MS OS／2 PM のプログラミングについて

行 方 常 幸

目　　次

1．はじめに

パーソナルコンピュータの OS である MS−DOS の世界では，今，操作の統一，絵を利用して視覚に訴えることによる使いやすさ，を目指したＧＵＩ（グラフィカル　ユーザ　インターフェース）の話で持ち切りである。今年はじめの MS−Windows（Ver. 3）発表以来，巷では MS−Windows 一色の感があるが，ここではシングルユーザマルチタスク OS の MS OS／2 の GUI である PM（プレゼンテーションマネージャ）のプログラミング方法を具体例を通して述べる。取り上げた具体例は「フルスクリーン用プログラムの始動」（STARTFUL. EXE）である。この具体例を取り上げた理由を以下に述べる。PM 用のアプリケーションは手元にほとんどないため，私が OS／2 を利用するのは PM のプログラミング学習であるが，その際利用するエディタ，コンパイラ，リンカ，デバッガ等は PM 用のアプリケーションではなくフルスクリーン用のアプリケーションである。すなわち PM をアプリケーション

間の移動中継場所として利用しているわけである。PMの画面で目的のフルスクリーン用アプリケーションに移動する場合，一番簡単な方法は目的のアイコンをダブルクリックする（マウスボタンをすばやく2回押し離す）ことである。その際，アプリケーション毎にアイコンが異なっていないと話にならないわけであるが，残念ながら通常の方法でフルスクリーン用アプリケーションを始動させると，皆同じアイコンになってしまう。しかし幸いなことに「ソフトウェア開発支援ツール」には「アイコンエディタ」が添付されており，自分の好きなようにアイコンを作成できる。あとはアプリケーションとアイコンを対応させればよいのだが，日本語MS OS／2に添付されているアプリケーション始動プログラム「プログラムの始動」では対応させることができない。そこでこの「プログラムの始動」の部分機能を持ちアイコンを対応させる「フルスクリーン用プログラムの始動」(STARTFUL.EXE) を作成した。

2．PMプログラミング

PM（プレゼンテーションマネージャ）用のプログラムを作成するには次のものが必要である。

（1）　Cコンパイラ

（2）　ソフトウェア開発支援ツール

（2）にはプログラミングの際にインクルードするヘッダファイル，リンクするときに必要なインポートライブラリOS2.LIB，リソースコンパイラRC.EXE，アイコンを作るときに利用するICONEDIT.EXE等が含まれている。

エディタ等を利用して作成しなければならないファイルは，通常，プログラムソースファイル（拡張子.Cを付けておく），モジュール定義ファイル（拡張子.DEFを付けておく），リソース定義ファイル（拡張子.RCを付けておく），メイクファイル（拡張子なし），ヘッダファイル（拡張子.Hを付けておく），等である。

今回作成したファイル（STARTFUL.＊）を基にその内容を以下に簡単に説明する（付録を参照）。

［モジュール定義ファイル（STARTFUL.DEF）］

このファイルはリンク時に利用される。NAME文でこのプログラムがダイナミックリンクライブラリではなく普通のアプリケーションであることを示し，名前（STARTFUL）を定義し，WINDOWAPIでPM用アプリーケーションであることを宣言する。DESCRIPTION文で文字列をEXEファイルに埋め込み，PROTMODE文でプロテクトモード用アプリケーションであることを宣言する。HEAPSIZE，STACKSIZE文でヒープ，スタックの大きさを宣言する。スタックサイズはPM用アプリケーションの場合8キロバイトにすることが勧められている。

［ヘッダファイル（STARTFUL.H）］

プログラムを読みやすくするための定数を定義している。すなわち，STARTFUL.RC，STARTFUL.Cで利用されているメニュー，アイコン，ダイアログボックス等を区別する識別子が定義されている。

［メイクファイル（STARTFUL）］

コマンドラインで

A：＞MAKE STARTFUL＜CR＞

とすることにより日付をチェックしてコンパイル，リンクなどを実行させるファイルである。コンパイラオプション，リンカオプションの意味は次の通りである。

／Zp：構造体をなるべく詰めて格納させる

／ASw：スモールモデル，DS！＝SSを指示

／c：コンパイルのみで，リンクは行わない

／G2s：80286用のコードを生成し，関数呼出の際にスタックチェックを行わない

／W3：最も厳しい警告を生成させる

／align：16：セグメントを16バイト境界に並べる

［リソース定義ファイル（STARTFUL.RC)］

次節で述べる。

［プログラムソースファイル（STARTFUL.C)］

次節で述べるが，その骨格（SKELETON.C 図. 1）についてここで述べる。関数のうち Dev，Dos，Win で始まるものはあらかじめ用意されている MS OS／2 のシステム関数である。SKELETON.C を STARTFUL.DEF と同様のモジュール定義ファイルと共にコンパイル，リンクし，実行ファイル SKELETON.EXE を作り実行させると，中が塵だらけのウィンドウが PM

```
#define                 INCL_WIN
#include                <os2.h>

MRESULT CALLBACK ClientWndProc(HWND, USHORT, MPARAM, MPARAM);

int main(void)
        {
        static CHAR             szClientClass  = "クラス名";
        static ULONG    flFrameFlags = FCF_TITLEBAR | FCF_SYSMENU |
                                       FCF_SIZEBORDER | FCF_MINMAX |
                                       FCF_SHELLPOSITION | FCF_TASKLIST;
        HAB             hab;
        HMQ             hmq;
        HWND    hwndFrame, hwndClient;
        QMSG    qmsg;
        hab = WinInitialize(0);
        hmq = WinCreateMsgQueue(hab, 0);
        WinRegisterClass(hab, szClientClass, ClientWndProc, 0L, 0);
        hwndFrame = WinCreateStdWindow(HWND_DESKTOP, WS_VISIBLE, &flFrameFlags,
                        szClientClass, NULL, 0L, NULL, 0, &hwndClient);
        WinSendMsg(hwndFrame, WM_SETICON,
                WinQuerySysPointer(HWND_DESKTOP, SPTR_APPICON, FALSE), NULL);
        while (WinGetMsg(hab, &qmsg, NULL, 0, 0))
                WinDispatchMsg(hab, &qmsg);
        WinDestroyWindow(hwndFrame);
        WinDestroyMsgQueue(hmq);
        WinTerminate(hab);
        return 0;
        }

MRESULT CALLBACK ClientWndProc(HWND hwnd, USHORT msg, MPARAM mp1, MPARAM mp2)
        {
        return WinDefWindowProc(hwnd, msg, mp1, mp2);
        }
```

図. 1　プログラムソースファイル（SKELETON. C)

画面に作成される（図. 2参照)。SKELETON.C は 2 つの関数から構成されている。main 関数では最低限次のことを行う。ウィンドウ作成の下準備(WinInitialize, WinCreateMsgQueue, WinRegisterClass)，ウィンドウの作成(WinCreateStdWindow)，メッセージループ（while 文のところ)，ウィンドウ破棄の後始末（WinDestroyWindow, WinDestroyMsgQueue, WinTerminate）である。この SKELETON.C では以上の処理に加えてメッセージループに入る前に，アイコンをシステムから借りてきている。もう 1 つの関数 ClientWndProc（ウィンドウプロシージャと呼ばれる）が実際上のプログラムの本体である。このウィンドウプロシージャにウィンドウの動作を記述する。次節で述べる STARTFUL.C ではこの ClientWndProc に後で述べるような動作を記述している。SKELETON.C では何もせずにただデフォルトのウィンドウプロシージャ WinDefWindowProc に全てを任しているため，図. 2 のようにただウィンドウ枠を表示しただけで仕事は何もしない，ウィンドウ枠内の塵でさえ消去しない。

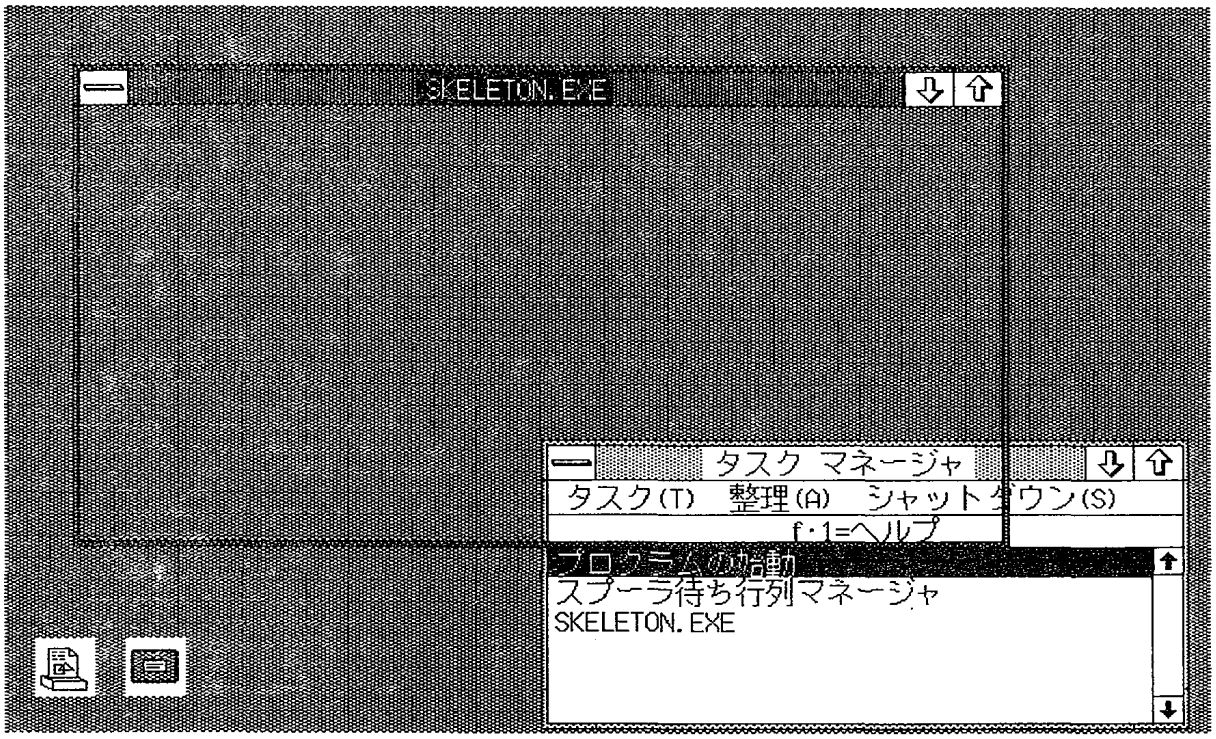

図. 2 SKELETON.EXE を実行したところ

図. 2をもう少し詳しく見ると,ウィンドウの上部に「SKELETON.EXE」とタイトルがあり，その左にシステムメニューボックスと呼ばれる真中に横棒の入った箱があり，右端には下向き矢印（最小化ボックス），上向き矢印（最大化ボックス）がある。更に，タスクマネージャのウィンドウを見ると「SKELETON.EXE」がリストに載っている。これらを実現させるのはSKELETON.Cのmain関数内でウィンドウ作成時に呼び出すWinCreateStdWindowの3番目の引数flFrameFlagsのFCF_TITLEBAR，FCF_SYSMENU，FCF_MINMAX，FCF_TASKLISTである。更に，最初のウィンドウの位置と大きさをシステムに適当に決めてもらい，ウィンドウのサイズを変更できるようにするために，FCF_SHELLPOSITION，FCF_SIZEBORDERとしている。次節で述べるSTARTFUL.Cはmain関数の少しの修正と，ClientWndProcでの仕事の記述が大部分である。

3．フルスクリーン用プログラムの始動（STARTFUL.EXE）について

「はじめに」の所でも述べたがSTARTFUL.EXEの目的はフルスクリーン用アプリケーションをアイコンと対応させて始動させることであるが，これは案外簡単で,始動させたいファイル名,必要ならそれへの引数,タイトル名,アイコンファイル名等を指定してシステム関数DosStartSessionを呼び出せば良い。あとはこれをウィンドウ内で操作のしやすいようにプログラムするだけである。まずMS OS／2に添付している「プログラムの始動」を参考にしてみる（図. 3参照)。タイトルの下にメニュー(「プログラム（P)」,「グループ（G)」,「ヘルプ」）があり，その下にグループ名，その下に，リストボックスがある。このリストボックス内で，例えば，マウスでダブルクリックすると，その時選択されていた項目のプログラムを始動させる。従って,「フルスクリーン用プログラムの始動」（STARTFUL.EXE）の外観を同じようにする（図. 4参照）。図. 4のウィンドウで枠をフレームウィンドウという。フレームウィンドウの中にシステムメニューボックス，タイトルバー，最小化最大化ボックスがあり，その下にメニューがあり，その下の部分がクライアント

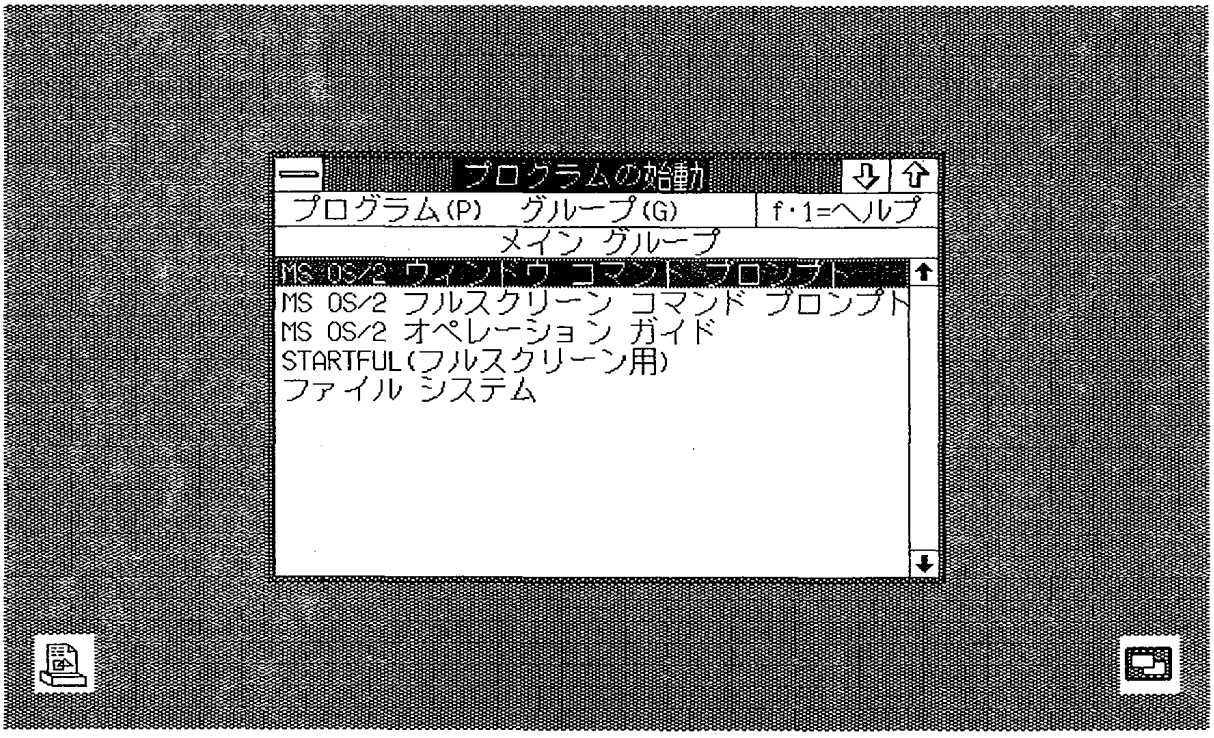

図. 3 ＰＭ に添付されている「プログラムの始動」

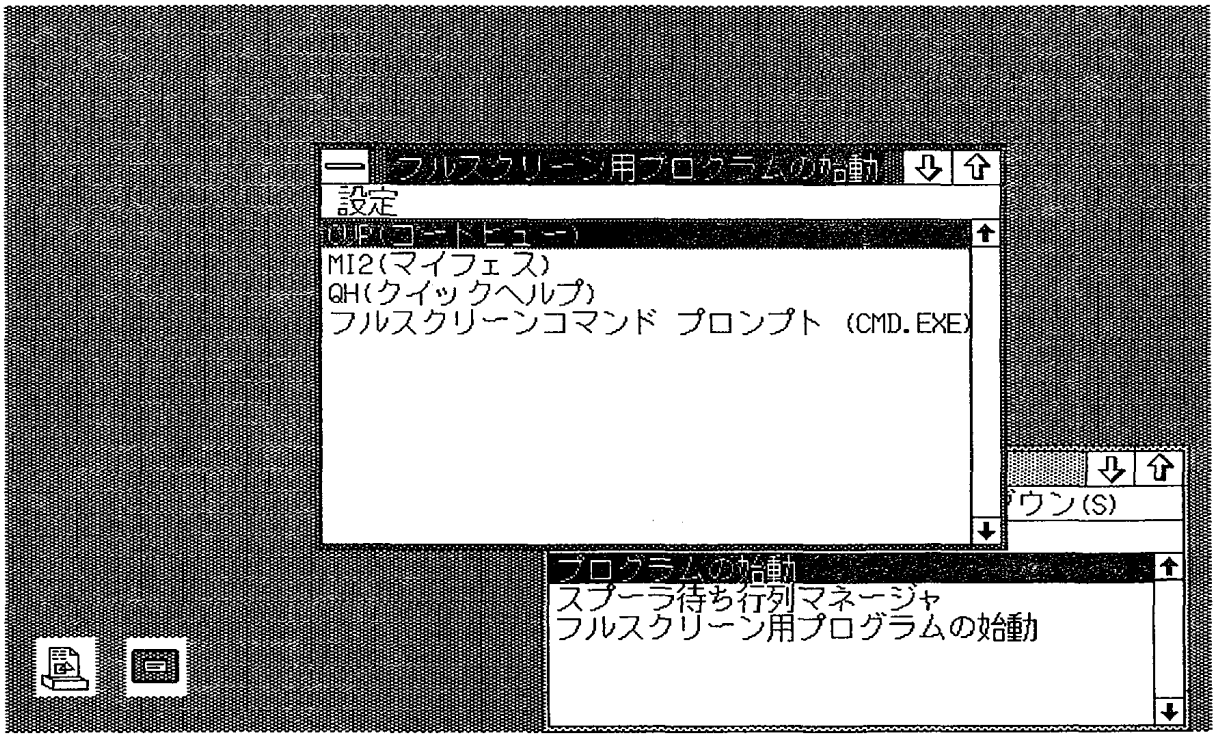

図. 4 「フルスクリーン用プログラムの始動」を実行したところ

ウィンドウである。図. 4ではクライアントウィンドウいっぱいにリストボックスが表示されている。また，STARTFUL.EXEの機能は次のような必要最小限のものを持たせることにする。「設定」メニューから「プログラムの追加，変更，削除」を選ぶことにより，始動すべきプログラムをリストボックスへ追加，変更，削除する(図. 5～8参照)。具体的には，STARTFUL.EXE始動時に図. 4のようなリストボックスを作成し，リストに載せるプログラムをOS2.INIから転記する。図. 4のリストボックスにはすでに4個プログラムが入っているが，1番最初の始動時にはOS2.INIにはプログラムリストはないので，上の3つはなく初期設定のCMD.EXEだけである。その後はユーザからの入力を待つ。「CVP（コードビュー)」の所にあるカーソルを矢印キーやマウスで移動し，始動したいプログラムをマウスでダブルクリックするか，リターンキーを押すことにより選択すると，リストボックスからWM__CONTROLメッセージが送られるので，選択されたプログラムを調べ，それを始動させる。図. 4でリストボックスの1番下の項目を選び，「設定」メニューを

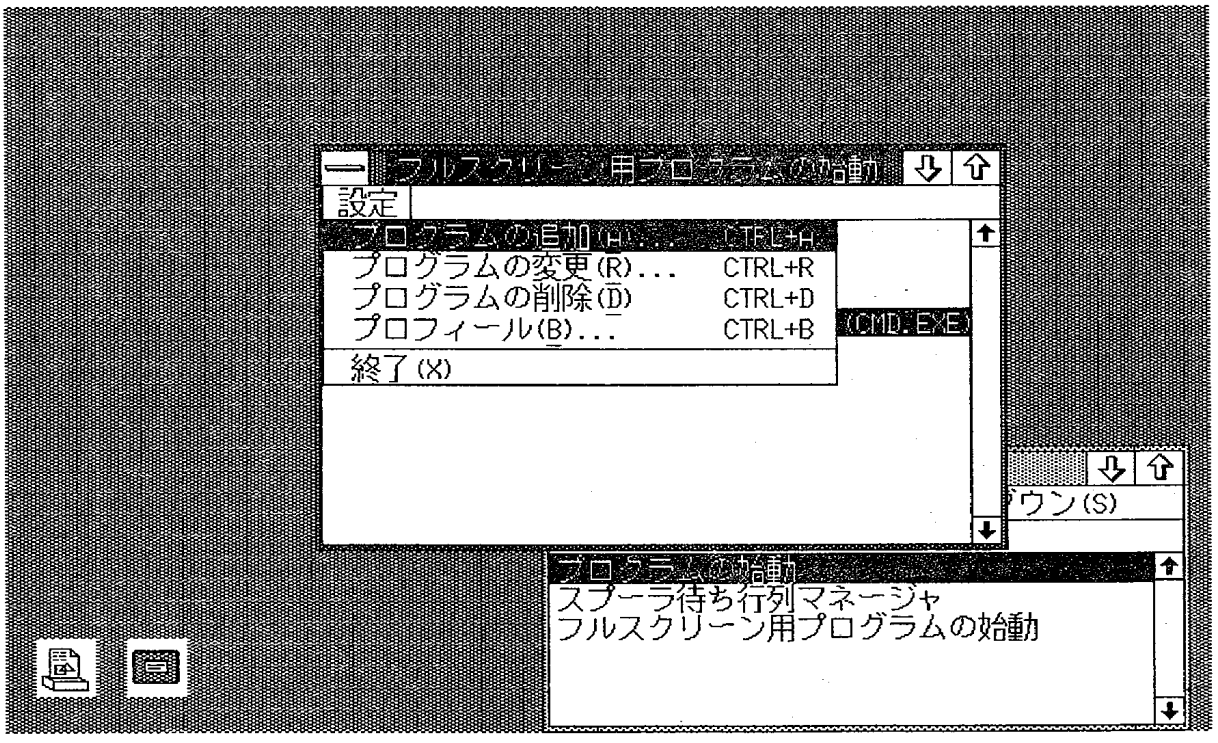

図. 5　「設定」メニューを選んだところ

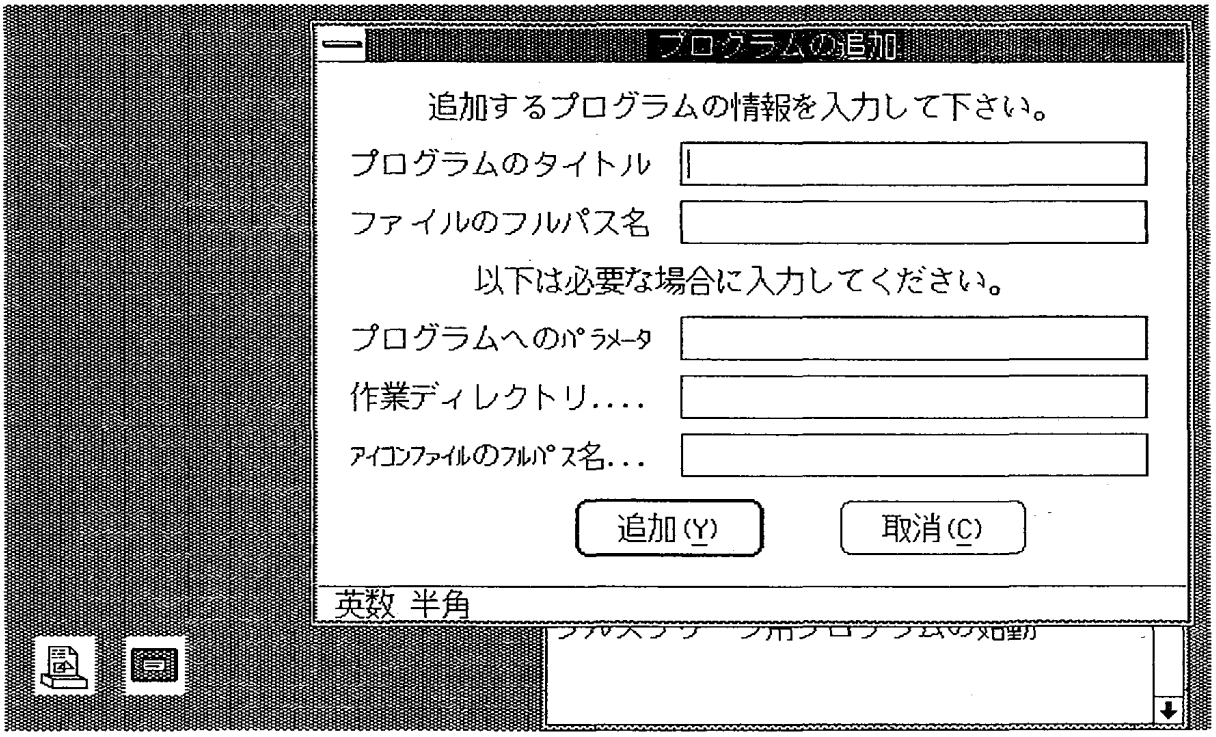

図. 6 「プログラムの追加」を選んだところ

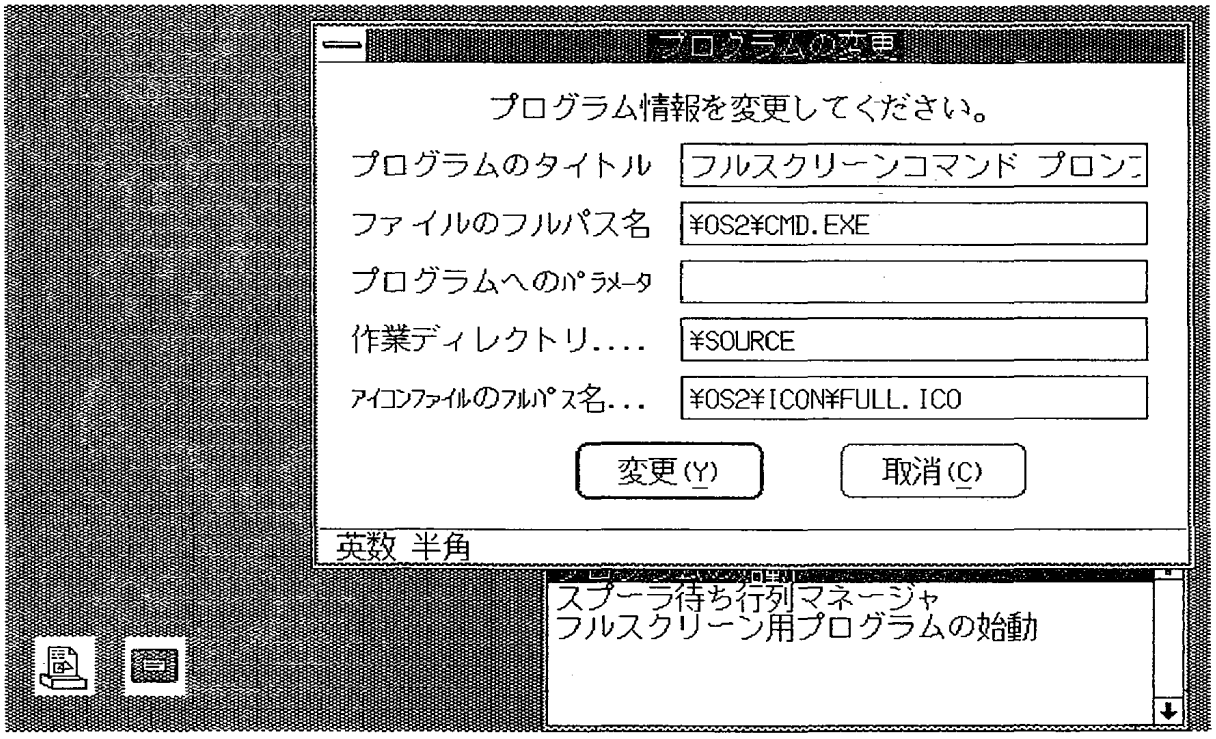

図. 7 「プログラムの変更」を選んだところ

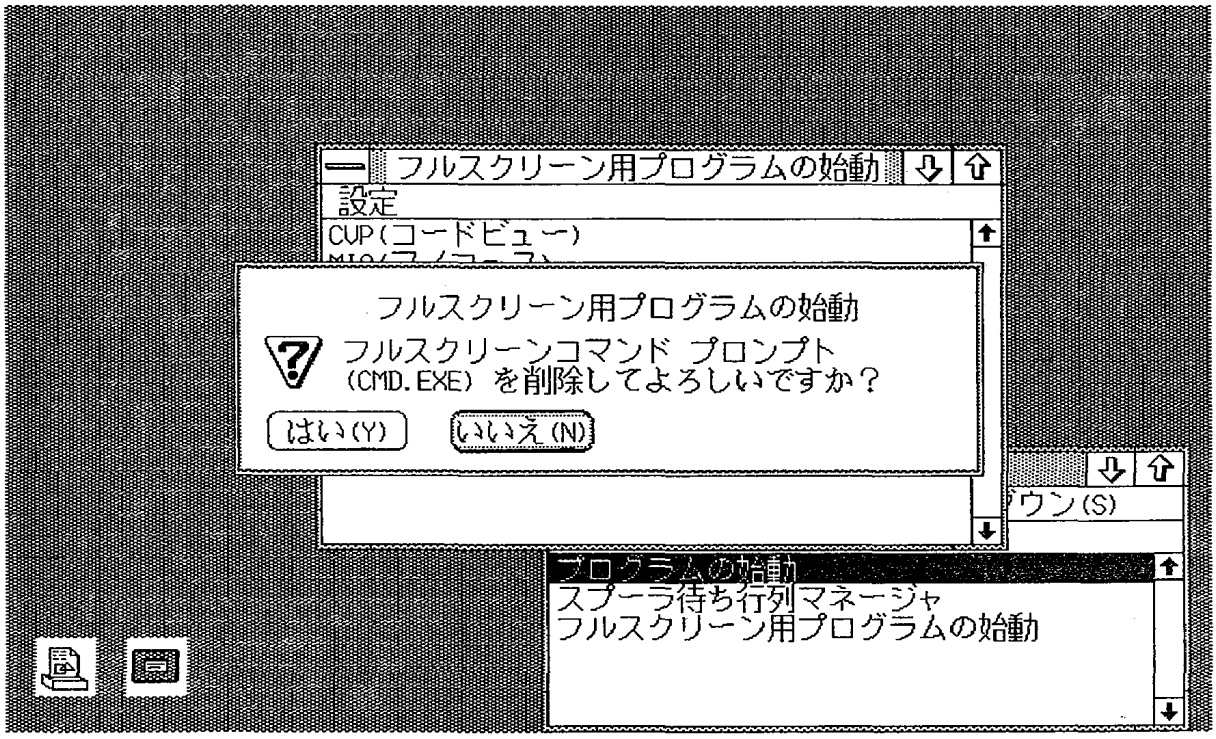

図. 8 「プログラムの削除」を選んだところ

選択したのが図. 5である。図. 5では「プログラムの追加，変更，削除，プロフィール，終了」が選択できる。これら5つのメニュー項目が選択されるとWM_COMMANDメッセージが送られるので，ダイアログボックス等を表示してその項目に必要な情報をユーザから獲得する。「プロフィール（B)」を選択すると，図. 9のような簡単なプログラムの説明をしたダイアログボックスを表示させる。「終了（X)」を選択すると，図. 10のようなメッセージボックスで動作の確認をする。「プログラムの追加（A)」を選択すると，図. 6のようなダイアログボックスを表示する。1行目のタイトルは図. 4のリストに表示されるもので，2行目以下はシステム関数DosStartSessionの引数に利用される情報である。「プログラムの変更（R)」を選択すると，図. 7のようなダイアログボックスを表示する。「プログラムの削除（D)」を選択すると，図. 8のようなメッセージボックスで確認する。図. 4のリストボックスにある4つのプログラムを始動後，PMの画面へ戻り自分も最小化したのが図. 11である。画面下方の左から4個目から7個目までのアイコンがアイコンエディタ

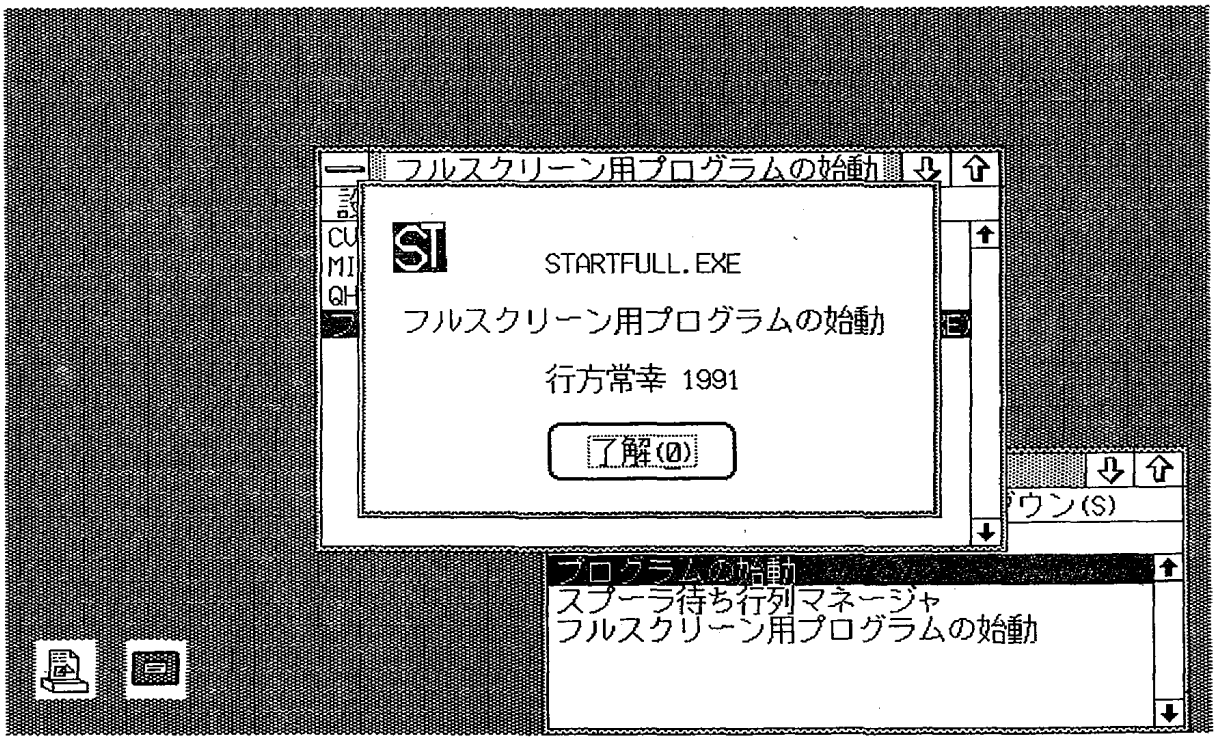

図. 9 「プロフィール」を選んだところ

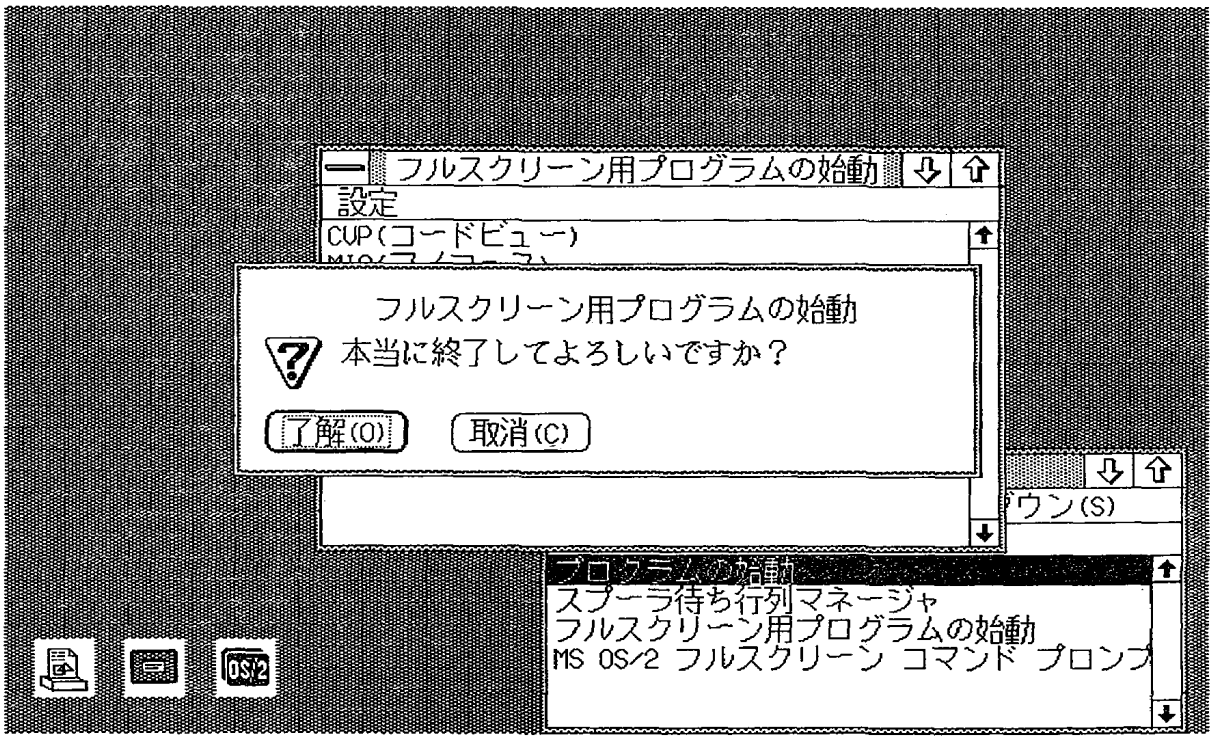

図. 10 「終了」を選んだところ

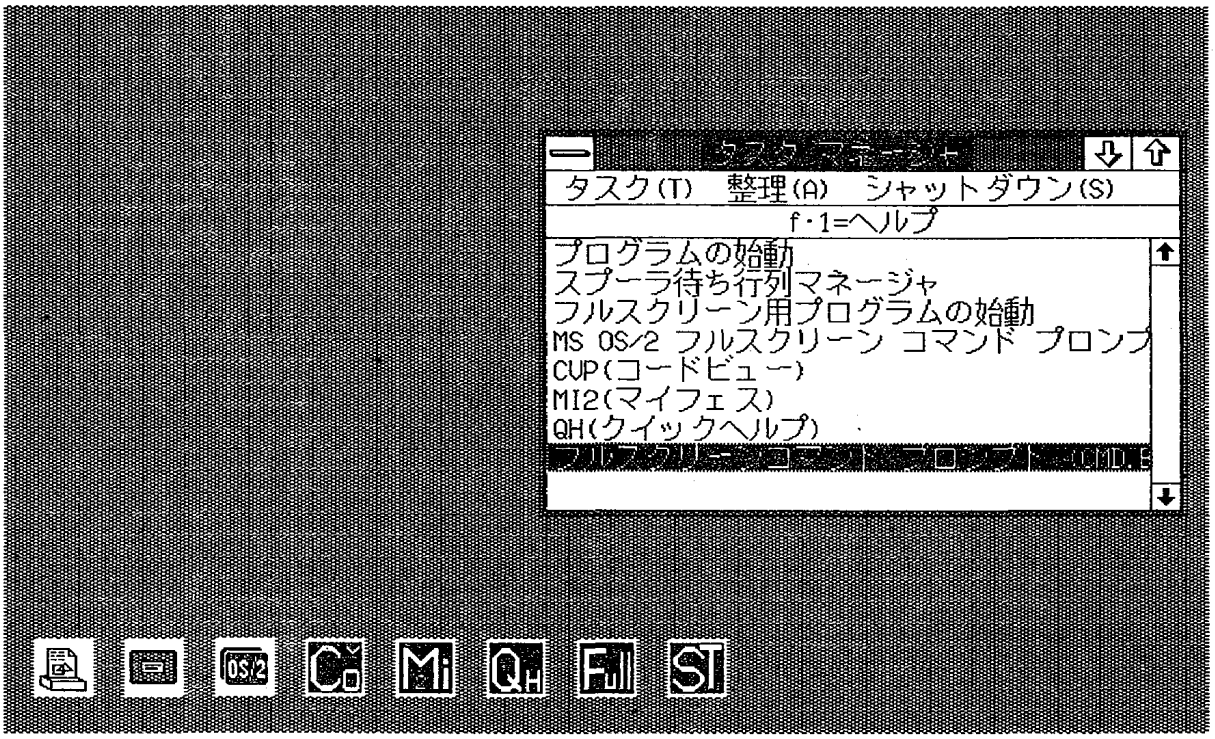

図.11　アイコンを表示させたところ

(ICONEDIT.EXE) で作成して，プログラムの追加の項目「アイコンファイルのフルパス名」で指定したものである。左から3個目のアイコンは「プログラムの始動」でフルスクリーン用アプリケーションを始動した時に現れるものである。

以上の機能を実現するための主な部分がプログラムソースファイル (STARTFUL.C)，リソース定義ファイル (STARTFUL.RC) である。

STARTFUL.RC ではアイコン，メニュー，ダイアログボックス等の画面上のスタイル等を定義し，識別子を利用して STARTFUL.C で参照可能なようにしている。まず POINTER 文を利用して STARTFUL.EXE のアイコン（図. 11の左から8個目のアイコン）を ID__RESOURCE で参照できるように定義している。ここで startful. ico は ICONEDIT. EXE で作成したファイルである。次に，MENU 文でメニューを定義し ID__RESOURCE で参照できるようにしている。これで，図. 4，図. 5 のようなメニューが出来上がる。例えば，「プログラムの追加（A）」を選択すると，そのメニュー項目

を定義しているMENUITEM文に書かれているIDM__ADD__PROGがWM__COMMANDメッセージの一部としてウィンドウプロシージャ ClientWndProcへ送られる。このメッセージを受け取ったときに必要な処理を行うわけである。最後のACCELTABLE文で短縮キー入力を定義しID__RESOURCEで参照できるようにしている。例えば，CTRL+A（^A）とキー入力すればメニュー項目「プログラムの追加（A）」を選択したのと同じ効果（IDM__ADD__PROGを含むメッセージの送付）が得られる。これらアイコン，メニュー，アクセラレータテーブルはすべて同じ識別子ID__RESOURCEで参照されている。残りのDLGTEMPLATE文でIDD__ABOUT，IDD__ADD__PROG，IDD__REV__PROG，IDD__INPUT__PARAMの4個のダイアログボックスの形を定義している。DIALOG文で枠のフレームウィンドウを定義し，その中にさらにいくつかのウィンドウを定義している(CTEXT, LTEXT, ENTRYFIELD)。例えば，ENTRYFIELDはキー入力が可能なウィンドウを定義し，「IDD__...」はこのウィンドウの識別子で，その後の数字はこのウィンドウの位置と大きさを表している。ここでは，日本語の入力を可能にするためにはフレームウィンドウを定義したDIALOG文の所にFCF__DBE__APPSTATと指定しておくだけで良いことに注意する。さて，リソース定義ファイルSTARTFUL.RCでメニュー構造，各メニュー項目の識別子，またダイアログボックスの構造，ボタンなどのダイアログボックスに含まれるウィンドウの識別子等を定義したが，これを使って目的とする処理をプログラムしているのがSTARTFUL.Cである。

STARTFUL.Cのmain関数はSKELETON.Cのmain関数とほとんど変わらない。データの前処理，必要ならプロフィルデータの削除，さらに，終了時の確認（図. 10）をしている。図. 10のようなメッセージボックスは文字列，アイコン，ボタンを引数で指定してWinMessageBoxを呼び出すだけでよい。SKELETON.Cと大きく違う所はClientWndProcの内容と，この他に3つのウィンドウプロシージャ（AddRevProgDlgProcは図. 6，7のダイアログボックスの動作を記述している，AboutDlgProcは図. 9のダイア

ログボックスに対応）があることである。その他の関数は上記の4つのウィンドウプロシージャから呼び出される関数である。次に，プログラムの主要な部分を少し詳しく説明する。

STARTFUL.Cのmain関数内のWinCreateStdWindowで図. 4のウィンドウを作成している。SKELETON.Cと異なり第2引数でWS__VISIBLEを指定していない。第3引数でFCF__SHELLPOSITIONの代わりにFCF__MENU，FCF__ICON，FCF__ACCELTABLEを指定している。第8引数でID__RESOURCEと指定している。第2引数でWS__VISIBLEを指定するとウィンドウが画面に表示される。ここではウィンドウ作成中に少し細工したいので,作成時には非表示とし後(ウィンドウプロシージャClientWndProcの中)で表示させるようにする。第3引数でFCF__SHELLPOSITIONを指定しない代わりに後（ウィンドウプロシージャ中）でウィンドウの位置と大きさを指定する。STARTFUL.RCでメニュー，アイコン，アクセラレータテーブルを定義したが，これらを実際に組み込むには，第3引数にFCF__MENU，FCF__ICON，FCF__ACCELTABLEを指定し，さらに第8引数でこれらの識別子ID__RESOURCEを指定するだけでよい。

ウィンドウの動作を記述するウィンドウプロシージャは，switch…caseの構文で他から送付されたメッセージを処理する。これがメッセージ駆動型のプログラムと呼ばれる由縁で，送付元はシステムであったり，自分自身であったりする。また,メッセージの処理中にメッセージの送付も行う場合もある。図. 4のメニュー「設定」の下の部分(クライアントウィンドウ)の動作を記述するウィンドウプロシージャClientWndProcはWM__CREATE, WM__SIZE, WM__CONTROL，WM__COMMAND，WM__DESTROYの5つのメッセージを処理している。

[WM__CREATE]

WM__CREATEはウィンドウが作成される時に1度だけ送付されるメッセージで，ウィンドウの初期化の処理を行う。ここではタイトルを日本語で付ける，タスクマネージャのリストに日本語で載せる，リストボックスを作成す

る，フレームウィンドウの位置と大きさを決め，可視状態にする，リストボックスにインプットフォーカスを設定する，OS2.INIからリストボックスに載せるプログラムリストを読み込み（GetProfileData）セット（SetProgram List）する，である。以下でもう少し詳しく説明する。main関数でウィンドウをWinCreateStdWindowで作成するとき，FCF__TITLEBAR，FCF__TASKLISTを指定するとタイトルが付けられタスクマネージャのリストにプログラム名が載るが，通常では，これらは「STARTFUL.EXE」となってしまう。そこでこれらを「フルスクリーン用プログラムの始動」にするために少し工夫する。FCF__TASKLISTを指定せずにウィンドウを作成する。その後フレームウィンドウに日本語のタイトルを設定する（WinSetWindow Text)。タスクマネージャのリストに載せるために，必要なパラメータを設定して，WinAddSwitchEntry関数を呼ぶ。WinCreateWindow関数でリストボックスを作成し，以下識別子IDL__MYLISTで参照するようにしている。この時点ではリストボックスの位置と大きさは設定されていない。後のWM__SIZEの所で設定する。main関数でフレームウィンドウを作成したが，その時にはまだ位置と大きさを決めていなかったし，不可視状態であった。そこでフレームウィンドウの左下が画面の左下から1／4づつの位置にきて，日本語のタイトルが丁度入るように大きさを設定し（WinSetWindowPos)，可視状態にする（WinShowWindow)。矢印キーでカーソルがプログラム間を移動できるように，インプットフォーカスをリストボックスに設定する（Win SetFocus)。

[WM__SIZE]

WM__SIZEはウィンドウの大きさが変わった時に送付されるメッセージで，ここではすでに作成済みのリストボックスの大きさを修正している（Win SetWindowPos)。工夫した注意点はリストボックスの枠を表示させないため枠の巾の分だけ位置をずらし大きさも大きくしたことである。

[WM__CONTROL]

このメッセージの送付元がリストボックス（IDL__MYLIST）で，カーソ

ル位置のプログラムが選択された場合（LN__ENTER），そのプログラムを始動させる（StartSession)。

[WM__COMMAND]

このメッセージがメニュー項目の選択により送付された場合（IDM__ADD__PROG，IDM__REV__PROG，IDM__DEL__PROG，IDM__ABOUTの場合），ダイアログボックス等を作成して必要な処理を行う。例えば，「プログラムの変更（IDM__REV__PROG)」に対してはダイアログボックス作成時（WinDlgBox）にそのウィンドウプロシージャをAddRevProgDlgProc（第3引数），その構造をSTARTFUL.RCで識別子IDD__REV__PROGで参照可能にしたものにして（第5引数），処理をダイアログボックスのウィンドウプロシージャに任せている。

[WM__DESTROY]

このメッセージはウィンドウが破棄される時に送付されるもので，ウィンドウの後始末を行う。ここでは処理途中に獲得したメモリを返却し(MemFree)，タスクマネージャへの登録を破棄する（WinRemoveSwitchEntry)。

ダイアログボックスの動作を記述しているウィンドウプロシージャのうちAddRevProgDlgProcを説明する。このウィンドウプロシージャは図. 6，7の2つのダイアログボックスの処理を行っている。AddRevProgDlgProcは3つのメッセージWM__INITDLG，WM__CLOSE，WM__COMMANDを処理している。

[WM__INITDLG]

このメッセージは通常のウィンドウプロシージャのWM__CREATEと同様にダイアログボックスが作成される時に1度だけ送付されるメッセージである。ここでは，まずダイアログボックス内のキー入力を受け付けるENTRY-FIELDの長さを設定し，さらに「プログラムの変更(R)」のダイアログボックスの場合は現在のプログラムの情報をENTRYFIELDに設定している。

[WM__CLOSE]

システムメニューボックスの「クローズ（C)」を選択したときに送付され

るメッセージである。ここではWinDismissDlg関数を呼び出してダイアログボックスを破棄している。

[WM_COMMAND]

ここではこのメッセージの送付元がダイアログボックスの一番下の行にある「追加（変更）」,「取消」の場合だけ処理をしている。「追加（変更）」の場合，プログラムのデータをバッファにコピーして，追加（InsertProgram），または変更（ReviseProgram）の処理を行う。「取消」の場合はダイアログボックスを破棄する。

詳しくは述べないが,「プログラムの追加,変更,削除」の際にはEXEファイル，およびICOファイルが指定したディレクトリにあるか確認し，ない場合にはその由を伝え訂正させる。追加，変更，削除の度にプログラムリストをタイトルの1文字目でソートし，その結果をOS2.INIへ書き込む。等の処理をしている。

4．おわりに

PM用のエレガントではないが，少しは実際の役に立つプログラムを作成したのでその概要の説明を行った。利用したPMの部品はリストボックスとダイアログボックスである。ウィンドウプロシージャで処理しているメッセージはまだほんの少しである。このプログラムの拡張として，ヘルプ機能を付ける，MS OS／2の本領であるマルチスレッド化を行う，等が考えられる。冗長と思われるが，付録としてソースプログラムを添付した。

参 考 文 献

[1] PETZOLD, C. "PROGRAMMING THE OS/2 PRESENTATION MANAGER", Microsoft PRESS, 1989

[2] NEC版「日本語MS OS／2ソフトウェア開発支援ツール」マニュアル

付 録

```
*************************************
*モジュール定義ファイル(STARTFUL.DEF)*
*************************************

NAME STARTFUL WINDOWAPI
DESCRIPTION 'Start Program for Full Screen Mode (C) Tsuneyuki Namekata, 1991'
PROTMODE
HEAPSIZE        1024
STACKSIZE       8192
```

```
*****************************
*ヘッダファイル(STARTFUL.H)*
*****************************

#define VALID   0                       /* 変更不可 */

#define INVALID_PROGRAM_FILE        1
#define INVALID_WORKDIR             2   /* 順序の変更不可 */
#define INVALID_ICON_FILE           3   /* 連続していること */
#define MEM_ERROR                   4

#define ADD_DLG_TITLE         "プログラムの追加"
#define REV_DLG_TITLE         "プログラムの変更"
#define INPUT_PARAM_TITLE     "パラメータの指定"

#define PRG_NUMBER_MAX    32          /* <= 256 */
#define LENGTH            256
#define PRG_TITLE_MAX     60

#define IDL_MYLIST  1

#define ID_RESOURCE 1
#define IDM_ADD_PROG      10
#define IDM_REV_PROG      11
#define IDM_DEL_PROG      12
#define IDM_ABOUT           13

#define IDD_ADD_PROG      20    /* 順序の変更不可 */
#define IDD_REV_PROG      21    /* 連続していること */

#define IDD_ABOUT           22
#define IDD_INPUT_PARAM   23

#define IDD_PRG_TITLE       30
#define IDD_PRG_PATHNAME    31
#define IDD_PARAMETER         32
#define IDD_WORKDIR           33
#define IDD_ICON_PATHNAME    34
```

```
*************************
*メイクファイル(STARTFUL)*
*************************

startful.obj : startful.c startful.h
    cl /Zp /ASw /c /G2s /W3 startful.c

startful.res : startful.rc startful.ico startful.h
    rc -r startful

startful.exe : startful.obj startful.def
    link startful, /align:16, NUL, os2, startful
    rc -cp 932 startful.res

startful.exe : startful.res
    rc -cp 932 startful.res
```

```
*********************************
*リソース定義ファイル(STARTFUL.RC)*
*********************************

#define     INCL_NLS
#include    <os2.h>
#include    "startful.h"

POINTER     ID_RESOURCE    startful.ico
```

```
MENU   ID_RESOURCE
   {
   SUBMENU"設定", -1
       {
       MENUITEM " プログラムの追加(~A)...¥tCTRL+A", IDM_ADD_PROG
       MENUITEM " プログラムの変更(~R)...¥tCTRL+R", IDM_REV_PROG
       MENUITEM " プログラムの削除(~D)¥tCTRL+D", IDM_DEL_PROG
       MENUITEM " プロフィール(~B)...¥tCTRL+B", IDM_ABOUT
       MENUITEM SEPARATOR
       MENUITEM " 終了(~X)", SC_CLOSE, MIS_SYSCOMMAND
       }
    }

DLGTEMPLATE IDD_ABOUT
   {
   DIALOG "", -1, 10, 10, 150, 88, , FCF _DLGBORDER
       {
       CTEXT "STARTFULL.EXE", -1, 10, 64, 130, 8
       CTEXT "フルスクリーン用プログラムの始動", -1, 10, 48, 130, 8
       ICON ID_RESOURCE, -1, 8, 64, 0, 0
       CTEXT "行方常幸 1991", -1, 10, 32, 130, 8
       DEFPUSHBUTTON "了解(~O)", DID_OK, 50, 8, 50, 16
       }
    }

DLGTEMPLATE IDD_ADD_PROG
   {
   DIALOG ADD_DLG_TITLE, -1, 10, 10, 230, 144,, FCF_DLGBORDER | FCF_TITLEBAR | FCF_SYSMENU | FCF_DBE_APPSTA
       {
           CTEXT "追加するプログラムの情報を入力して下さい。", -1, 10, 128, 210, 8
           LTEXT "プログラムのタイトル", -1, 10, 112, 80, 8
           ENTRYFIELD "", IDD_PRG_TITLE, 100, 112, 120, 8, ES_MARGIN
           LTEXT "ファイルのフルパス名", -1, 10, 96, 80, 8
           ENTRYFIELD "", IDD_PRG_PATHNAME, 100, 96, 120, 8, ES_MARGIN
           CTEXT "以下は必要な場合に入力してください。", -1, 10, 80, 210, 8
           LTEXT "プログラムへのﾊﾟﾗﾒｰﾀ", -1, 10, 64, 80, 8
           ENTRYFIELD "", IDD_PARAMETER, 100, 64, 120, 8, ES_MARGIN
           LTEXT "作業ディレクトリ....", -1, 10, 48, 80, 8
           ENTRYFIELD "", IDD_WORKDIR, 100, 48, 120, 8, ES_MARGIN
           LTEXT "ｱｲｺﾝﾌｧｲﾙのﾌﾙﾊﾟｽ名...", -1, 10, 32, 80, 8
           ENTRYFIELD "", IDD_ICON_PATHNAME, 100, 32, 120, 8, ES_MARGIN
           DEFPUSHBUTTON "追加(~Y)", DID_OK, 70, 8, 50, 16, WS_GROUP
           CONTROL "取消(~C)", DID_CANCEL, 140, 8, 50, 16, WC_BUTTON, BS_PUSHBUTTON | WS_VISIBLE
       }
    }

DLGTEMPLATE IDD_REV_PROG
   {
   DIALOG REV_DLG_TITLE, -1, 10, 10, 230, 128,, FCF_DLGBORDER | FCF_TITLEBAR | FCF_SYSMENU | FCF_DBE_APPSTAT
       {
           CTEXT "プログラム情報を変更してください。", -1, 10, 112, 210, 8
           LTEXT "プログラムのタイトル", -1, 10, 96, 80, 8
           ENTRYFIELD "", IDD_PRG_TITLE, 100, 96, 120, 8, ES_MARGIN
           LTEXT "ファイルのフルパス名", -1, 10, 80, 80, 8
           ENTRYFIELD "", IDD_PRG_PATHNAME, 100, 80, 120, 8, ES_MARGIN
           LTEXT "プログラムへのﾊﾟﾗﾒｰﾀ", -1, 10, 64, 80, 8
           ENTRYFIELD "", IDD_PARAMETER, 100, 64, 120, 8, ES_MARGIN
           LTEXT "作業ディレクトリ....", -1, 10, 48, 80, 8
           ENTRYFIELD "", IDD_WORKDIR, 100, 48, 120, 8, ES_MARGIN
           LTEXT "ｱｲｺﾝﾌｧｲﾙのﾌﾙﾊﾟｽ名...", -1, 10, 32, 80, 8
           ENTRYFIELD "", IDD_ICON_PATHNAME, 100, 32, 120, 8, ES_MARGIN
           DEFPUSHBUTTON "変更(~Y)", DID_OK, 70, 8, 50, 16, WS_GROUP
           CONTROL "取消(~C)", DID_CANCEL, 140, 8, 50, 16, WC_BUTTON, BS_PUSHBUTTON | WS_VISIBLE
       }
    }

DLGTEMPLATE IDD_INPUT_PARAM
   {
   DIALOG INPUT_PARAM_TITLE, -1, 10, 10, 190, 64,, FCF_DLGBORDER | FCF_TITLEBAR | FCF_SYSMENU | FCF_DBE_APPSTAT
       {
           CTEXT "", IDD_PRG_TITLE, 10, 48, 170, 8
           LTEXT "パラメータ",-1, 10, 32, 40, 8
           ENTRYFIELD"", IDD_PARAMETER, 60, 32, 120, 8, ES_MARGIN
           DEFPUSHBUTTON "始動(~S)", DID_OK, 35, 8, 50, 16, WS_GROUP
           CONTROL "取消(~C)", DID_CANCEL, 105, 8, 50, 16, WC_BUTTON, BS_PUSHBUTTON | WS_VISIBLE
       }
    }

ACCELTABLE ID_RESOURCE
   {
   "^A", IDM_ADD_PROG
   "^R", IDM_REV_PROG
   "^D", IDM_DEL_PROG
```

```
   "^B", IDM_ABOUT
    }
*****************************************************************************************************************************

*****************************************************************************************************************************
*************************************
*プログラムソースファイル(STARTFUL.C)*
*************************************

#define     INCL_DOSSESMGR
#define     INCL_WIN
#define     INCL_GPI
#include    <os2.h>
#include    <math.h>
#include    <stdio.h>
#include    <string.h>
#include    <stdlib.h>
#include    "startful.h"

typedef struct _FULLSESSION
   {
    CHAR    *pData;
    USHORT  DataLength;
    CHAR    *ProgramTitle;
    CHAR    *PathName;
    CHAR    *Parameter;
    CHAR    *WorkDir;
    CHAR    *IconFileName;
    }  FULLSESSION;

typedef struct
   {
    CHAR    **ppChar;
    USHORT  Count;
    CHAR    Char 1 ;
    }  PROGRAMPROFILEDATA;

typedef struct
   {
    CHAR    *PgmInputs;
    USHORT  Index;
    USHORT  Len;
    }  INPUTPARAM;

typedef INPUTPARAM FAR *PINPUTPARAM;

typedef struct
   {
    CHAR    *Buffer;
    USHORT  Type;
    }  ADDREVPARAM;

typedef ADDREVPARAM FAR *PADDREVPARAM;

MRESULT CALLBACK ClientWndProc(HWND, USHORT, MPARAM, MPARAM);
MRESULT CALLBACK AddRevProgDlgProc(HWND, USHORT, MPARAM, MPARAM);
MRESULT CALLBACK AboutDlgProc(HWND, USHORT, MPARAM, MPARAM);
MRESULT CALLBACK InputParamDlgProc(HWND, USHORT, MPARAM, MPARAM);

CHAR    **MemFree(PROGRAMPROFILEDATA *pProfileData ,CHAR *pData);
BOOL    GetProfileData(VOID);
BOOL    WriteProfileData(USHORT usIndex);
BOOL    SetProgramList(VOID);
USHORT  SelectPosToArrayPos(USHORT usSelect);
USHORT  StartSession(HWND hwnd, CHAR *pchWorkBuf1, USHORT Buf1Len, CHAR *pchWorkBuf2, USHORT Buf2Len);
USHORT  AddRevDlgErrorMessage(HWND hwnd, USHORT usAddOrRev, USHORT usResult);
BOOL    IsExist(CHAR *pszName, USHORT usAttribute, CHAR ch);
USHORT  mystrupr(CHAR *string);
USHORT  IsValidProgramData(CHAR *pchProgramData);
USHORT  Memalloc(CHAR *pchBuffer, USHORT usBufLen, CHAR **pAlloc);
VOID    MakeFullSession(FULLSESSION *pFullSession, CHAR *pszProgramData, USHORT usDataLength);
USHORT  InsertProgram(CHAR *pszBuffer, USHORT usBufLen);
USHORT  ReviseProgram(SHORT sSelect, CHAR *pszBuffer, USHORT usBufLen);
VOID    Delete(USHORT usSelect);
BOOL    DeleteProgram(CHAR *pszBuf, USHORT BufLen);

HAB     hab;
HWND    hwndListBox;
CHAR    szClientClass   = "フルスクリーン用プログラムの始動";

struct
   {
    BYTE    Count;
    BYTE    StartPos;
```

```
    } ProgramCountPos = { 1, 1 } ;

BYTE    abProgramOrder PRG__NUMBER__MAX  =  { 0xFF, 0xFF } ;

CHAR    Program1   = "フルスクリーン コマンドプロンプト(CMD.EXE)"
                     "  "
                     "¥¥OS2¥¥CMD.EXE"
                     "  "
                     ""
                     "  "
                     "¥¥"
                     "  "
                     ""
                     "  ";

FULLSESSION aFullSession PRG__NUMBER__MAX  =
                {
                NULL,
                NULL,
                NULL,
                NULL,
                NULL,
                NULL,
                NULL,
                (PCHAR)Program1,
                sizeof(Program1),
                "フルスクリーン コマンドプロンプト(CMD.EXE)",
                "¥¥OS2¥¥CMD.EXE",
                "",
                "¥¥",
                ""
                } ;

CHAR    szAppName     = "STARTFUL";
CHAR    szKeyName0    = "ProgramCountPos";
CHAR    szKeyName1    = "abProgramOrder";
CHAR    szProgram 2   = "0";

PROGRAMPROFILEDATA  *pProfileData;
CHAR    szMainBuffer 4*LENGTH + PRG__TITLE__MAX + 6 ;

CHAR    **MemFree(PROGRAMPROFILEDATA *pProfileData ,CHAR *pData)
   {
    if (!pData)
        {
        if (!pProfileData)
            return NULL;
        MemFree((PROGRAMPROFILEDATA *)pProfileData->ppChar, NULL);
        free(pProfileData);
        return NULL;
        }
    else
        {
        CHAR **ppchTmp;
        if (!pProfileData)
            return NULL;
        if (pData != pProfileData->Char)
            {
            pProfileData->ppChar =
                MemFree((PROGRAMPROFILEDATA *)pProfileData->ppChar, pData);
            return (CHAR **)pProfileData;
            }
        if (!(--(pProfileData->Count)))
            {
            ppchTmp = pProfileData->ppChar;
            free(pProfileData);
            return ppchTmp;
            }
        return (CHAR **)pProfileData;
        }
    }

BOOL    GetProfileData(VOID)
      {
    USHORT  cCount, cSize, cTotalSize;
    SHORT   sIndex;
    CHAR    *pchTmp;
    cCount = sizeof(ProgramCountPos);
    WinQueryProfileData(hab, szAppName, szKeyName0, &ProgramCountPos, &cCount);
    if (!ProgramCountPos.Count)
       {
        ProgramCountPos.StartPos = 1;
        ProgramCountPos.Count = 1;
        return FALSE;
```

```
        }
    cCount = sizeof(abProgramOrder);
    WinQueryProfileData(hab, szAppName, szKeyName1, abProgramOrder, &cCount);
    sIndex = ProgramCountPos.StartPos;
    cTotalSize = 0;
    while (sIndex != 0xFF)
        {
        *szProgram = (CHAR)('0' + sIndex);
        cSize = 0;
        WinQueryProfileSize(hab, szAppName,     szProgram, &cSize);
        cTotalSize += cSize;
        sIndex = abProgramOrder sIndex ;
        }
    if (cTotalSize)
        {
        if (!(pProfileData = (PROGRAMPROFILEDATA *)malloc(cTotalSize + sizeof(pProfileDatappChar)
                                                          + sizeof(pProfileData->Count))))
        return FALSE;
    pProfileData->ppChar = NULL;
    pProfileData->Count = ProgramCountPos.Count;
    pchTmp = pProfileData->Char;
    sIndex = ProgramCountPos.StartPos;
    if (sIndex == 0xFF)
        ProgramCountPos.StartPos = 1;
    while (sIndex != 0xFF)
        {
        *szProgram = (CHAR)('0' + sIndex);
        cSize = 0;
        WinQueryProfileSize(hab, szAppName, szProgram, &cSize);
        cTotalSize = cSize;
        WinQueryProfileData(hab, szAppName,     szProgram, pchTmp, &cTotalSize);
        if (cTotalSize != cSize)
            return FALSE;
        MakeFullSession(&aFullSession sIndex , pchTmp, cSize);
        pchTmp += cTotalSize;
        sIndex = abProgramOrder sIndex ;
        }
    }
    return TRUE;
    }

BOOL    WriteProfileData(USHORT usIndex)
    {
    WinWriteProfileData(hab, szAppName, szKeyName0, &ProgramCountPos, sizeof(ProgramCountPos));
    WinWriteProfileData(hab, szAppName, szKeyName1, abProgramOrder, sizeof(abProgramOrder));
    *szProgram = (CHAR)('0' + usIndex);
    WinWriteProfileData(hab, szAppName, szProgram, aFullSession usIndex .pData, aFullSession usIndex .DataLength);
    return TRUE;
    }

BOOL    SetProgramList(VOID)
    {
    SHORT   sIndex;
    CHAR    *pszTitle;
    sIndex = ProgramCountPos.StartPos;
    while (sIndex != 0xFF)
        {
        pszTitle = aFullSession sIndex .ProgramTitle;
        WinSendMsg(hwndListBox, LM_INSERTITEM, MPFROM2SHORT(LIT_END, 0), MPFROMP(pszTitle));
        sIndex = abProgramOrder sIndex ;
        }
    if (ProgramCountPos.Count)
        WinSendMsg(hwndListBox, LM_SELECTITEM, NULL, MPFROMSHORT(TRUE));
    return TRUE;
    }

USHORT  SelectPosToArrayPos(USHORT usSelect)
    {
    USHORT  usIndex, usCount;
    for (usIndex = ProgramCountPos.StartPos, usCount = 0; usCount < usSelect; )
        {
        usIndex = abProgramOrder usIndex ;
        usCount++;
        }
    return usIndex;
    }

USHORT  StartSession(HWND hwnd, CHAR *pchWorkBuf1, USHORT usBuf1Len, CHAR *pchWorkBuf2, USHORT usBuf2Len)
    {
    static STARTDATA    stdata;
    INPUTPARAM          InputParam;
    USHORT              usIndex, usSelect, usDisk, cbPath;
    ULONG               ulDrives;
    CHAR                *pszTmp, *pszPath;
```

```
    PID                 pidTmp;
    USHORT              idSession, usReturn;
    if (usBuflLen < LENGTH)
        return FALSE;
    usSelect = (USHORT)WinSendMsg(hwndListBox, LM_QUERYSELECTION, NULL, NULL);
    usIndex = SelectPosToArrayPos(usSelect);
    DosQCurDisk(&usDisk, &ulDrives);
    cbPath = 0;
    DosQCurDir(usDisk, NULL, &cbPath);
    if (!(pszPath = malloc(cbPath)))
        return FALSE;
    DosQCurDir(usDisk, pszPath, &cbPath);
    pszTmp = aFullSession usIndex .WorkDir;
    if (pszTmp 0  != 0)
        {
        if (pszTmp 0  == '¥¥')
            DosChDir(&pszTmp 0 , 0L);
        if (pszTmp 1  == ':')
            {
            DosSelectDisk(pszTmp 0  - '@');
            DosChDir(&pszTmp 2 , 0L);
            }
        }
    stData.Length = sizeof(stData);
    stData.Related = FALSE;
    stData.FgBg = FALSE;
    stData.PgmTitle = aFullSession usIndex .ProgramTitle;
    strcpy(pchWorkBuf1 + 2, aFullSession usIndex .PathName);
    if (pchWorkBuf1 2  == '¥¥')
        {
        pchWorkBuf1 0  = (CHAR)(usDisk + '@');
        pchWorkBuf1 1  = ':';
        stData.PgmName = pchWorkBuf1;
        }
    else
        {
        stData.PgmName  = aFullSession usIndex .PathName;
        }
    pszTmp = strcpy(pchWorkBuf2, aFullSession usIndex .Parameter);
    if ((*pszTmp) && (pszTmp = strchr(pszTmp, '?')))
        {
        *pszTmp = 0;
        InputParam.Index = usIndex;
        InputParam.PgmInputs = pchWorkBuf2;
        InputParam.Len = usBuf2Len;
        if (!WinDlgBox(HWND_DESKTOP, hwnd, InputParamDlgProc, NULL, IDD_INPUT_PARAM, (PVOID)&InputParam))
            return FALSE;
        stData.PgmInputs = InputParam.PgmInputs;
        }
    else
        {
        stData.PgmInputs = aFullSession usIndex .Parameter;
        }
    stData.InheritOpt = 1;
    stData.SessionType = 1;
    stData.IconFile = aFullSession usIndex .IconFileName;
    usReturn = DosStartSession(&stData, &idSession, &pidTmp);
    DosSelectDisk(usDisk);
    DosChDir(pszPath , 0L);
    free((CHAR *)pszPath);
    return usReturn;
    }

USHORT  AddRevDlgErrorMessage(HWND hwnd, USHORT usAddOrRev, USHORT usResult)
    {
    static CHAR*    pszTitle 2  =  {
                        ADD_DLG_TITLE,
                        REV_DLG_TITLE
                        } ;
    static CHAR*    pszMsg 4  =  {
                        "プログラムファイル名 の設定が正しくありません！",
                        "作業ディレクトリ の設定が正しくありません！",
                        "アイコンファイル名 の設定が正しくありません！",
                        "メモリエラーです！") ;
    return WinMessageBox(HWND_DESKTOP, hwnd, pszMsg usResult - INVALID_PROGRAM_FILE ,
                            pszTitle usAddOrRev - IDD_ADD_PROG , 0, MB_OKCANCEL | MB_ICONEXCLAMATION);
    }

BOOL    IsExist(CHAR *pszName, USHORT usAttribute, CHAR ch)
    {
    CHAR        szBuffer LENGTH ;
    CHAR        *pszTmp;
    HDIR        hdir = HDIR_CREATE;
    USHORT      usSearchCount = 1;
```

```
    FILEFINDBUF     findbuf 1 ;
    pszTmp = strcpy(szBuffer, pszName);
    if ((pszTmp 0  != '¥¥') && (!((pszTmp 1  == ':') && (pszTmp 2  == '¥¥'))))
        return FALSE;
    if (usAttribute == FILE_DIRECTORY)
        {
        USHORT usIndex;
        for (usIndex = 0; pszTmp usIndex ; usIndex++);
        if (pszTmp usIndex - 1  == '¥¥')
           {
            pszTmp usIndex++  = '*';
            pszTmp usIndex  = 0;
            }
        }
    else if (usAttribute == FILE_NORMAL)
       {
        USHORT  usIndex;
        CHAR    charTmp;
        BOOL    fExtension = FALSE;
        for (usIndex = 0; charTmp = pszTmp usIndex ; usIndex++)
            {
            if ((charTmp == '*') | | (charTmp == '?'))
                return FALSE;
            if (charTmp == '.')
                {
                fExtension = TRUE;
                break;
                }
            }
        if (fExtension)
           {
            if (ch == 'E')
               {
                if (pszTmp ++usIndex  != 'E')
                    return FALSE;
                if (pszTmp ++usIndex  != 'X')
                    return FALSE;
                if (pszTmp ++usIndex  != 'E')
                    return FALSE;
                }
            else if (ch == 'I')
               {
                if (pszTmp ++usIndex  != 'I')
                    return FALSE;
                if (pszTmp ++usIndex  != 'C')
                    return FALSE;
                if (pszTmp ++usIndex  != 'O')
                    return FALSE;
                }
            else
                return FALSE;
            }
        }
    else
        return FALSE;
    if (DosFindFirst(pszTmp, &hdir, usAttribute, findbuf, sizeof(findbuf), &usSearchCount, 0L))
       {
        DosFindClose(hdir);
        return FALSE;
        }
    else
       {
        DosFindClose(hdir);
        return TRUE;
        }
    }

USHORT  mystrupr(CHAR *string)
   {
    USHORT  usLength;
    BYTE    *str;
    for (str = string, usLength = 0; *str; str++, usLength++)
        {
        if (((*str >= 0x81) && (*str <= 0x9F)) | | ((*str >= 0xE0) && (*str <= 0xFC)))
            {
            if (*(++str) == 0)
                break;
            else
                usLength++;
            }
        else if ((*str >= 'a') && (*str <= 'z'))
            *str += 'A' - 'a';
        }
    return usLength;
```

```
    }

USHORT  IsValidProgramData(CHAR *pchProgramData)
    {
    CHAR    *pszTmp;
    pszTmp = pchProgramData;
    for (; *pszTmp; pszTmp++);
    pszTmp++;
    if (!IsExist(pszTmp, FILE_NORMAL, 'E'))
        return INVALID_PROGRAM_FILE;
    for (; *pszTmp; pszTmp++);
    ++pszTmp;
    for (; *pszTmp; pszTmp++);
    pszTmp++;
    if (!(*pszTmp))
        return VALID;
    if (!IsExist(pszTmp, FILE_DIRECTORY, ' '))
        return INVALID_WORKDIR;
    for (; *pszTmp; pszTmp++);
    pszTmp++;
    if (!(*pszTmp))
        return VALID;
    if (!IsExist(pszTmp, FILE_NORMAL, 'I'))
        return INVALID_ICON_FILE;
    return VALID;
    }

USHORT  Memalloc(CHAR *pchBuffer, USHORT usBufLen, CHAR **pAlloc)
    {
    PROGRAMPROFILEDATA      *pProfileDataTmp0;
    PROGRAMPROFILEDATA      *pProfileDataTmp1;
    USHORT                  usReturn;
    if (usReturn = IsValidProgramData(pchBuffer))
        return usReturn;
    if (!(pProfileDataTmp0 = (PROGRAMPROFILEDATA *)malloc(usBufLen + sizeof(pProfileDataTmp0->ppChar)
                                                                   + sizeof(pProfileDataTmp0->Count))))
        return MEM_ERROR;
    pProfileDataTmp0->ppChar = NULL;
    pProfileDataTmp0->Count = 1;
    if (!pProfileData)
        {
        pProfileData = pProfileDataTmp0;
        }
    else
        {
        for (pProfileDataTmp1 = pProfileData; pProfileDataTmp1->ppChar;
                                pProfileDataTmp1 = (PROGRAMPROFILEDATA *)pProfileDataTmp1->ppChar);
        pProfileDataTmp1->ppChar = (CHAR **)pProfileDataTmp0;
        }
    *pAlloc = pProfileDataTmp0->Char;
    memcpy(*pAlloc, pchBuffer, usBufLen);
    return usReturn;
    }

VOID    MakeFullSession(FULLSESSION *pFullSession, CHAR *pszProgramData, USHORT usDataLength)
    {
    CHAR    *pszTmp;
    pFullSession->pData = pszProgramData;
    pFullSession->DataLength = usDataLength;
    pFullSession->ProgramTitle = pszTmp = pszProgramData;
    for (; *pszTmp; pszTmp++);
    pFullSession->PathName = ++pszTmp;
    for (; *pszTmp; pszTmp++);
    pFullSession->Parameter = ++pszTmp;
    for (; *pszTmp; pszTmp++);
    pFullSession->WorkDir = ++pszTmp;
    for (; *pszTmp; pszTmp++);
    pFullSession->IconFileName = ++pszTmp;
    }

USHORT  InsertProgram(CHAR *pszBuffer, USHORT usBufLen)
    {
    CHAR    *pchTmp;
    USHORT  usReturn, usIndex, usIndexPre, usIndexNext, usSelect;
    BYTE    bTmp;
    if (usReturn = Memalloc(pszBuffer, usBufLen, &pchTmp))
        return usReturn;
    for (usIndex = 0; abProgramOrder usIndex ; usIndex++);
    MakeFullSession(&aFullSession usIndex , pchTmp, usBufLen);
    if (ProgramCountPos.StartPos == 0xFF)
        {
        usSelect = 0;
        usIndexPre = 0;
        ProgramCountPos.StartPos = (BYTE)usIndex;
```

```
        abProgramOrder usIndex  = 0xFF;
        }
    else
        {
        usSelect = 0;
        usIndexPre = usIndexNext = ProgramCountPos.StartPos;
        while (TRUE)
            {
            if (strcmp(aFullSession usIndex .ProgramTitle, aFullSession usIndexNext .ProgramTitle) <= 0)
                break;
            usSelect++;
            usIndexPre = usIndexNext;
            usIndexNext = abProgramOrder usIndexNext ;
            if (usIndexNext == 0xFF)
                break;
            }
        if (usIndexPre == usIndexNext)
            {
            bTmp = ProgramCountPos.StartPos;
            ProgramCountPos.StartPos = (BYTE)usIndex;
            abProgramOrder usIndex  = bTmp;
            }
        else
            {
            bTmp = abProgramOrder usIndexPre ;
            abProgramOrder usIndexPre  = (BYTE)usIndex;
            abProgramOrder usIndex  = bTmp;
            }
        }
    WinSendMsg(hwndListBox, LM_INSERTITEM, MPFROMSHORT(usSelect), aFullSession usIndex .ProgramTitle);
    if (!ProgramCountPos.Count++)
        WinSendMsg(hwndListBox, LM_SELECTITEM, MPFROMSHORT(0), MPFROMSHORT(TRUE));
    WriteProfileData(usIndex);
    return VALID;
    }

USHORT  ReviseProgram(SHORT sSelect, CHAR *pszBuffer, USHORT usBufLen)
    {
    Delete(sSelect);
    return InsertProgram(pszBuffer, usBufLen);
    }

VOID    Delete(USHORT usSelect)
    {
    USHORT  usDelIndex, usIndex, usItemCount;
    usDelIndex = SelectPosToArrayPos(usSelect);
    usIndex = ProgramCountPos.StartPos;
    if (!(usIndex - usDelIndex))
        {
        ProgramCountPos.StartPos = abProgramOrder usDelIndex ;
        }
    else
        {
        for (; abProgramOrder usIndex  != (BYTE)usDelIndex; usIndex++);
        abProgramOrder usIndex  = abProgramOrder usDelIndex ;
        }
    abProgramOrder usDelIndex  = 0;
    usItemCount = --ProgramCountPos.Count;
    WinSendMsg(hwndListBox, LM_DELETEITEM, MPFROMSHORT(usSelect), NULL);
    if (usItemCount> 0)
        {
        if (usItemCount - usSelect < 1)
            usSelect = usItemCount - 1;
        WinSendMsg(hwndListBox, LM_SELECTITEM, MPFROMSHORT(usSelect), MPFROMSHORT(TRUE));
        }
    MemFree(pProfileData, aFullSession usDelIndex .pData);
    aFullSession usDelIndex .pData = NULL;
    WriteProfileData(usDelIndex);
    }

BOOL    DeleteProgram(CHAR *pszBuf, USHORT BufLen)
    {
    static CHAR     szMsg   = " を削除してよろしいですか？";
    USHORT          usSelect;
    usSelect = (USHORT)WinSendMsg(hwndListBox, LM_QUERYSELECTION, NULL, NULL);
    WinSendMsg(hwndListBox, LM_QUERYITEMTEXT, MPFROM2SHORT(usSelect, BufLen - sizeof(szMsg)), MPFROMP(pszBuf));
    strcat(pszBuf, szMsg);
    if (MBID_YES != WinMessageBox(HWND_DESKTOP, hwndListBox, pszBuf, szClientClass, 0,
                                        MB_YESNO | MB_ICONQUESTION | MB_DEFBUTTON2))
        return FALSE;
    Delete(usSelect);
    return TRUE;
    }
```

```
USHORT  main(SHORT argc)
    {
    static ULONG    flFrameFlags = FCF__TITLEBAR | FCF__SYSMENU | FCF__SIZEBORDER | FCF__MINMAX
                                                    | FCF__MENU | FCF__ICON | FCF__ACCELTABLE;
    HMQ     hmq;
    QMSG    qmsg;
    HWND    hwndFrame, hwndClient;
    USHORT  usIndex, usCount;
    CHAR    ch;
    for (usIndex = 0, usCount = 0; ch = Program[ usIndex ; usIndex++)
        {
        if (ch == ' ')
            {
            usCount++;
            if (usCount != 1)
                Program[ usIndex  = 0;
            }
        }
    hab = WinInitialize(0);
    if (argc > 1)
        {
        WinWriteProfileData(hab, szAppName, NULL, NULL, 0);
        WinTerminate(hab);
        return 1;
        }
    hmq = WinCreateMsgQueue(hab, 0);
    WinRegisterClass(hab, szClientClass, ClientWndProc, CS__SIZEREDRAW, 0);
    hwndFrame = WinCreateStdWindow(HWND__DESKTOP, 0L, &flFrameFlags, szClientClass,
                                            NULL, 0L, NULL, ID__RESOURCE, &hwndClient);
    while (TRUE)
        {
        while (WinGetMsg(hab, &qmsg, NULL, 0, 0))
            WinDispatchMsg(hab, &qmsg);
        if (MBID__OK == WinMessageBox(HWND__DESKTOP, hwndClient, "本当に終了してよろしいですか？",
                                            szClientClass, 0, MB__OKCANCEL | MB__ICONQUESTION))
            break;
        WinCancelShutdown(hmq, FALSE);
        }
    WinDestroyWindow(hwndFrame);
    WinDestroyMsgQueue(hmq);
    WinTerminate(hab);
    return 0;
    }

MRESULT CALLBACK ClientWndProc(HWND hwnd, USHORT msg, MPARAM mp1, MPARAM mp2)
    {
    static HWND     hwndFrame;
    static SHORT    cxClient, cyClient, cxBorder, cyBorder;
    static LONG     cxChar, cyChar;
    static SWCNTRL  swctl;
    static HSWITCH  hswitch;
    PID             pid;
    HDC             hdc;
    SHORT           cxFullScreen, cyFullScreen;
    ADDREVPARAM     AddRevParam;
    switch(msg)
        {
        case WM__CREATE:
            hdc = WinOpenWindowDC(hwnd);
            DevQueryCaps(hdc, CAPS__CHAR__WIDTH, 1L, &cxChar);
            DevQueryCaps(hdc, CAPS__CHAR__HEIGHT, 1L, &cyChar);
            hwndFrame = WinQueryWindow(hwnd, QW__PARENT, FALSE);
            WinSetWindowText(hwndFrame, szClientClass);
            WinQueryWindowProcess(hwndFrame, &pid, NULL);
            swctl.hwnd = hwndFrame;
            swctl.hwndIcon = WinSendMsg(hwndFrame, WM__QUERYICON, NULL, NULL);
            swctl.hprog = NULL;
            swctl.idProcess = pid;
            swctl.idSession = NULL;
            swctl.uchVisibility = SWL__VISIBLE;
            swctl.fbJump = SWL__JUMPABLE;
            strcpy(swctl.szSwtitle, szClientClass);
            hswitch = WinAddSwitchEntry(&swctl);
            hwndListBox = WinCreateWindow(hwnd, WC__LISTBOX, "", WS__VISIBLE | LS__NOADJUSTPOS,
                                        0, 0, 0, 0, hwnd, HWND__TOP, IDL__MYLIST, NULL, NULL);
            WinSendMsg(hwndListBox, LM__DELETEALL, NULL, NULL);
            cxFullScreen = (SHORT)WinQuerySysValue(HWND__DESKTOP, SV__CXSCREEN);
            cyFullScreen = (SHORT)WinQuerySysValue(HWND__DESKTOP, SV__CYSCREEN);
            cxBorder = (SHORT)WinQuerySysValue (HWND__DESKTOP, SV__CXBORDER);
            cyBorder = (SHORT)WinQuerySysValue (HWND__DESKTOP, SV__CYBORDER);
            WinSetWindowPos(hwndFrame, NULL, cxFullScreen / 4, cyFullScreen / 4,
                            46 * (SHORT)cxChar, 14 * (SHORT)cyChar, SWP__MOVE | SWP__SIZE);
            WinShowWindow(hwndFrame, TRUE);
            WinSetFocus(HWND__DESKTOP, hwndListBox);
```

```
            GetProfileData();
            SetProgramList();
            return 0;
        case WM_SIZE:
            cxClient = SHORT1FROMMP(mp2);
            cyClient = SHORT2FROMMP(mp2);
            WinSetWindowPos(hwndListBox, NULL, -cxBorder, -cyBorder, cxClient + 2*cxBorder, cyClient + 2*cyBorder,
                                                                                    SWP_MOVE | SWP_SIZE);
            return 0;
        case WM_CONTROL:
            switch (SHORT1FROMMP(mp1))
                {
                case IDL_MYLIST:
                    switch (SHORT2FROMMP(mp1))
                        {
                        case LN_ENTER:
                            StartSession(hwnd, szMainBuffer, 2*LENGTH, szMainBuffer + 2*LENGTH, 2*LENGTH);
                            return 0;
                        break;
                        }
                    break;
                }
            break;
        case WM_COMMAND:
            AddRevParam.Buffer = szMainBuffer;
            switch (COMMANDMSG(&msg)->cmd)
                {
                case IDM_ADD_PROG:
                    if (ProgramCountPos.Count > PRG_NUMBER_MAX)
                        {
                        WinMessageBox(HWND_DESKTOP, hwnd, "これ以上プログラムを追加できません。!",
                                                        szClientClass, 0, MB_OK | MB_ICONEXCLAMATION);
                        return 0;
                        }
                    AddRevParam.Type = IDD_ADD_PROG;
                    WinDlgBox(HWND_DESKTOP, hwnd, AddRevProgDlgProc, NULL, IDD_ADD_PROG, (PVOID)&AddRevParam);
                    return 0;
                case IDM_REV_PROG:
                    if (!ProgramCountPos.Count)
                        return 0;
                    AddRevParam.Type = IDD_REV_PROG;
                    WinDlgBox(HWND_DESKTOP, hwnd, AddRevProgDlgProc, NULL, IDD_REV_PROG, (PVOID)&AddRevParam);
                    return 0;
                case IDM_DEL_PROG:
                    if (!ProgramCountPos.Count)
                        return 0;
                    DeleteProgram(szMainBuffer, 2*LENGTH);
                    return 0;
                case IDM_ABOUT:
                    WinDlgBox(HWND_DESKTOP, hwnd, AboutDlgProc, NULL, IDD_ABOUT, NULL);
                    return 0;
                }
            break;
        case WM_DESTROY:
            if (pProfileData)
                MemFree(pProfileData, NULL);
            WinRemoveSwitchEntry(hswitch);
            return 0;
        }
    return WinDefWindowProc(hwnd, msg, mp1, mp2);
    }

MRESULT CALLBACK AddRevProgDlgProc(HWND hwnd, USHORT msg, MPARAM mp1, MPARAM mp2)
    {
    static USHORT           usAddOrRev;
    static SHORT            sSelect;
    static PADDREVPARAM     pAddRevParam;
    static CHAR             *pchTmp;
    USHORT                  usBufLen, usResult, usIndex;
    switch (msg)
        {
        case WM_INITDLG:
            pAddRevParam = mp2;
            pchTmp = pAddRevParam->Buffer;
            usAddOrRev = pAddRevParam->Type;
            WinSendDlgItemMsg(hwnd, IDD_PRG_TITLE, EM_SETTEXTLIMIT, MPFROM2SHORT(PRG_TITLE_MAX, 0), NULL);
            WinSendDlgItemMsg(hwnd, IDD_PRG_PATHNAME, EM_SETTEXTLIMIT, MPFROM2SHORT(LENGTH, 0), NULL);
            WinSendDlgItemMsg(hwnd, IDD_PARAMETER, EM_SETTEXTLIMIT, MPFROM2SHORT(LENGTH, 0), NULL);
            WinSendDlgItemMsg(hwnd, IDD_WORKDIR, EM_SETTEXTLIMIT, MPFROM2SHORT(LENGTH, 0), NULL);
            WinSendDlgItemMsg(hwnd, IDD_ICON_PATHNAME, EM_SETTEXTLIMIT, MPFROM2SHORT(LENGTH, 0), NULL);
            if (usAddOrRev == IDD_REV_PROG)
                {
                sSelect = (USHORT)WinSendMsg(hwndListBox, LM_QUERYSELECTION, NULL, NULL);
                usIndex = SelectPosToArrayPos(sSelect);
```

```
            WinSetDlgItemText(hwnd, IDD__PRG__TITLE, aFullSession usIndex .ProgramTitle);
            WinSetDlgItemText(hwnd, IDD__PRG__PATHNAME, aFullSession usIndex .PathName);
            WinSetDlgItemText(hwnd, IDD__PARAMETER, aFullSession usIndex .Parameter);
            WinSetDlgItemText(hwnd, IDD__WORKDIR, aFullSession usIndex .WorkDir);
            WinSetDlgItemText(hwnd, IDD__ICON__PATHNAME, aFullSession usIndex .IconFileName);
            }
        return 0;
    case WM__CLOSE:
        WinDismissDlg(hwnd, FALSE);
        return 0;
    case WM__COMMAND:
        switch (COMMANDMSG(&msg)->cmd)
            {
            case DID__OK:
                WinQueryDlgItemText(hwnd, IDD__PRG__TITLE, PRG__TITLE__MAX, pchTmp);
                pchTmp += strlen(pchTmp);
                *pchTmp++ = 0;
                WinQueryDlgItemText(hwnd, IDD__PRG__PATHNAME, LENGTH, pchTmp);
                pchTmp += mystrupr(pchTmp);
                *pchTmp++ = 0;
                WinQueryDlgItemText(hwnd, IDD__PARAMETER, LENGTH, pchTmp);
                pchTmp += mystrupr(pchTmp);
                *pchTmp++ = 0;
                WinQueryDlgItemText(hwnd, IDD__WORKDIR, LENGTH, pchTmp);
                pchTmp += mystrupr(pchTmp);
                *pchTmp++ = 0;
                WinQueryDlgItemText(hwnd, IDD__ICON__PATHNAME, LENGTH, pchTmp);
                pchTmp += mystrupr(pchTmp);
                *pchTmp++ = 0;
                *pchTmp++ = 0;
                usBufLen = pchTmp - pAddRevParam->Buffer;
                switch (usAddOrRev)
                    {
                    case IDD__ADD__PROG:
                        usResult = InsertProgram(pAddRevParam->Buffer, usBufLen);
                        break;
                    case IDD__REV__PROG:
                        usResult = ReviseProgram(sSelect, pAddRevParam->Buffer, usBufLen);
                        break;
                    default:
                        WinDismissDlg(hwnd, FALSE);
                        return 0;
                    }
                if (usResult)
                    {
                    if (MBID__CANCEL == AddRevDlgErrorMessage(hwnd, usAddOrRev, usResult))
                        {
                        WinDismissDlg(hwnd, FALSE);
                        return 0;
                        }
                    else
                        {
                        switch (usResult)
                            {
                            case INVALID__PROGRAM__FILE:
                                WinSetFocus(HWND__DESKTOP, WinWindowFromID(hwnd, IDD__PRG__PATHNAME));
                                break;
                            case INVALID__WORKDIR:
                                WinSetFocus(HWND__DESKTOP, WinWindowFromID(hwnd, IDD__WORKDIR));
                                break;
                            case INVALID__ICON__FILE:
                                WinSetFocus(HWND__DESKTOP, WinWindowFromID(hwnd, IDD__ICON__PATHNAME));
                                break;
                            }
                        return 0;
                        }
                    }
                WinDismissDlg(hwnd, TRUE);
                return 0;
            case DID__CANCEL:
                WinDismissDlg(hwnd, FALSE);
                return 0;
            }
        break;
        }
    return WinDefDlgProc(hwnd, msg, mp1, mp2);
    }

MRESULT CALLBACK AboutDlgProc(HWND hwnd, USHORT msg, MPARAM mp1, MPARAM mp2)
    {
    switch (msg)
        {
        case WM__COMMAND:
            switch (COMMANDMSG(&msg)->cmd)
```

```
            {
        case DID_OK:
        case DID_CANCEL:
            WinDismissDlg(hwnd, TRUE);
            return 0;
        }
    break;
    }
return WinDefDlgProc(hwnd, msg, mp1, mp2);
}

MRESULT CALLBACK InputParamDlgProc(HWND hwnd, USHORT msg, MPARAM mp1, MPARAM mp2)
{
static PINPUTPARAM      pInputParam;
USHORT                  usIndex;
switch (msg)
    {
    case WM_INITDLG:
        pInputParam = mp2;
        usIndex = pInputParam->Index;
        WinSendDlgItemMsg(hwnd, IDD_PARAMETER, EM_SETTEXTLIMIT, MPFROM2SHORT(LENGTH, 0), NULL);
        WinSetDlgItemText(hwnd, IDD_PRG_TITLE, aFullSession usIndex .ProgramTitle);
        WinSetDlgItemText(hwnd, IDD_PARAMETER, pInputParam->PgmInputs);
        return 0;
    case WM_CLOSE:
        WinDismissDlg(hwnd, FALSE);
        return 0;
    case WM_COMMAND:
        switch (COMMANDMSG(&msg)->cmd)
            {
            case DID_OK:
                WinQueryDlgItemText(hwnd, IDD_PARAMETER, pInputParam->Len, pInputParam->PgmInputs);
                WinDismissDlg(hwnd, TRUE);
                return 0;
            case DID_CANCEL:
                WinDismissDlg(hwnd, FALSE);
                return 0;
            }
        break;
    }
return WinDefDlgProc(hwnd, msg, mp1, mp2);
}

*****************************************************************************************************************
```