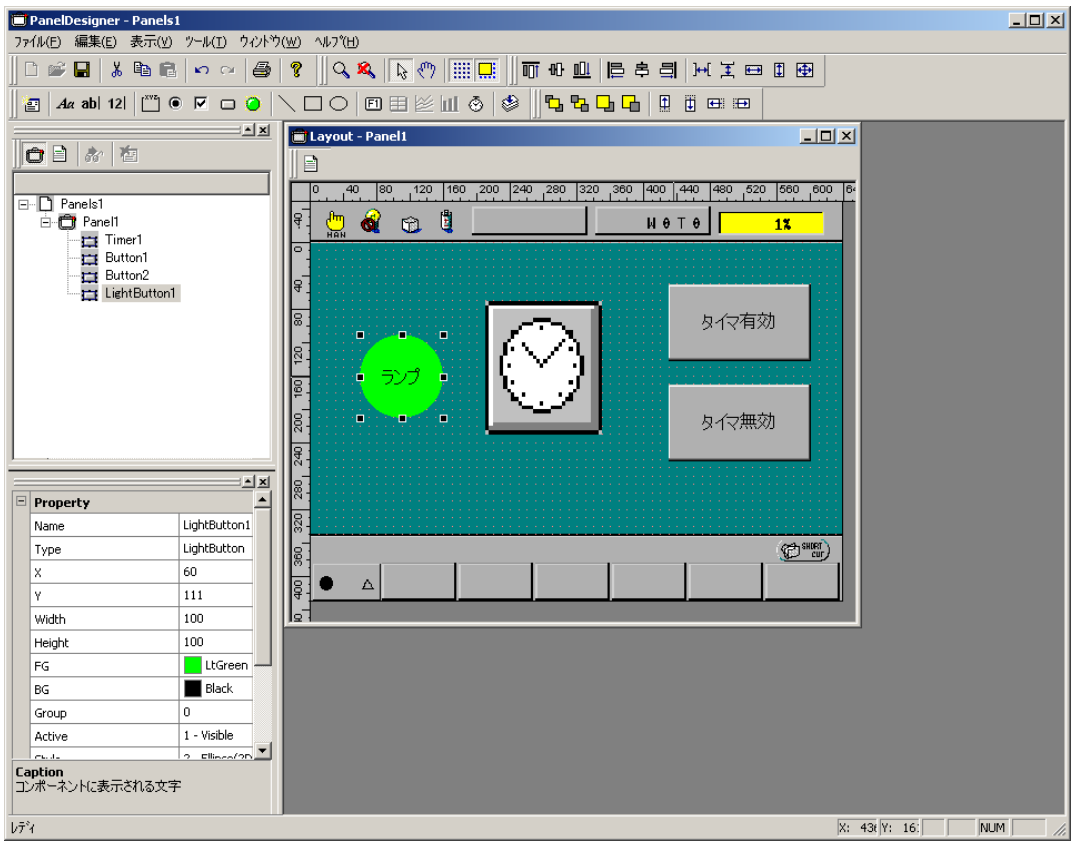
ボタンと同様に操作盤エディタを起動し，画面にタイマーを配置します。タイマーは操作盤エディタ上では任意の場所に配置できますが，実際にペンダントでは表示は行われません。

STEP 2

＜タイマーパラメータ変更＞

タイマーのパラメータは主に有効無効を指定するActive，タイマ間隔を指定するIntervalです。それらを使い，ボタンでタイマーの有効無効を設定し，タイマーのTIMERアクションでランプをON/OFFする例を次に示します。

まず，操作盤エディタで部品(タイマー，ボタン2個，ランプ)を配置します。



STEP 3　＜タイマーアクション記述＞

次にタイマーのTIMERアクション処理，ボタンのClickedアクション処理を記述します。TIMERアクションではランプがONならばOFFに，OFFならばONに切り替える処理を記述します。

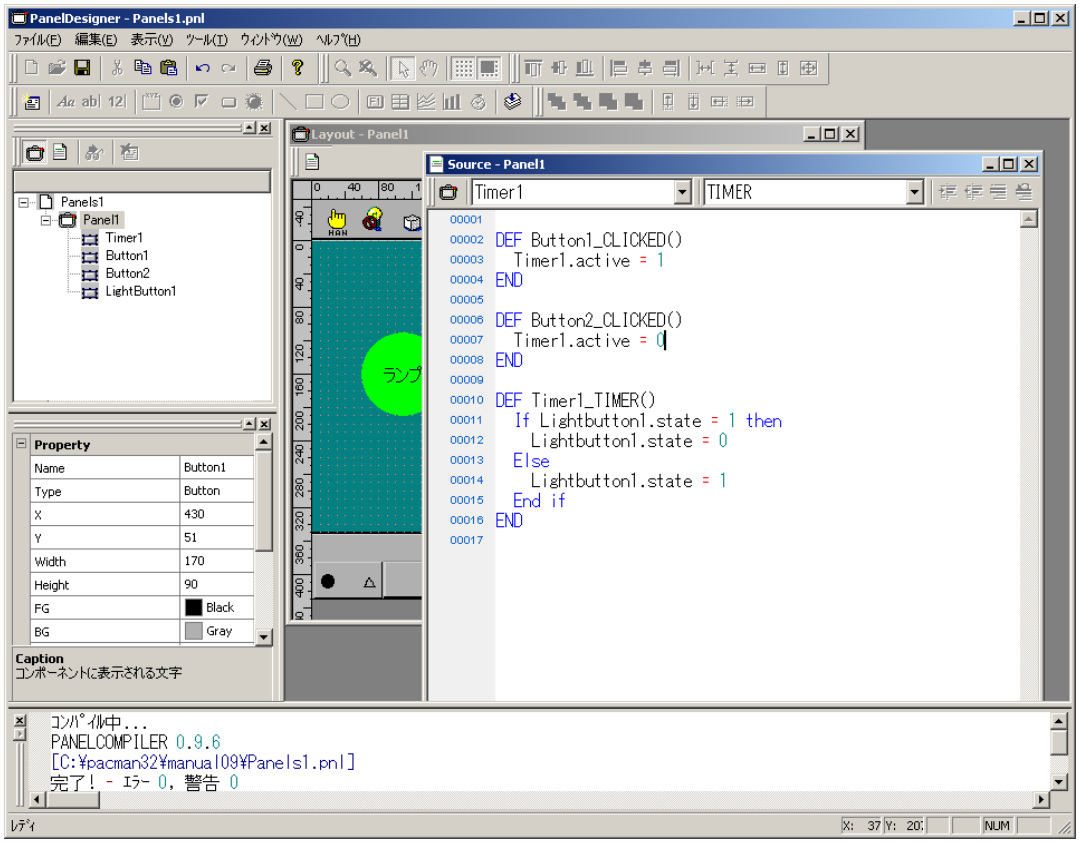
```
If Lightbutton1.state = 1 then
  Lightbutton1.state = 0
Else
  Lightbutton1.state = 1
End if
```

Button1(caption:タイマ有効)のClickedアクションではタイマーを有効に

```
Timer1.active = 1
```

Button2(caption:タイマ無効)のClickedアクションではタイマーを無効にします。

```
Timer1.active = 0
```



(11)　線

線部品は画面上に指定したパターンで線を描画します。このあとの円，四角と同じようにこれら描画のためだけの部品はアクションを持ちません。パラメータ変更は同一画面上の他部品から行うことになります。

■線作成の例

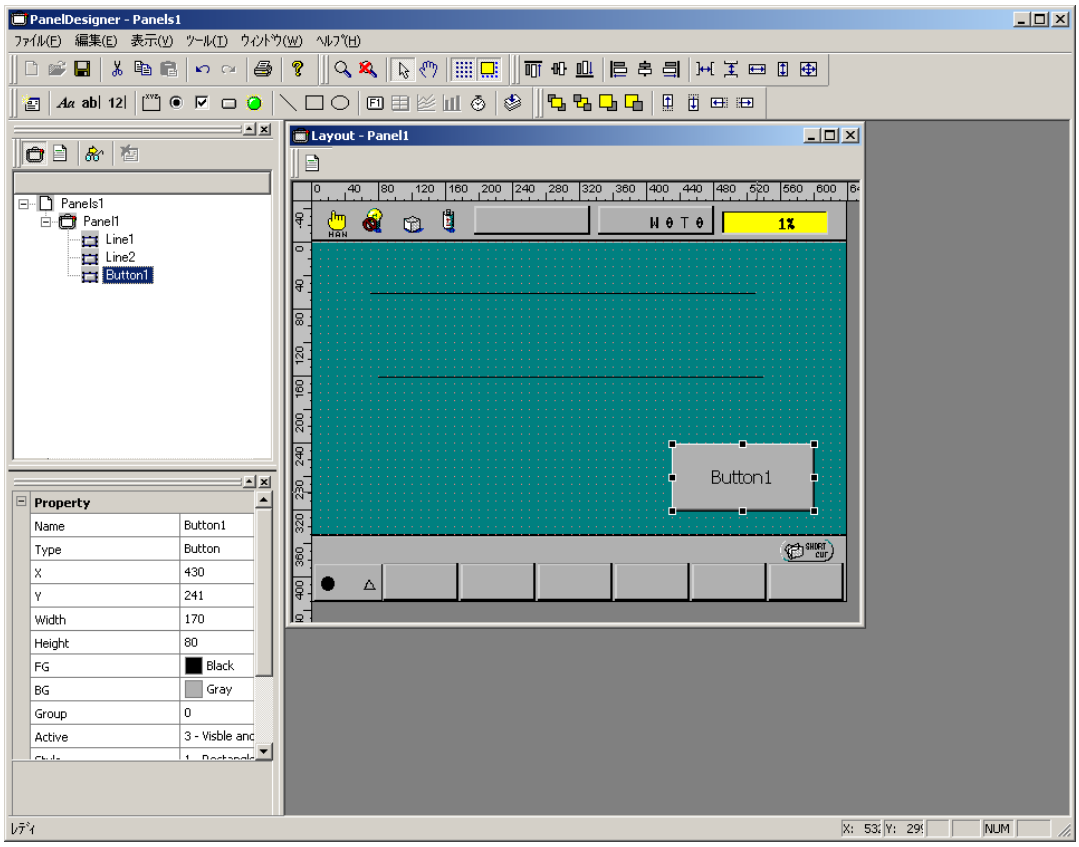
STEP 1

ボタンと同様に操作盤エディタを起動し，画面に線を描画します。

STEP 2

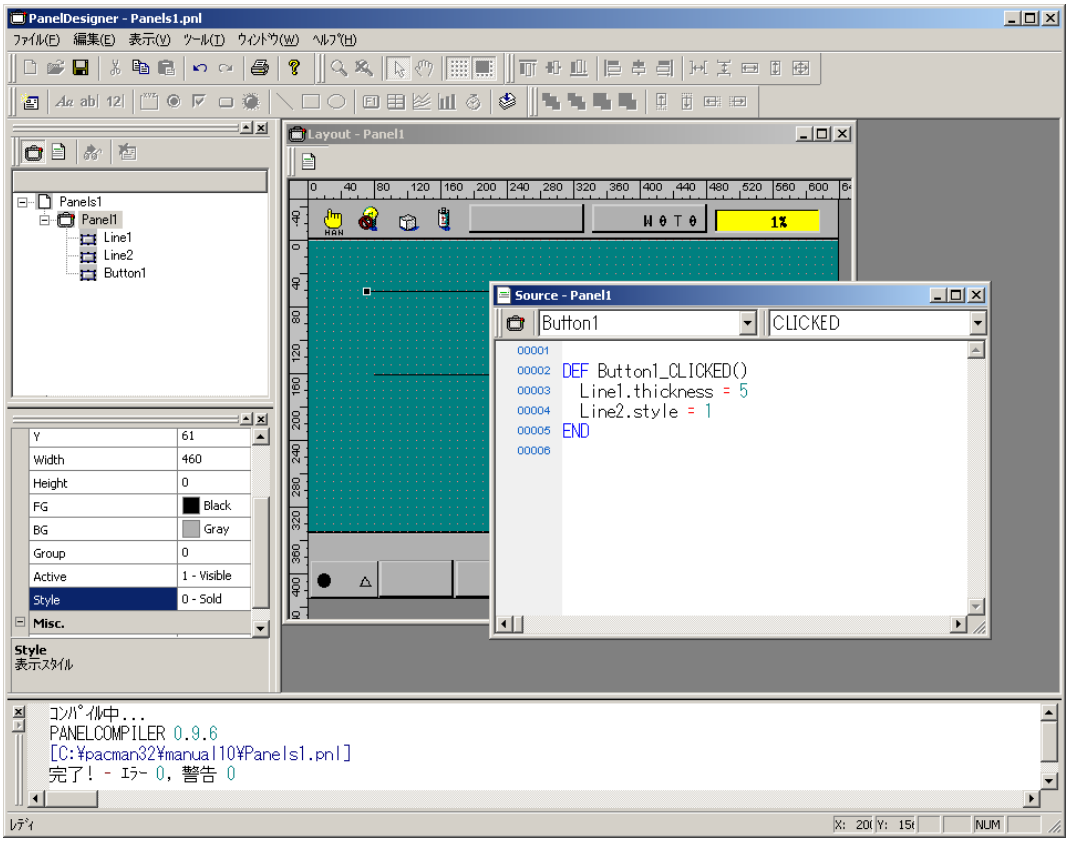
＜線パラメータ変更＞

線部品には主に線種（style），と線の太さがあります。以下にボタンが押されたときに線の太さと線種を変更する例を示します。
操作盤エディタで線2本とボタンを配置します。



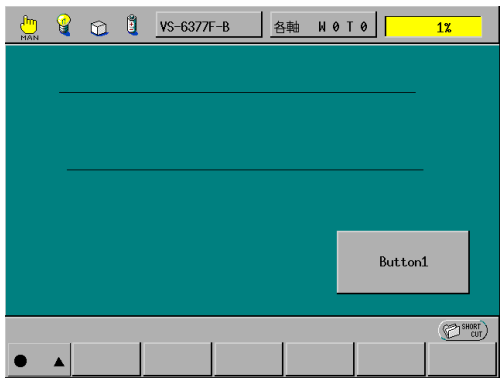
STEP 3

ボタンが押されたときに線１の太さを５ドットに，線２の線種を破線にするようソース編集画面で記述します。

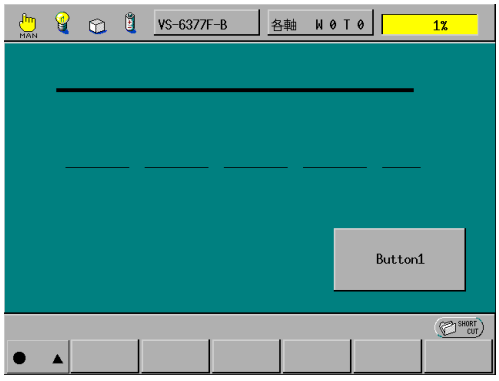


STEP 4

これでボタンを押すと線の太さと線種が変わる画面が作成できます。



ボタン押し後



(12)　円

円部品は画面上に指定したパターンで円を描画します。線，四角と同じようにこれら描画のためだけの部品はアクションを持ちません。パラメータ変更は同一画面上の他部品から行うことになります。

■円作成の例

STEP 1　ボタンと同様に操作盤エディタを起動し，画面に円を描画します。

STEP 2　**＜円パラメータ変更＞**

円部品には線部品と同様に，主に線種(style)，と線の太さがあります。変更・参照は線部品と同様に行うことができます。

(13)　四角

四角部品は画面上に指定したパターンで資格を描画します。円，線と同じようにこれら描画のためだけの部品はアクションを持ちません。パラメータ変更は同一画面上の他部品から行うことになります。

■四角作成の例

STEP 1　ボタンと同様に操作盤エディタを起動し，画面に四角を描画します。

STEP 2　**＜四角パラメータ変更＞**

四角部品には線部品と同様に，主に線種(style)，と線の太さがあります。変更・参照は線部品と同様に行うことができます。

(14)　照光式ボタン

照光式ボタンの操作は通常のボタンと同様で，表示はランプ機能を持つ特殊な部品です。そのため，押されたアクション，離されたアクション，リフレッシュアクションそれぞれに対して動作を記述することができます。ランプ，チェックボックスと同様にON/OFF状態はパラメータ(state)でアクセスすることが可能です。

■照光式ボタン作成の例

STEP 1

配置は通常のボタンと同様に操作盤エディタで行うことができます。

STEP 2

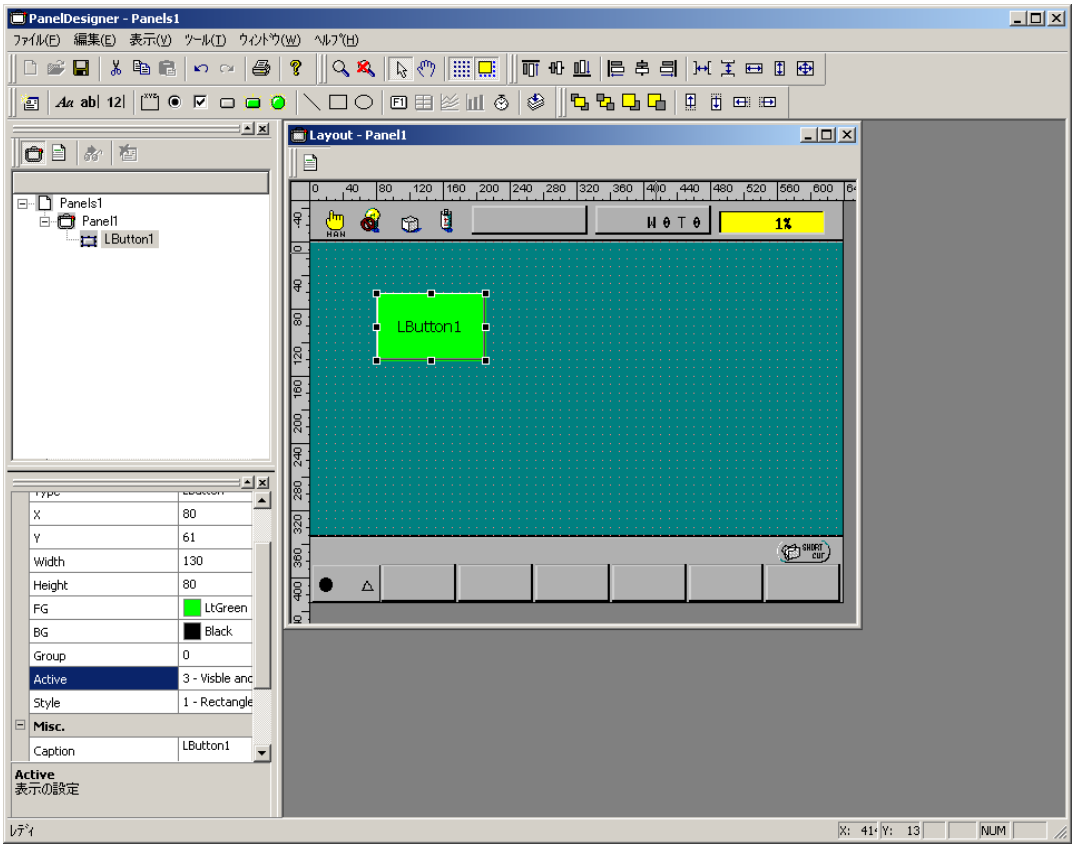
＜照光式ボタンアクション記述＞

照光式ボタンの持つアクションはClicked，Released，Refreshを持ちますのでそれぞれについて処理を記述することができます。

STEP 3

＜照光式ボタンパラメータ変更＞

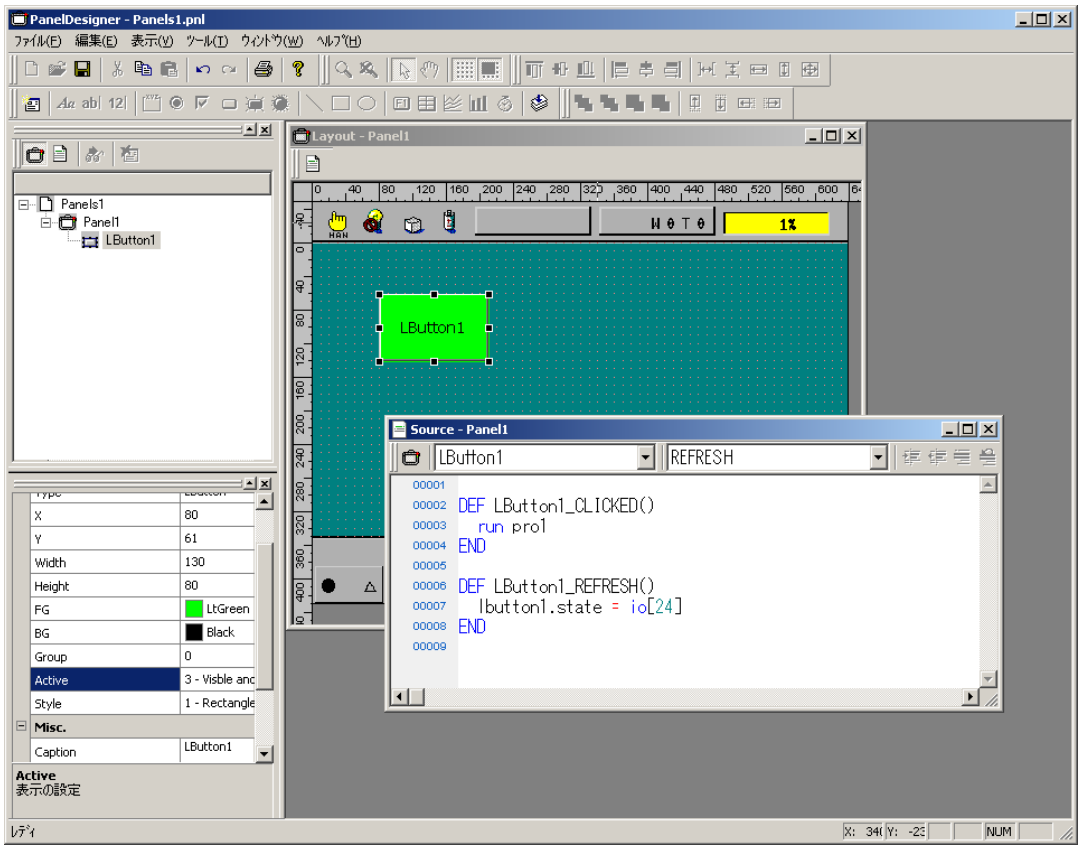
ここでは，照光式ボタンによりプログラム起動とI/O状態の表示を行う例を示します。照光式ボタンが押されたときに同一フォルダのプログラム（起動語2秒でIO[24]をONする）を起動し，IO[24]の状態を照光式ボタンに反映します。
まずそれぞれの部品を配置します。



STEP 4

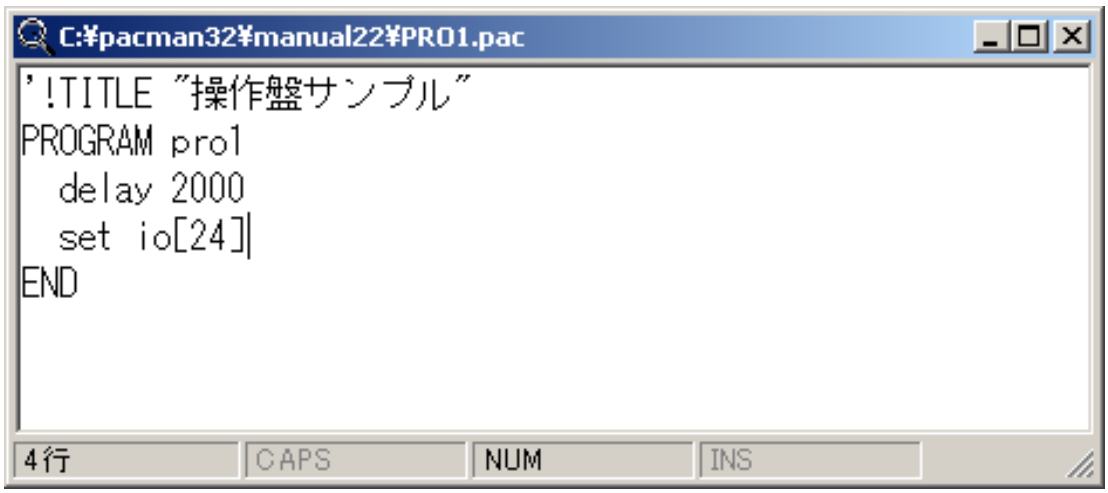
次に，上記機能を実現するための部品アクション記述を記述します。

```
lbutton1.state = io[24]     'io[24]の状態を lbutton1 の状態に反映させる
```



STEP 5

さらに起動するプログラムをPACマネージャで記述します。



これらをコンパイルしコントローラに転送することで上記機能が実現できます。

2.3 PAC 言語，システムとのインタフェース

PAC言語と操作盤とのデータ交換はグローバル変数，フォルダ変数を介して行うことができます。またシステムとのインタフェースはsysstateコマンド，I/O変数で実現します。

2.3.1 PAC 変数の取得・表示

PAC変数にはグローバル変数，ローカル変数，フォルダ変数が存在しますが，操作盤からアクセスできるのはグローバル変数とフォルダ変数です。操作盤から呼び出す場合，グローバル変数は宣言なしで使用できますがフォルダ変数はEXTERN宣言で使用することを記述しないと使用することはできません。以下にそれらの値を操作盤に表示する例を示します。

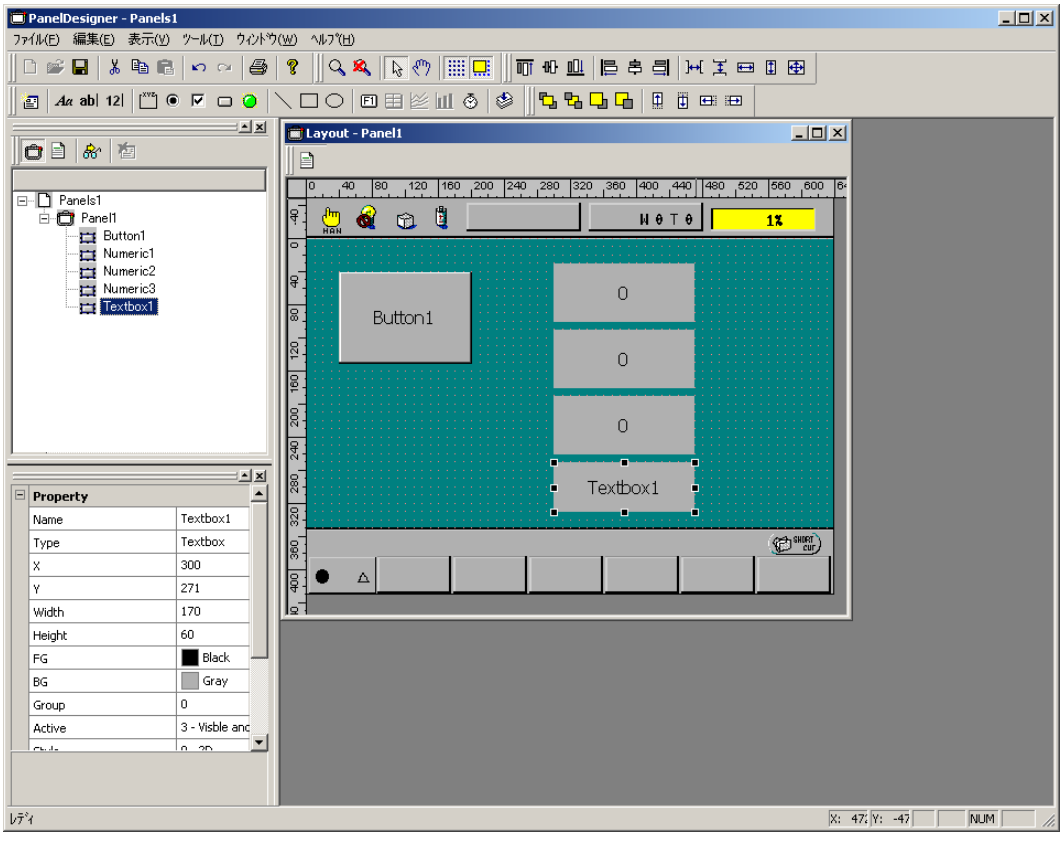
■グローバル変数を操作盤に表示する例

グローバル変数へのアクセスは
　変数種類（I：整数,F：単精度実数,D：倍精度実数,S：文字列）［変数番号］
　の形式で実行されます。
たとえばグローバル整数変数の10番にアクセスするにはI[10]と記述します。

操作盤でボタンが押されたときにI，F,，D，S型のグローバル変数を取得しそれぞれを数値入力ボックス,テキストボックスに表示を行う例を以下に示します。

STEP 1

まず，TPDesignerでデータ取得処理を記述するボタン，取得したI，F,，D型グローバル変数の表示を行う数値入力ボックス,S型グローバル変数表示を行うテキストボックスを配置します。

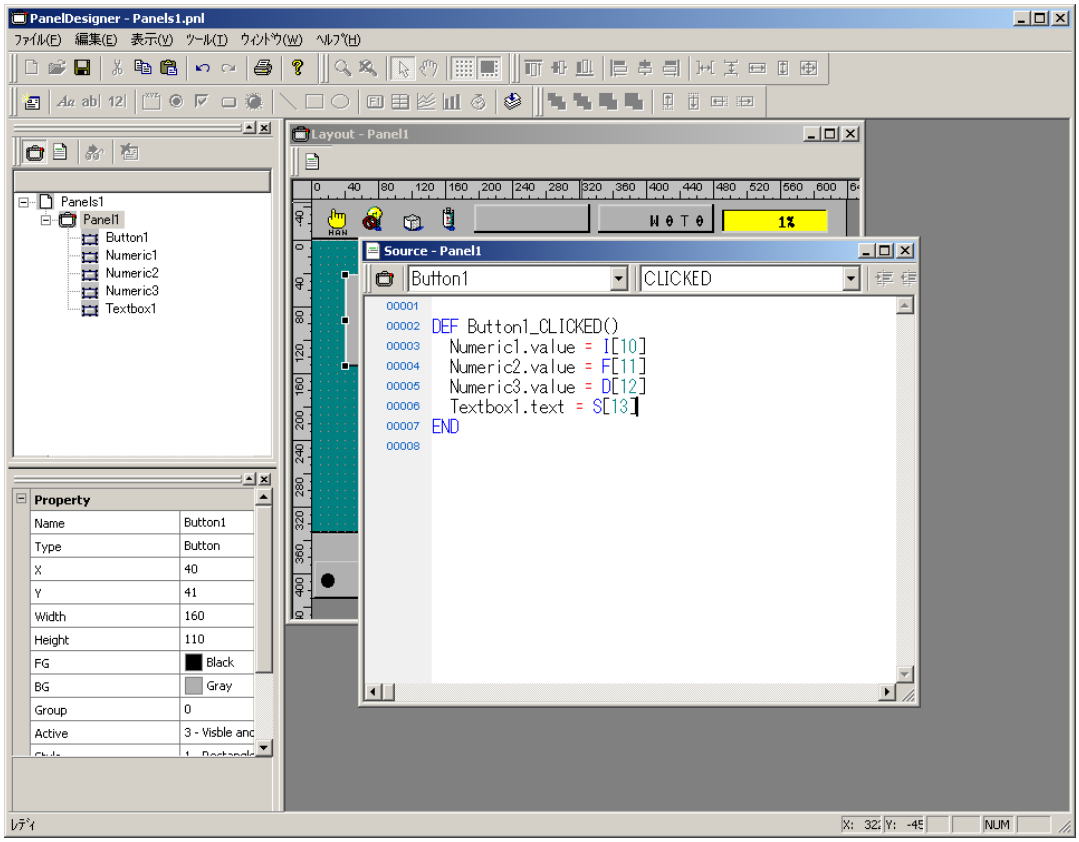


STEP 2

次にソース編集画面でボタンが押されたときの処理を記述します。

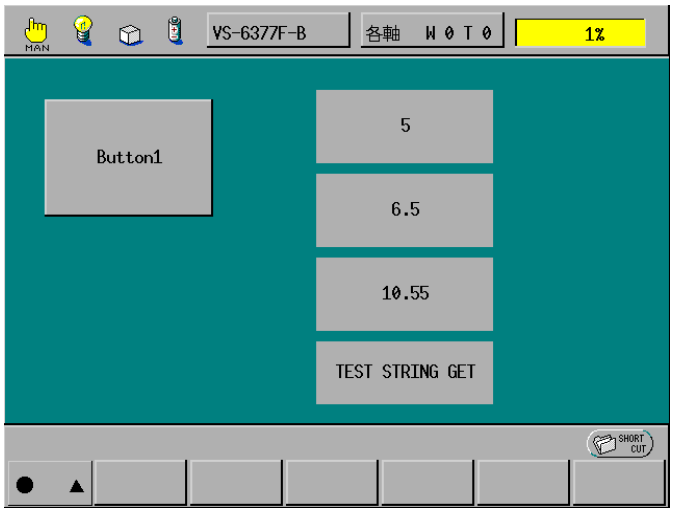
ここではI,F,D,S型グローバル変数のそれぞれ10番，11番，12番，13番を数値入力ボックス，テキストボックスの値に代入するように指定します。

```
Numeric1.value = I[10]
Numeric2.value = F[11]
Numeric3.value = D[12]
Textbox1.text  = S[13]
```



STEP 3

以下にコントローラに転送しボタンを押したときの結果を示します。



■フォルダ変数を操作盤に表示する例

フォルダ変数へのアクセスはボタン等のアクションプログラム内にEXTERN型宣言(DEFINT, DEFSNG, DEFDBL,DEFSNG)フォルダ変数の形式で記述することにより可能となります。
たとえばフォルダ整数変数のitestにアクセスするにはアクションプログラム内に「EXTERN DEFINT itest」と宣言しプログラム内で「itest」と記述します。

例として操作盤でボタンが押されたときに整数，単精度実数，倍精度実数，文字列型のフォルダ変数を取得しそれぞれを数値入力ボックス，テキストボックスに表示を行う例を以下に示します。

STEP 1

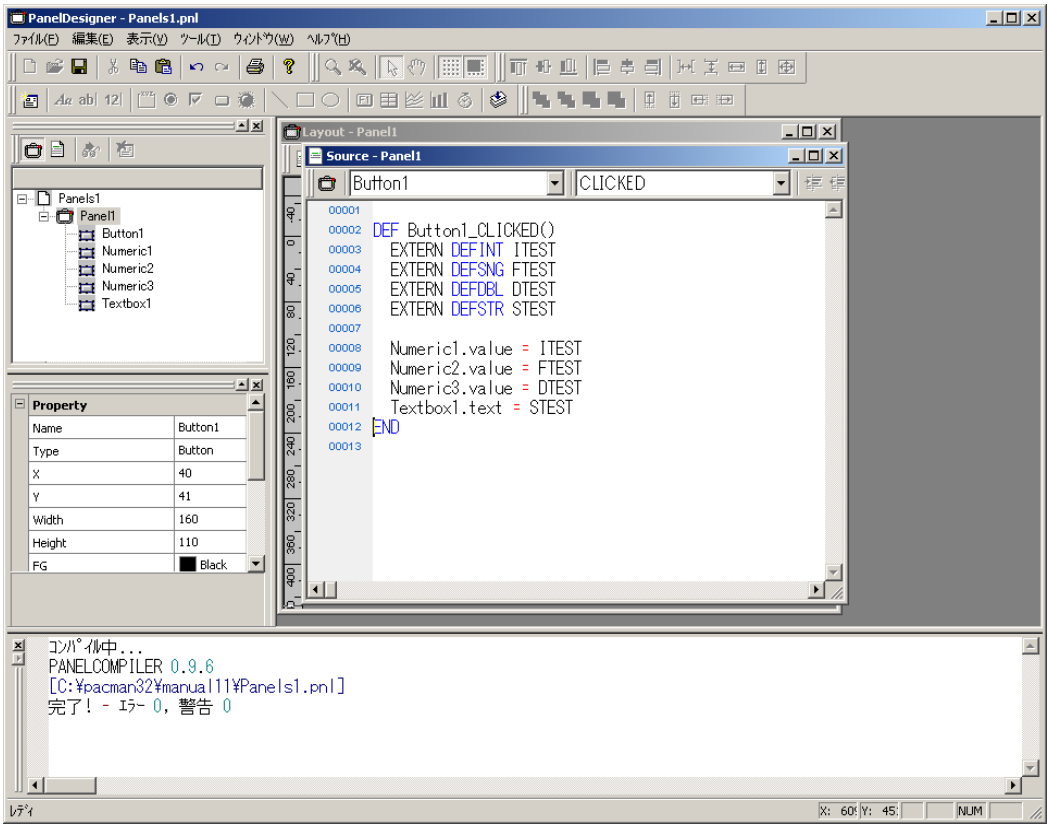
まず,操作盤エディタでデータ取得処理を記述するボタン,取得した整数,単精度実数,,倍精度実数型フォルダ変数の表示を行う数値入力ボックス,文字列型フォルダ変数表示を行うテキストボックスを配置します。
グローバル変数で示した例と同様に操作盤エディタで部品を配置します。

STEP 2

次にソース編集画面でボタンが押されたときの処理を記述します。

ここでは整数，単精度実数,，倍精度実数，文字列型フォルダ変数のそれぞれitest, ftest, dtest, stestを数値入力ボックス，テキストボックスの値に代入するように指定します。

```
EXTERN DEFINT ITEST
EXTERN DEFSNG FTEST
EXTERN DEFDBL DTEST
EXTERN DEFSTR STEST
Numeric1.value = ITEST
Numeric2.value = FTEST
Numeric3.value = DTEST
Textbox1.text = STEST
```



2.3.2 PAC 変数の変更

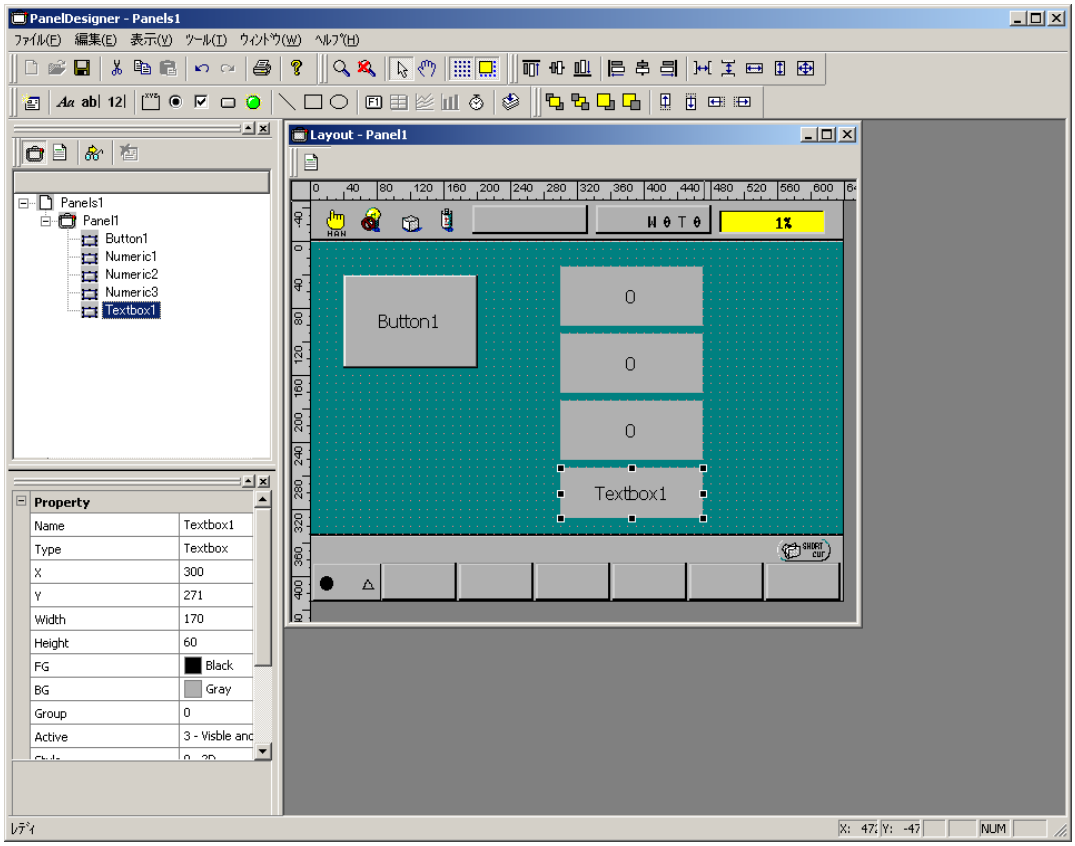
変更に関しても「2.3.1　PAC変数の取得・表示」と同様の形式でアクセスできます。

■グローバル変数の例

ここでは操作盤でボタンが押されたときに数値入力ボックス，テキストボックスの値をI，F，D，S型のグローバル変数に格納する例を以下に示します。

STEP 1

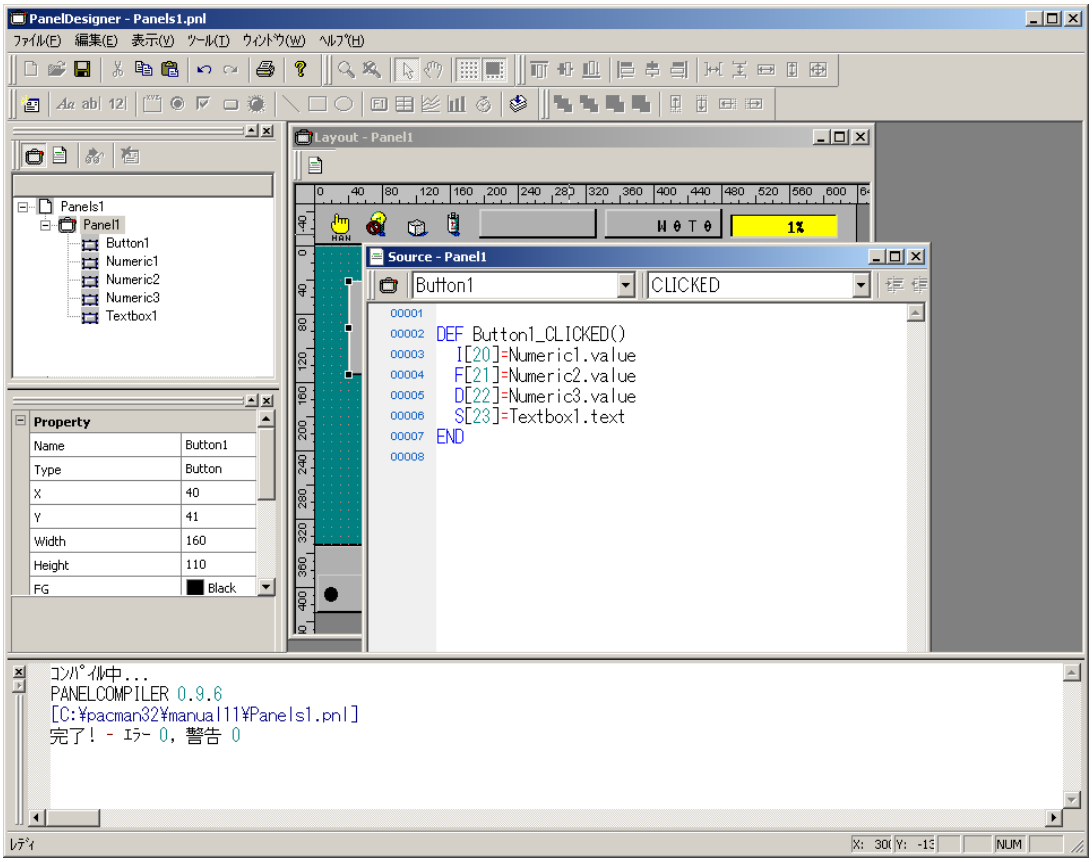
まず，操作盤エディタでデータ格納処理を記述するボタン，I，F，D型グローバル変数へ書き込むデータを持つ数値入力ボックス，S型グローバル変数へ書き込みを行うデータを持つテキストボックスを配置します。



STEP 2

次にソース編集画面でボタンが押されたときの処理を記述します。
ここではI,F,D,S型グローバル変数のそれぞれ20番，21番，22番，23番へ数値入力ボックス，テキストボックスの値を代入するように指定します。

```
I[20]=Numeric1.value
F[21]=Numeric2.value
D[22]=Numeric3.value
S[23]=Textbox1.text
```



■フォルダ変数の例

フォルダ変数へのアクセスは「2.3.1　PAC変数の取得・表示」の場合と同一です。操作盤でボタンが押されたときに，数値入力ボックス，テキストボックスの値を取得し，整数，単精度実数，倍精度実数，文字列型のフォルダ変数にそれぞれを格納する例を以下に示します。

STEP 1

まず，操作盤エディタでデータ取得処理を記述するボタン，取得した整数，単精度実数，倍精度実数型フォルダ変数の表示を行う数値入力ボックス，文字列型フォルダ変数表示を行うテキストボックスを配置します。

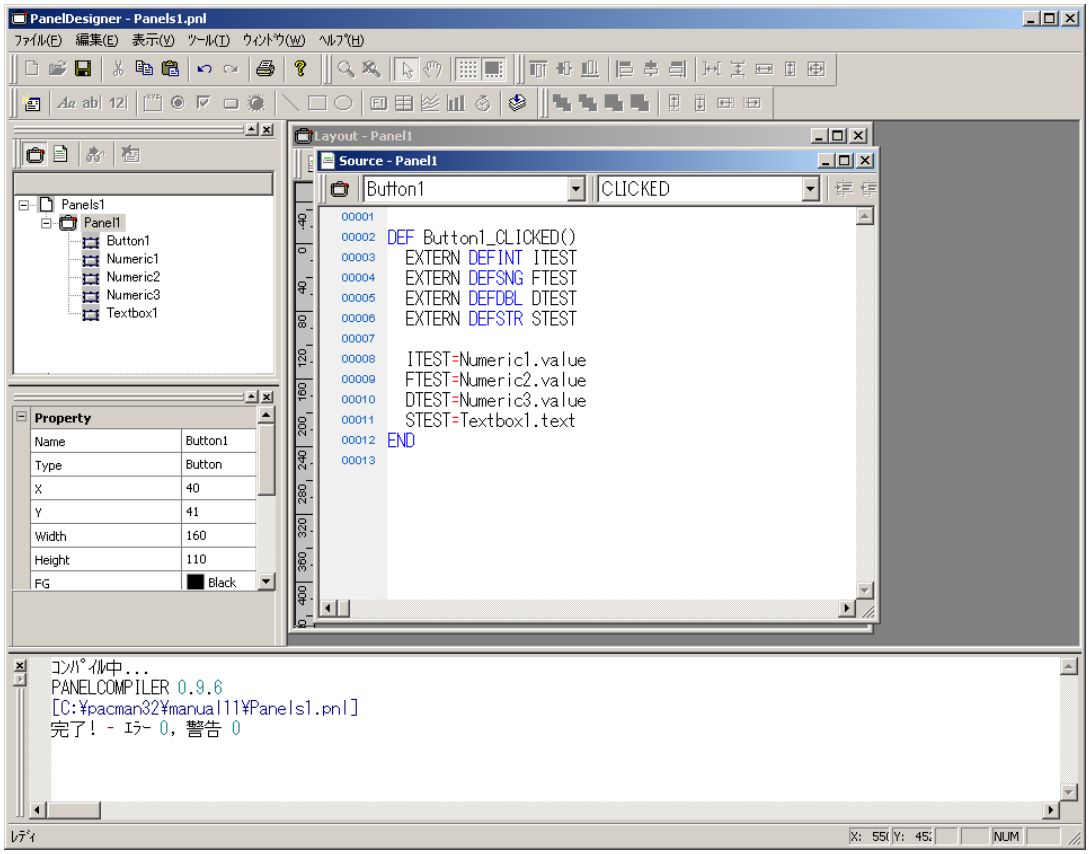
グローバル変数で示した例と同様に操作盤エディタで部品を配置します。

STEP 2

次にソース編集画面でボタンが押されたときの処理を記述します。
ここでは整数，単精度実数，倍精度実数，文字列型フォルダ変数のそれぞれitest，ftest，dtest，stestを数値入力ボックス，テキストボックスの値に代入するように指定します。

```
EXTERN DEFINT ITEST
EXTERN DEFSNG FTEST
EXTERN DEFDBL DTEST
EXTERN DEFSTR STEST

ITEST=Numeric1.value
FTEST=Numeric2.value
DTEST=Numeric3.value
STEST=Textbox1.text
```



2.3.3 I/O 状態の取得

ロボットコントローラのI/O状態を取得するためにはI/Oグローバル変数を使う方法とDEFIO宣言によるローカルI/O変数を使用する方法があります。後者は他のローカル変数とまとめて後述します。ここではI/Oグローバル変数を使う方法について説明します。

■I/Oグローバル変数の例

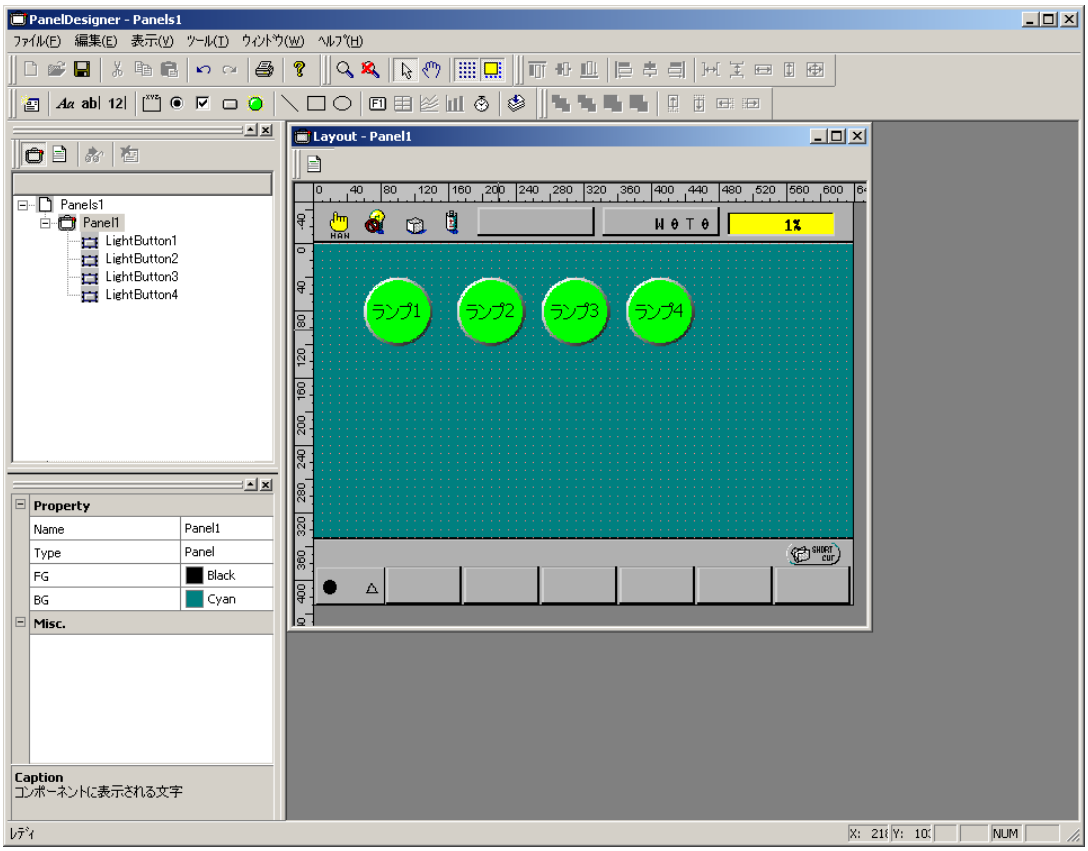
I/Oグローバル変数へのアクセスは次のの形式で行います。

IO［IO番号］

例としてI/O番号24～27の状態を定期的に取得し，それをランプの状態として表示します。

STEP 1

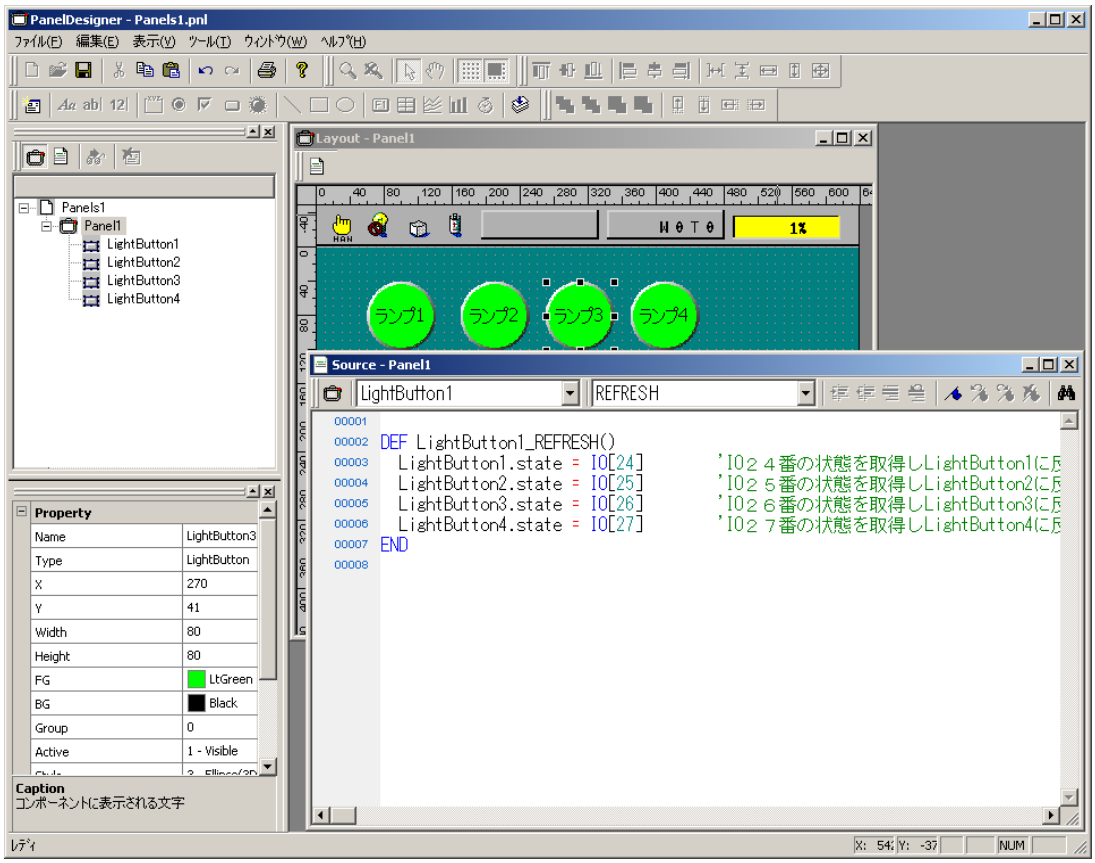
操作盤エディタでI/O状態を表示するランプ4個を配置します。



STEP 2

次にソース編集画面でI/O状態を取得し，ランプに反映する処理を記述します。
定期的に処理を行うためにランプのREFRESHイベントを利用します。
ここではソース編集画面のコンボボックスからLightButton1とREFRESHを選択しプログラムの外側を作成し，内部に処理を記述します。

```
LightButton1.state = IO[24]     'I/O 24番の状態を取得しLightButton1に反映
LightButton2.state = IO[25]     'I/O 25番の状態を取得しLightButton2に反映
LightButton3.state = IO[26]     'I/O 26番の状態を取得しLightButton3に反映
LightButton4.state = IO[27]     'I/O 27番の状態を取得しLightButton4に反映
```



2.3.4 I/O 状態の変更

システムのI/O状態変更を行うにはSET/RESETコマンドを使用します。

ONする場合： SET IO[IO番号]

OFFする場合： RESET IO[IO番号]

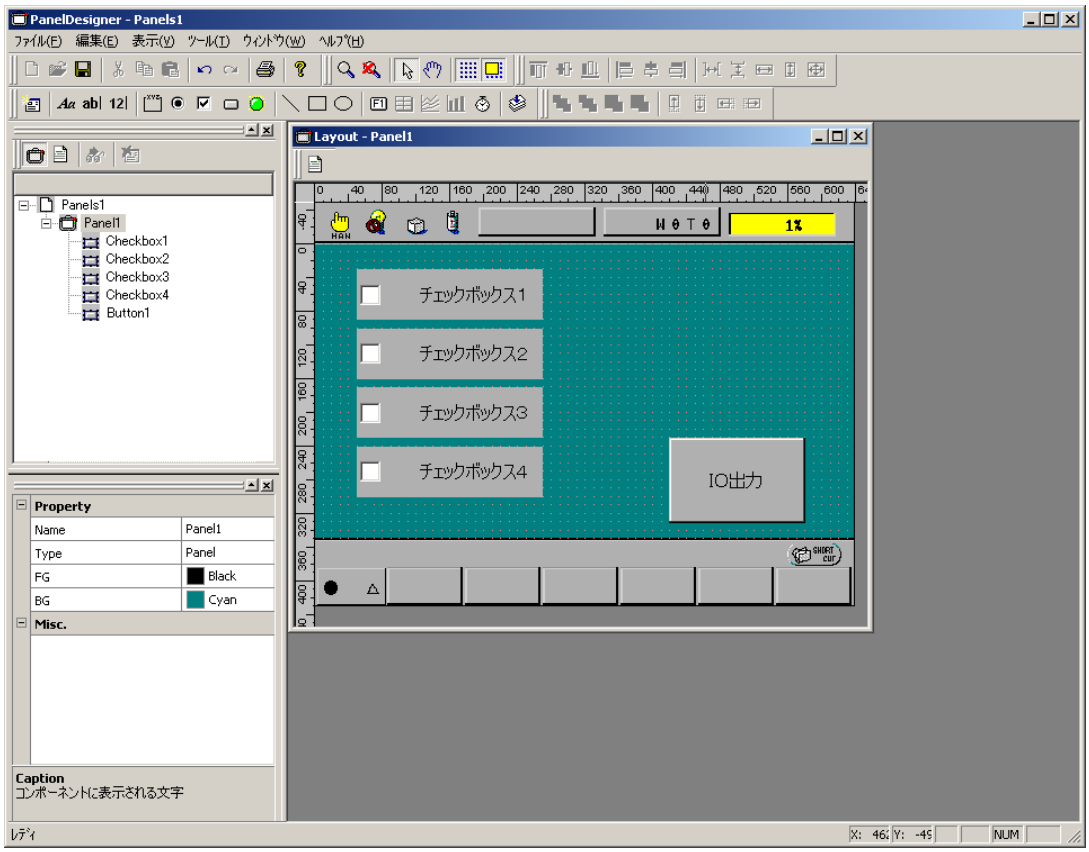
の形式で指定します。

■操作盤でのI/O状態の変更例

例として，ボタンが押されたときにチェックボックスの状態をシステムのI/O状態（I/O番号28～31）に反映します。

STEP 1

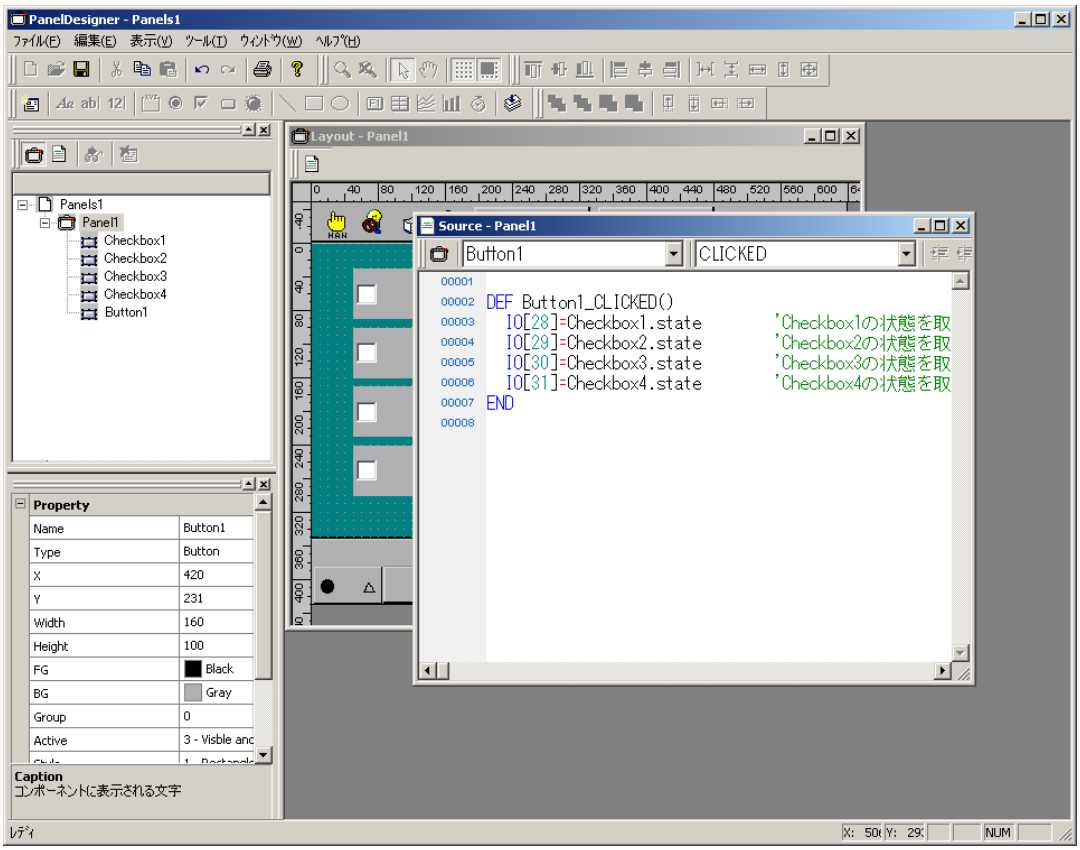
操作盤エディタでチェックボックスを4個と、押されたときにI/Oへのデータ出力を行うボタンを配置します。



STEP 2

次にソース編集画面でチェックボックス状態を取得し，I/0へ出力する処理を記述します。

```
IO[28]=Checkbox1.state        'Checkbox1 の状態を取得し I/0  28 番に反映
IO[29]=Checkbox2.state        'Checkbox2 の状態を取得し I/0  29 番に反映
IO[30]=Checkbox3.state        'Checkbox3 の状態を取得し I/0  30 番に反映
IO[31]=Checkbox4.state        'Checkbox4 の状態を取得し I/0  31 番に反映
```



2.3.5　システム状態の取得

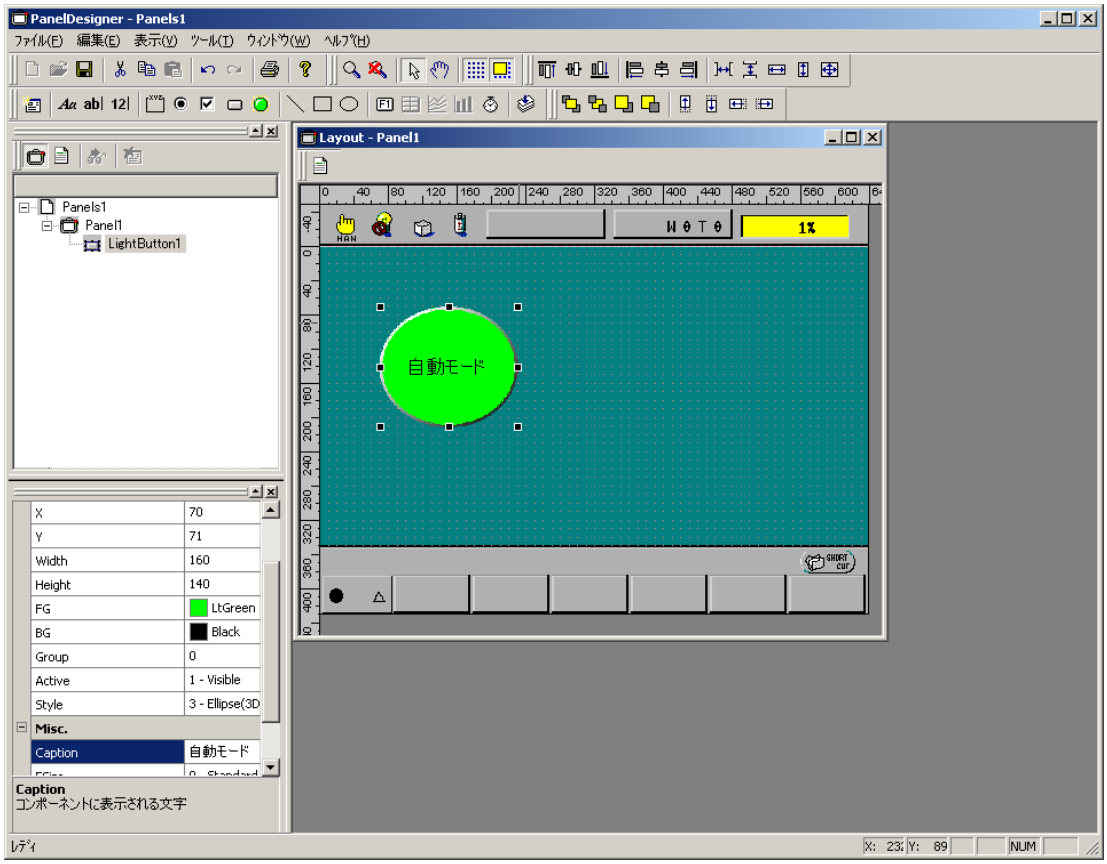
システム状態の取得にはSYSSTATEコマンドを使用してください。
SYSSTATEコマンドの詳細については後述のコマンドリファレンスを参照してください。

■操作盤でのシステム状態の取得例

ここではコントローラの自動モード状態を取得しそれをランプ状態に反映させる例を示します。

STEP 1

操作盤エディタでランプを配置します。

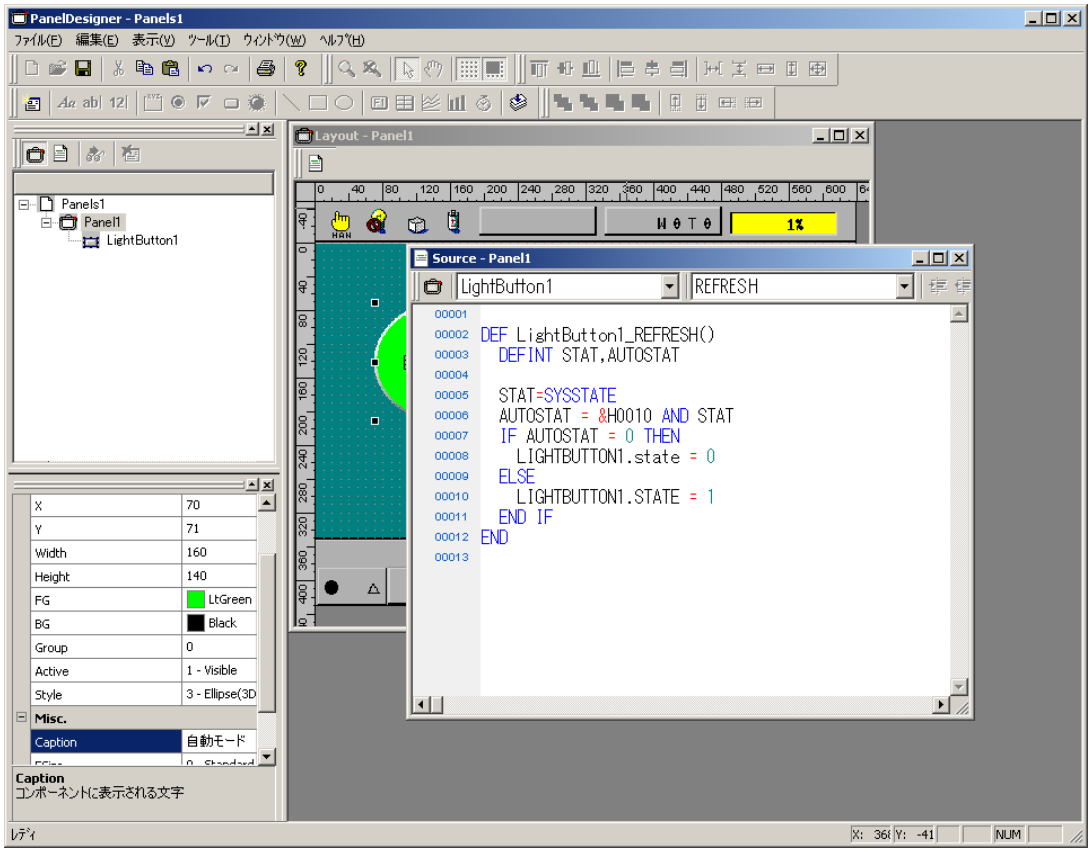


STEP 2

ランプのREFRESHアクションで自動モード状態を取得しそれを表示する処理を記述します。ここではLightButton1のREFRESHアクションを用い、コントローラ状態取得、モード情報の切り出し、ランプの点灯・消灯を行います。

```
DEFINT STAT, AUTOSTAT

STAT=SYSSTATE
AUTOSTAT = &H0010 AND STAT
IF AUTOSTAT = 0 THEN
  LIGHTBUTTON1.state = 0
ELSE
  LIGHTBUTTON1.STATE = 1
END IF
```



2.4　ページ切り替え

ページ切り替えはPAGE_CHANGEコマンドで行います。このコマンドでは同一フォルダ内の別画面への移動と別フォルダの画面への移動も可能です。
書式は以下の書式で指定します。

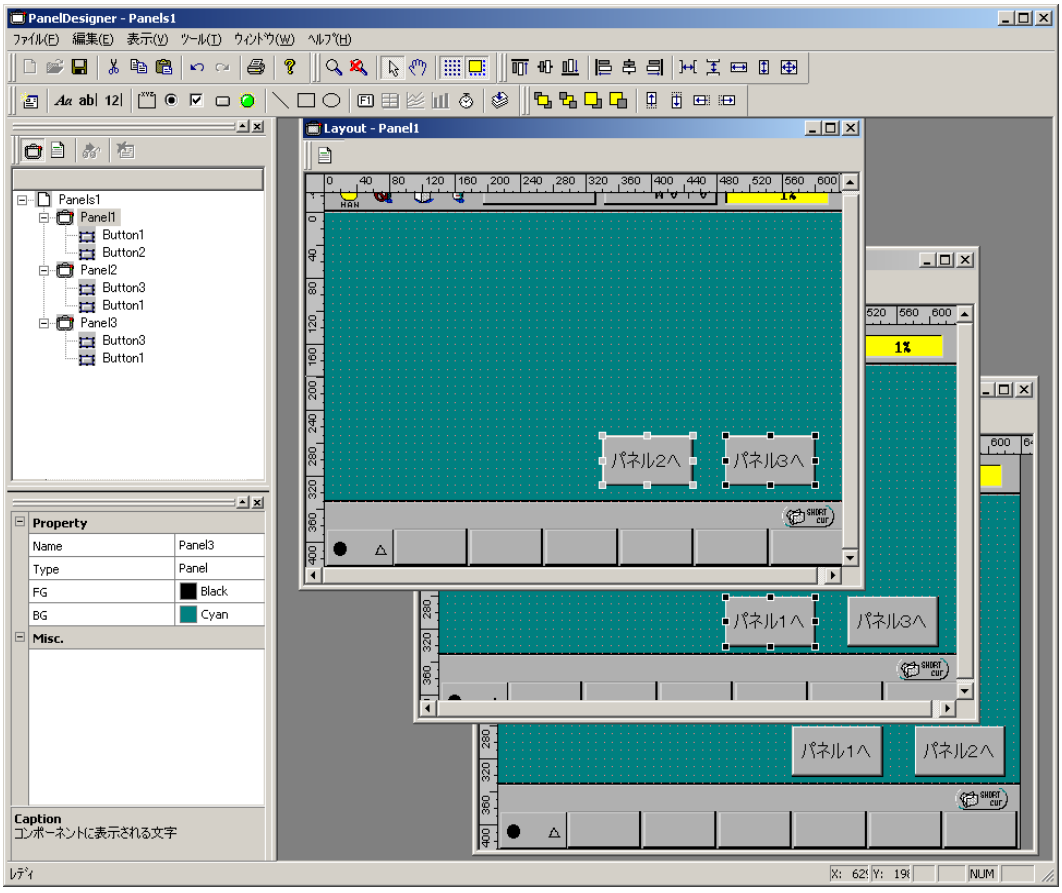
同一フォルダ内での移動：　PAGE_CHANGE　パネル名
別フォルダへの移動：　PAGE_CHANGE　パス名．パネル名
ルートフォルダに移動:　PAGE_CHANGE　￥パネル名

2.4.1　ページ移動の例

ここでは同一フォルダ内3画面の移動をボタンが押されたときに行う例を示します。それぞれの画面では別画面に移動を行うボタン2個を持つように作成します。

STEP 1

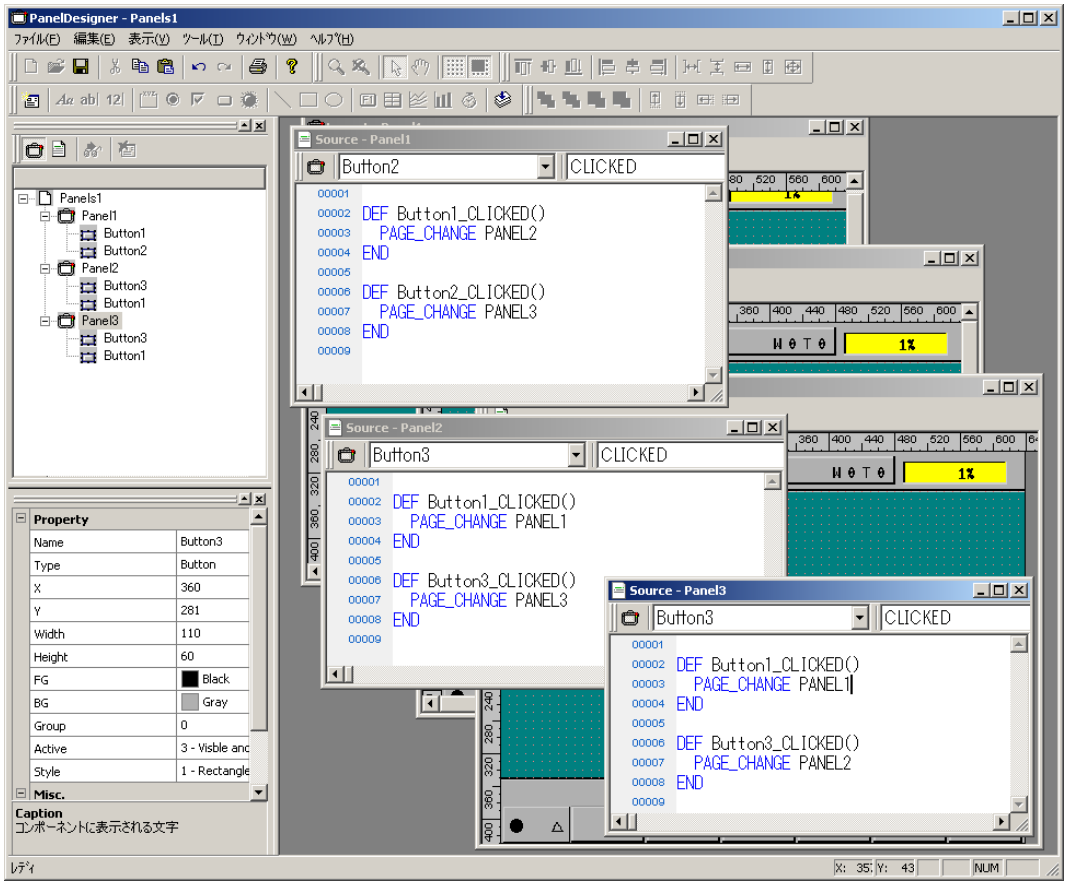
操作盤エディタで画面を3個とそれぞれに2個づつボタンを配置し，ボタンの表示文字列をそれぞれ別画面の名前などにします。



STEP 2

次にソース編集画面でそれぞれのボタンが押されたときに別の画面へ移動するよう処理を記述します。

```
PAGE_CHANGE PANEL1          'パネル名 PANEL1 の画面へ移動
PAGE_CHANGE PANEL2          'パネル名 PANEL2 の画面へ移動
PAGE_CHANGE PANEL3          'パネル名 PANEL3 の画面へ移動
```



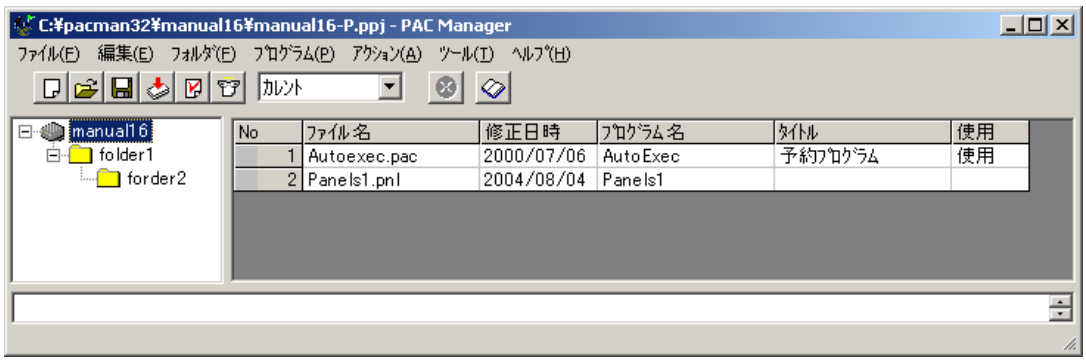
これで３画面間の移動を行うことが可能となります。

2.4.2　フォルダ間移動の例

ここでは3階層フォルダの画面移動をボタンが押されたときに行う例を示します。それぞれの画面では別画面に移動を行うボタンを持つように作成します。

STEP 1

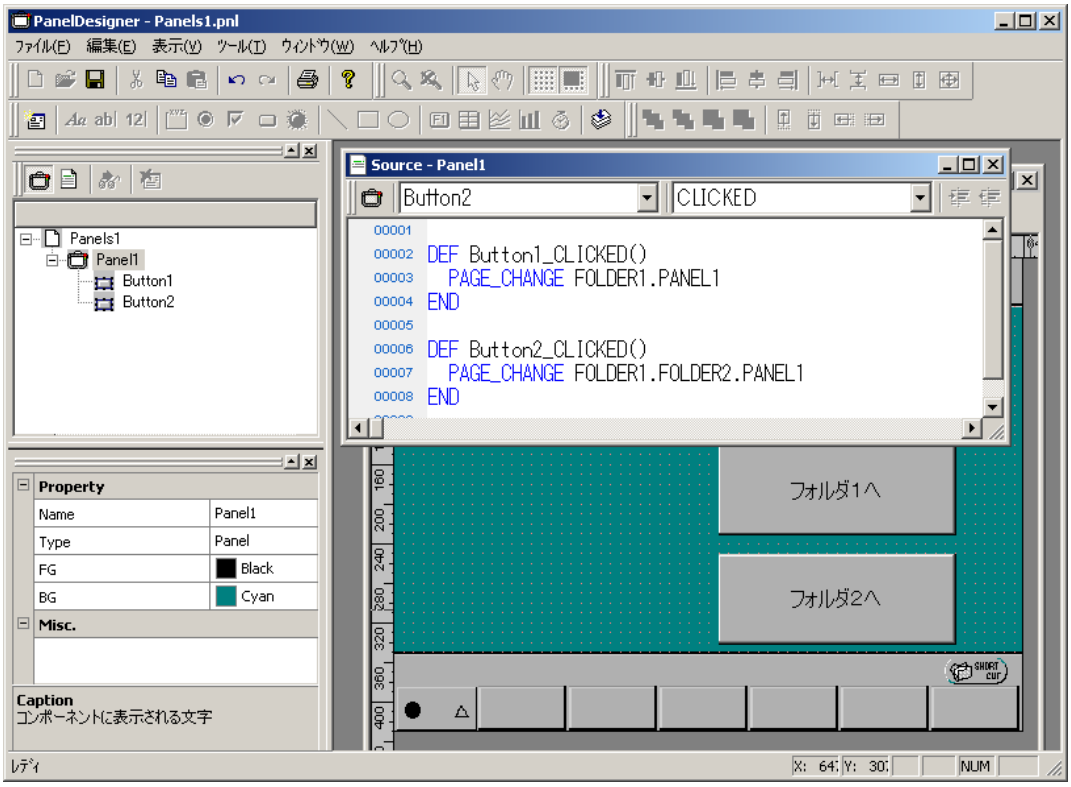
PAC Managerで3階層のフォルダを作成し操作盤エディタでそれぞれのフォルダに画面を作成し、それぞれに2個づつボタンを配置し、ボタンの表示文字列をそれぞれ別画面の名前などにします。



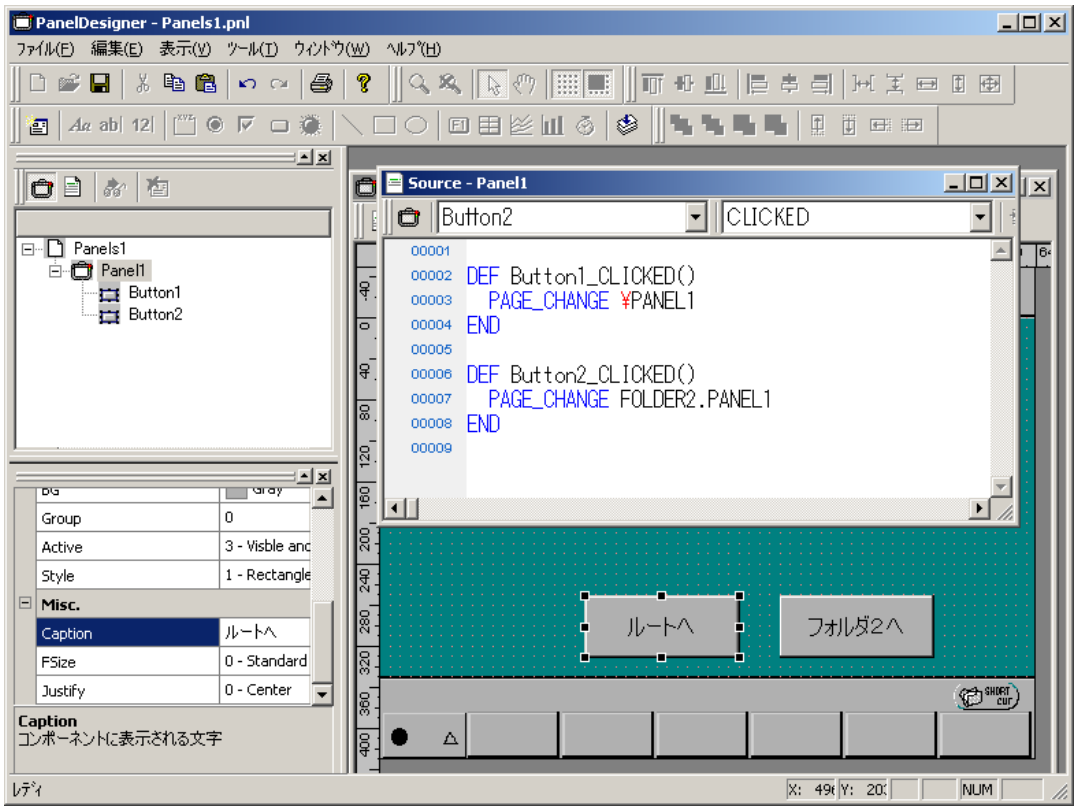
STEP 2

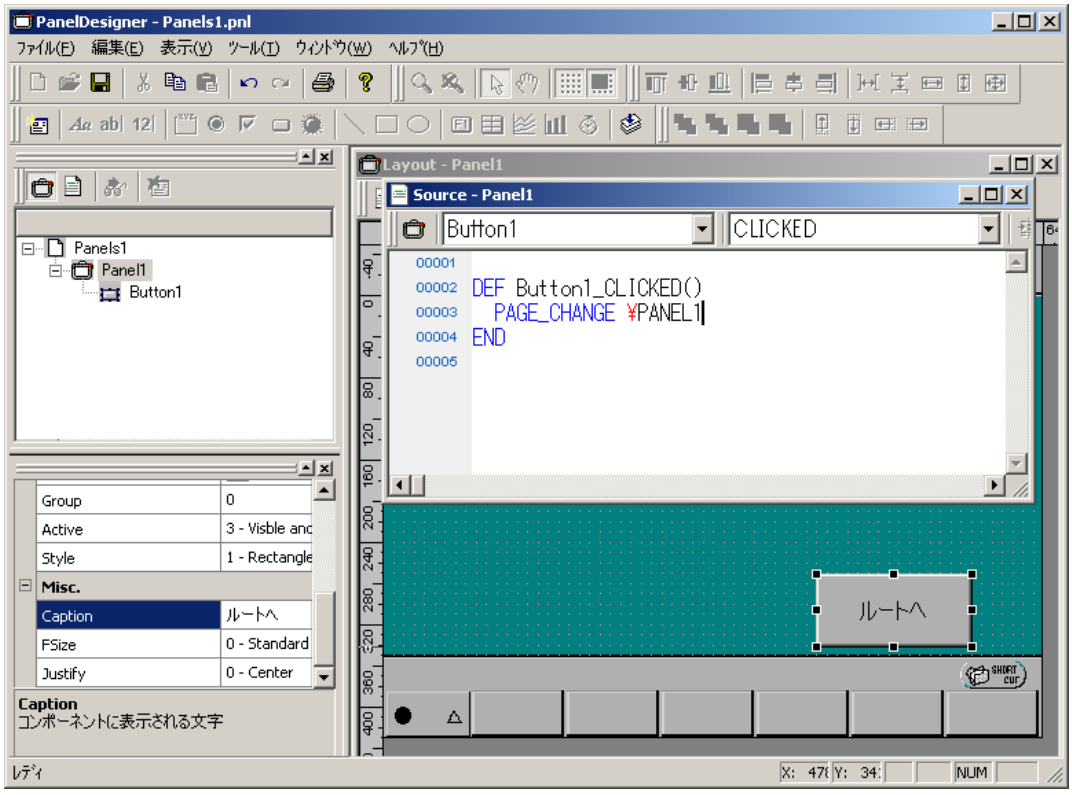
さらにソース編集画面でボタンが押されたときの処理（ページ切り替え）を記述します。

```
PAGE_CHANGE FOLDER1.PANEL1        'FOLDER1 の PANEL1 の画面へ相対移動 PAGE_CHANGE
FOLDER2.PANEL1                    'FOLDER2 の PANEL1 の画面へ相対移動 PAGE_CHANGE
FOLDER1.FOLDER2.PANEL1            'FOLDER1.FOLDER2 の PANEL1 の画面へ相対移動
PAGE_CHANGE ¥PANEL3               'ルートフォルダのパネル名 PANEL1 の画面へ絶対移動
```



STEP 2
(続き)





これで別フォルダの３画面間の移動を行うことが可能となります。

2.5　流れ制御

操作盤制御言語では流れ制御文として条件分岐を行うIF～END IF、IF THEN ELSE、繰り返しを行うFOR～NEXTが使用できます。
それぞれを使用した例を以下に示します。

2.5.1　条件分岐

■IF～END IFの例

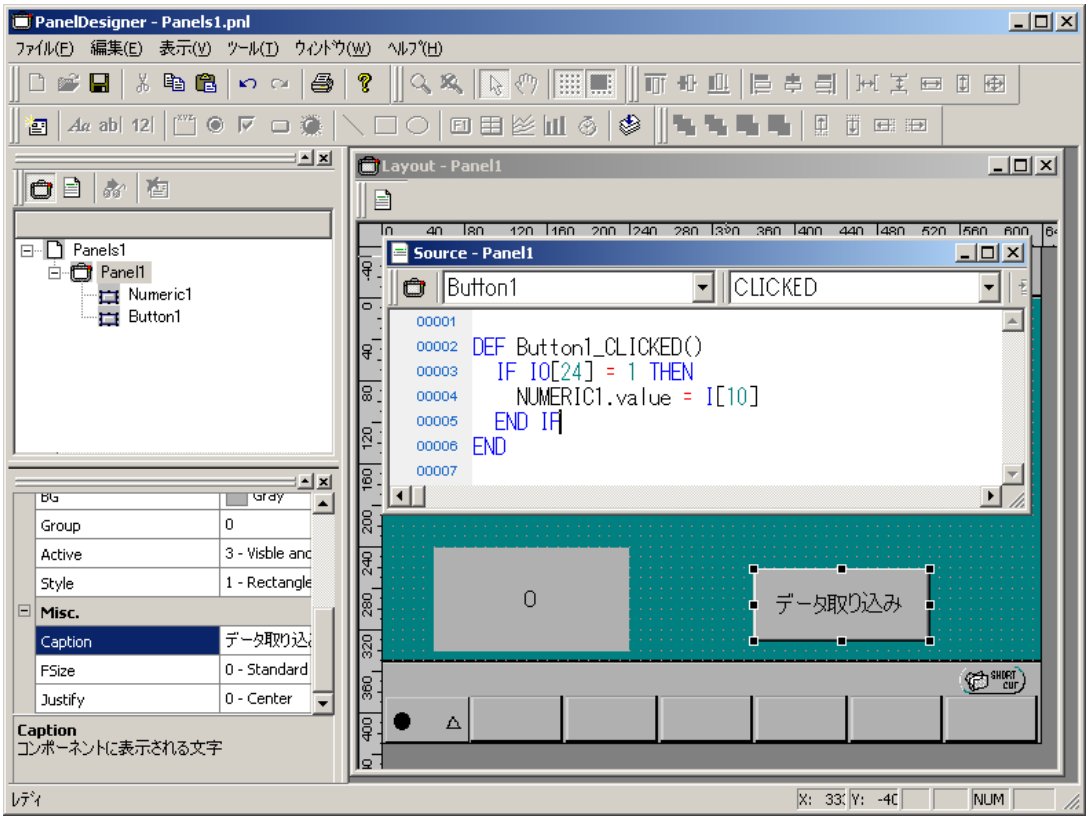
ここではIF文を用い、グローバル変数の値の数値入力ボックスへの取り込みをI/O状態により選択的に行う例を示します。

STEP 1

操作盤エディタで数値入力ボックスと取り込みアクションを指定するボタンを配置します。

STEP 2

次にソース編集画面で、ボタンが押されたときにI/O 24番がONならばグローバルI型変数1/0番を取り込み、OFFならば行わないという処理を記述します。



これで選択的にデータを取り込む機能を実現できます。

■SELECT～CASEの例

ここではSELECT～CASE文を用いて選択的に処理を行う例として、数値入力ボックスの値に応じて起動させるプログラムを選択させます。

STEP 1

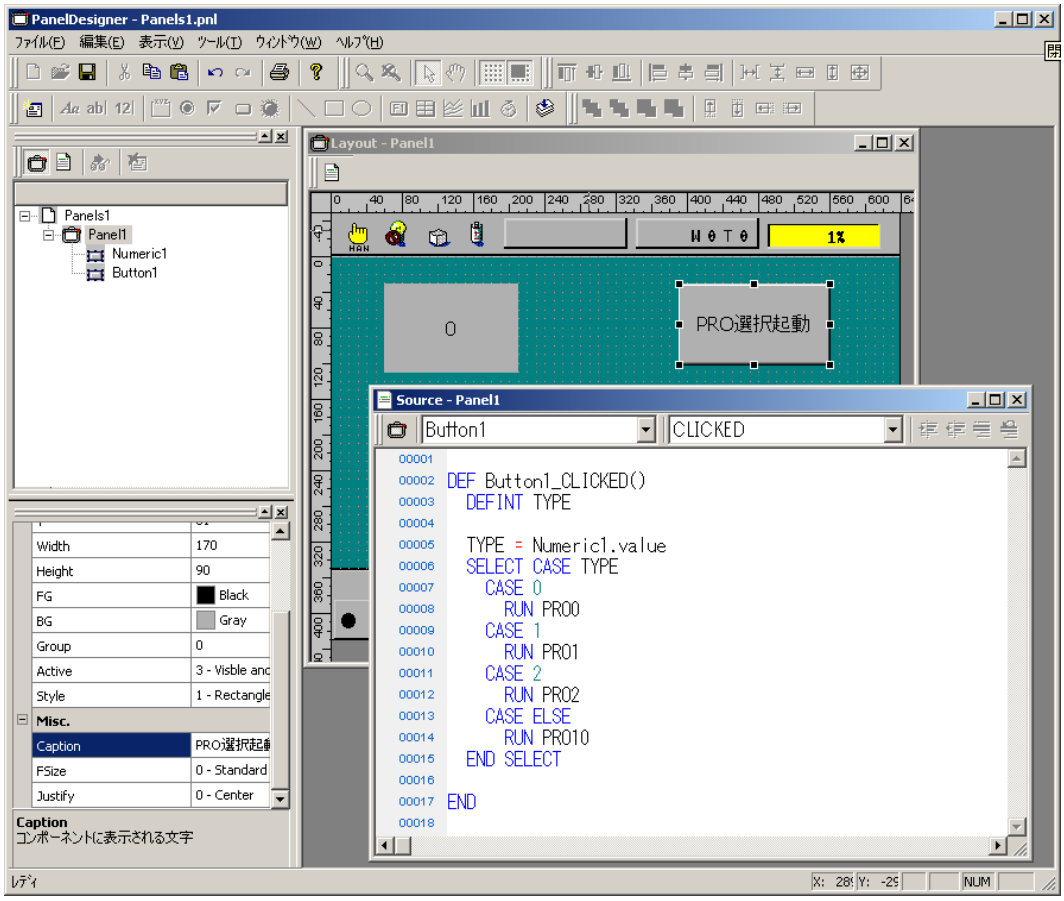
操作盤エディタで押されたときにプログラムを選択的に起動させるボタンと数値入力ボックスを配置します。

STEP 2

次にソース編集画面でボタンを押したときの処理を記述します。

```
DEFINT TYPE

 TYPE = Numeric1.value
 SELECT CASE TYPE
   CASE 0
     RUN PRO0
   CASE 1
     RUN PRO1
   CASE 2
     RUN PRO2
   CASE ELSE
     RUN PRO10
 END SELECT
```



これにより選択的にプログラムを起動する機能が実現できます。

2.5.2 繰り返し

■FOR～NEXT の例

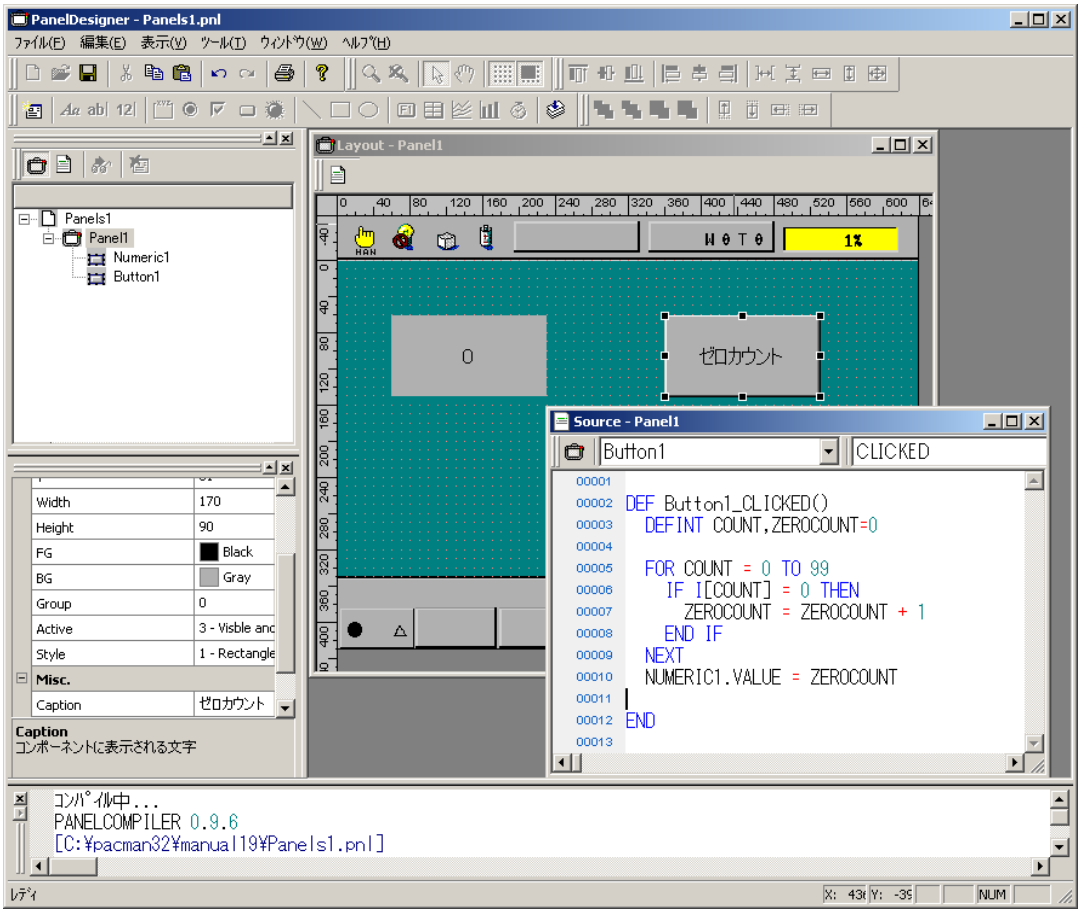
ここではFOR～NEXT文を用いてグローバルI型変数の0番～99番の中に0がいくつあるのかを数えそれを数値入力ボックスに表示する例を示します。

STEP 1 操作盤エディタで押されたときにデータを取得し、0の数を数えるボタンとカウント結果を表示する数値入力ボックスを配置します。

STEP 2 次にソース編集画面でボタンを押したときの処理を記述します。

```
DEFINT COUNT,ZEROCOUNT=0

FOR COUNT = 0 TO 99
  IF I[COUNT] = 0 THEN
    ZEROCOUNT = ZEROCOUNT + 1
  END IF
NEXT
NUMERIC1.VALUE = ZEROCOUNT
```



これによりグローバル変数中の0の数をカウントする機能を実現できます。

2.6 ローカル変数

操作盤制御言語ではローカル変数として整数型、単精度実数型、倍精度実数型、文字列型、I/O型が使用できます。
ローカル変数は宣言を行うそれぞれの部品イベントプログラム内でのみアクセス可能です。

```
DEF Button1_CLICKED()
  DEF Button1_CLICKED()
  DEFINT COUNT,ZEROCOUNT=0

  FOR COUNT = 0 TO 99
    IF I[COUNT] = 0 THEN
      ZEROCOUNT = ZEROCOUNT + 1
    END IF
  NEXT
  NUMERIC1.VALUE = ZEROCOUNT
END
```

COUNT, ZEROCOUNT の有効範囲

■ローカル変数の使用例

ボタンが押されたときにグローバル変数をローカル変数に取り込み、データ処理を行いグローバル変数にデータを書き戻す機能を示します。

STEP 1 操作盤エディタで押されたときにデータ処理を行うボタンを配置します。

STEP 2

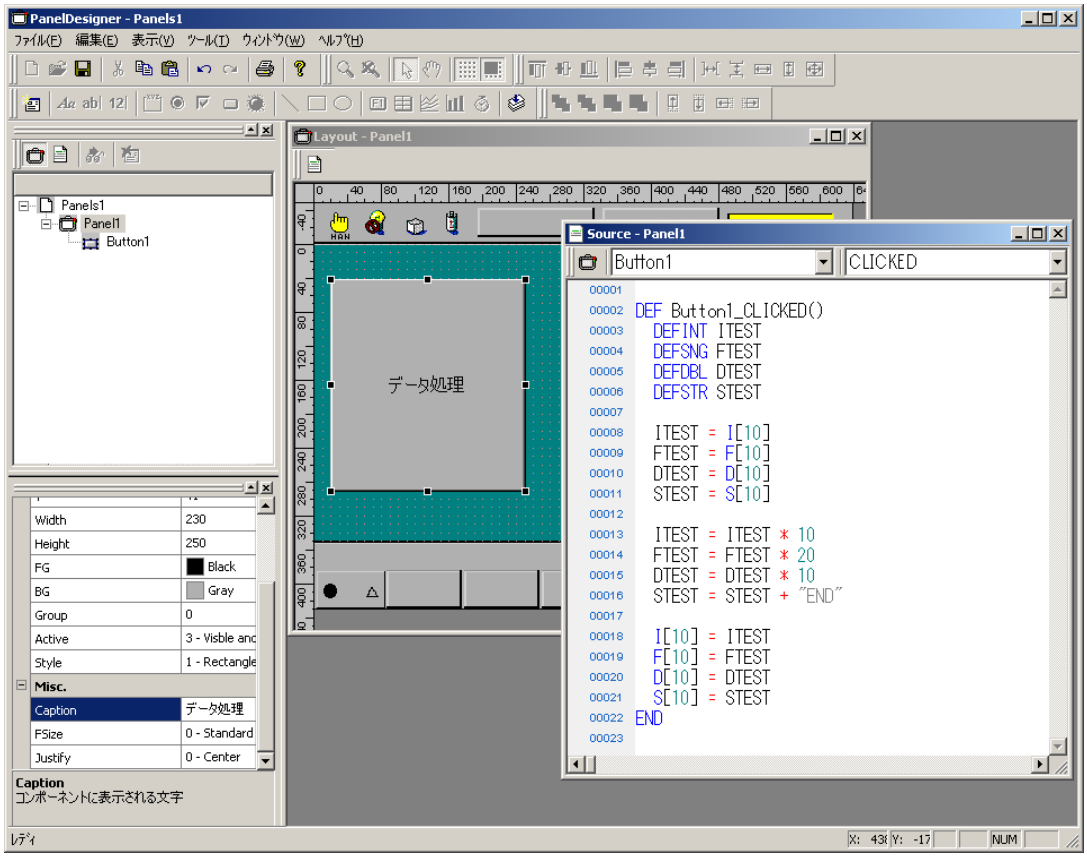
ソース編集画面でローカル整数変数にグローバルI型10番のデータを取り込み、10倍しグローバルI型10番に書き戻す処理、ローカル単精度実数変数にグローバルF型10番のデータを取り込み、20倍しグローバルF型10番に書き戻す処理、ローカル倍精度実数変数にグローバルD型10番のデータを取り込み、30倍しグローバルD型10番に書き戻す処理，ローカル文字列変数にグローバルS型10番のデータを取り込み、文字列“end”を連結しグローバルS型10番に書き戻す処理を記述します。

```
DEFINT ITEST
DEFSNG FTEST
DEFDBL DTEST
DEFSTR STEST

ITEST = I[10]
FTEST = F[10]
DTEST = D[10]
STEST = S[10]

ITEST = ITEST * 10
FTEST = FTEST * 20
DTEST = DTEST * 10
STEST = STEST + “END”

I[10] = ITEST
F[10] = FTEST
D[10] = DTEST
S[10] = STEST
```



これでローカル変数を用いてデータ処理を行う機能を実現できます。

第3章 操作盤制御言語の構成要素

3.1 言語要素

操作盤制御言語を構成する要素には、以下のものがあります。

●識別子……構成要素を識別するための名前
●変　数……データを一時的に記憶するもの
●定　数……一定した値を持つデータ
●演算子……2つの値の間で計算をする記号
●式……値を求めるための構成要素の組み合わせ
●コマンド……処理を実行するようPAC言語に組み込まれた命令

3.2 名前

操作盤記述言語には、プログラムの中にあるさまざまな要素を識別するための規則があります。この章では、この規則について説明します。なお、コマンド、変数を表す名前は、次の規則に従います。

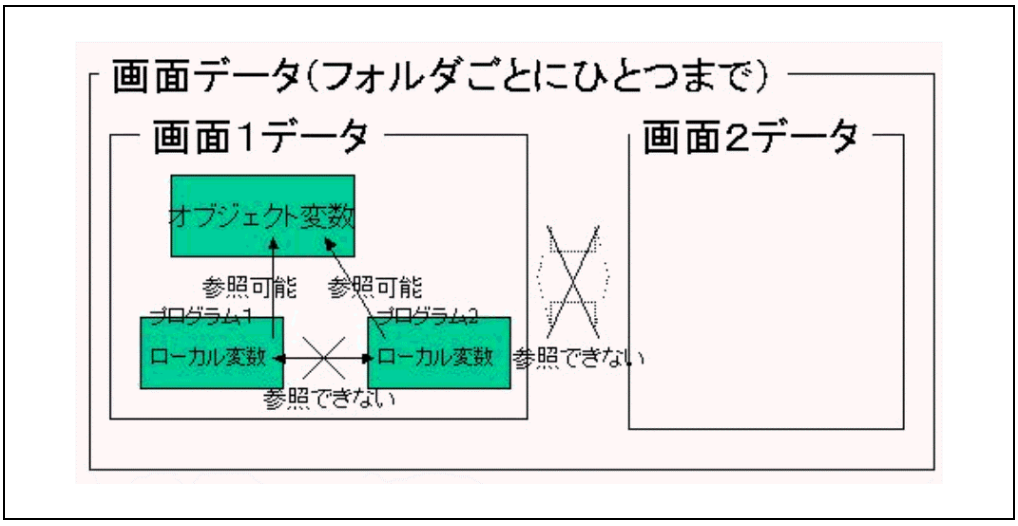
●名前は、文字（半角アルファベット。大文字と小文字の区別はしない）または、決められた記号で始まらなければなりません。
●名前には、文字、数字、アンダースコアが使用できます。
名前の先頭文字はアルファベットでなければなりません。
●ピリオド、スラッシュ、バックスラッシュ、スペース、コロン、セミコロン、シングルクォート、ダブルクォーテーション、アスタリスクは使用できません。
●+、-、*、/、(、)などの、演算子として使われる文字は使用できません。
●名前を他の語と見分けるために、スペース文字を名前と他の語との間に置きます。
●名前に使用できる文字数は、最大64文字です.

3.3 識別子変数

3.3.1 変数

変数は、プログラム中で使うデータを一時的にしまっておくものです。グローバル変数とローカル変数、操作盤部品オブジェクト変数があります。
グローバル変数は、どの操作盤プログラムからも共通して使用できる変数です。
ローカル変数は、一つのプログラムの中でのみ有効な変数です。同時に実行される他のプログラムの中に、同じ名前のローカル変数があっても、それぞれが属するプログラムの中で働き、相互に影響することはありません。
操作盤部品オブジェクト変数は，同一操作盤ファイルの中でのみ有効な変数です。

部品オブジェクトとプログラムの関係を図に示します。



3.3.2 グローバル変数

グローバル変数の名前は、型を表すアルファベット（I、F、D、S、IO）の後ろにカッコ（[]）を付けて表します。I/O変数だけはアルファベットが2文字（IO）になります。
変数名は、システムで予約されていますから、宣言をすることなく使えます。
グローバル変数には、下記に示す型が使えます。

● I型：整数型　（範囲：-2147483648～+2147483647）
例）　I[1]
● F型：単精度実数型　（-3.402823E+38～3.402823E+38）
例）F[1]
● D型：倍精度実数型　（-1.7976931348623157E+308～1.7976931348623157E+308）
例）D[1]
● S型：文字列型（最大で243文字）
例）S[1]
●IO型：I/O型
例）IO[1]

3.3.3 ローカル変数

ローカル変数には、グローバル変数と同様に、下記の型の変数が使えます。

- I型：整数型（範囲：-2147483648～+2147483647）
- F型：単精度実数型（-3.402823E+38～3.402823E+38）
- D型：倍精度実数型（-1.7976931348623157E+308～1.7976931348623157E+308）
- S型：文字列型（最大で243文字）
- IO型：I/O型

ローカル変数は、型宣言命令を使用して、型宣言をした後に使用できます。
注：PAC言語と異なり，間接参照，後置詞は使用不可能ですのでご注意ください。

3.3.4 操作盤部品オブジェクト変数

操作盤エディタで作成した操作盤画面の部品のプロパティを変更参照するための変数は以下のように使用します。
オブジェクト変数は　オブジェクト名.属性　の形式で変更･取得を行います。
例：
①部品名　Button1　の　キャプションを“ボタン”に変更する。
　Button1.caption =“ボタン”

②部品名　LightButton1　の　状態を　I[1]に取り込む。
　I[1] = LightButton1.state

それぞれの部品の持つ変数ととりうる値，型を以下の表に示します。

操作盤部品オブジェクト変数

部品種類	変数名/型	意味	備考
ボタン アクション CLICKED RELEASED	X/整数	左上 X 座標	描画領域の X 軸基準位置：TP 描画範囲内
	Y/整数	左上 Y 座標	描画領域の Y 軸基準位置：TP 描画範囲内
	Width/整数	幅	･X 位置を基準に左側 (-)，右側 (+) ･Y 位置+高さが Y 位置範囲内にあること
	Height/整数	高さ	･Y 位置を基準に上 (-)，下 (+) ･X 位置+幅が X 位置範囲内にあること
	Fg/整数	前面色	-1: White, 0: Black, 1: Blue, 2: Green, 3: Cyan, 4: Red, 5: Magenta, 6: Brown, 7: LightGray, 8: Gray, 9: LightBlue, 10: LightGreen, 11: LightCyan, 12: LightRed, 13: LightMagenta, 14: Yellow
	Bg/整数	背面色	↑(Fg と同様)
	Group/整数	グループ番号	属するグループ番号
	Active/整数	有効/無効設定	0：非表示・無効、1：表示・無効、2：非表示・有効、 3：表示・有効 注：「3」のみアクション発生
	Style/整数	表示スタイル	0：2D 四角、1：3D 四角、2：2D 丸、3：3D 丸
	Caption/文字列	表示文字列	文字列：半角 80 文字まで
	Fsize/整数	フォントサイズ	0：極小、1：小、2：標準、3：大
	Justify/整数	表示文字位置	0：中心、1：右寄せ、2：左寄せ

部品種類	変数名/型	意味	備考
ラベル	X/整数	左上 X 座標	描画領域の X 軸基準位置：TP 描画範囲内
	Y/整数	左上 Y 座標	描画領域の Y 軸基準位置：TP 描画範囲内
	Width/整数	幅	･X 位置を基準に左側 (-), 右側 (+) ･Y 位置+高さが Y 位置範囲内にあること
	Height/整数	高さ	･Y 位置を基準に上 (-), 下 (+) ･X 位置+幅が X 位置範囲内にあること
	Fg/整数	前面色	-1: White, 0: Black, 1: Blue, 2: Green, 3: Cyan, 4: Red, 5: Magenta, 6: Brown, 7: LightGray, 8: Gray, 9: LightBlue, 10: LightGreen, 11: LightCyan, 12: LightRed, 13: LightMagenta, 14: Yellow
	Bg/整数	背面色	↑(Fg と同様)
	Group/整数	グループ番号	属するグループ番号
	Active/整数	有効/無効設定	0：非表示、1：表示
	Caption/文字列	表示文字列	文字列：半角 80 文字まで
	Fsize/整数	フォントサイズ	0：極小、1：小、2：標準、3：大
	Justify/整数	表示文字位置	0：中心、1：右寄せ、2：左寄せ
ランプ アクション REFRESH	X/整数	左上 X 座標	描画領域の X 軸基準位置：TP 描画範囲内
	Y/整数	左上 Y 座標	描画領域の Y 軸基準位置：TP 描画範囲内
	Width/整数	幅	･X 位置を基準に左側 (-), 右側 (+) ･Y 位置+高さが Y 位置範囲内にあること
	Height/整数	高さ	･Y 位置を基準に上 (-), 下 (+) ･X 位置+幅が X 位置範囲内にあること
	Fg/整数	前面色	-1: White, 0: Black, 1: Blue, 2: Green, 3: Cyan, 4: Red, 5: Magenta, 6: Brown, 7: LightGray, 8: Gray, 9: LightBlue, 10: LightGreen, 11: LightCyan, 12: LightRed, 13: LightMagenta, 14: Yellow
	Bg/整数	背面色	↑(Fg と同様)
	Group/整数	グループ番号	属するグループ番号
	Active/整数	有効/無効設定	0：非表示、1：表示
	Style/整数	表示スタイル	0：2D 四角、1：3D 四角、2：2D 丸、3：3D 丸
	Caption/文字列	表示文字列	文字列：半角 80 文字まで
	Fsize/整数	フォントサイズ	0：極小、1：小、2：標準、3：大
	Justify/整数	表示文字位置	0：中心、1：右寄せ、2：左寄せ 注：スタイルが 2, 3 の場合は無効となります。
	State/整数	状態	0：消灯、1：点灯

部品種類	変数名/型	意味	備考
線	X/整数	左上 X 座標	描画領域の X 軸基準位置：TP 描画範囲内
	Y/整数	左上 Y 座標	描画領域の Y 軸基準位置：TP 描画範囲内
	Width/整数	幅	･X 位置を基準に左側 (-), 右側 (+) ･Y 位置+高さが Y 位置範囲内にあること
	Height/整数	高さ	･Y 位置を基準に上 (-), 下 (+) ･X 位置+幅が X 位置範囲内にあること
	Fg/整数	前面色	-1: White, 0: Black, 1: Blue, 2: Green, 3: Cyan, 4: Red, 5: Magenta, 6: Brown, 7: LightGray, 8: Gray, 9: LightBlue, 10: LightGreen, 11: LightCyan, 12: LightRed, 13: LightMagenta, 14: Yellow
	Bg/整数	背面色	↑(Fg と同様)
	Group/整数	グループ番号	属するグループ番号
	Active/整数	有効/無効設定	0：非表示、1：表示
	Style/整数	表示スタイル	0 ：実線、 1～ 7：DashStyle (dashOnOff)・PenDash (1)～(7)、 8～14：DashStyle (dashDouble)・PenDash (1)～(7)
	Thickness/整数	線の太さ	「0」の時はライン幅「2」
数値入力 ボタン アクション CLICKED RELEASED	X/整数	左上 X 座標	描画領域の X 軸基準位置：TP 描画範囲内
	Y/整数	左上 Y 座標	描画領域の Y 軸基準位置：TP 描画範囲内
	Width/整数	幅	･X 位置を基準に左側 (-), 右側 (+) ･Y 位置+高さが Y 位置範囲内にあること
	Height/整数	高さ	･Y 位置を基準に上 (-), 下 (+) ･X 位置+幅が X 位置範囲内にあること
	Fg/整数	前面色	-1: White, 0: Black, 1: Blue, 2: Green, 3: Cyan, 4: Red, 5: Magenta, 6: Brown, 7: LightGray, 8: Gray, 9: LightBlue, 10: LightGreen, 11: LightCyan, 12: LightRed, 13: LightMagenta, 14: Yellow
	Bg/整数	背面色	↑(Fg と同様)
	Group/整数	グループ番号	属するグループ番号
	Active/整数	有効/無効設定	0：非表示・無効、1：表示・無効、2：非表示・有効、 3：表示・有効 注：「3」のみアクション発生
	Style/整数	表示スタイル	0：2D、1：3D
	Caption/文字列	表示文字列	文字列：半角 80 文字まで
	Fsize/整数	フォントサイズ	0：極小、1：小、2：標準、3：大
	Justify/整数	表示文字位置	0：中心、1：右寄せ、2：左寄せ
	Value/実数	入力値	実数：D 型と同等

部品種類	変数名/型	意味	備考
円（楕円）	X/整数	左上 X 座標	描画領域の X 軸基準位置：TP 描画範囲内
	Y/整数	左上 Y 座標	描画領域の Y 軸基準位置：TP 描画範囲内
	Width/整数	幅	･X 位置を基準に左側 (-), 右側 (+) ･Y 位置+高さが Y 位置範囲内にあること
	Height/整数	高さ	･Y 位置を基準に上 (-), 下 (+) ･X 位置+幅が X 位置範囲内にあること
	Fg/整数	前面色	-1: White, 0: Black, 1: Blue, 2: Green, 3: Cyan, 4: Red, 5: Magenta, 6: Brown, 7: LightGray, 8: Gray, 9: LightBlue, 10: LightGreen, 11: LightCyan, 12: LightRed, 13: LightMagenta, 14: Yellow
	Bg/整数	背面色	↑(Fg と同様)
	Group/整数	グループ番号	属するグループ番号
	Active/整数	有効/無効設定	0：非表示、1：表示
	Style/整数	表示スタイル	0 ：実線、 1～ 7：DashStyle (dashOnOff)・PenDash (1)～(7)、 8～14：DashStyle (dashDouble)・PenDash (1)～(7)
	Thickness/整数	線の太さ	「0」の時は塗りつぶし
四角	X/整数	左上 X 座標	描画領域の X 軸基準位置：TP 描画範囲内
	Y/整数	左上 Y 座標	描画領域の Y 軸基準位置：TP 描画範囲内
	Width/整数	幅	･X 位置を基準に左側 (-), 右側 (+) ･Y 位置+高さが Y 位置範囲内にあること
	Height/整数	高さ	･Y 位置を基準に上 (-), 下 (+) ･X 位置+幅が X 位置範囲内にあること
	Fg/整数	前面色	-1: White, 0: Black, 1: Blue, 2: Green, 3: Cyan, 4: Red, 5: Magenta, 6: Brown, 7: LightGray, 8: Gray, 9: LightBlue, 10: LightGreen, 11: LightCyan, 12: LightRed, 13: LightMagenta, 14: Yellow
	Bg/整数	背面色	↑(Fg と同様)
	Group/整数	グループ番号	属するグループ番号
	Active/整数	有効/無効設定	0：非表示、1：表示
	Style/整数	表示スタイル	0 ：実線、 1～ 7：DashStyle (dashOnOff)・PenDash (1)～(7)、 8～14：DashStyle (dashDouble)・PenDash (1)～(7)
	Thickness/整数	線の太さ	「0」の時は塗りつぶし

部品種類	変数名/型	意味	備考
テキスト ボックス アクション CLICKED RELEASED	X/整数	左上 X 座標	描画領域の X 軸基準位置：TP 描画範囲内
	Y/整数	左上 Y 座標	描画領域の Y 軸基準位置：TP 描画範囲内
	Width/整数	幅	･X 位置を基準に左側 (-), 右側 (+) ･Y 位置+高さが Y 位置範囲内にあること
	Height/整数	高さ	･Y 位置を基準に上 (-), 下 (+) ･X 位置+幅が X 位置範囲内にあること
	Fg/整数	前面色	-1: White, 0: Black, 1: Blue, 2: Green, 3: Cyan, 4: Red, 5: Magenta, 6: Brown, 7: LightGray, 8: Gray, 9: LightBlue, 10: LightGreen, 11: LightCyan, 12: LightRed, 13: LightMagenta, 14: Yellow
	Bg/整数	背面色	↑(Fg と同様)
	Group/整数	グループ番号	属するグループ番号
	Active/整数	有効/無効設定	0：非表示・無効、1：表示・無効、2：非表示・有効、 3：表示・有効 注：「3」のみアクション発生
	Style/整数	表示スタイル	0：2D、1：3D
	Caption/文字列	表示文字列	文字列：半角 80 文字まで
	Fsize/整数	フォントサイズ	0：極小、1：小、2：標準、3：大
	Justify/整数	表示文字位置	0：中心、1：右寄せ、2：左寄せ
	Text/整数	入力テキスト	文字列：S 型と同等 注：2 バイトコードの表示はできますが、ペンダントでの編集ができないため、ペンダントでの編集時に 2 バイトコードは正しく表示されませんのでご注意ください。
グループ	X/整数	左上 X 座標	描画領域の X 軸基準位置：TP 描画範囲内
	Y/整数	左上 Y 座標	描画領域の Y 軸基準位置：TP 描画範囲内
	Width/整数	幅	･X 位置を基準に左側 (-), 右側 (+) ･Y 位置+高さが Y 位置範囲内にあること
	Height/整数	高さ	･Y 位置を基準に上 (-), 下 (+) ･X 位置+幅が X 位置範囲内にあること
	Fg/整数	前面色	-1: White, 0: Black, 1: Blue, 2: Green, 3: Cyan, 4: Red, 5: Magenta, 6: Brown, 7: LightGray, 8: Gray, 9: LightBlue, 10: LightGreen, 11: LightCyan, 12: LightRed, 13: LightMagenta, 14: Yellow
	Bg/整数	背面色	↑(Fg と同様)
	Group/整数	グループ番号	属するグループ番号
	Active/整数	有効/無効設定	0：非表示、1：表示
	Caption/文字列	表示文字列	文字列：半角 80 文字まで
	Fsize/整数	フォントサイズ	0：極小、1：小、2：標準、3：大
	Justify/整数	表示文字位置	0：中心、1：右寄せ、2：左寄せ
	Thickness/整数	線の太さ	「0」の時はライン幅「2」
	Mygroup/整数	管理するグループ	管理するグループ番号

部品種類	変数名/型	意味	備考
チェック ボックス アクション CLICKED RELEASED	X/整数	左上 X 座標	描画領域の X 軸基準位置：TP 描画範囲内
	Y/整数	左上 Y 座標	描画領域の Y 軸基準位置：TP 描画範囲内
	Width/整数	幅	･X 位置を基準に左側 (-), 右側 (+) ･Y 位置+高さが Y 位置範囲内にあること
	Height/整数	高さ	･Y 位置を基準に上 (-), 下 (+) ･X 位置+幅が X 位置範囲内にあること
	Fg/整数	前面色	-1: White, 0: Black, 1: Blue, 2: Green, 3: Cyan, 4: Red, 5: Magenta, 6: Brown, 7: LightGray, 8: Gray, 9: LightBlue, 10: LightGreen, 11: LightCyan, 12: LightRed, 13: LightMagenta, 14: Yellow
	Bg/整数	背面色	↑(Fg と同様)
	Group/整数	グループ番号	属するグループ番号
	Active/整数	有効/無効設定	0：非表示・無効、1：表示・無効、2：非表示・有効、 3：表示・有効 注：「3」のみアクション発生
	Style/整数	表示スタイル	0：2D チェック、1：3D チェック、2：3D ボタン、 3：3D ボタン 注：2D トグルは描画できないので 3D になる。
	Caption/文字列	表示文字列	文字列：半角 80 文字まで 注：Caption に格納されている文字列の長さが長い場合、 ボタンの上に文字列が重なりますのでご注意ください。
	Fsize/整数	フォントサイズ	0：標準、1：小、2：大
	Justify/整数	表示文字位置	0：中心、1：右寄せ、2：左寄せ
	State/整数	状態	0：OFF、1：ON

部品種類	変数名/型	意味	備考
ラジオ ボタン アクション CLICKED RELEASED	X/整数	左上 X 座標	描画領域の X 軸基準位置：TP 描画範囲内
	Y/整数	左上 Y 座標	描画領域の Y 軸基準位置：TP 描画範囲内
	Width/整数	幅	･X 位置を基準に左側 (-), 右側 (+) ･Y 位置+高さが Y 位置範囲内にあること
	Height/整数	高さ	･Y 位置を基準に上 (-), 下 (+) ･X 位置+幅が X 位置範囲内にあること
	Fg/整数	前面色	-1: White, 0: Black, 1: Blue, 2: Green, 3: Cyan, 4: Red, 5: Magenta, 6: Brown, 7: LightGray, 8: Gray, 9: LightBlue, 10: LightGreen, 11: LightCyan, 12: LightRed, 13: LightMagenta, 14: Yellow
	Bg/整数	背面色	↑(Fg と同様)
	Group/整数	グループ番号	属するグループ番号
	Active/整数	有効/無効設定	0：非表示・無効、1：表示・無効、2：非表示・有効、 3：表示・有効 注：「3」のみアクション発生
	Style/整数	表示スタイル	0：2D ラジオ、1：3D ラジオ、2：3D ボタン、 3：3D ボタン 注：2D トグルは描画できないので 3D になる。
	Caption/文字列	表示文字列	文字列：半角 80 文字まで 注：Caption に格納されている文字列の長さが長い場合、 ボタンの上に文字列が重なりますのでご注意ください。
	Fsize/整数	フォントサイズ	0：極小、1：小、2：標準、3：大
	Justify/ 整数	表示文字位置	0：中心、1：右寄せ、2：左寄せ
	State/ 整数	状態	0：OFF、1：ON
ファンクションキー 割り付け アクション CLICKED	Caption/文字列	表示文字列	文字列
	Index/整数	ファンクションキー番号	ファンクションキーの番号 1 から 12 まで 注：プログラム中で変更はできますが取得・変更はできませんのでご注意ください。
タイマ アクション TIMER	X/整数	左上 X 座標	描画領域の X 軸基準位置：TP 描画範囲内
	Y/整数	左上 Y 座標	描画領域の Y 軸基準位置：TP 描画範囲内
	Group/整数	グループ番号	属するグループ番号
	Active/整数	有効/無効設定	0：無効、1：有効
	Interval/整数	タイマ間隔	アクションを発生する間隔を設定する。 単位：msec

部品種類	変数名/型	意味	備考
照光式 ボタン アクション CLICKED RELEASED REFRESH	X/整数	左上 X 座標	描画領域の X 軸基準位置：TP 描画範囲内
	Y/整数	左上 Y 座標	描画領域の Y 軸基準位置：TP 描画範囲内
	Width/整数	幅	･X 位置を基準に左側 (-), 右側 (+) ･Y 位置+高さが Y 位置範囲内にあること
	Height/整数	高さ	･Y 位置を基準に上 (-), 下 (+) ･X 位置+幅が X 位置範囲内にあること
	Fg/整数	前面色	-1: White, 0: Black, 1: Blue, 2: Green, 3: Cyan, 4: Red, 5: Magenta, 6: Brown, 7: LightGray, 8: Gray, 9: LightBlue, 10: LightGreen, 11: LightCyan, 12: LightRed, 13: LightMagenta, 14: Yellow
	Bg/整数	背面色	↑(Fg と同様)
	Group/整数	グループ番号	属するグループ番号
	Active/整数	有効/無効設定	0：非表示・無効、1：表示・無効、2：非表示・有効、 3：表示・有効 注：「3」のみ CLICKED, RELEASED アクション発生
	Style/整数	表示スタイル	0：2D 四角、1：3D 四角、2：2D 丸、3：3D 丸
	Caption/文字列	表示文字列	文字列：半角 80 文字まで
	Fsize/整数	フォントサイズ	0：極小、1：小、2：標準、3：大
	Justify/整数	表示文字位置	０：中心、１：右寄せ、２：左寄せ 注：スタイルが 2, 3 の場合は無効となります。
	State/整数	状態	0：消灯、1：点灯

3.3.5 フォルダ変数

操作盤プログラムからはフォルダ変数の参照・変更を行うことができます。
ただし,あらかじめ同一フォルダ内のPACプログラムでフォルダ変数が宣言されている必要があります。

フォルダ変数を使用するためにはEXTERN宣言をつけて型宣言を行います。

```
例：  EXTERN  DEFINT  AAA      ‘変数名AAAのフォルダ変数を宣言する。
```

フォルダ変数の値参照，値変更は通常の変数と同様に行います。

```
例：  AAA  =  LightButton1.state   ‘ランプLightButton1の状態を
                                   フォルダ変数AAAに取り込む。
      I[2]  =  AAA                 ‘フォルダ変数AAAの値をグローバル
                                   変数I[2]に格納する。
```

3.4 プログラム

操作盤プログラムは次の形式で行わせたい動作のみ記述します。

```
DEF　オブジェクト名__アクション
 行わせたい動作
END
```

操作盤エディタにおいてオブジェクト名とアクションを選択すると，
「DEF　オブジェクト名__アクション」と「END」の部分が自動生成されますので，内部のみ記述を行ってください。（2.2.2　部品のアクションを参照）
アクションには下表のものがあり、使用可能なアクションは、部品の種類によって異なります。

アクション名	動作内容
CLICKED	部品が押された
RELEASED	部品が離された
TIMER	インターバル時間が経過した
REFRESH	画面リフレッシュが行われた

また，プログラム内のローカル変数は他のプログラムからは参照できません。

3.5 データ型

操作盤制御言語で扱うデータには、文字列、数値、I/Oの型があります。
以下に、これらのデータ型について説明します。

(1)文字列

文字列（String）型データは、S型とも呼びます。
最大で243文字までの文字列を含むことができます。

(2)数値

数値型データには、次の3種類があります。
I型：整数型　(範囲：-2147483648～+2147483647)
F型：単精度実数型　(-3.402823E+38～3.402823E+38)
D型：倍精度実数型　(-1.79769313486231E+308～1.79769313486231E+308)

(3)I/O（ON/OFF）

I/O型のデータは、I/Oポートの状態（ONまたはOFF）を値として持ちます。

3.6 データ型の変換

異なるデータ型の間で、型を変換することが可能です。

(1)数値

数値データは、次の規則に従って型変換を行なうことができます。

●数値データを、違った型の数値変数に代入すると、数値は変数の型に合わせて変換されます。
●型の違う数値どうしで演算を行なうと、精度の高い方の型に合わせて演算します。
●論理演算では、数値は整数に変換して演算し、結果は整数になります。
●実数が整数に変換される場合には、その実数の値を超えない最大の整数に丸められます。
●倍精度実数が単精度実数に代入された場合、結果は有効数字7桁に丸めたものになります。

3.7 定数

定数は、固定した値を持つ式です。
操作盤制御言語の定数は次のように分類できます。

数値データ	I型：整数型定数 F型：単精度実数型定数 D型：倍精度実数型定数
文字列データ	S型：文字列型定数（最大で243文字）

以下に、各型の定数について説明します。

(1)整数型定数

-2147483648 から +2147483647 の範囲の整数による定数です。
整数型定数には、10進、2進の2種類の表記形式があります。

10進形式

10進数で表現される整数型定数です。
例：32767
-125
+10

2進形式

2進数で表現される整数型定数です。
数値の先頭に「&B」を付け、0または1の並びで表します。
2進形式で整数型定数の数値の範囲を表記すると、
&B0～&B11111111111111111111111111111111
（32ビットの範囲内）となります。

例：&B110
&B0011

(2)単精度実数型定数

7桁の有効桁精度を持つ実数型の定数です。
値の範囲は-3.402823E+38～3.402823E+38です。
単精度実数型定数には、次の２つの表記形式があります。

●Eを使った指数形式
●上記指定がなく、7桁以下の実数
　例：1256.3
　　-9.345E-06

(3)倍精度実数型定数

15桁の有効桁精度を持つ実数型の定数です。
値の範囲は-1.79769313486231E+308～1.79769313486231E+308です。
倍精度実数型定数には、次の表記形式となります。

●7桁を超え15桁以下の実数
　例：1256.325468
　　-9.345E-06

(4)文字列定数

文字列型定数は、文字列を表す定数です。
文字列は、前後をダブルクォーテーション「”」で囲んで表記します。
文字列の長さは、128文字以内でなければなりません。

例：“PAC”

3.8　式と演算子

値を返すものを式といいます。定数や変数のように単独で値を持つ式と、演算子によって結合された複数の要素からなる式があります。 PAC言語のすべてのデータ型の値が、式によって表されます。
以下の部分で説明する演算子を用いると、式の中で演算を行うことができます。

(1)代入演算子

変数への代入は、代入文で、代入演算子「=」を使って行ないます。代入演算子の右側の値が、左側へ代入されます。

(2)算術演算子

算術演算では、下表に示す演算子を使います。
演算は、表に示す優先順位に従って行なわれます。

算術演算子

算術演算子	演算内容	演算の優先順位
^	指数（べき乗）演算	高
-	負符号	↑
*、/	乗算、除算	
MOD	剰余	↓
+、-	加算、減算	低

算術演算子

被除数＼除数	+	0	-
+	+	エラー	+
0	0	エラー	0
-	-	エラー	+

(3)関係演算子

関係演算子は、2つの数値を比較する場合に用います。結果はブール値（真「1」、偽「0」）で得られ、フロー制御文の条件式などで用います。

関係演算子

関係演算子	演算内容
=	等しい
<>	等しくない
<	小さい
>	大きい
<=	小さいか等しい
=.	等しい（近似比較）
>=	大きいか等しい

(4)論理演算子

論理演算子は、ビット演算を行ないます。
整数型以外の数値は、整数型に変換されてから演算されます。

論理演算子

論理演算子	演算内容
NOT	否定
AND	論理積
OR	論理和
XOR	排他的論理和

例：ビット演算
I1 = &B1100　XOR　&B0101
この場合の結果は、&B1001 となります。

(5)文字列演算子

文字列は、文字列演算子「+」によって連結することができます。

例：A =“ABC”+“DEF”　　’Aは“ABCDEF”になります。

(6)算術演算子、論理演算子、関係演算子の優先順位

演算子同士の優先順位は下記を参照してください。

演算子	演算内容	優先順位
^	指数（べき乗）	高
-	負符号	↑
*, /	乗算、除算	
MOD	剰余	
+, -	加算、減算	
NOT	否定	
AND	論理積	
OR	論理和	
XOR	排他的論理和	↓
=, <>, <, >, <=, >=	関係演算子	低

優先順位が同じ場合は式の左から右に評価されます。（ ）で閉じた場合は括弧内の演算が先に処理されます。（優先順位が高くなる）

第4章 操作盤制御言語の文法

4.1 文と行

操作盤記述言語のプログラムは、複数の行から構成されます。
任意の行に、一つの文を記述できます。
1行の長さは、255バイトまでです。
文は、PAC言語による処理を記述する最小の単位であり、一つのコマンドで構成されます。
コマンドは、コマンドの名称と、コマンドに与える情報（パラメータ）から構成されます。

4.2 文字セット

操作盤制御言語で使用できる文字は、英文字（アルファベット）、数字、特殊記号です。英文字は、大文字と小文字の区別はしません。
特殊記号は、算術演算子（ +, -, *, / ）のほかに、次のようなものがあります。

●コンマ（ , ）
　パラメータが並ぶ場合の区切りに使用します。
●シングルクォート（ ’ ）
　REMコマンドの代用として使用します。
●ダブルクォーテーション（ ” ）
　任意の文字列の前後を囲むことで、文字列データの指定に使用します。
●スペース
　命令の名前の前後に必ず必要です。

4.3 予約語

コマンドの名前や演算子など、操作盤制御言語が処理を行なう上で、使用方法を決めているものを予約語といいます。

下記の「操作盤予約語一覧」に示す予約語は変数名，パネル名として使用することはできません。

＜操作盤予約語一覧＞

if, then, else, elseif, while, do, return, print, add_widget, msgbox, page_change, set, reset, run, kill, suspend, suspendall, killall, caption, fg, bg, timeout, defint, defsng, defdbl, defstr, defio, in, out, break, continue, var, def, pend, for, refresh, extern, begin, end, wend, next, endif, status, str$, continuerun, io, i, f, d, s, sysstate, curoptmode, time$, date$, timer, select, case, is, to

4.4 宣言文

変数や定数、関数など、名前や型を決めておく必要のあるものは、プログラム中で使用する前に、宣言文で定義を宣言します。
宣言文は、大きく分けて、次の種類があります。

(1)型宣言

変数および定数の型を宣言します。

型宣言命令

次の型宣言のコマンドを使って、変数の型を宣言することができます。

型宣言命令

種　類	コマンド	使用例
整数型	DEFINT	DEFINT AA,AB
単精度実数型	DEFSNG	DEFSNG BA,BB
倍精度実数型	DEFDBL	DEFDBL CA,CB
文字列型	DEFSTR	DEFSTR DA,DB

これらのコマンドでは、型を宣言するのと同時に、変数の初期化をすることができます。

使用例：

```
DEFINT AA = 1      'AAを整数型とし1を代入します。
DEFSNG BB(10)      'BBを要素数10の単精度実数型にします。
```

(2)配列宣言

配列宣言のための宣言文では､型宣言命令を使用し､要素数を指定することで､I/O変数以外のすべての型の配列を作ることができます。
ただし、配列の初期化は、宣言時にはできません。
配列の添字は0以上でなくてはなりません。配列の次元は3次元までです。
配列の要素の総数は32767を上限とします。

配列宣言の例：

```
DEFINT CC(3,3,3)   'CCを整数型の3次元配列とします。
```

(3)I/O 変数宣言

変数名と特定のI/Oポートを対応付けます。

I/O変数宣言

種　類	コマンド	使用例
I/O変数宣言	DEFIO	DEFIO PORT = BYTE, 104

4.5　代入文

代入文は、各型の変数に値を設定します。
数値代入文、文字列代入文の2つに大別されます。

(1)数値代入文

数値代入文では、数値変数に値を代入します。

例： D[2] = 3.14　　'D[2]に3.14を代入します。

(2)文字列代入文

文字列代入文は、文字列型変数に文字列を代入します。

例：S[2] = “DENSO” 'S[2]に“DENSO”を代入します。

4.6　フロー制御文

プログラムの各文の実行順序を制御するために、フロー制御文を使います。
フロー制御は、条件分岐、選択、繰り返しの３つに大別されます。

(1)条件分岐

IF～THEN～ELSE文またはIF～END IF文を使い、指定した条件が成立するかどうかにより、分岐先が決められます。
IFの直後に記述した関係式の値が真（TRUE(1)）であればTHEN以降を、そうでなければELSE以降を実行します。

(2)選択

指定した式の値によって、実行する処理を選びます。

SELECT CASE文では、SELECT行のCASE以降に算術式を置き、この算術式を満たす値を持つCASE行以降を、次のCASE行またはEND SELECT行まで実行します。いずれのCASE行も等しくない場合は、CASE ELSE以降をEND SELECT行まで実行します。

(3)繰り返し

指定した条件によって、繰り返すかどうかを制御します。

FOR～NEXT文では、FOR行のFOR以降に繰り返し条件を置き、この条件が満たされるまで、対応するNEXT行までの間の処理を繰り返します。

4.7 入出力制御文

入出力制御文には、DI/DO制御文、操作盤制御文の3つがあります。

(1)DI/DO 制御文

DI/DO制御文は、I/Oポートの入出力を制御します。

DI/DO制御コマンド

動作の種類	コマンド
データ読み込み	IN
データ書き込み	OUT
I/Oポート ON	SET
I/Oポート OFF	RESET

(2)ティーチングペンダント制御文

ティーチングペンダント制御文は、ティーチングペンダントの入出力の設定をします。

ティーチングペンダント制御コマンド

動作の種類	コマンド
メッセージ画面出力	MSGBOX
ページ表示	PAGE_CHANGE
画面の再描画	REFRESH

4.8 マルチタスク制御文

マルチタスク制御文には、タスク制御文があります。

(1)タスク制御文

タスク制御文は、そのタスク制御文が属するタスク以外のタスクを制御します。

タスク制御コマンド

動作の種類	コマンド
タスクの生成・起動	RUN
タスクの中断	SUSPEND
タスクの削除	KILL
全タスクの中断	SUSPENDALL
全タスクの削除	KILLALL
コンティニュー起動	CONTINUERUN

4.9 関数

文字列を次のコマンドにより制御します。

文字列関数

動作の種類	関数
文字列への変換	STR$
文字コードの入力	CHR$

4.10 システム情報

システム情報は、次に示すコマンドによって得られます。

システム情報コマンド

動作の種類	コマンド
プログラムの状態を取得	STATUS
動作モードの取得	CUROPTMODE
コントローラステータスの取得	SYSSTATE

4.11 プリ･プロセッサ

プリプロセッサ文は、プログラムを実行形式に翻訳（コンパイル）する際に、文字列の置き換え、またはファイルの取り込みを制御します。

プリプロセッサコマンド

動作の種類	コマンド
定数、マクロ名を文字列に置換	#define
ファイルの取り込み	#include

第5章 コマンドリファレンス

5.1 操作盤制御コマンド一覧

機能区分		コマンド	機能	4軸	6軸
宣言文					
ローカル変数	I型	DEFINT	整数型変数を宣言します。整数の範囲は -2147483648 ～ 2147483647 です。	◎	◎
	F型	DEFSNG	単精度実数型変数を宣言します。単精度実数の範囲は -3.4E-38 ～ 3.4E+38 です。	◎	◎
	D型	DEFDBL	倍精度実数型変数を宣言します。倍精度実数の範囲は -1.7D+308 ～ 1.7D+308 です。	◎	◎
	S型	DEFSTR	文字列型変数を宣言します。文字列の長さは、243 文字までです。	◎	◎
	I/O型	DEFIO	入出力ポートに対応するI/O変数を宣言します。	◎	◎
フロー制御文					
繰り返し		FOR～NEXT	FOR～NEXT までの区間中にある一連の命令を繰り返して実行します。	◎	◎
条件分岐		IF～END IF	IF～END IF 間の条件式の条件判断を行います。	◎	◎
		SELECT CASE	複数条件判断を行います。	◎	◎
入出力制御文					
I/Oポート		IN	I/O変数で示されるI/Oポートからデータを読込ます。	◎	◎
		OUT	I/O変数で示されるI/Oポートにデータを出力します。	◎	◎
		SET	I/Oポートを ON にします。	◎	◎
		RESET	I/Oポートを OFF にします。	◎	◎
TP操作盤		MSGBOX	メッセージボックスを表示します。	◎	◎
		PAGE_CHANGE	ページの表示を行います。	◎	◎
		REFRESH	画面の再描画を行います。	◎	◎
マルチタスク制御文					
タスク制御		RUN	別プログラムを並列起動します。	◎	◎
		KILL	タスクを強制終了します。	◎	◎
		SUSPEND	タスクを一時停止します。	◎	◎
		SUSPENDALL	特権タスク以外の全てのプログラムを停止します。	◎	◎
		KILLALL	特権タスク以外の全てのタスクを強制終了します。	◎	◎
定数					
組み込み定数		OFF	OFF(0)の値を与えます。	◎	◎
		ON	ON(1)の値を与えます。	◎	◎
		PI	πの値を与えます。	◎	◎
		FALSE	ブール値の偽(0)の値を与えます。	◎	◎
		TRUE	ブール値の真(1)の値を与えます。	◎	◎
関数					
文字列		STR$	文字列への変換を行います。	◎	◎
		CHR$	文字コードを入力します。	◎	◎
時刻／日付制御					
時刻／日付		DATE$	現在の日付を得ます。	◎	◎
		TIME$	現在の時刻を得ます。	◎	◎
		TIMER	経過時間を得ます。	◎	◎
システム情報					
動作モード		STATUS	プログラムの状態を得ます。	◎	◎
		CUROPTMODE	動作モードを取得します	◎	◎
		SYSSTATE	コントローラのステータスを取得します。	◎	◎
プリ･プロセッサ					
記号定数・マクロ定義		#define	プログラム中の指定した定数またはマクロ名を、指定文字列で置替えます。	◎	◎
ファイル取り込み		#include	プリプロセッサプログラムを取り込みます。	◎	◎

5.2 宣言文

DEFINT (ステートメント)

機能　I 型変数（整数型変数）を宣言します。整数の範囲は - 2147483648 ～ 2147483647 です。

書式　DEFINT <変数名>[=<定数>][,<変数名>[=<定数>]…]

説明　<変数名>で指定した変数を整数型の変数として宣言します。<変数名>に続けて等号と定数を書くことにより、宣言と同時に初期化を行なえます。
“,”で区切ることにより、一度に複数の変数名を宣言できます。

関連項目　DEFDBL、DEFSNG、DEFSTR

用例

```
DEFINT lix, liy, liz          'lix, liy, liz をＩ型変数として宣言します。
DEFINT lix = 1                'lix をＩ型変数として宣言し､初期値=1 を与えます。
```

DEFSNG (ステートメント)

機能　F 型変数（単精度実数型変数）を宣言します。単精度実数の範囲は -3.4E-38 ～ 3.4E+38 です。

書式　DEFSNG <変数名>[=<定数>][,<変数名>[=<定数>]…]

説明　<変数名>で指定した変数を単精度実数型として宣言します。<変数名>に続けて等号と定数を書くことにより、宣言と同時に初期化を行なえます。
“,”で区切ることにより、一度に複数の変数名を指定できます。

関連項目　DEFDBL、DEFINT、DEFSTR

用例

```
DEFSNG lfx, lfy, lfz          'lfx, lfy, lfz をＦ型変数として宣言します。
DEFSNG lfx = 1.0              'lfx をＦ型変数として宣言し､初期値=1.0 を
                              '与えます。
```

DEFDBL（ステートメント）

機能　D 型変数（倍精度実数型変数）を宣言します。倍精度実数の範囲は -1.7D+308 ～ 1.7D+308 です。

書式　DEFDBL <変数名>[=<定数>][,<変数名>[=<定数>]…]

説明　<変数名>で指定した変数を倍精度実数型として宣言します。<変数名>に続けて等号と定数を書くことにより、宣言と同時に初期化を行なえます。
“,”で区切ることにより、一度に複数の変数名を指定できます。

関連項目　DEFINT、DEFSNG、DEFSTR

用例

```
DEFDBL ldx, ldy, ldz        'ldx, ldy, ldz を D 型変数として宣言します。
DEFDBL ldx = 1.0            'ldx を D 型変数として宣言し、初期値=1.0 を
                            '与えます。
```

DEFSTR（ステートメント）

機能　S 型変数（文字列型変数）を宣言します。文字列の長さは、247 文字までです。

書式　DEFSTR <変数名>[=<定数>][,<変数名>[=<定数>]…]

説明　<変数名>で指定した変数を文字列型として宣言します。<変数名>に続けて等号と定数を書くことにより、宣言と同時に初期化を行なえます。
“,”で区切ることにより、一度に複数の変数名を指定できます。

関連項目　DEFDBL、DEFINT、DEFSNG

用例

```
DEFSTR lsx, lsy, lsz        'lsx, lsy, lsz を S 型変数として宣言します。
DEFSTR lsx = "DENSO"        'lsx を S 型変数として宣言し、初期値="DENSO"を
                              与えます。
```

DEFIO（ステートメント）

機能

入出力ポートに対応する IO 変数（I/O 変数）を宣言します。

書式

DEFIO〈変数名〉=〈I/O 変数の型〉,〈ポートアドレス〉[,〈マスク情報〉]

説明

〈変数名〉で指定した変数を I/O 変数として宣言します。

〈I/O 変数の型〉　I/O 変数の型を選択します。I/O 変数の型は、BIT、BYTE、WORD、INTEGER があります。BIT 型は 1 ビット、BYTE 型は 8 ビット、WORD 型は 16 ビット、INTEGER 型は 32 ビットの範囲を指定します。

〈ポートアドレス〉　入出力ポートでの開始番号を指定します。

〈マスク情報〉　入力ポートの場合、入力データとマスク情報の AND をとります。

出力ポートの場合、出力データとマスク情報の AND をとり出力しますが、マスクがセットされていないビットの出力状態は変化しません。

関連項目

IN、OUT、SET、RESET

用例

```
DEFIO samp1 = BIT, 1                    'samp1 をポート 1 から始まる BIT 型 I/O 変数として宣
                                        '言します。samp1 の返値は、ポート 1 の状態を表す、0
                                        'または 1 の 1 ビット整数となります。
DEFIO samp2 = BYTE, 10, &B00010000      'samp2 をポート 10 から始まる BYTE 型 I/O 変数のマス
                                        'ク情報を付けて宣言します。samp2 の返値は、ポート 10
                                        'の状態を表す、0 または 16 の 8 ビット整数となります。
DEFIO samp3 = WORD, 15                  'samp3 をポート 15 から始まる WORD 型 I/O 変数として
                                        '宣言します。samp3 の返値は、ポート 15 から 30 の状態
                                        'を表す、0～&Hffff の 16 ビット整数となります。
DEFIO samp4 = INTEGER, 1                'samp4 をポート 1 から始まる INTEGER 型 I/O 変数とし
                                        'て宣言します。samp4 の返値は、ポート 1 から 32 の状
                                        '態を表す、0～&Hffffffff の 32 ビット整数となります。
```

注意事項

WORD,INTEGER の場合、最上位ビットにあたるポートは符号ビットとなります。

数値の範囲と、最上位ビットにあたるポート番号は下記のとおりとなります。

WORD	数値範囲　：-32768 ～ 32767 最上位ビットポート番号　：開始ポートアドレス + 15
INTEGER	数値範囲　：-2147483648 ～ 2147483647 最上位ビットポート番号　：開始ポートアドレス + 31

5.3　フロー制御文

FOR～NEXT（ステートメント）

機能　FOR～NEXT までの区間中にある一連の命令を繰り返して実行します。

書式

```
FOR 〈変数名〉 = 〈初期値〉 TO 〈最終値〉 [STEP 〈増分〉]
  :
NEXT [〈変数名〉]
```

説明

FOR～NEXT までの区間中にある一連の命令を、FOR 行で指定した条件に従って繰り返して実行します。

〈初期値〉は、〈変数名〉で指定した変数の初期値を設定します。

〈最終値〉は、〈変数名〉で指定した変数の最終値を設定します。

〈増分〉は、初期値と最終値との間の増分を設定します。STEP を省略すると増分は 1 とみなされます。また，〈増分〉には負の値は指定できません.

１つの FOR～NEXT の中にはもう１つの FOR～NEXT を置くことができます(入れ子構造、ネスト構造等と呼ぶ)。

この場合、それぞれの〈変数名〉には別のものを使わなければなりません。

また、このとき、１つの FOR～NEXT は完全に他の FOR～NEXT の内部になければなりません。

用例

```
DEFINT li1
 FOR li1 = 1 TO 5          'FOR～NEXT の処理を 5 回繰り返します。

NEXT                       '繰り返します。
```

IF～END IF（ステートメント）

機能

IF～END IF 間の条件式の条件判断を行ないます。

書式

```
IF〈条件式〉THEN
   :
[ELSEIF〈条件式〉THEN]
   :
[ELSE]
   :
END IF
```

説明

〈条件式〉の条件によってプログラムの実行を制御します。

IF 文の〈条件式〉が真(0 以外)ならば IF～ELSEIF 文の間の文を実行し、〈条件式〉が偽(0)ならば ELSEIF 文の〈条件式〉を判断します。同様に、ELSEIF～ELSE、ELSE～END IF と実行していきます。

関連項目

IF～THEN～ELSE

用例

```
DIM li1 As Integer
IF li1 = 0 THEN              'li1 が 0 の場合。
PAGE_CHANGE  PANEL1          'ページ名 PANEL1 に移動します。
ELSEIF li1 = 1 THEN          'li1 が 1 の場合。
PAGE_CHANGE  PANEL2          'ページ名 PANEL2 に移動します。
ELSEIF li1 = 2 THEN          'li1 が 2 の場合。
PAGE_CHANGE  PANEL3          'ページ名 PANEL3 に移動します。
ELSE                         'li1 がその他の場合。
PAGE_CHANGE  PANEL4          'ページ名 PANEL4 に移動します。
END IF                       'IF 文の終了を宣言します。
```

SELECT CASE（ステートメント）

機能

複数条件判断を行ないます。

書式

```
SELECT CASE 〈式〉
   CASE 〈項目〉[,〈項目〉...]
      :
   [CASE ELSE]
END SELECT
```

説明

〈式〉の値が CASE 文の〈項目〉に一致する場合、その CASE 以降の一連の命令を実行します。
〈式〉には、数式または文字列を指定できます。
〈項目〉では、変数、定数、式および条件式を指定することができます。
条件式は、次のように指定できます。

・〈数式 1〉 TO 〈数式 2〉

〈式〉の結果が〈数式 1〉以上、〈数式 2〉以下であるかどうか調べます。
文字列の場合は使用できません。

・IS 〈比較演算子〉〈数式〉

〈式〉の結果と〈数式〉の値を比較します。

文字列の場合、〈比較演算子〉は“=”のみとなります。
CASE ELSE 文は、すべての CASE 文が一致しなかった場合に実行されます。
また、CASE ELSE 文は、END SELECT 文の前に置かなければなりません。

関連項目

IF～END IF

用例

```
SELECT CASE Index                ’Index の値が CASE 文の値と一致した場合、
                                   そのコマンドが実行されます。
  CASE 0                         ’Index が 0 の場合。
    Button1.caption = “0”
  CASE 1                         ’Index が 1 の場合。
    Button1.caption = “1”
  CASE 2                         ’Index が 2 の場合。
    Button1.caption = “2”
  CASE 3                         ’Index が 3 の場合。
    Button1.caption = “3”
  CASE 4                         ’Index が 4 の場合。
    Button1.caption = “4”
  CASE 5                         ’Index が 5 の場合。
    Button1.caption = “5”
  CASE 6 TO 8                    ’Index が 6～8 の場合。
    Button1.caption = “6-8”
  CASE IS≧9                      ’Index が 9 以上の場合。
    Button1.caption = “9-”
END SELECT                       ’複数条件判断文の終了を宣言します。
```

5.4 入出力制御文

IN (ステートメント)

機能　I/O 変数で示される I/O ポートからデータを読み込みます。

書式　IN 〈算術変数名〉 = 〈I/O 変数〉

説明　〈I/O 変数〉で示される I/O ポートのデータを〈算術変数名〉で指定した変数に代入します。

〈I/O 変数〉は、DEFIO 文で宣言された変数か、または I/O 型変数です。

関連項目　OUT、DEFIO

用例

```
DEFINT Li1, Li2
DEFIO samp1 = INTEGER, 220  'samp1 をポート 220 から始まる INTEGER 型 I/O 変数と
                            'して宣言します。
IN Li1 = samp1              'samp1 のデータを Li1 へ代入します。
IN Li2 = IO[240]            'ポート 240 のデータを Li2 へ代入します。
OUT samp1 = Li1             'Li1 のデータを samp1 で宣言したポートから出力しま
                            'す。
OUT IO[240] = Li2           'Li2 のデータをポート 240 から出力します。
```

OUT (ステートメント)

機能　I/O 変数で示される I/O ポートにデータを出力します。

書式　OUT 〈I/O 変数〉 = 〈出力データ〉

説明　〈I/O 変数〉で示されるポートアドレスに〈出力データ〉の値を出力します。

〈I/O 変数〉は、DEFIO 文で宣言された変数か、または I/O 型変数です。

関連項目　IN、DEFIO

用例

```
DEFINT Li1, Li2
DEFIO samp1 = INTEGER, 220  'samp1 をポート 220 から始まる INTEGER 型 I/O 変数と
                            'して宣言します。
IN Li1 = samp1              'samp1 のデータを Li1 へ代入します。
IN Li2 = IO[240]            'ポート 240 のデータを Li2 へ代入します。
OUT samp1 = Li1             'Li1 のデータを samp1 で宣言したポートから出力しま
                            'す。
OUT IO[240] = Li2           'Li2 のデータをポート 240 から出力します。
```

SET（ステートメント）

機能　　I/O ポートを ON にします。

書式　　SET <I/O 変数>

説明　　<I/O 変数>で指定されたポートを ON にします。

関連項目　　RESET、DEFIO

用例

```
SET IO[240]                 'BIT 型ポート 240 を ON にします。
SET IO[SOL1]                'I/O 変数 SOL1 で示されるポートを ON にします。
SET IO[104 TO 110]          'BIT 型ポート 104～110 を ON にします。
IF IO[242] THEN
  RESET IO[240]             'BIT 型ポート 240 を OFF にします。
  RESET IO[SOL1]            'I/O 変数 SOL1 で示されるポートを OFF にします。
  RESET IO[104 TO 110]      'BIT 型ポート 104～110 を OFF にします。
ENDIF
```

RESET（ステートメント）

機能　　I/O ポートを OFF にします。

書式　　RESET <I/O 変数>

説明　　<I/O 変数>で指定されたポートを OFF にします。

関連項目　　SET、DEFIO

用例

```
SET IO[240]                 'BIT 型ポート 240 を ON にします。
SET IO[241], 40             'BIT 型ポート 241 を 40ms 間 ON にします。
SET IO[SOL1]                'I/O 変数 SOL1 で示されるポートを ON にします。
SET IO[104 TO 110]          'BIT 型ポート 104～110 を ON にします。
IF IO[242] THEN
  RESET IO[240]             'BIT 型ポート 240 を OFF にします。
  RESET IO[SOL1]            'I/O 変数 SOL1 で示されるポートを OFF にします。
  RESET IO[104 TO 110]      'BIT 型ポート 104～110 を OFF にします。
ENDIF
```

MSGBOX（ステートメント）

機能　　メッセージをティーチングペンダントのカラー液晶に表示します。

書式　　MSGBOX〈メッセージ文字列〉

説明　　指定したメッセージを、ティーチングペンダントのカラー液晶に表示します。

表示できるメッセージ文字列は最大 60 文字までです。

用例

```
MSGBOX　“Hello World !”
```

注意事項

ポップアップウィンドウを持つ部品（数値入力ボックス，テキストボックス）の CLICKED アクションで MSGBOX を記述しても実行されませんのでご注意ください。

PAGE_CHANGE（ステートメント）

機能　　ページの表示を行ないます。

書式　　PAGE_CHANGE 〈ページ名〉 [,〈フォルダ戻り数〉]

＜ページ名＞　　操作盤に配置されるページの名前

〈フォルダ戻り数〉 今いるフォルダから上に戻るフォルダ数

説明　　ページ名で指定した番号のページを操作盤画面に表示します。

用例

```
page_change panel１              ’指定画面の表示
page_change panel１,2            ’フォルダを 2 個上に上がりそのフォ
                                   の panel1 を表示する
```

5.5 マルチタスク制御文

RUN（ステートメント）

機能　別プログラムを並列起動します。

書式　RUN <プログラム名> [(<引数>[,<引数>…])][,<RUN オプション>]

説明　現在実行中のプログラムから、<プログラム名>に指定したプログラムを並列に起動します。自プログラムを RUN することはできません。

<引数>には、値渡しのみ使用できます。参照渡しを指定した場合は自動的に値渡しになりますが、ローカル配列は使用することができません。

<RUN オプション>には、PRIORITY(もしくは P)、CYCLE(もしくは C)があります。

PRIORITY(もしくは P)

プログラムの優先度を指定します。省略するとデフォルト値 128 として処置します。数値が小さいほど優先度が高くなります。設定範囲は、102～255 の間です。

注: 特権タスクの優先順位は変更できません。

CYCLE(もしくは C)

スイッチング周期（プログラムを繰り返し起動する場合のサイクル毎の間隔）を指定します。単位は、msec です。設定範囲は、1～2147483647 の間です。

引数を伴うプログラムは、CYCLE オプションを付けて起動できません。

用例

```
DEFINT Li1 = 1, Li2 = 2, Li3 = 3
RUN samp1, C = 1000                 'samp1 を並列に (C = 1000) 起動します。
RUN samp2 (Li1)                     'samp2 に Li1 の引数を付けて並列に起動します。
RUN samp3 (Li1, Li2), PRIORITY = 129
                                    'samp3 に Li1,Li2 の引数を付けて並列に(P = 129)
                                     起動します。
RUN samp4 (Li1, Li2), PRIORITY = 150
                                    'samp4 に Li1,Li2 の引数を付けて並列に(P = 150)
                                     起動します。
RUN samp5 (Li1, Li2, Li3), PRIORITY = 120
                                    'samp5 に Li1, Li2, Li3 の引数を付けて並列に
                                    (P = 120) 起動します。
```

KILL（ステートメント）

機能　　タスクを強制終了します。

書式　　KILL〈プログラム名〉

説明　　〈プログラム名〉で指定したタスク(プログラム)の実行を強制終了します。

自プログラムをKILLすることはできません。自プログラムをKILLするとエラーが発生します。自プログラムを強制終了するには、STOP命令を使ってください。

関連項目　　SUSPEND

用例

```
RUN samp1                    'samp1を並列に起動します。
      ・
      ・
KILL samp1                   'samp1を終了します。
```

SUSPEND（ステートメント）

機能　　タスクを一時停止します。

書式　　SUSPEND〈プログラム名〉

説明　　指定したタスクを一時停止します。

自プログラムをSUSPENDすることはできません。

関連項目　　KILL

用例

```
SUSPEND samp1                'samp1のタスクの実行を一時停止します。
```

SUSPENDALL（ステートメント）

機能　特権タスク以外の全てのプログラムを停止します。

書式　SUSPENDALL

説明　特権タスク以外の全タスクを停止させ、停止中のステップから動作させることのできる“コンティ停”状態となり、ロボット運転中出力を OFF します。

関連項目　SUSPEND、KILLALL

用例

```
SUSPENDALL                    ‘全タスク瞬時停止　コンティ停状態へ
```

KILLALL（ステートメント）

機能　特権タスク以外の全てのタスクを強制終了します。（プログラムリセットに相当）

書式　KILLALL

説明　ロボットは現在実行中の特権タスク以外の全タスクを強制終了させ、ロボット運転中出力を OFF します。

関連項目　KILL、SUSPENDALL

用例

```
KILLALL                       ‘全タスク停止　プログラムリセット状態
```

CONTINUERUN（ステートメント）

機能　コンティニュー停止中の全てのタスクを再起動します。

書式　CONTINUERUN

説明　ロボットは現在コンティニュー停止中の全タスクを再起動させます。

関連項目　KILL、SUSPENDALL

用例

```
CONTINUERUN                   ‘全タスク再起動
```

5.6 定数

OFF (組み込み定数)

機能　　OFF(0)の値を与えます。

書式　　OFF

説明　　式に対して、OFF(0)の値を与えます。

関連項目　　ON

用例

```
IF I0 = TRUE THEN           'ブール値の真(1)の値を与えます。
  I1 = ON                   '整数型変数にON(1)の値を代入します。
ELSEIF I0 = FALSE THEN      'ブール値の偽(0)の値を与えます。
  I1 = OFF                  '整数型変数にOFF(0)の値を代入します。
ELSE
  D1 = PI                   '実数型変数にπを代入します。
ENDIF
```

ON (組み込み定数)

機能　　ON(1)の値を与えます。

書式　　ON

説明　　式に対して、ON(1)の値を与えます。

関連項目　　OFF

用例

```
IF I0 = TRUE THEN           'ブール値の真(1)の値を与えます。
  I1 = ON                   '整数型変数にON(1)の値を代入します。
ELSEIF I0 = FALSE THEN      'ブール値の偽(0)の値を与えます。
  I1 = OFF                  '整数型変数にOFF(0)の値を代入します。
ELSE
  D1 = PI                   '実数型変数にπを代入します。
ENDIF
```

PI (組み込み定数)

機能　　πの値を与えます。

書式　　PI

説明　　倍精度型でπの値を返します。

用例

```
IF I0 = TRUE THEN           'ブール値の真(1)の値を与えます。
  I1 = ON                   '整数型変数にON(1)の値を代入します。
ELSEIF I0 = FALSE THEN      'ブール値の偽(0)の値を与えます。
  I1 = OFF                  '整数型変数にOFF(0)の値を代入します。
ELSE
  D1 = PI                   '実数型変数にπを代入します。
ENDIF
```

FALSE (組み込み定数)

機能　　ブール値の偽(0)の値を与えます。

書式　　FALSE

説明　　式に対して、ブール値の偽(0)の値を与えます。

関連項目　　TRUE

用例

```
IF I0 = TRUE THEN           'ブール値の真(1)の値を与えます。
  I1 = ON                   '整数型変数にON(1)の値を代入します。
ELSEIF I0 = FALSE THEN      'ブール値の偽(0)の値を与えます。
  I1 = OFF                  '整数型変数にOFF(0)の値を代入します。
ELSE
  D1 = PI                   '実数型変数にπを代入します。
ENDIF
```

TRUE (組み込み定数)

機能

ブール値の真(1)の値を与えます。

書式

TRUE

説明

式に対して、ブール値の真(1)の値を与えます。

関連項目

FALSE

用例

```
IF I0 = TRUE THEN           'ブール値の真(1)の値を与えます。
  I1 = ON                   '整数型変数にON(1)の値を代入します。
ELSEIF I0 = FALSE THEN      'ブール値の偽(0)の値を与えます。
  I1 = OFF                  '整数型変数にOFF(0)の値を代入します。
ELSE
  D1 = PI                   '実数型変数にπを代入します。
ENDIF
```

5.7 時刻/日付制御

DATE$ (システム変数)

機能　　現在の日付を得ます。

書式　　DATE$

説明　　現在の日付が“yyyy/mm/dd”(年/月/日）の形で格納されています。

関連項目　　TIME$

用例

```
defstr  ls1
ls1 = DATE$                    '現在の日付を ls1 に代入します。
```

TIME$ (システム変数)

機能　　現在の時刻を得ます。

書式　　TIME$

説明　　現在の時刻が“hh:mm:ss”(時：分：秒）の形で格納されています。
時刻表示は 24 時制です。

関連項目　　DATE$

用例

```
defstr ls1
ls1 = TIME$                    '現在の時刻を ls1 に代入します。
```

TIMER (システム変数)

機能　　経過時間を得ます。

書式　　TIMER

説明　　コントローラの電源を入れた時点を基準(0)としてミリ秒で計った経過時間を得ます。

> 注意：経過時間が2147483647ミリ秒を超えた場合、-2147483648ミリ秒を基準に経過時間を返します。

用例

```
DEFINT li1, li2, li3
li1 = TIMER                    '基準時間からの経過時間を li1 に代入します。
```

5.8 文字列関数

STR$ (関数)

機能　　数値から文字列に変換します。

書式　　STR$(<数式>)

説明　　<数式>に指定した値を文字列に変換します。

関連項目　　CHR$

用例

```
DEFSTR ls1, ls2
ls1 = STR$(20)          '20 を文字列に変換し ls1 に代入します。
ls2 = STR$(li1)         'li1 を文字列に変換し ls2 に代入します。
```

CHR$ (関数)

機能　　ASCII コードを文字に変換します。

書式　　CHR$(<数式>)

説明　　<数式>に指定した値のキャラクタコードを持つ文字を得ます。

関連項目　　STR$

用例

```
DEFSTR ls1, ls2
ls1 = CHR$(49)          '49 のキャラクタコードを持つ文字を ls1 に代入します。
ls2 = CHR$(&H4E)        '&H4E のキャラクタコードを持つ文字を ls2 に代入します。
```

5.9　システム情報

CUROPTMODE（ステートメント）

機能　　動作モードを取得します。
1:手動、2:ティーチチェック、3:内部自動、4:外部自動

書式　　CUROPTMODE

説明　　現在の動作モード（手動、ティーチチェック、内部自動、外部自動）の情報を取得します。

用例

```
I[1]　＝　CUROPTMODE　　‘動作モード取得
```

SYSSTATE（ステートメント）

機能　　コントローラのステータスを取得します。

書式　　SYSSTATE

説明　　コントローラステータスを取得します。I/O 割付の設定により有効なデータは変化します。取得可能なデータを下記に示します。

Bit	0	ロボット運転中
	1	ロボット異常
	2	サーボＯＮ中
	3	ロボット初期化完了（I/O 標準モード選択時）/ ロボット電源入り完了（I/O 互換モード選択時）
	4	自動モード
	5	外部モード
	6	バッテリ切れ警告
	7	ロボット警告
	8	コンティニュースタート許可
	9	ＳＳモード
	10	ロボット停止
	11	自動運転イネーブル
	12～15	予約
	16	プログラムスタートリセット（I/O 互換モード選択時）
	17	ＣＡＬ完了（I/O 互換モード選択時）
	18	ティーチング中（I/O 互換モード選択時）
	19	１サイクル完了（I/O 互換モード選択時）
	20～23	予約
	24	コマンド処理完了（I/O 標準モード選択時）
	25～31	予約

用例

```
I[1] = SYSSTATE                    ‘システム状態取得
```

STATUS (関数)

機能　　プログラムの状態を得ます。

書式　　STATUS(<プログラム名>)

説明　　<プログラム名>で指定したプログラムの状態が整数で格納されています。

値	状　　態	
1	running	実行中
2	stopping	停止中
3	suspend	一時停止中
4	delay	遅延中
5	pending	待機中
6	step stopped	ステップ停止中

用例

```
defint  li1
li1 = STATUS(samp1)          'samp1 のプログラムの状態を整数で li1 に代入します。
```

注意事項　　自分自身の STATUS を取得することはできません。

#define (プリプロセッサステートメント)

機能

プログラム中の指定した定数またはマクロ名を、指定文字列で置き換えます。

書式

#define 〈記号定数〉〈文字列〉

または

#define 〈マクロ名(引数)〉〈引数を含む文字列〉

説明

プログラム中の〈記号定数〉または〈マクロ名〉を指定文字列で置き換えます。マクロ名の場合には、引数も含めて置き換えられます。

プログラム中の〈記号定数〉または、〈マクロ名〉がダブルクォーテーション“ ”で囲まれた文字列は置き換えられません。

#define 文は 1 行に書かなければなりません。

〈記号定数〉と〈文字列〉の間には、必ず 1 個以上のスペース文字を置かなければなりません。

マクロ名と引数を囲む（ ）との間に、スペース文字を置いてはいけません。

〈記号定数〉と〈マクロ名〉は、64 文字以内でなければなりません。

マクロ名は 1 つのプログラムで最大 2048 個まで使えます。また、マクロ関数の引数は数に制限がありません。

用例

```
#DEFINE NAME “Denso Corporation”
                              ’NAME という記号定数に“Denso Corporation”を割り
                               ’当てます。
S1 = NAME                     ’S1 に“Denso Corporation”が代入されます。
```

#include (プリプロセッサステートメント)

機能

プリプロセッサプログラムを取り込みます。

書式

#include “[パス]ファイル名”

#include 〈[パス]ファイル名〉

説明

#include 文を置いた場所に、プリプロセッサプログラムファイルを取り込みます。ファイルのパスを省略すると“ ”の場合は、カレントディレクトリ、システムディレクトリの順で、ファイルを探します。〈 〉の場合は、システムディレクトリだけからファイルを探します。パスをフルパスで指定すると、そのディレクトリだけを探します。

#include 文で指定したファイルの中に、さらに#include 文を含むことができ、8 回までのネストが可能です。

指定可能なファイルの拡張子は H です。

用例

```
#include “samp1.h”          ’samp1.h ファイルをこの行に展開します。
```

RC7 コントローラ用
操作盤機能説明書

取扱説明書　追補版

初　版 2005 年　 1 月

株式会社デンソーウェーブ FA 事業部

5G**C

株式会社デンソーウェーブ
FA 事業部
410002-6470-R1