トリプル・アイ／Win　Version3.0

iii／Win

interactive i-o implement for win32

トリプル・アイ／Win　概要ご紹介資料

2010年12月1日

株式会社パーシモンシステム

Persimmon system Corporation

はじめに

トリプル・アイ／Win（以下iii/winと記す）は、MS-DOS版トリプル・アイ（以下iii/dosと記す）及びOS/2版トリプル・アイ／２（以下iii/2と記す）との高い互換性を有するシリーズ最新、最上位の機能を持ったwin32版統合画面フロントプロセッサーです。

ホームページを含むiii/winのドキュメント類は基本的にiii/dosまたはiii/2をご存じの方を前提にこれらの製品から拡張された機能や変更された点を中心に記述しています。

これは当初iii/winのターゲット市場をiii/dosまたはiii/2からの移行案件に絞って考えていたためで、御検討いただくお客様はiiiシリーズのご経験者の方であることを前提としていました。

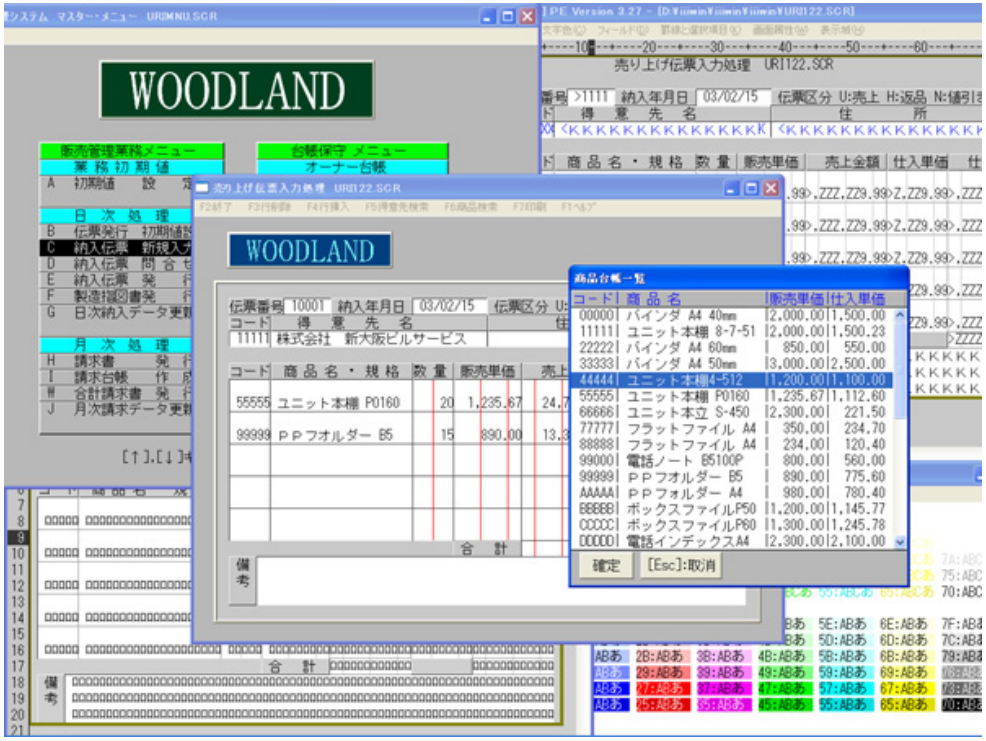
しかし、最近ではiiiシリーズをご経験でない方がiii/winでまったくの新規アプリケーションを開発したいというお話しや、移行案件であっても実際に移行作業を行われる技術者の方はiii/dosやiii/2での開発経験が無いというケースも増えて参りました。

そこでそのようなお客様にもiii/winを早くご理解いただくお手伝いのために本資料を作成しました。

iii/dosやiii/2をかつて使用していたが、しばらく触っていないので忘れてしまった。というような方のお役にも立てるかと存じます。

本資料はあくまでも一部機能を抜粋した概要書であり、iii/winの全ての機能説明を網羅しているものではありませんことをご承知ください。

DOS、OS/2といったプラットフォーム上で、多くのソフトウェア開発者の方々から絶大な指示をいただいてきたトリプル・アイ。



そして今再び、多くのトリプル・アイ資産がトリプル・アイ／Ｗｉｎによって**Win32**プラットフォーム上に蘇り活躍しています。

トリプル・アイ／Winとはどのようなツールか

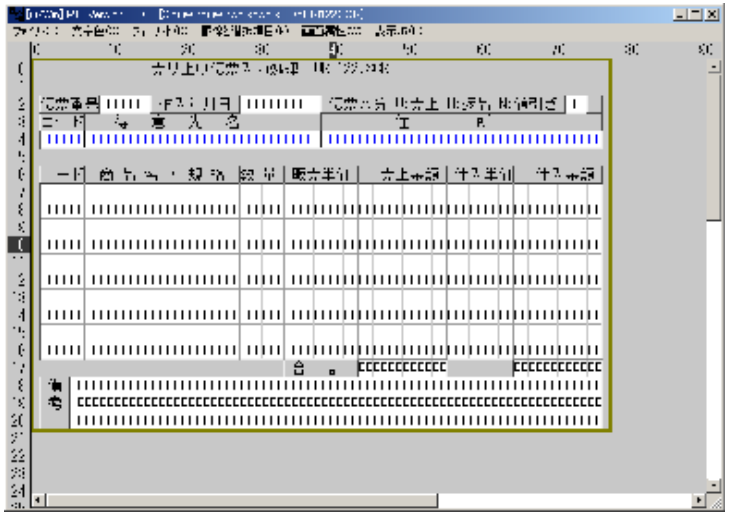
iii/winでは画面フォームや入出力フィールドをワープロ感覚の専用画面設計エディター（PE.EXE）で視覚的に作成し、実行時にはその設計した画面への入出力動作をiii/winの画面フロントプロセッサー（PSRUN.DLL)がアプリケーションプログラムに代わって行います。

iii/winはアプリケーションプログラムから独立した統合画面フロントプロセッサーです。

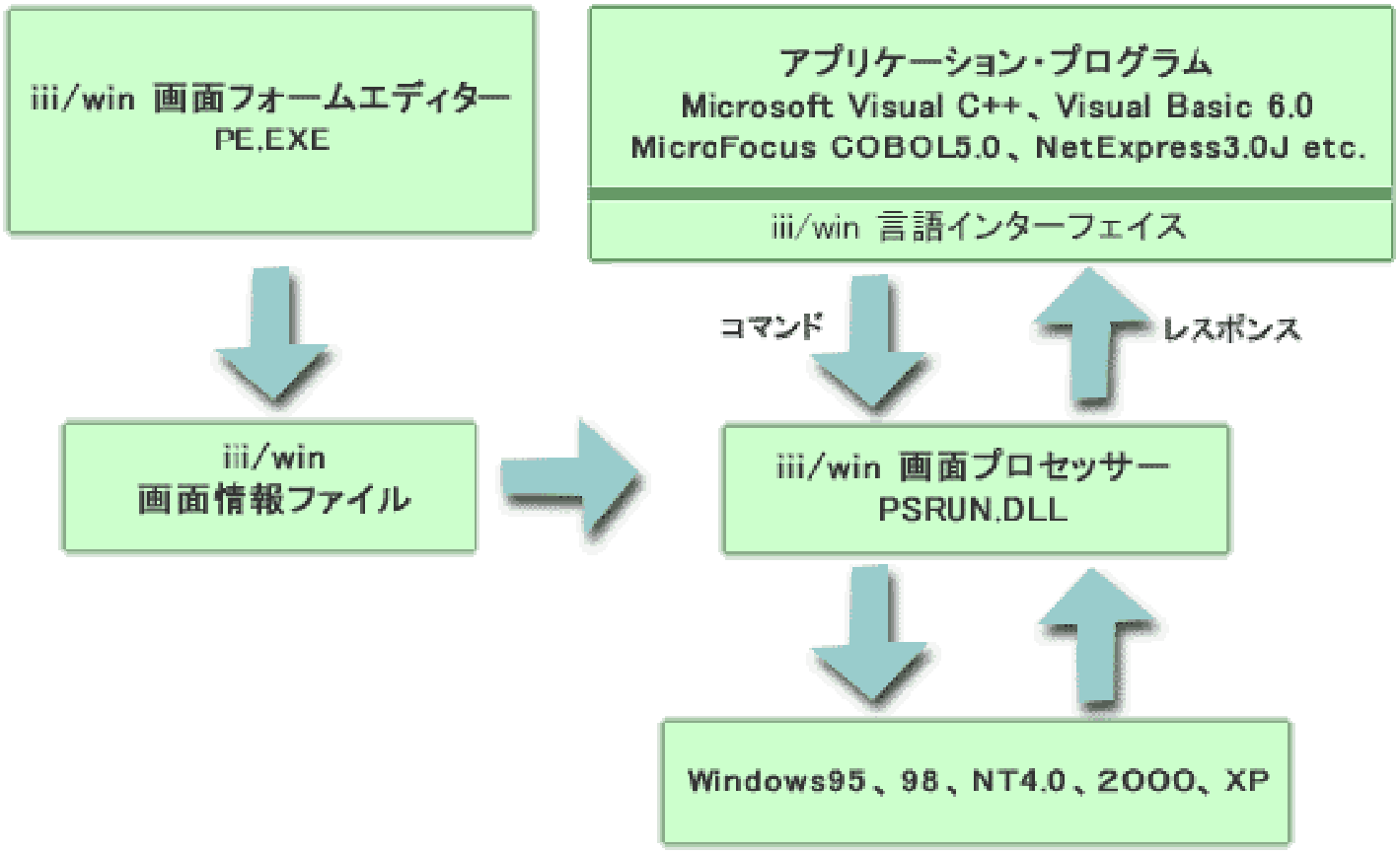
iii/winを使用して作成した画面は多様な言語からiii/winの画面プロセッサー（PSRUN.DLL）に各種iiiコマンドで指示（リクエスト）することにより使用できます。

iii/winの言語インターフェースには現在2種類あります。1つはVisualC++6.0及びVisualBasic6.0、でご利用いただくための関数形式のアクセスライブラリ（PSDLI.DLL）、もうひとつはNetExpresを含むMicroForcusCOBOLでご利用いただくためのコンソールインターフェース（CSIEXEC.EXE)です。

どちらでご利用いただく場合も、本体である画面プロセッサーや画面エディター、ならびに作成された画面情報ファイルは共通です。



トリプル・アイ／Winの概略構成図



トリプル・アイ／Winを構成するソフトウェア

１．画面設計用の画面フォームエディター　PE

入出力画面の設計を行うフルスクリーン型エディターです。

お望みのアプリケーション画面を視覚的に設計することができます。他のツールのようにコントロールをツールボックスからドッラグアンドドロップしてプロパティ設定するのではなく、テキストエディターイメージの編集ソフトであることが特徴となっています。

設計された画面は1画面毎に、画面情報ファイルとして保存されます。

２．画面制御を行う画面プロセッサー　PSRUN

画面情報ファイルの内容とアプリケーション・プログラムからのコマンド指示に基づき、画面表示やフィールドでの入出力などのさまざまな画面制御を実行します。

３．言語インターフェース　PSDLI、CSIEXEC

アプリケーション・プログラムと画面プロセッサーPSRUNの間で、コマンド（指示）やレスポンス（返事）のやりとりを行うためのインターフェースです。

アプリケーション画面（実行画面）について

iii/winでは、アプリケーション画面をワープロ感覚で作成します。一部の機能についてはマウス使った設定も行えますが、基本的にはキーボード操作のみで画面を作成できます。

作成（設計）した画面は１画面毎にファイルに格納され、このファイルを画面情報ファイルと呼びます。親画面から呼ばれる子画面も独立した１つの画面情報ファイルです。

画面情報ファイルは、画面を構成する罫線や色、固定文字などのバックグランド情報と、フィールド、選択項目の他、ボタンやリストボックス、メニューなどの各種GUIコントロール、ファンクションキー定義など、及びタイトルバーやフォント、ウインドウ枠線の種類などウインドウスタイルなどの属性の集まりです。

この画面情報ファイルを画面プロセッサーPSRUNが言語で構築されるプログラムからの指示に基づいて呼び出し、アプリケーション画面（実行画面）とします。

全ての画面の動作は、画面プロセッサーがアプリケーションプログラムからの指示（iiiコマンド）に基づいて制御します。

一般的な事務処理用画面にが要求されるさまざまな入力時の動作を画面プロセッサーが一手に引き受けます。アプリケーションの開発者は細かい画面操作に関するコーディングから解放され、Windowsアプリにありがちな特有なイベントハンドリングやフォーカス制御に悩まされることもなくアプリケーションの本質に神経を集中させることができます。

フルスクリーン型入力画面1

iii/winを使用することで簡単にフルスクリーン型入力画面が構築できます。

フルスクリーン型入力画面はフィールドと呼ぶ画面上の入力項目に対する入力をENTERキーで終了することができ、その場合次の入力は右隣のフィールドにかかります。

入力終了したフィールドの右隣にフィールドが存在しない場合は、下方向に次の行内を探し、行の最も左に位置するフィールドに入力がかかります。

こうしてENTERキーによる入力終了を繰り返した場合、画面の一番左上に位置するフィールドを起点として左から右、上から下の順番に入力が移動していきます。また上下左右方向の矢印キーでも入力終了が行え、この場合は入力終了したキーの矢印が示す方向に隣接するフィールドへ次の入力がかかります。ファンクションキーで入力終了した場合は再度同じフィールドに入力がかかります。もちろんマウスを使用した場合は、マウスがクリックしたフィールドが次の入力対象となります。

このようにマウスも使用可能ですが、基本的にはキーボード操作のみで自由に画面上の各フィールドに入力の移行が行えるような画面を「トリプル・アイ」シリーズでは伝統的に「フルスクリーン型入力画面」と呼んでいます。

タブキーやマウスを中心としたWindowsの標準的な操作方法は、非定型な対話型の処理には向きますが、基幹系を中心とした定型的な事務処理にはあまり効率的な操作とは言えません。

フルスクリーン型入力画面2

また、フルスクリーン型入力画面とは、プログラムの指示により次の入力場所が決まるのではなく、オペレータが自由に決められる画面でもあります。

iiiシリーズは、MS-DOSの上でもイベント駆動的なアプリケーションの構築をサポートする機能を有していました。iii/winでもその仕組みを踏襲しています。

Windowsはイベント駆動を基本とするOSです。オペレータの画面操作により発生したイベントはOSにより順にキューに貯められます。一般にWindowsアプリケーションはそのイベント毎に行うべき処理を記述していく形で形成されます。

しかし現実にはWindowsのイベントは非常に細かく多種に存在し、ひとつのイベント発生が次の予期しないイベントを誘発することもあるなど、意図した通りの動きのコーディングを行うには習熟を要し、かなり面倒でもあります。

このようなプラットフォーム上で各種GUIコントロールのハンドリングと整合性のある形で、Enterキーや矢印キーのハンドリングを行うことは意外と面倒なものです。

iii/winのフルスクリーン型入力画面はINPUTコマンドのループを構築することで実現します。

そのループ内で条件分岐を記述します。ひとつのフィールドの入力動作に伴い、フィールドの値の変化の有無とその入力を終了させる契機となったイベントの種類です。

これが簡単に分かりやすくコーディングできるのがiii/winの大きな特徴のひとつです。

画面（実行画面）の作成について

Windowsの画面開発では、文字や数字の可変値を入力する画面項目をテキストボックス等の文字入力用コントロール、そして出力のみの項目はラベル等を使用することが多いかと思いますが、iii/winの場合ここではこれらのコントロールは使用せず全て実行時にGDI描画で作成します。iii/winでは、iii/dos、iii/2の時からの伝統で、画面上で文字や数値の可変値を扱う役割を担う機能を入出力フィールド（又は単にフィールド）と呼びます。

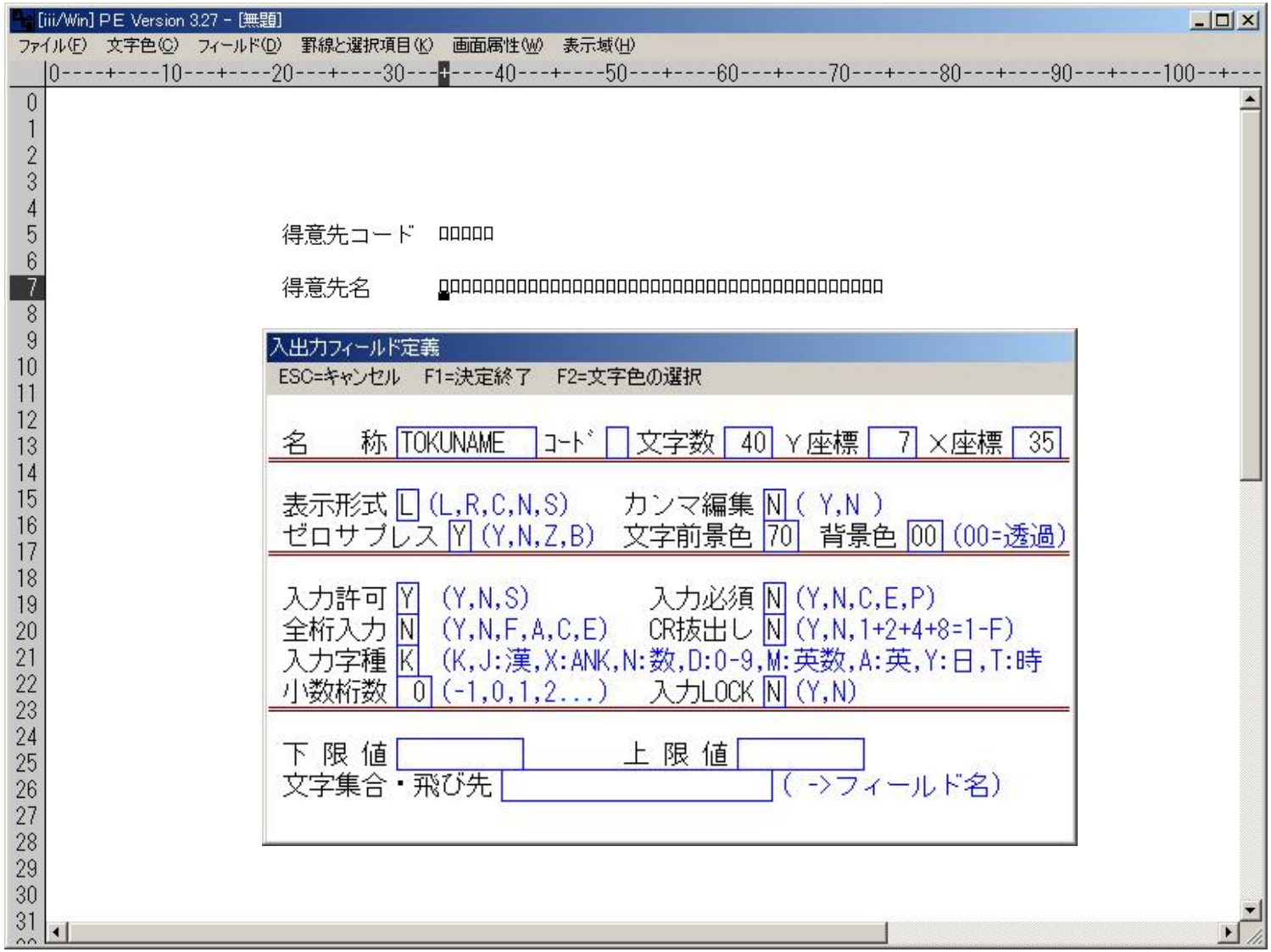
iii/winのフィールドはコントロールではないということは重要です。コントロールはOSからみるとそれぞれがハンドルで管理されるウィンドウであり、これを使用しないことでiii/winで作成した画面はシステム資源をあまり消費せず、非常に軽く、高速に動作します。

作成には、まず画面フォームエディターPE.EXE（起動はPEW.BAT）を立ち上げ、編集画面上の、フィールドを作成したい場所に矢印キーかマウスでカーソルを移動します。ここでファンクションキーF10を押すか、メニューバーから「フィールド」－＞「定義：入出力フィールド」を選択することで入出力フィールド定義パネルを開き、必要な項目を設定していきます。

定義したフィールドは、PEの編集画面上にはフィールドマーカー（‘ロ’が半角１文字）として表れます。このマーカー文字列は分割は行えませんが、手前に空白を挿入すると右へ移動、空白を削除すると左へ移動、またマークの削除でフィールド定義した文字数（フィールド長）を削除、‘ロ’の挿入で増加でき、まさにテキストエディターイメージの編集が行えます。

その他にも強力なショートカット機能をサポートしている上、iii/winの画面はあえてキャラクタ座標を採用しておりデザインも楽な上、アプリからの操作も簡単です。

入出力フィールドの属性定義パネルの設定例



入出力フィールドの定義内容

定義項目	入力内容	注釈・機能
名称	名前	10文字以内の英数字 （先頭は必ず英文字）
コード	任意1文字	処理グループの識別コード
文字数	フィールド長	最大150文字
X座標、Y座標	初期値自動セット	フィールド開始位置の座標値
表示形式	L R C N S	左詰 右詰 センタリング 無編集 シークレット（非表示）
カンマ編集	Y N	カンマ編集する（数値のみ） カンマ編集しない
ゼロサプレス	Y N Z B	ゼロサプレスする（N、T、Y） セロサプレスしない ゼロ埋（左余白部0、D、Y、T） ゼロサプレスで値0はブランク
文字全景色	色番号（178色）	色選択パネル（F2）から選択
文字背景色	色番号（178色）	色選択パネル（F2）から選択
入力許可	Y N	入力可能フィールド 出力専用フィールド

定義項目	入力内容	注釈・機能
入力必須	Y	ブランクでの入力終了不可
	N	ブランクでの入力終了可
	C	ブランク入力終了可で入力完了チェック（ICCHECK）対象
	E	入力無動作フィールド
	P	プロテクトフィールド
全桁入力	Y	全桁の入力指定
	N	任意桁数の入力
	F	全桁入力時自動抜け出し
	A	右端入力時自動抜け出し
CR抜け出し	Y	矢印キーでの入力終了不可
	N	矢印キーでの入力終了可
	1からF	有効な矢印キーを[→] =1、[←]=2、[↑]=4、[↓]=8、とし合計値を16進数で指定
入力字種	J	全角のみ入力可
	K	全字種入力可
	X	半角のみ入力可
	N	数値（‘0’～‘9’、‘-’、‘.’）
	D	数字（‘0’～‘9）のみ入力可
	M	英、数字専用（大文字変換）
	A	英字専用（大文字変換）
	Y	日付専用フィールド
	T	時刻専用フィールド

定義項目	入力内容	注釈・機能
少数桁数	桁数 －1 －1	小数部桁指定（四捨五入） 桁指定無し（数値フィールド） 日付、時刻自動設定（Y、T）
入力ロック	Y N	フルスクリーン入力スキップ フルスクリーン入力対象 ゼロ埋（左余白部0、D、Y、T） ゼロサプレスで値0はブランク
上限値	上限の値	上限値入力チェック指定
下限値	下限の値	下限値入力チェック設定
文字集合	入力許可文字列	入力許可文字（複数可）指定
飛び先	次入力場所（フィールド名）	ENTERキーによる次入力場所を通常ルールから変更

フィールド以外の画面編集について

画面構成要素には、可変値を扱うフィールドとは別に、各項目のタイトルなどの固定文字や罫線や背景などもあります。

iii／winでは、これらもフィールド編集と同時に行っていきます。

固定的な文字や文字列は、まさにテキストエディターイメージで編集できます。これらもフィールドと同じくラベル等のコントロールのは配置はなくiii/winが実行時にGDIで描画します。

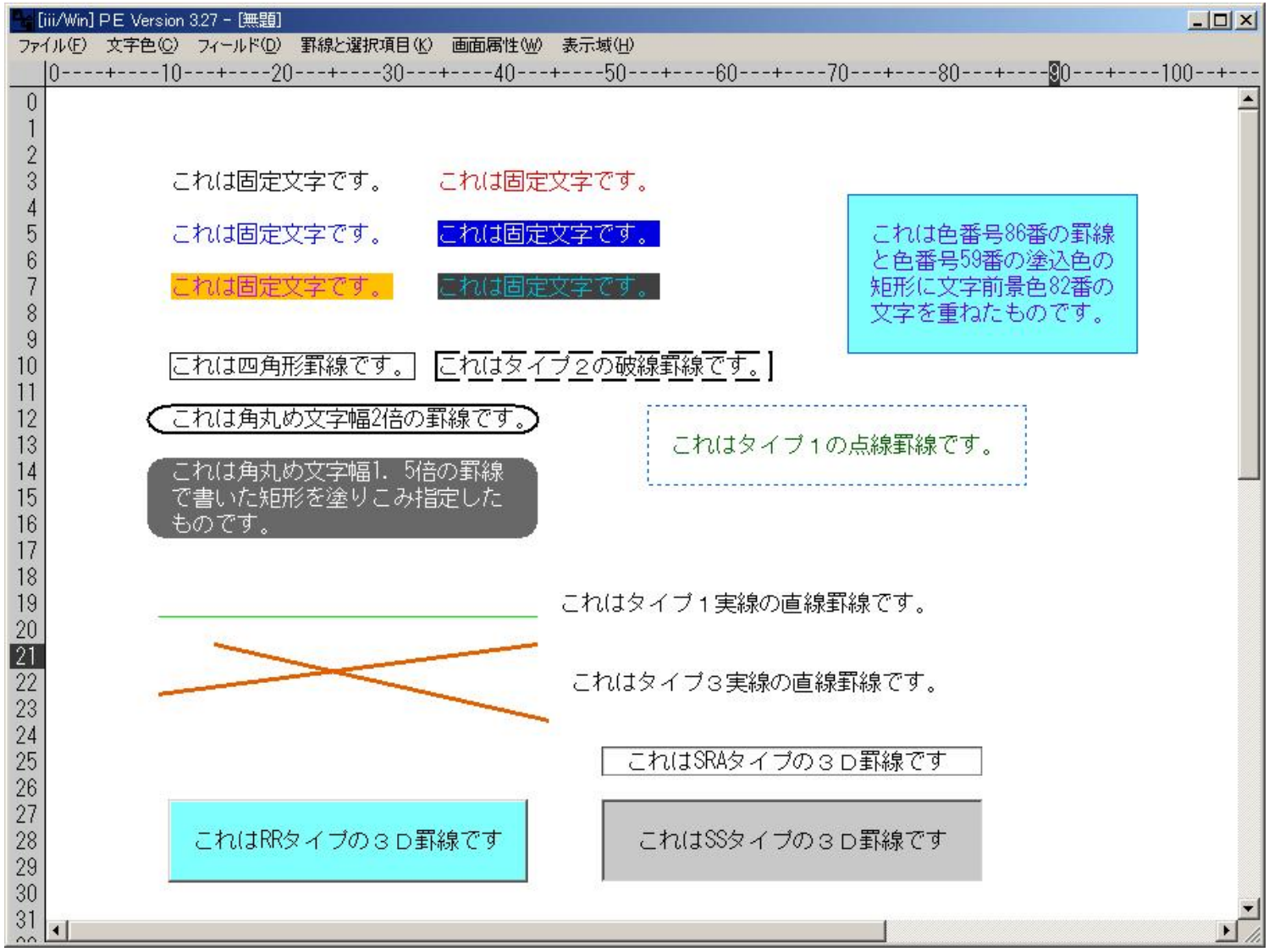
画面フォームエディターPEの編集画面上で、文字列を書き始めたい場所にカーソルを矢印キーかマウスで移動し、その状態で直接文字キーを押すと文字が書けます。編集にはテキストエディターイメージのショートカットを利用できます。

F1で文字前景色の選択、F3で線色の選択、F5で線種選択などを行います。

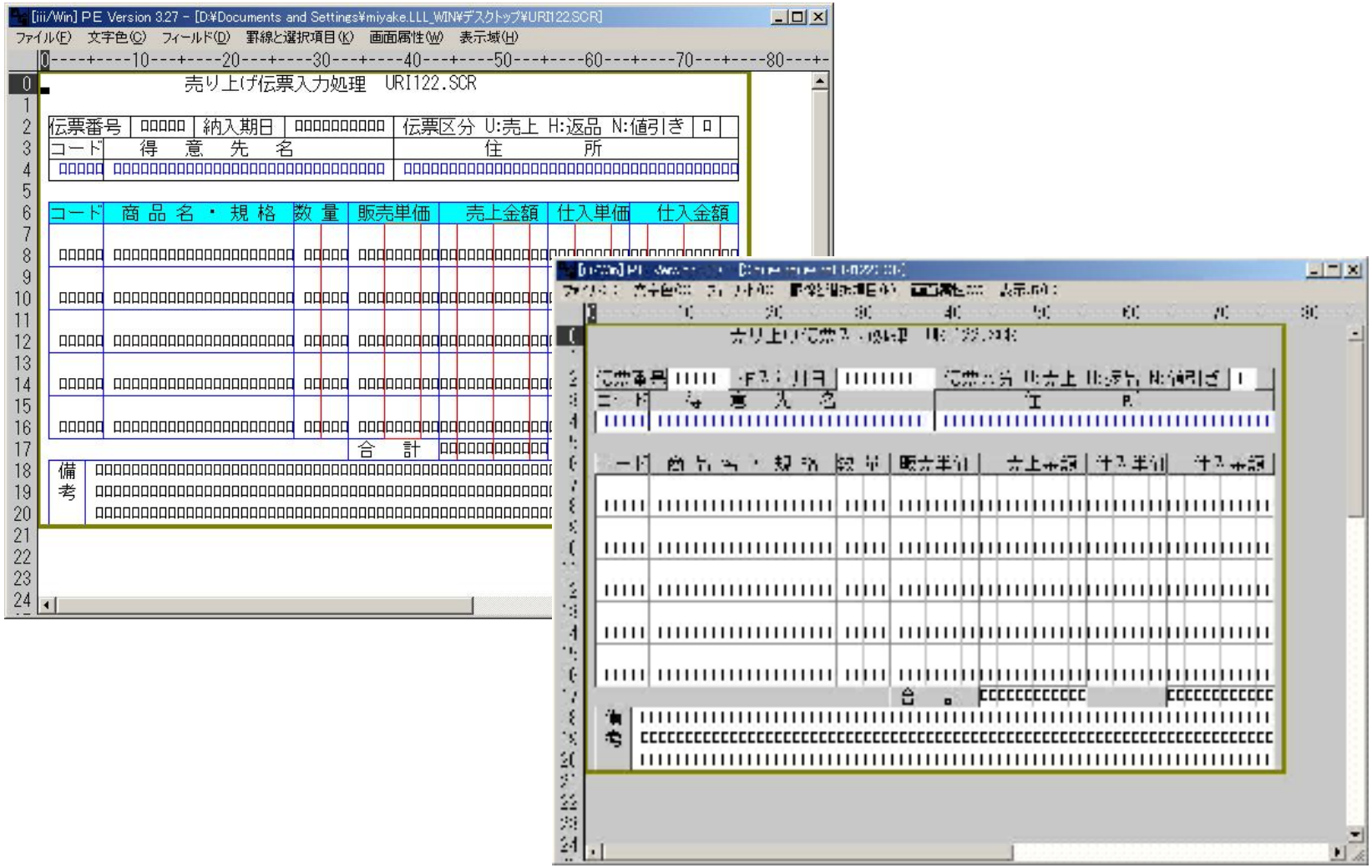
罫線は、起点（四角形の場合は左上角、直線は左端）でCTRL+KG、と押しそのまま矢印キーかマウスで終点へカーソルを移動してCTRL+KEとキーを押せば描画できます。消去はCTRL+KDです。

文字、罫線、面塗込みに対し、178色の色が選択できます。iii/winの罫線もいわゆるwindowsのコントロール（オブジェクト）ではありません。こっらもGDIでダイレクトに画面に描画されるグラフィックの線です。すなわち軽いプログラムの構築にも寄与しています。

画面フォームエディターによる固定項目の編集画面例



画面フォームエディターによる入出力フィールドの編集画面例



画面フォームエディター（PE）のキー操作１

文字編集キー		
キー操作		機能（動作）
Insert	Ctrl + v	挿入・上書きモードの交互切り替え
↑ ↓ → ←	Ctrl + E Ctrl + X Ctrl + D Ctrl + S	１行上へカーソルを移動 １行下へカーソルを移動 １字右へカーソルを移動 １字左へカーソルを移動
  Tab Enter	Ctrl + F Ctrl + A Ctrl + I	１ワード右へカーソルを移動 １ワード左へカーソルを移動 ４字分右へカーソルを移動 次の行へカーソルを移動
Backspace Delete	Ctrl + H Ctrl + G Ctrl + T	１字バック・スペース カーソル位置の１字削除 カーソル位置から行末まで削除
 Enter	Ctrl + Y Ctrl + N	１行削除 １行挿入（Enterは行頭でのみ）
	Ctrl + O Ctrl + P	真上の行を１行複写 既存の文字の色を現行色にする

画面フォームエディター（PE）でのキー操作２

スクロール操作	
キー操作	機能（動作）
Home	編集領域の先頭の行に移行する
End	編集領域の定義済み部分の末尾行に移行する
Page Down	表示内容の上シフト（末尾に進む）
Page Up	表示内容の下シフト（先頭の進む）
Ctrl + C	Page Down と同じ
Ctrl + R	Page Up と同じ
Ctrl + Home	編集領域の最左部へ移行する
Ctrl + End	その行の定義済み部分の右端に移行する
Ctrl + Page Down	表示内容の左シフト（末尾に進む）
Ctrl + Page Up	表示内容の右シフト（先頭に進む）

カラムゲージ表示	
キー操作	動作
Ctrl + L	ゲージ表示のON/OFF状態を交互に切り替える

画面フォームエディター（PE）でのキー操作3

グラフィック罫線定義と編集		
キー操作		機能（動作）
ラバーバンド非表示時	Ctrl + KG	開始位置の指定(G=Graphic) **この操作でラバーバンド（矩形）が出現。そこを起点にカーソル位置変更動作に伴いラバーバンド図形が伸縮します。続く本表下段のキー操作により動作が確定します。**
	Ctrl + KD	削除(D=Delete)　カーソル位置を左上点とする最上部の罫線の削除
	Ctrl + KO	削除(O=Omit)　カーソル位置を左上点とする最下部の罫線の削除
	Ctrl + KN	カーソル位置から下にある全ての罫線の端点を1行下に移動
	Ctrl + KY	カーソル位置から上にある全ての罫線の端点を1行上に移動
	Ctrl + KR	カーソル位置から右にある全ての罫線の端点を1字分1列右移動
	Ctrl + KH	カーソル位置から左にある全ての罫線の端点を1字分1列左移動
ラバーバンド表示時	Ctrl + KG	開始位置指定の変更指定(G=Graphic)
	Ctrl + KE	上書き決定(E=Enter)　既存罫線の最上部に描画
	Ctrl + KI	下書き決定(I=Insert)　既存罫線の最下部に描画
	Ctrl + KC	途中放棄(C=Cancel)　開始位置指定の解除（ラバーバンド消去）
	Ctrl + KD	削除(D=Delete)　ラバーバンド矩形内に完全に含まれる全ての罫線の削除
	Ctrl + KA	色変更(A=Attribute)　ラバーンバンド矩形内に完全に含まれる全ての罫線の色変更
	Ctrl + KN	ラバーバンド位置から下にある全ての罫線の端点を1行下に移動
	Ctrl + KY	ラバーバンド位置から上にある全ての罫線の端点を1行上に移動
	Ctrl + KR	ラバーバンド位置から右にある全ての罫線の端点を1字分1列右移動
	Ctrl + KH	ラバーバンド位置から左にある全ての罫線の端点を1字分1列左移動

ファンクションキーとメニューバー定義

iii／winにはメニューバーの作成方法が2種類用意されています。

１.「DOS形式のファンクションキー定義」

２.「メニューテンプレート定義」

DOS形式のファンクションキー定義とは、iii/dosからの移行性を重視したものでiii/dosで定義されていたファンクションキーの定義をそのままメニューバーの定義として流用するものです。

メニューテンプレート定義とは、ファンクションキー定義とは関係なく、最初からwindowsのメニューを作成していく機能です。各メニュー項目にはアクセラレータ、またはニーモニックとしてファンクションキーを割り付けることができます。

両方の定義は矛盾を生じますので、1つの画面にはどちらか一方の定義しか採用できません。メニューテンプレート定義が存在するとDOS形式の定義が有っても無視します。

メニューテンプレートの方が階層メニューが使えるなど機能性は高いですが、DOS形式の方が機能が低い分だけハンドリングは簡単です。

iii/dosで作成した画面情報ファイルをiii/winのPEに読み込んだ場合は、自動的に一旦DOS形式ファンクション定義として情報が引き継がれます。これは後で変更可能です。

DOS形式のPFキー定義によるメニューバー作成

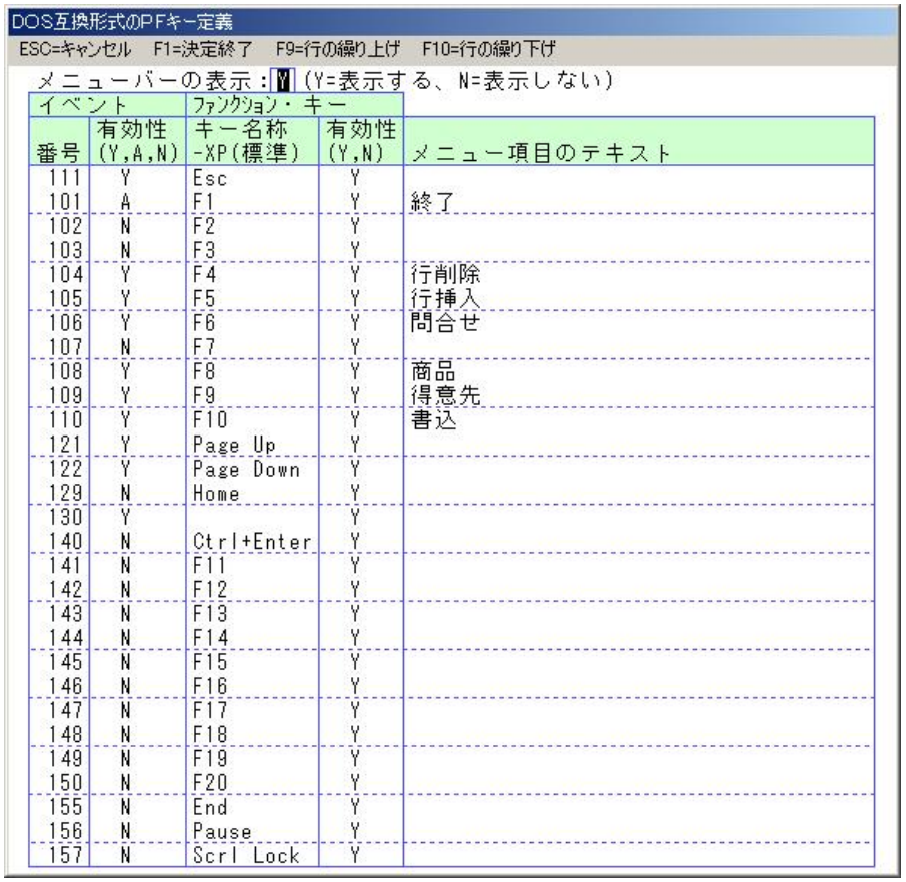
PEのメニュー[画面属性]－＞[DOS形式のPFキー定義]を選択すると、右の定義パネルが開きます。

ここでメニューバーの表示の有無を始め、各ファンクションキーの有効無効、またキーが対応するメニューバー上の項目テキストを設定していきます。

ファンクションキーには、ESC=111、

F1＝101、などのイベント番号が予め割り振られており、iii／winの入力コマンドのレスポンスとしてこのイベント番号がアプリに戻ります。

右の定義内容での実行イメージは次ページにあります。

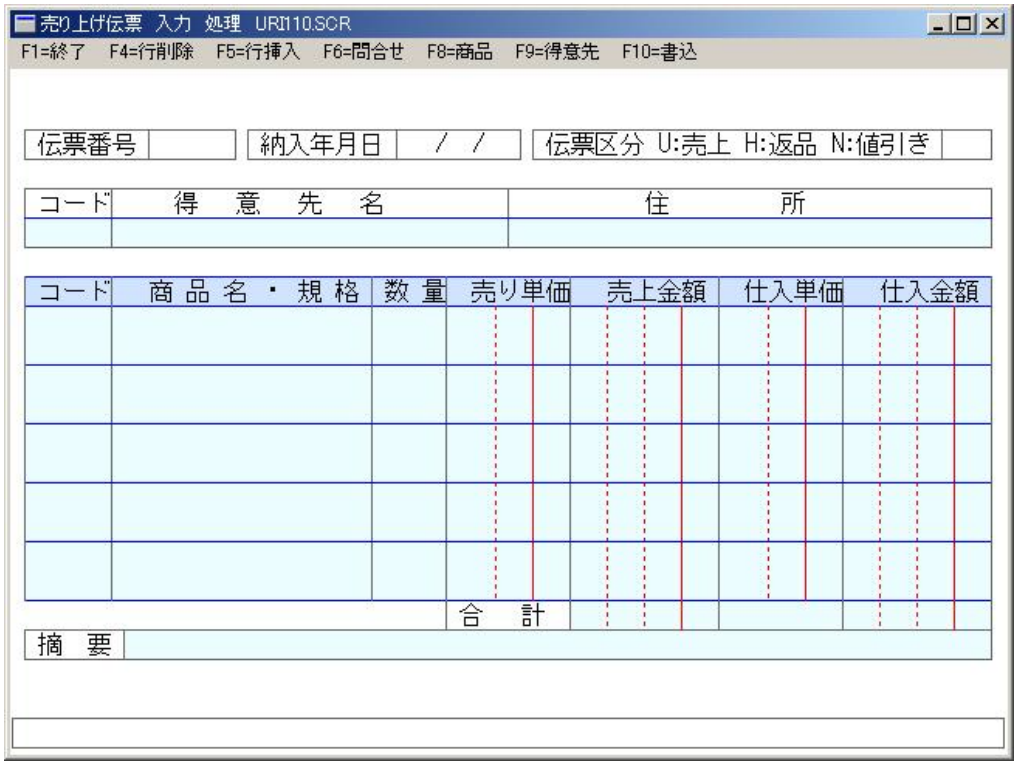

DOS互換形式のPFキー定義

ESC=キャンセル　F1=決定終了　F9=行の繰り上げ　F10=行の繰り下げ

メニューバーの表示：Y (Y=表示する、N=表示しない)

イベント		ファンクション・キー		
番号	有効性 (Y,A,N)	キー名称 -XP(標準)	有効性 (Y,N)	メニュー項目のテキスト
111	Y	Esc	Y	
101	A	F1	Y	終了
102	N	F2	Y	
103	N	F3	Y	
104	Y	F4	Y	行削除
105	Y	F5	Y	行挿入
106	Y	F6	Y	問合せ
107	N	F7	Y	
108	Y	F8	Y	商品
109	Y	F9	Y	得意先
110	Y	F10	Y	書込
121	Y	Page Up	Y	
122	Y	Page Down	Y	
129	N	Home	Y	
130	Y		Y	
140	N	Ctrl+Enter	Y	
141	N	F11	Y	
142	N	F12	Y	
143	N	F13	Y	
144	N	F14	Y	
145	N	F15	Y	
146	N	F16	Y	
147	N	F17	Y	
148	N	F18	Y	
149	N	F19	Y	
150	N	F20	Y	
155	N	End	Y	
156	N	Pause	Y	
157	N	Scrl Lock	Y	

DOS形式のPFキー定義によるメニューバー（実行イメージ）



この画面は、iii/dosで作成してあった画面情報ファイルをiii/winに移行した例です。

起動パラメータで上記のようにメニュー項目に F1= など頭書文字を附加できます。

メニューテンプレート定義

「DOS形式のPFキー定義」が、まずファンクションキーありきでそれのガイダンスとしてメニューバーを利用しているのに対し、「メニューテンプレート定義」では考え方が全く逆転し、まずメニューありきで、そのメニューをキーで選択することもできるという機能です。

１階層につき１８までの項目を持つメニューが4階層まで作成できます。

そしてメニュー項目に対してアクセラレータ（ショートカット）キー定義を行うことでファンクションキーなどでもそのメニューが選べるようになります。

アクセラレータキーにはファンクションキーー以外の文字キーや数字キーなども使えます。CTRL+、ALT+、SHIFT+などベースキーと組み合わたキー操作も指定できます。アクセラレータは強力です。メニュー面のアクティブ状態に関係なく有効ですのでひとつの親画面内の複数メニューに対し同一キーを割り当てることはできません。

さらにメニューテンプレートでは、アクセラレータキーとは別にアクセス（ニーモニック）キーも使用できます。これはOFFICEのトップレベルメニューなどでもお馴染みのもので、メニュー項目の右隣にアンダーバー付き文字で表されます。これはALTキーを押してから、またはALTキーを押しながらその文字を押すとメニュー項目が選択されるという機能です。

アクセスキーはそのメニュー面がアクティブな時だけ有効です。従ってメニュー面が異なれば親となる画面は同じでも同一のキーが指定できます。また最終的なコマンドを指すメニューではなくサブメニューを持つ親メニューの呼び出しにも指定することもできます。

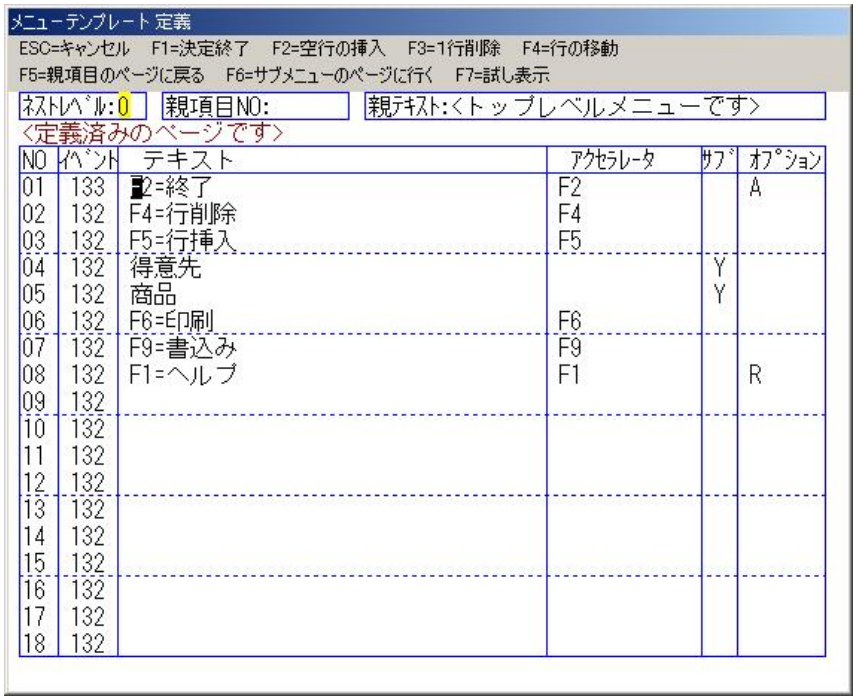
メニューテンプレート定義によるメニューバー作成

PEのメニュー[画面属性]－＞[メニューテンプレート定義]を選択すると、右の定義パネルが開きます。

ここでメニューバー上に表示するメニュー項目のテキスト、アクセラレータ、サブメニューの有無、オプションを指定します。

メニューのイベント番号の初期値は全て132です。132から139の値を指定できます。

18^4 個のメニュー項目に対し、3桁のイベント番号だけでは当然ハンドリングできません。



NO	ｲﾍﾞﾝﾄ	テキスト	ｱｸｾﾗﾚｰﾀ	ｻﾌﾞ	ｵﾌﾟｼｮﾝ
01	133	■2=終了	F2		A
02	132	F4=行削除	F4		
03	132	F5=行挿入	F5		
04	132	得意先		Y	
05	132	商品		Y	
06	132	F6=印刷	F6		
07	132	F9=書込み	F9		
08	132	F1=ヘルプ	F1		R
09	132				
10	132				
11	132				
12	132				
13	132				
14	132				
15	132				
16	132				
17	132				
18	132				

実際にはイベント番号とは別にメニューID Noが割り振られていますので、この番号を使って選択された項目の識別を行います。

メニューテンプレート定義によるメニューバー（実行イメージ）

このようにプルダウン式の階層メニューが簡単に作成できます。階層が深くてもアクセラレーター指定があればファンクションキーによる一発選択が可能です。

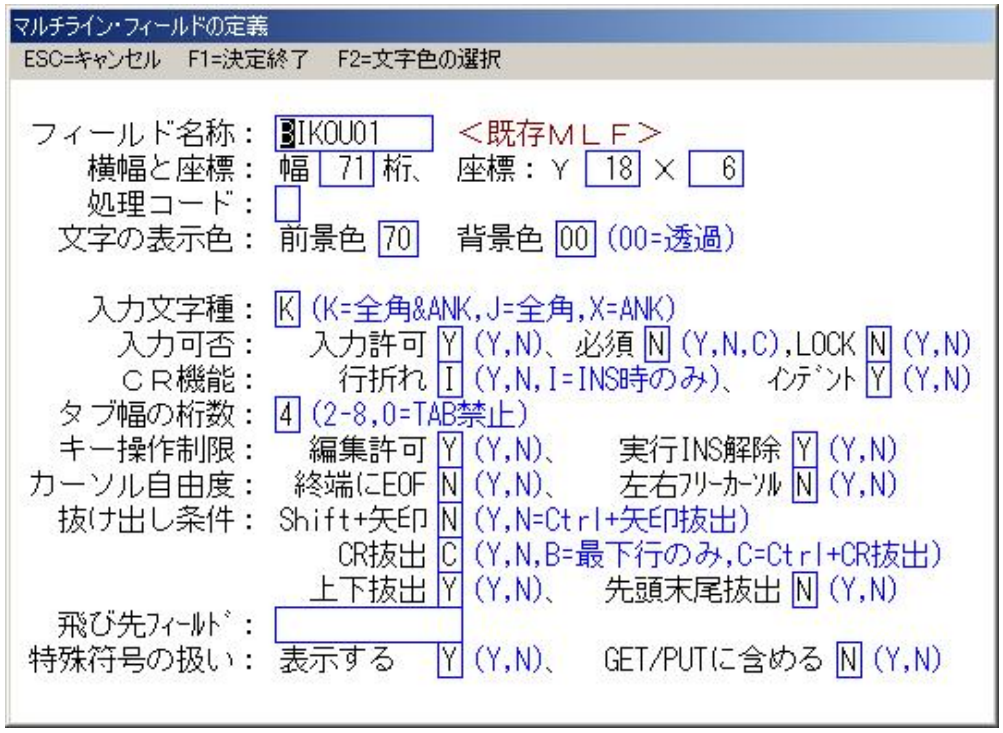
マルチライン・フィールド

マルチラインフィールド（MLF）はテキストエディターの一種であり、複数行にわたるテキストを表示したり編集する機能を提供します。

MLFを利用すれば、備考欄やメモ欄などの文章の編集を簡単に、しかも効率的に行うことができます。

これも、iii/winが独自に提供する機能です。

Windowsの標準コントロールではありません。



その他の画面部品

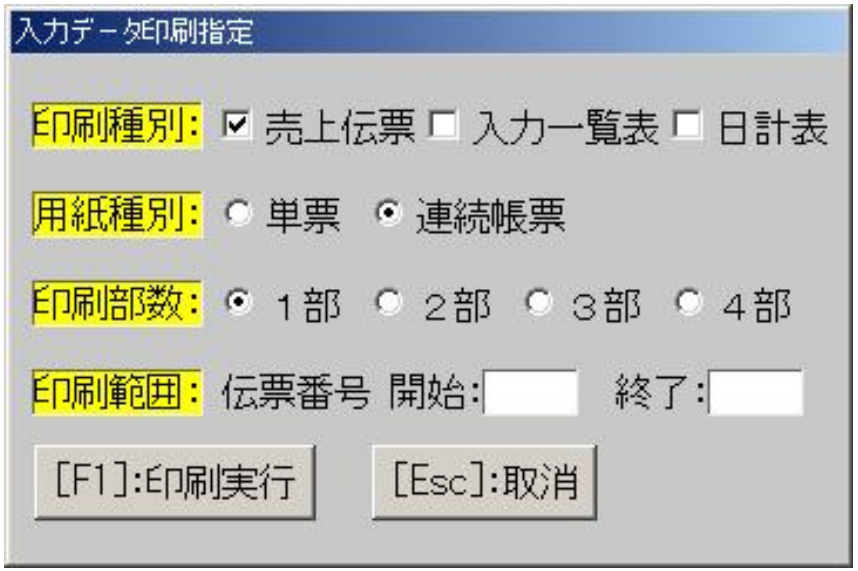
Windowsの画面ではボタンやリストボックスなど各種GUIコントロールを使用することがあります。

iii/winではこれらもPEでフィールドマークとして編集、作成します。

iii/winでは、下記の基本的なGUIコントロールをサポートしています。

1. プッシュボタン
2. ラジオボタン
3. チェックボックス
4. 3ステートチェックボックス
5. リストボックス

※ その他の標準コントールや市販のActiveXコントロールなどは使用できません。



これらは、iii/winがGDIで独自に描画するのではなく、Windowsのコントロールを使用します。

すなわちハンドルなどWindowsのシステム資源を消費します。

フルスクリーン型入力画面の実現方法1

作成した画面に対して、アプリケーションプログラムからiiiコマンドの入力命令INPUT にパラメータ /NETXV を附加（INPUT|/NEXTV）して実行していただくと、最初（画面右上）のフィールド入力がかかります。

一回のINPUTコマンドは一つのフィールドに1回入力動作を行って終了します。

フィールド作成時に実際につけたフィールド名をINPUTコマンドのパラメータとすると、画面上での位置に関わらず、その名前のフィールドに入力がかかりますがこのようなコーディングではフルスクリーン型入力の実現は難しいといえます。

実際にはINPUTコマンドのパラメータには多くの場合、上記でご説明した /NEXTV という値を指定していただきます。

これはiii／winの予約語である特殊フィールド名のひとつで「次の入力場所」を意味し、実際のフィールドにはつけられない名前です。

詳細は後述しますが、Enterキーで入力終了すると、「次の入力場所」は右隣のフィールドです。矢印キーならその方向、マウスならクリックしたフィールドが「次の入力場所」となります。

すなわち終了条件（例えば終了に割り付けられたファンクションキーなどが押される）になるまでINPUT|/NEXTVを繰り返す（ループする）コーディングにより勝手にフルスクリーン型入力となる画面が実現できます。入力された値はiii／winが画面上で保持しています。必要な時にいつでもGETコマンドでアプリに取り込むことができます。

フルスクリーン型入力画面の実現方法2

出来上がった画面の各フィールドには、作成時の設定がありますので、入力が不許可のフィールドや入力をLOCKされたフィールドはスキップする他、字種、上下限値、少数桁数、日付、時間などの入力チェックや出力編集など、フィールド作成時に定義した項目の指定に従い必要な処理を行う機能は既に自動的に実現しています。C言語での一般的なコーディング例は以下の通りです。

```
while()
{
    psinpt("/NETXV");
}
```

iiiコマンド　INPUT　に対応するアクセスライブラリ関数がpsinpt()です。

上記サンプルにはループの終了条件がありませんので、強制終了がかかるまで永久に表示中の画面のフィールドに対し入力動作を繰り返します。

一般的にはINPUTからの戻り値を見て処理の分岐を行うロジックを組み合わせます。

フルスクリーン型入力画面の実現方法3

iiiコマンド INPUT （C関数ではpsinpt()) のパラメータについて説明します。

画面上に実在するフィールド名を具体的に指定した場合は、そのフィールドに入力がかかります。

実在するフィールド名の変わりにiii予約語である特殊フィールド名を指定した場合は、iii／winによって決められているルールに従い入力フィールドが確定します。

フルスクリーン型入力を行うための特殊フィールド名は3つあります。

1. /NEXT
2. **/NEXTV**
3. /VNEXT

もっとも一般的に使用されるのは2. の /NEXTV でしょう。

フルスクリーン型入力画面の実現方法4

前述の3つの特殊フィールド名は、いずれもフルスクリーン型入力において「次のフィールド（正確には次に入力を行うべきフィールド）」を意味します。

基本形である ／NEXT は、直前の入力がENTERキーで終了されていた場合、直前に入力が完了したフィールドを起点として次のフィールドは右隣を指します。

右隣にフィールドが存在しなければ直下の行の左端に位置するフィールドを指します。ここまでは/NEXT と /NEXTV の動作は同じです。

異なるのは上下矢印キー[↑][↓]が押された時の動作です。

[↓]の場合 /NEXTは、まず直下の行の左端から順に左から右、上から下へ次の入力フィールドを探しますが、/NEXTVではフィールド開始位置のX座標が同じフィールドがあれば行の近さよりも優先的にそのフィールドへ入力を行います。[↑]も同様です。

これは複数行で構成される伝票明細のような画面を想定した場合、例えば1行おきに数量欄が存在するなどということは良くある話しですが、そのような場合の縦方向移動は、次の数量欄に移動できたほうが自然なケースが多いからです。

ちなみにもうひとつの /VNEXT はENTERキーによる入力終了時の次のフィールドは横軸ではなく縦軸優先となり、上から下、左から右に移動するという指定です。

フルスクリーン型入力画面の実現方法5

INPUTコマンドが完了すると、アプリケーションに制御が戻ります。

この時アプリケーションはiii／winの画面プロセッサーからのレスポンスを受け処理の分岐を行います。

```
int   EventNo = 0;          //  PSRUNからのレポンス格納用変数
while(EventNo!=133)    //  F2押下により終了するINPUTループ
{
        EventNo = psinpt("/NETXV");    //  フルスクリーン型
        swich(EventNo){                        //  イベントによる処理分岐
                case134:uriage_syori();break;  //ユーザー作成処理
                case135:siire_syori();break;    // ユーザー作成処理
                default:break;}
}
```

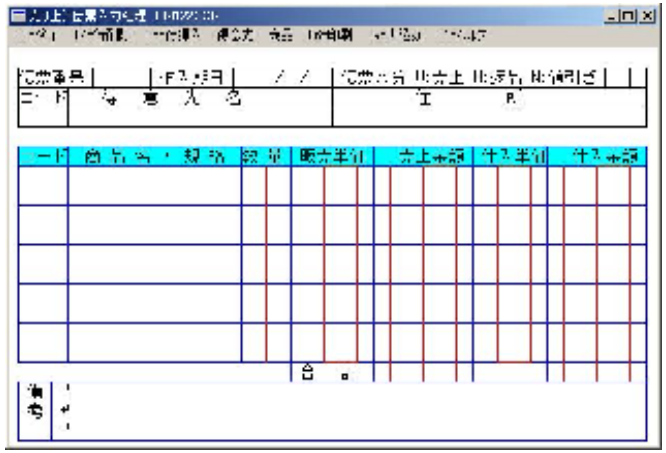
レスポンスの133、134、135は、それぞれ対応するメニュー項目のイベント番号です。イベント133のアクセラレータにF2を指定すると、F2が押下で終了します。

フルスクリーン型入力画面の実現方法6

前述の処理の前に、初期化処理、画面表示、後ろに終了処理を加えると、もっとも簡単なフルスクリーン型入力プログラムが完成します。

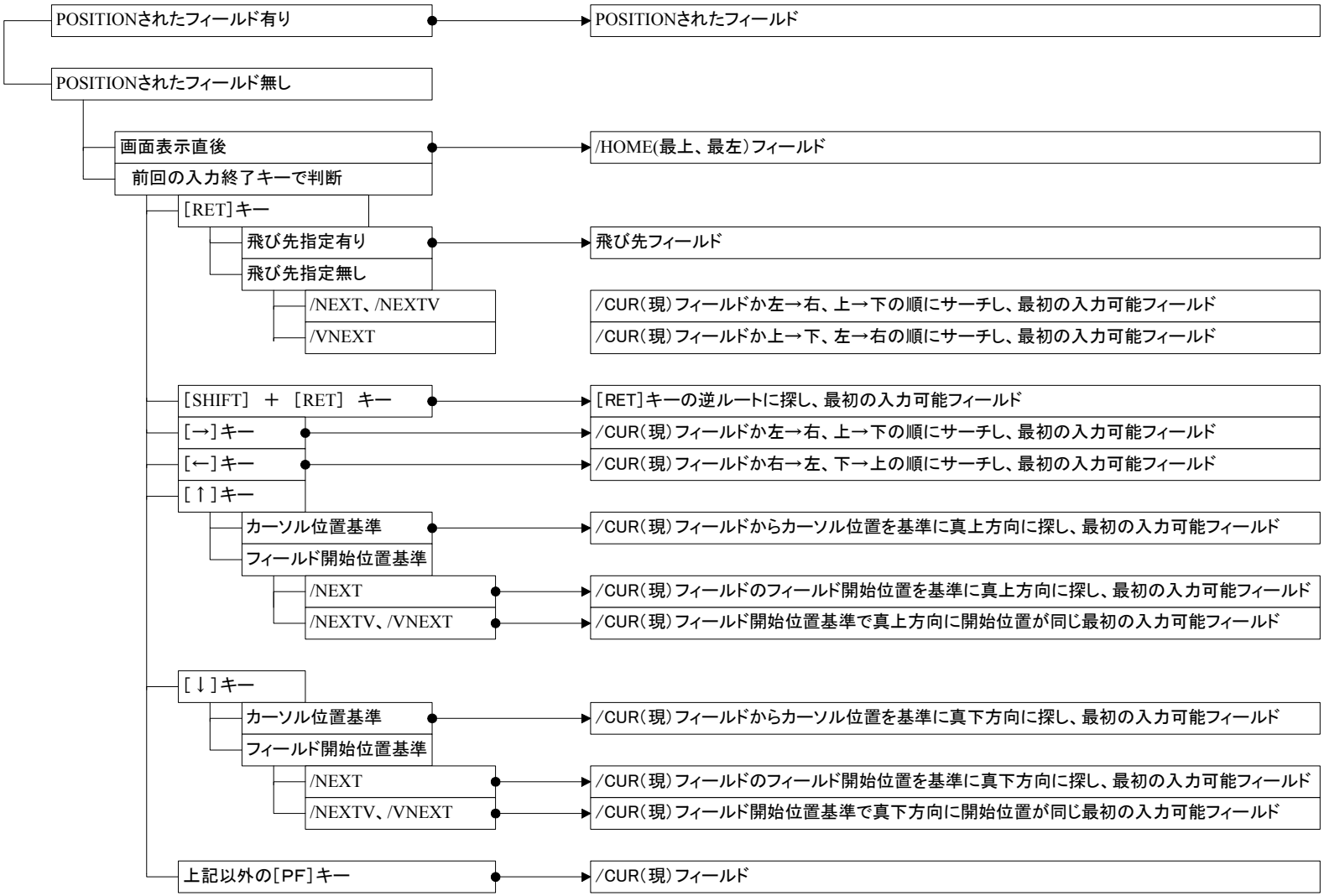
```
#include   <stdio.h>
#include   <psdli.h>     // iii/Win用ヘッダ・ファイル
#define    WIN32_LEAN_AND_MEAN
#include   <windows.h>
int WINAPI WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,LPSTR lpCmdLine,int nCmdShow)
int  EventNo = 0;          //  PSRUNからのレポンス格納用変数
psinit()                   //  PSRUNの初期化
psscrn("URI122.SCR");      //  親画面  URI122.SCR  の表示
while(EventNo!=133)  {         //  F2押下により終了するINPUTループ

                EventNo = psinpt("/NETXV");    //  フルスクリーン型
                swich(EventNo){                //  イベントによる処理分岐
                        //  case134:uriage_syori();break;  //ユーザー作成処理
                        //  case135:siire_syori();break;    // ユーザー作成処理
                            default:break;
                }
}
psquit()                   //  PSRUNの終了処理
```



このプログラムは実際に動作します。
これだけで入力チェック、出力編集、フルスクリーン型カーソル制御機能を備えています。

次のフィールド（/NEXT、/NETXV、/VNEXT）の指す位置



/NEXT系以外の特殊フィールド名（汎用）

フィールドを操作するiiiコマンドには、通常パラメータとして操作対象となるフィールド名を具体的に指定する、というのが基本的な用法です。しかし実際のコーディングでは具体名ではなく抽象名で操作対象を指定できた方が効率が良いケースが多々あります。

この特殊フィールド名の活用度合いが、アプリの開発効率に大きく影響します。

特殊フィールド名は、‘/’で始まるiiiの予約語であり実際のフィールド名には使用できません。

/HOME	画面上の最初（最上で最左の）の入力可能フィールド
/NHOME	現フィールドの次の行以下に存在する最初の入力可能フィールド
/CUR	現フィールド(最も最近に入力したフィールド)
/MOUSE0	マウス・クリックのイベントが発生したフィールド
/+n、/-n	現フィールドからの相対位置
/LAST	画面上の最後（最下で最右）のフィールド
/LASTI	画面上の最後（最下で最右）の入力可能フィールド
/nn	オフセット（通し番号）によるフィールド指定
/TOPOFLINE	現フィールドと同じ行の最も左のフィールド
/*配列フィールド名	現フィールドと同じ行の該当の配列フィールド

特定のiii/winコマンドでのみ利用可能な特殊フィールド

特殊フィールド名	機　　能	対象コマンド
/EVENT	イベント入力待ち（ウエイト有り)	INPUT
/FKEY	同上	
/EVENTNW	イベント入力待ち（ウエイト無し）	INPUT
/FKEYNW	同上	
/CPOS	フィールド入力終了時のカーソル位置取得	OFFSET
/CPOSnn	次のフィールド入力時のカーソル初期位置指定	POSITION
/LCPOS	次の入力時の前回入力カーソル位置継承指定	POSITION
/AUTO	入力フィールド文字色自動反転指定	COLOR
/#nn	選択項目に含まれるフィールドの オフセット	OFFSET
/Fnn	指定フィールドを含む選択項目の番号取得	OFFSET
/ALL	全てのフィールド	CLEAR
/ALLU	全ての入力許可フィールド	CLEAR
/ALLL	全ての入力不許可フィールド	CLEAR
/E	最後の連番を持つ配列フィールド	POP、PUSH
/Y行番号	指定行に出力する	PUT
/MOUSE1～9	/MOUSE0以外の各種マウスベント付帯情報	OFFSET
#nn	選択項目を擬似的なフィールドとして扱う	PUT、GETなど

トリプル・アイ／Win コマンドの送出とレスポンスの受け取り

iii/winとプログラム言語で作成するアプリケーションプログラムは、コマンドとレスポンスのやりとりにより連携しますが、連携用に専用の言語インターフェースを用意しています。

VisualC++、VisualBasic用のアクセスライブラリと、MicorForcusCOBOL用のコンソールインターフェースの2種類の言語ンターフェースがあります。

アクセスライブラリは以下のプロダクトで構成されます。

psdli.dll アクセスライブラリ本体

psdli.lib VisualC++用インポートライブラリ

psdli.h VisualC++用ヘッダーファイル

psvbi.bas VisualBasic用関数宣言ならびに下請けプロシージャファイル

iii2.ini 起動パラーメータファイル（サンプル）

コンソールインターフェース

csiexec.exe コンソールインターフェース本体

アクセスライブラリとコンソールインターフェース

アクセスライブラリは、iii/winコマンドに対応する関数群がライブラリとして格納されたDLLです。

これをVisualC++6.0、またはVisualBasic6.0にLINKして使用します。関数名がコマンド、レスポンスは関数の値としてアプリケーションに戻ります。

コンソールインターフェース（CSIEXE.EXE)は、アクセスライブラリと異なり、アプリケーションにはLINKしない独立した実行プログラムです。ただし単独で起動することはできず画面プロセッサーPSRUN.EXEから起動する必要があります。

これはプログラム言語が持つコンソールへの入出力機能をリダイレクトによって横取りするソフトです。MicrofocusCOBOLであれば、display と accept を横取りします。

すなわちアプリケーションからiii/winへのリクエストは、display 命令でiiiコマンドを文字列として出力することによって行い、レスポンスは直後の accept 命令によって受け取ります。

display “@INPUT|/NEXTV”

accept eventno

このインターフェースは言語とリンクしません。従って.netのSystem.Console.WriteLine 等、標準入出力機能が使用できればCOBOLからでなくてもアクセス可能です。

使用言語別ランタイムルーチン

iii/winアプリケーションを前提とするアプリケーションプログラムを開発して、エンドユーザー環境に配布し運用させるために、エンドユーザー環境毎に必要となる画面情報ファイル、及び開発されたアプリケーションプログラム以外のiii／winプロダクト（ランタイムルーチン）は、PSRUN.DLL（iii／win画面プロセッサー本体）、PSRUN.EXE（PSRUN.DLLを単独実行させるプログラム）、PSDLI.DLL（C言語用関数ライブラリ）、CSIEXEC.EXE（コンソールインターフェース）、III2.INI（iii／win動作環境定義ファイル）、の5つです。

このうちどれとどれが実際に配布する必要があるかは、開発に使用されたプログラム言語によって異なります。

VisualC++	**PSRUN.DLL、PSDLI.DLL、III2.INI**
VisualBasic	**PSRUN.DLL、PSRUN.EXE、PSDLI.DLL、III2.INI**
MicorforcusCOBOL	**PSRUN.DLL、PSRUN.EXE、CSIEXEC.EXE**

これらはいずれもWINDOWSへのレジストリを必要としません。

.NETプラットフォームとは関係なく、昔からWIN32上でXCOPYベースのインストールが行えることもiii／winアプリケーションの特長です。

※ 製品としてのランタイムライセンスに媒体で提供されるのは、PSRUN.DLL、PSRUN.EXEのみです、他のモジュールは必要に応じて言語インターフェース本体製品媒体からコピーしてご使用いただきます。

COBOLによるフルスクリーン型入力画面プログラム

```
 IDENTIFICATION      DIVISION.
         PROGRAM-ID.         CSIDEMO.
 ENVIRONMENT         DIVISION.
 DATA                DIVISION.
      WORKING-STORAGE     SECTION.
      01  WORK-DATA.
          02   EVENT-NO     PIC  9(04) COMP.
  PROCEDURE DIVISION.
*           iii/winの初期化と親画面URI122.SCRの初期表示
         DISPLAY "INIT|C“
         DISPLAY "SCREEN|URI122.SCR"
         MOVE    1 TO EVENT-NO
*                   フルスクリーン型入力ループ
         PERFORM TEST AFTER UNTIL SEL-NO = 133
            DISPLAY "@INPUT|/NEXTV"
           ACCEPT  EVENT-NO
            EVALUATE   EVENT-NO
               WHEN 134
 *                   ユーザ作成処理
               WHEN 135
 *                   ユーザ作成処理
            END-EVALUATE
         END-PERFORM.
*                    終了処理
        DISPLAY "QUIT"
        STOP    RUN.
```

先ほどC言語のコーディングで説明した最も短いフルスクリーン型入力画面プログラムのサンプルをCOBOLで書くと左のようになります。

Display命令を使用し、COBOLプログラムからすれば、あたかもコンソール画面に文字列を出力するかのようにiiiコマンド（＋パラメータ）を書きます。

コマンドとパラメータ、及び複数のパラメータ間は、‘|’で区切ります。

コマンド文字列の頭に‘@’を付けると画面プロセッサーPSRUNは、アプリケーションにレスポンスを返します。‘@’を付加した時は必ず直後でaccept し、レスポンスを受け取らなければなりません。逆に直前に‘@’付きコマンドが無いのにaccept を行ってはいけません。

VisualBasic、その他言語によるフルスクリーン型入力画面プログラム

```
Attribute VB_Name = "Module1"

Sub Main()
   Dim EventNo As Integer       ' 選択項目の番号

   PsInit                       ' 画面プロセッサを初期化
   PsScrn “URI122.SCR”     ‘ 親画面uri122.scrを表示
   EventNo = 1
   Do
      ' フルスクリーン入力
      EventNo = PsInpt("/nextv")
      If EventNo = 133 Then Exit Do  ' メニュー項目[F2=終了]

      Select Case EventNo
         Case 134   ‘ ユーザー作成処理
         Case 135   ‘ ユーザー作成処理
       End Select
   Loop

   ' 終了処理
   PsQuit
End Sub
```

同じサンプルをVisualBasicで書くと左のようになります。

画面の処理をiii/winに行わせるプログラムにおいては、どのようなプログラム言語を用いても、ほぼ同じようなコーディングスタイルになることがお分かりいただけると思います。

ちなみに、C言語関数の呼び出しが可能ならアクセスライブラリ、標準コンソールへの入出力が可能ならコンソールインターフェースを用いることで、iii/win公式サポート対象以外の言語からもiii/winは呼び出せる可能性があることもお分かりいただけると思います。

前述のとおりコンソールインターフェースは.net の、System.Console.WriteLine 等でも使用できます。

また、下記の言語では、コンパイラメーカー様がiii/winの呼出機能をサポートされています。

Visual DATAFLEX　フレックスコンサルタント様

ACUE COBOL　　東京システムハウス様

画面表示1

iii/winアプリケーションの画面には、SCREENコマンドで表示される親画面と、WINDOWコマンドで表示される子画面（子ウインドウ）の２種類があります。

親画面はアプリケーションの１つのプロセスに対して1つだけ持て、これは必須です。

子画面はあってもなくても構いません。１つの親画面に対し、複数個の子画面が持てます。

親画面も子画面も、iii／win画面フォームエディターPE.EXEで作成する段階では同じです。機能的には親画面と子画面で差はありません。

同じ画面情報ファイルを呼び出す時に、SCREENコマンドを使用すると親画面として動作し、WINDOWコマンドを使用すると子画面として動作します。

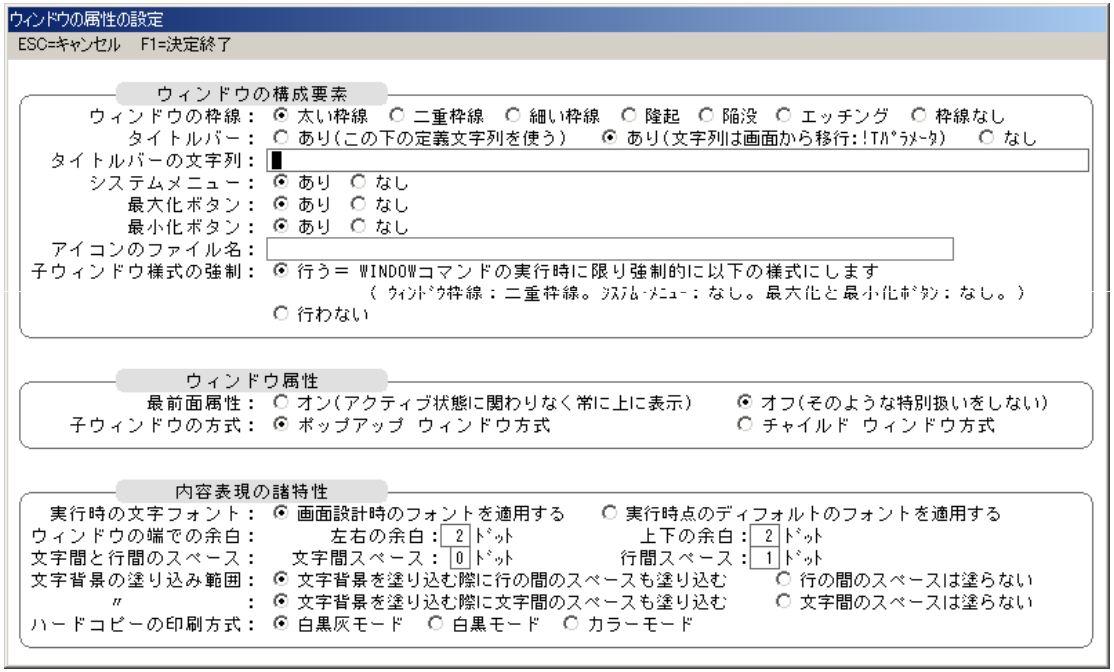
ただしWINDOWコマンドで表示される子画面にはデフォルトの「ポップアップ方式」と「チャイルドウインドウ方式」という2種類の表示方式が存在します。

ポップアップ方式では、親画面と子画面はOSレベルではどちらもデスクトップを親とする兄弟ウインドウです。ただしiii／winが強制的にアクティブ制御を行うことにより子画面の表示中は、親画面に制御が移動しないようになっていますが、親の表示領域の外に移動したり親のドラッグ動作には追随しないなどの特徴があります。

チャイルドウインドウ方式とは、SCREENで表示されている親画面のOSレベルでのチャイルドウインドウです。親をドラッグすると相対座標を維持したまま一緒に移動し、表示も親のボーダーラインでクリップされます。

画面表示2

[ウインドウ属性の設定]－＞[ウインドウ属性]－＞[子ウインドウの方式]でどちらの方式を選ぶかで、子ウインドウとしての動作が決定します。



一般に検索ウインドウや照会画面は、「ポップアップ方式」、あたかも親画面の一部領域（エリア）のように使いたい場合は「チャイルドウインドウ方式」を選択します。

画面表示3

もともとSCREEN（コマンド）は、DOS版においては物理画面全体に表示される親画面を指し、WINDOW（コマンド）は、既に親画面が表示されている状態で、入力項目の候補一覧の検索用や、項目が多い画面の補助入力画面、またはヘルプ画面などに使用される、物理画面の一部領域（矩形）だけを使用して親画面の上に重ねて描画される部分画面を指しています。iii/winでは当然ですがSCREENもWINDOWもWin32で生成するWindowです。

アプリケーションからみたiii/winの画面操作には、ごく一部の機能を除き、ウインドウハンドルという概念がありませんので、画面アクセスコマンドにいちいちハンドルをセットする必要がなく取り回しが大変楽になっています。

これはシングルタスクであるDOSの世界では、同時に複数の入力待ち画面が存在することはありえなかったことから来ています。iii／winの内部構造はマルチスレッドであり、1つのPSRUNプロセスは自分が生成した親画面や複数の子画面を全て管理しています。

しかしアプリケーションからは最後に表示したウインドウ（SCREEN、WINDOW）、または意図を持ってアクティブ化したウインドウ（WIN_ACTIVATED、WIN_CLOSE、RESTORE）が自動的にアクティブとなり、iii／winコマンドはそのアクティブ画面に対して作用します。

どうしても複数画面で同時に入力待ちを行う必要がある場合は、PSRUNを複数使用してマルチプロセスアプリケーションとすれば実現可能です。

画面表示4

親画面表示にはiiiコマンドSCREENを使用します。

C言語　　void psscrn(char* ScreenName);

VisualBasic　　sub PsScrn （ScreenName$）

COBOL　　display “SCREEN|ScreenName”

子画面の表示にはiiiコマンドWINDOWを使用します。

C言語　　void pswndw(char* ScreenName);

VisualBasic　　sub PsWndw （ScreenName$）

COBOL　　display “WINDOW|ScreenName”

SCREENコマンド、WINDOWコマンドには、ウインドウの体裁、位置、大きさなどに関する指定を行うパラメータが多く存在します。

例えば見えないウインドウ（不可視属性）を生成したければ

void psscrn(char* ScreenName); のScreenNameに画面情報ファイル名にパラメータ区切り符号 ‘|’とパタメータ‘H’を結合して渡します。

psscrn（”URI122.SCR|H”）;

画面の大きさ1

SCREENコマンドで表示される親画面は、iii/dosでは必ず物理画面の全体を使用して描画していました。

しかしWindowsでは親画面といえども必ずしも全画面表示が適しているとは限りません。そもそも画面の大きさの概念もWindowsでは物理的なものではなくデスクトップ自体が仮想画面です。その設計にあたっては使用するディスプレイの物理的大きさ、解像度、使用フォントのポイント数等も意識する必要があります。

WINDOWコマンドで呼び出される子画面の大きさはアプリケーションの目的により、さらに自由に決められる必要があるでしょう。

iii/winには、画面の作成時に「表示域」および「データ域」という2つの概念で画面の大きさを指定する機能があります。

デフォルトでは画面の大きさは「表示域」も「データ域」も横80文字×縦25行となっています。しかしこの範囲を超える座標位置にフィールドや固定文字などを配置すると自動的にその項目を含む範囲までがデータ域として確保されます。

物理的に作成できる画面の最大サイズは、横150文字×縦66行です。開発者はこの範囲の中で自由に画面を設計することができます。

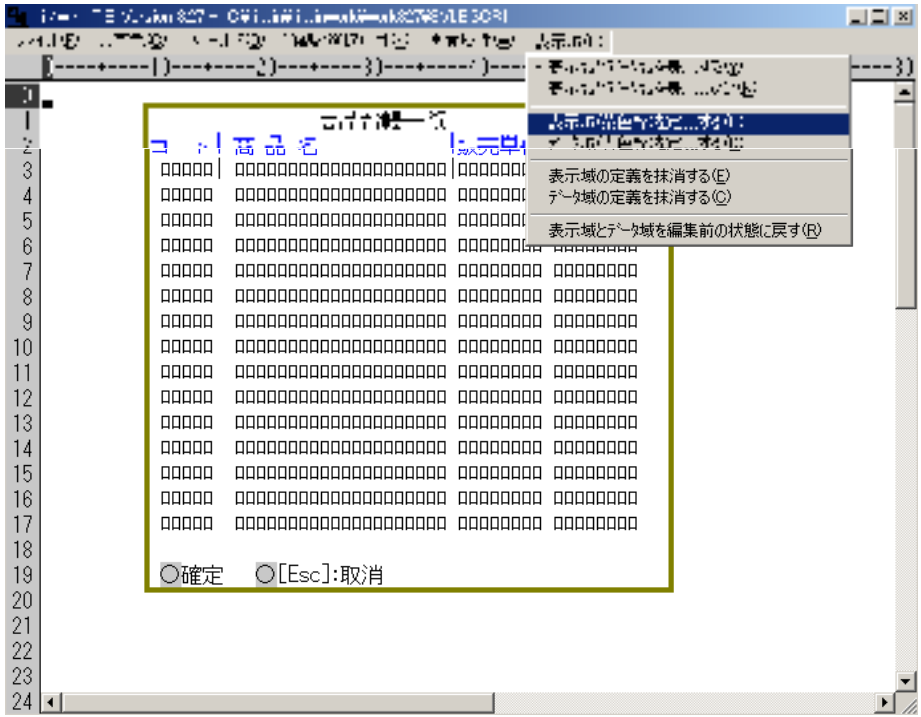
画面の大きさ2

設計した画面の一部が表示域としてははみ出しても、データ域の範囲内であれば画面としては有効です。実行時にははみ出た部分が操作できるように画面の縦横に自動的にスクロールバーが附加されスクロールが有効になり、オペレータが自由にスクロールバーを操作できるようになります。

スクロールに関するコーディングを、アプリケーション側で意識することは不要です。iii／winが自動で行います。

アプリケーションプログラムは常に設計した画面の全てが見えているという前提でコーディングでしてください。

実際には、見えている範囲からはみ出た場所のフィールドに入力がかかるとiii/winが自動的に画面全体をスクロールさせ入力状態のフィールドが見える位置まで画面を移動させ、オペレータの操作を可能にします。



iii/win の INIT:初期化 とQUIT:終了

説明の都合上話しが前後しますが、iii/winの使用開始にあたっては必ず初期化コマンド INIT 実行します。

VisulaC++ 及びVisualBasicでは、それぞれに用意された専用関数を使用します。

VisualC++ void psinit();

VisualBasic Sub PsInit ()

Cobolの場合は、iiiコマンド INIT に初期化タイプパラメータ ‘C’ を附加し、display命令で発行します。

display “INIT|C”

iii/winの使用終了時には必ず終了コマンド QUIT を実行します。

VisulaC++ 及びVisualBasicでは、それぞれに用意された専用関数を使用します。

VisualC++ void psquit();

VisualBasic Sub PsInit ()

Cobolの場合は、iiiコマンド QUIT をそのまま display命令で発行します。

display “QUIT”

フィールドへの値出力

画面への値の出力には、iiiコマンド PUT を使います。

C言語用では、変数の型べつに用意された関数を使用します。

void pscput(char* フィールド名、char 出力値);	フィールドへの一文字出力
void psiput(char* フィールド名、int 出力値);	フィールドへの整数出力
void pslput(char* フィールド名、int 出力値);	フィールドへの長整数出力
void psrput(char* フィールド名、int 出力値);	フィールドへの実数出力
void psfput(char* フィールド名、int 出力値);	フィールドへの実数出力(精度確保)
void pssput(char* フィールド名、int 出力値);	フィールドへの文字列出力

ちなみに配列フィールドに対し、配列番号をインクリメントしながらアクセスできる専用関数も各型別に存在します。

(例) void pssputn(char* 配列フィールド名、int 配列番号、int 出力値); 配列フィールドへの文字列出力

VisualBasicもC言語と同様です。

(例) Sub PsLPut (FieldName$,FiledData%) Sub PsRPutN (FiledName$,ArrayNo%,FieldData#)

Cobolの場合は、iiiコマンド PUT をそのまま使い、display命令で発行します。

(例) display “PUT|TOKUISAKI|大阪商事”

display “PUT|” [フィールド名] “|” [出力値]

※ 下段は、COBOLの変数を使用し文字列合成して出力する例です。[ ]が変数です。

フィールドからの値受取

画面からの値取得には、iiiコマンド GET を使います。

C言語用では、変数の型べつに用意された関数を使用します。

char pscget(char* フィールド名);	フィールドから一文字受取
int psiget(char* フィールド名);	フィールドへの整数受取
long pslget(char* フィールド名);	フィールドへの長整数受取
double psrget(char* フィールド名);	フィールドへの実数受取
double psfget(char* フィールド名);	フィールドへの実数受取（精度確保）
void pssget(char* フィールド名、char* フィールド値<レスポンス>);	フィールドへの文字列受取

ちなみに配列フィールドに対し、配列番号をインクリメントしながらアクセスできる専用関数も各型別に存在します。

（例） double psrgetn(char* 配列フィールド名、int 配列番号); 配列フィールドからの文字列受取

VisualBasicもC言語と同様です。

（例） Function PsRGet# (FieldName$) Sub PsSGetN (FiledName$,ArrayNo%,FieldData$)

Cobolの場合は、iiiコマンド GET にレスポンス要求を示す@を附加してdisplay命令で発行、acceptで受取ます。

（例） display “@GET|TOKUISAKI”
accept tokuisaki

iii／winコマンドについて

いくつかの代表的な iii／winコマンドを簡単に紹介しましたが、実際には大変多くのコマンドがあります。またC言語やVisualBasic用の言語インターフェースではダイレクトにiii/winコマンドを記述するのではなく、各言語用関数を用意し、言語の持つ変数の型べつにアクセスが簡単に行えるようにしています。

iii／winコマンドには、過去の製品との互換性から3つの系統があります。

1. **最初のiiiシリーズであるiii／dosの時から存在する基本コマンド**
2. **2つ目のiiiシリーズであるiii／2で追加されたコマンド**
3. **上記製品用には存在せず、iii／winで始めて追加されたコマンド**

2. と3. は主にOSの提供するGUIを意識したコマンドになります。

2. 3. では極力関数の数自体を増やさないようにしました。汎用関数SETとQUERYを用意し、実行時にそれぞれに与えるキーワードにより動作を変えるようにしています。

また、各言語の型を意識する関数は便利なようですが、標準サポート以外の言語での使用を希望される場合には、型の意識が仇になることもあります。そこで用意したのが汎用関数PsCmdです。CやVBからもCOBOLやSCDEB（スクリーンデバッガー）で使用するiii／winコマンドそのものでアクセスすることを可能にしました。

iii/winコマンド一覧 （その1）

iii/winコマンド（COBOL、SCDEB）	C言語関数	VisualBasic	注釈・機能
ABORTDA	psabda()	PsAbDa	アボート・キーの解除
ABORTEN	psaben()	PsAbEn	アボート・キー指定
	psabort()		入力コマンドの強制終了（マルチスレッド用）
ACLEAR	paaclr()	PsAClr	指定範囲のフィールド消去
AFILE	psafnm()	PsAfnm	アクティブファイル名の取得
ALOCK	psalck()	PsAlck	指定範囲のフィールド（選択項目）ロック
ALARM	psalrm()	PsAlrm	アラーム表示
ARRAYNO	psarno()	PsArNo%	フィールドの配列番号
AULOCK	psaulk()	PsAuLk	指定範囲のフィールド（選択項目）ロック解除
BRINK	psbrnk()	PsBrnk	フィールドのブリンク ＜dos互換用、winでは無動作＞
CLS	pschgt()	PsCls	画面消去
CODE	pscode()	PsCode$	フィールドの処理コード取得
DATESET	psdeset()	PsDSet	日付、時間のセット
	psergt()	PsErGt%	エラー番号受取
ERROR	pserr()	PsErr	エラー表示
CLEAR	psfclr()	PsFClr	フィールドクリア
COLOR	psfcol()	PsFCol	フィールド文字色変更（iii/dos互換色コード）
FNAMEF	psffnm()	PsFfnm	フィールド名を得る
FKEYDA	psfkda()	PsFkDa	メニュー項目（ファンクションキー）を無効にする
FKEYEN	psfken()	PsFkEn	メニュー項目（ファンクションキー）を有効にする
FOFF	psfkof()	PsFkOf	メニューバーの表示を消す
FON	psfkon()	PsFkOn	メニューバーの表示

iii/winコマンド一覧 （その2）

iii/winコマンド（COBOL、SCDEB）	C言語関数	VisualBasic	注釈・機能
FKEYDISP	psfkst()	PsFkSt	メニュー項目（ファンクションキー）表示内容の変更
FLENGTH	psflen()	PsFLen%	フィールド内文字数
FNAME	psfnnm()	PsFnnm	フィールド名を得る
GET	ps?get()	Ps?Get?	フィールド値の取得
ICCHECK	psichk()	PsIchk	入力未完了フィールドの検査
ICHANGED	pschnd()	PsChnd%	フィールドの変化テスト
INIT	psinit()	PsInit	イニシャライズ（初期化処理）
INPUT	psinpt()	PsInpt%	フィールド（イベント）入力
LOCK	pslock()	PsLock	フィールド（選択項目）ロック
OFFSET	psofst()	PsOfst%	フィールド（選択項目）番号、カーソル位置
PATH	pspath()	PsPath	画面情報ファイルのサーチパス指定
POP	ps?pop()	Ps?Pop	配列フィールドデータ上シフト（及び値セット）
POSITION	psposn()	PsPosn	フィールド（カーソル位置）ポジショニング
PREVALUE	ps?pgt()	Ps?pgt?	入力直前のフィールド値
PUSH	ps?push()	Ps?Push	配列フィールドデータ下シフト（及び値セット）
PUT	ps?put()	Ps?Put	フィールドへのデータ出力
QUIT	psquit()	PsQuit	クイット（終了処理）
RETORE	psrstr()	PsRstr	セーブされた画面の復帰
SAVE	pssave()	PsSave	画面のセーブ
SCREEN	psscrn()	PsScrn	親画面表示
SELECT	psslct()	PsSlct%	項目選択（番号返し）
SELECTS	psslcts()	PsSlctS%	項目選択（内容文字列返し）

iii/winコマンド一覧 （その3）

iii/winコマンド（COBOL、SCDEB）	C言語関数	VisualBasic	注釈・機能
SLSTAT	psslst()	PsSlSt%	システム行競合検査＜dos互換用、win無動作＞
UNBRINK	psubrk()	PsUBrk	ブリンク解除　＜dos互換用、winでは無動作＞
UNLOCK	psulck()	PsULck	選択項目のロック解除
WINDOW	pswndw()	PsWndw	名前によるウインドウ（子画面）表示
WINDOWN	pswndwn()	PsWndwN	番号によるウインドウ（子画面）表示
YNSELECT	pschgt()	PsChgt$	１文字選択入力
HcPrint	PsHcPrint()	PsHcPrint%	画面の印刷
FColor	PsFColor()	PsFColor	フィールドの色指定（新色コード体系用）
	PsFColorN()	PsFColorN	同上　配列フィールド用
SET	psset()	PsSet	キーワード（後述）と併用してコマンドを形成
QUERY	ps?query()	Ps?Query	キーワード（後述）と併用してコマンドを形成
	ita()		int型変数を文字列のポインターへ変換
	lta()		Long型変数を文字列のポインターへ変換
	PsCmd()	PsCmd$	SCDEB（COBOL）コマンド書式でのコマンド発行
	PsCallFunc（）		指定した関数をPSRUNのスレッド内で実行
	PsIniFile()	PsIniFile	起動パラメータ格納ファイルの指定
	PsTblAccess()		Psrun内部のフィールド定義情報読出、変更
	PsReadFieldTbl()		画面情報ファイルのフィールド定義読出

iii/winコマンド一覧 （その4）

iii/winコマンド（COBOL、SCDEB）	C言語関数	VisualBasic	注釈・機能
PsError	PsError()	PsError	エラーメッセージを表示
FKeyText	PsFkText()	PsFkText	メニュー項目のテキスト変更
LinePut	PsLinePut()	PsLinePut	指定行へのメッセージ表示
MsgBox	PsMsgBox()	PsMsgBox	モーダルメッセージボックスの表示
MsgPanel	PsMsgPanel()	PsMsgPanel	モードレスメッセージボックスの表示
		PsTrap	コマンドエラー発生時にVBのエラーを生成する

iii/winコマンド一覧 （その5）

iii/win（SET）コマンド	注釈・機能
BTN_TEXT	ボタンへの文字列設定
FLD_NOTHIDE	隠したくないフィールド群の登録
FLD_EMERGED	スクロールして指定のフィールドを出現させる
FONT	フォントの指定
IME_MODE	IME変換動作モードの指定
IME_ONOFF	IME強制ON/OFFの指定
KB_FOCUS	リストボックスへのキーボードフォーカス設定
LB_INSERT	リストボックスへのデータ投入時の挿入／置換モード設定
LB_FILEDATA	リストボックスへファイルからデータをロードする
LB_SELECTNO	リストボックスの指定行を選択状態にする
LB_TOPNO	リストボックスの最上行に表示するデータ行の指定
MN_CHECKED	チェックマークのON/OFF状態設定
MN_DISABLED	ディスエーブル状態の設定
MN_TEXT	メニュー項目テキストの設定

iii/winコマンド一覧 （その5）

iii/win（SET）コマンド	注釈・機能
TB_FLASH	タイトルバーのフラッシュ
TB_TITLE	タイトルバーの文字列設定
WIN_ICON	ウィンドウへのアイコンの設定
WIN_CLOSED	指定子ウィンドウの破壊
WIN_ACTIVATED	指定ウィンドウのアクティブ化とコマンド作用対象指定
WIN_POS	ウィンドウの位置とサイズの変更
WIN_POSAT	ウィンドウの表示位置の登録
WIN_SHOW	ウィンドウ可視状態の変更

iii/winコマンド一覧 （その6）

iii/win（QUERY）コマンド	注釈・機能
FB_Color	フィールド前景色の取得
FF_Color	フィールド背景色の取得
LB_COUNT	リストボックスのデータ件数問い合わせ
LB_SERCH	リストボックスのフィールドデータ検索
LB_SELECTNO	リストボックスの選択されているデータ行番号取得
LB_TOPNO	リストボックスの最上行に表示されているデータ行番号取得
MN_1STCHECKED	チェックされているメニュー項目IDの取得
MN_CHECKED	メニュー項目のチェックマークON/OFF状態の取得
MN_DISABLED	メニュー項目のディスエーブル状態の取得
MN_SELECTNO	選択されたメニュー項目のID取得
MN_TEXT	メニュー項目テキストの取得
MP_RESULT	メッセージパネル終了コードの取得
SEL_MAXNO	選択項目最大番号の取得
WIN_HANDLE	ウィンドウハンドルの取得

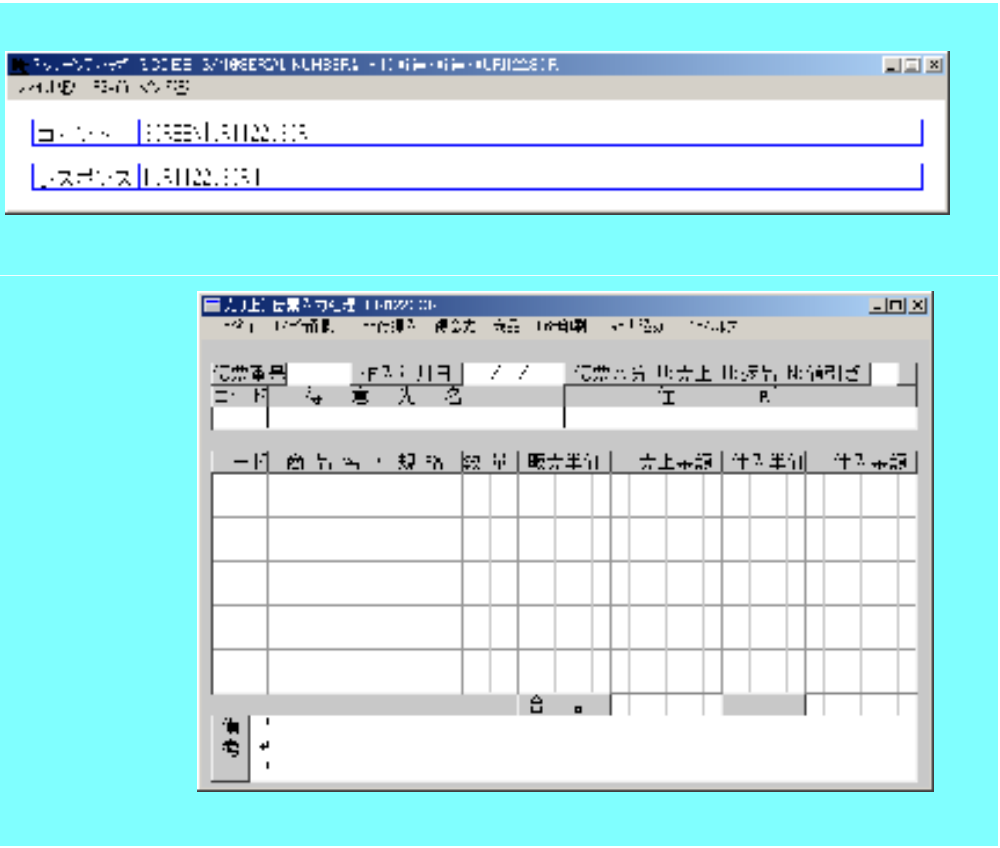
スクリーンデッバッガー（SCDEB）

SCDEBは、アプリケーションプログラムから切り離して、画面だけを単独で実行させるデバッグ用ツールです。

SCDEB.BATというバッチファイルを起動することで、画面プロセッサーPSRUNがデバッグモードで立ちあがります。

本来はアプリケーションが発行するiii／winコマンドを手動で発行することにより実際に画面動作をシミュレーションできます。

開発者が意図した通りの画面情報ファイルができているかどうかを確認できる、結構便利な機能です。



おわりに

「はじめに」でもお断りしているように、本資料はあくまでも概要書であり全ての機能説明を網羅しているものではありません。また本書はiii/winの製品の一部を構成するものでもありません。

もちろんマニュアルと重複する内容を含みますが、実際にiii/winが備えている機能全体から言えば、極々一部の説明しか過ぎません。

iii/winの全容や詳細をお知りになりたかったり、実際にiii/winを使った開発作業を行われる場合には、必ずiii/winユーザーズマニュアルをご覧下さい。

iii/winを皆様のシステム開発にご活用いただけますことを願っております。

お問い合わせ先

iii／winに関するご質問、ご要望、ご意見などを下記へお寄せください。

また、以下のURLにてホームページを公開しています。
こちらもご参照ください。

http://www.persimmon-system.co.jp/products/iiiwin/iiiwin_main.html

PERSIMMON SYSTEM

株式会社パーシモンシステム

〒542-0081 大阪市中央区南船場2丁目6番10号 ツチノビル4階

E-mail: iii@persimmon-system.co.jp TEL:06-6125-3510 FAX:06-6125-3511