Net Express

Dialog System ユーザーガイド

Micro Focus Dialog System[tm]

Dialog System ユーザーガイド

Micro Focus(R)

第8号
1998年10月

はじめに

このユーザーガイドでは、グラフィカルユーザーインタフェースを紹介するとともに、Dialog Systemがその特徴をどのように生かしているかについて説明します。ここでは、おもなメニュー項目を順番に見てゆきます。そして、簡単なスクリーンセットの作成と使用について説明し、おもな機能およびアプリケーション開発の一般的手順を示します。また、上級機能についても取り扱います。本書では、例やサンプルプログラム、そしてDialog Systemの最大限に活用するためのヒントを豊富に紹介します。

対象読者

本書は、以前のDialog System製品を使った経験がある人々からMicro Focus社製品に初めて接する人々を含む、Dialog Systemを使用するすべてのプログラマとシステム設計者を対象としています。また、事務計算およびMicrosoft Windowsの使い方について一般的な概念を理解している必要があります。

関連マニュアル

- *Dialog System Reference* が入ったヘルプ
- NetExpressおよびCOBOLシステムの他の構成要素に関するオンラインヘルプ

表記上の規則

- Enterは、復帰改行または実行キーを表します。入力すべきコマンドが示されている所ではEnterの表記はありません。コマンド行の終わりでは、Enterキーを押すものとして扱います。
- 16進数は引用符で囲み、先頭に小文字のxまたはhが付いています。（例、x"9D"、h"03FF"）。"x"とは16進数が文字列を示すときに用いられ、"h"は数値を表すときに用いられます。
- PIC Xは、COMP-XとCOMP-5データ型と共に、PIC 99よりも用いられます。PIC XはPIC 99とは異なり、データ項目の長さを示しているため、特定バイト数の2進数項目を定義するためのCOMP-Xの使用が明確になります。
- 環境名の見出しは、環境に特有の情報を示すために用います。その例を次に示します。

  Windows:
  Windows固有の情報

- このソフトウェアを使用する環境によって、画面表示や表示メニューがマニュアルの記載内容と異なることがあります（例えば、バージョンの違いによるなど）。多少の相違があっても、ソフトウェアの操作には支障ありません。
- 環境によってはこのマニュアルで示すキーが使用できない場合もあります。ステータスキーやファンクションキーなどのキーを示している場合、実際にこれを押したり、指を離したりするのではなく、論理上の操作

になります。環境によっては、使用中のキーボードに同じキーがなければ、*『リリースノート』*を参照して、同じ機能を持つキーを見つけてください。

- 使用するグラフィカルユーザーインタフェース（GUI）環境によっては、その動作が異なる場合があります。これは使用するオペレーティングシステムの動作の違いによるもので、Dialog Systemに起因するものではありません。このような場合の処理方法については、その旨記載してあります。

- 「OS/2」は、Microsoft Operating System/2（MS OS/2）とIBM OS/2の両方を意味します。同様に、「DOS」は、ご使用のパーソナルコンピュータにより、IBM DOSとMS-DOSのどちらにも該当します。また、「PM」はOS/2 プレゼンテーションマネージャを指します。

- 「ウィンドウ」という用語は画面上の線で囲まれた枠の中の領域を意味し、通常、全画面より小さい領域です。「Windows」はMicrosoft Windows 3.1以上を意味します。

- 本文中特に指示がなければ､プレゼンテーションマネージャに適用される情報は､プレゼンテーションマネージャ 32ビットも対象となり、同様にWindowsの情報は、Windows NT、Windows 95も対象となります。

- このマニュアルではオンライン・ヘルプについて言及しません。この機能は、作業に応じて変わりますので、メニューから[ヘルプ]を選択するか、ダイアログボックスの[ヘルプ]ボタンを選んでその内容をご覧ください。

Tコマンド行の形式についての表記は以下の通りです。

- イタリック体または日本語で書かれた単語はユーザーが作成する名称を表します。

- 大括弧[　]で囲まれた項目は、入力してもしなくても構いません。

- 中括弧{　}で囲まれた項目は、その中の一つを選択しなければなりません。中括弧の中の選択項目が一つしかない場合は、中括弧は繰り返しを示します。

- 中括弧{　}や大括弧[　]の後の省略記号（...）は、{　}または[　]の中の内容を反復できることを表しています。その回数は、特に指定がなければ無限に反復されます。省略記号を大括弧と共に用いるなら、その内容も一緒に省略することができます。

- コマンド行が一行を超える場合、次の行に続けることができます。次の行は字下げされます。

- コマンド行は、/オプションと、-オプションのいずれかで指定できます。

目次

第1章　グラフィカルユーザーインタフェース

この章では、グラフィカルユーザーインタフェース（GUI）を紹介し、その使い方について説明します。

Dialog Systemには、実用的なGUIを短期間で簡単に構築できるソフトウェアが用意されています。また、Dialog Systemの総合的なテスト機能を使用すると、インタフェースを使用するプログラムを作り始めるまえに、インタフェーステストおよびプロトタイプ化を行うことができます。Dialog Systemには次のような機能があります。

- プログラムプログラムやメインプログラムから独立したユーザーインタフェースを作成できます。
- アプリケーションプログラムに影響を与えることなく、ユーザーインタフェースを定義および変更できます。
- 1つのプログラムについて複数のユーザーインタフェースを作成できます。
- サポートを提供するCOBOLプログラムがなくても、インタフェースをプロトタイプ化およびテストできます。

  したがって、Dialog Systemから出ることなく、プログラムとは無関係に、データを入力したり、出力を受け取ったり、ウィンドウ間を移動することができます。
- ランタイムにおけるユーザーとのすべての対話は、アプリケーションプログラムからDialog Systemへの呼び出しによって取り扱われます。

  したがって、アプリケーションプログラムが小さくて済み、保守も簡単です。
- Dialog Systemは、プログラムとユーザーインタフェース間の通信用のコーディング済みデータ構造を提供することにより、プログラムの作成を支援します。
- Dialog Systemは異なる環境の間で移植可能です。

用語

オンラインヘルプの*用語集*では、Dialog System のマニュアルで使用されている用語が詳しく説明されています。1つの項目について2つの意味がある場合、用語集ではその両方について説明するとともに、Dialog System で使用されている方を示しています。

1.1 アプリケーションに GUI を使用する目的

GUI の目的は、次のような特徴を持ったインタフェースを提供することです。

- 簡単に学習できる
- 1度学習するだけでよい（アプリケーション間で一貫性がある）

- 簡単に使用できる

GUI は、アプリケーションによって提供される機能よりも、ユーザーを自分自身の仕事に専念させます。GUI は直感的で、ユーザーによる実験や探求を促進します。また、間違いも許されます。つまり、ユーザーは自分自身の方法で自由なペースで学習できることになります。

GUI をしばらく使用していなかったユーザーが再び使用する場合、使い方をすぐに思い出すことができます。もし、思い出せなくても初めから試すことができます。

キャラクタベースのアプリケーションインタフェースは、階層状のメニューを使用しています。このようなメニューでは、ユーザーは目的の機能まで各種のレベルを移動しなければなりません。ただし、ショートカットキーを覚える方法もあります。

GUI アプリケーションは、行うべきタスクに、より直接的にアプローチします。ユーザーは、オブジェクトを選択してから操作を選択します。これによって、いくつかのメニューレベルを削減できます。マウスのポインタ操作とクリック操作によって、ユーザーは、インタフェース内を簡単に移動して必要なタスクを実行することができます。

GUI は、関連する領域とのリンクによりヘルプを容易に提供することができます。これによってユーザーは、階層ヘルプツリー内の移動など、キャラクタベースのシステムで提供される複雑なヘルプ機能を使用しなくても、タスクを実行するのに必要な背景知識を簡単に得ることができます。

1.2 Dialog System の役割

Dialog System は、ユーザーインタフェースのライフサイクルの5つの段階でユーザーを支援します。

- ユーザーインタフェースの定義

  Dialog System を使って、アプリケーションが使用するデータとスクリーン上のプレゼンテーションを定義します。これには、ユーザーがデータを入力したりインタフェース内を移動する方法が含まれます。この情報は、スクリーンセットと呼ばれ、.gs ファイルとして保存されます。

- プロトタイプ化とテスト

  Dialog System テスト機能を使って、プログラムが実行されているものとして、データを入力したりインタフェース内を移動します。出力をシミュレートしたり、ユーザーに何が返されるかを監視します。

- 呼出しプログラムの統合化インタフェースを詳しくテストしたら、Dialog System を使って、データ構造やランタイムインタフェースの詳細が入った COPY ファイルを生成します。

- アプリケーションの実行

  Dialog System をランタイムモードで使って、呼出しプログラムに制御が戻るまで、ユーザーとのダイアログを取り扱い、スクリーンセットを使用します。

- アプリケーションの保守

  Dialog System を使って、呼出しプログラムに影響を与えることなく、ユーザーインタフェースを変更およびカスタマイズします。同様に、ユーザーインタフェースを変更せずに、呼出しプログラムを変更することができます。両方を変更するのは、ユーザーインタフェースとプログラムの間で受け渡しされるデータ項目を変更する必要があるときだけです。

1.3 GUI システムの使用

Dialog System は、マウスで使用するように設計されています。マウスを使うと、スクリーン上のオブジェクトの選択、移動、サイズ変更を行うことができます。これらの操作は、キーボードを使って行うこともできますが、マウスの方が便利です。キーボードは、テキストや数値データを入力するのに使用します。

次の各項では、GUI 環境が初めてという方のために、マウス、ウィンドウ、コントロールオブジェクトの使い方について説明します。

1.3.1 マウス操作

マウスは、スクリーン上の位置を示すポインタを操作します。スクリーン上には、ウィンドウ内の位置を示す選択カーソルもあります。

マウスポインタの形状は変化することがあります。たとえば、テキストフィールドでは、I字形で、テキストが追加される正確な位置を示します。サイズ変更可能なウィンドウの角では両矢印になり、処理を待っている間は時計（環境によって文字盤のある時計や砂時計）になります。マウスポインタの形状は、デスクトップ上の特定の場所で何ができるかを示す手がかりとなります。

選択や移動などのマウス操作を行うまえに、マウスポインタをスクリーン上の適切な位置に移動します。たとえば、メニューオプションを選択したい場合、ポインタをそのオプションに移動します。

マウスを使って、スクリーン上の各種ボタンを操作したり、オブジェクトを選択したり、項目を選択したり、スクリーン上の情報をスクロールすることができます。

次のマウス操作のリストでは、各操作に使用するマウスボタンは示していません。これは、オペレーティングシステムによって動作が異なるためです。

- クリックする-マウスボタンを押して、離します。

  項目またはオブジェクトを選択したり、プッシュボタンやラジオボタンを押したり、チェックボックスやチェックマーク選択を切り替えます。

- クリックおよびドラッグする（ラバーバンドとも呼ばれる）-マウスボタンを押したまま、マウスポインタを希望の方向に移動し、マウスボタンを離します。

複数の項目を選択するためにその領域を囲むボックスを描いたり、選択されたオブジェクトを移動したり、サイズ変更操作でオブジェクトの境界を変更します。

- ダブルクリックする-マウスボタンを押して離す操作を2回繰り返します。

  リスト項目を選択し、デフォルトのプッシュボタンを押します。Dialog System のオブジェクトでは、そのオブジェクトの属性ダイアログボックスが表示されます。

- 離す-押していたマウスボタンを離します。

  スクロールを止めたり、移動操作でオブジェクトのドラッグを止めたり、サイズ変更操作でウィンドウまたはオブジェクトの境界の移動を止めます。

移動や選択など、Dialog System に関する他のボタン動作は構成できます。Dialog System に使用するために、特定の動作を付けることができます。Dialog System には、このための各種の標準パターンがあります。何を使用するかは、オペレーティングシステム環境、慣れているソフトウェア、使用しているマウスの種類（2ボタンまたは3ボタン）によって異なります。

1.3.2 ウィンドウとメニュー

デスクトップとは、スクリーン上の作業領域です。ウィンドウは、スクリーンの全体または一部を表すオブジェクトです。スクリーンの端を越えるデスクトップ上にウィンドウを作成することができます。ウィンドウはデスクトップ上で移動することもできます。

最初に作成するウィンドウは一次ウィンドウと呼ばれます。このウィンドウの上には他のウィンドウがないので、これはデスクトップの子ということになります。これは、デスクトップがその親であることを意味します。複数の一次ウィンドウを作成することもできます。

任意の一次ウィンドウは、二次ウィンドウと呼ばれる子ウィンドウを持つことができます。この二次ウィンドウは、他の二次ウィンドウを持つこともできます。

一次ウィンドウはと二次ウィンドウについては*「ウィンドウオブジェクト」*の章で詳しく解説しています。

メインメニューバーは、Dialog System を起動したときにデスクトップ上に表示されます。Windows と Motif では、メインメニューバーは縮小サイズ Dialog System ウィンドウとともに表示されます。

サブメニューは、必要な選択項目をクリックすることによって、メニューバーからプルダウンすることができます。サブメニューがプルダウンされている間に別の選択項目をクリックした場合、現在のメニューが閉じ、別のメニューがプルダウンされます。

特定の操作を行うまでは、選択ができないことがあります（たとえば、そこに入れるウィンドウまたはダイアログボックスを作成するまでは、プッシュボタンなどのコントロールオブジェクトを作成できません）。選択できない項目は淡色で表示されます（選択できる項目は濃い色で表示されます）。この場合、選択項目は淡色表示またはグレー表示

されているといいます。淡色表示された選択項目をクリックした場合、警報音が鳴りますが、何も起こりません。

メニューの選択項目では、以下のことができます。

- 追加のメニューをプルダウンします。これらは、選択項目の後ろに続く矢印（→）によって示されます。
- 二次ウィンドウまたはダイアログボックスを表示します。これらは、選択項目の後ろに続く省略記号（...）によって示されます。
- チェックマーク（3）された選択項目になります。これらは、オンまたはオフの2つの状態を持っています。選択項目をクリックすると、その状態が反転します。オンの場合、チェックマークは、プレゼンテーションマネージャまたは Windows では前に表示されます。Motif では、チェックボックス内のボタンが押された状態になります。これらの選択項目は、トグルと呼ばれることもあります。
- 操作が直接実行されるようにします。これらには、特別な識別はありません。

Aすべてのメニュー選択項目は、前のリストで説明した操作のいずれか1つだけ持つことができ、適切に（たとえば、省略記号またはチェックマークによって）デザインすることができます。

1.3.2.1 ウィンドウの操作

この項では、マウスとアイコン（ウィンドウの大部分を形成する）を使って、ウィンドウを操作するさまざまな方法について説明します。

各コンポーネントを示したウィンドウを図 1-1 に示します。

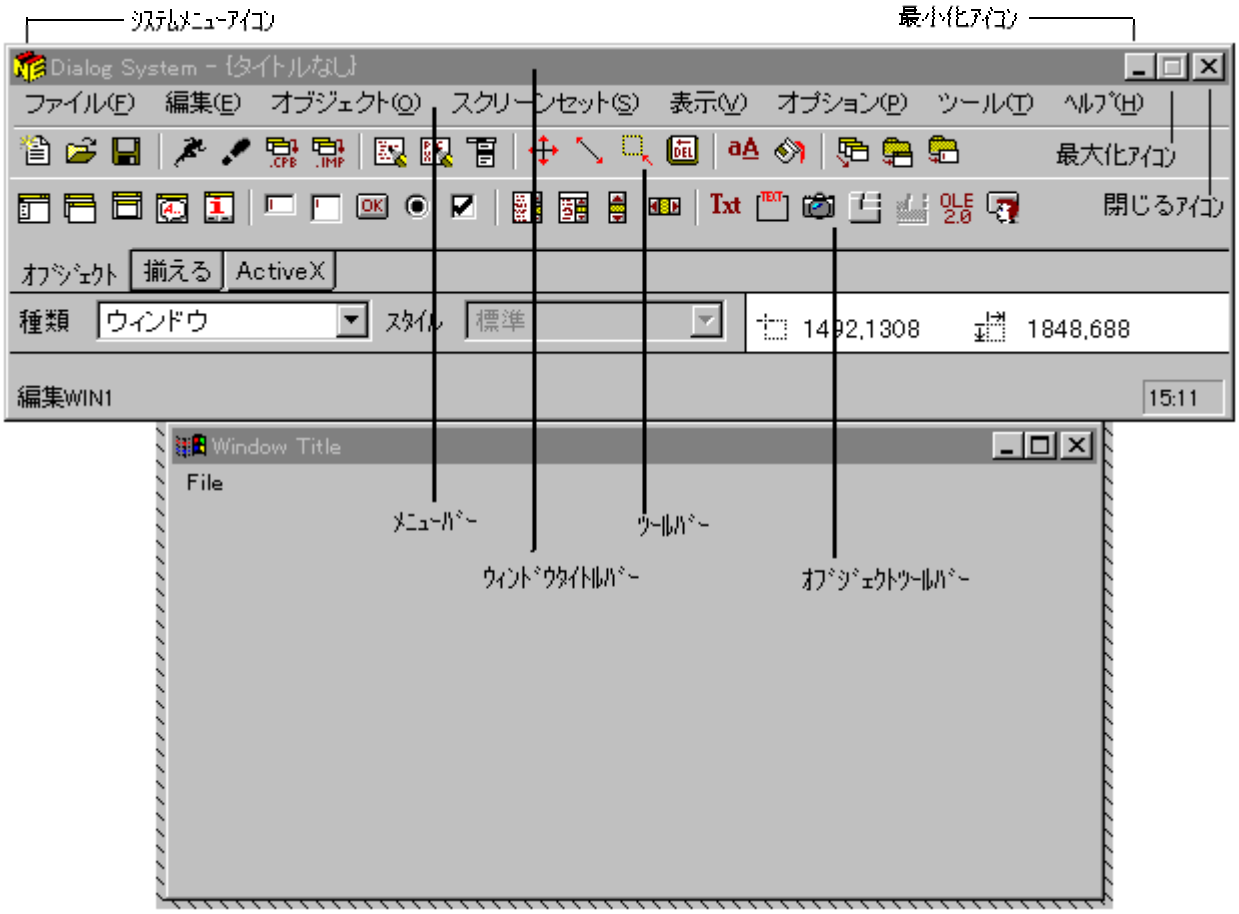


図 1-1 ウィンドウのコンポーネント

一次ウィンドウを使用していない場合、タスクバー上にアイコンと呼ばれる小さな絵記号にウィンドウを縮小することができます。

最小化アイコンがあるウィンドウは、マウスによって最小化できます。これには、ウィンドウの左上隅にある最小化アイコンをクリックします。実際のアイコンの形状は、オペレーティングシステムによって異なります。ウィンドウを復元するには、ウィンドウアイコンをダブルクリックします（前の「マウス操作」の項を参照）。

ウィンドウを移動したり、ウィンドウのサイズを変更することもできます。これらの操作は、マウスで行うのが最適です。ウィンドウを移動するには、マウスポインタをウィンドウのタイトルバーに移動し、ウィンドウを新しい場所にドラッグし、マウスボタンを離します。このときに押すボタンは、マウスの設定によって異なります。

同様にウィンドウをサイズ変更するには、マウスポインタをウィンドウの上下左右または角の境界に移動してから、クリックおよびドラッグします。マウスポインタは、ウィンドウをサイズ変更するための適切な位置にある場合、形状が変化します。ウィンドウの上下左右または角をドラッグすると、ウィンドウの形が変化します。ウィンドウが求めるサイズと形になったら、マウスボタンを離します。

ウィンドウを閉じるには、ウィンドウの左上隅にあるシステムメニューアイコンをダブルクリックします。

デスクトップ上のウィンドウ間を移動するには、移動する先のウィンドウの一部をクリックします（タイトルバーに

ある任意のアイコンをクリックすると、そのアイコンがアクティブになります）。

目的のウィンドウが他のウィンドウの下に隠れて見えない場合、タスクバーでウィンドウを切り換えます。

現在のウィンドウ（アクティブウィンドウ）は、境界が色付きで表示されています。

ウィンドウ操作についての詳細は、各オペレーティングシステムのマニュアルを参照してください。

1.3.3 ダイアログボックス

ダイアログボックスは、ユーザーが特定のデータを入力するためのウィンドウです。ダイアログボックスは、サイズを変更したり、メニューバーを持つことはできません。ダイアログボックスには、Dialog System の任意のコントロールオブジェクト（プッシュボタンやリストボックスなど）を追加することができます。ダイアログボックスは、別のウィンドウまたはデスクトップによって作成されます（ダイアログボックスは、それを作成したウィンドウの子と呼ばれ、そのウィンドウはダイアログボックスの親と呼ばれます）。

ダイアログボックスは、ユーザーの操作によって閉じられるまで表示されています。通常は、ユーザーがプッシュボタンを押して、Dialog System がユーザーの操作を受け入れるようにします（たとえば、リストからの選択を受け入れるための OK ボタン）。別の方法として、プッシュボタンを押して、Dialog System がユーザーの操作を無視するようにすることができます（たとえば、入力を無視してダイアログボックスを閉じるためのキャンセル ボタン）。

ダイアログボックスは、ファイル選択ボックスを作成するのによく使用されます。これによって、ユーザーはファイルやその他の項目を選択したり、他のフィールドに選択項目を入力することができます。

1.3.4 メッセージボックス

メッセージボックスは、ユーザーにメッセージを伝えるためにスクリーン上に表示されるもう1つのウィンドウです。メッセージボックスには、メッセージを与えるためのテキストとグラフィクス、メッセージに対して応答するためのプッシュボタンが含まれます。メッセージボックスは、ユーザーがいずれかのプッシュボタンを押して消去しなければなりません。

メッセージボックスは、誤ったデータが入力された場合に警告を与えたり、破壊的な操作（オブジェクトの削除など）を続けてよいかをユーザーに確認するのに使用できます。

1.3.5 コントロール

コントロールは、ユーザーがアプリケーションと対話できるように、ウィンドウまたはダイアログボックスに追加することができる Dialog System のオブジェクトです。コントロールオブジェクトが配置できるのは、ウィンドウまたはダイアログボックスの内側にだけです。Dialog System には、入力フィールド、プッシュボタン、リストボックス、選択ボックス、ビットマップなどのさまざまなコントロールオブジェクトがあります。コントロールオブジェクトについての詳細は、『Dialog System リファレンス』の*「オブジェクトと属性」*の章を参照してください。

1.3.6 選択オブジェクト

GUI システムでは、ユーザーがオブジェクトを選択し、そのオブジェクトに適用すべき操作を選択することによって、アプリケーションを起動することができます。

Dialog System には、以下の選択方法があります。これらは、ほとんどのオブジェクトに対して利用できます。

- 1つのオブジェクトを選択するには、そのオブジェクトにマウスポインタを移動し、クリックします。
- マウスで複数のオブジェクトを選択するには、いずれかのマウスボタンが選択（*「マウス操作」*の項を参照）に設定されていなければなりません。

  シフトキーを押すと個々のオブジェクトをそれらをクリックすることで選択でき、クリックアンドドラッグより便利です。
- メニューまたはキーボードを使ってオブジェクトを選択するには、[編集]メニューの[選択]または[すべてを選択]を使用します。「選択」では、1つまたは複数のオブジェクトが選択されます。「すべてを選択」では、現在のウィンドウまたはダイアログボックスにあるすべてのオブジェクトが選択されます。

Dialog System には、以下の選択方法もあります。これらは、特定のオブジェクトだけに対して利用できるものです。

- メニューの選択項目を選択するには、その選択項目にマウスポインタを移動してクリックするか、カーソル矢印を使って選択項目を選択し Enter キーを押します。
- 選択ボックス内の項目を選択するには、希望する項目が表示されるまでボックス内のリストをスクロールし、マウスポインタを希望する項目に移動し、クリックします。リストの項目をダブルクリックした場合、デフォルト操作が実行されます。他の項目をクリックした場合、最初の選択は取り消され、新しい項目が選択状態になります。
- リストボックス内の項目を選択するには、希望する項目が表示されるまでボックス内のリストをスクロールし、マウスポインタを希望する項目に移動し、クリックします。リストの項目をダブルクリックした場合、デフォルト操作が実行されます。リストボックスは、1つの項目だけ、複数の項目、複数の隣接する項目のいずれかを選択するように設定することができます。
- ラジオボタンの項目を選択するには、ボタンをクリックします。ボタンは、選択されていることを示すために、押された表示になります。同じコントロールグループにある別のボタンを選択した場合、前のボタンは選択解除されます（コントロールグループ内の1つのボタンだけが選択できます）。
- チェックボックスの選択項目を選択するには、チェックボックスをクリックします。選択されている場合、チェックボックスの状態が反転され、選択解除されます。選択されていない場合は、選択状態になります。チェックボックスの項目はいくつでもクリックできます。
- プッシュボタン動作を選択するには、ボタンをクリックします。ボタンは、押された表示に変化します。ボ

タンを押すと操作が始まります。

1.3.7 スクロール

ウィンドウによっては、スクロール可能な領域があります。このようなウィンドウには、右端にスクロールバー（縦スクロール用）があり、下端にスクロールバー（横スクロール用）があります。各スクロールバーには、バーに沿って移動するスライダと、スクロールの方向を示す矢印があります。

- 1行ずつスクロールするには、スクロールしたい方向のスクロール矢印をクリックします。
- 1スクリーンずつスクロールするには、スライダの隣のスクロールバー自身をクリックします。クリックした場所に向かってスクロールします。
- 一度に複数のスクリーンをスクロールするには、適切なスクロールバーにあるスライダをクリックし、マウスを使って、スライダを希望する方向にドラッグします。ウィンドウの希望する部分に達したら、マウスボタンを離します。

1.4 次の章の紹介

次の章*「Dialog System入門」*では、Dialog Systemを使用する際に必要な基本概念について説明します。

第2章　Dialog Systemの概要

前の章では、グラフィカルユーザーインタフェース（GUI）を使うことの利点について述べました。この章では、COBOLアプリケーションのためのGUIを作成するとき、Dialog Systemを使用する際に必要な基本概念について説明します。ここでは、次の項目を取り扱います。

- Dialog Systemの利点
- Dialog Systemの各構成部分の概要
- Dialog Systemを使ってアプリケーションを作成するための手順

2.1 Dialog Systemの利点

Dialog System には、次の利点があります。

- メインプログラムロジックからのユーザーインタフェースの独立

  プログラムが実行時にユーザーとの対話を必要とする場合、これを取り扱うために Dialog System を呼び出します。このような独立性によって、構造のよいプログラムが作成しやすくなります。

- プログラムからのユーザーインタフェースの独立

  同一プログラム用に複数の異なるユーザーインタフェースを作成することができます。

- グラフィカルオブジェクトの簡単な定義

  Dialog System には、アプリケーションに合わせてカスタマイズできるグラフィカルオブジェクトのライブラリが用意されています。Dialog System は、ウィンドウのスタックなどのオブジェクトの動作を管理します。スクリーンの取り扱いに煩わされることはありません。これは、定義時も実行時にも Dialog System によって行われます。

- 入力、出力、ウィンドウ間移動の完全な取り扱い

  ダイアログと呼ばれる簡単な命令のセットを使って、Dialog System に対するユーザーの入力や出力を取り扱ったり、ユーザーに何を表示するかを決めるように指示することができます。

- 自動データブロック生成

  Dialog System は、呼出しプログラムにデータブロックのための定義を提供することができます。このデータブロックは、実行時に Dialog System とのインタフェースに必要です。

- 妥当性機能

Dialog System は、入力のほとんどの妥当性を取り扱い、ユーザーにエラーメッセージを表示することができます。呼出しプログラムも、必要に応じてこれらのエラーメッセージを使用することができます。

- プロトタイプ化とテスト

  Dialog System は、インタフェースを実行している呼出しプログラムへのインタフェースを同じ方法で実行することができます。これによって、結果の妥当性に高い信頼性が与えられます。完全なインタフェースができるのを待つ必要はありません。インタフェースの開発中に一部分をテストすることができます。

2.2 Dialog Systemの機能の概要

Dialog Systemを使うと、Windowsシステムに表示するウィンドウやダイアログボックスを設計および編集したり、このインタフェースとアプリケーションプログラムとの間でデータを受け渡しできます。Dialog Systemには次の2つの要素によって、これらの機能を実現しています。

- 定義ソフトウェアーユーザーインタフェースに表示する項目を作成および調整することができます。
- Dsgrunプログラムーインタフェースおよびアプリケーションプログラムの両方と通信士、インタフェース構成要素のランタイム動作を制御します。

この節では、定義ソフトウェアの概要とそれを使ってユーザーインタフェースを作成する方法について、次の項目に分けて説明します。

- インタフェースとその要素の設計.
- ダイアログを使ってインタフェース要素をアクティブ化する方法.
- データブロックを使って各構成部分を呼び出しプログラムとリンクする方法.

2.2.1 インタフェースの設計

GUIを持つDialog Systemアプリケーションを開発するための第一歩は、データモデルの設計です。このモデルは、ユーザーインタフェースによって捕獲され、ユーザーに提示するためにアプリケーションによって提供されるデータを定義します。
このデータモデルは呼び出しプログラムに渡さなければいけない基本的なデータ項目です。またこれらの項目を妥当性検査することを
決めなくてはいけません。
ユーザーインタフェースに好ましい特徴のいくつかを次に示します。

ユーザーによるコントロール　ユーザーがインタフェースをコントロールします。ユーザーが、アプリケーションの一連の動作を決めます。

使いやすさ	ユーザーは、ウィンドウ内およびウィンドウ間を簡単で自然に移動できなければなりません。情報は、順序立てて整理し、読みやすい形で表示します。
一貫性	ユーザーインタフェースは、アプリケーション内およびアプリケーション間の両方で一貫していなければなりません。たとえば、ウィンドウの終了選択項目は、常に「ファイル」メニューの最後のオプションになっています。また、削除操作を行う場合、必ず確認が必要です。
支援	ユーザーには、いまシステムのどこにいるか、いまどんな操作を行ったところか、次のどんな操作を行うべきかを明確でわかりやすく示すフィードバックを与えます。データ入力フィールドに項目を入力するより、リストから項目を選択する方がよくできたインタフェースといえます。
寛容性	ユーザーは誤った操作をすることがよくあります。どんなエラーか、なぜエラーが起こったのか、どうしたらエラーを修正できるかなどについて、エラーメッセージで説明します。
効率	システムの応答時間は、できるだけ短くします。
完全性	たとえば、選択項目のリストがある場合、すべての項目が有効でかつ選択可能であるべきです。
論理性	たとえば、メニューの選択項目は論理的に配置します。つまり、最もよく選択される項目を最初に並べたり、項目をアルファベット順に並べたりします。

ユーザーがどのようにしてデータ（ユーザーインタフェース）と対話するかにつても考える必要があります。

ユーザーインタフェースとして、シンプルなメニュー選択やデータ入力フィールドから、メニューバー、ラジオボタン、その他のコントロールが揃った本格的なウィンドウまで、さまざまな形態のものが作成できます。

どんなインタフェースにするかは、次のような条件を参考に決めます。

- ユーザーの種類と熟練度
- 利用可能なハードウェアとソフトウェア資源
- 実行すべきタスクの複雑さ

また、どの場合でも次のことを考慮してください。

- 標準的な操作順序を決めるのはユーザーです。
- インタフェースをコントロールするのは、プログラムではなく、ユーザーです。
- ユーザーとコンピューターとの対話は、できるかぎり自然でシンプルにするべきです。

ユーザーインタフェースのために定義する基本的な視覚要素をオブジェクトと呼びます。オブジェクト（ウィンドウ、ダイアログボックス、押しボタンなど）は、画面の中の囲まれた領域に表示されます。画面には、他のすべてのオブジェクトが表示されます。

Dialog Systemには、大きく分けて2種類のオブジェクトがあります。

2.2.1.1 ウィンドウオブジェクト

これは最も基本的な（かつ最も重要な）オブジェクトです。一次オブジェクトと二次オブジェクトがあり、ダイアログボックスやメッセージボックスと非常によく似ています。ウィンドウとダイアログボックスの表示は、定義時とランタイムの両方で同じです。

ウィンドウオブジェクトはいつでも定義でき、デスクトップ上のどこにでも配置できます。このオブジェクトは、タイトルバーをドラッグして移動したり、サイズ調整可能な境界を使ってサイズを変更することができます。ウィンドウオブジェクトを選択すると、色と影の付いた境界で囲まれます。

詳細は、「*ウィンドウオブジェクト*」の章を参照してください。

2.2.1.2 コントロールオブジェクト

他のすべてのオブジェクトはコントロールオブジェクトです。このオブジェクトは、ウィンドウに表示され、入力フィールド、押しボタン、ラジオボタン、チェックボックス、リストボックスなどがあります。

Dialog Systemのデフォルトオブジェクトの範囲にはない、他のコントロールオブジェクトも定義できます。このようなコントロールオブジェクトを定義した場合、Dialog Systemは専用のコントロールプログラムを生成することができます。

次の2種類のコントロールを定義できます。

- ユーザーコントロール

  このコントロールは、Dialog Systemでは描画不可能なオブジェクトに関するコンテナまたはアウトラインの定義を可能にします。

  定義時には、生成されたコントロールプログラムの名前がコントロールのアウトラインの内側に表示されます。わかりやすい名前を指定すると、コントロールとそのGUIにおける目的を識別するのに役立ちます。

- ActiveXコントロール

  サードパーティーが提供するコントロールです。これらのコントロールをランタイムに作成および操作するには、プログラムを作る必要があります。このコントロールを選択すると、ランタイムと同じような表示になります。

これらコントロールについては、「*コントロールオブジェクト*」と「*独自コントロールのプログラミング*」の章を参

照してください。

2.2.1.3 オブジェクトの属性

すべてのウィンドウオブジェクトとコントロールオブジェクトは、背景色などの属性を持っています。属性を変えることによって、オブジェクトの表示や動作を指定できます。

オブジェクトのデフォルト属性を使用するには、[オプション]メニューの[含める]を選択し、 [属性の自動設定]を選びます。この場合、Dialog Systemはオブジェクトのデフォルト属性値を使用し、[属性]ダイアログボックスが表示されません。オブジェクトの属性を変更するには、[編集]メニューの[属性]を選択するか、オブジェクトをダブルクリックして、表示された[属性]ダイアログボックスを使用します。

[属性の自動設定]を選択しなかった場合、オブジェクトに関する[属性]ダイアログボックスが直ちに表示され、属性を設定できます。この時点でダイアログボックスの[キャンセル]をクリックすると、オブジェクトは取り除かれます。[OK]をクリックすると、ダイアログボックスに表示された属性を使ってオブジェクトが定義されます。

メッセージボックス、ビットマップ、ActiveX、またはユーザーコントロールを定義する場合、 [属性の自動設定]の選択には関係なく[属性]ダイアログボックスが常に表示されます。

2.2.1.4 オブジェクトの操作

あるオブジェクトを操作するには、それを現在のオブジェクトにしなければなりません。オブジェクトを定義した直後は、自動的に現在のオブジェクトになっています。

2.2.1.5 オブジェクトの命名

すべてのオブジェクトに共通の属性は名前です。名前が必須のオブジェクトとオプションのオブジェクトがあります。オブジェクトには、一意な名前を与えなければなりません。

COBOLプログラム内の変数に関する命名規定を準備することは、よいプログラミング習慣です。一貫性のある命名規則を使うと、自分自身または他のプログラマやシステム設計者がアプリケーションを容易に理解できます。

また、ユーザーインタフェース内のオブジェクトについても同様の命名規定を用意することをお勧めします。

次に示すようにいくつかの項目をつなぐと、簡単な命名規定を作ることができます。

ウィンドウ名-オブジェクト情報-オブジェクトタイプ

各項目の意味は次のようになっています。

ウィンドウ名	オブジェクトがあるウィンドウの名前。オブジェクト自身がウィンドウの場合、そのウィンドウの名前。

オブジェクト情報　　オブジェクトに関する何らかの識別情報。この情報は、ウィンドウ上のオブジェクトを一意に識別する必要があります（例、押しボタンに表示されるテキスト）。

オブジェクトタイプ　　オブジェクトのタイプを識別する省略名。たとえば、ウィンドウは*win*、押しボタンは*pb*、ダイアログボックスは*db*など。

この規定を使うと、新入社員に関する情報を集めるウィンドウには、NEW-EMPLOYEE-WINという名前をつけることができます。また、このウィンドウの押しボタンには、NEW-EMPLOYEE-OK-PBという名前をつけることができます。給与情報の入力に使用するフィールドには、NEW-EMPLOYEE-SALARY-EFとするとよいでしょう。

2.2.2 ダイアログを使う

ユーザーインタフェースは、単なるグラフィクス表示ではありません。完全な指定では、ユーザーとコンピュータが対話する方法と、ユーザーインタフェースソフトウェアがアプリケーションソフトウェアと対話する方法を記述する必要があります。

表示を定義したあとは、ユーザーとマシン間の実行時の対話を定義しなければなりません。この対話のことをダイアログと呼びます。ダイアログは、イベントとファンクションから構成されます。イベントが起こると、そのイベントに関連づけられているファンクションが実行されます。イベントは、ユーザーが押すキーボード上のキー、もしくはメニューの選択または選択されているオブジェクトによって起こります。

たとえば、BUTTON-SELECTED という Dialog System イベントは、ユーザーが（マウスまたはキーボードによって）プッシュボタンを選択したときに起こります。選択されたボタンが[入力]の場合は、ユーザーが他の情報を入力するために新しいウィンドウを作成する CREATE-WINDOW というファンクションが関連づけられているかもしれません。

Dialog Systemを使うと、ユーザーと表示オブジェクトとの間のダイアログを作成したり、カスタマイズすることができます。

ダイアログ文についての詳細は、ヘルプの「*ダイアログの使い方*」と「*Dialog Systemの概要*」を参照してください。

2.2.2.1 イベント

イベントは、ユーザーインタフェースにおける何らかの変化を表します。イベントは、ウィンドウ、ビットマップ、リストボックス、ボタンなどのオブジェクトに対して起こります。イベントをトリガーする操作にはいろいろあります。

- ユーザーがキーを押した。
- マウスポインタが押しボタンの上にあるときに、マウスボタンを押した。
- ウィンドウまたはコントロールがフォーカスを受け取った。

- 妥当性検査エラーが起こった。
- ユーザーがスクロールバーのスライダを動かした。

すべてのイベントは内部的に同じです。しかし、便宜上次の3種類に分けています。

- メニューイベント－メニューからオプションが選択された。たとえば、[ファイル]メニューから[終了]を選択した。
- オブジェクトイベント－ウィンドウ、押しボタン、ラジオボタン、リストボックス、ダイアログボックスなどのオブジェクトがアクティブになった。
- キーボードイベント－キーボードのキーが押された。

イベントが起こると、Dialog Systemは関連するコントロールダイアログ、関連するウィンドウダイアログ、グローバルダイアログをこの順序で検索します。イベントは、Dialog定義を作成するときに定義します。イベントは、それに関連づけられるFunctionsファンクションによって定義されます。

2.2.2.2 ファンクション

ファンクションは、Dialog Systemが何かを行うための命令です。ファンクションはイベントと関連づけられており、次のような場合に動作します。

- リストされているイベントが起こった場合
- リストされている手続きが実行された場合

Dialog Systemには、豊富なファンクションがあります。これには、SET-FOCUSやINVOKE-MESSAGE-BOXなどのようにスクリーンセット内を移動するものや、あるデータ項目から別のデータ項目に移動するためのMOVEなどのように、他のプログラミング言語の命令と似たファンクションもあります。

各ファンクションについての詳細は、ヘルプの「*「ダイアログの定義: ファンクション」*」トピックを参照してください。

2.2.2.3 手続き

手続きは、その下にファンクションのリスト（すなわちサブルーチン）を持ち、任意の名前をつけることができます。ここでは、手続きをイベントと同様に考えることができます。

2.2.3 データブロックとスクリーンセットの使い方

Dialog Systemはスクリーンセットと呼ばれるファイルを作成します。このファイルには、インタフェースに関して作成されたすべてのウィンドウとダイアログボックスの定義が保存されます。スクリーンセットは*filename*.gsというファイルです。これには、入力フィールドなど、インタフェースの中で必要な画面上の任意のオブジェクトが含まれ

ます。

インタフェースを実行すると、入力フィールドなどのオブジェクトにはインタフェースとアプリケーションプログラムの間で受け渡しすべきデータが入っています。これには、画面上のオブジェクトをアプリケーションプログラムで定義されたデータ項目と関連づけます。Dialog Systemでは、このようなデータ項目をマスタフィールドと呼びます。関連づけを行うには、コントロールの属性を編集するときに、適切なマスタフィールドを選択します。

これらのデータ定義はスクリーンセットファイルに保持され、データブロックを形成します。生成されたプログラムがDialog Systemのランタイムサポートモジュールdsgrunを呼び出すとき、データブロックをパラメータとして渡します。

スクリーンセットファイルには、次の要素が保持されます。

- データブロック

  すべての入出力に関するデータ定義

- ウィンドウとダイアログボックスに関するオブジェクト（妥当性検査規則を含む）

- これらのオブジェクトと関連づけられたアクション

  これらはダイアログイベントおよびファンクションと呼ばれ、ユーザーが定義します。

2.3 Dialog Systemを使ったアプリケーションの作成手順

Dialog Systemを起動するには、

- IDEの[ツール]メニューから[Dialog System]を選択します。
  または
  [NetExpressコマンドプロンプト]を選択し、[dswin]と入力します。

Dialog Systemを使ってアプリケーションを作成するには、次の手順に従ってください。

1. データと妥当性検査を定義します。

   「*データ定義とスクリーンセットの作成*」の章にある「*データ定義作成の手順*」とヘルプの「*ユーザーデータの妥当性検査*」トピックを参照してください。

2. ウィンドウ、ダイアログボックス、メッセージボックスを定義します。

   「*ウィンドウオブジェクト*」と「*コントロールオブジェクト*」の章を参照してください。

3. コントロールオブジェクト（入力フィールド、テキストフィールド、ラジオボタンなど）を定義し、ウィンドウやダイアログボックスに配置します。

「*ウィンドウオブジェクト*」、「*コントロールオブジェクト*」、「*ダイアログの使い方*」の章を参照してください。

4. スクリーンセットを保存します。

   スクリーンセットは、定義プロセスの各段階が終わるたびに保存するとよいでしょう。スクリーンセットを初めて保存するときは、[名前をつけて保存]を使用します。保存するファイルの名前とディレクトリを入力するためのダイアログボックスが表示されます。

   サンプルスクリーンセットを保存したあとは、いつでも[保存]を使って、スクリーンセットを保存できます。別のバージョンのスクリーンセットを試してみたい場合、[名前をつけて保存]を使用し、新しい名前のバージョンを作成します。

5. スクリーンセットの検査

   スクリーンセットアニメータを使って、スクリーンセットの動作を調べます。ヘルプの「*スクリーンセットアニメータ*」を参照してください。

6. ダイアログを定義します。

   オブジェクトの動作を制御するオブジェクトダイアログを定義します。

7. スクリーンセットをもう一度テストします。

   ヘルプの「*スクリーンセットアニメータ*」を参照してください。

8. 必要に応じてスクリーンセットを変更します。

   前の方のステップに戻ります。

9. スクリーンセットからCOBOLコピーファイルを生成します。

   「*データ定義とスクリーンセットの作成*」の章にある「*データ定義作成の手順*」を参照してください。

10. Dialog Systemのランタイムシステムへの呼び出しを使って、COBOLアプリケーションプログラムを作成します。

    「*スクリーンセットの使い方*」の章を参照してください。

11. COBOLプログラムをアニメートおよびテストします。

    ヘルプの「*デバッグ*」トピックを参照してください。

12. アプリケーションをパッケージングします。

    ヘルプの「*コンパイルとリンク*」トピックを参照してください。

この順序に必ずしも忠実に従う必要はありません。データを定義するまえにオブジェクトを定義することも可能です。しかし、ダイアログに取り付けるオブジェクトを定義しないうちにオブジェクトダイアログを定義することはできません。

第3章　データ定義とスクリーンセットの作成

次の節では、Dialog Systemを使ってアプリケーションを作成するために必要な手順について説明しています。

3.1 データモデルの設計

「*Dialog Systemの概要*」の章で見てきたように、Dialog Systemアプリケーションを開発する最初の手順は、データ定義を作成するときに利用できる、データモデルを設計することです。データモデルは、インタフェースを実行したときに、スクリーンセットが呼出しプログラムに渡さなければならない、基本的なデータ項目を定義したものです。

データ定義は、呼出しプログラムのデータモデルをベースにします。データ定義の作成をはじめる前に、ユーザーインタフェースが得るデータを識別します。このデータは、スクリーンセットが呼出しプログラムに渡す基本的なデータ項目になります。また、これらの項目に対する妥当性検査の基準も決めます。

3.1.1 データと妥当性検査の定義

開発サイクルの手順では、まず最初に、入力データと出力データのデータモデルを作成します。このデータモデルには、次のようなデータの特性があります。

- データタイプ。たとえば、整数、文字列、計算など。
- データサイズ。
- 妥当性検査の必要条件。
- データ項目間の関係。

モデルを作成すると、Dialog Systemとプログラムの間で渡されるデータを定義することができます。

スクリーンセット全体に関して、設計するときに考慮すべきことがいくつかあります。詳しくは、「*スクリーンセットの使い方*」の章の「*スクリーンセットの使用の管理*」を参照してください。

3.2 データ定義

インタフェースとアプリケーションプログラムの間でデータを渡すために、入力フィールドなどのいくつかのオブジェクトを、アプリケーションプログラムに定義されたデータ項目(マスタフィールド)に関連付けます。これがデータ定義です。

マスタフィールドは、ユーザーが入力フィールドに入力したデータを保持します。コントロールの属性を編集するときに、適切なマスタフィールド を選択することによって、関連付けを行います。

3.2.1 プロンプトモードと非プロンプトモード

Dialog Systemでは、プロンプトモードを提供することによって、経験の浅いユーザーでもデータを入力できるようにしています。プロンプトモードと非プロンプトモードの違いは、次のとおりです。

- プロンプトモードでは、入力する行のタイプに応じて、注釈、フィールド、グループ開始または終了のいずれかのプロンプトを表示します。
- 非プロンプトモードで入力する場合、行の入力が終わると、入力項目は書式化されます。項目はスペースで区切る必要があります。

3.2.2 注釈

データ定義に注釈を付けるのは、よい習慣といえます。アスタリスクで始まる行は、どれも注釈として扱われ、妥当性検査または再書式化は行われません。 たとえば、次はコメント行の例です。

```
* これはコメント行です
```

3.2.3 データ定義作成の手順

以下の手順にしたがって、データ定義を作成します。

- データ定義に含ませるために、データ項目を識別する
- データタイプコードを割り当てる
- データグループを使う
- データの従属性を調べる
- 妥当性検査の基準を指定し、エラーメッセージを関連付ける

3.2.4 データブロック

データブロックには、Dialog Systemのデータ定義ウィンドウで定義するデータ項目があります。スクリーンセットはそれぞれユニークなデータブロックを持っており、コピーファイルとして呼出しプログラムに組み込まれます。

実行時には、Dialog Systemを呼び出すまえに、呼出しプログラムがこのレコードにユーザーデータを移動します。Dialog Systemがデータブロックに配置したユーザー入力は、制御が戻ったときに、プログラムによって使用されます。

関連するデータ項目は、グループに置くこともできますが、リストボックスなどのオブジェクトにマップされます。

データ定義ウィンドウのデータブロックは、表示、定義、編集することができます。データ定義ウィンドウを表示するには、次のように行います。

- メインウィンドウの[スクリーンセット]メニューから[データブロック]を選択します。

データブロックの簡単なサンプルを、次に示します。

フィールド名	書式	長さ	妥当性検査
GROUP-ITEMS	Group	10	
FIELD-1	9	4.00	R
FIELD-2	X	10.00	R
FIELD-3	9	6.00	R
GROUP-POSITION	9	2.00	
S-FIELD-1	9	4.00	
S-FIELD-2	X	10.00	
S-FIELD-3	9	6.00	
COMMENT	X	40.00	
COMMENTS-TITLE	X	25.00	
ARRAY-SIZE	9	2.00	
OBJECT	OBJ-REF		

データブロックの書式は、Dialog Systemによってコントロールされます。

- プロンプトモードで作成された入力は、自動的に書式化されます。
- 非プロンプトモードで作成された入力は、行の入力が終わると、書式化されます。
- 行の中の各項目は、空白で区切られます。

エラーが起こると、行の再入力を促すプロンプトが表示されます。データ定義の入力は、すべて大文字に変換されます。

注: OBJ-REFは、固定長のため、長さの入力は必要ありません。

データ定義は、スクリーンセットにCOBOLコピーファイルを生成したときに、コピーされます。この*screenset-name.cpb*と呼ばれるコピーファイルは、呼出しプログラムとDialog System間で渡されるデータブロックを構成します。スクリー

ンセットを実行すると、呼出しプログラムは、Dialog Systemを呼び出す前に、データブロックにデータを移動します。Dialog Systemは、コントロールを呼び出しプログラムに戻す前に、データブロックにデータを移動することができます。

3.2.4.1 データブロックコピーファイル

データブロックコピーファイルには、スクリーンセットデータ項目と関連するフィールド番号についての説明があります。

コピーファイルとは、COBOLレコード定義のことであり、コンパイル時に呼出しプログラムに組み込まれ、スクリーンセットの一部として定義したデータ項目に完全に依存します。これらを変更する場合は、適合するようにプログラムを変更する必要があります。

これを確実にするために、チェックの機構が利用されます。データブロックコピーファイルの書式は、次のようなものです。

- データブロックは、COBOLプログラムの作業場所節のlevel-01レコードと一致します。
- 単一のデータ項目は、レコードのlevel-03データ項目と一致します。
- データグループは、level-03グループデータ項目に一致します。
- データグループ項目は、level-05データ項目に一致します。

3.2.4.2 データ項目

データ定義に定義されたすべてのデータ項目は、スクリーンセットではグローバルであり、スクリーンセットダイアログで自由に参照することができます。

標準的なデータ項目は、入力フィールドまたはリスト項目にマップします。データ項目は、タイプによって分類され、グループに配置され、リストボックスなどのオブジェクトにマップされます。データ項目は、フラグとして設定して、ダイアログから参照したり、呼出しプログラムに値を運ぶことができます。

データブロックの各データ項目には、次のものがあります。

- 最大20バイトまでの名前。
- 呼出しプログラムがデータタイプを決めることができるデータタイプコード。データタイプコードは、次のどれかです。

  X　英数文字。

  9　数値(最大18文字までの)、整数部と小数部を含みます。少数点以下の小数部の桁数は、9を超えてはいけません。

A　英文字。

S　符号付き数値。

C　計算(バイト数は、1.0、2.0、4.0または8.0でなければなりません)。 計算データ項目は、どの入力フィールドにも、表示または設定することはできません。

C5　COMP-5 (バイト数は、1.0、2.0、4.0または8.0でなければなりません)。 COMP-5データ項目は、どの入力フィールドにも、表示または設定することはできません。

N　DBCS (N)。2バイト文字セット。

G　DBCS (G)。2バイト文字セット。

OBJ-REF　オブジェクト参照、クラスオブジェクトまたはインスタンスオブジェクトへの参照を格納するために使われます。

- データ項目のサイズ。この項目の意味は、選択したデータタイプによって、次のように変わります。

データタイプコード	サイズフィールドの意味
XまたはA	文字数。
9またはS	小数点の上位と下位の桁数。
CまたはC5	格納されたバイト数。サイズは格納できる数の範囲を、次のように決めます。 1.00から255まで 2.00から65, 535まで 4.00から4,294,967,295まで 8.00から18,446,744,073,709,551,615まで 状況によっては、Dialog Systemが値の最後の18桁だけを操作することに注意しましょう。
NまたはG	文字数。各文字は2バイトを占めます。
OBJ-REF	オブジェクト参照のサイズは、固定となります。オブジェクト参照のサイズを指定することはできません。

次のサンプルは、数値および符号付き数値データ項目に入力できる、最大値および最小値の範囲を示しています。

```
LARGEST-VALUE              9     18.0
```

```
SMALLEST VALUE            S     0.9
```

次のサンプルは、小数点の上位3桁と下位2桁の数値データ項目に対する、データブロックの入力を示しています。

```
ARBITRARY-DATA-NAME       9     3.2
```

データ項目の外観は、提示中の入力フィールドの絵文字によってコントロールされます。

次のサンプルは、オブジェクト参照の定義方法を示しています。

```
MAIN-WINDOW-SBAR-OBJREF       OBJ-REF
```

3.2.5 データグループの使い方

データグループは、同じ種類のデータ項目の集まりが、複数にわたって発生したり繰り返される場合に利用されます。データグループのデータ項目は､データブロックおよび呼び出しプログラムの添え字付けによって参照されます｡データグループは、呼出しプログラムから見ると、隣接するデータ項目であり、各項目は、インデックスとしてサブスクリプトを使って、グループの位置に相対してアクセスされます。データグループが繰り返しを1回だけ行うものとして定義されると、データ項目は呼出しプログラムの中に添え字付けを保持しません(しかし、Dialog Systemでは添え字付けを使う必要があります)。

グループの入力項目は、グループ名とたくさんの繰り返しから構成されます。たとえば、次のようになります。

```
GROUP-1                   150
```

あるグループに属すデータ項目は、グループ名の下にリストされ、インデントが付けられます。たとえば、次のようになります。

```
GROUP-1                   150
    DATA-ITEM-1             X    10.0
    DATA-ITEM-2             X    10.0
    DATA-ITEM-3             X    10.0
```

データブロックのデータ項目の順序は、それを削除したり再定義したり、データ定義ウィンドウの[編集]メニューから[コピー]、[貼り付け]、[削除]ファンクションを使う以外には、変更することはできません。

3.2.6 従属度

従属度とは、データ項目、オブジェクト、ダイアログのあいだの関連を説明するために使われる用語です。従属度は、次のように、2つのカテゴリに分類されます。

- データ項目がオブジェクトのマスタフィールドであるために、オブジェクトに依存するもの。
- データ項目がダイアログによって参照されるために、依存するもの。

メインウィンドウの[ビュー]メニューから[従属度]を選択したり、データ定義ウィンドウの[オプション]メニューから

[ビューの従属度]を選択することによって、名前が付けられたオブジェクトの従属度を問い合わせることができます。

Customerサンプルスクリーンセットの従属度を示す、従属度クエリーダイアログボックスを、図 3-1 に示します。

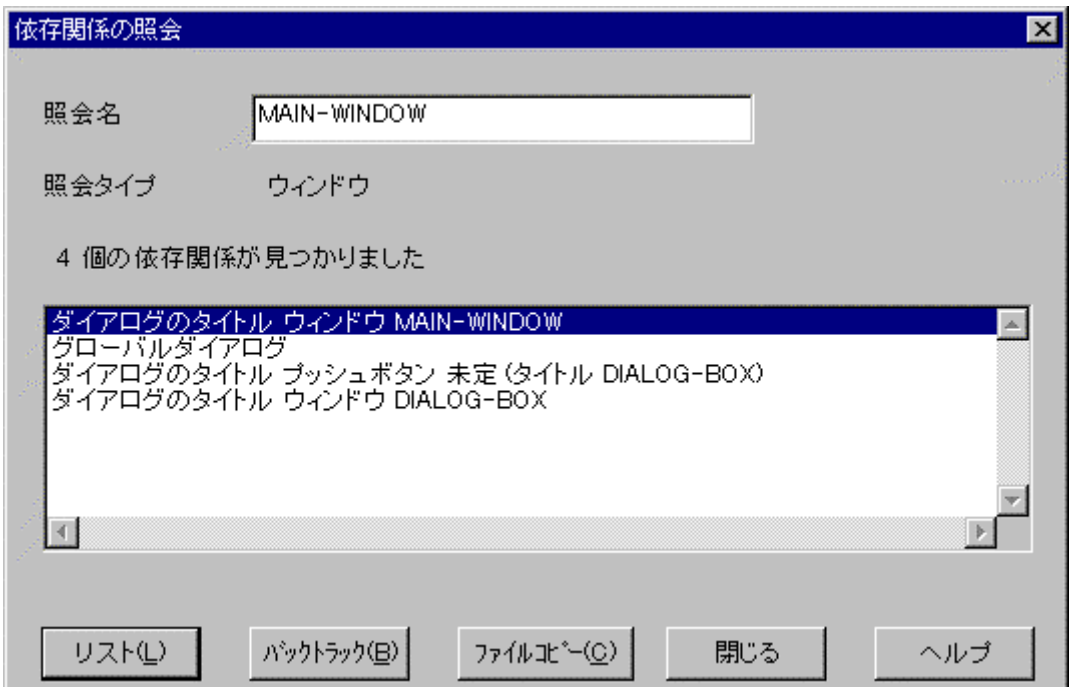


図 3-1 従属度クエリーダイアログボックス

データ定義ウィンドウから従属度を問い合わせると、データ定義ウィンドウで現在ハイライトされている行のデータ項目の従属度について、ダイアログボックスが説明します。メインウィンドウから従属度を問い合わせると、メインウィンドでハイライトされているオブジェクトの従属度を、ダイアログボックスが説明します。

従属度クエリーダイアログボックスでは、ダイアログボックスに表示されている、データ項目またはオブジェクトの従属度をリストすることができます。関連する行をハイライトし、次に[リスト]をクリックします。この機能によって、必要に応じて、従属度を連鎖的に追うことができます。

3.2.7 ユーザーデータの妥当性検査

Dialog Systemでは、ユーザーデータの入力について、特定の基準に対する妥当性を、入力フィールドを介して、チェックすることができます。

ダイアログにVALIDATEファンクションを使ったときだけ、妥当性検査は実行されます。たとえば、メインウィンドウのすべての入力フィールドをチェックするには、次のように実行することができます。

```
VALIDATE MAIN-WINDOW
```

メインウィンドウに選択ボックスを追加する場合、このファンクションはこれらの妥当性検査も行います。単一行の入力フィールドまたは選択ボックスの妥当性検査を行うこともできます。

妥当性検査が不成功の場合は、VAL-ERRORイベントが生成されます。VAL-ERRORイベントはたくさんの方法で使

うことができます。たとえば、妥当性検査が不成功になったフィールドに入力フォーカスを戻して設定することができます。この場合、$EVENT-DATAレジスタには、妥当性検査が不成功になった入力フィールドのIDが含まれます。

```
VAL-ERROR
    SET-FOCUS $EVENT-DATA
```

エラーメッセージを表示するために、メッセージボックスを使うことができます。1つのメッセージを表示するメッセージボックスを定義するか、適切なエラーメッセージのテキストをメッセージボックスに移すことによって、さまざまなメッセージを表示できるように定義します。

2番目の方法でエラーメッセージを表示するには、データ定義にエラーメッセージとエラーメッセージのフィールドを定義する必要があります。Customerサンプルスクリーンセットでは、エラーメッセージフィールドにERR-MSGを使用しています。VAL-ERRORイベントが起こると、次のダイアログが処理されます。

```
VAL-ERROR
    SET-FOCUS $EVENT-DATA
    INVOKE-MESSAGE-BOX FIELD-ERROR ERR-MSG $EVENT-DATA
```

FIELD-ERRORメッセージボックスは、ERR-MSGの内容を表示するために使用されます。ダイアログの3行目の$EVENT-DATAレジスタは、メッセージボックスへの応答として、ユーザーがどのボタンをクリックしたか(たとえば「OK」または「キャンセル」)を示すコードを記述するために使用されます。

3.2.7.1 妥当性検査の基準

次の妥当性検査の基準を、組み合わせて使うことができます。

範囲/テーブル	データがゼロを含む指定値の範囲内または範囲外にあります。必要に応じていくつかの範囲を指定することができます。指定された値は、数値または英数文字です。2バイトの範囲を使うことはできません。
日付	データは指定書式の有効な日付です。DBCSデータ項目にデータの妥当性検査を実行することはできません。
Null	データはNullまたは非Null(ゼロまたは空白)です。Nullは、DBCSの空白、英文字または英数文字のフィールド、数値フィールドのゼロです。データ項目がすべてNullを含む場合は、妥当性検査が不成功になるように選択することができます。
ユーザー定義	VALIDATEファンクションの処理の一部として呼び出されるプログラム名を指定します。プログラム名は、パスを含み最大256文字の長さが可能です。

Customerサンプルスクリーンセットでは、次の3つのデータ項目の妥当性検査が行われています。

- C-LIMIT – 1000から5000までの範囲/テーブルの妥当性検査。

- C-AREA – N、S、E、Wの値の範囲/テーブルの妥当性検査。
- C-ORD-DT – DDMMYY形式の日付の妥当性検査。

VALIDATEファンクションとVAL-ERRORイベントについて、詳しくは、ヘルプの「*ダイアログ文:ファンクション*」および「*ダイアログ文:イベント*」の各トピックを参照してください。

3.2.8 エラーメッセージ定義

データの妥当性検査を指定する場合は、エラーメッセージに関するオプションがあります。このエラーメッセージは、妥当性検査が不成功に終わった場合に、データブロックに定義されたエラーメッセージフィールドに設定されます。エラーメッセージを使う方法は、要件によって変わります。

次の規則は、エラーメッセージの定義に適用されます。

- すべてのエラーメッセージは、スクリーンセットにグローバルに適用されます。
- エラーメッセーフィールドに同時に保持できるエラーメッセージの文字列は一つだけです。次のエラーメッセージが、エラーメッセージフィールドの既存の内容を上書きします。
- スクリーンセットに同時に関連付けられるエラーメッセージファイルは一つだけです。

エラーメッセージを妥当性検査に関連付けるには、データ定義ウィンドウで妥当性検査が行われる項目をハイライトします。[妥当性検査]メニューを使って、関連する妥当性検査ダイアログボックスを表示し、[エラー]をクリックします。エラーメッセージ定義ダイアログボックスが表示されます。

デフォルトのエラーメッセージファイルは、dserror.errです。新しいエラーファイルを作成したり、[ファイル]を選択して、既存のエラーファイルをロードすることができます。

Customerサンプルスクリーンセットでは、エラーファイルcustomer.errがすでに定義されています。このファイルを選択して、[OK]をクリックすると、エラーメッセージ定義ダイアログボックスに戻ります。

Customerサンプルプログラムのエラーファイルには、3つのエラーメッセージがあります。

- 001 地域コードは、N、S、EまたはWのどれかにする必要があります。

  データ項目 C-AREA の妥当性検査エラーに使います。
- 002 クレジットの限度額は、1000 - 5000の範囲にする必要があります。

  データ項目 C-LIMIT の妥当性検査エラーに使います。
- 003 日付は、DD/MM/YY形式にする必要があります。

  データ項目 C-ORD-DT の妥当性検査エラーに使います。

新しいエラーメッセージを指定するには、[エラー番号]に番号を、[エラーテキスト]にエラーテキストを指定してから、[挿入]を押します。

エラーメッセージを選択して、妥当性検査に関連付けるには、メッセージをハイライトにして、[OK]をクリックします。妥当性検査ダイアログボックスに戻ります。選択したエラーメッセージの数は、[エラーメッセージ番号]に表示されます。

妥当性検査が、前述のように、エラーメッセージのリストのデータ項目へのデータ入力のどれかで不成功になった場合は、VAL-ERRORイベントが発生し、適切なエラーメッセージが、エラーメッセージフィールドERR-MSGに設定されます。

また、メインウィンドウから[スクリーンセット]メニューの[エラーメッセージ]を選択することによって、エラーメッセージダイアログボックスを使うこともできます。これによって、エラーメッセージとエラーメッセージファイルを編集することができますが、妥当性検査にメッセージを関連付けることはできません。

エラーメッセージファイルの作成とエラーメッセージの定義については、ヘルプの「*データ定義と妥当性検査*」のトピックを参照してください。

3.2.9 オブジェクトの選択

アプリケーション開発プロセスの次の手順は､アプリケーションに適切なオブジェクトを選択することです｡オブジェクトは、Dialog Systemの構築単位であり、「*ウィンドウオブジェクト*」と「*コントロールオブジェクト*」の2つの章で説明されています。

3.2.10 詳細情報

データ定義については、ヘルプの「*データ定義と妥当性検査*」トピックで見ることができます。特に、データグループの使い方、データグループの内部サイズ、生成されたデータブロックの内容について記載されています。

第4章　ウィンドウオブジェクト

前の章では、Dialog Systemで使用される基本概念と、グラフィカルユーザーインタフェースの設計に必要な、最初の手順について説明しました。この章では、ユーザーが作成するアプリケーションに適したウィンドウオブジェクトについて説明します。

オブジェクトとは、Dialog Systemの構築単位のことをいいます。オブジェクトには2つの種類があります。「*コントロールオブジェクト*」の章で説明するコントロールオブジェクトと、この章で説明するウィンドウオブジェクトです。この章では、次の内容について説明します。

- ウィンドウの構成要素。ウィンドウの属性と特定の属性をウィンドウに設定する理由も説明します。
- 一次ウィンドウと二次ウィンドウの関係とその効果。効果の例にはクリッピングなどがあります。
- ウィンドウ定義の代替方法。
- メニューバーオプションの操作方法。

ウィンドウは、ユーザーインタフェースの視覚的要素の基礎となるオブジェクトです。ウィンドウは、たいへん柔軟性に富んでいます。ユーザーによるサイズの変更が可能であり、プルダウンメニューを伴うメニューバー形式のメニューも備えています。他のウィンドウを含むあらゆるオブジェクトをウィンドウに配置することができます。

4.1 画面レイアウトのためのオブジェクト定義

構築するユーザーインタフェースは、あらゆる形態をとることができます。単純なメニュー選択とデータ入力フィールドから、メニューバー、ラジオボタン、その他のコントロールを完備した、十分に開発されたウィンドウにいたるまで、さまざまなものがあります。

どのくらい高度なインタフェースを選択するかは、次に示す、多くの要素によって決まります。

- 対称となるユーザー、およびユーザーのトレーニングのレベル
- 利用可能なハードウェアとソフトウェアのリソース
- 実行されるタスクの複雑さ

すべての場合について、次のことがいえます。

- ユーザーは、アクションの一般的な経過を決定する
- プログラムというよりはユーザーが、インタフェースを管理する
- ユーザーのコンピュータとの対話は、できるかぎり単純で自然なものにする

4.2 ウィンドウの構成要素

標準的なウィンドウの視覚的な構成要素を図 4-1 に示します。

図 4-1 標準的な GUI ウィンドウ

ウィンドウ用に選択する構成要素は、そのウィンドウで何を表示し、ユーザーがアプリケーションに対してどのように反応してほしいかによって異なります。

タイトルバー	ウィンドウの一番上にある領域で、ウィンドウのタイトルが入っています。図 7-1 に示すウィンドウのタイトルは一次ウィンドウです。
最小化ボタンと最大化ボタン	ウィンドウの右上にあたるタイトルバーの隣にあります。これらのアイコンを使うと、ウィンドウを簡単に最小化または最大化したり、ウィンドウをもとのサイズに戻すことができます。
境界線	ウィンドウの範囲を定義します。境界線は、サイズ変更できる（ユーザーがウィンドウのサイズを調整できる)場合とサイズ変更できない（ウィンドウのサイズが固定されている)場合があります。ウィンドウは、境界線を付けずに表示することもできます。
メニューバー	タイトルバーのすぐ下にあります。メニューバーには、アプリケーションがユーザー

	に提供するオプションがあります。図 4-1 では、「ファイル」、「編集」、「表示」、「ヘルプ」 が選択できます。
	GUI は、各メニューバー項目についてプルダウンメニュー（アクションメニューとも呼ばれる)を持っているのが通常です。
	ウィンドウに表示されている情報量と位置の目安を視覚的に示します。スクロールバーを使うと、表示される情報を調整することができます。
スクロールバー	水平スクロールバー-ウィンドウの下の境界の上にあります。横方向の表示を調整します。
	垂直スクロールバー-ウィンドウの右の境界の左にあります。縦方向の表示を調整します。
システムメニュー	タイトルバーの左側にあります。各環境で利用できる標準のウィンドウ操作機能をメニューまたはキーボードによってアクセスします。
クライアント領域	ウィンドウの残りの部分。この領域には、コントロールがあります。また、ユーザーはほとんどのアプリケーションタスクをここで行います。

4.3 デスクトップ

デスクトップとは、画面上の作業領域のことです。このデスクトップでウィンドウを作成することができます。インタフェースでは、非常に多くのウィンドウを持たせることができます(ヘルプの「*Dialog Systemの制限*」のトピックを参照してください)が、インタフェースの最初の実行で表示されるウィンドウは1つだけです。このウィンドウは、最初のウィンドウと呼ばれます。

デスクトップでは、次の種類のウィンドウを作成することができます。

- 一次ウィンドウ
- 二次ウィンドウ

4.3.1 一次ウィンドウ

最初に作成したウィンドウは、一次ウィンドウと呼ばれます。他のウィンドウが上位に存在しないため、デスクトップの子(またはデスクトップを親ということができます)にあたります。これは、他のどのウィンドウにも依存しないで存在するため、他のウィンドウのどのアクションも、一次ウィンドウに影響を与えることはありません。複数の一次ウィンドウを作成することができます。

4.3.2 二次ウィンドウ

あらゆる一次ウィンドウは、二次ウィンドウと呼ばれる、子ウィンドウを持つことができます。このウィンドウもまた他の二次ウィンドウを持つことができます。二次(または子)ウィンドウは、親ウィンドウをサポートするために利用されます。二次ウィンドウのアクションの多くは、一次ウィンドウの範囲によって決まります。二次ウィンドウを表示すると、それに関連する一次ウィンドウもまた表示されます。このウィンドウ間の関係は、ユーザーがアプリケーションと対話する方法に重大な影響を与えます。

たとえば、個人ユースのアプリケーションには、一次ウィンドウの新入社員ウィンドウがあり、住所、近親者、その他の個人情報を入力するための複数の二次ウィンドウがあります。

最初のウィンドウとして二次ウィンドウを選択すると、スクリーンセットをはじめて実行したときに、二次ウィンドウとその親ウィンドウの両方が表示されます。

4.3.3 一次ウィンドウと二次ウィンドウの関係

一次ウィンドウと二次ウィンドウの関係をあらわす特性には、次のものがあります。

- 一次ウィンドウを削除すると、そのウィンドウに関連する、すべての二次ウィンドウも削除されます。
- 一次ウィンドウを移動すると、クリップされたすべての二次ウィンドウも共に移動し、一次ウィンドウでの相対的な位置が保たれます。
- 一次ウィンドウを最小化すると、その二次ウィンドウはすべて画面から消えます。一次ウィンドウを元の位置に戻すと、すべての二次ウィンドウも、元の位置に戻ります。

注: 一次ウィンドウの親は、デスクトップです。一次ウィンドウは、実行時にSET-DESKTOP-WINDOWファンクションを使って、他のウィンドウの子に変更することができます。このファンクションの説明は、「*ダイアログの使い方*」の章の「*デフォルト親ウィンドウの変更*」の項目と、ヘルプの「*ダイアログの文: ファンクション*」のトピックを参照してください。

二次ウィンドウとダイアログボックスは、情報を表示し収集するという点で、機能的によく似ています。「*ダイアログボックス対ウィンドウ*」では、いつダイアログを使い、いつ二次ウィンドウを使うのか、ヒントを提供しています。

4.4 クリップ

Dialog System では、二次ウィンドウを一次ウィンドウ内でクリップするかどうかを選択することができます。あるウィンドウの情報がその親ウィンドウの境界によって切り取られている場合、そのウィンドウはクリップされているといいます。

クリップされた二次ウィンドウとクリップされていない二次ウィンドウを図 4-2 に示します。

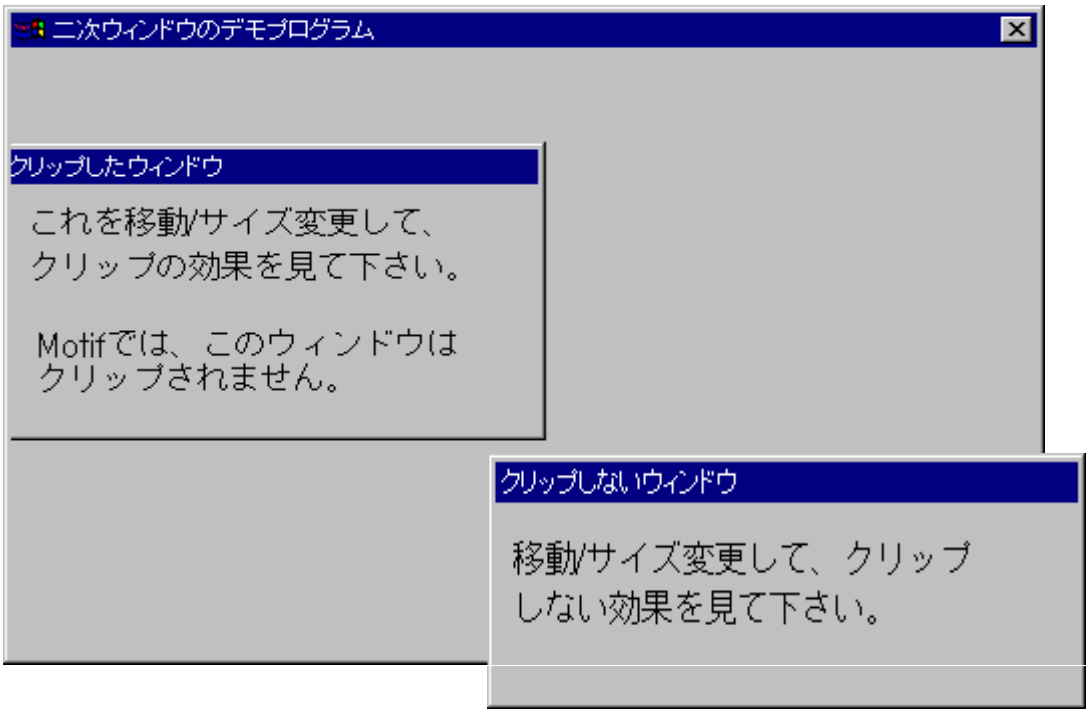


図 4-2 クリップされたウィンドウとされてないウィンドウ

画面をある特定の視覚的レイアウトに保ちたい場合、二次ウィンドウをクリップするとよい場合があります。たとえば、ユーザーが見慣れている用紙形式に近いようなウィンドウを表示する場合、クリップされた二次ウィンドウを使用すると、視覚的な同一感を保つことができます。

一方、クリップされていない二次ウィンドウは柔軟性をもたらします。ユーザーは、二次ウィンドウをデスクトップ上の任意の場所に移動でき、好みの画面レイアウトにすることができます。

サンプルアプリケーション Objects は、クリップされた二次ウィンドウとクリップされていない二次ウィンドウの効果を示しています。

4.5 ウィンドウの定義

他のウィンドウオブジェクトを定義する前に、一次ウィンドウを定義する必要があります。そのあとで、クリップされた二次ウィンドウまたはクリップされていない二次ウィンドウのどちらかを定義することができます。

定義したウィンドウがどのタイプであっても、メインウィンドウの左上角にボックスがあらわれます。ウィンドウを配置してサイズを設定するには、マウスポインタを移動してウィンドウの左上角を任意の位置に置き、そのあとでマウスをクリックして固定します。次に、ウィンドウの右下角の任意の位置までマウスを移動し、再度クリックして固定します。すると、新しいウィンドウが表示されます。

ダイアログボックスをクリップされた子ウィンドウの子として指定すると、その親は実際にはクリップされた子ウィ

ンドウではなく、クリップされた子ウィンドウの親になります。作成時には、新しいダイアログボックスは、クリップされた子ウィンドウの相対位置に配置されますが、その後、クリップされた子ウィンドウの親の相対位置に配置されます。その結果、クリップされたウィンドウがクリップされない子ウィンドウを持つスクリーンセットを描くときには、制限があります。子ウィンドウは、通常の方法で新しい位置に正しく移動することができません。まず、その属性をクリップされたウィンドウに変更する必要があります。移動したあとで、クリップ属性は削除することができます。

ウィンドウのサイズには制約はなく、Dialog Systemメインウィンドウより大きくしたり、メインウィンドウの外側に配置することができます(デスクトップモードがオンの場合)。スクリーンセットを実行すると、Dialog Systemはユーザーが作成したとおりにウィンドウをすべて正確に配置してサイズを設定します。

4.5.1 ウィンドウ属性ダイアログボックス

ウィンドウの視覚的な構成要素については、この章の前半の「*ウィンドウの構成要素*」で説明しました。図 4-3 に示すように、ウィンドウ属性ダイアログボックスでは、ウィンドウの属性を選択することができます。


ウィンドウの属性
一般
ｵﾌﾟｼｮﾝ
高度なｵﾌﾟｼｮﾝ
名前
WIN1
ﾀｲﾄﾙ
Window Title
親
DESKTOP
ｱｲｺﾝｲﾒｰｼﾞ
OK
ｱｲｺﾝ(I)...
接続(C)
ｷｬﾝｾﾙ
ﾍﾙﾌﾟ


ウィンドウの属性
一般
ｵﾌﾟｼｮﾝ
高度なｵﾌﾟｼｮﾝ
境界のﾀｲﾌﾟ
境界なし(o)
境界不変(B)
境界可変(z)
装飾
ﾀｲﾄﾙﾊﾞｰ(T)
ｼｽﾃﾑﾒﾆｭｰ(S)
最小化ｱｲｺﾝ(N)
最大化ｱｲｺﾝ(X)
水平ｽｸﾛｰﾙ(H)
垂直ｽｸﾛｰﾙ(V)
ﾒﾆｭｰ(M)
OK
ｱｲｺﾝ(I)...
接続(C)
ｷｬﾝｾﾙ
ﾍﾙﾌﾟ


ウィンドウの属性
一般
ｵﾌﾟｼｮﾝ
高度なｵﾌﾟｼｮﾝ
ｽｸﾛｰﾙ
実行時ｻｲｽﾞ(R)
幅
1848
高さ
688
ｸﾘｯﾌﾟ(I)
OK
ｱｲｺﾝ(I)...
接続(C)
ｷｬﾝｾﾙ
ﾍﾙﾌﾟ

図 4-3 ウィンドウ属性ダイアログボックス

これらの属性については、ヘルプを参照してください。

4.5.2 ウィンドウの操作

一度、ウィンドウの視覚的な特性を定義すると、ダイアログを使ってウィンドウを操作することができます。詳しくは、「*ダイアログの使い方*」の章の「*ウィンドウダイアログ*」を参照してください。

4.6 ダイアログボックス

ダイアログボックスは、情報を表示または取得するのに、もっともよく利用されます。この情報は、テキストフィールド、データ入力フィールド、ボタンなどの、ダイアログボックスに表示されるコントロールを通じて表示されたり取得されます(最大255個のオブジェクトをダイアログボックスに配置することができます)。ダイアログボックスには、次の属性があります。

- サイズの変更ができない、または、メニューバーを持つことができません。
- Dialog Systemコントロールオブジェクトを追加できます。
- ダイアログボックスは、他のウィンドウまたはデスクトップによって作成され、作成したウィンドウの子と呼ばれます。作成したウィンドウは、ダイアログボックスの親と呼ばれます。
- ダイアログボックスは、ユーザーのアクションによって閉じられるまで、その場所に存在します。

  一般的には、ユーザーがボタンをクリックすると、Dialog Systemはユーザーのアクションを受け入れます(たとえば、[OK]をクリックすると、リストの選択項目を受け入れます)。あるいは、ユーザーがボタンをクリックすると、Dialog Systemはユーザーのアクションを無視することもあります(たとえば、[キャンセル]をクリックすると、ダイアログボックスは閉じて、入力は無視されます)。
- ウィンドウと同じダイアログを使って、ダイアログボックスを操作することができます。

図 4-4 では、標準的なダイアログボックスをいくつか示しています。

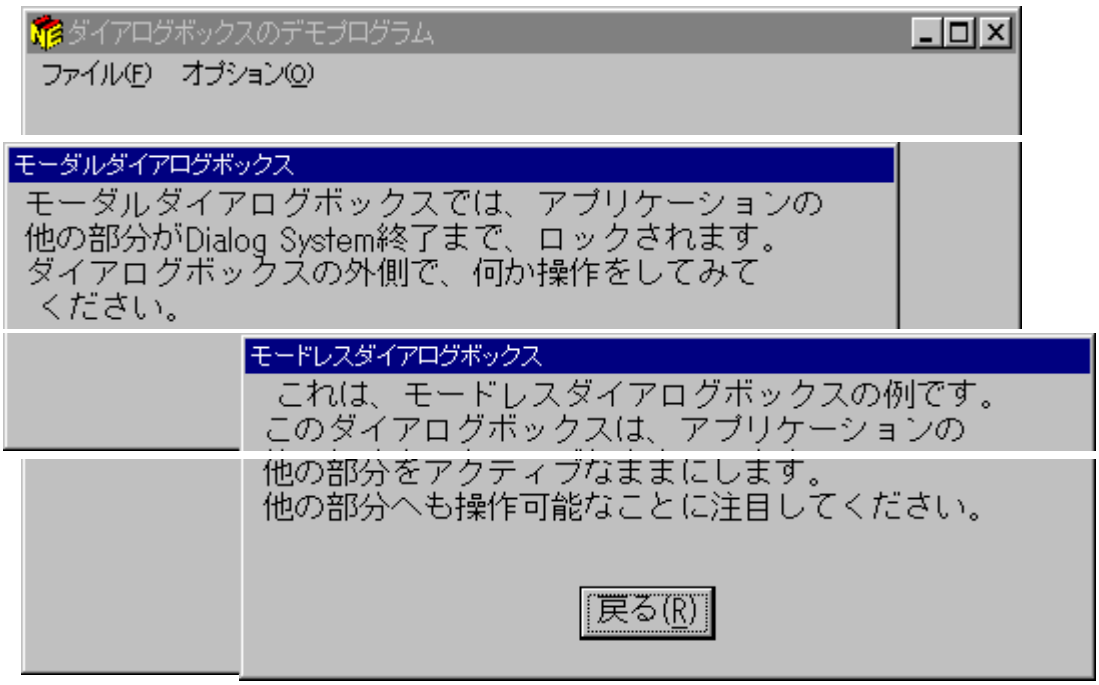


図 4-4 標準的なダイアログボックス

4.6.1 モーダルとモードレス

ダイアログボックスは、アプリケーションモーダル、親モーダル、またはモードレスにすることができます。その中のどれを選択するかは、ユーザーがどのようにダイアログボックスに応えるのかによって決まります。

親モーダルダイアログボックスは、ユーザーがダイアログボックスを終了するまで、ダイアログボックスが作成されたウィンドウが凍結状態であることを意味します。つまり、モーダルダイアログボックスが終了するまでは、ウィンドウにアクセスすることはできません。しかし、他の一次ウィンドウおよびアプリケーション外のオブジェクトには、アクセスすることができます。

アプリケーションモーダルダイアログボックスは、ユーザーがダイアログボックスを終了するまでは、ウィンドウ階層の他のすべてとアプリケーション内の全オブジェクトが凍結状態であることを意味します。

アプリケーションモーダルダイアログボックスを使って、ユーザーにファイル名を入力するよう問い合わせることができます。アプリケーションは、ユーザーがファイル名を入力する(またはアクションを取り消す)まで、先に進むことができません。

Dialog Systemそのものは、モーダルダイアログボックスを使用しています。たとえば、製品情報ダイアログボックスはモーダルです(このダイアログボックスを表示するには、[ヘルプ]メニューの[製品情報]を選択してください)。

モードレスダイアログボックスは、ダイアログボックスが表示されても、アプリケーションの他の部分がアクティブのままであることを意味します。表示または要求された情報は、アプリケーションを継続するにあたって、すぐに必要であるとはかぎりません。

モードレスダイアログボックスは、アプリケーションとの対話においてユーザーに最大限の柔軟性を提供するため、

より好ましい選択といえます。しかし、前述のように、アプリケーションモーダルダイアログボックスが必要な場合もあります。

Objectsサンプルアプリケーションでは、アプリケーションのモーダルダイアログボックスとモードレスダイアログボックスの違いについて説明しています。スクリーンセットを実行して、それぞれのオプションの効果を参照してみてください。

4.6.2 ダイアログボックス対ウィンドウ

これまで見てきたように、ウィンドウとダイアログボックスはとてもよく似ています。実際、ダイアログボックスはウィンドウの特別な種類といえます。「*グラフィカルユーザーインタフェース*」の章の「*ダイアログボックス*」を参照してください。

アプリケーションでは、ダイアログボックスとウィンドウのどちらを使うのか、しばしば選択に迫られることがあります。いつダイアログボックスを使い、いつウィンドウを使えばよいのかを決めるために、次のガイドラインを提案します。

次の場合は、ウィンドウではなくダイアログボックスを使います。

- アプリケーションがモーダル入力を必要とする場合。たとえば、アプリケーションがユーザーの入力を要求し、ユーザーが応じるまで続行できない場合は、モーダルダイアログボックスを使います。
- 提示または収集する情報量が少なく、メニューバーを必要とせず、ダイアログボックスに収まる場合。

次の場合は、ダイアログボックスではなくウィンドウを使います。

- メニューバーの選択可能なリストを選択するよう、アプリケーションがユーザーに要求する場合。
- 提示または収集する情報がウィンドウに収まらないために、ユーザーがウィンドウのサイズを変えたりスクロールして、データやコントロール全体を見なければならない場合。
- オブジェクトが子ウィンドウを持つ必要がある場合。ウィンドウを使うことによって、アプリケーションはより論理的な設計になります。

4.7 メッセージボックス

メッセージボックスは、ユーザーにメッセージを提供するために利用されます。メッセージボックスに用意されているのは､情報を表示するためのテキストまたはグラフィックスと､ユーザーがアクションを選択したりメッセージボックスを削除するためのボタンだけです。通常、これらのメッセージは、 "ファイルがみつからない"イベントのような、何らかのイベントの結果として、アプリケーションが表示します。

メッセージボックスのサイズと位置を定義することはできません。

図 4-5 は、標準的なメッセージボックスを示しています。

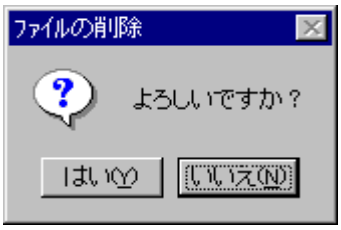


図 4-5 標準的なメッセージボックス

Dialog Systemでは、メッセージの性質によって、次のような汎用のメッセージボックスを提供しています。アイコンがそれぞれ違うために、各タイプを識別することができます。

通知	応答しても、しなくてもかまわないような状況をユーザーに通知します。たとえば、ユーザーにプリンタの用紙切れを知らせることは、ユーザーの現在の作業には影響しません。
情報	ユーザーに情報を伝えます。メッセージを読んだという確認以外は、ユーザーからの応答は必要ありません。たとえば、製品情報メッセージは、情報を伝えるメッセージです。
警告	望ましくない結果が起こりうる状況を示します。たとえば、ユーザーが開いたファイルの数が、限度数に近づきつつあるときに、警告メッセージを発行することができます。
問い合わせ	ユーザーの応答が必要な状況を示します。たとえば、ユーザーが[ファイルの削除]オプションを選択すると、削除を確認するようユーザーに問い合わせるメッセージが表示されます。
致命的	ユーザーが操作を続行すると、望ましくない結果が引き起こされる状況を示します。たとえば、ユーザーがファイルの最大数を開き、さらに操作を続行すると、システムは停止してしまいます。

メッセージボックスには、ユーザーがメッセージに応答できるように、少なくとも1つのプッシュボタンが必要です。Dialog Systemでは、ユーザーが使用できるあらかじめ定義されたプッシュボタンの組み合わせを、次のように6つ提供しています。

- OK
- OK/キャンセル
- 再試行/キャンセル
- 中止/再試行/無視

- はい/いいえ
- はい/いいえ/キャンセル

たとえば、

- 情報を伝えるあらゆるメッセージで、ユーザーがメッセージを受け取ったことを確認する[OK]ボタンを使用することができます。
- ファイルを削除するかどうかの確認を求めるメッセージボックスで、[はい/いいえ]ボタンの組み合わせを使用することができます。

どのボタンをユーザーが選択したのかは、ダイアログを使って決めることができます。クリックされたボタンを識別するコードは、INVOKE-MESSAGE-BOXファンクションの説明にリストされています。ヘルプの「*ダイアログ文: ファンクション*」トピックを参照してください。

例として、ユーザーがメニューから[ファイルの削除]を選択したことを想定してみます。メッセージボックス(上記の図 4-5)が実行され、ユーザーは、ファイルを削除してもよいかを確認することができます。

4.8 メニュー

メニューには、3つのタイプがあります。

- メニューバー
- プルダウンメニュー
- コンテキストメニュー

4.8.1 メニューバー

メニューバーは、ウィンドウ?の最上部に位置するコマンドのリストです。もっと正確に表現するなら、コマンドのグループのリストといえます。というのも、通常のメニューバーは、あるコマンドを選択すると、次に選択する項目が提示されるからです。

メニューバーの定義について詳しくは、ヘルプの「*オブジェクトと属性*」のトピックを参照してください。

メニューの選択項目では、次のことができます。

- プルダウンメニューを追加します。選択項目に続いて矢印が表示されます。

  後述の「*プルダウンメニュー*」を参照してください。

- 二次ウィンドウまたはダイアログボックスを表示させます。選択項目に続いて省略記号(...)が表示されます。

- チェックマークが付きます。オンとオフの2つの状態があります。選択項目をクリックすると、状態が切り替わります。オンの場合は、チェックマークが選択項目の先頭に表示されます。このような選択項目は、トグルとも呼ばれます。

- 直接アクションを実行します。この機能を識別するための特別な表示は行われません。

メニュー項目を追加すると、上記のリストのどれか1つのアクションが実行され、それに応じて適切な表示(たとえば省略記号またはチェックマークなど)が行われます。

4.8.2 プルダウンメニュー

メニューバーの項目を選択すると、その項目からプルダウンメニューが表示されます。これらのプルダウンメニューから、オブジェクトに対してアクションを実行することができます。図 4-6 は、標準的なプルダウンメニューを示しています。

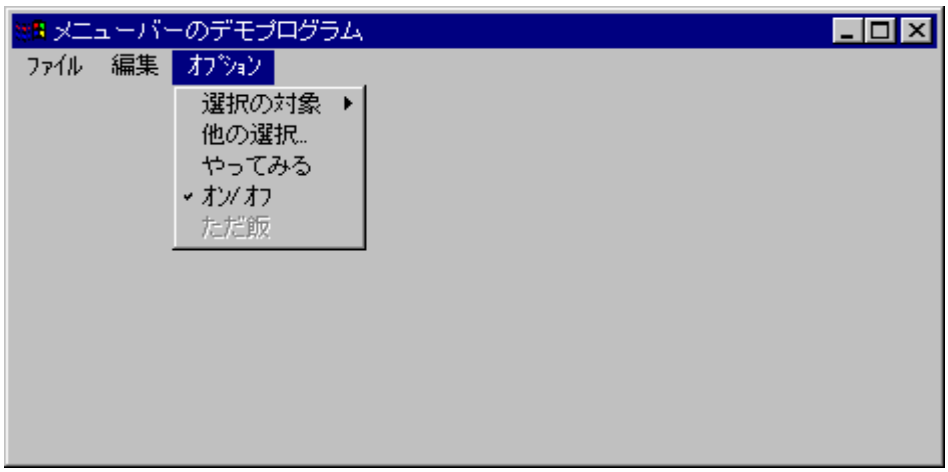


図 4-6 標準的なプルダウンメニュー

項目を選択したあとで起こるアクションには、いくつかの種類があります。通常、メニュー項目には、アクションの種類を示す視覚的なマークが表示されます。次のリストでは、そのマークとそれに関連するアクションについて説明します。

矢印	現在のメニュー項目の横に、別のプルダウンメニューを表示して、ユーザーが選択できるメニュー項目をさらに提供します。
チェックマーク	項目が有効かどうかを示します。チェックマークがオンの場合、その項目は操作中であることを示します。オフの場合、その項目は操作中ではありません。チェックマークの選択がオフの場合は、直接実行する通常のメニュー項目と見分けることができません。
淡色表示	その項目が選択できないことを示します。

省略記号(...)	ファイルのリストから選択したり、ラジオボタンから選択するなど、より複雑な選択項目を提供するダイアログボックスまたは二次ウィンドウを表示します。
マークなし	ただちにアクションを実行します。

メニューダイアログのサンプルについては、「*ダイアログの使い方*」の章の「*メニューバーダイアログ*」および「*選択の使用可能と使用禁止*」で検討しています。

ヘルプの「*オブジェクトと属性*」では、メニューバーの定義方法について説明しています。

4.8.3 コンテキストメニュー

コンテキストメニューは、ユーザーがDialog Systemの作業領域で操作するための、すばやく簡単な方法を提供します。画面上で右クリックすると、クリックした項目に適したメニュー選択項目が表示されます。たとえば、メイン定義ウィンドウのプッシュボタンオブジェクトと、ダイアログ定義ウィンドウのダイアログ行とでは、右クリックしたときに表示されるコンテキストメニューは異なります。

コンテキストメニューの例を、以下に示します。

[スクリーンセット]メニューから[グローバルダイアログ]を選択します。

ダイアログの任意の行を右クリックします。

Dialog Systemは図 4-7 に示すように、状況に応じたコンテキストメニューを表示します。

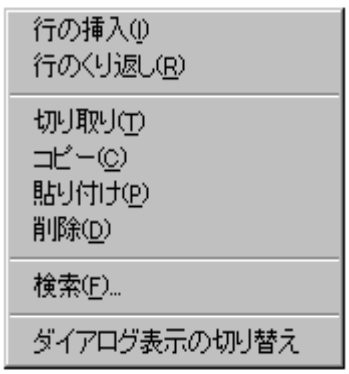


図 4-7 グローバルダイアログのコンテキストメニュー

4.9 アイコンの設定

アイコンとは、小さい絵図のことであり、ウィンドウをアイコン化したときに表示されるものです。ウィンドウまたはダイアログボックスを作成するときは、アイコン化したときに使用される、適切なアイコンを設定することができます。

アイコンをウィンドウに設定するには、次のように行います。

1. ウィンドウを選択します。

2. [編集]メニューから[属性]を選択します。

   ウィンドウ属性ダイアログボックスが表示されます。

3. [アイコン]をクリックします。

   図 4-8 で示すように、アイコン選択ダイアログボックスが表示されます。このダイアログボックスには、ビットマップファイルで利用可能なアイコン名のリストが表示されます。ds.icnファイルは、常に、デフォルトです。

   図 4-8 アイコン選択ダイアログボックス

4. リストからアイコン名を選択します(そのイメージが表示されます)。

5. [OK]をクリックすると、ウィンドウ属性ダイアログボックスに戻ります。

   アイコン化のアイコンを確実に選択するようにしてください。そうでなければ、そのアイコンは使用されないことになります。

6. [OK]をクリックすると、オブジェクト定義ウィンドウに戻ります。

第5章　コントロールオブジェクト

前の章では、ウィンドウオブジェクトについて見てきました。この章では、次の内容を説明します。

- コントロールオブジェクトとその使い方
- コントロールのグループ化
- コントロールの整列

5.1 コントロールオブジェクト

コントロールオブジェクトは、ウィジェットまたはガジェットとも呼ばれます。これらのオブジェクトは、ウィンドウまたはダイアログボックスの外側に存在することはできません。したがって、コントロールオブジェクトを定義するまえに、ウィンドウまたはダイアログボックスを定義し、選択しなければなりません。

コントロールは、その種類によって次のことに使用できます。

- データの入力（たとえば、代替値を提示し、ユーザーにいずれかを選択させる）
- 異なる表示間の移動
- データの提示

この章で説明するコントロールには、次のものがあります。

コントロール	用途
テキストフィールドと入力フィールド	データ項目の読込みと入力を行い、ラベル用のテキストを提供します。
プッシュボタン	ユーザーがすぐに実行する動作を選択できるようにします。
ラジオボタン	グループ化した場合、排他的な選択項目（1つしか選択できない）の固定されたセットをユーザーに提供します。
チェックボックス	排他的ではない選択項目（複数を選択できる）のセットをユーザーに提供します。ユーザーは、いくつかのまたはすべての項目を選択することができます。
リストボックス	ユーザーが1つまたは複数を選択する項目のリストを表示します。
選択ボックス	スクロール可能な選択項目のリストを表示し、選択したものを入力フィールドに配置します。

スクロールバー	入力フィールドなどの他のコントロールと共に使用した場合、ユーザーが特定の値まで「スクロール」することができます。
グループボックス	ユーザーにとって見やすくなるように、1つまたは複数のコントロールを囲みます。
タブコントロールと タブコントロールページ	普通のページとタブ見出しページからなるスパイラルノートのような形で情報を編成します。
OLEコントロール	外部のオブジェクト（例えばスプレッドシートなど）を呼出すことができるウィンドウを表示します。
ユーザーコントロール	Dialog System定義ソフトウェア以外のソフトウェアで作成したオブジェクトを取り込みます。ユーザーコントロールの使い方は、実装内容によって変わります。
ActiveXコントロール	あらかじめ定義された機能を持つオブジェクトを表示します。通常は、サードパーティが提供します。

5.1.1 テキストフィールドと入力フィールド

もっとも基本的なコントロールは、入力フィールドです。ここでは、次の内容について説明します。

- テキスト、単一行入力フィールドおよび複数行入力フィールドの特徴
- 他のコントロールと連携して入力フィールドを使う方法

5.1.1.1 テキストの表示（テキストオブジェクト）

テキストオブジェクトによりユーザーは、どんな情報が表示されているか、または求められているかを理解することができます。テキストフィールドはラベルと考えることができます。テキストフィールドの使用例を図 5-1 に示します。

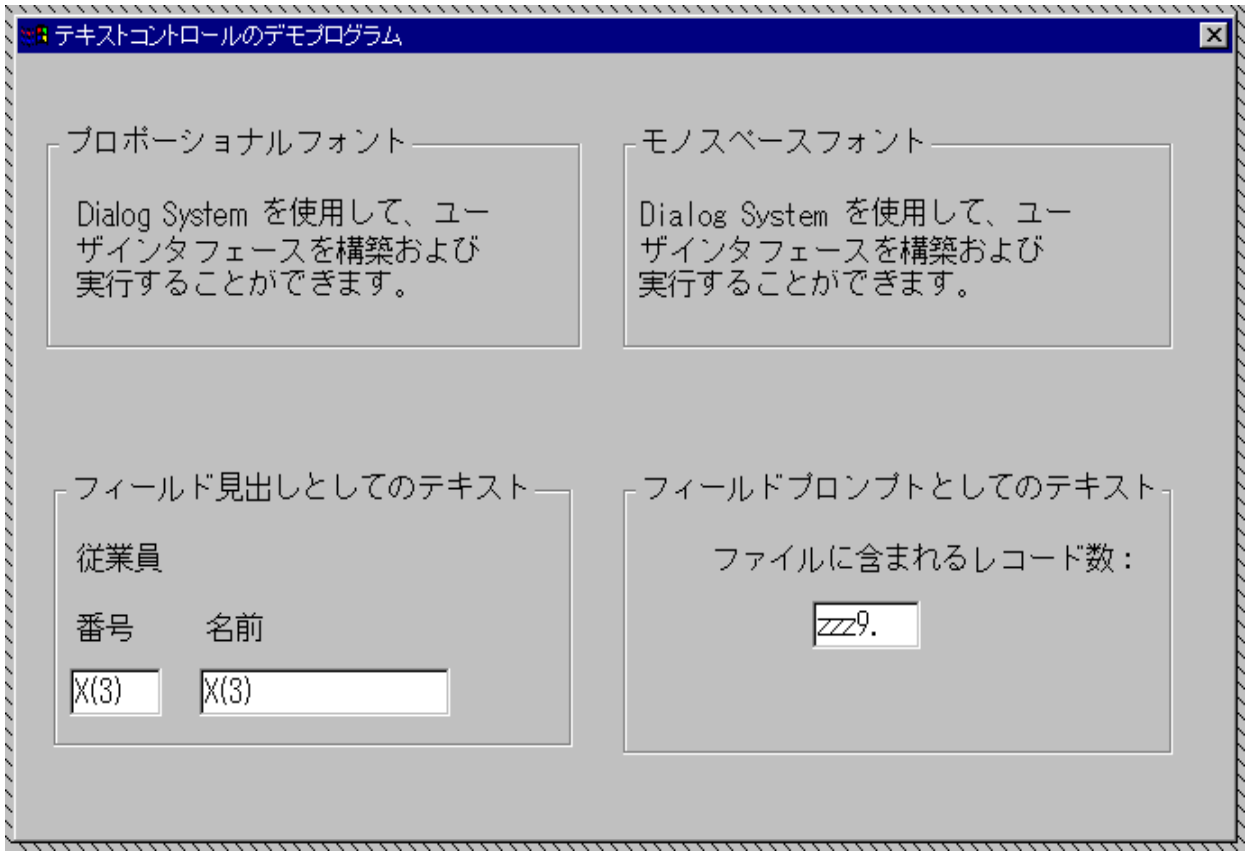


図 5-1 テキストフィールドの例

テキストフィールドは、次のものとして使用できます。

- フィールドまたはカラム見出し
- フィールドプロンプト
- 説明テキスト

テキストフィールドは静的です。つまりテキストコントロールには、イベントやファンクションは関連付けられません。ステータス行や動的プロンプトなど、実行時に変更できるテキストフィールドが必要な場合、表示専用入力フィールドを使用します。詳しくは、この章の「*表示専用入力フィールド*」の項を参照してください。

テキストフィールドの定義方法について詳しくは、「*オブジェクトと属性*」の章を参照してください。

5.1.1.2 入力フィールドによる入力の獲得

入力フィールドは、実行時にデータ項目の内容を表示し、ユーザーが内容を編集できるようにします。入力フィールドはデータ項目として可能
な値がはっきりしない場合に使用します。この場合、ユーザーが必要な情報を入力しなければなりません。.

データ項目の内容が変化した場合（たとえば、プログラムがデータ項目に新しい情報を挿入した場合）、変更を反映するまえに、入力フィールドをデータブロックから再生しなければなりません。入力フィールドは、次のような場合に再生されます。

- 実行時にその親ウィンドウが作成された場合
- それ自身か、その親ウィンドウが、ダイアログファンクション REFRESH-OBJECT によって明示的に再生された場合
- 入力フォーカスが設定されている場合

親ウィンドウがSHOW-WINDOWファンクションで表示されている場合、入力フィールドはリフレッシュ(再表示)されません。

数値入力フィールドにフォーカスを設定したとき、関連するマスタフィールドに有効な数値データが入っていない場合は、その値はゼロに設定されます。

Dialog System には、単一行入力と複数行入力（MLE）の2種類の入力フィールドがあります。

5.1.1.2.1 単一行入力フィールド

単一行入力フィールドは、最もシンプルな入力フィールドで、1行のテキストだけを表示または収集することができます。

例として、Customer サンプルでは、顧客コード、名前、住所、信用限界などの情報をユーザーが提供しなければなりません。これらの情報は、入力フィールドを使って入力します。

単一行入力フィールドの例を図 5-2 に示します。

図 5-2 入力フィールドの例

5.1.1.2.2 入力フィールドとコントロールとの使用

入力インタフェースを他のコントロールフィールドといっしょに使用することで、ユーザーが値を選択しやすくなります。

たとえば、データ項目の値が特定の範囲内にあることがわかっている場合、入力フィールドといっしょにスクロールバーを使用すると、ユーザーは「スクロール」によって適切な値を入力できます。

入力フィールドとスクロールバーを使用した例を図 5-3 に示します。

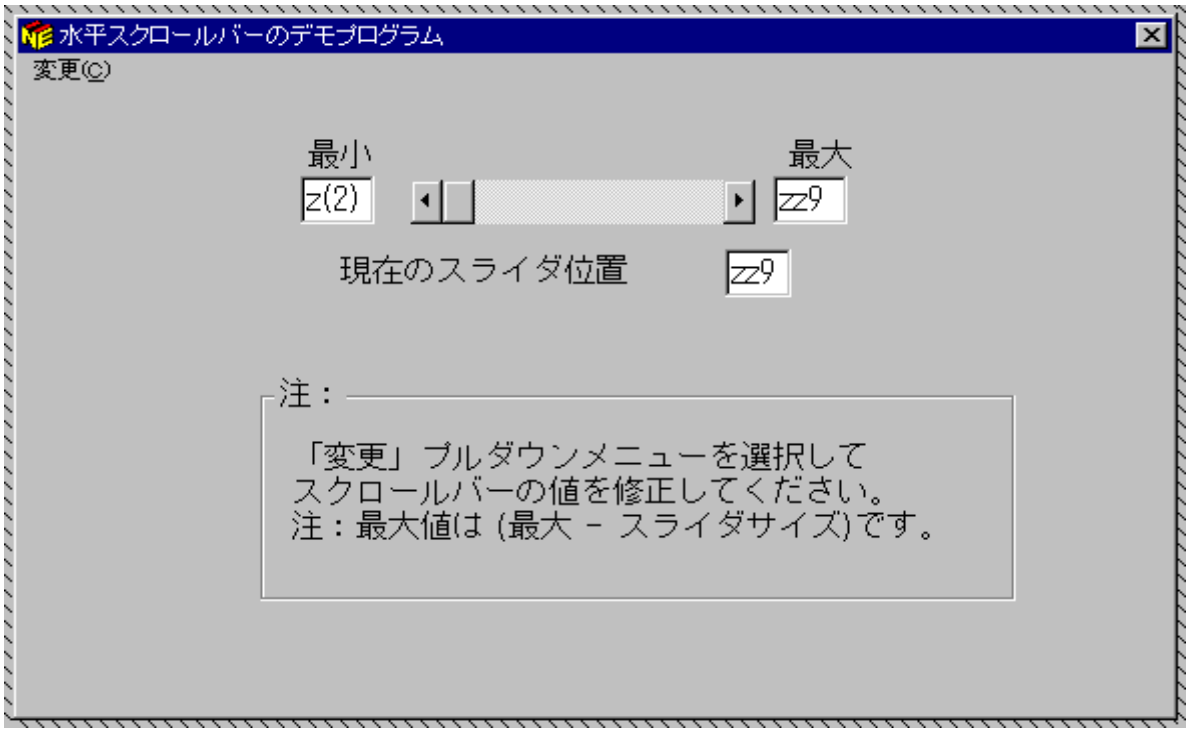


図 5-3 入力フィールドとスクロールバーの使用

この機能の実装に必要なダイアログの例は、「*サンプルプログラム*」の章で見ることができます。

ヘルプの「*データ定義と妥当性検査*」のトピックでは、データ定義を使った妥当性検査の設定方法について説明しています。

5.1.1.2.3 表示専用入力フィールド

入力フィールドによっては、表示専用にしたい場合があります。たとえば、ユーザーが従業員に関する住所情報を更新するためのダイアログボックスを作成したいとします。この場合、従業員の氏名などは、ユーザーにフィールドを見せても、更新はさせたくありません。

表示専用フィールドのもう1つの用途として、動的なプロンプトまたはステータス行を作成することができます。たとえば、定義時にプロンプトの正確な語句が決まっていない場合もあります。この場合、プロンプトを表示専用入力フィールドにしてから、実行時に正しいテキストを与えることことができます。

動的プロンプトの作成方法を説明するために、`generic-prompt` というデータブロック内の項目に関連付けられた表示専用入力フィールドがあるとします。プログラムでは、次のような文を使って項目に書き込みます。

```
move "Dynamic Prompt" to generic-prompt
```

`generic-prompt` と関連付けられた入力フィールドが再生されると、新しいプロンプトが表示されます。

表示専用は、単一行入力フィールドの属性です。入力フィールドを表示専用に設定する方法については、「*Dialog System の概要*」の章を参照してください。

5.1.1.2.4 自動スワイプ

自動スワイプは、ユーザーがフィールドをタブ移動したときに、そのフィールドと関連付けられているすべてのデータが選択されるような入力フィールドの属性です。ユーザーがフィールドに何らかのデータを入力すると、新しいデータが入力されるまえにそのフィールド内はクリアされます。

ユーザーがデフォルト値を使用しないことがわかっている場合、自動スワイプを使ってそのデフォルト値をクリアすることができます。たとえば、入力フィールドを使ってファイル名を入力する場合、デフォルトを "filename.ext" のような説明文字列にするとよいかもしれません。これは、ファイル名として適切なフォーマットをユーザーに示すことになります。しかし、このデフォルトのファイル名を使用することは、まずありません。

ユーザーがフィールドにデータを入力すると、既存のテキストはクリアされ、フィールドの最初の位置からデータ入力が始まります。

SET-AUTOSWIPE-STATE ファンクションを使うと、入力フィールドの自動スワイプ状態を変更することができます。前の例では、SET-AUTOSWIPE-STATE をオフにしておくと、ユーザーがこのフィールドにタブ移動したとき、データを入力するまえにフィールドはクリアされません。詳しくは、ヘルプの*ダイアログステートメントの項*を参照して下さい。

5.1.1.3 複数行入力フィールド

複数行入力フィールド（MLE）では、単一行入力フィールドと同様にデータ項目の内容を表示および編集することができます。しかし、このコントロールが最も便利なのは、製品の長い説明文など、大量の情報を収集する場合です。

MLE は、水平と垂直のスクロールバーを持ったボックスとして表示されます。これらのスクロールバーによって、MLE 内のデータの表示を調整することができます。スクロールバーに関連付けられたダイアログはありません。つまり MLE に関するすべてのスクロールバーの動作は、Dialog System によって制御されます。

2つの MLE を図 5-4 に示します。左側はワードラップオプションが設定されており、右側は設定されていません。

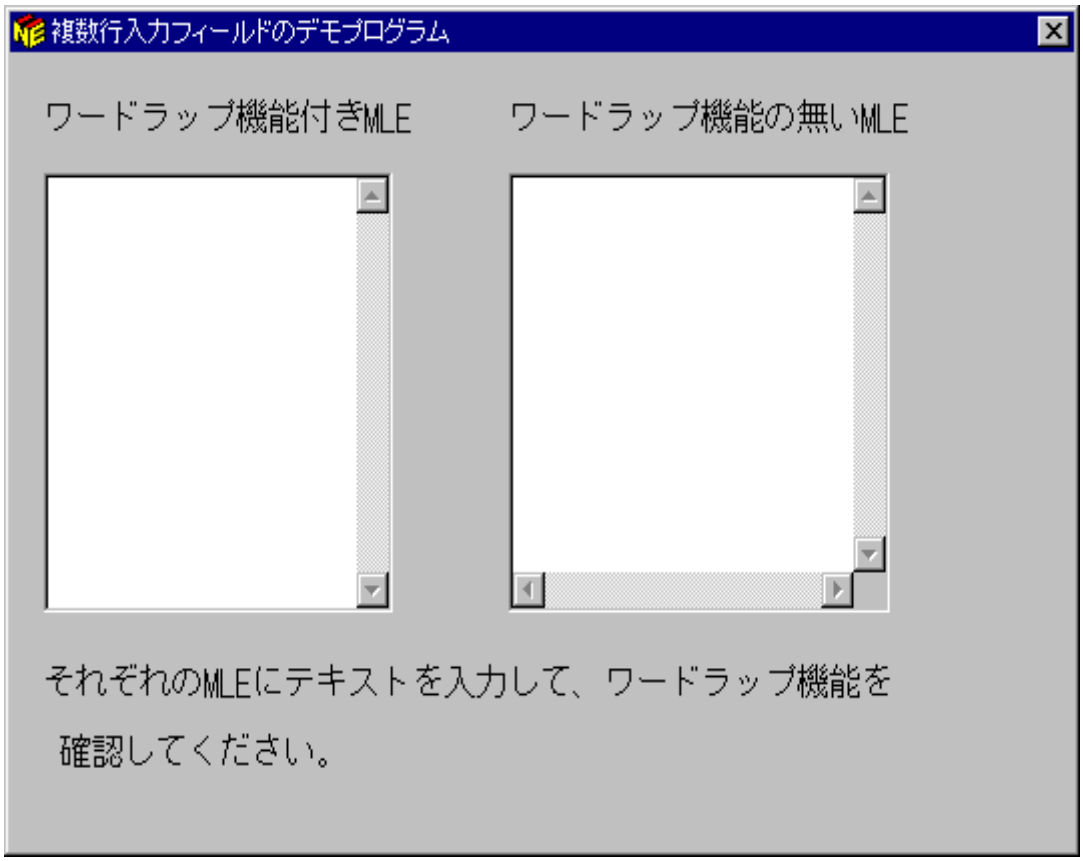


図 5-4 複数行入力フィールド

MLE に関連付けるデータ項目には、英数字、DBCS 文字、改行（ASCII の 10 進 10、16 進 0A）を入れることができます。改行は、一般的な解釈どおりに後続の文字を MLE の表示領域の次の行に配置します。スクロールバーを使うと、右または下の境界を超えた文字を見ることができます。

MLE をワードラップに設定すると、テキストは右の境界を超えることなく、次の行に折り返されます。右端に入り切らない単語は分割されます。

Objects サンプルプログラムには、ワードラップオプションを設定した場合と、設定しない場合の MLE の違いを示す例があります。

5.1.1.4 MLE の編集

MLE にテキストをいれたり、MLE 内のテキストを編集するには、次の手順に従ってください。

- アプリケーションプログラムの関連するデータ項目にテキストを移動します。
- ダイアログを使ってテキストをデータ項目に移動します。この場合、改行は挿入できません。
- クリップボードから MLE にテキストを直接入力します。
- ユーザーが、MLE の中で情報を直接編集できるようにします。

5.1.1.5 MLE の再生

実行時に MLE のデータ項目を変更した場合、変更結果が見えるようにするには、MLE を再生する必要があります。MLE は、次のような場合に再生されます。

- その親ウィンドウが実行時に作成された場合
- 自身またはその親ウィンドウが、ダイアログファンクション REFRESH-OBJECT によって明示的に再生された場合

ユーザーが MLE を変更すると、その結果は、何もしなくても直ちにデータ項目に反映されます。

5.1.2 プッシュボタン

プッシュボタンは、ユーザーが操作を選択するのに使用します。プッシュボタン内のテキストによって示される操作が直ちに実行されます。プッシュボタンを持つウィンドウの例を図 5-5 に示します。

図 5-5 標準的なプッシュボタン

5.1.2.1 プッシュボタンへのビットマップの割り当て

プッシュボタンを定義する場合、プッシュボタンの属性ウィンドウを使って、プッシュボタンにテキストの代わりにビットマップのセットを付けることができます。プッシュボタンが持てる3つの状態（通常の状態、使用禁止の状態、押下状態）について、ビットマップを定義することができます。

通常状態のビットマップは必ず定義する必要があります。押下状態または使用禁止状態のビットマップを定義しない場合、通常状態のビットマップが使用されます。

しかし、実行時にビットマップを設定または変更したい場合もあります。詳しくは、「*ビットマップを使用したマウスポインタの変更*」の章を参照してください。

5.1.3 ラジオボタン

プッシュボタンとは異なり、ラジオボタンは、相互に排他的な項目（1つしか選択できない）のセットをユーザーに

示すためにグループ化することができます。ラジオボタンの使用例を図 5-6 に示します。この例では、ユーザーはいずれか1つの方法を選択しなければなりません。

図 5-6 標準的なラジオボタン

ラジオボタンをグループ化すると、一度にグループ内のいずれか1つのボタンだけが選択できるという動作特性になります。ユーザーが新しいボタンを選択すると、まえに選択されたボタンは選択解除されます。コントロールグループの定義方法については、この章の「*コントロールのグループ化*」の節を参照してください。

ラジオボタンに関連付けられるダイアログは、プッシュボタンのダイアログと同じです。たとえば、ラジオボタン VISA は、次のようなダイアログを持つことができます。

```
BUTTON-SELECTED
    BRANCH-TO-PROCEDURE BILL-TO-VISA
```

BILL-TO-VISA 手続きには、必要な請求用のファンクションがあります。

注：ラジオボタンは、コントロールグループ内の他のラジオボタンを選択することによってのみ選択解除することができます。

ラジオボタンの定義については、「*オブジェクトと属性*」の章を参照してください。

5.1.4 チェックボックス

チェックボックスは、ユーザーが相互に排他的ではない項目を選択できることを除き、ラジオボタンとよく似ています。たとえば、図 5-7 に示すように、「当てはまるものはすべて選ぶ」というような状況では、チェックボックスを使用することができます。

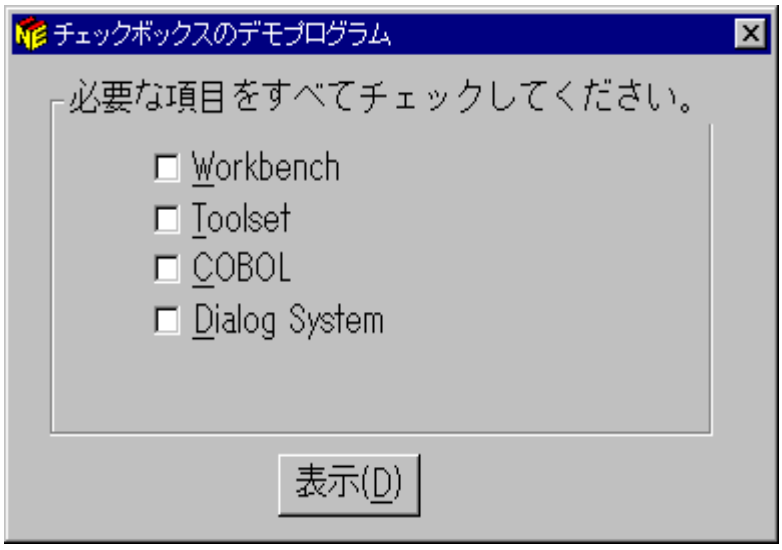


図 5-7 チェックボックスのグループ

チェックボックスには、数値データ項目を付属できます。チェックボックスがセットされている場合、そのチェックボックスに関連付けられたデータ項目は1に設定されます。チェックボックスがセットされていない場合、データ項目は0に設定されます。

詳しくは、「*サンプルプログラム*」の章を参照してください。

5.1.5 リストボックス

リストボックスを使用すると、ユーザーが1つまたは複数を選択できる項目のリストを表示します。リストボックスは、次のものからできています。

- 項目のリストが入ったウィンドウ
- 表示されていない部分の情報を見るのに使用する最低1つのスクロールバー

Dialog System には、3種類のリストボックスがあります。

- 単一選択リストボックスでは、リストから1つの項目だけを選択することができます。
- 複数選択リストボックスでは、任意数（リストボックスの制限値まで）の項目を選択できます。
- 範囲選択リストボックスでは、任意数（リストボックスの制限値まで）の隣接する項目を選択できます。

[オブジェクト] サンプルアプリケーションには、単一選択リストボックスと複数選択リストボックスの違いを示す例があります。

リストボックスの例を図 5-8 に示します。

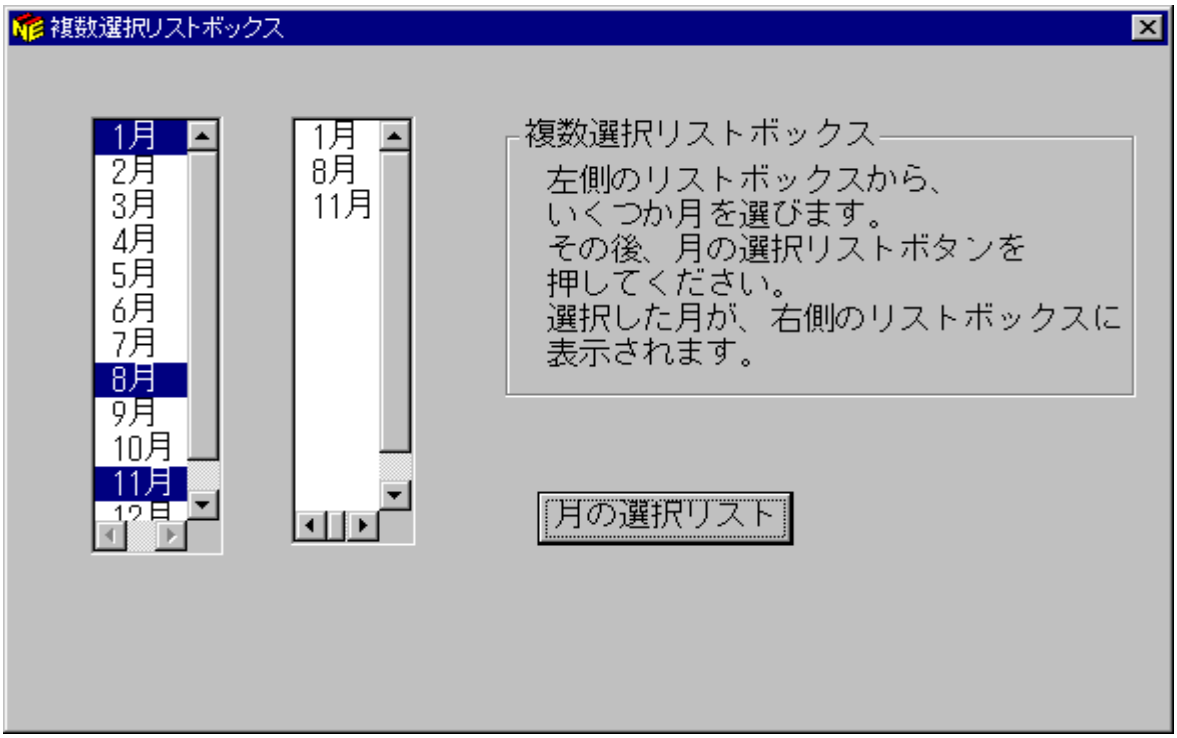


図 5-8 リストボックスの例

リストボックスの定義については、「*オブジェクトと属性*」の章を参照してください。

5.1.5.1 リストボックスへの項目の追加

リストボックスに項目を挿入するには、次の方法があります。

- プログラムから渡されたグループ項目を使う。

  リストボックスの各行がグループのオカレンス（要素）を示すように、リストボックスをグループ項目と関連付けることができます。グループ内の選択されたリストボックスが、各行のフィールドを占有します。

  これは、最も一般的なリストボックスの使用方法です。プログラムの例については、「*サンプルプログラム*」の章を参照してください。

- 定義機能を使った項目の追加

  定義時にリストボックスの属性の一部として項目を入力することができます。この項目は、実行時にリストボックスが作成されると、リストに挿入されます。したがって、この項目は他の挿入されたリスト項目とまったく同じで、変更または削除したり、項目間に他の項目を挿入することができます。

  これは、リストに常に必要な少数の項目が入っている場合、項目を追加するのに便利な方法です。この方法によるリストボックスへの項目の追加については、「*オブジェクトと属性*」の章を参照してください。

- 実行時にダイアログでINSERT-LIST-ITEMファンクションを使う。

  これは、リストの項目が少なく、データブロックをできるだけ小さくしたいときに、リストボックスに項目を設定する効果的な方法です。小さいリストボックスに項目を設定する時間は、それほど必要ありませんが、実行するダイアログ文を追加するのは、多少時間がかかります。

- 区切られた文字をプログラムから渡す。

  この方法には、2つの利点があります。

  - スクリーンセットのダイアログ文の数を減らします。
  - Dialog System以外のリストを保守することができます。たとえば、ASCIIファイルにデータを保存して、そのデータをエディタで保守し、実行時にプログラムで読み込むことができます。このように、プログラムやスクリーンセットに影響を与えることなく、データの比較的小さい変更を行うことができます。

  これは、プログラムとDialog Systemで渡されるデータブロックのサイズを増やしますが、データ項目をリストに１個づつ追加するよりも速くなります。

  例については、「*サンプルプログラム*」の章を参照してください。

5.1.6 選択ボックス

選択ボックスは、スクロール可能なリストボックスと別のコントロール（入力フィールド、ボタン、またはその両方）を組み合わせたコントロールです。選択ボックスは、コンビネーションボックスまたはコンボボックスと呼ばれることもあります。リストボックスには、ユーザーが選択できるスクロール可能な項目があり、選択された項目は入力フィールドに配置されます。

利用可能な3種類の選択ボックスを図 5-9 に示します。

図 5-9 標準的な選択ボックス

選択ボックスのタイプ:

簡易（または固定）選択ボックス	入力フィールドとリストボックスからできています。ユーザーは、入力フィールドにデータを入力するか、リストから項目を選択することができます。選択された項目は入力フィールドに配置されます。
ドロップダウン選択ボックス	右端の矢印ボタンと隠れたリストボックスを持つ入力フィールドからできています。ユーザーが矢印ボタンを押すと、リストボックスが表示されます。ユーザーが項目を選択するか、他の場所をクリックすると、リストボックスは再び隠れます。
ドロップダウンリスト	右端の矢印ボタンと隠れたリストボックスを持つ表示専用フィールドからできています。ユーザーは、表示専用フィールドにはデータを入力できません。選択は、リストから行わなければなりません。これは、現在の選択だけを表示するために縮小されたリストボックスと考えることができます。

3種類の選択ボックスを、まったく同じ方法で定義することができます。その場合は、選択ボックス属性ダイアログボックスで、指定したいタイプを属性として選択します。

選択ボックスをウィンドウまたはダイアログボックスに個々に追加するには、次のように行います。

1. ウィンドウまたはダイアログボックスを選択します
2. [オブジェクト]ツールバーの[選択ボックス]をクリックする
   - または -
   [オブジェクト]メニューの[選択ボックス]をクリックします。

ウィンドウまたはダイアログボックスで、ドロップダウンリストが他のコントロールを覆い隠すような場所に、選択ボックスを定義するときは､隠れたコントロールの前に選択ボックスがフォーカスを受け取るようにします｡選択ボックスを最初に定義するか、[編集]メニューの[コントロール]を使ってコントロールの順序を変えるかのどちらかを行います。そうすることによって、実行時にすべてのコントロールが適切に表示されるようになります。

5.1.6.1 入力フィールド

選択ボックスの中の入力フィールドは、マスタフィールドと呼ばれるデータ項目にリンクしています。通常は、入力フィールドにテキストを置いて、デフォルトの選択項目を提供します。

5.1.6.1.1 入力フィールドのリフレッシュ(再表示)

実行時に、入力フィールドのマスタフィールドの内容を表示するには、選択ボックスをリフレッシュする必要があります。選択ボックスは、次の場合にリフレッシュされます。

- 親ウィンドウが実行時に作成されたとき
- REFRESH-OBJECTダイアログファンクションによって、親ウィンドウが明示的にリフレッシュされたとき

SHOW-WINDOWファンクションを使って選択ボックスを表示すると、リフレッシュされません。

5.1.6.1.2 入力フィールドの変更

ユーザーが入力したり、リストボックスの項目を選択して、入力フィールドを変更すると、ユーザー側にはなんのアクションもありませんが、変更内容はただちにマスタフィールドに設定されます。

選択ボックスの項目を選択すると、ITEM-SELECTEDイベントが発生します。

次の方法で、テキストの項目をリストボックスに設定することができます。

- 選択ボックスの属性の一部として、定義時に、項目を入力します。

  これらの項目は、選択ボックスが作成されたときに、リストに挿入されます。そのあとの処理は、他のリスト項目と同様に行われます。たとえば、削除したり、他の項目を挿入することができます。

- INSERT-LIST-ITEMまたはINSERT-MANY-LIST-ITEMSおよびDELETE-LIST-ITEMの各ダイアログファンクションを使って、実行時に項目を挿入したり、削除します。

この方法(とほかの方法)でリストボックスに項目を追加することについては、「*サンプルプログラム*」の章の「*ダイアログを使った項目の追加*」を参照してください。

5.1.7 スクロールバー

リストボックスと MLE では、リストボックスに表示されている情報の相対位置と量がスクロールバーによって示されます。スクロールバーを使うと、表示される情報を調整することができます。また、Dialog System では、スクロールバーを独立したコントロールとして作成することもできます。

たとえば、入力フィールドでスクロールバーを使うと、ユーザーは「スクロール」によって値を設定することができます。サンプルアプリケーション Objects は、このような使い方を示しています。

水平スクロールバーの例を図 5-10 に示します。

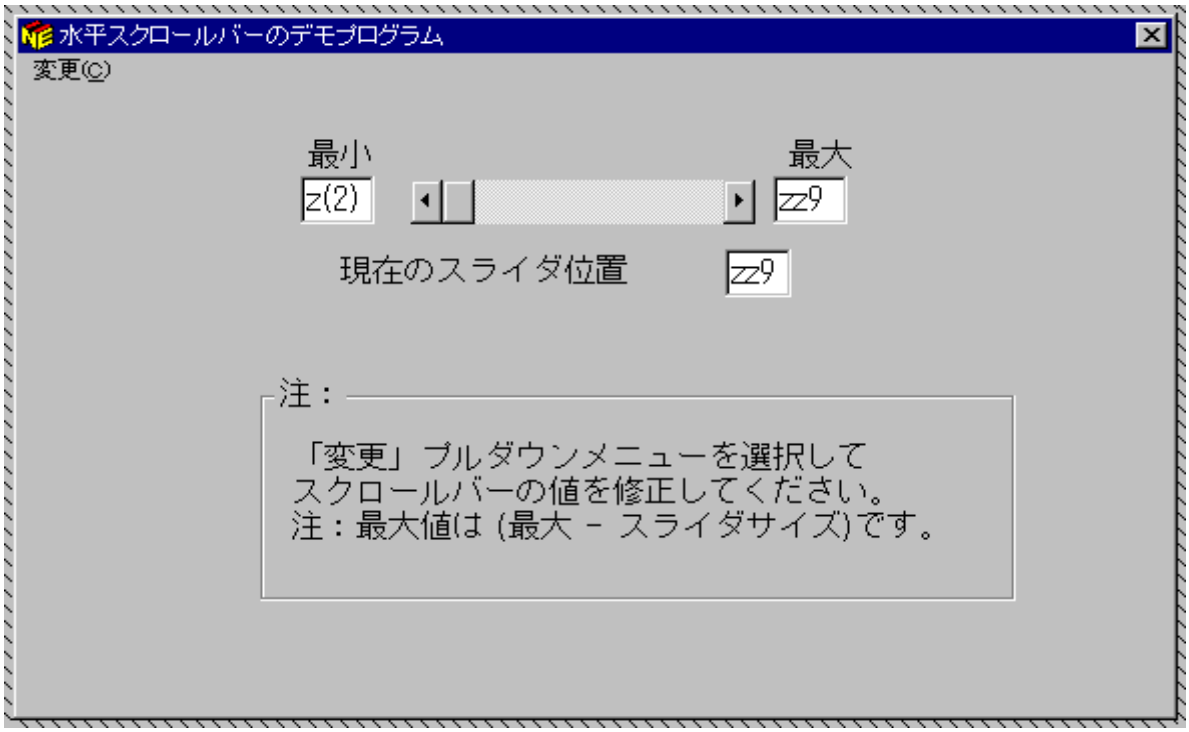


図 5-10 水平スクロールバー

スクロールバー属性ダイアログボックスによって、スライダの範囲、スライダの位置、スライダのサイズなどのスクロールバー属性にデフォルトを割り当てることができます。ダイアログを使うと、これらの属性を変更することができます。

「*チュートリアル - サンプルスクリーンセットの作成*」および「*チュートリアル - サンプルスクリーンセットの使い方*」の各章では、コードが記述されたサンプルを参照することができます。スクロールバーの定義については、ヘルプの「*オブジェクトと属性*」のトピックを参照してください。

5.1.8 グループボックス

グループボックスは、コントロールを囲むグラフィックの枠です。これは、単にコントロールを見やすくするためだけに使われます。グループボックスによって、ユーザーはいくつかのコントロールがグループになっていることがわかります。グループボックスの例を図 5-11 に示します。

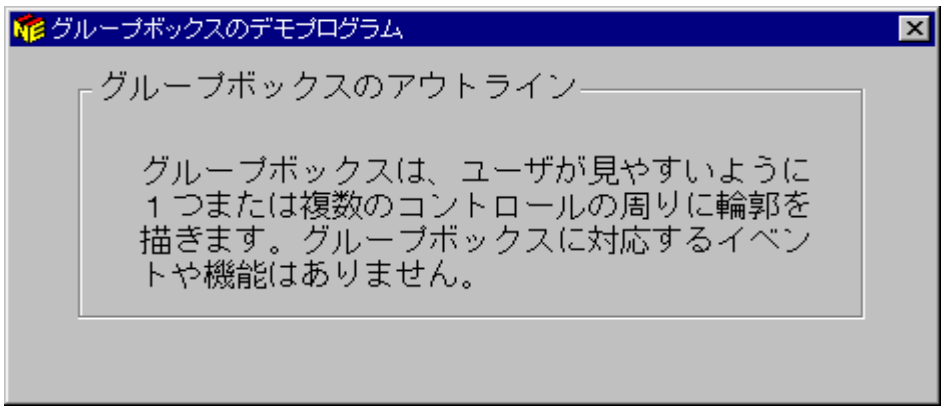


図 5-11 グループボックスの例

テキストボックスと同様に、グループボックスにイベントやファンクションが関連付けられることはありません。

グループボックスの定義については、「*オブジェクトと属性*」を参照してください。

5.1.9 タブコントロール

タブコントロールは、直感的で使いやすい方法で、関連する情報を編成することができます。図 5-12 は、タブコントロールの例を示しています。

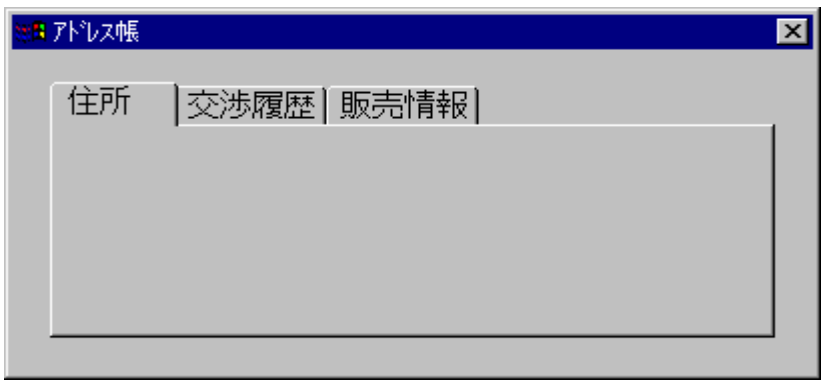


図 5-12 標準的なタブコントロール

タブコントロールは、個別のページから構成されます。これらのページを組み合わせたものが一つのコントロール単位であり、各ページはコントロールの一方の辺(通常は最上部)に沿ったタブとして表示されます。タブを選択すると、関連するページが表示されます。

タブには、テキストまたはビットマップを取り込むことができます。すべてのタブが一行に収まらない場合は、スクロール矢印でタブを移動するか、タブがすべて見えるように複数行にわたってタブをラップするか、どちらかの方法で表示することができます。

タブコントロールの可視領域は、常に一番上のページです。ページには、他のタブコントロールまたはウィンドウを除く、あらゆるDialog Systemコントロールオブジェクトを取り込むことができます。

「*サンプルプログラム*」の章では、コードが記述されたサンプルを参照することができます。

5.1.10 OLE2 **コントロール**

OLE2 コントロールは、内側にスプレッドシートやビットマップなどの外部のデータやオブジェクトを表示することができるウィンドウです。

例えば、絵をデザインした編集するのにペイントブラシアプリケーションを使用して、この絵をOLE2ウィンドウへロードすることができます。

5.1.11 **ユーザーコントロール**

ユーザーコントロールは、Dialog System定義ソフトウェアでは作成できないオブジェクトのコンテナとして使用されます。

ユーザーコントロールのイベントとファンクションは、そのオブジェクトに関連するコントロールプログラムに実装されます。一般的には、コントロールプログラムはクラスライブラリを使って、オブジェクトを作成および保守するものです。

ユーザーコントロールは、定義時にコントロールプログラムを生成することによって、作成したり操作します。コントロールプログラムを使う利点は、オブジェクトが完全にユーザー定義可能であるという一方で、Dialog Systemやランタイムソフトウェアに統合されるという利点も持ち合わせていることです。

ユーザーコントロールオブジェクトと定義ソフトウェアを統合することによって、次の利点が提供されます。

- 実行時のオブジェクトの自動的な作成と表示。
- オブジェクトのタイプによっては、子ウィンドウおよびガジェットのサイズ変更と移動の機能だけでなく、さまざまな解像度の画面をサポートします。詳しくは、「*上級テクニック*」の章とヘルプの「*ダイナミックなウィンドウのサイズ設定*」のトピックを参照してください。
- オブジェクトの位置とサイズは、他のDialog Systemオブジェクトと同じ方法で、簡単に定義したり修正することができます。
- オブジェクトの順番は、他のDialog Systemオブジェクトと同じ方法で、再指定することができます。そのため、実行時にタブ順を定義することができます。ユーザーは、既存のDialog Systemオブジェクトとユーザーコントロールオブジェクトの間をタブで移動する順序を定義することができます。

- あらゆる標準的な整列機能は、ユーザーコントロールに使うことができ、他のDialog Systemオブジェクトと調整することができます。オブジェクトの整列については、この章の「コントロールの整列」を参照してください。

ユーザーコントロールオブジェクトの作成と操作については、「*独自のコントロールのプログラミング*」の章を参照してください。

5.1.12 ActiveXコントロール

ActiveXコントロールは、サードパーティベンダーによって提供され、Dialog Systemアプリケーションに統合することができます。前述のユーザーコントロールの利点は、スクリーンセットでActiveXコントロールを使うときにも適用されます。ActiveXコントロールを使った他の利点には、次のものがあります。

- ActiveXコントロールを定義時に視覚的に描くことができます。
- スクリーンセットの初期属性を保存するため、コントロールの属性ページダイアログを表示することができます。
- 属性を介して実行時のコントロールの外観と動作を操作するコードを記述する方法、メソッドとイベント、Dialog Systemプログラミングアシスタントの使い方のヘルプを得ることができます。詳しくは、「*独自のコントロールのプログラミング*」の章を参照してください。

アプリケーションで使いたいコントロールは、開発用のライセンスを購入する必要があります。また、コントロールの開発と配布については、ベンダーの使用許諾契約書によって拘束されます。

5.2 コントロールのグループ化

入力フィールド、ラジオボタン、プッシュボタンなどのコントロールが実行時にウィンドウまたはダイアログボックスに表示されている場合、ユーザーは、Tab キーを使ってそれらのコントロール間でキーボードの入力フォーカスを移動することができます。

いくつかのコントロールが論理グループ（ラジオボタンなど）を構成していることがあります。この場合、ユーザーはすべてのコントロールに注目するか、すべてを無視するかのどちらかです。したがって、Tab キーを押すと、フォーカスがあるグループに移動し、さらに押すと、次のコントロールまたはグループに移動すると便利です。グループ内のコントロール間を移動するには、カーソルキーを使います。（これもデフォルト動作なので、変更することもできます。）

このような動作を実現するには、複数のコントロールをコントロールグループにまとめます。

複数のラジオボタンのいずれか1つだけが選択できるようにするには、それらをコントロールグループにまとめなければなりません。いずれかひとつのラジオボタンを選択すると、他のラジオボタンの選択が解除されます。

通常、ユーザーが最もよく使用するコントロールまたはコントロールグループからフォーカスを開始し、Tab キー

を押すたびに操作に適した順序でフォーカスが移動するようにします。Tab キーを押したときに、フォーカスが最初のコントロールに移動し、カーソルキーを使って操作に適した順序でフォーカスが移動するように、グループ内のコントロールの順序も指定する必要があります。

注：コントロールグループの最初の項目が選択解除されている場合、Tab キーによってそのグループに移動することはできません。

それぞれのウィンドウまたはダイアログボックスの中でコントロールまたはグループの順序を指定することができます。具体的な手順については、「*Dialog System の概要*」を参照してください。

5.3 コントロールの整列

Dialog System には、グラフィカル環境内でオブジェクトを移動またはサイズ調整するときに、オブジェクトの位置を簡単に揃えるための整列機能があります。

ほとんどの整列機能は、整列バーから利用できます。このバーは、「オプション」メニューの「含める」から「整列バー」を選択することによって、オンまたはオフに切り替えることができます。整列バーについては、「*Dialog System の概要*」を参照してください。

整列オプションを選択した場合、コントロールを定義するときにその位置を正確に設定する必要はありません。必要なコントロールを適当に設定し、「整列バー」ボタンをクリックすると、間隔やサイズを調整することができます。

ほとんどの整列操作には、いくつかの整列機能の組み合わせが必要です。たとえば、ウィンドウの下部に4つのプッシュボタンを設定する場合、次のような手順を実行します。

1. プッシュボタンを、求める大体の位置に配置します。
2. [編集]メニューの[選択]を選択するか、領域選択のツールバーを使って、オブジェクトを囲みます。これによって、すべてのオブジェクトをグループとして取り扱うことができます。
3. 下に揃えるボタンをクリックします。
4. 幅をサイズ変更するボタンをクリックします。
5. 横方向を均等化するボタンをクリックします。

整列の間隔が適切でない場合は、代わりの方法でボタンを整列します。

1. [水平に並べる]を選択します。

これには間隔をあける効果はありません。

2. 次を行うことによって、グループのサイズを設定します。

   - [編集]メニューの[サイズ]を選択します。
     -または-
   - 選択したコントロール上で、サイズを設定する操作を行います。

   これはコントロールの間隔を広げます。

4つのプッシュボタンが誤った順序に並んでいる場合、シャッフルするボタンを使って、順序を変えることができます。たとえば、[OK]、[挿入]、[キャンセル]、[ヘルプ]というプッシュボタンがあり、これらを[ヘルプ]、[キャンセル]、[挿入]、[OK]という順序にしたいとします。

ボタンを再配置するには、まず配置したい順序でボタンを選択します。次のように行います。

1. [ヘルプ]プッシュボタンをクリックします。
2. [Ctrl]キーを押しながら、[キャンセル]プッシュボタンをクリックします。

   この手順を[挿入]および[OK]プッシュボタンについて繰り返します。

3. 整列ツールバーの[オブジェクトの再配置]をクリックします。

   オブジェクトは、[ヘルプ]、[キャンセル]、[挿入]そして[OK]の順になります。

各ボタンは、選択した順番にグループの中でお互いに入れ替わります。デフォルトの交換順序は、左から右へとなります。つまり、この例では、[OK]ボタンが最初に入れ替わり、次に[挿入]が、3番目に[キャンセル]が、4番目に[ヘルプ]が交換されます。

どの整列機能でも、[編集]メニューの[元に戻す]を選択すると最後の操作をやり直すことができます。

ウィンドウの右端に上から下に向かって4つのプッシュボタンを配置するための手順を次に示します。

1. まえと同様に対象となるコントロールをグループとして選択します。
2. 選択したグループをウィンドウの上部に異動します。
3. 縦方向にタイル状に並べるボタンをクリックします。
4. 右に揃えるボタンをクリックします。
5. [編集]メニューの[サイズ]を選択して、コントロールの間隔を広げます。

   コントロールがウィンドウ内の適切な位置に配置されたら、必要に応じてシャッフルするボタンを使って順序を変更します。

5.4 サンプルプログラム

インストールディレクトリDialogsystem¥demo¥objectsに提供されている、Objectsサンプルプログラムを実行すると、Dialog Systemコントロールオブジェクトの例を見ることができます。

5.5 ビットマップを使う

この章では、Dialog System インタフェースにおけるビットマップグラフィックス、アイコン、マウスポインタの使い方について説明します。Dialog System では、ユーザーインタフェースの中で次のオブジェクトについて使用したいビットマップを選択することができます。

ビットマップオブジェクト	ユーザーが選択できるオブジェクトを装飾します。
アイコン	最小化されたウィンドウまたはダイアログボックスを表す小さなビットマップ。「*ウィンドウオブジェクト*」の章の「*アイコンの設定*」を参照してください。
マウスポインタ	マウスポインタを示すためにビットマップを使用します。マウスまたはシステムの状態を表します。
プッシュボタン	ボタンを装飾したり、ボタンの状態を表します。

Dialog System には、そこで使われるオブジェクトのビットマップと、使用する環境の標準ビットマップが用意されています。ユーザーがスクリーンセットで使用する独自のビットマップ、アイコン、マウスポインタを追加することもできます。詳細については、「*ビットマップ、アイコン、マウスポインタ*」の章を参照してください。

5.5.1 ビットマップの定義

ビットマップは、ウィンドウまたはダイアログボックス内のどこにでも配置できます。ビットマップの主な目的は、ボタンに注目を引きつけたり、ウィンドウまたはダイアログボックス内にロゴなどの項目を配置するなどの装飾用です。他のオブジェクトと似ていますが、ダイアログを付けることはできません。しかし、ビットマップをクリックすると、BITMAP-EVENT が起こり、ビットマップのオブジェクトハンドルが $EVENT-DATA レジスタに書き込まれます。ビットマップにMOUSE-OVERイベントに反応するダイアログを定義することもできます。

属性ダイアログボックスから、マスタフィールドをビットマップに割り当てることができます。これによって、表示されたビットマップのイメージを、実行時に動的に変更することができます。詳しくは、ヘルプの「*コントロールの使い方*」および「*ビットマップ、アイコン、マウスポインタ*」の各トピックを参照してください。

ウィンドウまたはダイアログボックスにビットマップを置くには、次のように行います。

1. メインウィンドウで[オブジェクト]メニューの[ビットマップ]を選択します。
   -または-

[オブジェクト]ツールバーでビットマップアイコンをクリックします。

図 5-13 に示す、ビットマップの選択ダイアログボックスが表示されます。

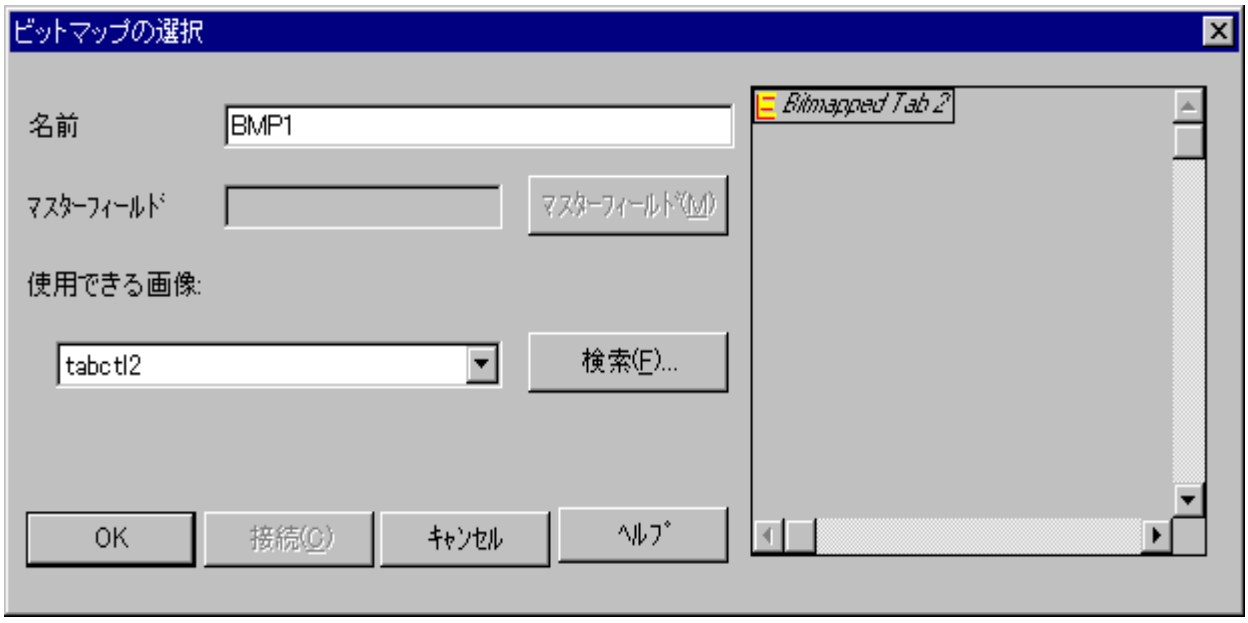


図 5-13 ビットマップの選択ダイアログボックス

2. [名前]にビットマップ名を指定します。
   -または-
   [使用できる画像]から名前を選択します。

3. 選択したあとで、[OK]をクリックします。

   ビットマップの選択ダイアログボックスが閉じ、ウィンドウまたはダイアログボックスにビットマップの外形が表示されます。

4. マウスを使ってビットマップを位置づけ、クリックします。

   ビットマップが表示されます。

[編集]メニューの[コントロール]を使うと、ビットマップのあとで他のコントロールをリストに置くことができます。[コントロール]は、ウィンドウが非表示または再表示されるまで、表示には影響を与えません。

5.6 ビットマップ化されたプッシュボタン

Dialog System には、プッシュボタンのデフォルト表示が用意されています。プッシュボタンを変更するには、ビットマップ化されたプッシュボタンを使用することができます。

たとえば、ボタンをグラフィックシンボルに運びたいときや、プッシュボタンの外観をその状況によって変えたいときがあります。

ビットマップをボタンと関連付けるには、次のようにします。

1. ボタンをダブルクリックします。

   -または-

   ボタンを選択してから、メインウィンドウで[編集]メニューの[属性]を選択します。

   図 5-14 に示すような、プッシュボタン属性ダイアログボックスが表示されます。

   図 5-14 プッシュボタン属性ダイアログボックス

2. [オプション]タブをクリックします。

   図 5-15 のように、利用可能なオプションが表示されます。

図 5-15 プッシュボタン属性ダイアログボックス - オプション

3. [ビットマップ]を選択します。

   このオプションにチェックマークが付けられ、ボタン状態ビットマップオプションが利用できるようになります。3つの選択項目があります。

   - 通常
   - 押下
   - 使用不可

4. ボタン状態ビットマップドロップダウンリストから、必要なビットマップファイルを選択します。

   このビットマップファイル名は、選択されたビットマップファイルとして表示されます。

5. [OK]をクリックすると、メインウィンドウに戻ります。

ウィンドウのツールバーを作成するとき、ユーザーは独自バージョンのツールバーコントロールプログラムをカスタマイズして、ネイティブなWin32ツールバーを実装したいと考えるかもしれません。

第6章　ダイアログの使い方

これまでの章では、スクリーンセットのデータ記述と視覚的概念を作成する方法について述べました。ここでは、これらすべての実行に必要なダイアログについて説明します。

- ダイアログとそのレベル
- イベント、ファンクション、手続き
- 特殊レジスタ
- ダイアログのイベントとファンクションの重要なサンプル

6.1 ダイアログとは何か？

ダイアログは、どんなイベントが起こり、それに対してどのように応答すべきかをシステムに指示します。簡単なイベントと応答の例として、「ユーザーが[次のウィンドウ]というボタンを押したら、新しいウィンドウにフォーカスを設定する」などがあります。 これを実行するコードは、アプリケーションプログラムがアクセスする、ダイアログを使って指定します。

ダイアログは、次のものから構成されます。

- 応答する必要があり、起こり得るイベントの名前
- 各イベントについて、それが起こったときに実行すべき命令（ファンクションと呼ばれる）
- オプションとして、呼出し可能なファンクションのリストを持つ手続き（サブルーチンと呼ばれる）
- オプションとして、ダイアログを読みやすくするための注釈

イベントまたは手続きの名前によって、前のイベントのダイアログが終了します。たとえば、ウィンドウに付属する、以下のダイアログ部分について考えてみます。

```
CLOSED-WINDOW
    SET-EXIT-FLAG
    RETC
F1
    MOVE 1 ACTION
    RETC
    SET-FOCUS EMPLOYEE-WIN
  DELETE-EMPLOYEE-PROC
```

```
    MOVE 2 ACTION
    RETC
    SET-FOCUS EMPLOYEE-WIN
```

ファンクション SET-EXIT-FLAG と RETC は、CLOSED-WINDOW イベントに付属します。つまり RETC は、CLOSED-WINDOW イベントに関連付けられた最後のファンクションです。ファンクション MOVE 1 ACTION、RETC、SET-FOCUS EMPLOYEE-WIN は、F1 イベントに付属し、以下も同様です。

6.1.1 注釈

COBOLプログラムの場合と同様に、ダイアログ文でも、コードを読みやすくするために、注釈を使用することができます。注釈を使うと、以下のことができます。

- 作成したダイアログで、混乱する可能性のある文を、わかりやすく説明します。
- ともすれば(他の状況では)意味のあるダイアログ文の行を、実行しないようにします。
- 読みやすくするために、イベントと手続きを分離します。

注釈行は、行の先頭がアスタリスク(*)ではじまります。

次のサンプルダイアログは、注釈を使った例を示しています。

```
*********************
 SCREENSET-INITIALIZED
*
* Save the handles of each of the bitmaps
*
     MOVE-OBJECT-HANDLE DENMARK DENMARK-HANDLE
     MOVE-OBJECT-HANDLE FRANCE FRANCE-HANDLE
*     MOVE-OBJECT-HANDLE US US-HANDLE
*     MOVE-OBJECT-HANDLE UK UK-HANDLE
     SET-FOCUS GERMAN-DLG
*********************
```

6.1.2 ダイアログのレベル

ダイアログは、イベントが発生するウィンドウやコントロール、またはインタフェース全般に対して設定します。ウィンドウ、コントロール、インタフェース全般は、オブジェクトの3つのクラスをあらわすものですが、それぞれに設定されるダイアログの間には、基本的な違いはありません。各オブジェクトのダイアログは、ダイアログテーブルに

格納されます。オブジェクトのダイアログのクラスには、次のものがあります。

- コントロールダイアログ
- ウィンドウダイアログ
- グローバルダイアログ

6.1.2.1 コントロールダイアログ

このダイアログは、テキスト、ユーザーコントロール、AcriveXコントロールを除く、あらゆるコントロールに設定されます。

たとえば、次のダイアログはプッシュボタンに設定されたものであり、ウィンドウのタイトルを変更します。

```
BUTTON-SELECTED
    MOVE "Customer Address Details" NEW-TITLE
    SET-OBJECT-LABEL CUSTOMER-ADDRESS-WIN NEW-TITLE
    SET-FOCUS CUSTOMER-ADDRESS-WIN
```

6.1.2.2 ウィンドウダイアログ

このダイアログは、あらゆるウィンドウ、ダイアログボックス、またはタブコントロールページに設定されます。

たとえば、次のダイアログはウィンドウに設定されたものであり、ユーザーによる「閉じる」オプションの選択を操作します。

```
CLOSED-WINDOW
     SET-EXIT-FLAG
     RETC
```

6.1.2.3 グローバルダイアログ

このダイアログはスクリーンセット全体に付属します。

たとえば、次のダイアログはリストボックスに対して初期設定を行います。

```
SCREENSET-INITIALIZED
    INSERT-LIST-ITEM MONTH-LB "Jan" 1
    INSERT-LIST-ITEM MONTH-LB "Feb" 2
    INSERT-LIST-ITEM MONTH-LB "Mar" 3
```

6.1.2.4 ダイアログ文を置く場所

特定のオブジェクトやウィンドウに関するダイアログを定義する場所は、ユーザーの要求によって変わります。次の場合があります。

- 手続きが、複数のコントロールやウィンドウオブジェクトによって呼び出される場合は、グローバルダイアログに置きます。
- 手続きが、あるオブジェクトに特定の場合は、直接そのオブジェクトやオブジェクトが配置されたウィンドウに設定することができます。

よく構造化されたダイアログは、効率よく実行を行い、保守とデバッグがより簡単になります。ダイアログを記述するときは、どのプログラミング言語にも適用されるコーディングの原則に準拠するようにしましょう。

6.1.3 ダイアログのタイプ

ダイアログには、次の3つのタイプがあります。

- イベント
- 手続き名
- ファンクション

6.1.3.1 イベント

スクリーンセットに渡されるイベントは、はっきりと定義する必要があります。イベントが定義されていないと、無視されることになります。ダイアログのイベントは、グローバルダイアログ、ウィンドウダイアログ、個々のコントロールダイアログで定義することができます。

Dialog Systemでは、各ダイアログテーブルのイベントの数を、255に制限しています。

イベントは、ユーザーインタフェースにおける何らかの変化を表します。たとえば、次のような変化があります。

- ユーザーがキーを押した。
- ウィンドウまたはコントロールがフォーカスを受け取った。
- 妥当性検査エラーが起こった。
- ユーザーがスクロールバーのスライダを動かした。

6.1.3.1.1 Dialog System がイベントダイアログを探す方法

イベントが発生すると、Dialog Systemはイベントを検索します。最初にコントロールダイアログを、次にそのコントロールを含むウィンドウのダイアログを、最後にグローバル(スクリーンセット)ダイアログを検索します。

注: 同じ規則が、手続きにも適用されます。コントロールのダイアログで手続名を呼び出したとき、そこで見つからない場合は、ウィンドウダイアログで手続きが検索され、次にグローバルダイアログで検索されます。

これら3つの場所のいずれにもイベントがリストされていない場合、コントロールレベル、ウィンドウレベル、そしてグローバルレベルから ANY-OTHER-EVENT を探します。

ANY-OTHER-EVENT が見つからない場合、操作は行われません。

Dialog Systemがイベントを見つけると、そのイベントに設定されたファンクションは、1ステップずつ処理されます。この処理は、制御が他の手続きに分岐しないかぎり、イベントにリストされたファンクションの終わりまで連続して行われます。処理は他のイベントによって区切られます。Dialog Systemは、イベントをキューに入れ、それを必要に応じて処理することによって、一度に一つのイベントを処理していきます。Dialog Systemは、各ダイアログテーブルのイベントの数を、255に制限しています。

Dialog Systemのイベントは、Panels V2モジュールによって管理されています。このモジュールは、イベントがいつ発生するのかを認識し、それにしたがって処理を行います。通常は、呼出しプログラムがPanels V2からイベントの情報を受け取ることはありません。しかし、これらのイベントを、dssysinf.cpyコピーファイルの中のds-event-blockを介して、呼出しプログラムに返すことができます。このファイルは、呼出しプログラムの作業場所節にコピーし、Dialog Systemの呼出しの中で指定する必要があります。ユーザーがこれを行いたいと考えるのは、スクリーンイベントがDialog Systemにはっきりと定義されていない場所です。

ヘルプの「*ダイアログ文:イベント*」トピックでは、イベントのすべての説明を参照することができます。

注： ウィンドウが二次ウィンドウの場合、一次ウィンドウダイアログは検索されません。

6.1.3.2 ファンクション

ファンクションは、Dialog System が何かを行うための命令です。ファンクションは、次のような場合に動作します。

- リストされているイベントが起こった場合
- リストされている手続きが実行された場合

各イベントまたは手続きの下に入力できるファンクションの数は、ファンクションまたはイベントのバッファのサイズ（2K）と、ダイアログテーブルの最大サイズ（64K）によって制限されます。

最大 2048 個のファンクションを入力できますが、実際の最大数はファンクションによって異なります。平均的な長さのファンクションでは、341 がより現実的な制限値です。これより多くのファンクションが必要な場合、特別なファンクションを含む手続きを呼び出すことができます。次に例を示します。

```
...
    function-340
    function-341
    EXECUTE-PROCEDURE FUNCTIONS-342-TO-350
    ...
  FUNCTIONS-342-TO-350
    function-342
    function-343
    function-344
    ...
```

function-340 から function-344 は、オンラインヘルプにリストされている Dialog System の各ファンクションです。

Dialog System には、豊富なファンクションがあります。これには、SET-FOCUS や INVOKE-MESSAGE-BOX などのようにスクリーンセット内を移動するものや、あるデータ項目から別のデータ項目にデータを移動するための MOVE のように、他のプログラミング言語の命令と似たファンクションもあります。

各ファンクションについての詳細は、*「ダイアログの定義：ファンクション」*の章を参照してください。

6.1.3.3 手続き

手続きは、その下にファンクションのリスト（すなわちサブルーチン）を持ち、任意の名前をつけることができます。ここでは、手続きをイベントと同様に考えることができます。つまり EXECUTE-PROCEDURE または BRANCH-TO-PROCEDURE ファンクションが実行されると、「イベント」が「起こり」ます。

たとえば、次のダイアログ部分は、手続きを使った「ループ」のコーディングを示しています。このループは、リストの処理に使用できます。

```
...
BUTTON-SELECTED
    MOVE 1 INDX
    BRANCH-TO-PROCEDURE CLEAR-LOOP
  CLEAR-LOOP
    MOVE "    " SELECTED-ITEM(INDX)
```

```
    INCREMENT INDX
    IF= INDX MAX-LOOP-SIZE EXIT-CLEAR-LOOP
    BRANCH-TO-PROCEDURE CLEAR-LOOP
  EXIT-CLEAR-LOOP
    ...
```

6.2 特殊レジスタ

Dialog System には、パラメータとして使用できるレジスタと変数がいくつかあります。

$REGISTER
: 汎用の内部レジスタ。配列へのインデックスとして、または数値を一時的に格納するためなどに使用します。

$NULL
: デフォルト値を提供したり、実際のパラメータが必要ないときに他のパラメータとの区切りを示すためのパラメータ。

$CONTROL
: 現在選択されているコントロール。コントロールの名前がわからないような汎用の手続きで使用できます。たとえば、次に示すように CLEAR-OBJECT ファンクションと共に使用します。

```
F3
   CLEAR-OBJECT $CONTROL
```

ユーザーが F3 ボタンを押すと、現在のオブジェクトがクリアされます。

$WINDOW
: 現在選択されているウィンドウ。たとえば、前の例と同じように CLEAR-OBJECT ファンクションと共に使用します。

```
F2
   CLEAR-OBJECT $WINDOW
```

$EVENT-DATA
: 何らかのイベントによって書き込まれるレジスタ。たとえば、VAL-ERROR イベントと共に使用します。妥当性検査エラーが検出された場合、VAL-ERROR イベントが引き起こされ、エラーが起こったフィールドの識別子が $EVENT-DATA に書き込まれます。これによって、ユーザーがデータを修正するために、エラーが起こったフィールドにフォーカスを設定することができます。このような $EVENT-DATA の使い方については、第9章*「テキスト、フォント、色」*を参照してください。

$EVENT-DATA にデータを書き込むイベントについては、オンラインヘルプの*「ダイアログの定義：イベント」*の章を参照してください。

$INSTANCE　　特定のノートブックページを参照するためのインスタンス番号。

6.3 重要なイベントとファンクション

Dialog System には、スクリーンセットの動作を制御するための豊富なイベントとファンクションがあります。しかし、最初に何をするのか、どのようにしてメニュー項目を選択するのか、どうやってプログラムに戻るのか、また、スクリーンセットに戻ると何が起こるのか、というようなことについて理解する必要があります。

以下の節では、いくつかの基本的なファンクションについて説明し、そのファンクションに適していると思われるダイアログのサンプルを紹介します。

6.3.1 スクリーンセットの初期設定

スクリーンセットに初めて入ったとき、スクリーンセットのデフォルト状態、ラジオボタンの状態、データグループのサイズ、デフォルト値、どのメニューオプションを使用可能にするかなどを設定したい場合があります。これには、SCREENSET-INITIALIZED イベントを使用します。次に例を示します。

```
1 SCREENSET-INITIALIZED
2     SET-BUTTON-STATE OK-BUTTON 1
3     SET-BUTTON-STATE CANCEL-BUTTON 0
4     ENABLE-OBJECT SAVE-AS-PB
5     DISABLE-OBJECT SAVE-PB
6     SET-DATA-GROUP-SIZE SALES-GROUP 100
7     MOVE "*.*" SELECTION-CRITERIA
```

行1:

このイベントは、スクリーンセットに初めて入ったとき（または呼出しプログラムが Dialog System に対して新しいスクリーンセットを要求した場合）に引き起こされます。

```
 SCREENSET-INITIALIZED
```

行2**〜**3:

SET-BUTTON-STATE は、チェックボックス、プッシュボタン、またはラジオボタンの状態を設定します。これら2つの文は、OK-BUTTON をオンにし、CANCEL-BUTTON をオフにします。

```
     SET-BUTTON-STATE OK-BUTTON 1
     SET-BUTTON-STATE CANCEL-BUTTON 0
```

行4**〜**5:

ENABLE-OBJECT ファンクションは SAVE-AS プッシュボタンを使用可能にし、DISABLE-OBJECT ファンクションは SAVE プッシュボタンを使用禁止にします。これらは、新しいファイルがあるときにユーザーが「名前を付けて保存」オプションを使ってファイル名を入力するようにした場合などに使用します。

```
ENABLE-OBJECT SAVE-AS-PB
DISABLE-OBJECT SAVE-PB
```

行6:

SET-DATA-GROUP-SIZE ファンクションは、データグループの内部サイズを定義します。このサイズを超えるすべての要素は保持されますが、リストボックスからはアクセスできません。

```
SET-DATA-GROUP-SIZE SALES-GROUP 100
```

行7:

この文は、入力フィールドのデフォルト値を設定します。

```
MOVE "*.*" SELECTION-CRITERIA
```

6.3.2 ウィンドウダイアログ

ウィンドウの視覚的な特性を定義したら、ダイアログを使ってウィンドウの作成および初期設定をすることができます。

6.3.2.1 ウィンドウの作成

ウィンドウを作成するためのダイアログの例を次に示します。

```
F2
    CREATE-WINDOW WINDOW1
```

F2 は、ユーザーが F2 キーを押したときに起こるイベントで、WINDOW1 は、作成するウィンドウの名前です。これは、ウィンドウを定義するときに入力した名前です。

このダイアログ文は、ウィンドウの初期設定を行い、表示できるようにします。見た目には何も起こりませんが、ウィンドウがバックグラウンドで作成されます。できたウィンドウは、SHOW-WINDOW ダイアログ文を使って表示することができます。

ウィンドウの初期設定は、あまり複雑な作業ではありませんが、若干の時間はかかります。これを避けるためには、ウィンドウが必要になるまえに作成し、表示する準備ができたら、SHOW-WINDOW によって表示するようにします。

6.3.2.2 最初のウィンドウの表示

スクリーンセットを起動するときに最初に実行したファンクションとして、アプリケーションのためのメインウインドウとダイアログボックスの表示があります。デフォルトでは、スクリーンセットを定義したときに最初に定義したウインドウまたはダイアログボックスが、実行時に最初に表示されるウインドウオブジェクトとなります。最初のウインドウとして、別のウインドウまたはダイアログボックスを指定するには２つの方法があります。

定義メニューから最初のウインドウを選択し、定義機能を使って、アプリケーションのメインウインドウまたはダイアログボックスを最初のウインドウとして定義することができます。最初のウインドウは、実行時に表示されるウインドウオブジェクトを定義したスクリーンセットの属性です。この方法によって最初のウインドウを指定する場合、ダイアログを追加する必要はありません。

しかし、フォーカスを明示的に設定したり、アプリケーションのメインウインドウを表示したい場合があります。たとえば定義時にはどのウインドウがメインウインドウかわからず、実行時に選択したい場合があります。このような場合は、SET-FOCUSファンクションまたはSHOW-WINDOWファンクションを使用します。

しかし、フォーカスを明示的に設定したり、アプリケーションのメインウインドウを表示したい場合があります。たとえば定義時にはどのウインドウがメインウインドウかわからず、実行時に選択したい場合があります。このような場合は、SET-FOCUSファンクション、SET-FIRST-WINDOWファンクション またはSHOW-WINDOWファンクションを使用します。　SHOW-WINDOWファンクションを使用する場合には、その前に以下のファンクションを発行して最初のウィンドウを明示的に閉じる必要があります。

```
SET-FIRST-WINDOW $NULL
```

SET-FIRST-WINDOWファンクションを使用する場合には必要はありません。

6.3.2.3 ウィンドウの表示

SHOW-WINDOW は、ウィンドウを表示しますが、ウィンドウにフォーカスを設定することはしません（あるオブジェクトに対してキーボードまたはマウスによる対話が行われる場合、そのオブジェクトには入力フォーカス（またはフォーカス)があるといいます)。

ウィンドウがまだ作成されていない場合は、SHOW-WINDOW はウィンドウを自動的に作成します。

ウィンドウを表示するダイアログの例を次に示します。

```
F3
    SHOW-WINDOW WINDOW2
```

F3 は、ユーザーが F3 キーを押したときに起こるイベントです。WINDOW2 は、表示するウィンドウの名前です。

ウィンドウに親ウィンドウがある場合、それも表示されます。

6.3.2.4 ウィンドウの非表示

UNSHOW-WINDOW は、ウィンドウ、すべての子ウィンドウ、コントロールを非表示にします。この場合、ウィンドウはなくなったわけではなく、再表示する場合にも、作成しなおす必要はありません。

UNSHOW-WINDOW を使うと、クライアント領域を整理することができます。

ウィンドウを再表示するには、SHOW-WINDOW を使用します。ウィンドウ、すべての子ウィンドウ、コントロールが、非表示になる前とまったく同じ状態で再表示されます。

UNSHOW-WINDOW には２つのパラメータがあります。１番目のパラメータには、非表示にするウィンドウを指定します（$WINDOW は、現在選択されているウィンドウを示します)。2番目のパラメータは、フォーカスを設定したいウィンドウを指定します。

次に例を示します。

```
F4
    UNSHOW-WINDOW $WINDOW WINDOW2
```

このダイアログ文は、現在フォーカスがあるウィンドウ（$WINDOW)を非表示にし、フォーカスを WINDOW2 に設定します。

*「スクリーンセットを使用する」*の章では、DELETE-WINDOW ではなく、UNSHOW-WINDOW を使用する場合の利点と欠点について述べています。

6.3.2.5 デフォルト親ウィンドウの変更

SET-DESKTOP-WINDOW は、パラメータによって指定されたウィンドウを、通常はデスクトップの子となるすべてのオブジェクトの親に指定します。

たとえば、WIN1（WIN1-HAND を取り扱う)、WIN2、WIN3 の３つの一次ウィンドウがあり、すべてがデスクトップの子であるとします。そして、次のファンクションを実行すると、

```
    SET-DESKTOP-WINDOW WIN1-HAND
```

次のようになります。

- WIN1 は、デスクトップの子のままです。
- WIN2 と WIN3 は、WIN1 の子になります。

このファンクションの用途の1つとして、現在のスクリーンセットからあるファンクションを選択したときに、スクリーンセットをスタックに入れて、新しいスクリーンセットを起動する場合があります。ただし、新しいスクリーンセットの最初のウィンドウは、最初のスクリーンセットの現在のウィンドウの子にしたいとします。

これには、最初のスクリーンセットで次のファンクションを使用します。

```
...
MOVE-OBJECT-HANDLE WIN1 $REGISTER
SET-DESKTOP-WINDOW $REGISTER
RETC
```

そして、プログラムの中で２番目のスクリーンセットをロードします。`WIN1` は、２番目のスクリーンセットにあるすべてのウィンドウの親ウィンドウになります。

*複数のスクリーンセット*の詳しい使い方については、第 13 章「呼出しインタフェースの使い方」を参照してください。

デフォルトの親ウィンドウをデスクトップにリセットするには、次のファンクションを使います。

```
SET-DESKTOP-WINDOW $NULL
```

注： すでに作成されたウィンドウは変化しません。このファンクションは、通常はデスクトップの子として作成される新しいウィンドウに対してのみ動作します。

6.3.2.6 ウィンドウの削除

ウィンドウを削除することは、視覚的には、ウィンドウを非表示にするのと同じことです。しかし、削除の場合、ウィンドウ、すべての子ウィンドウ、ウィンドウに付けられているすべてのコントロールの要求が取り除かれます。このウィンドウの他の要求を参照する場合は、ウィンドウを作成しなければなりません。

ウィンドウを削除するためのダイアログの例を次に示します。

```
F6
    DELETE-WINDOW WINDOW2 WINDOW1
```

`WINDOW2` は、削除するウィンドウで、`WINDOW1` は、フォーカスを設定するウィンドウです。

セッションのログオンウィンドウなど、そのセッションで以後にウィンドウが不必要にならないかぎり、DELETE-WINDOW は使用せずに UNSHOW-WINDOW を使用してください。

*「スクリーンセットを使用する」*の章では、DELETE-WINDOW ではなく、UNSHOW-WINDOW を使用する場合の利点と欠点について述べています。

6.3.2.7 ウィンドウへのフォーカスの設定

SET-FOCUS ダイアログは、キーボードとマウスのフォーカスをウィンドウに設定します。これによって、キーボー

ドまたはマウスとの対話がそのウィンドウに対して行われます。

このファンクションは、必要に応じて、フォーカスを設定するまえにウィンドウを作成および表示します。ウィンドウにフォーカスを設定するダイアログ文の例を次に示します。

```
F7
    SET-FOCUS WINDOW2
```

WINDOW2 は、フォーカスを設定するウィンドウです。

6.3.2.8 ウィンドウの移動

MOVE-WINDOW は、ウィンドウを現在の位置から新しい位置へ移動することによって、画面上でそのウィンドウを再編成します。たとえば、いくつかの値を監視および表示するためのウィンドウを作成した場合、他のウィンドウが表示されたときに、前のウィンドウを非表示にするのではなく、MOVE-WINDOW を使って移動することができます。

構文の例を次に示します:

```
F8
    MOVE-WINDOW $WINDOW 2 5
```

$WINDOW は、現在のウィンドウを示します。2は、「上」向き、5はウィンドウを移動する量を示します。これらのパラメータについての詳細は、『Dialog System リファレンス』の MOVE-WINDOW に関する説明を参照してください。

6.3.2.9 ウィンドウタイトルの変更

同様の機能を実行するウィンドウが複数必要な場合、1つのウィンドウを使用し、SET-OBJECT-LABEL（これは、Dialog System バージョン 2.1 の SET-TITLE に代わるファンクションです）を使ってウィンドウのタイトルだけを変更することができます。構文の例を次に示します。

```
F9
    SET-OBJECT-LABEL WINDOW1 NEW-TITLE
```

WINDOW1 は、ウィンドウを指定します。NEW-TITLE は、新しい名前が入ったデータ項目です。

注： ウィンドウが非表示の場合、SET-OBJECT-LABEL によってそのウィンドウが自動的に表示されることはありません。しかし、ウィンドウが表示されている場合、変更されたタイトルはすぐに表示されます。

6.3.2.10 ウィンドウを閉じる

ウィンドウのシステムメニューを図 7-2 に示してあります。このメニューには、[閉じる]というオプションがあります。このオプションを選択すると、ウィンドウが自動的に閉じます。このときに別の操作（たとえばアプリケーションを終了するなど)を実行することもできます。これには、CLOSE-WINDOW イベントを使用します。次に例を示します。

```
CLOSED-WINDOW
    SET-FLAG EXIT-FLAG
    RETC
```

このダイアログ文は、「閉じる」イベントを検出し、ユーザーがユーザーインタフェースを終了したいことを示すフラグをセットし、呼出しプログラムに戻ります。プログラムは、ファイルを閉じ、その他、アプリケーションの終了に必要なプロセスを実行します。

6.3.3 ボタンを押す

BUTTON-SELECTED は、プッシュボタン、チェックボックス、またはラジオボタンに関連付けられる主要なイベントです。このイベントは、ボタンをクリックしたり、チェックボックスが選択された場合に引き起こされます。

次に例を示します:

```
BUTTON-SELECTED
    MOVE 0 ORD-NO($REGISTER)
    MOVE 0 ORD-DATE($REGISTER)
    MOVE 0 ORD-VAL($REGISTER)
    MOVE 0 ORD-BAL($REGISTER)
    UPDATE-LIST-ITEM ORDER-BOX $NULL $EVENT-DATA
    RETC
    REFRESH-OBJECT-TOTAL
```

BUTTON-SELECTED イベントが引き起こされると、そのイベントに関連付けられているすべてのファンクションが実行されます。

6.3.3.1 ボタンの状態の設定と獲得

SET-BUTTON-STATE ファンクションと GET-BUTTON-STATE ファンクションは、ボタンとチェックボックスの状態を制御するのに使用します。スクリーンセットに初めて入ったときにボタンの状態を設定する方法については、*「スクリーンセットの初期設定」*の節を参照してください。

スクリーンセットの実行中にボタンの状態を設定および検索することもできます。たとえば、次のダイアログ部分は、

呼出しプログラムで実行された操作に従ってラジオボタンの状態を設定します。

```
BUTTON-SELECTED
    MOVE 2 ACTION-CODE
    RETC
    SET-BUTTON-STATE FILE-ACCESSED-RB 1
    ...
```

マスタフィールドに1から 10 の値を移動することによって、チェックボックスの状態を制御することもできます。たとえば、チェックボックスに結合されているマスタフィールドである `check-credit` というデータ項目がある場合、`check-credit` に1または0を移動することによって、チェックボックスの状態を変更できます。チェックボックスオブジェクトがすでに作成されている場合、状態の変化を表示するには、再生（リフレッシュ）する必要があります。

6.3.4 メニューバーダイアログ

ユーザーがプルダウンメニューのアクションを選択した場合に、それに応えるダイアログを定義することもできます。しかし、ダイアログをアクション項目に追加する前に、アクションに名前を付ける必要があります。アクションの命名について詳しくは、ヘルプの「*オブジェクトと属性*」のトピックを参照してください。

サンプルとして、Saledataサンプルアプリケーションから、次のダイアログを掲載します。この中のINSERT-CHOICE、CHANGE-CHOICE、およびDELETE-CHOICEは、メニューバーのアクションに付けられた名前です。ダイアログの中では、選択項目の前に付いた"@"は、メニュー項目であることをあらわしています。

```
 @INSERT-CHOICE
     IF= $REGISTER 0 NO-SELECTION-PROC
     CLEAR-OBJECT INSERT-DB
     SET-FOCUS INSERT-DB
 @CHANGE-CHOICE
     IF= $REGISTER 0 NO-SELECTION-PROC
     MOVE SALES-NAME($REGISTER) TMP-NAME
     MOVE SALES-REGION($REGISTER) TMP-REGION
     MOVE SALES-STATE($REGISTER) TMP-STATE
     REFRESH-OBJECT CHANGE-DB
     SET-FOCUS CHANGE-DB
 @DELETE-CHOICE
     IF= $REGISTER 0 NO-SELECTION-PROC
```

```
    MOVE SALES-NAME($REGISTER) TMP-NAME
    MOVE SALES-REGION($REGISTER) TMP-REGION
    MOVE SALES-STATE($REGISTER) TMP-STATE
    REFRESH-OBJECT DELETE-DB
    SET-FOCUS DELETE-DB
```

ユーザーがメニューバーの[変更]を選択すると、@CHANGE-CHOICEイベントがトリガーされ、このイベントのすべてのファンクションが実行されます。

6.3.4.1 選択を可能および不可能にする

メニュー項目が適用されない場合もあります。たとえば、更新されたばかりの新しいデータファイルが、保存されていない場合です。このファイルは新規のファイルであり、まだ名前が付けられていないため、[保存]よりも[名前をつけて保存](ファイル名を指定)を強制的にユーザーに選択させたいと考えます。この状況に対応したダイアログをコーディングする1つの方法として、次のようなものがあります。

```
1     ...
2     XIF= FILE-SAVED-FLAG 0 DISABLE-SAVE
3     ...
4   DISABLE-SAVE
5     DISABLE-MENU-CHOICE $WINDOW SAVE-CHOICE
6     ENABLE-MENU-CHOICE $WINDOW SAVE-AS-CHOICE
7     ...
```

行2:

```
    XIF= FILE-SAVED-FLAG 0 DISABLE-SAVE
```

XIFは、条件のファンクションです。FILE-SAVE-FLAGは、データブロックのデータ項目です。0という値は、保存されていないことをあらわします。DISABLE-SAVEは、実行する手続きです。この文は、ファイルがまだ保存されていない場合は、DISABLE-SAVE手続きを実行するという意味です。

行4:

```
    DISABLE-SAVE
```

実行する手続きを開始します。

行5:

```
    DISABLE-MENU-CHOICE $WINDOW SAVE-CHOICE
```

この文は、[保存]項目を使用不可にします。

行6:

```
    ENABLE-MENU-CHOICE $WINDOW SAVE-AS-CHOICE
```

この文は、[名前を付けて保存]を選択可能にします。

6.3.4.2 メニュー項目の選択

メニューバー項目は、ある項目を選択しときに何が起こるのかを定義するものです。たとえば、次のコードは、[編集]メニュー項目の[切り取り]、[コピー]、[貼り付け]の各オプションをコーディングする、ひとつの方法を示しています。

```
...
@CUT-CHOICE
    MOVE 5 ACTION-CODE
    RETC
@COPY-CHOICE
    MOVE 6 ACTION-CODE
    RETC
@PASTE-CHOICE
    MOVE 7 ACTION-CODE
    RETC
...
```

注: 先頭カラムの@は、メニュー項目であることを示します。

「*ウィンドウオブジェクト*」の章の「*メニューバー*」では、メニューバー項目を使用可能および使用不可能にする方法を含む、メニューバー項目を操作するダイアログが説明されています。

6.3.5 入力の妥当性検査

VALIDATE ファンクションを使うと、ウィンドウに付属されている特定のフィールドまたはすべてのフィールドを妥当性検査することができます。次に例を示します。

```
VALIDATE SALARY-RANGE-EF
```

SALARY-RANGE-EF 入力フィールドだけを妥当性検査します。

次の文は、SALARY-DETAILS-WIN ウィンドウ上のすべてのフィールドを妥当性検査します。

```
VALIDATE SALARY-DETAILS-WIN
```

エラーが検出された場合、VAL-ERROR イベントが引き起こされます。

たとえば、次のダイアログ部分は、現在のウィンドウにあるすべてのフィールドを妥当性検査するためのコーディングの例です。

```
BUTTON-SELECTED
    VALIDATE $WINDOW
    INVOKE-MESSAGE-BOX ERROR-MB "All fields OK" $REGISTER
    RETC
VAL-ERROR
    INVOKE-MESSAGE-BOX ERROR-MB ERROR-MSG-FIELD $REGISTER
    SET-FOCUS $EVENT-DATA
```

構文についての詳細は、第3章: *「データ定義とスクリーンセットの作成」*の*「入力フィールドの妥当性検査」*の節を参照してください。

6.3.6 手続きの使い方

一般に手続きは共通ルーチンとして使用します。たとえば、キャンセル機能を起動するには、少なくとも2つの方法があります。すなわち、[キャンセル]ボタンを使用するか、Escape キーを押すかです。これをコーディングするための最もよい方法は、[キャンセル]ボタンが選択された場合も、Escape キーが押された場合も、共通の手続きを呼び出すことです。次のダイアログ部分は、この方法を示しています。

```
BUTTON-SELECTED
    BRANCH-TO-PROCEDURE CANCEL-PROC
ESC
    BRANCH-TO-PROCEDURE CANCEL-PROC
  CANCEL-PROC
    cancel-functions
    ...
```

Dialog System には、手続きを起動するために2つの方法があります。それは、手続きへの分岐（COBOL の GOTO 文と同じ）と、手続きの実行（COBOL の PERFORM 文と同じ）です。両者の違いを考えるために、次のダイアログ文を見てください。

```
F1
```

```
    BRANCH-TO-PROCEDURE CLEAR-OK
    RETC
F2
    EXECUTE-PROCEDURE CLEAR-OK
    RETC
```

最初の文（F1 キーを選択する）では、CLEAR-OK 手続きに制御が渡されます。次に何が起こるかは、CLEAR-OK に入っているファンクションによって異なります。RETC ファンクションは実行されません。

F2 キーを選択した場合、CLEAR-OK が実行されたあと、EXECUTE-PROCEDURE 文の次の文である RETCファンクションに制御が戻ります。

Dialog System には、いくつかの分岐機能も用意されています。どれを使用するかは、手続きに分岐するか、手続きを実行するかによって異なります。

たとえば、IF= ファンクションは、2つの値を比較し、両者が等しい場合は手続きに「分岐」します。XIF= ファンクションでは、2つの値を比較し、両者が等しい場合は手続きを「実行」します。

いずれの場合も、両方の値が等しくない場合、IF= 文または XIF= 文の次の文が実行されます。

6.3.7 呼出しプログラムへの制御の返却

スクリーンセットの実行中にプログラムに戻りたい場合があります。たとえば、データベースから別のデータを検索したり、ファイルを更新したり、複雑な妥当性検査を行う場合などです。RETC ファンクションを使うと、呼出しプログラムに制御を戻すことができます。

たとえば、次のダイアログ部分は、ファイル削除機能を実行するために呼出しプログラムに戻ります。

```
BUTTON-SELECTED
    SET-FLAG DELETE-FILE-FLAG
    RETC
```

6.3.8 呼出しプログラムからの制御の取り戻し

通常、プログラムから戻ると、RETC の次にある文が実行されます。この動作は、必要に応じて変更できます。

プログラムから戻ったときに実行されるいくつかの動作を次に示します。

- ウィンドウを再生する。たとえば、プログラムによって関連するデータ項目が変更され、その結果をユーザーに見せたい場合
- スクリーンセットの状態をリセットします。たとえば、プログラムがファイルを保存したあと、[上書き保存]ボタンと[名前を付けて保存]ボタンの状態をリセットしたい場合

- オブジェクトをリセットする。たとえば、データベース内に表示したい追加データをデータベースからアクセスした場合
- プログラムによって実行された操作に基づいてデータの状態をチェックする。たとえば、プログラムの中で複雑な妥当性検査を行う必要があり、その結果を確認したい場合

構文の例として、次のダイアログ部分は、呼出しプログラムから戻ったあとに SALES-LIST リストボックスを再生し、ウィンドウを表示します。

```
@SORT-NAME-CHOICE
    MOVE 2 ACTION-CODE
    RETC
    REFRESH-OBJECT SALES-LIST
    SHOW-WINDOW SALES-WIN
```

6.4 ウィンドウマネージャによってとらえられるイベント

イベントには、Dialog System に入るまえに、Windows OS によってとらえられるものがあります。

たとえば、標準の Windows OS には、いくつかのアクセラレータキーがあり、メニュー上のファンクションにすばやくアクセスできます。たとえば、Alt キー+F4 キーは、アクティブなウィンドウとそれに関連するすべてのウィンドウを画面から消去します。Alt キー+F4 キーを押すことによって起こるイベントのダイアログを定義すると、そのイベントは、プレゼンテーションマネージャによってとらえられ、Dialog System には到達しません。

したがって、キーを押すことに対して定義された次のダイアログは、Dialog System がイベントを検出しないので実行されることはありません。

```
AF4
    SET-FOCUS NEW-EMPLOYEE-WIN
```

6.5 サンプルプログラム

ウインドウオブジェクトのいくつかの属性と動作を確認するために、Objectsというサンプルアプリケーションがデモンストレーションディレクトリに入っています。このアプリケーションは、COBOLプログラム、スクリーンセット、エラーファイルからできています。このアプリケーションを実行したら、デモンストレーションを見たいオブジェクトをオブジェクトメニューから選択します。.

プログラムをコンパイル、アニメート、実行するには、COBOLシステムのヘルプを参照して下さい。

例として、実行時にメニューバーの選択から動的操作を行うには、Dialog Systemのデモンストレーションディレクトリのdynmenuを参照してください。

6.6 サンプルダイアログ

この章に示したものより複雑なダイアログももちろん可能です。たとえば、次のダイアログ部分は、Dialog System の定義機能から取り出したものです（Dialog System 自身も Dialog System のアプリケーションです）。次の部分は、マウス構成オプションに付属し、マウスの構成を再定義するのに使用されます。

```
...
 @MOUSE-CONFIG-PD
     MOVE LEFT-BUTTON TMP-LEFT-BUTTON
     MOVE MIDDLE-BUTTON TMP-MIDDLE-BUTTON
     MOVE RIGHT-BUTTON TMP-RIGHT-BUTTON
     MOVE MOUSE-DEFAULT $REGISTER
     REFRESH-OBJECT LEFT-BTN-SB
     REFRESH-OBJECT MIDDLE-BTN-SB
     REFRESH-OBJECT RIGHT-BTN-SB
     IF= 1 MOUSE-DEFAULT DS21
     IF= 2 MOUSE-DEFAULT CUA
     IF= 3 MOUSE-DEFAULT BTN3
* Must be user defined action,
     EXECUTE-PROCEDURE ENABLE-CHOICES
     SET-BUTTON-STATE USER-DEFINED-RB 1
     SET-FOCUS USER-DEFINED-RB
* DS2.1 mouse behavior,
   DS21
     EXECUTE-PROCEDURE DISABLE-CHOICES
     SET-BUTTON-STATE DS21-RB 1
     SET-FOCUS DS21-RB
* CUA  behavior,
   CUA
     SET-BUTTON-STATE CUA89-RB 1
     EXECUTE-PROCEDURE DISABLE-CHOICES
     SET-FOCUS CUA89-RB
*
```

```
DISABLE-CHOICES
  DISABLE-OBJECT LEFT-BTN-SB
  DISABLE-OBJECT MIDDLE-BTN-SB
  DISABLE-OBJECT RIGHT-BTN-SB
ENABLE-CHOICES
  ENABLE-OBJECT LEFT-BTN-SB
  ENABLE-OBJECT MIDDLE-BTN-SB
  ENABLE-OBJECT RIGHT-BTN-SB
...
```

すべての機能が Dialog System 内にあることに注意してください。COBOL プログラムに制御が戻ることはありません。

第7章　スクリーンセットの使い方

前の章では、データ定義とアプリケーションのユーザーインタフェースを作成する方法について説明しました。この章では、次の内容について説明します。

- 呼出しインタフェースと、それによってスクリーンセットがCOBOLプログラムと通信できるようにする方法
- インタフェースにヘルプを追加する方法
- アプリケーションを最適化するために考慮すべき点

7.1 呼出しインタフェース

ここでは、次の内容について説明します。

1. スクリーンセットからCOBOLコピーファイルを生成する
2. 呼出しインタフェース - その構造と使い方
3. COBOLアプリケーションプログラムを記述する
4. スクリーンセットとCOBOLプログラムをデバッグしてアニメートする
5. アプリケーションをパッケージ化する

7.1.1 データブロックコピーファイルの生成

データブロックコピーファイルには、実行時に呼出しプログラムからDialog Systemに渡されるデータブロックが定義されます。このコピーファイルは呼出しプログラムに取り込む必要があります。コピーファイルにはバージョンチェック情報も含まれます。

スクリーンセットからコピーファイルを生成するには、次のように行います。

- コピーファイルの生成方法を決定する、スクリーンセットのシステム設定オプションを選択します。

  定義ソフトウェアをシステム設定して、さまざまな方法でコピーファイルを書き出すことができます。必要なコピーファイルの種類については、呼出しプログラムをMicro Focus COBOLとANSI COBOL標準のどちらを使って記述しているかによって変わります。コピーファイルのオプションをシステム設定する方法については、ヘルプの「*Dialog Systemシステム設定*」のトピックを参照してください。

- 定義ソフトウェアのメインウィンドウで、[ファイル]メニューから[生成/データブロックコピーファイル]を選択します。

ダイアログボックスが表示され、コピーファイルのパスとファイル名を選択することができます。デフォルトは、*スクリーンセット名.cpb*となります。

- コピーファイル名を選択して[OK]をクリックします。

  メッセージボックスが表示され、生成および保存中のコピーファイルの割合(%)が表示されます。保存が終わると、メッセージボックスが閉じます。

7.1.1.1 コピーファイル生成オプション

コピーファイルを生成するときは、次のいくつかのオプションを考慮します。

- データ項目名が、許容範囲の30文字を超えるときは、30文字に切り捨てられます。

  Dialog Systemは、この結果に対するコメントを、フルネームに引用符を付けて、コピーファイルの項目の横に表示します。

- [スクリーンセットIDの接頭辞が付けられたフィールド]システム設定オプションがオンの場合は、データブロックのあらゆるデータ名にスクリーンセットIDの接頭辞が付けられます。データ項目名は、接頭辞を入れて30文字以内にする必要があります。

  このオプションは、次のように変更することができます。

  - 特定のスクリーンセットは、[オプション]メニューの[システム設定、スクリーンセット]を使って、適切なチェックボックスを設定します。
  - すべてのスクリーンセットは、システム設定ファイルds.cfgを編集するか、[オプション]メニューの[システム設定、デフォルト]を使います。

- 名前が切り捨てられた結果、2つのデータ項目名が一致することがあります。そのような場合は、コンパイル時にエラーが起こります。

  これを訂正するには、データ定義ウィンドウでデータ項目名のどちらかを変更し、コピーファイルを再生成します。

注: 次のようにコマンドラインを指定すると、データブロックコピーファイルを生成することができます。

```
dswin/g screenset-name
```

これは、コピーファイルを生成して終了します。

コピーファイルの生成について詳しくは、ヘルプの「*Dialog System概要*」、「 *呼出しインタフェース*」、「*データ*

定義と妥当性検査」の各トピックを参照してください。

7.1.2 呼出しインタフェースの構造

ここでは、Dialog Systemを呼び出す一番ストレートな方法について説明します。Dialog Systemの初心者は、ここで説明する知識があれば、簡単なアプリケーションを開発することができるようになります。

Dialog System呼出しインターフェイスの構造は、非常に単純であり、呼出しプログラムとDialog Systemの間で渡される、次の2つの情報から構成されます。

- コントロールブロック - コントロールブロックコピーファイルのDS-CONTROL-BLOCKに該当 - は、呼出しプログラムとDialog Systemの間をコントロールするために使われます。
- データブロック - データブロックコピーファイルのDATA-BLOCKに該当 - は、呼出しプログラムとDialog Systemの間にユーザーデータを運ぶために使われます。

スクリーンセットを使用するには、Dialog Systemのコントロールブロックとデータブロックを使って、COBOLプログラムからDialog Systemランタイムモジュールdsgrunを呼び出す必要があります。Dialog Systemの基本的な呼出しは、次の形式で行います。

```
CALL "DSGRUN" USING DS-CONTROL-BLOCK,
                    DATA-BLOCK
```

ランタイムシステムは、最初にカレントディレクトリでスクリーンセットを検索し、次にCOBDIR環境変数に指定されているディレクトリを検索します。

スクリーンセットシステム設定ダイアログボックスの[スクリーンセットIDの接頭辞が付けられたフィールド]でコピーファイルを生成すると、データブロックにスクリーンセット名の接頭辞が付けられます。そのため、この接頭辞をDialog Systemに対する呼出しで指定する必要があります。たとえば、インストールディレクトリDialogSystem¥demoにあるサンプルプログラムentries.cblでは、"entry"というスクリーンセットIDを持つスクリーンセットを使用しており、次のように呼出しを行っています。

```
CALL "DSGRUN" USING DS-CONTROL-BLOCK,
                    ENTRY-DATA-BLOCK
```

Dialog Systemに対する最初の呼出しのあとで、呼出しが不成功に終わった場合のルーチンを次のように指定します。

```
IF NOT DS-NO-ERROR
   MOVE DS-ERROR-CODE TO DISPLAY-ERROR-NO
   DISPLAY "DS ERROR NO: " DISPLAY-ERROR-NO
  PERFORM PROGRAM-TERMINATE
END-IF
```

COBOLプログラムの中でDialog Systemを実際に呼び出す位置は、重要ではありません。プログラムがDialog Systemを呼び出す必要があるときは、いつでも呼び出せるように別ルーチンに置くのがよい習慣です。

ヘルプの「*呼出しインタフェース*」のトピックでは、スクリーンセットの実行方法をコントロールするために、コントロールブロックで指定可能なオプションについて、説明しています。

呼出しインタフェースをさらに高度に使う方法については、「*上級テクニック*」および「*複数のスクリーンセット*」の章を参照してください。

7.1.2.1 スクリーンセットの使い方をコントロールする

呼出しプログラムがスクリーンセットの操作をコントロールする方法については、スクリーンセットと呼出しプログラムを設計する際に、詳細に検討しなければなりません。次の注意点を配慮する必要があります。

- 機能をどのように分割するか、別のスクリーンセットに分けるかどうか
- セキュリティを配慮すると、データを個別にアクセスするユーザーグループに、異なるスクリーンセットを提供するかどうか
- ひとりのユーザーが、データの表示、比較または編集を同時に行えるように、スクリーンセットの複数インスタンスを使うかどうか

Dialog System呼出しインタフェースを使った、画面コントロールの基本に関する情報は、ヘルプの「*呼出しインタフェース*」を参照してください。

7.1.2.2 複数のスクリーンセットの使い方

複数のスクリーンセットは、スクリーンセットスタックからプッシュおよびポップして、使うことができます。定義によると、FIFO(先入れ後出し)の方式をとっています。スクリーンセットをプッシュおよびポップして、次の操作に役立てることができます。

- 特定の機能用に使用したスクリーンセットが、必要なくなった場合に、画面から削除する
- プログラムの初期化中に、複数のスクリーンセットをロードし、必要になったらプッシュおよびポップ(または使用)する
- 未使用のウィンドウや入力フォーカス持たないウィンドウによって、乱されることのない表示を維持する

スクリーンセットスタックでスクリーンセットをプッシュするための必須条件はありません。あらゆるスクリーンセットやスクリーンセットの発生は、プッシュまたはポップすることができます。プッシュされたスクリーンセットは、通常、メモリにスタックされますが、メモリが小さい場合は、ディスクにページングされます。

スクリーンセットスタックにスクリーンセットをプッシュして、新しいスクリーンセットを開始するには、次のように、Dsgrunを呼び出します。

```
move ds-push-set to ds-control
call "dsgrun" using ds-control-block,
                    data-block
```

ds-push-setは、"S"という値をds-controlに置きます。既存のスクリーンセットは、スクリーンセットスタックにプッシュされます。あるスクリーンセットをスクリーンセットスタックからポップするときは、次のどちらかを使用することができます。

- ds-quit-setは、既存のスクリーンセットを閉じ、最初のスクリーンセットをスクリーンセットスタックの最上位からポップします。
- ds-use-setは、指定されたスクリーンセットをスクリーンセットスタックからポップしますが、既存のスクリーンセットは閉じません。

詳しくは、「*複数のスクリーンセット*」の章を参照してください。

7.1.2.3 同じスクリーンセットの複数インスタンスの使い方

Dialog Systemでは、同じスクリーンセットの複数インスタンスを、呼出しプログラムで使うことができます。この機能は、各グループ項目が同じ形式のデータグループを操作するときに役立ちます。同じスクリーンセットの複数インスタンスを使うことによって、必要に応じて、1つのスクリーンセットに複数の項目を表示させたり、比較させたり、更新させることができます。

同じスクリーンセットの複数インスタンスを使うと、プログラムの中で、次のことを行う必要があります。

- スクリーンセットのインスタンスの数をトレースする
- Dsgrunに渡されるデータブロックが、スクリーンセットのインスタンスにとって正しいことを確認する

複数インスタンスを使っており、最初にスクリーンセットが"N"または"S"の呼出しで起動している場合は、インスタンス値が割り当てられ、ds-screenset-instanceコントロールブロックのフィールドに置かれます。

インスタンス値は、特定のスクリーンセットのインスタンスにユニークなものです。インスタンス値は特定の順番で割り当てられないため、アプリケーションでインスタンス値をトレースする必要があります。

dssysinf.cpyの中のds-event-screenset-idおよびds-event-screenset-instance-noを調べることによって、アクティブなインスタンスをトレースすることができます。これはプログラムの作業場所節にコピーする必要があります。

dssysinf.cpyについて詳しくは、「*Panels V2の使い方*」の章を参照してください。

スクリーンセットの新しいインスタンスのロードを指定するには、Dsgrunを呼び出すときに、ds-controlをds-use-instance-setに設定します。

定義ソフトウェアを使って、スクリーンセットAnimatorを通して実行する場合は、スクリーンセットの複数インスタンスの使用はサポートされていません。しかし、アプリケーションからDsgrunを呼び出す場合は、複数インスタンスがサポートされています。

詳しくは、「*複数のスクリーンセット*」の章を参照してください。

7.1.3 COBOL**アプリケーションプログラムの書き方**

データブロックコピーファイルには、ユーザーデータだけでなく、Dialog Systemがスクリーンセットから呼び出されたときにチェックする、スクリーンセットのバージョン番号も含まれます。そのために、呼出しプログラムは、Dialog Systemランタイムを呼び出す前に、コントロールブロックのデータ項目にバージョン番号をコピーする必要があります。

7.1.3.1 **コントロールブロック**

コントロールブロックは、呼出しプログラムとDialog Systemの間でコントロール情報やデータを運ぶために利用されます。

コントロールブロックは、次のものから構成されます。

- データブロック、コントロールブロックおよびスクリーンセットのバージョンをチェックするバージョンチェック項目
- Dialog Systemにファンクションの実行を伝える入力フィールド

  スクリーンセットをコントロールするために使用する定数の名前と値など
- 値を返す出力フィールド

  エラーコードと妥当性検査フィールドなど。

Dialog Systemには、3つのバージョンコントロールのブロックコピーファイルが提供されています。

- ds-cntrl.ans

  ANSI-85準拠のCOBOLで使います。
- ds-cntrl.mf

  Micro Focus準拠のCOBOL(COMP-5)で使います。
- dscntrlx.mf

  Micro Focus準拠のCOBOL(COMP-X)で使います。 3をVERSION-NOではなくDS-VERSION-NOへMoveして使います。

呼出しプログラムを書くときは、次のステートメントを使って、コピーファイルをプログラムの作業場所節にコピーする必要があります。

```
COPY "DS-CNTRL.MF".
```

ANSI-85準拠のCOBOLを使う場合は、*ds-cntrl.ans*コピーファイルを使う必要があります。

データブロックフィールドの情報は、コントロールブロックと共に、Dialog Systemと呼出しプログラムの間で双方向に渡されます。これは、制御が一方から他方へ渡されるときは、いつでも行われます。

また、スクリーンセット名とDialog Systemの動作をコントロールする他の情報が、コントロールブロックに含まれていることも確認する必要があります。

Dialog Systemでは、プログラムがユーザーの行動を決めるのではなく、ユーザーがプログラムの次に実行する機能を決めることもできます。Dialog Systemから返されたデータブロックのフラグには、ユーザーの動作によって引き起こされた値が入っており、次に何を行うのかをプログラムに伝えます。

プログラムは、次のようなさまざまな方法で、応答することができます。

- 保存された情報を修正する
- データベースから追加情報を検索する
- 結果またはエラーメッセージを表示する
- アプリケーションの利用を支援するものを提供する

  単純なアプリケーションの場合は、Dialog Systemのメッセージボックスで、ヘルプを完全に操作することができます。ヘルプを操作するための代替方法は、ヘルプフラグが設定されたときにいつでもヘルプを提供するようなコードをプログラムに書くことです。
- ユーザーが入力した情報に妥当性検査を行う
- ユーザーからの追加入力を要求する

  レコードを保存または消去したあとで、Dialog Systemに戻って行います。

「*チュートリアル - サンプルスクリーンセットの使い方*」の章のサンプルスクリーンセット(「*チュートリアル - サンプルスクリーンセットの作成*」の章で作成したもの)を利用した、COBOLプログラムを参照することができます。このプログラムは、デモプログラムとしてDialog Systemソフトウェアと共に提供されています。

7.1.4 スクリーンセットとCOBOLプログラムのデバッグとアニメート

NetExpressは、編集、デバッグ、アニメートのための統合環境を提供しています。

アプリケーションをデバッグしているときは、各プログラムのソースコードが別のウィンドウに表示されます。コー

ドをアニメートしているときは、各ステートメントが実行されるにつれて、ソースの各行が次々にハイライトされ、各ステートメントの結果を表示します。プログラムを実行する速さをコントロールしたり、実行を中断してデータ項目を調べたり変更することができます。詳しくは、ヘルプの「*デバッグ*」のトピックを参照してください。

プログラムをテストしたあとで、プログラムのスクリーンセットを個々に変更することができます。スクリーンセットとプログラムの組み合わが要求を満たすまで、この操作を続けることができます。

7.1.4.1 スクリーンセットのテスト

Dialog Systemの主要な利点は、スクリーンセットが完成するかなり前から、COBOLプログラムがなくても、インタフェースの部品を簡単にテストできることです。未完成のスクリーンセットや、スクリーンセットのごく一部のプロトタイプを作成して、それがどのようなものかを印象づけたり、アイディアを試してみることができます。

プロトタイプを作成することは、インタフェース開発期間の中で特に重要なものであり、これによって、タスクの多くを迅速に処理することができます。そのため、システムの作業用モデルをすばやく構築することができ、ユーザーはシステムが作動中の様子を端末で見ることができます。すると、迅速かつ費用をかけずにシステムを変更することができます。

スクリーンセットのテストをダイアログなしで行い、デスクトップのレイアウトが見栄えよいか、フィールドが正しく設定してあるか、などを確かめることができます。

7.1.4.1.1 スクリーンセットAnimatorの使い方

Dialog SystemスクリーンセットAnimatorは、Dialog Systemに提供されたユーティリティであり、作成したスクリーンセットのテストとデバックを行うことができます。スクリーンセットAnimatorを使うことによって、スクリーンセットを使うプログラムを書く以前でも、スクリーンセットの概観と雰囲気、機能をテストすることができます。スクリーンセットAnimaorを使うと、次のことができます。

- 操作オブジェクト、ダイアログ、データブロックをすべて用意して、個別にスクリーンセットを実行する
- ユーザーデータとコントロール値を設定し、スクリーンセットを呼び出し、操作を実行し、戻って必要な数だけ戻り値を調べる
- ダイアログのイベントとファンクションの実行をトレースする
- 定義モードに戻り、スクリーンセットまたはダイアログを変更し、スクリーンセットを再び実行する
- スクリーンセットAnimatorを使って呼出しプログラムを実行し、データをチェックして、スクリーンセットと呼出しプログラム間で渡される情報を制御する

メインウィンドウの[ファイル]メニューから[デバック]を選択します。スクリーンセットAnimatorウィンドウが、図 7-1のように表示されます。

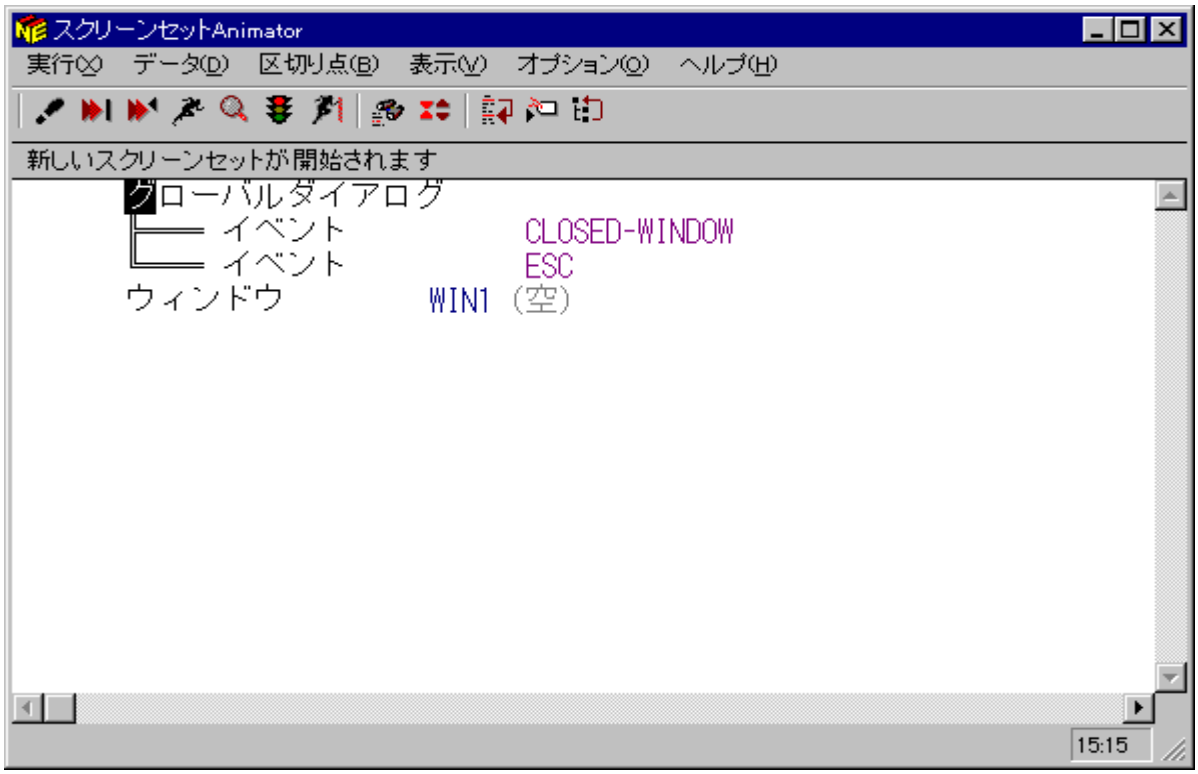


図 7-1 スクリーンセットAnimatorウィンドウ

この図の詳細は、ヘルプの「*スクリーンセットAnimator*」のトピックで説明されています。

スクリーンセットAnimatorからスクリーンセットを実行するには、[実行]メニューから[実行]を選択します。

スクリーンセットをテストするときは、変更値やデフォルト値でスクリーンセットの主な機能をチェックする必要があります。

スクリーンセットは、呼出しプログラムに戻る位置までくると、スクリーンセットAnimatorに戻り、そこでコントロールブロックおよびデータブロックに配置された値を調べることができます。

データブロックはDsgrunに渡されたデータの現在値を表示します。この時点でスクリーンセットAnimatorで見ることができるあらゆる情報を変更し、別のスクリーンセットの処理をテストすることができます。データブロックを変更するには、[データ]メニューから[調査]を選択してデータブロックを表示します。次に、データプッシュボタンを選択してデータブロック項目を表示し、選択プッシュボタンでデータブロック項目を変更することができます。

再度、値を変更して、必要なときはスクリーンセットを再実行し、呼出しプログラムの動作をシミュレートすることができます。

必要に応じてスクリーンセットを変更し、そのインタフェースが納得するまで再テストすることができます。

詳しくは、ヘルプの「*スクリーンAnimator*」トピックを参照してください。

7.1.4.2 ダイアログを定義する

ユーザーインタフェースは、単なるグラフィック表示を超えるものです。完成した仕様は、どのようにユーザーとコンピュータが対話し、どのようにユーザーインタフェースのソフトウェアがアプリケーションソフトウェアと対話するかを説明します。

一度、表示の概観を定義すると、ユーザーとマシンとの実行時の対話を定義することが必要になります。この対話はダイアログと呼ばれます。ダイアログはイベントとファンクションから構成されるものです。イベントが発生すると、そのイベントに関連するファンクションが実行されます。イベントは、キーボードのキーが押されたり、メニューが選択されたり、オブジェクトが選択されることによって引き起こされます。

たとえば、BUTTON-SELECTEDのようなDialog Systemのイベントは、ユーザーがプッシュボタンを選択(マウスまたはキーボードを使って)したときに発生します。選択されたボタンが[入力]の場合、それに関連するファンクションがCREATE-WINDOWであれば、ユーザーがさらに情報を入力する新しいウィンドウが作成されます。

Dialog Systemを使うことによって、ユーザーと表示オブジェクトの間のダイアログを作成またはカスタマイズすることができます。コントロールダイアログは個々のコントロールに影響し、ウィンドウダイアログは個々のウィンドウまたはダイアログボックスのあらゆるコントロールに影響し、グローバルダイアログはウィンドウとオブジェクトのすべてに影響します。イベントが起こると、Dialog Systemはまず関連するコントロールダイアログを検索し、次に関連するウィンドウダイアログを、最後にグローバルダイアログを検索します。

ダイアログのステートメントについて詳しくは、「*ダイアログの使い方*」の章と、ヘルプの「*Dialog System概要*」のトピックを参照してください。

7.1.4.3 スクリーンセットを再度テストする

スクリーンセットを再び保存して、インタフェースを再実行します。フィールドにデータを入力し、インタフェースの適切なラジオボタン、リスト項目、他のオブジェクトを選択します。

7.1.4.4 スクリーンセットの変更

スクリーンセットを再度テストすると、それを変更したくなります(たとえば画面のレイアウトの改良など)。スクリーンセットに満足するまで、ここで説明したあらゆる手順を繰り返すことができます。

今度は、スクリーンセットを含むユーザーインタフェースを利用するため、COBOLプログラムを書くことが必要になります。

Windows GUIアプリケーションウィザードでは、必要に応じて、自動的にCOBOLプログラムを書くことができます。「*Windows GUIアプリケーションウィザード*」の章を参照してください。「*サンプルプログラム*」の章には、とても簡単なプログラムを生成するサンプルコードが記載されています。このプログラムは、ユーザーの入力を読み込み、ユーザーの指示にしたがって、それを保存または消去するものです。

7.1.5 アプリケーションのパッケージ化

完成したアプリケーションを作るには、さまざまなサブタスクを完了する必要があります。

NetExpressから提供されるプロジェクト機能を使って、Dialog Systemアプリケーションを構築します。詳しくは、ヘルプの「*アプリケーションの構築*」トピックを参照してください。

ヘルプの「*コンパイル*」トピックには、製品用のアプリケーションを準備するために、次に何をすることが必要かを説明しています。

テストが完了すると､完成した製品をアセンブルする準備が整ったことになります｡完成した製品はディスクにコピーされ、エンドユーザーに送られ、他のマシンにロードされ、アプリケーションとして実行されます。

アプリケーションのサイズによって異なりますが、完成した製品は、次のものから構成されます。

- 実行可能モジュール。これは、業界標準の.exeおよび.dllのファイル形式です。
- ランタイムサポートファイル。これらのファイルは、ファイルI/Oやメモリ管理などの機能を実行します。

7.2 ヘルプの追加

Dialog System拡張機能を使って、Dialog Systemスクリーンセットから直接、Windowsのヘルプを表示することができます｡Dialog System拡張機能とは､CALLOUTダイアログファンクションを使って実装されたダイアログファンクションに与えられた用語です。Dialog System拡張機能は、Windowsのヘルプを表示したり、ファイル選択機能を提供するような、多くの定型的なプログラム作業を実行できるように、提供されたものです。Dialog System拡張機能について詳しくは、ヘルプの「*ダイアログシステム拡張機能*」トピックを参照してください。

Helpdemoスクリーンセットは、Dsonline Dialog System拡張機能を利用しています。これは、Windowsのヘルプを表示するDialog Systemの拡張機能です。「*チュートリアル - サンプルスクリーンセットの作成*」の章の「*ヘルプの追加*」では、Helpdemoスクリーンセットについて詳しく検討しています。

7.3 アプリケーションの最適化

ヘルプの「*効率的なプログラミング*」のトピックを見ると、COBOLプログラムを最適化するためのヒントを得ることができます。

ここでは、Dialog Systemアプリケーションを最適化するための、ヒントを追加して提供します。具体的には、次の内容が含まれます。

- COBDIR環境変数を使って、ディレクトリの検索時間を最適化する
- イベントのダイアログと手続きの検索を最適化する
- DELETE-WINDOWよりUNSHOW-WINDOWを使う

- オブジェクト名を最小化する
- ランタイム保存形式を使う
- `ds-no-name-info`フラグを設定する

7.3.1 ディレクトリの検索を制限する

Pathステートメントは、必要なプログラムまたはファイルがカレントディレクトリにない場合、指定されたドライブの指定されたディレクトリを検索するように、オペレーティングシステムに知らせるものです。Pathは、選択対称となるパスが定義された順番で、検索パスを定義します。

開発環境では、WindowsのシステムレジストリはCOBOLランタイムによって使用され、Dialog Systemインストールの場所と現在のNetExpressプロジェクトの場所を識別します。

アプリケーションが完成し、製品の準備ができたら、Path環境変数を修正して、アプリケーションに必要なディレクトリが、選択対称パスの最上位に近づけるようにします。

7.3.2 イベントダイアログの検索

「*ダイアログの使い方*」の章では、Dialog Systemがイベントと手続きの検索に使用する、次の3階層のダイアログについて説明しています。

- コントロールレベル
- ウィンドウレベル
- グローバルレベル

これは、イベントがプッシュボタンなどのコントロールで発生したときに、そのコントロールに設定されたダイアログにイベントがリストされているかどうかを、Dialog Systemが調べるためのものです。

コントロールにリストされていない場合は、このダイアログのイベントは、コントロールを含むウィンドウに設定されていると考えられます。ウィンドウでもイベントがリストされていない場合は、グローバルダイアログを検索します。

次のことを、試すようにしてください。

- ダイアログと手続きは、できるだけ下の階層に保持する
- グローバルダイアログは、一般的なダイアログと手続のためとっておく
- もっとも一般的なイベントは、イベントテーブルの最上位の近くに保持する

もちろん、ダイアログが増えれば増えるほど、実行速度は遅くなります。また、グローバルレベルでイベントや手続

きを定義すればするほど、やはり遅くなります。グローバルダイアログは一般的なルーチンを集めるには有効ですが、下の階層で見つからないイベントが起こるたびに検索されることも意味します。

7.3.3 UNSHOW-WINDOW 対 DELETE-WINDOW

UNSHOW-WINDOWとDELETE-WINDOWファンクションは似ています。どちらも、指定したウィンドウやダイアログボックスを非表示にして、入力フォーカスを他のどこかに設定します。

DELETE-WINDOWファンクションは、ウィンドウとそのコントロールを削除します。再びそのウィンドウを表示したい場合は、明示的または暗黙的のいずれかで再度作成する必要があります。(ウィンドウが作成されない場合、SHOW-WINDOWとSET-FOCUSファンクションが暗黙的に作成を行います。)ダイアログボックスとウィンドウを作成する(そしてすべてのオブジェクトをそこに定義)には、多少時間がかかります。

しかし、UNSHOW-WINDOWファンクションはウィンドウを作成したまま残しておくため、再度、すばやく表示することができます。

したがって、ウィンドウまたはダイアログボックスを非表示にして、あとで再度表示する場合は、UNSHOW-WINDOWファンクションを使います。しかし、システムログオンウィンドウなどのウィンドウまたはダイアログを終了する場合は、リソースを開放するDELETE-WINDOWファンクションを使います。

7.3.4 オブジェクトの名前を最小化する

プッシュボタンやチェックボックスなどのオブジェクトは、ダイアログで参照されないかぎり、名前を付ける必要はありません。ダイアログで参照するオブジェクトだけに名前を付けることによって、わずかな量の記憶域を開放し、Dialog Systemがオブジェクトの検索に費やす時間を減らします。

7.3.5 ランタイム保存形式

Dialog Systemアプリケーションを配布する場合は、ランタイム保存形式オプションの使用を考慮します。このオプションによって、スクリーンセットを実行するには十分な情報量を保存することができますが、検索して編集を行うには十分ではありません。

このオプションは、セキュリティ上の問題点に対処しており、アプリケーションに必要なディスク容量も減らします。小さくなったスクリーンセットのサイズは、元のサイズの3分の1ほどになります。

Dialog Systemは、それほど多くの情報をロードする必要がないため、わずかにパフォーマンスが向上することがあります。

注: 一度このオプションセットでスクリーンセットを保存すると、それを変更することはできません。したがって、

編集できるスクリーンセットのコピーを保存することが重要になります。

7.3.6 ds-no-name-infoの使い方

コントロールブロックに`ds-no-name-info`フラグが設定されている場合、 Dialog Systemは、呼出しプログラムに戻っても、スクリーンセットのヒープ領域を読み込んで、`ds-object-name`と`ds-window-name`の値を得ることはありません。これら2つのフィールドのどちらも使っていない場合は、このフラグをTRUE(真)に初期化します。たとえば、次のように行います。

```
move 1 to ds-no-name-info
```

これは、たとえばRETCファンクションのあとで、Dialog Systemからアプリケーションに戻ると、少しは速く終了するようになります。

クリーンセットAnimatorはスクリーンセットに保存された名前情報が必要であるため、このオプションを設定すると、スクリーンセットのデバックにスクリーンセットAnimatorを使えなくなります。

7.4 詳細情報

複数のスクリーンセットの使い方や同じスクリーンセットの複数インスタンスなどの、呼出しインタフェースのより高度な使い方については、「*複数のスクリーンセット*」の章を参照してください。

Win32の環境以外からNetWxpressにアプリケーションを移行する場合は、クロス環境の注意点のヘルプが記載された「*異なるプラットフォームへの移行*」の章を参照してください。

第8章　Windows GUIアプリケーションウィザード

前の章ではDialog Systemの基本要素を紹介しました。この章では、新しいWindows GUIアプリケーションウィザードが持つ機能について説明します。

- ツールバー、メニューバー、ステータスバーなどを持ったスクリーンセットをすばやく簡単に作成するための手段
- インストールされた任意のデータベースに対して容易にアクセスおよび問合せするためのルート

  これは、データアクセスオプションを選択し、ウィザードでSQLを定義するだけで実行できます。

ウィザードプロセスは、新しいスクリーンセットを作成するだけではなく、ユーザーが要求する機能に合わせて構成された関連COBOLプログラムを自動生成できます。スクリーンセットと関連COBOLプログラムは、必要に応じてプロジェクトに自動的に追加されます。

注意：Windows GUIアプリケーションウィザードが出力するプログラムとファイルは、ユーザーが独自のアプリケーションを開発するための出発点となります。これらは、あらゆる状況に対応する万能のコードではありませんが、基本機能の使い方を習得することを目的としています。ユーザーは、提供されたコードを個々のニーズに適合させることができます。

---

*「入門書」のWindows GUIアプリケーションの作成*の章には、このウィザードの実際の使い方に関する説明があります。

8.1 ウィザードの起動

Windows GUIアプリケーションウィザードを起動するには、次の3つの方法があります。

- NetExpress IDEから直接。[ファイル]メニューから[新規作成]を選択し、Dialog System スクリーンセットを選びます。
- IDEから直接。[ファイル]メニューから[新規作成]を選択し、[プロジェクト]を選びます。プロジェクトタイプとして[Windows GUIプロジェクト]を選択します。
- Dialog Systemから。[ファイル]メニューの[新規作成]を選択します。

注意：ウィザードをIDEから開く場合、プロジェクトが開いていないときは、新しいプロジェクトが作成されます。

この場合、プロジェクトの名前と場所を指定する必要があります。

8.2 ウィザードの使い方

この節では、ウィザードの各ステップの機能を簡単に説明します。

8.2.1 ステップ1：スクリーンセット名

新しいスクリーンセットの名前と場所が与えられます。これらは変更できます。

8.2.2 ステップ2：インタフェースタイプ

ここでは、新しいスクリーンセットのインタフェースをMultiple Document Interface（MDI）とSingle Document Interface（SDI）から選択します。MDIアプリケーションが必要な場合、インタフェースが持つMDI子の数を指定します。SDIでは、一次ウィンドウがいくつ必要かを指定します。

8.2.3 ステップ3：クラスライブラリ機能

新しいスクリーンセットには、ステータスバーや一次ウィンドウツールバーなどなどの機能を選択できます。すべての機能について、適切なデータブロック、ダイアログ、コントロールプログラムを生成できます。このステップを図 8-1に示します。

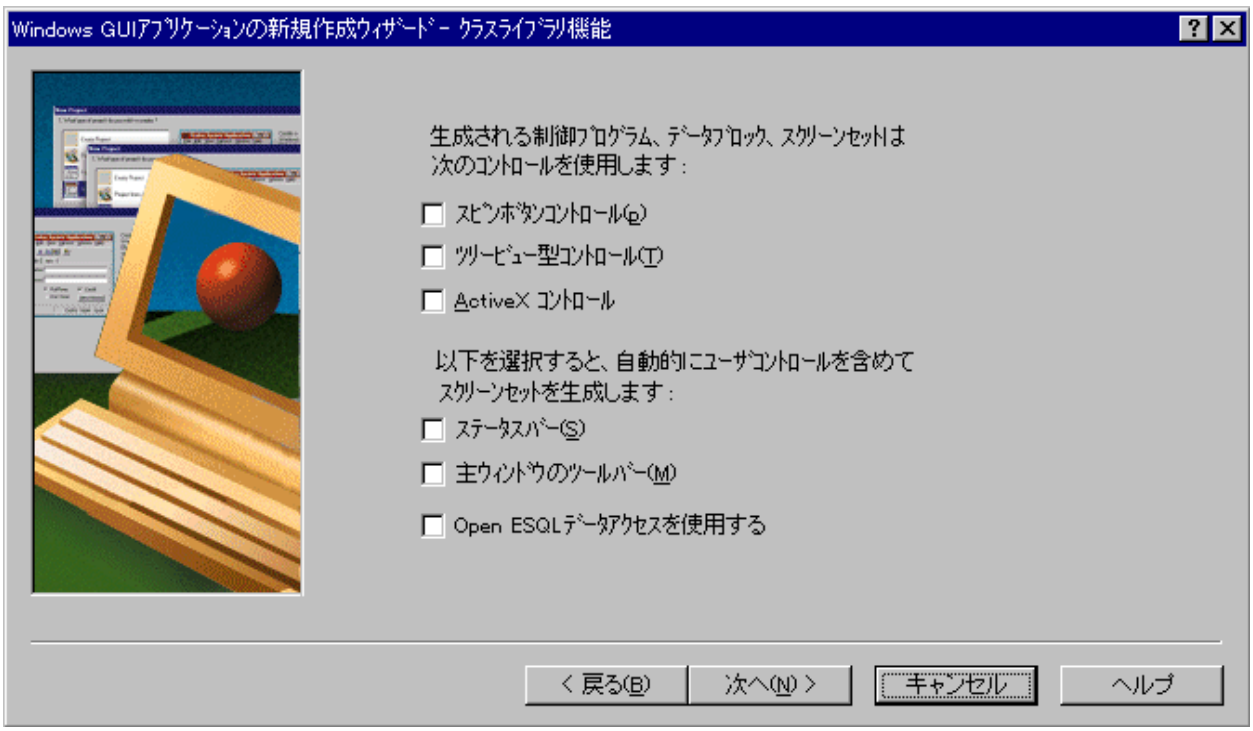


図 8-1 クラスファイル機能

[OpenESQLデータアクセスを使用する]を選択すると、インストールされた任意のODBCデータソースにアクセスし、次のステップで説明する問合せをセットアップできます。このオプションを選択しない場合、次のステップをスキップし、ステップ5に進んでください。

[OpenESQLデータアクセスを使用する]を選択した場合、スクリーンセットのステータスバーが自動的に選択されます。

8.2.4 ステップ4：クエリーの定義

ステップ3で[OpenESQLデータアクセスを使用する]を選択しなかった場合、このステップをスキップし、ステップ5に進んでください。

注意：インストール時にODBCドライバをインストールしなかった場合、このステップを実行するまえにインストールする必要があります。

このステップでは、次のことができます。

- システムに登録されているすべてのODBCデータソースを左側の子ウィンドウにリストする。

  OpenESQLアシスタントテクノロジを使用するインストール済みデータソースにアクセスします。

  データソースをセットアップしていない場合、NetExpressのインストール時に自動的にセットアップされるサンプルデータソースを利用できます。その中でNetExpress Sample2はsample.mdbというサンプルMicrosoft Accessデータベースを指し示します。これはNetExpressに付属しており、base¥demo¥smpldata¥accessディレクトリにインストールされています。

- テーブルを選択する。

  データベース名をダブルクリックすると、データベースに入っているテーブルがリストされます。このリストのテーブルをダブルクリックして、選択します。

- カラムを表示する。

  テーブルを選択すると、そのカラムが展開されます。カラムを選択するには、ダブルクリックします。カラムを選択すると、右側のクエリー子ウィンドウに*EXEC SQL*が追加されます。

- 問合せを選択する。

  カラムを選択したら、ツールバーの[クエリー実行]をクリックして、問合せを実行することができます。問合せ結果を見て、生成するアプリケーションがアクセスするデータを確認します。

注意：一次キーを少なくとも1つ選択しなければなりません。一次キーを選択しなかった場合、自動的に1つ選択されます。

このステップの画面を図 8-2 に示します。

図 8-2 クエリーの定義

この画面では、生成するアプリケーションで使用される問合せを構築およびテストします。この画面は、IDEの[OpenESQLアシスタントを表示]メニューから直接呼び出すことができます。

データベースへのアクセスとSQLの使い方の詳細は、*「データベースアクセス」*オンラインブックの*OpenESQL*の章を参照してください。

8.2.5 ステップ5：拡張

このステップでは、スクリーンセットに必要なDialog System拡張を使用できるようにするために、データブロック内にパラメータブロックをセットアップします。それぞれの拡張によって要件が異なります。

8.2.6 ステップ6：Dialog Systemのランタイム構成オプション

このオプションを選択すると、関連するダイアログコードが生成するスクリーンセットに挿入されます。

各オプションの詳細は、[ヘルプ]をクリックしてください

8.2.7 ステップ7：COBOLプログラムを生成する

[スケルトンCOBOLプログラムを生成]を選択すると、スケレトンCOBOLプログラムが生成されます。このプログラムは、dsgrunを呼び出すことによってアプリケーションを起動します。このプログラムにはデフォルト名が付いていますが、名前と場所はユーザーが指定できます。

このステップの画面を図 8-3 に示します。

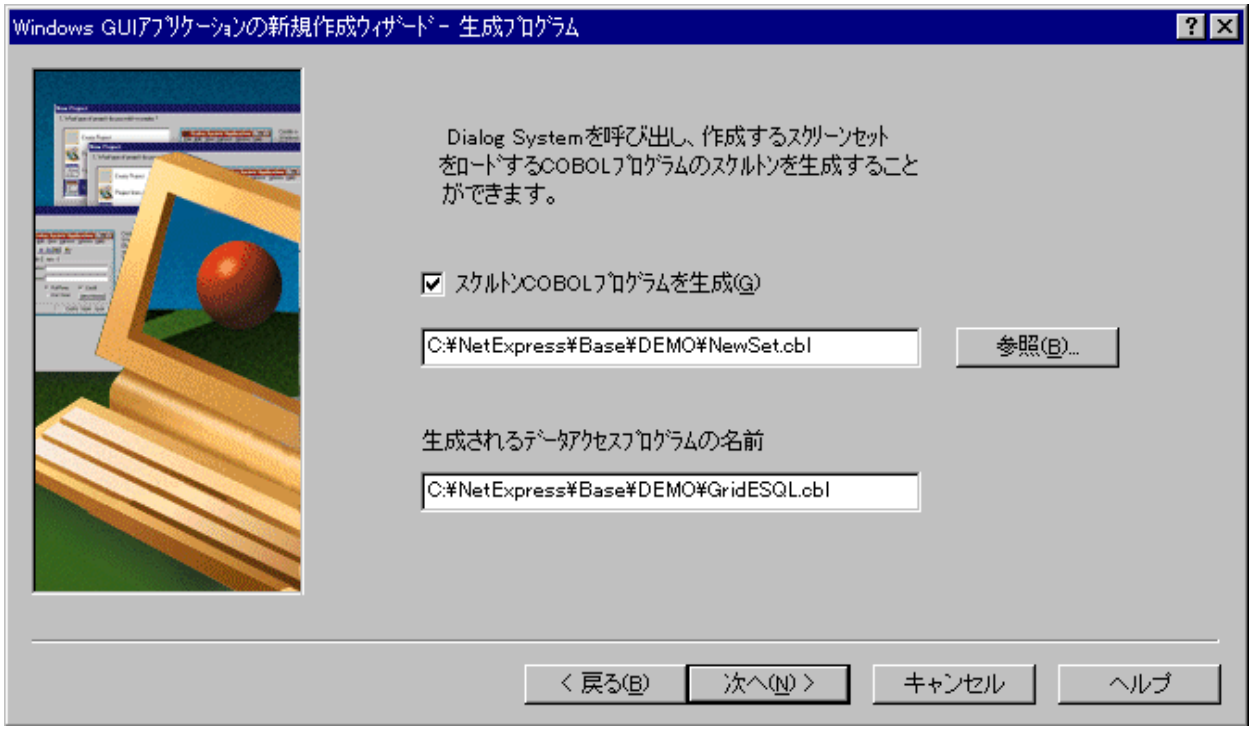


図 8-3 プログラムの生成

ステップ4で[OpenESQLデータアクセスを使用する]を選択した場合、[データアクセスプログラム名]の項目が表示されます。

8.2.8 ステップ8：選択したオプションの確認

このステップでは、これまでのステップで選択したオプションを確認します。前のステップに戻って、選択を変更することもできます。選択を受け入れると、プロジェクト、スクリーンセット、プログラムが作成されます。

8.3 ウィザードの出力

次のファイルがWindows GUIアプリケーションウィザードによって出力され、データアクセスオプションが選択されます。

- アプリケーションCOBOLプログラム（拡張子.cbl）
- スクリーンセット（拡張子.gs）

- ステータスバー（ファイル名sbar.cbl.）
- GridESQL.cbl。ステップ4で[グリッド]を選択した場合、グリッドコントロールを作成および管理します。

8.4 アプリケーションの実行

Windows GUIアプリケーションウィザードからの出力を使って、データベースの詳細を表示、問合せ、または変更するには、

1. IDEの[プロジェクト]メニューにある[すべてをリビルド]を選択して、プロジェクトを再構築します。
2. IDEのツールバーにある[実行]をクリックするか、[アニメート]メニューの[実行]を選択して、アプリケーションを実行します。

8.5 次の章の紹介

これでデータアクセスアプリケーションが生成されたので、データを操作することができます。詳しくは、次の章の*データアクセス*を参照してください。

第9章　データアクセス

前の章では、新しいWindows GUIアプリケーションウィザードによって、インストールされたデータベースに対するアクセスや問合せが簡単に行えることを示しました。この章では、データの操作方法について説明します。

注意：Windows GUIアプリケーションウィザードが出力するプログラムとファイルは、ユーザーが独自のアプリケーションを開発するための出発点となります。これらは、あらゆる状況に対応する万能のコードではありませんが、基本機能の使い方を習得することを目的としています。ユーザーは、提供されたコードを個々のニーズに適合させることができます。

---

*「入門書」のWindows GUIアプリケーションの作成*の章には、このウィザードの実際の使い方に関する説明があります。

9.1 Windows GUIアプリケーションウィザード

ウィザードプロセスは、新しいスクリーンセットを作成し、ユーザーが要求する機能に合わせて構成された関連COBOLプログラムを自動生成します。スクリーンセットと関連COBOLプログラムは、必要に応じてプロジェクトに自動的に追加されます。

インストールされたデータベースにアクセスするには、ステップ3:「スクリーンセット機能を選択する」でOpenESQLデータアクセスオプションを選んでおかなければなりません。

9.2 インストールされたデータベースへのアクセス

アプリケーションを構築したあとに問合せの結果を見るには、Windows GUIアプリケーションウィザードのステップ4で選択したオプションによって2種類の方法があります。

- リストボックスビュー

  または

- グリッドビュー

これらのビューは機能的には同じですが、インタフェースが異なります。グリッドビューは直観的、ストビューはWindowsの上級ユーザーに適しています。

グリッドビューで問合せを実行すると、すべてのデータがメモリに読み込まれます。このためデータをすばやく上下にスクロールできます。リストボックスビューでは、データの一部分だけがメモリに読み込まれます。実際には、リ

ストボックスの1画面分のデータだけが読み込まれます。したがって、大量のデータを使用している場合、リストボックスを使用する方が実用的です。非常に大きなアプリケーションで性能を向上する方法については、ヘルプの*性能チューニング*トピックを参照してください。

データベース問合せのグリッドビューの例を図 9-1 に示します。

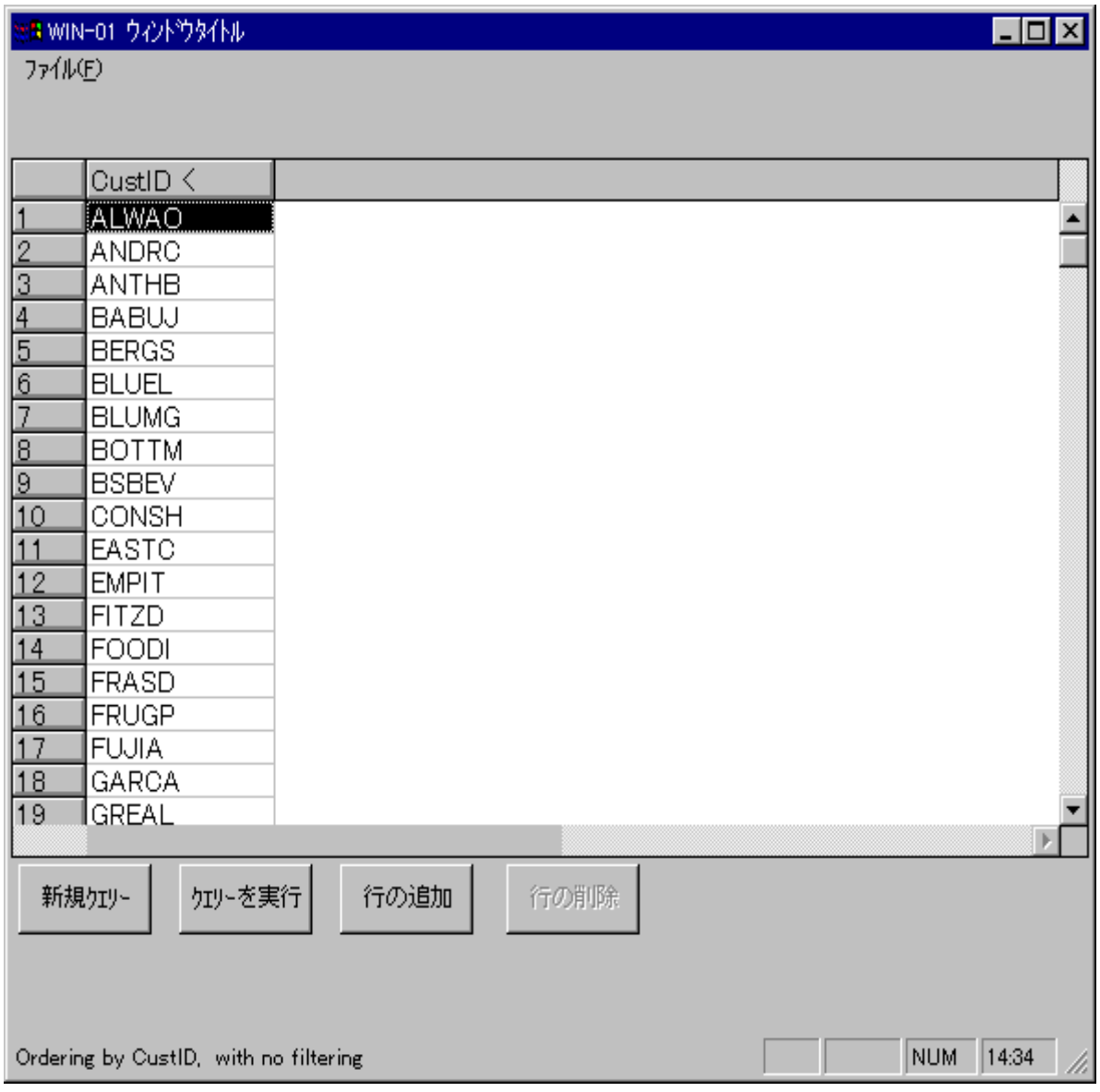


図 9-1 データベース問合せのグリッドビュー

データベースのリストボックスビューの例を図 9-2 に示します。

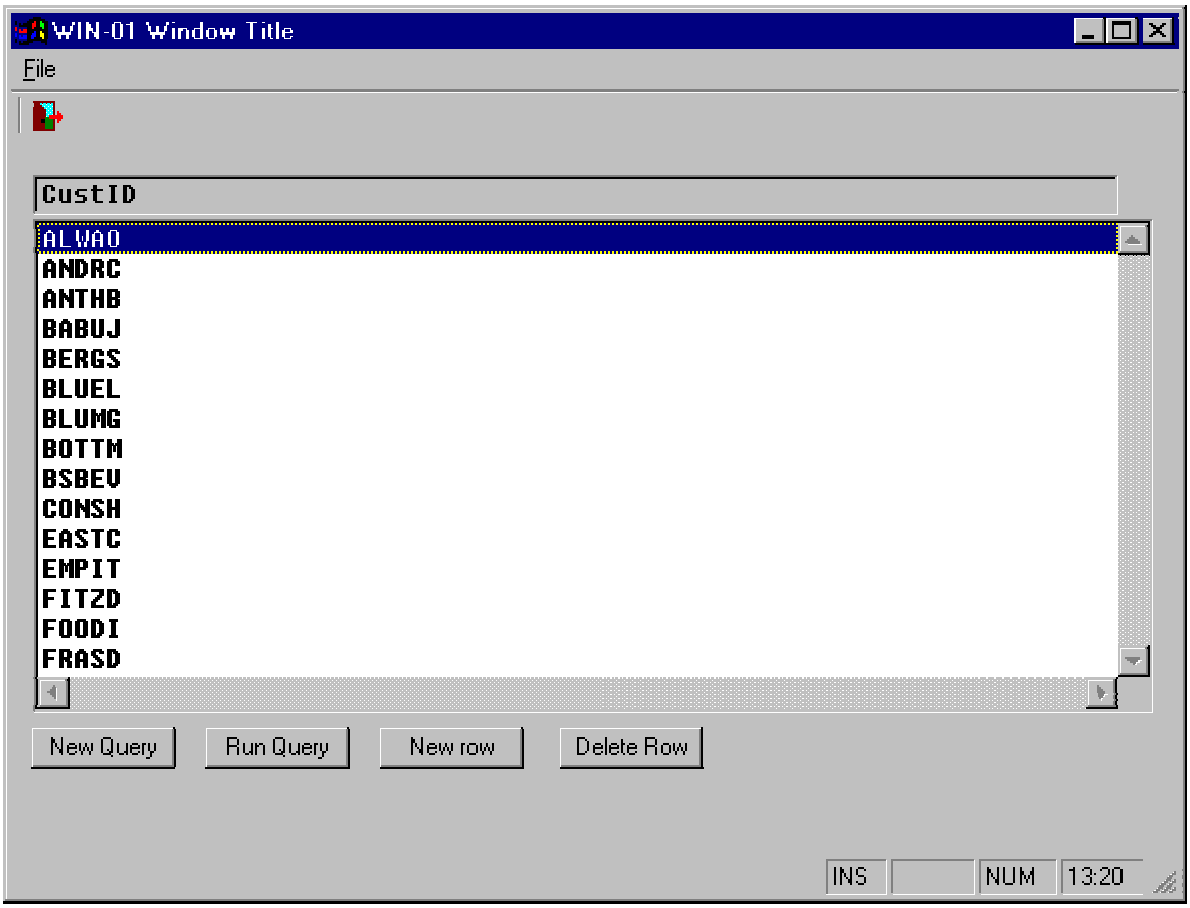


図 9-2 データベースのリストボックスビュー

画面の一番下のステータスバーには、マウスの下にあるオブジェクトに関する情報が表示されます。マウスをほかの部分に移動すると、このテキスト（実際にはメッセージボックス）には、どんな種類の機能が選択されたかが表示されます。メッセージボックスには、キーフィールドを編集しようとしたなど、そのときに起こったエラーに関する情報も表示されます。

9.3 データの操作

次の操作を実行できます。

- データベース内のデータを編集する。
- 新しい行を挿入する。
- 行を削除する。
- 新しい問合せを定義して、データを検索する。
- カラムによってデータをソートする。

9.3.1 データを編集する

	グリッドビュー	リストビュー
フィールドのデータを変更する	フィールドを選択し、変更します。キーフィールドは変えられません。ソートされたカラムのフィールドを編集すると、行全体が別の部分に移動することがあります。	変更する行を選択し、ダイアログボックスの中で変更します。ソートされたカラムのフィールドを編集すると、行全体が別の部分に移動することがあります。
次のカラムへ移動する	マウスまたは左右の矢印キーを使用します。	
フィールドのデータを削除する	フィールドを選択し、[削除]キーを押します。	変更する行を選択し、ダイアログボックスの中で変更します。ソートされたカラムのフィールドを編集すると、行全体が別の部分に移動することがあります。
画面をリフレッシュする	[クエリー実行]をクリックします。	[クエリー実行]をクリックします。

注意：1つの問合せを一度に複数開くことができます。この場合、Dialog Systemは問合せをディスクからリフレッシュしてから、データに対する編集をエコーします。これにより常に最新バージョンのデータが表示されます。

9.3.2 新しい行を挿入する

	グリッドビュー	リストビュー
新しい行を挿入する	挿入点の下の行を強調表示します。[行の追加]をクリックすると、空のグリッドが表示され、一番上の行に新しいデータを入力できます。	挿入点の下の行を強調表示します。[行の追加]をクリックすると、キーフィールドを除く各カラムについて入力フィールドを持つダイアログボックスが表示されます。どの入力フィールドについても、確認は行われません。
挿入をアクティブにする	F7を押します。更新結果がステータスバーに表示されます。	Enterを押します。更新結果がステータスバーに表示されます。

問合せの新しい行を見る	[クエリー実行]をクリックします。	

注意：新しい行がすぐには表示されないことがあります。これには、次の理由が考えられます。

- このデータが現在の問合せによって排除されている。新しい行を見るには、新しい問合せを選択する必要があります。
- 新しい行が現在ビューの外部にある。新しい行を見るには、その行の位置までスクロールする必要があります。

9.3.3 行を削除する

グリッドビューまたはリストビューのどちらでも、データ行を削除するには、データの横にある行番号をクリックし、[行の削除]をクリックします。

9.3.4 データを検索する

	グリッドビュー	リストビュー
データを検索する	1. [クエリーの新規作成]をクリックします。 2. 検索したいフィールドを強調表示します。 3. 検索基準を入力します。 4. フィールドを右クリックします。ポップアップメニューが表示されます。 5. 論理演算子（=、>、<、!=）を選択します。デフォルトは「等しい（=）」です。 6. [クエリー実行]をクリックします。検索結果が表示されます。	1. [クエリーの新規作成]をクリックします。 アプリケーションの各フィールドに関する項目を持つダイアログボックスが表示されます。 2. 検索基準を入力します。 3. フィールドを右クリックします。ポップアップメニューが表示されます。 4. 論理演算子（=、>、<、!=）を選択します。デフォルトは「等しい（=）」です。 5. [クエリー実行]をクリックします。検索結果が表示されます。

9.3.5 データをソートする

	グリッドビュー	リストビュー
カラム内のデータをソートする	カラム見出しバーをクリックします。次の項目が表示されます。 • データが昇順にソートされることを示す見出しバーの上矢印 • データが降順にソートされることを示す見出しバーの下矢印	リストボックスを右クリックします。ソートするカラムを選ぶためのポップアップメニューが表示されます。

第10章 独自コントロールのプログラミング

この章では、ユーザーコントロールやActiveXコントロールをDialog Systemアプリケーションで利用できるようにする方法について説明します。次のトピックを取り上げます。

- コントロールプログラム

  この章では、*コントロールプログラム*という用語を使用します。これは、ユーザーインタフェースのオブジェクトまたはコントロールを作成したり、操作するためのCOBOLソースプログラムです。

- ActiveXコントロール
- ユーザーコントロール

10.1 コントロールプログラム

コントロールプログラムを使うと、任意のオブジェクトをそれ対応するプログラム機能によって操作できます。各コントロールに対して、次のようにして関連するデータのリフレッシュ、削除、更新などの操作を実行できます。

- CALL-FUNCTIONの値を設定する。
- FUNCTION-DATAデータブロックグループに他のパラメータを設定する。
- コントロールプログラムに対してCALLOUTを使用する。

Dialog Systemには、次のコントロールをアプリケーションに適合するためのプログラムがあります。

- GUIクラスライブラリWin32コントロール
- ActiveXコントロール

各コントロールのソースコードは、できるだけ汎用性を持つように設計されています。したがって、これらのコードは、最小限の変更で実際のアプリケーションに適合させたり、再利用することができます。しかし、自由度の大きさはそれぞれのコントロールによって異なります。

- ステータスバーコントロールととスピンボタンコントロールは、変更なしで自由に利用できます。
- ツールバーコントロールとビューコントロールは変更なしで利用できますが、個々の要件に合わせてプログラムへのデータインタフェースを変える必要があります。
- ActiveXコントロールはインプリメントに依存します。

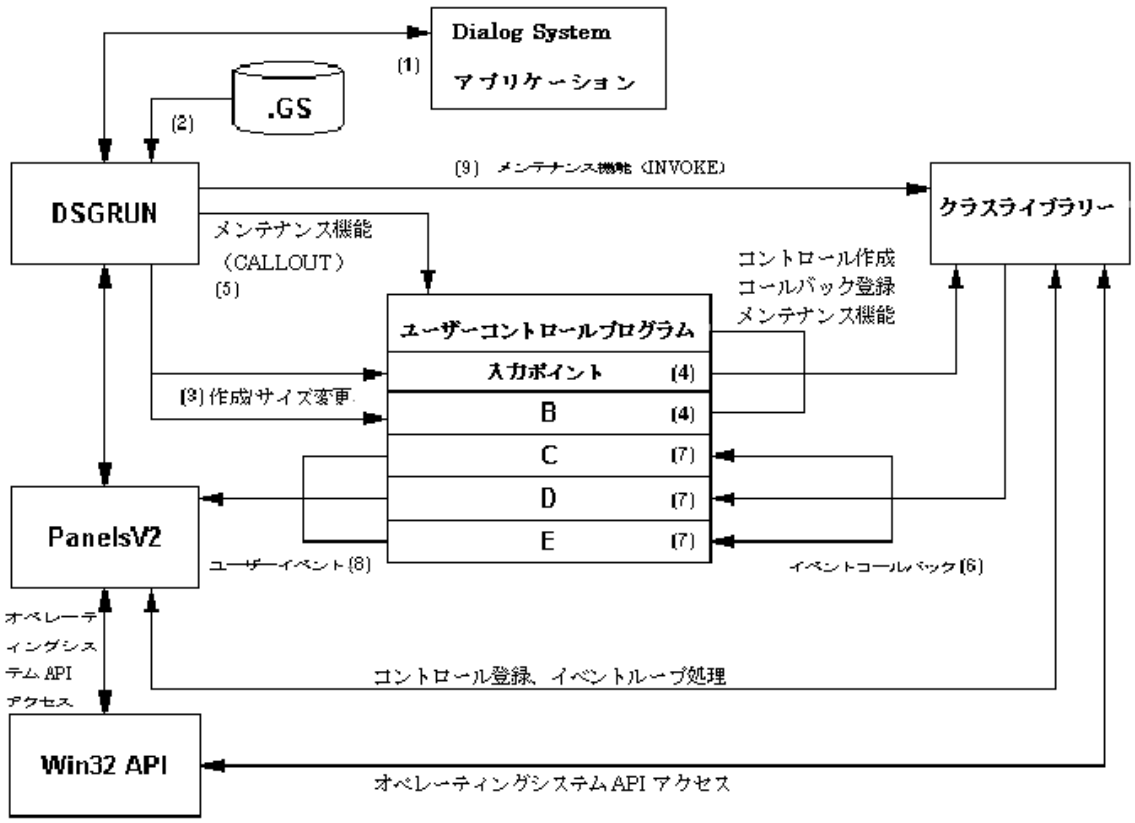
  これらのバージョンを個々の要件に合わせて適合する必要があります。しかし、それぞれのコントロールには、変更の必要がない汎用コードがあります。プログラムの修正については、*ユーザーコントロール*の節を

参照してください。

コントロールプログラムは、NetExpressクラスライブラリの呼び出しを使ってインプリメントされています。

10.1.1 ユーザーコントロールのインプリメントアーキテクチャ

Dialog Systemおよびクラスライブラリを使ってユーザーコントロールをインプリメントするためのアーキテクチャを次の図に示します。



ユーザーコントロールを作成する場合、

1. ウィンドウ上のユーザーコントロールをスクリーンセット内でペイントし、コントロールにマスタフィールドとプログラム名を関連付けます。
2. アプリケーションプログラムは dsgrun を呼び出し、スクリーンセットを通常の方法で提供します。
3. ユーザーコントロールが定義されたウィンドウをDialog Systemが作成すると、そのユーザーコントロールに関連付けられたプログラム内の入口点が呼び出されます。
4. 入口点のコードは、クラスライブラリがウィンドウオブジェクトを作成するのに必要なすべてのタスクを実行します。

ウィンドウが完全に作成されると、スクリーンセットダイアログは、コールバック登録を行うために、ユーザーコントロールプログラムに対してCALLOUTを実行します。これによって、プログラムの入口点が定義されたシステムイベントのオカレンスと関連付けられます。

5. 定義されたシステムイベントが起こると、指定された入口点のコードが実行されます。

6. 入口点のコードは、データブロックの更新を含む、必要なすべての処理を実行します。

7. 入口点のコードは、もとのPanels V2プログラムにメッセージを渡します。このメッセージは、Dialog Systemのスクリーンセット内でUSER-EVENTとして受け取られ、追加のダイアログ処理を実行することができます。

8. コントロールは、ユーザーコントロールプログラムに対するCALLOUTによって操作するか、Dialog SystemのINVOKE機能を通じてクラスライブラリによって直接操作することができます。

10.2 ActiveXコントロール

*コントロールオブジェクト*の章で述べたように、ActiveXコントロールはサードパーティベンダによって適用されており、Dialog System内に統合化できます。インタフェースの中でActiveXコントロールを使用する場合、個々の要件に合わせてカスタマイズする必要があります。

10.2.1 ActiveXコントロールの属性

ActiveXには、次の3種類の設計時属性があります。

- Dialog System属性

  ActiveXコントロールを選択するときに定義しなければならない属性です。

  [編集]メニューから[属性]を選択するか、コントロールをダブルクリックします。

- 一般属性リスト

  スクリーンセットからActiveXコントロールを選択すると、自動的に表示されます。

- ActiveX属性ページ

  ActiveXコントロールを右クリックします。これには、上の一般属性リストにある一般属性も含まれます。すべてのActiveXコントロールが属性シートを持っているわけではありません。

個々のアプリケーションにとって、デフォルト属性は不十分な場合がほとんどです。たとえば、スプレッドシートActiveXコントロールのデフォルト属性は2列×2行です。一方、Dialog SystemのクロックActiveXコントロールのようにデフォルトのままで十分なものもあります。

どの属性を設定すべきかを判断するには、ActiveXコントロールに関するドキュメントを参照してください。変更し

た属性詳細はスクリーンセットに格納されます。

10.2.2 ActiveXコントロールのカスタマイズ

ActiveXコントロールは、次の手順でカスタマイズします。

- 必要なActiveXコントロールを選択する。
- 対応するDialog System属性を定義する。
- プログラミングアシスタントを使ってActiveXコントロールをカスタマイズする。

10.2.2.1 ActiveXコントロールの選択

Dialog Systemのメニューから ActiveXコントロールを直接指定するには、

1. [ファイル]メニューから[インポート]を選択します。
2. サブメニューからActiveXコントロールを選択します。
3. システムに登録されたActiveXコントロールを示すリストボックスから、求めるActiveXコントロールを選択します。

   選択したコントロールを表すアイコンがActiveXツールバーに表示されます。このActiveXコントロールを再び使用したいときは、いつでもActiveXツールバーから直接選択できます。
4. ActiveXコントロールを関連付けるOBJ-REFと定義された項目がデータブロックにあるかを確認します。
5. ActiveXコントロールの位置とサイズを調整します。

   図 10-1 に示すようなActiveXコントロールの属性ダイアログボックスが自動的に表示されます。

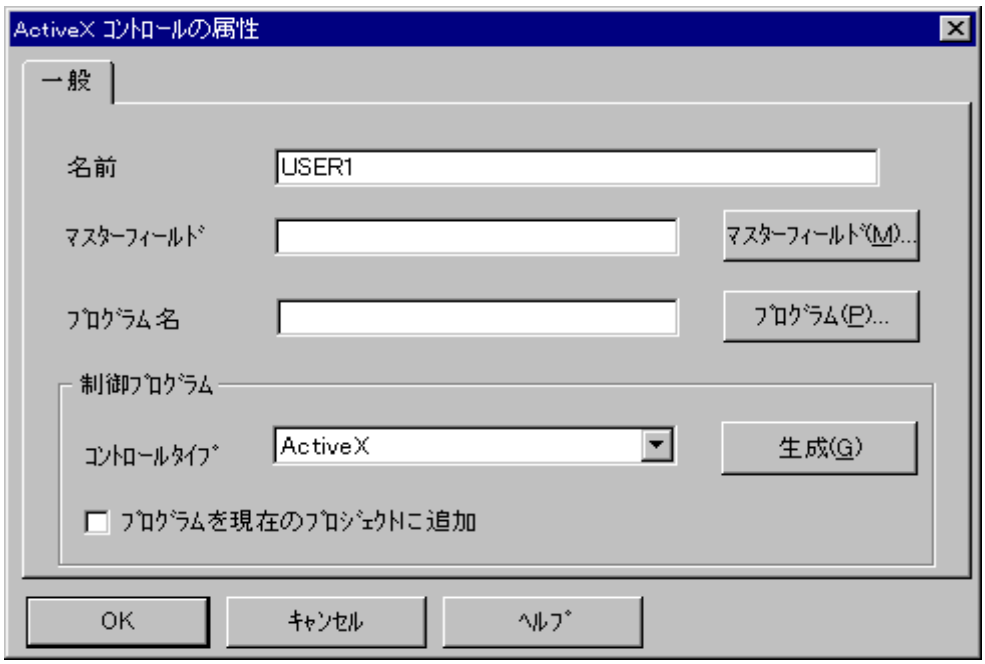


図 10-1 ActiveXコントロールの属性ダイアログボックス

10.2.2.2 ActiveXコントロールのDialog System属性の定義

ActiveXコントロールの属性ダイアログボックスで、

1. このコントロールに関するデータブロックマスタフィールドのOBJ-REF名を指定します。
2. コントロールプログラムの名前を指定します。
3. NetExpressプロジェクトが開いており、NetExpress IDEで利用可能なことを確認します。
4. [プログラムを現在のプロジェクトに追加]を選択します。
5. [生成]をクリックします。COBOLソースプログラムが生成されます。このソースプログラムは、使用しているコントロールに合わせて自動的にカスタマイズされており、開いたプロジェクトに追加されます。

   これが、スクリーンセットにあるActiveXコントロールのためのコントロールプログラムになります。
6. OKをクリックします。

10.2.2.3 プログラミングアシスタントを使ったActiveXコントロールプログラムのカスタマイズ

カスタマイズされたActiveXコントロールプログラムは、DialogSystem¥demo¥custgridディレクトリにあります。デモンストレーションに関するヘルプも用意されています。

ActiveXコントロールに合わせたコントロールプログラムのカスタマイズでは、次のことが行われます。

- ActiveXコントロールがトリガーするイベントのためのイベントハンドラを登録します。
- 必要なイベントを取り扱うためのコードをインプリメントします。
- ランタイム属性を設定し、ActiveXコントロールでメソッドを呼び出すためのコードを追加します。

ActiveXコントロールをカスタマイズするには、それがサポートするメソッド、属性、イベントを調べる必要があります。この作業を簡単にするために、プログラミングアシスタントにはプログラムに必要な定義済み機能を抽出するためのビジュアル環境が用意されています。プログラミングアシスタントは、次の処理を実行するためのコードを提供します。

- 属性を獲得または設定する。

  ActiveXコントロールは属性または属性（例、背景色）を持っています。これらは、読み取ったり、変更することができます。

  - 属性の現在状態を見つけるには、GET **属性名**文を使用します。
  - 属性を変更するには、SET **属性名**文を使用します。

- メソッドを呼び出す。

  メソッドは、ActiveXコントロールによって提供される定義済み機能です。この機能は、コントロールに対して何らかのアクションを実行するように指示します。そのとき、パラメータを送るか、値を返すか、またはその両方を行うことができます。

- イベントに対して応答する。

  イベントは、アクションが起こったことを示すためにActiveXコントロールによって提供されます。イベントは、イベントの詳細を記述したパラメータ情報を送ることができます。この情報は、ActiveXコントロール使用しているパラメータプログラムによって受け取られ、認識されます。

  コールバックを登録すると、コントロール上に関するイベントが起こったときにCOBOLコードのブロックが実行されます。たとえば、+/-ボタンをクリックして項目を畳むと、`MessageName`に詳しく記述された呼び出しがプログラムの入口点に対して行われます。入口点のコードは、コントロールのルートソースファイルに抜けます。

  ```
  MOVE ProgramID & z"Collapsed " TO MessageName
  ```

  EntryCallbackクラスの"新しい"メソッドを呼び出すには、次のコードを入力します。このメソッドは、実行するEntryコードブロックを詳しく記述し、EntryCallbackクラスのインスタンスに関するオブジェクト参照を返します。

  ```
  INVOKE EntryCallback "new" USING MessageName
  ```

```
RETURNING aCallback
```

EntryCallbackインスタンスが応答すべきイベントを確立します。

```
MOVE p2ce-itemcollapsed TO i
```

コールバックが起こるイベントを登録するために、コントロールインスタンスオブジェクト（そのsetEventメソッド）を呼び出します。2番目のパラメータは、まえに確立したEntryコールバックオブジェクト参照を指定します。

```
INVOKE aTreeView "setEvent" USING i aCallback
```

Entryコールバックオブジェクトは必要ないので破壊します。

```
INVOKE aCallback "Finalize" RETURNING aCallback
```

10.2.2.4 **プログラミングアシスタントの起動**

プログラミングアシスタントにアクセスするには、

1. ActiveXコントロールを右クリックします。
2. コンテキストメニューから[プログラミングアシスタント...]を選択します。

   図 10-2 に示すActiveX プログラミングアシスタントダイアログボックスが表示されます。

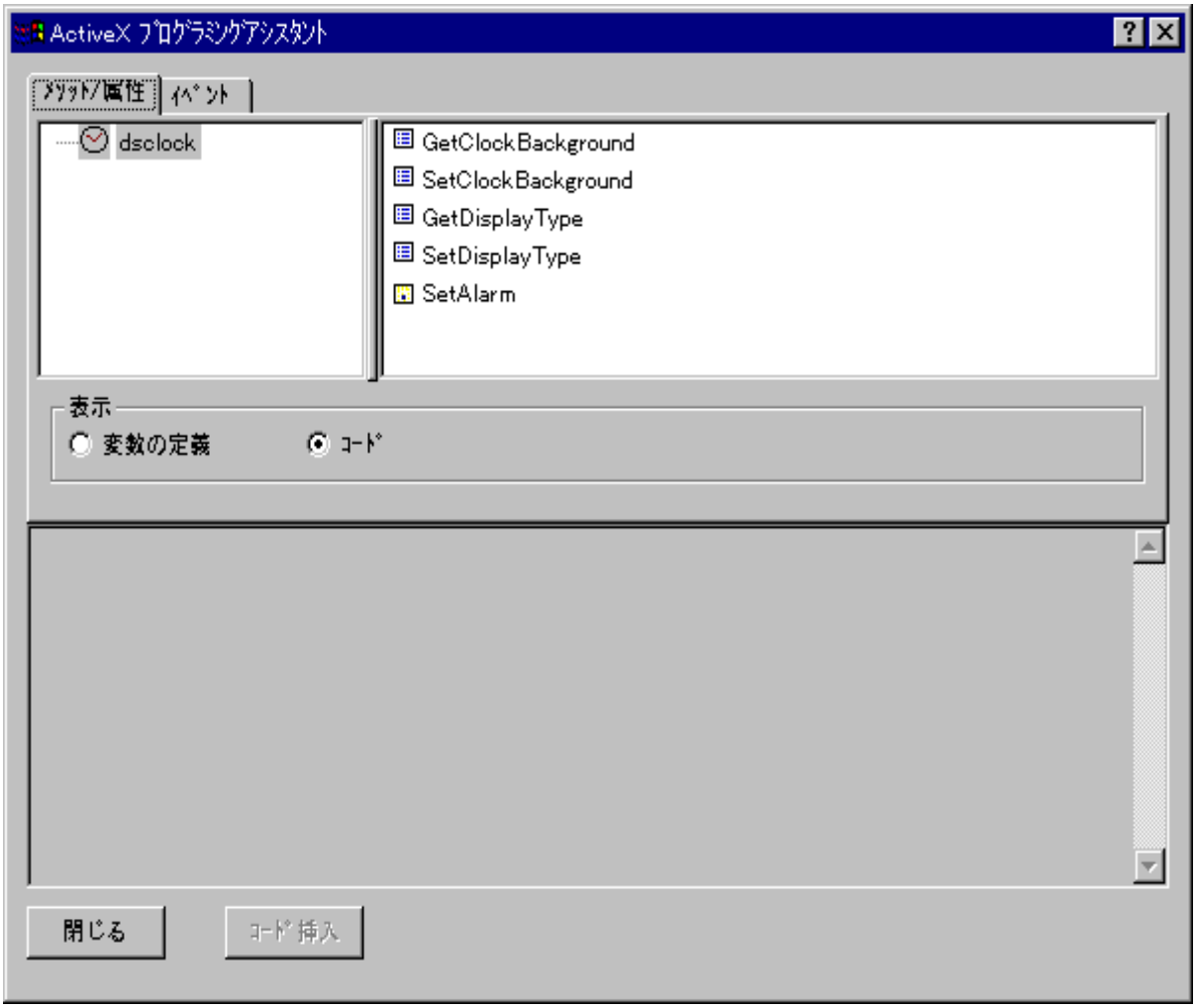


図 10-2 ActiveX プログラミングアシスタントダイアログボックスーメソッド/属性

このダイアログボックスには、2つのタブページがあり、メソッドと属性またはイベントに関連したコードを開いているCOBOLプログラムに直接挿入できます。

10.2.2.4.1 メソッドと属性

[メソッド/属性]タブを選択すると、上の図 10-2 に示すような画面が表示されます。

左側の子ウィンドウには、ActiveXコントロールとそのランタイムオブジェクト階層が表示されます。右側の子ウィンドウには、コントロールが提供するメソッドと属性が表示されます。

右側の子ウィンドウでメソッドまたは属性を選択すると、それに関連するCOBOLコードがダイアログボックスの下半分の子ウィンドウに表示されます。

このコードをActiveXを作成するプログラムに挿入するには、

1. NetExpress IDEでプログラムを開きます。
2. プログラムの新しいコードを挿入する位置にカーソルを移動します。
   このコードを挿入する位置は明示されていません。これは、メソッドや属性の挿入がユーザーのプログラムロジックに完全に依存するからです。コードは、コントロールの初期設定、イベントのコールバック、また

は単なるランタイム操作のコンテキストで使用できます。

3. プログラミングアシスタントウィンドウに戻ります。
4. [コード挿入]をクリックします。
   作成されたコードには、選択した機能に必要なすべてのパラメータと戻り値が入っています。

10.2.2.4.2 サブオブジェクト

ActiveXコントロールは、関連するサブオブジェクトを持つことができます。各サブオブジェクトは自身のメソッドや属性を持っており、それらは表示または編集できます。メソッドや属性を表示するには、左側の子ウィンドウでActiveXコントロールにつながっている+符号をクリックします。
一般にサブオブジェクトは、関連するイベントを持っていません。サブオブジェクトは、ActiveXコントロール内でグローバルに取り扱われます。

10.2.2.4.3 イベント

[イベント]タブを選択すると、図 10-3 に示すような画面が表示されます。

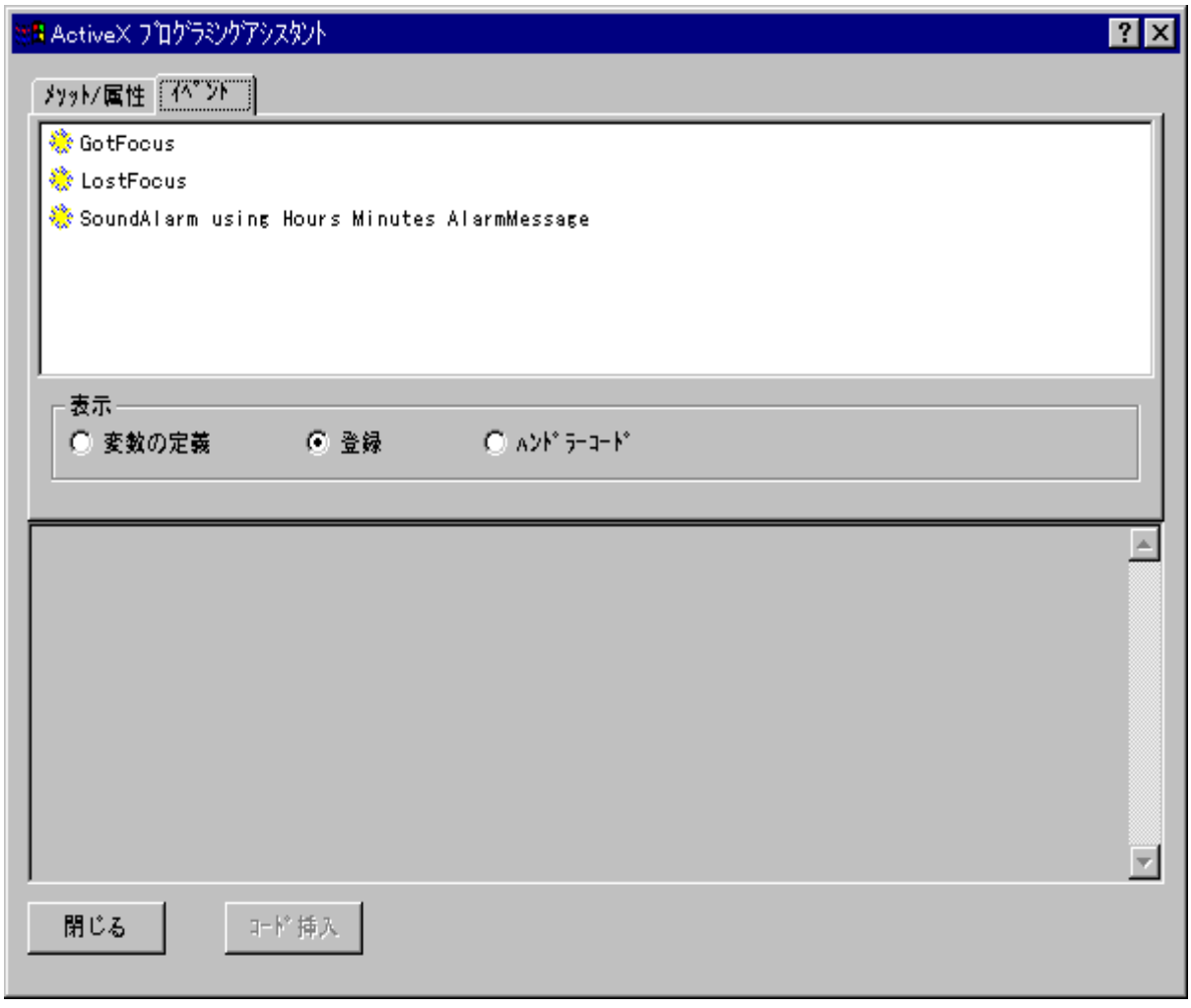


図 10-3 ActiveX プログラミングアシスタントダイアログボックスーイベント

ダイアログボックスの上半分には、イベント名がリストされます。各イベントの横には、そのイベントがActiveXコントロールプログラムに渡すパラメータが表示されることがあります。

メソッド/属性ビューと同様に、下半分には、上半分で選択されたイベントに関連したコードが表示されます。ラジオボタンには次のような機能があります。

10.2.2.4.4 **変数**

[変数の定義]ボタンを選択すると、上の子ウィンドウで選択したイベントと接続された変数が表示されます。変数は、

- ActiveXコントロールによって渡されるパラメータタイプと一致しています。

  パラメータは、イベントハンドラコードで定義された変数に取り込んでからでないと、コントロールプログラムからアクセスできません。

- アプリケーションプログラムのLocal-Storage Sectionに挿入しなければなりません。

10.2.2.4.5 **イベントの登録**

[登録]ボタンを選択すると、選択されたイベントに関するコールバックをコントロールプログラムが登録するのに必要なコードが表示されます。イベントコールバックを登録すると、2つのタスクが実行されます。

- イベントハンドラコードがある入口点を指定します。
- OLEクラスファイルに対してイベントを認識する必要があることを知らせます。

Program ID+On+イベント名という形式の入口点名が生成されます。たとえば、イベント名がBeforeDeleteRowの場合、Program IDOnBeforeDeleteRowという入口点名が生成されます。

注意：複数のコントロールプログラム間で入口点の一意性を保証するために、生成される入口名の前にはプログラムIDが付きます。たとえば、複数のActiveXコントロールの間で同じイベント名を使用することがあります。入口点にプログラムIDが付いていないと、OLEクラスライブラリはどのコントロールプログラムを呼び出すべきか判断できません。

このコードは、コントロールの作成時に実行されるコントロールプログラムのRegister-Callbacks Sectionに挿入する必要があります。

10.2.2.4.6 **イベントハンドラ**

[ハンドラーコード]ボタンを選択すると、

- イベントハンドリングコードが、ダイアログボックスの下の子ウィンドウに表示されます。

- [注釈を含める]チェックボックスが選択されます。

  このチェックボックスが選択された場合、表示されるコードには次のイベントが入っています。

  - 一般的なプログラム指示のための注釈
  - 変更可能な各パラメータに関するコメントアウトされた命令。これらはパラメータの変更方法を示しています。

イベントハンドリングコードは、イベントハンドラが*Addという注釈をコーディングした場所に挿入します。生成されるコントロールプログラムにサンプルコードがあります。

このコードを挿入するには、

1. NetExpress IDEでプログラムを開きます。
2. 新しいコードを挿入する位置をクリックします。

   このコードを挿入する場所は明示されます。コードは、コントロールの初期設定、イベントのコールバック、または単なるランタイム操作のコンテキストで使用できます。
3. プログラミングアシスタントウィンドウに戻ります。
4. [コード挿入]をクリックします。

   作成されたコードには、選択した機能に必要なすべてのパラメータと戻り値が入っています。

10.2.2.5 まとめ

ActiveXプログラミングアシスタントによって、ActiveXコントロールに親しみ、必要なコーディングを作成するための工数を劇的に削減することができます。

10.3 ユーザーコントロール

インタフェースの中でユーザーコントロールを使用する場合、個々の要件に合わせてカスタマイズする必要があります。ユーザーコントロールは次の手順でカスタマイズします。

- ユーザーコントロールを作成し、そのDialog System属性を定義する。
- 求めるコントロールタイプに合ったコントロールプログラムを生成する。
- コントロールプログラムをカスタマイズする。

10.3.1 ユーザーコントロールの作成

ユーザーコントロールオブジェクトを使って、NetExpressクラスライブラリの GUIオブジェクトをスクリーンセット

で利用することができます。

ユーザーコントロールは以下のようにして作成します。

1. コントロールを関連付けるタイプOBJ-REFのデータブロック項目を定義します。
2. ユーザーコントロールを追加したいウィンドウを選択します。
3. [オブジェクト]メニューから[ユーザーコントロール]を選択するか、[オブジェクト]ツールバーにあるユーザーコントロールをクリックします。
4. ユーザーコントロールの位置とサイズを調整します。

   図 10-4 に示すようなユーザーコントロールダイアログボックスが表示されます。

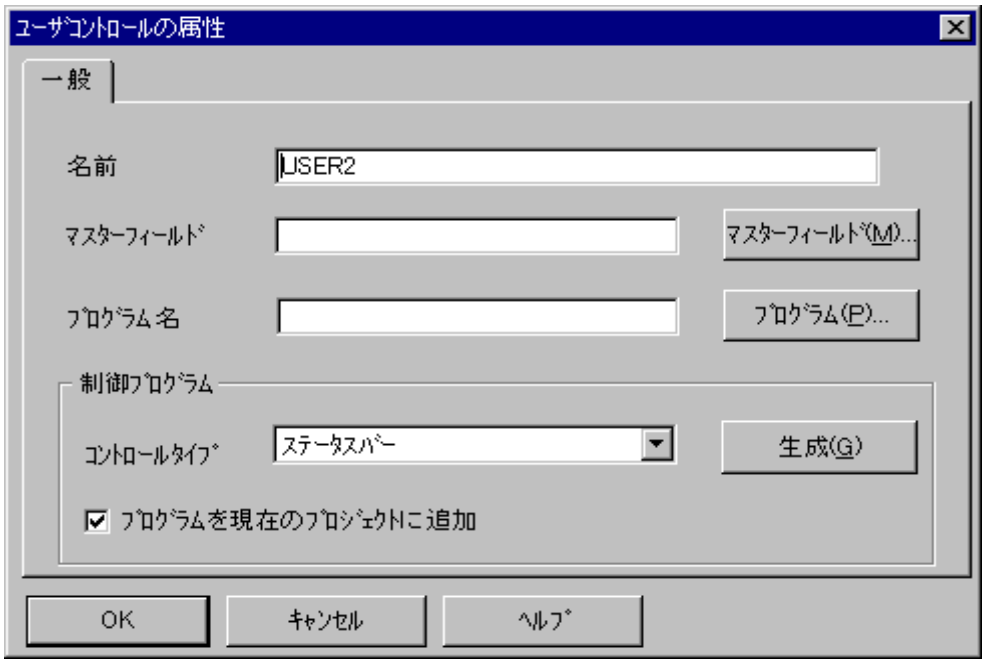


図 10-4 ユーザーコントロールの属性ダイアログボックス

5. ユーザーコントロールの属性ダイアログボックスの次の項目について情報を入力します。

   1. ユーザーコントロールの名前を指定します。MAIN-WINDOW-STATUS-BARのようにできるだけわかりやすい名前にします。
   2. マスタフィールド名として、関連するOBJ-REFデータ項目を指定します。
   3. ユーザーコントロールプログラムの名前を指定します。

      NetExpressクラスライブラリで定義されたクラスプログラムとは異なる名前を指定する必要があります。カスタマイズするコントロールプログラムに新しい名前を付けたら、必要な機能を持つよう

に変更することができます。

4. ドロップダウンリストから必要なタイプのコントロールを選択します。
5. NetExpressプロジェクトが開いており、NetExpress IDEで利用可能なことを確認します。
6. [プログラムを現在のプロジェクトに追加]を選択します。
7. [生成]をクリックします。COBOLソースプログラムが生成されます。このソースプログラムは、使用しているコントロールに合わせて自動的にカスタマイズされており、開いたプロジェクトに追加されます。

   これが、スクリーンセットにあるユーザーコントロールのためのコントロールプログラムになります。
8. OKをクリックします。

10.3.2 ユーザーコントロールタイプ

どんな場合も、DialogSystem¥Sourceディレクトリにあるfuncdata.impファイルにリストされたデータブロック定義をインポートする必要があります。Dialog Systemの「ファイル」/「インポート」/「スクリーンセット」メニュー項目を選択し、ダイアログボックスが表示されたら、インポートするファイルとしてfuncdata.impを選びます。

次の各項では、生成されたプログラムをスクリーンセットといっしょに使用するための変更について詳しく説明します。

10.3.2.1 スピンボタン

COBOLコードの変更は必要ありません。生成されたプログラムによって通知されるイベントに対して応答するために、スピンボタンの親ウィンドウにUSER-EVENTダイアログイベントをインプリメントするだけです。これによって、ダイアログコードの中では、データブロックマスタフィールドがコントロールプログラムによって渡される新しい値に更新されるようになります。

例:

```
USER-EVENT
    XIF=$EVENT-DATA 34580 UPDATE-MASTER
    UPDATE-MASTER
    * ユーザーコントロールプログラムはNUMERIC-VALUEフィールド
    * を更新する。このコードは要求されたフィールドを更新する
    MOVE NUMERIC-VALUE(1) MY-NUMERIC-FIELD
```

10.3.2.2 **ステータスバー**

COBOLコードの変更は必要ありません。次の要素をインプリメントするだけです。これらについては、本書の後の章で詳しく取り上げます。

- ヒントテキストステータスバーセクションに関するMOUSE-OVERイベントを設定するためのダイアログコード。これについてはヘルプに解説されています、
- Window-movedおよびwindow-sizedイベントに対して応答するダイアログイベント。これらは、ステータスバーのサイズを変更するためのコントロールプログラムをCALLOUTします。
- トグルキー（Insert/Overstrike、CAPS、Num Lock）の状態と現在のシステム時刻をリフレッシュするために、通常のCALLOUTを使用可能にするTIMEOUT手続き。

この手続きについては、*チュートリアル―ステータスバーの追加とカスタマイズ*で詳しく取り上げます。.

10.3.2.3 **ツリービュー**

生成されたツリービューコントロールプログラムを使用する場合、まず、tviewdata.impファイルをインポートして、スクリーンセットのデータブロックの必要な項目にデータをいれます。

ツリービューコントロールプログラムを使用するには、Created Treeコントロールに挿入したいデータをATVIEW-PARMSデータブロックグループにいれます。詳しくはヘルプを参照してください。

生成されたプログラムには、コールバック登録とイベントハンドラの例が入っています。これらの例は個々のニーズに合わせてカスタマイズできます。

10.3.2.4 **ツールバー**

生成されたツールバーコントロールプログラムを使用する場合、まず、tbardata.impライブラリをインポートして、スクリーンセットのデータブロックの必要な項目にデータをいれます。

ツールバーコントロールプログラムを使用するには、プログラムが使用するメニューとツールバーボタンの定義をカスタマイズします。これには、使用される構造を定義するWORKING-STORAGE COPYFILEを編集します。

つぎにユーザーによるメニュー/ボタン項目の選択に対して応答し、スクリーンセット内にすでに定義されている既存のメニュー選択項目ダイアログの実行を含む、必要な動作を実行するためのコードを作成します。

10.3.2.5 **ユーザー定義**

ユーザーコントロールオブジェクトを作成するとき、任意のコントロールを作成するための構造を提供する汎用スケレトンコントロールプログラムを生成できます。

10.3.3 まとめ

生成されたプログラムは、変更なしに正常にコンパイルおよび実行できます。つぎにコードを変更して、アプリケーションで必要な追加機能を実行したり、前に見たように生成されたコードをそのまま使用することもできます。

最終的な目的は、同じ手続きCOBOLインタフェースを通じて、すべてのクラスライブラリコントロールへのアクセスを提供するフルセットのコントロールプログラムを提供することです。このプロセスは、NetExpressのDialog Systemの将来リリースで完成する予定です。

第11章 複数のスクリーンセット

この章では、Dialog Systemで複数のプログラムやスクリーンセットを使用する方法について説明します。ここでは、Dsrunnerとコールインタフェースを使ったアプリケーションにおける複数のスクリーンセットの管理を取り上げます。

複数のスクリーンセットを使用する場合、Dialog Systemを呼び出すには次の2つの方法があります。

- Dsrunnerを使用する。

  この方法では、特に複数のマルチスクリーンセット／マルチモジュールのDialog Systemアプリケーションを開発している場合、Dialog Systemを最も速く簡単に呼び出すことができます。

- Routerによってコールインタフェースを使用する（具体的方法については、この章の後の方を参照）。

11.1 Dsrunner

ほとんどのアプリケーションは、Dsrunnerを使用できるはずです。これは、Dialog Systemを呼び出すための最も簡単な方法です。Dsrunnerによって、複数のスクリーンセットと複数のサブプログラムモジュールを持つモジュール形式のDialog Systemアプリケーションを開発できます。これにより、ユーザーはビジネスロジックとサポートスクリーンセットに集中でき、Dialog Systemの呼び出しに関する詳細を気にする必要がありません。

Dsrunnerは、メインウィンドウのテンプレートとなるDsrunnerスクリーンセットをロードします。これをもとに、他のスクリーンセットを起動することができます。Dsrunnerは、必要に応じて任意のサブプログラムやスクリーンセットの切り換えを取り扱います。このためユーザーは、複数のスクリーンセットまたは複数のサブプログラムを切り換えるためのコードを用意する必要がありません。

11.1.1 Dsrunnerのアーキテクチャ

Dsrunnerは、アプリケーションをサブプログラムとして実行するプログラムです。したがって、Dsrunnerを使って、スクリーンセットのロードと関連するサブプログラムの呼び出しを伴うスクリーンセットを起動します。このスクリーンセットからさらに別のスクリーンセットを起動することができます。この場合、データブロックを正しくセットアップしておかなければなりません。

複数のスクリーンセットと複数のプログラムを使用する場合、イベントが起こったときに正しいスクリーンセットとプログラムがロードされるように、プログラムとスクリーンセットの切り換え方法をインプリメントする必要があります。Dialog Systemには、これの切り替えをインプリメントしたRouterというサンプルプログラムが用意されています。このプログラムについては、この章の後の方で説明します。

Dsrunnerは、Routerが行うすべての作業に加え、他の作業も行います。

11.1.2 Dsrunnerの動作

Dsrunnerはアプリケーションのメインプログラムです。通常、アプリケーションには他のスクリーンセットやサブプログラムもあります。メイン（Dsrunner）スクリーンセットは、アプリケーションの最初のウィンドウを提供し、Dsrunnerプログラムは、他のスクリーンセットやサブプログラムをロードするための機能を提供します。

各スクリーンセットには、（オプションで）関連するサブプログラムがあります。スクリーンセットとサブプログラムを関連付けるには、両方の名前（ファイル拡張子を除く）を同じにします。たとえば、dsrnr.gsというスクリーンセットがある場合、それに関連するサブプログラムはdsrnr.cblという名前にします。

スクリーンセットおよびそれに関連するサブプログラムを起動すると、

1. Dsrunnerは、スクリーンセットに関するデータブロックのためにメモリを割り当て、それをLOW-VALUESに初期設定します。
2. スクリーンセットをDialog Systemにロードするまえにサブプログラムを呼び出します。

   これによって、必要な初期設定が実行されます。
3. 初期設定が終わると、サブプログラムはEXIT PROGRAMを実行します。
4. EXIT PROGRAMによって、Dsrunnerはスクリーンセットをロードします。
5. スクリーンセットは、SCREENSET-INITIALIZEDロジックを実行してから、イベント処理ループに入ります。

   スクリーンセットがRETC命令を実行した場合、Dsrunnerに戻ります。
6. つぎにDsrunnerは、Dsrunner機能が要求されているかどうかを調べます（有効なSIGNATUREと機能コードが指定されているかをチェックします）。
   - 機能が要求されている場合、Dsrunnerはその機能を実行し、スクリーンセットに戻ります。
   - 機能が要求されていない場合（通常の場合）、Dsrunnerはサブプログラムを呼び出します（CALL）。
7. サブプログラムは、
   1. 要求された機能を実行します。
   2. データブロックを更新します。
   3. EXIT PROGRAMを実行して、Dsrunnerに制御を返します。

   Dsrunnerはスクリーンセットに制御を返します。

11.1.2.1 **パラメータ**

サブプログラムを呼び出すときに、4つのパラメータが渡されます。これらは、プログラムのLinkage Sectionになければなりません。これらはサブプログラムのパラメータなので、プログラムのWorking-Storage Sectionにあってはなりません。

次のパラメータがあります。

- screenset-data-block （必須）

  screenset.cpbによって与えられます。

  screenset-data-blockは、スクリーンセットのために提供されるデータブロックです。サブプログラムを初めて呼び出したとき、このデータブロックはLOW-VALUESに初期設定されます。Dsrunnerは、データブロックのバージョン番号チェックを行いません。したがって、スクリーンセットのデータブロックとサブプログラムのデータブロックの同期には十分注意しなければなりません。

- dsrunner-info-block - Optional and can be ignored.

  dsrunner.cpyによって与えられます。

- ds-event-block（オプション。無視してもよい）

  dssysinf.cpyによって与えられます。これは、Dsgrunが返すのとまったく同じです。

- ds-control-block（オプション。無視してもよい）

  ds-cntrl.cpyによって与えられます。DsrunnerがDsgrunを呼び出すのに使用されます。このパラメータには、DsrunnerがDsgrunの呼び出しを発行するための値を含んでいます。また、戻り時にDsgrunによって返される値も含んでいます。これには、エラーコード、ウィンドウ名、オブジェクト名などがあります。ds-control-blockについての詳細は、*スクリーンセットの使い方*の章の*コントロールブロック*を参照してください。.

  サブプログラムは、スクリーンセットの実行方法を変更するためにds-control-blockのds-clear-dialogフィールドとds-procedureフィールドを修正します。他のフィールドは、Dsrunnerの制御下にあり、変更できません。

11.1.2.2 Dsrunner**のスクリーンセット**

Dsrunnerは、ユーザーによる特別な操作なしに任意のスクリーンセットを実行できます。この場合、[オプション]メニューの[構成]、Screensetで、screenset-idが定義されていなければなりません。Dsrunnerがスクリーンセットを切り換えるには、screenset-idが必要です。

DsrunnerスクリーンセットはDialog Systemに付属しており、dsrunner.gsはDialogSystem¥binサブディレクトリにあります。このスクリーンセットは、変更可能なテンプレートです。これとは別にユーザーが独自のスクリーンセットを作成することができます。この場合、データブロックを正しくセットアップし、必要なグローバルダイアログを用意し

なければなりません。

データブロック見出し

Dsrunnerスクリーンセットには、データブロックの始めに次のフィールドがなければなりません。これらのフィールドは制御情報とディスパッチ情報を保存します。

```
DSRUNNER-DATA-ITEMS       1
  DSRUNNER-SIGNATURE      X 8.0
  DSRUNNER-FUNCTION-CODE  X 4.0
  DSRUNNER-RETURN-CODE    C 2.0
  DSRUNNER-PARAM-NUMERIC  C 4.0
  DSRUNNER-PARAM-STRING   X 256.0
```

- DSRUNNER-SIGNATURE

  シグナチャです。Dsrunnerによってすでに起動されたスクリーンセットから複数のスクリーンセットを順番に起動するには、Dsrunnerスクリーンセットであることを示すシグナチャをデータブロックの中で指定する必要があります。

- DSRUNNER-FUNCTION-CODE

  機能コードです。

- DSRUNNER-RETURN-CODE

  戻りコードです。

- DSRUNNER-PARAM-NUMERICとDSRUNNER-PARAM-STRING

  数値パラメータと文字列パラメータです。

メインスクリーンセットがこれらの規則に従っていない場合も、起動することは可能です。しかし、他のアプリケーションは起動されません。

共有メモリバッファを使用したい場合、DSRUNNER-DATA-ITEMSグループの直後に次のフィールドを挿入しなければなりません。

```
SHARED-MEMORY-BUFFER
  YOUR-DATA                任意の形式またはサイズ
```

スクリーンセットのデータブロックの始めには、Dsrunnerが使用するいくつかのフィールドを取っておく必要があります。このためにdsrunner.impというファイルをデータブロックの最初の部分にインポートできます。このインポートファイルには、グローバルダイアログが次のことを行うのに必要な文も入っています。

- Dsrunnerのシグナチャをセットアップする。
- スクリーンセットの切り替えを可能にする。
- 適切に終了する。

Dsrunnerスクリーンセットの要件

Dsrunnerといっしょに使用するスクリーンセットは、次の条件を満たしていなければなりません。

- ロードされるスクリーンセットついて一意なscreenset-idを含んでいること。
- 次のダイアログを使って自身およびサブプログラムを閉じること。

  ```
  SET-EXIT-FLAG
    RETC
  ```

  このダイアログは、グローバルでも、ローカルでもよく、任意のイベントに接続できます。閉じることをサブプログラムに知らせるには、独自のターミネーションフラグを設定し、終了フラグを設定するまえにRETCを実行し、Dsrunnerに対するRETCを実行します。

- 複数のスクリーンセットを取り扱う場合、次のグローバルダイアログを含んでいること。

  ```
  OTHER-SCREENSET
    REPEAT-EVENT
    RETC
  ```

Dsrunnerのグローバルダイアログ

提供されるDsrunnerスクリーンセットのグローバルダイアログの主要部分を次に示します。

```
OTHER-SCREENSET
  REPEAT-EVENT
  RETC
```

このダイアログによって、非アクティブなスクリーンセットに関して起こるイベントが反復されます（非アクティブなスクリーンセットに関してスタックされます）。RETCによって、Dsrunnerはアクティブなスクリーンセットを正しいスクリーンセットに切り換えます。これは、複数のスクリーンセットを制御する場合に必要です。

```
SCREENSET-INITIALIZED
  MOVE "DSRUNNER" DSRUNNER-SIGNATURE(1)
```

このダイアログは、このスクリーンセットのデータブロックをDsrunnerに関してセットアップすることを示すシグナチャを設定します。データブロックをセットアップしない場合、Dsrunnerはすべての機能コードを無視し、RETCはスクリーンセットに関連したサブプログラムだけを呼び出します。

```
CLOSED-WINDOW
   EXECUTE-PROCEDURE  EXIT-PROGRAM
CLOSEDOWN
   EXECUTE-PROCEDURE  EXIT-PROGRAM
  EXIT-PROGRAM
   SET-EXIT-FLAG
   RETC
```

アプリケーションを閉じる要求に対して、SET-EXIT-FLAGが発行されます。これによって、DsrunnerはRETCのあとに終了します。サブプログラムの中で終了処理 を実行したい場合、独自のターミネーションフラグを設定し、終了フラグを設定するまえにRETCを実行し、Dsrunnerに対するRETCを実行します。

```
 OPEN-SCREENSET
   MOVE "file" DSRUNNER-FUNCTION-CODE(1)
   MOVE "*.gs" DSRUNNER-PARAM-STRING(1)
   RETC
   IFNOT= DSRUNNER-RETURN-CODE(1) 0 OPEN-SCREENSET-ERROR
```

*** 結果のファイル名は**`param-string`**に残る**

```
   MOVE "lnch" DSRUNNER-FUNCTION-CODE(1)
   RETC
 OPEN-SCREENSET-ERROR
```

このダイアログは、Dsrunner機能の実行方法を示しています。この例では、ファイルリクエスタを示し、ファイル名を獲得しています。そして、スクリーンセットを起動するための機能を実行しています。

11.1.3 Dsrunner**プログラムと機能**

Dsrunnerプログラムは、データブロックの最初の数フィールドによって特定の機能を提供します。次の機能があります。

- 新しいアプリケーション（スクリーンセット）またはアプリケーションのインスタンスの起動と停止
- デバッグのためのDialog Systemトレースモードのオンオフ
- 共有メモリ領域の作成と使用

機能の一覧については、ヘルプの*Dsrunnerの機能*トピックを参照してください。

11.1.4 Dsrunner機能の使い方

Dialog SystemはDsrunnerによって呼び出されるため、DsrunnerスクリーンセットからRETCを実行するまえに、Dsrunner機能のコードをDsrunnerのデータブロック内に設定します。たとえば、

```
LAUNCH-SCREENSET
  MOVE "lnch" DSRUNNER-FUNCTION-CODE(1)
  MOVE "screenset-name" DSRUNNER-PARAM-STRING(1)
  RETC
  MOVE DSRUNNER-PARAM-NUMERIC(1) SAVED-SS-INSTANCE
```

この例では、`screenset-name`が起動するスクリーンセットの名前です。この機能はスクリーンセットインスタンスを返します。これは、`SAVED-SS-INSTANCE`として保存されます。

11.1.5 コマンド行を使ったスクリーンセットの起動

Dsrunnerは、コマンド行から実行するように設計されています。

runw dsrunner [スクリーンセット名 /l /d スクリーンセット名]

ただし、

スクリーンセット名は、ロードするスクリーンセットの名前です。これは、最初のパラメータとして入力することも、/l "スクリーンセットをロードする"パラメータに続けて入力することもできます。/dスイッチも指定できます。

/dは、スクリーンセットアニメータを直ちにイネーブルにします。スクリーンセットアニメータは、最初のスクリーンセットにあるダイアログの最初行を実行するときに呼び出されます。

使用しているランタイム環境に合わせてアプリケーションをパッケージングできるように、Dsrunnerは.obj形式と.gnt形式の両方で提供されます。

11.1.6 NetExpress IDEでのスクリーンセットの起動

スクリーンセットは、NetExpress IDEでDsrunnerを通じて起動することもできます。

1. [アニメート]メニューから[設定]を選択します。
2. [アニメート開始位置]でdsrunnerを指定します。
3. [コマンド行パラメータ]で上記のオプション（/l、/d）を指定します。

プログラムにブレークポイントがあると、実行が停止し、関連するプログラムがだバッグ／編集ウィンドウにロードされます。

11.1.7 プログラムからのスクリーンセットの起動

コマンド行を使用するかわりに、プログラムからDsrunnerを呼び出すことができます。これによって、アプリケーションに独自の名前を付け、初期スクリーンセット名をコマンド行ではなくプログラムの中で指定することができます。これは、アプリケーションが自身のコマンド行引数を持っている場合に便利です。また、ファイルを開くなど、アプリケーション固有の初期設定を行うときにも便利です。

11.1.8 スクリーンセットの起動

スクリーンセットを起動するには、

1. Dsrunnerウィンドウの[ファイル]メニューから[開く]を選択します。
2. スクリーンセットを選択し、OKをクリックします。

   選択したスクリーンセットがロードされ、関連するプログラムが実行されます。

上の手順を繰り返すことによって、別のスクリーンセットを選択し、実行することができます。

11.1.9 アプリケーションの起動

この項では、Dsrunnerのアーキテクチャ を見て、Dsrunnerを効率的に使用するにはどうしたらよいかを考えます。Dsrunnerスクリーンセットは、個々の要件に合わせて変更できます。提供されたスクリーンセットは例に過ぎません。データブロックは変更できますが、`dsrunner-info-block`に影響を与えるような変更は行わないでください。

スクリーンセットを起動すると、Dsrunnerは次の手順を実行します。

1. Dsrunnerウィンドウの[ファイル]メニューから[開く]を選択すると、スクリーンセット名の入力を指示するプロンプトが表示されます。
2. Dsrunnerは、スクリーンセットが正しいか、すなわちscreenset-idが指定されているかをチェックします。スクリーンセットが、
   - 正しくない場合、そのことを示すメッセージボックスが表示されます。
   - 正しい場合、動的に割り当てられたデータブロックに十分なメモリが与えられ、LOW-VALUESに初期設定されます。
3. スクリーンセットとルートファイル名（およびパス）が同じプログラムが呼び出されます。

   このプログラムは、現在のディレクトリまたは$COBDIR（通常のCOBOLプログラム検索規則を適用）にある有効なCOBOL実行可能プログラムです。プログラムが見つからない場合、そのことを示すメッセージメッセージボックスが表示されます。
4. データブロックの初期設定を含む、必要な初期設定を実行するためにサブプログラムが呼び出されます。

次のパラメータが渡されます。

- `Data-Block`
- `Dsrunner-Info-Block`

  これはdsrunner.cpyによって提供され、次の情報が入っています。

  - `screenset-id`
  - `ds-session-id`
  - `screenset-instance-number`
  - エラーコード
- `Ds-Event-Block`

5. サブプログラムが初期化されたら、戻りコードゼロでDsrunnerに戻らなければなりません。

   - 非ゼロの戻りコードで戻った場合、Dsrunnerはメッセージボックスを表示し、エラーをシグナリングします。
   - ゼロの戻りコードで戻った場合、Dsrunnerはスクリーンセットをロードします（スクリーンセットが初期化されます）。

     スクリーンセットをロードするためにDialog Systemを呼び出すときにエラー場起こった場合、サブプログラムが呼び出されます。このとき、Dsrunnerの情報ブロックにはerror-codesが指定されます。
6. スクリーンセットがRETCを実行すると、サブプログラムが呼び出されます。
7. アプリケーションを閉じるには、スクリーンセットはダイアログの中でSET-EXIT-FLAGしなければなりません。

11.1.9.1 サンプルサブプログラムの実行

Dsrnrは、Dialog Systemに付属のサンプルサブプログラムです。このプログラムの起動方法とサンプルコードの主要なセクションについては、*サンプルプログラム*の章を参照してください。

11.2 複数のスクリーンセットとRouterプログラム

Dialog Systemシステムのランタイムシステムは、プログラムすることによって次のように複数のスクリーンセットを使用できます。

- 複数のスクリーンセット

- 同じスクリーンセットの複数のインスタンス

これらの機能を使うと次のことができます。

- ユーザーインタフェースを複数の論理構成要素に分割する。
- 同じスクリーンセットの複数のコピーを使用する。
- すべてのエラーメッセージを1つのファイルにまとめる。

スクリーンセットと呼び出しプログラムを設計する場合、スクリーンセットの取り扱いを制御する方法について、次のことを詳しく考慮する必要があります。

- 機能をどのように分割するか、複数のスクリーンセットにまとめる必要があるかどうか
- セキュリティ上の理由から、データへのアクセスが異なる個々のユーザーグループに異なるスクリーンセットを提供するかどうか
- スクリーンセットの複数のインスタンスを使用して、ユーザーがデータを同時に表示、比較、または編集できるようにするかどうか

Dialog Systemの呼び出しインタフェースを使ったスクリーン制御の基本について詳しくは、ヘルプの*「呼び出しインタフェース」*トピックを参照してください。

11.2.1 複数のスクリーンセットの使用

スタックに対するプッシュとポップによって、複数のスクリーンセットを使用できます。これは、その定義から先入れ先出し操作です。スクリーンセットのプッシュとポップは、次の操作に便利です。

- 特定の機能に使用されるスクリーンセットを必要なくなったときに表示から取り除く。
- プログラムの初期設定時に複数のスクリーンセットをロードし、プッシュし、必要になったときにポップする（使用する）。
- 未使用または入力フォーカスがないウィンドウによって、画面がいっぱいにならないようにする。

スクリーンセットをスクリーンセットスタックにプッシュする際の前提条件はなく、任意のスクリーンセットまたはスクリーンセットのオカレンスをプッシュまたはポップできます。通常、プッシュされたスクリーンセットは、メモリにスタッキングされますが、メモリが不足している場合、ディスクにページングされます。

11.2.2 複数のプログラムとスクリーンセットの使用

大規模なアプリケーションを開発する場合、コンポーネントとともに別々のモジュールを作成し、各モジュールに自身のスクリーンセットを関連付けるのがよい方法です。それぞれの構成要素は、専用のインタフェースを持つ独立したCOBOLプログラムになります。

たとえば、メインのデータ入力構成要素と2つのユーティリティ構成要素（一方は印刷機能、もう一方は管理機能を取り扱う）を持つアプリケーションを構築しているとします。

1つのアプリケーションの中で複数のスクリーンセットを使用することができます。それぞれのスクリーンセットがスクリーンセットスタックに配置されます。このスタックは、プログラムを呼び出すときに使用されるコールスタックと概念的に同じです。複数のスクリーンセットを順番に呼び出すのは簡単です。具体的には、`ds-control`に値N、`ds-set-name`に新しいスクリーンセット名をいれて、Dialog Systemを呼び出すだけです。デフォルトでは、古いセットが画面から消去されてから、新しいセットが起動します。

複数のプログラムがあり、それぞれにスクリーンセットが関連付けられている場合、イベントが起こったときに正しいプログラムとスクリーンセットがアクティブになるようにする必要があります。次に説明するサンプルプログラムRouterは、複数のプログラムがあり、それぞれにスクリーンセットが関連付けられたアプリケーションを構築する方法を示しています。

11.2.3 用語と概念

Routerプログラムを見るまえに、いくつかの用語と概念を理解している必要があります。

11.2.3.1 アクティブなスクリーンセット

複数のスクリーンセットがロードされている場合、アクティブなスクリーンセットとアクティブではないスクリーンセットを区別する必要があります。現在アクティブではないスクリーンセットによって表示されるグラフィカルオブジェクトは、アクティブなスクリーンセットによって表示されるものと外見に違いはありません。アクティブなスクリーンセットは、現在イベントを受け取っているスクリーンセットです。一度に1つのスクリーンセットだけがイベントを受け取ることができます。

一般に複数のスクリーンセットを使用する場合、Dialog Systemを呼び出し、`ds-control`に`ds-use-set`を設定することによって、アクティブなスクリーンセットを指定します。`ds-set-name`で指定されたスクリーンセットはアクティブなスクリーンセットになります。この呼び出しは非常に高速で、スワッピングに伴うオーバーヘッドも最小限です。

11.2.3.2 他のスクリーンセットに関するイベント

現在アクティブなスクリーンセット以外のスクリーンセットに関するイベントが起こった場合、現在のスクリーンセットではOTHER-SCREENSETイベントという特殊なイベントが起こります。このイベントは、別のスクリーンセットに通知すべきイベントが起こったことを現在のスクリーンセットに知らせるだけです。これに対する動作は、アクティブなスクリーンセットのロジックに依存します。

OTHER-SCREENSETが（通常はグローバルダイアログ内に）見つからない場合、何も起こりません。別のスクリーンセットに関するイベントを検出できるように、OTHER-SCREENSETを指定しなければなりません。このようなイベントが起こった場合、適切なスクリーンセットをアクティブにします。最もよく行われる方法としては、スクリーンセットを変更するために、OTHER-SCREENSETイベントが起こったことを示すフラグをセットし、呼び出しプロ

グラムに戻ります。

アクティブではないスクリーンセットについて起こったすべてのイベントは、アクティブなスクリーンセットにOTHER-SCREENSETイベントとして返されます。

正しいスクリーンセットを識別するには、Dialog Systemを呼び出し、`ds-event-block`を指定します。Dialog Systemが呼び出しプログラムに戻ったとき、イベントに本来対応するスクリーンセットのscreenset-idが`ds-event-screenset-id`に入っています。このscreenset-idは、[オプション]メニューの[構成/スクリーンセット]で指定します。Screenset-idの指定は必須ではありませんが、指定しない場合、スタック内のスクリーンセットを区別できないので、複数のスクリーンセットを使用することができません。

正しいスクリーンセットがアクティブになっている場合、OTHER-SCREENSETイベントでREPEAT-EVENTダイアログ機能が指定されているときは、もとのイベントが繰り返されます。

注意：繰り返されるイベントはOTHER-SCREENSETではありません。正しいスクリーンセットがアクティブだった場合に起こったイベントです。

11.2.4 Routerを使った複数スクリーンセットのサンプルアプリケーション

Routerを使ってDialog Systemで複数のスクリーンセットを取り扱う方法を示すサンプルアプリケーションが、サンプルディレクトリにあります。この例では、メインプログラムのProgramaがサブプログラムのProgrambとProgramcを呼び出します。各プログラムは自身のスクリーンセットを持っています。

4番目のプログラムのRouterは、ルーティング機能を取り扱います。このプログラムの唯一の目的は、どのプログラム（およびスクリーンセット）を次に実行すべきかを判断し、そのプログラムを呼び出します。

アプリケーションの構造を図 11-1 に示します。

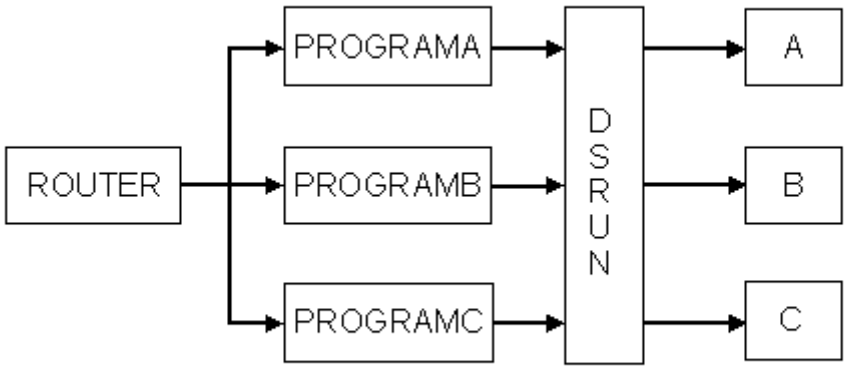


図 11-1 Routerアプリケーションの構造

11.2.5 複数のスクリーンセットインスタンスの使用

複数のスクリーンセットを使用するだけではなく、同じスクリーンセットの複数のインスタンスを使用することができます。複数のインスタンスは、複数のスクリーンセットとほとんど同じ方法で使用します。実際の違いはスクリーンセットの識別です。この項を読むまえに、*複数のプログラムとスクリーンセットの使用*の項を参照してください。

複数のスクリーンセットインスタンスは、次のような場合に使用できます。

- 1つのウィンドウとそれに関連するコントロールを含んだスクリーンセット
- 各グループ項目が同じ形式であるようなデータグループの操作。同じスクリーンセットの複数のインスタンスを使用すると、1つのスクリーンセットで必要なグループ項目を表示、比較、または更新できます。

同じスクリーンセットの複数のインスタンスを使用するには、プログラムで次のことを行う必要があります。

- スクリーンセットのインスタンスの数を追跡する。
- 各スクリーンセットインスタンスに対して適切なデータブロックがDsgrunに渡されるようにする。

複数のインスタンスを使用する場合、最小に"N"または"S"呼び出しによってスクリーンセットを起動します。

新しいインスタンスを作成するには、

1. Dialog Systemを呼び出して、新しいスクリーンセットをスタックにプッシュします。
2. `ds-control`で`ds-push-set`を指定します。
3. Dialog Systemが戻ると、`ds-screenset-instance`制御部ロックフィールドに割り当てられたインスタンス値が配置されます。

インスタンス値は常に返されますが、この値が必要なのは、スクリーンセットの複数のインスタンスを使用するときだけです。

インスタンス値は、そのスクリーンセットインスタンスに対して一意です。この値は決まった順序で指定されるわけではないので、アプリケーションは値を追跡しなければなりません。

11.2.5.1 アクティブなインスタンスの値の追跡

アクティブなインスタンスの値は、dssysinf.cpyの中の`ds-event-screenset-id`と`ds-event-screenset-instance-no`を調べることによって追跡できます。dssysinf.cpyは、プログラムのWorking-Storage Sectionにコピーしなければなりません。dssysinf.cpyの詳細は、*Panels V2の使い方*の章を参照してください。

スクリーンセットの新しいインスタンスのロードを指示するには、

- Dsgrunを呼び出すときに`ds-control`を`ds-use-instance-set`に設定します。

個々のイベントに関する適切なインスタンスを識別するには、

- `ds-event-screenset-instance`を調べます。これには、適切なインスタンス値が入っています。

特定のインスタンス値でDialog Systemを呼び出すには、

1. `ds-event-screenset-instance-no`を`ds-instance`に移動します。
2. `ds-event-screenset-id`を`ds-set-name`に移動することによって、必要なスクリーンセットを指定します。
3. Dialog Systemを呼び出します。

注意：

- スクリーンセットの複数のインスタンスを使用するには、[オプション]メニューから[構成/スクリーンセット]を選択して、screenset-idをスクリーンセットの名前に設定しなければなりません。
- 定義ソフトウェアを使い、スクリーンセットアニメータによって実行している場合、スクリーンセットの複数のインスタンスの使用はサポートされません。しかし、アプリケーションからDsgrunを呼び出した場合、複数のインスタンスがサポートされます。スクリーンセットアニメータについての詳細は、ヘルプの*スクリーンセットアニメータ*の項を参照してください。

11.2.5.2 正しいデータブロックの使い方

スクリーンセットの複数のインスタンスを使用している場合、データブロックの複数のコピーを用意しなければなり

ません。これには、いくつかの方法があります。

- 作成されるインスタンスの数がわかっている場合、次のようなコードを使用するのが最も簡単です。

  ```
  copy "program.cpb"
  ```

  data-block-aをdata-block-bに置き換えます。

- データブロックのヒープまたはスタックをインプリメントします。

  スクリーンセットをスクリーンセットスタックにプッシュし、新しいスクリーンセットを起動するには、次のようなコードを使ってDsgrunを呼び出します。

  ```
  move ds-push-set to ds-control
  call "dsgrun" using ds-control-block,
                      data-block
  ```

  ds-push-setは値"S"をds-controlに配置します。既存のスクリーンセットはスクリーンセットスタックにプッシュされます。

  スクリーンセットスタックからスクリーンセットをポップするときは、次のどちらかのコードが使用できます。

  - ds-quit-set

    既存のスクリーンセットを閉じ、スクリーンセットスタックの一番上にある最初のスクリーンセットをポップします。

  - ds-use-set

    指定されたスクリーンセットをスクリーンセットスタックからポップします。既存のスクリーンセットは閉じません。

詳しくは、ヘルプの*スクリーンセットアニメータトピック*を参照してください。

11.2.5.3 複数のインスタンスに関するサンプルプログラム

プログラムによる呼び出しインタフェースの使用を示す2つのサンプルプログラムがあります。

- push-pop.cblは、スクリーンセットのプッシュとポップの使用に関するデモンストレーションです。
- custom1.cblは、同じスクリーンセットの複数のインスタンスを使用します。

これらのプログラムの主要なコードセクションについては、*サンプルプログラム*の章を参照してください。

11.2.6 Routerプログラム

Routerプログラムのロジックを見るまえに、RouterおよびRouterによって呼び出されるすべてのプログラムが共有するデータ領域を表すCOPYファイルを見てみます。

次のコードはCOPYファイルrouter.cpyを示しています。これには、プログラム名と2つのフラグが入っています。

`program-name`には、次に呼び出すプログラムの名前が入っています。

`cancel-on-return`は、Routerに戻ったときにキャンセルされるプログラムによってTRUEに設定されます。メインプログラム（Programa）がキャンセルを要求した場合、Routerを`exit-on-return`に設定し、メイン実行ループを終了します。

```
 1 01  program-control.
 2
 3    03  dispatch-flag          pic 9(2) comp-5.
 4        88 cancel-on-return    value 1 false 0.
 5
 6    03  exit-flag             pic 9(2) comp-5.
 7        88 exit-on-return      value 1 false 0.
 8
 9    03  program-name          pic X(8).
```

Routerは、メインプログラムProgramaを呼び出すことによって起動します。Programaは、特定の機能を取り扱うためにサブプログラムが必要であると判断した場合、サブプログラムの名前を`program-name`に書き込み、終了します。つぎにRouterは`program-name`にあるプログラムを呼び出します。

```
29 main-section.
30
31*    通常の方法では終了しないことを確認する
32     initialize exit-flag
33
34*    メインプログラムを呼び出すことによって起動する
35     move main-program to program-name
36
37*    終了が要求されるまで、program-nameにあるプログラムを呼び出す
38     perform until exit-on-return
```

```
39
40*      どれを呼び出したかを覚えておく
41       move program-name to dispatched-program
42
43*      Program-Nameにあるプログラムをディスパッチする
44       call program-name using
45       if cancel-on-return
46*        最後のプログラムがキャンセルを要求した場合、キャンセルを実行する
47         set cancel-on-return to false
48         cancel dispatched-program
49         if dispatched-program not = main-program
50*          サブプログラムがキャンセルされた場合、メインプログラムを再ロードする
51           move main-program to program-name
52         else
53*          メインプログラムがキャンセルを要求した場合、終了を要求する
54           set exit-on-return to true
55         end-if
56       end-if
57     end-perform
58
59     stop run.
```

プログラムをキャンセルしたい場合、プログラムがRouterに戻るまえに、`cancel-on-return`を設定しなければなりません。`main-program`に指定されたプログラムがキャンセルを要求した場合、Routerはメインプログラムをキャンセルしたあとに終了します。

11.2.7 メインプログラム

この項では、Routerが呼び出すメインプログラム（Programa）のソースを示します。2つのサブプログラム（ProgrambとProgramc）はほとんど同じです。

行31**〜**44**：**

```
31 procedure division using program-control.
32
```

```
 33 main-section.
 34     if new-instance
 35*    最初、新しいスクリーンセットをスタックにプッシュする
 36         perform new-set-instance
 37     else
 38*    初期設定が終わったら、既存のスクリーンセットを使用する
 39         perform use-set-instance
 40     end-if
 41*    アクティブな間はDialog Systemを呼び出す
 42     perform until program-name not = this-program-name
 43         perform call-ds
 44     exit-program.
```

Programaのメインセクションは、スクリーンセットインスタンスが作成されたかどうかをチェックします。作成されている場合、プログラムはそれを使用します。作成されていない場合、プログラムはインスタンスを作成します。

インスタンスが揃ったら、Dialog Systemを呼び出してそれを表示します。`program-name`にあるプログラムの名前がProgramaであるかぎり、Dialog Systemが呼び出されます。

`program-name`がProgramaではない場合、プログラムは終了します（そして、Routerに戻ります）。Routerは、`program-name`にあるプログラムを呼び出します。

行46**〜**60**：**

```
 46 new-set-instance
 47 ...
 58 ...
 59*    新しいスクリーンセットをスタックにプッシュする
 60     move ds-push-set to ds-control.
```

`new-set-instance`の最も重要な部分は、`ds-push-set`の指定です。これによって、Dialog Systemを呼び出すときに、スクリーンセットがスタックにプッシュされます。

行62**〜**66**：**

```
 62 use-set-instance.
 63*    スタックにある既存のスクリーンセットを使用する
 64     move ds-use-set to ds-control
```

```
65*    これが呼び出されたもの
66     move this-program-name to ds-set-name.
```

作成されたスクリーンセットを使用するには、Dialog Systemにスクリーンセット名を知らせ、スタックからそのスクリーンセットを使用するように指示します。

行67**〜**101**：**

```
67 ...
68 call-ds.
69*    標準の初期設定と呼び出し
70     initialize programa-flags
71     call "dsgrun" using ds-control-block
72                         programa-data-block
73                         ds-event-block
74     evaluate true
75*    このスクリーンセットフラグがセットされている場合、routerに終了を指示する
76         when programa-terminate-true
77             set exit-on-return to true
78 ...
87 ...
88         when programa-other-set-true
89             move ds-event-screenset-id to program-name
90*    Programbメニュー項目が選択された場合、このフラグがセットされる
91         when programa-program-b-true
92*            Programbのディスパッチを要求する
93             move "programb" to program-name
94*    Programcメニュー項目が選択された場合、このフラグがセットされる
95         when programa-program-c-true
96*            Programcのディスパッチを要求する
97             move "programc" to program-name
98*    ここに関するイベントである
99         when other
```

```
100             move "Hello A" to programa-field1
101     end-evaluate.
```

このコードは、標準的なDialog Systemの呼び出しを締め質得ます。3つのパラメータをすべて指定し、そのあとに評価が続きます。評価では、スクリーンセットダイアログで設定されたフラグに基づいて、プログラム全体の動作を指示します。

スクリーンセットダイアログで設定されたフラグによって、Programaを終了するか、スクリーンセットとプログラムを切り換えるか、サブプログラムを呼び出すか、イベントを取り扱います。

このコードで重要なのは、OTHER-SET-TRUEフラグがセットされたときのプログラムとスクリーンセットの切り換えです。Dialog Systemは、イベントが起こったスクリーンセットのScreenset-ID（[オプション]メニューの[構成/スクリーンセット]によって設定）をds-event-screenset-idに書き込みます。

これは、Programaが正しいscreenset-idを検出し、それによってプログラム名を検出することを意味しています。これが可能なのは、この例ではスクリーンセットとそれに関連するプログラムを同じファイル名で呼び出しているからです。スクリーンセットとそれに関連するプログラムの名前が異なる場合、スクリーンセットを識別してから、適切なプログラムを呼び出す必要があります。

Programaはds-screenset-nameにscreenset-idを書き込み、終了します。Routerは識別されたプログラムを呼び出し、正しいスクリーンセットをロードします。

11.2.8 複数のスクリーンセットのダイアログ

次のダイアログ例は、複数のスクリーンセットを取り扱うのに必要なスクリーンセットダイアログを示しています。ここにリストされたダイアログは、Programaから取り出したものですが、他のスクリーンセットについても同じです。

Dialog Systemは、現在ロードされたスクリーンセット以外のすくりーセットでイベントが起こったことを検出するためにOTHER-SCREENSETを使用します。この例では、イベントは各スクリーンセットのグローバルダイアログテーブルにあります。

```
OTHER-SCREENSET
    SET-FLAG OTHER-SET(1)
    REPEAT-EVENT
    RETC
```

このようなイベントが検出されると、OTHER-SETフラグがセットされ、プログラムに対して別のスクリーンセットを使用するようにシグナリングします。

つぎにREPEAT-EVENT機能が実行されます。この機能は、Dialog Systemに対して次回入力を調べるときに最後のイベントを繰り返すように指示します。これは、次のスクリーンセットがロードされ、制御が次のプログラムに渡ったときに起こります。REPEAT-EVENTのあと、スクリーンセットはRETCを使って呼び出しプログラムに戻ります。

呼び出しプログラム（ここではPrograma）は、スクリーンセットのフラグをチェックできます。フラグがセットされている場合、プログラムは（スクリーンセット名とプログラム名が同じだとみなし）screenset-idを`ds-set-name`に書き込み、Routerに戻ります。Routerは適切なプログラムを呼び出し、正しいスクリーンセットをロードします。そして、ProgramaスクリーンセットによってREPEAT-EVENTが発行されるため、OTHER-SCREENSETイベントを起こしたイベントが繰り返されます。

注意：繰り返されるイベントはOTHER-SCREENSETではありません。正しいスクリーンセットがアクティブだった場合に起こったイベントです。

これらの機能についての詳細は、ヘルプの*「ダイアログの定義: ファンクション」*トピックを参照してください。

11.2.9 イベントシーケンス

前の項では、スクリーンセット切り換えの全体的プロセスと主要なイベントについて説明しました。しかし、実際のプログラムフローはこのように単純ではありません。制御が適切なプログラムに渡るまえに、いくつかのプログラム切り換えが行われます。

これはなぜでしょうか。理由はフォーカスの変更があるからです。あるグラフィカルオブジェクトから別のグラフィカルオブジェクトへ入力フォーカスが移動すると、2つのイベントが起こります。フォーカスを失うオブジェクトはLOST-FOCUSイベントを受け取り、フォーカスを得るオブジェクトはGAINED-FOCUSイベントを受け取ります。

フォーカスを失うオブジェクトがコントロールオブジェクトの場合、そのオブジェクトが入ったウィンドウに関するLOST-FOCUSイベントも起こります。同様にフォーカスを得るオブジェクトがコントロールオブジェクトの場合、そのオブジェクトが入ったウィンドウに関するGAINED-FOCUSも起こります。

次の表は、AのウィンドウからBのウィンドウへの単純のフォーカス移動があったときに、実際に起こるイベントシーケンスを示しています。

システムイベント	アクティブなスクリーンセット	イベントのスクリーンセット	DSイベント
Mouse-button-1-down	A	B	OTHER-SCREENSET
Lost-focus (window A)	B	A	OTHER-SCREENSET
Gained-focus (window B)	A	B	OTHER-SCREENSET
Mouse-button-1-up	B	B	ANY-OTHER-EVENT

各スクリーンセットはOTHER-SCREENSETイベントをトラップし、Dialog Systemから呼び出しプログラムに戻ります。そして、Dialog Systemが正しいスクリーンセット（`ds-event-screenset-id`によって識別される）といっ

しょに呼び出されます。

Aのウィンドウにフォーカスがあるときにbのウィンドウでマウスボタンを押すと、このテーブルに示すイベントシーケンスが起こります。

ウィンドウBのシステムイベントMouse-button-1-downによって、Dialog Systemはアクティブなスクリーンセットに OTHER-SCREENSETイベントを提示します。これによって、スクリーンセットBに切り換えるチャンスがプログラム Aに与えられます。

次のシステムイベントはフォーカスをウィンドウBです。しかし、表を見るとわかるように、このイベントはBへの切り換え後に起こります。これによって、BでOTHER-SCREENSETイベントが起こります。

もう1つのスクリーンセット切り替えがあります。Aがアクティブになったときにbがフォーカスを得るため、別の OTHER-SCREENSETイベントが起こります。これによって、Bがアクティブになります。マウスボタンアップイベントがANY-OTHER-EVENTとしてBに提示されます（すべてのマウスイベントは、このイベントに変換されます）。

スクリーンセットのスワップの原理を理解していると、このイベントシーケンスで悩むことはないでしょう。これは、スクリーンセットのスワップがいくつ起こっても、安定状態に達したときに正しいスクリーンセットがアクティブになるからです。

注意：この例では、マウスボタンダウンイベントが失われます。これを防ぐには、OTHER-SCREENSETダイアログでREPEAT-EVENT機能を使用します。

11.2.9.1 イベントの反復

まず、イベントを繰り返す必要があるかどうかを判断しなければなりません。イベントを繰り返す目的は明らかです。イベントが必要な場合、正しいスクリーンセットで繰り返さなければなりません。しかし、イベントが必要ないため、繰り返さなくてもよい場合がしばしばあります。

フォーカスがwindowAのbuttonAにあるとき、windowBのbuttonBをクリックすると、次のイベントシーケンスが起こります。

```
Mouse-Button-1-Down
Mouse-Button-1-Up
Lost-Focus on buttonA
Lost-Focus on windowA
Gained-Focus on windowB
Gained-Focus on buttonB
```

```
Button-Clicked on buttonB
```

これは、REPEAT-EVENTが必要ないことを示しています。Button-ClickedイベントによってBUTTON-SELECTEDイベントが起こる前に、windowBのGained-FocusがScreensetBをロードしているからです。

11.2.10 フォーカスの設定

Dialog Systemを呼び出し、`ds-control`に`ds-use-set`を指定することによってスクリーンセットをアクティブにしても、そのスクリーンセットが自動的にフォーカスを得るわけではありません。ユーザーが（新しいウィンドウまたはコントロールを選択することによって）スクリーンセットを切り換えた場合、ユーザーの操作によってGAINED-FOCUSイベントが起こるため、フォーカスは新しいスクリーンセットに移動します。

プログラムによって切り替えが起こる場合、Dialog Systemを呼び出すまえに`ds-procedure`にダイアログ手続き名を指定しなければなりません。SET-FOCUSダイアログ機能を使って、適切なウィンドウにフォーカスを設定します。

11.3 次の章の紹介

呼び出しインタフェースについての詳細は、ヘルプの*「呼び出しインタフェース」*トピックを参照してください。このトピックでは、イベントブロックを含むコントロールブロック、データブロック、スクリーンセットアニメータの使い方、バージョンチェック、呼び出しプログラムがDialog Systemに返す値について説明しています。

第12章 異なるプラットフォームへの移行

Dialog Systemの本バージョンは、主にWindows 95およびWindows NTで実行するためのものです。

この章では、これらの環境の主な違いについて説明し、複数の環境で使用されるスクリーンセットを開発するための指針を示します。

スクリーンセットは、上記のいずれの環境でも作成でき、他の環境で実行することができます。これは、システムが適切な Dialog System ランタイムソフトウェアを使用するためです（呼出しプログラムでは、このことを意識する必要はありません）。オブジェクト定義には、異なる環境間で移植できないものがあります。Dialog System は、オプションで移植性に関する警告を提供します。[オプション」メニューの「含める]から[移植性に関する警告]を選ぶことによって、この警告を使用可能にした場合、移植できないオブジェクトを作成しようとすると、警告メッセージが表示されます。

移植の問題に取り組む場合は、アプリケーションを徹底的にテストすることをお勧めします。Dialog Systemでは、環境が絶えず変化しているという側面のために、移植に関するあらゆる違反の検出を保証することはできません。

12.1 環境間の相違

多くのオブジェクトやコントロールは、サポートされている環境間で、表面上はほとんど同じに見えます。ラジオボタンやチェックボックスなどのオブジェクトは、プレゼンテーションマネージャ、ウィンドウズ の違いを最もよく表している例です。

Dialog System が作成するオブジェクトは、現在の環境のオブジェクトに似た表示になっています。したがって、プレゼンテーションマネージャのもとで作成する場合、ラジオボタンは長円で囲まれ、この長円は選択されると色が付きます。Motif では、同じボタンが立体的なダイヤモンド形ボタンで、選択されると引っ込んだ表示になります。ある環境から別の環境へスクリーンセットを移動すると、オブジェクトの表示も変化します。

マウスの動作は、環境によって異なりますが、アプリケーションの設計によって決めることができます。

オブジェクトやメニュー項目には、ある環境だけで利用できるものがあります。たとえば、OLE2オブジェクトは、Windowsだけで利用することができます。

アプリケーションを移植可能にするには、各環境に固有の利点を利用するのではなく、アプリケーションを実行する環境で共通のファンクションや機能を使用し、すべての環境でアプリケーションが同じ動作を行なうようにしなければなりません。アプリケーションの移植性が必要ない場合、特定の環境だけを対象に設計し、その環境に固有の利点を利用することができます。

たとえば、32ビットWindowsのインタフェースを利用すると、COBOLシステムのクラスライブラリを使って、Dialog Systemのインタフェースを拡張することができます。しかし、そのインタフェースは以前のバージョンのDialog Systemに移植することはできません。

12.1.1 デスクトップモード

環境間の大きな違いの1つに Dialog System の起動方法があります。

Presentation Manager:

プレゼンテーションマネージャでは、Dialog System は Dialog System ウィンドウで起動します。各ウィンドウは、Dialog System ウィンドウ内のクリップされたウィンドウとして作成されます。デスクトップモードでは、デスクトップで直接オブジェクトを作成します。

Windows:
Windows では、Dialog System はデスクトップモードで起動します。これは、いずれの環境でも、メニューによってクリップされたウィンドウを作成できないためです。これによって、最初に作成したいウィンドウが、Dialog System ウィンドウ内にあった場合にクリップされるため、問題が起こります。デスクトップ上で直接ウィンドウを作成すると、この問題を避けることができます。

Windows:
Windows では、ウィンドウ上のメニューバーを見るには、デスクトップモードでなければなりません。しかし、メニューバーはデスクトップモードでなくても編集できます。

12.2 グラフィカル環境とGUIエミュレーション環境のための開発

GUIエミュレーションは、Dialog Systemの16ビットバージョンでサポートされています。GUIエミュレーションに移植可能なスクリーンセットを作成するための詳しい情報は、16ビット製品のマニュアルを参照してください。

12.3 移植性に関する一般的指針

ここで示す指針は、主にグラフィカルインタフェース間の移植性に関するものです。スクリーンセットをGUIエミュレーション環境に移植できるようにする場合は、16ビット製品マニュアルの「*グラフィカル環境とGUIエミュレーション環境のための開発*」に記載されている指針を考慮してください。

- 異なる解像度間の移植性については、低解像度環境では、高解像度環境と同じ量の情報を表示できないことに注意する必要があります。たとえば、XGA 画面では VGA 画面より多くの情報を表示できます。このため、スクリーンセットは最も低い解像度で定義する必要があります。これによって、アプリケーションのウィンドウが画面の右端または下端からはみ出すのを防ぐことができます。

  しかし、Dialog Systemでは、ランタイム用に複数の解像度をサポートしています。ヘルプの「*Dsgrunの呼出し*」と「*呼出しインタフェース*」の各トピック、および「*上級テクニック*」の章を参照してください。

- 利用可能な画面空間に気をつけましょう。たとえば、43行のGUIエミュレーション画面では、標準VGAの解像度より多くの情報を保持することができますが、25行の画面ではそれができません。

- 各フォントがすべての環境で利用できるとは限りません。すべてのグラフィカル環境で必ず利用できるのは、

デフォルトフォント、システムモノスペースフォント、システムプロポーショナルフォントだけです。（GUIエミュレーションでは、複数のフォントをサポートしていません。このモードでは、システムモノスペースフォントを使用します。）

- System システムプロポーショナルフォントは、環境によって異なります。これは、静的なテキストと、ボタンなどのオブジェクト内に定義されたテキストの両方に影響を与えます。ボタンは、その中のテキストに合わせて定義されるため特に重要です。これによって、ボタンを作成した環境には関係なく、ボタンテキストが必ず見えます。

  しかし、これはボタンの幅がテキストに合わせて変化するわけではありません。このため、たとえば、VGAで横方向に互いに近づいて配置されたボタンが、XGA で見ると重なって見えることがあります。ボタンのサイズを変更すると、重なりを防ぐことができます。しかし、ボタンテキストがクリップされることがあります。（システムプロポーショナルフォントを使用するのではなく）特定のポイントサイズのタイプフェースを変更するとうまく行く場合があります。

  デフォルトのプッシュボタンの境界は、ボタンの定義された領域を囲んで描かれます。環境によっては（たとえば Motif では）、これが他より強調されます。ここでも、オブジェクトを近づけて配置しないようにするしか解決法はありません。

- 常に[テキストに揃える]（ボタンの属性ダイアログボックス）を使用します。各ボタングループについて、[テキストに揃える]の使用を混ぜないでください。この属性は、グループにあるすべてのボタンについて選択するか、まったく選択しないかのどちらかにします。

  たとえば、同じテキストが入った2つのプッシュボタンがあるとします。両方とも、最初は[テキストに揃える]が選択されています。一方のボタンの[テキストに揃える]の選択を解除した場合、いずれのボタンのサイズの違いも認識できません。両方とも同じサイズです。つぎに、この定義を保存し、別の環境でロードしてみます。ほとんどの場合、サイズが違うことに気付くはずです。実際、選択したテキストによっては、一方のボタンではテキストがクリップされることがあります。

- フォントの移植性。スタイル名が指定されたフォントは、環境間で移植可能です。スタイル名を持つフォントは、まったく異なるフォントを使用するように実行時に再マップすることができます。このようなマップを行うために、Dialog System は、マップ情報が入っているバイナリファイルを探します。これは、フォントサイドファイルです。詳しくは、ヘルプの*「フォントサイドファイル」*を参照してください。

- GUIエミュレーションへの移植に関しては、整列ダイアログボックスの[エミュレーション]を選択します。このオプションはデフォルトとしてds.cfgに設定することができます。ヘルプの「*Dialog Systemの概要*」トピックを参照してください。

12.4 他のクロス環境に関する注意点

3種類のすべてのグラフィカル環境で使用するスクリーンセットを作成する場合、[オプション」メニューの「含める]から[移植性に関する警告]を選択してください。これによって、特定のタイプのオブジェクトを作成する場合、関係

に問題があると、警告が表示されます。

警告が表示された場合も、（それを無視し）原因となったオブジェクトの作成を続けることはできます。しかし、警告で示された環境でスクリーンセットを実行しようとすると、思い通りの表示が行われません。ほとんどの場合、すべてのターゲット環境でサポートされているオブジェクトを使用するように、スクリーンセットの一部を設計しなおすことができます。

特に以下のことに注意してください。

- Windows では、クリップされた子ウィンドウはメニューバーを持つことができません。プレゼンテーションマネージャでは持つことができます。

  Windows では、

- クリップされていないウィンドウは、親ウィンドウが非表示の場合も非表示にはなりません。ダイアログを作成する際は、このことに考慮する必要があります。
- Windows では、クリップされていないウィンドウは、タイトルバーを持つ必要があります。
- Windows では、中央揃えと右寄せが利用できません。
- Windows では、プッシュボタンとスクロールバーの色設定は無効です。
- Windows では、タイトルバー付きのウィンドウは境界を持っていなければなりません。
- Windows では、メッセージボックスは常に移動可能です。
- 読み取り専用の複数行入力フィールドは、Windows 3.0 ではサポートされていませんが、Windows 3.1 ではサポートされています。
- 境界のないプッシュボタンは、Windows ではサポートされていません。

12.5 後方互換性に関する注意点

次の2つの注意点があります。

- ノートブック
- コンテナ

12.5.1 ノートブック

タブコントロールオブジェクトは、Dialog Systemの前バージョンで使用したノートブックコントロールオブジェクトに置き換わるものです。タブコントロールの機能は、ノートブックと似ていますが、次の機能については、ノートブックコントロールで使うことができても、タブコントロールでは使えません。

- 結合

  ノートブックの後方ページのAND演算および結合属性は無視されます。

- マイナータブ

  マイナータブはメジャータブに変換されます。

- タブがないページ

  タブコントロールのすべてのページにはタブがあります。

- タブテキストの整列

  タブに表示されるビットマップとテキストは、常にタブの中央に位置します。

- タブの形

  タブコントロールページのタブの形は固定です。

- ステータスラインのテキスト

  タブコントロールにはステータスラインはありません。

- タブの形

  タブコントロールのタブは常に長方形です。

さらに、Windows 95の基本リリースでは、タブの方向をサポートしていません。しかし、タブの方向をサポートするMicrosoft Internet Explorer3.0以降には、オペレーティングシステムに対するアップグレードが含まれています。また、ノートブックと類似しない点として、タブテキストまたはビットマップが完全にタブの内側に収まらない場合に、表示が不正になることが挙げられます。

Dialog Systemは、前バージョンで作成されたノートブックオブジェクトの属性を保持しています。しかし、これらの属性はもはやサポートされないため、定義用のソフトウェアで編集することができません。

12.5.2 コンテナ

Dialog Systemでのコンテナの実装は、リストビューと呼ばれるWindowsコントロールを使って行われます。このコントロールは、コンテナオブジェクトで使用した機能のサブセットを提供します。

リストビューコントロールの主な制約として、リストビューの最初のカラムにはアイコンだけが含まれるようにして、他のカラムにはテキストだけが含まれるようにします。

また、次のコンテナの機能はサポートされていません。

- 総合タイトル

  リストビューには、総合タイトルはありません。総合タイトルが指定されている場合は、無視されます。

- リストビューカラムのフォーマットは、Windowsによって定義される

  カラムヘッダまたはデータのフォーマットに使われるフラグは、整列、セパレータ、カラム幅の設定を含めて、無視されます。

- Deltaイベントはサポートされない

  Deltaイベントの設定は無視され、Deltaイベントは生成されません。

- 一度作成したカラムは、非表示にはできない

  Dscnr Dialog System拡張機能のScおよびHcファンクションは無効です。

- フォントは設定できない

新規コンテナオブジェクトをアプリケーションに実装する場合は、GUIクラスライブラリのリストビューオブジェクトの使用を考慮しましょう。このオブジェクトは、リストビューオブジェクト全体にわたるコントロールを提供します。

12.6 互換性のまとめ

	環境				
	プレゼンテーションマネージャ	Windows	GUI エミュレーション	Windows NT and 95 (16-bit)	Windows NT and 95 (32-bit)
3-D オブジェクト	x	y	x	y	y
ビットマップ	y	y	y	y	y
チェックボックス	y	y	y	y	y
色	y(9)	y(9)	x	y(9)	y(9)
コンテナ	y	y	x	y	x(11)
ダイアログボックス	y	y	y	y	y
入力フィールド	y	y	y	y	y
グループボックス	y	y	y	y	y

フォント	y(8)	y(8)	x	y(8)	y(8)
アイコン/ビットマップボタン	y	y	x	y	y
リストボックス	y	y	y	y	y
リストビュー	x	x	x	x	y
メニューバー	y	y	y	y	y
メッセージボックス	y	y (4)	y	y (4)	y (4)
複数行入力フィールド	y	y (5)	y	y	y
一次ウィンドウ	y (1)	y (1)	y (1)	y (1)	y (1)
プッシュボタン	y	y (6)	y	y (6)	y (6)
ラジオボタン	y	y	y	y	y
スクロールバー	y	y	y	y	y
二次ウィンドウ(クリップする)	y	y (3)	y	y (3)	y (3)
二次ウィンドウ(クリップしない)	y	y(2)	y	y (2)	y (2)
選択ボックス	y	y	y	y	y
タブコントロール	y	y	x	y	x(10)
タブコントロールページ	x	x	x	x	y(10)
テキスト	y	y	y(7)	y	y
ユーザーコントロール	x	x	x	x	y

注:

- [システム位置] は無効です。
- 黒、青、茶、紺、緑、紫、赤、白、黄の各色は、すべての環境で共通です。

第13章 Panels V2 の使い方

Dialog System は Panels Version 2 という技術を持っています。

Dialog System で Panels V2 を使用すると、次のことが可能になります。

- Dialog System だけの場合と比べ、オブジェクトを細かく制御できます。

  たとえば、Panels V2 呼出しを使うと、システムメニューをオフにすることができます。

- Dialog System がサポートしていないような Panels V2 機能をアクセスすることができます。

  たとえば、定義機能ソフトウェアのラバーバンドテクニックは、Dialog System にはありませんが、Panels V2 では利用できます。

この章では、Panels V2 を使って Dialog System アプリケーションをどのように強化できるかについて説明します。

13.1 Panels V2 の呼出し

Panels V2 呼出し文の例を次に示します。

```
call "PANELS2" using p2-parameter-block
                     p2d-dialog-box-record
                     new-title-buffer
                     attribute-buffer
```

Pan2win は Pan2win を呼出し、Pan2win は適切な Windows API 呼出しを実行します。

Panels V2 が提供するファンクションセットは、使用する環境には関係なくすべて同じです。ただし、各環境が表示機能をサポートしている必要があります。

Dialog System は、この技術に基づいています。

13.2 Dialog System と Panels V2 のイベント

Panels V2 への呼出しを持つ標準的な Dialog System アプリケーションを図 13-1 に示します。

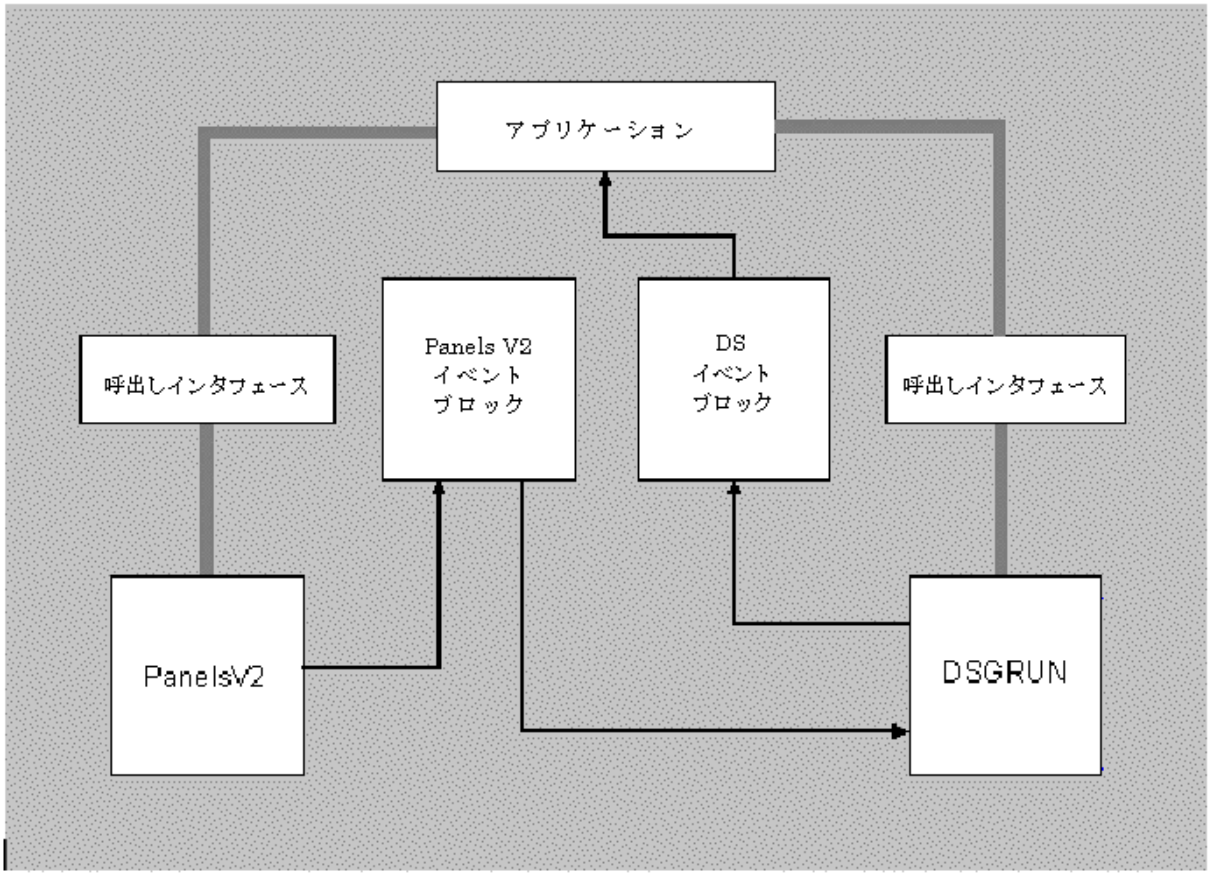


図 13-1 Dialog System/Panels V2 Events

イベントがどのように取り扱われるかに注意してください。Panels V2 とアプリケーションを直接結ぶイベントの流れはありません。イベントが起こると、Panels V2 はそれを検出し、Dialog System（Dsgrun）を通じてイベント情報を返します。Dialog System は、そのイベント情報を Dialog System イベントブロック（dssyinf.cpy）を通じてプログラムに返します。Dialog System またはアプリケーションのどちらで、どのようにイベントを取り扱いたいかを判断するためには、ダイアログを使用しなければなりません。

13.3 COPY ファイル

プログラムの Working-Storage Section には、いくつかの COPY ファイルをいれておかなければなりません。次に例を示します。

```
working-storage section.
    copy "ds-cntrl.mf".
    copy "clip.cpb".
    copy "pan2link.cpy".
    copy "dssysinf.cpy".
```

最初の2つは、いままでにも見てきた Dialog System の制御ブロックとデータブロックで、これらはスクリーンセッ

トから生成されます。これらのファイルについては、ヘルプの*「呼出しインタフェース」*の項を参照してください。

残りの2つの COPY ファイルについては、次の2つの項で説明します。

13.3.1 Panels V2 COPY ファイル（pan2link.cpy）

COBOL プログラムには、pan2link.cpy COPY ファイルもいれておく必要があります。

このファイルには、次のものが入っています。

- レコード定義
- あらかじめ決められたレベル 78 パラメータ定義
- Micro Focus キーリスト

このファイルは、必ず必要なわけではありませんが、このファイルの中の定義によって、アプリケーションの Panels V2 部分の記述が大幅に簡略化できます。

このファイルをざっと見て、内容をつかんでおいてください。

13.3.2 Dialog System イベントブロック（dssysinf.cpy）

このファイルには、Dialog System イベントブロックの定義が入っています。イベントが起こると、そのイベントに関する情報が Panels V2 イベントブロックを通じて Dialog System に渡されます（図 13-1 を参照）。dssysinf.cpy は、Panels V2 イベントブロックに、Dialog System で使用するフィールドを追加したものにすぎません。

```
 01 ds-event-block.
 78 ds-eb-start        value next.
    03 ds-ancestor                       pic 9(9)  comp-5.
    03 ds-descendant                     pic 9(9)  comp-5.
    03 ds-screenset-id                   pic x(8).
    03 ds-event-screenset-details.
       05 ds-event-screenset-id          pic x(8).
       05 ds-event-screenset-instance-no pic 9(2)  comp-x.
    03 ds-event-reserved                 pic x(11).
    03 ds-event-type                     pic 9(4)  comp-5.
    03 ds-event-data.
       05 ds-gadget-event-data.
          07 ds-gadget-type              pic 9(2)  comp-x.
```

```
         07 ds-gadget-command           pic 9(2)  comp-x.
         07 ds-gadget-id                pic 9(4)  comp-5.
         07 ds-gadget-return            pic 9(4)  comp-5.
      05 ds-mouse-event-data.
         07 ds-mouse-x                  pic s9(4) comp-5.
         07 ds-mouse-y                  pic s9(4) comp-5.
         07 ds-mouse-state              pic 9(2)  comp-x.
         07 ds-mouse-moved-flag         pic 9(2)  comp-x.
         07 ds-mouse-over               pic 9(2)  comp-x.
         07 filler                      pic x(3).
 78 ds-eb-size         value next - ds-eb-start.

      05 ds-keyboard-event-data redefines ds-mouse-event-data.
         07 ds-char-1.
            09 ds-byte-1                pic 9(2)  comp-x.
         07 ds-char-2.
            09 ds-byte-2                pic 9(2)  comp-x.
         07 filler                      pic x(8).
      05 ds-window-event-data redefines ds-mouse-event-data.
         07 ds-window-x                 pic s9(9) comp-5.
         07 ds-window-y                 pic s9(9) comp-5.
         07 ds-window-command           pic 9(2)  comp-x.
         07 filler                      pic x.
```

dssysinf.cpy の中で特に注目すべきフィールドは、`ds-descendant` と `ds-ancestor` です。Dialog System がオブジェクトを作成するとき、ハンドルと呼ばれるオブジェクトのための一意の識別子を返します。オブジェクトに関するイベントが起こると、Panels V2 は、オブジェクトのハンドルと、そのオブジェクトの親ウィンドウのハンドルを Dialog System に返します。これら2つのハンドルが `ds-descendant` と `ds-ancestor` で、イベントが起こったときに書き込まれます。

注： dssysinf.cpy を使用した 2.1 のプログラムがある場合、再コンパイルする必要があります。

13.4 Dialog System/Panels V2 アプリケーションの作成

Dialog SystemおよびPanels V2 アプリケーションでは、次の手順に従わなければなりません。

- Dialog System と Panels V2 の協同動作を確立します。
- Dialog System オブジェクトを Panels V2 に対して識別します。
- Panels V2 ファンクションを実行します。

これらの手順について説明するために、次の各項では、プッシュボタンの名前を付け直す簡単な Panels V2 ファンクションの例を示します。これは、ダイアログファンクション（SET- OBJECT-LABEL）を使って行うこともできますが、例では、Panels V2 の呼出しインタフェースの主要なファンクションを示すのを目的にしています。

13.4.1 Dialog System と Panels V2 の通信の確立

COBOL プログラムでは、次のような文によって Dialog System を初期設定します。

```
call "dsgrun" using ds-control-block
                    data-block
                    ds-event-block
```

Dialog System が返す項目の1つに ds-session-id（制御ブロックに格納される）があります。ds-session-id は、Dialog System セッションのスレッド識別子です。アプリケーションが Panels V2 との通信で Dialog System と協同動作するためには、この識別子を Panels V2 に知らせなければなりません。これには、次のような文を使用します。

```
move ds-session-id to p2-mf-reserved
```

p2-mf-reserved は、Panels V2 のパラメータブロックのデータ項目です。

この文は、Dialog System と Panels V2 の間に必要な通信リンクを確立します。これによって、Panels V2 呼出しを直接実行できます。

13.4.2 Dialog System オブジェクトを Panels V2 に対して識別する

MOVE-OBJECT-HANDLE ファンクション を使うと、オブジェクトのハンドルをデータブロックの数値データ項目に保存することができます。これによって、同じオブジェクトが、アプリケーションの Panels V2 部分と Dialog System 部分の両方で参照されます。次に例を示します。

```
MOVE-OBJECT-HANDLE RENAME-PB PB-HAND
MOVE-OBJECT-HANDLE RENAME-WIN WIND-HAND
```

この文は、RENAME-PB というプッシュボタンのハンドルを数値項目 PB-HAND に格納し、RENAME-WIN とい

うウィンドウのハンドルを WIND-HAND に格納します。RENAME-WIN と WIND-HAND は、次のように定義されています。

```
RENAME-HAND           C    4.00
WIND-HAND             C    4.00
```

これで、アプリケーション（そして Panels V2）は、Dialog System のオブジェクトにアクセスできます。

13.4.3 Panels V2 ファンクションの実行

プッシュボタンのタイトルを変更するためのコードを次に示します。

```
 1 rename-button section.
 2
 3   initialize p2-parameter-block
 4   initialize p2g-button-record
 5   move 250                  to p2g-button-text-length
 6   move pb-hand              to p2-descendant
 7   move ds-session-id        to p2-mf-reserved
 8   move pf-get-button-details to p2-function
 9   call "PANELS2" using p2-parameter-block
10                        p2g-button-record
11                        text-buffer
12   end-call
13   perform varying ndx1 from 30 by -1 until ndx1 = 1 or
14             rename-text(ndx1:1) not = " "
15      continue
16   end-perform
17   move ndx1                 to p2g-button-text-length
18   move pf-set-button-details to p2-function
19   call "PANELS2" using p2-parameter-block
20                        p2g-button-record
21                        rename-text
22   end-call.
```

行1:

```
 rename-button section.
```

このセクションは、Panels V2 の呼出しインタフェースを示しています。

行3〜5:

```
    initialize p2-parameter-block
    initialize p2g-button-record
    move 250                to p2g-button-text-length
```

いくつかの Panels V2 パラメータを初期化します。

行6:

```
    move pb-hand             to p2-descendant
```

`pb-hand` は、プッシュボタンのハンドルです。このフィールドは、データブロック内にあります。

行7:

```
    move ds-session-id       to p2-mf-reserved
```

`ds-session-id` は、Dialog System セッションのスレッド識別子です。詳しくは、この章の*「Dialog System と Panels V2 の通信の確立」*の項を参照してください。

行8:

```
    move pf-get-button-details to p2-function
```

`pf-get-button-details` は、プッシュボタンの詳細を検索するための Panels V2 ファンクションです。ここでは、タイトルを変更するだけなので、残りはすべて同じです。このファンクションの詳細については、『Panels V2 リファレンス』の「ファンクション」の章を参照してください。

行9〜12:

```
    call "PANELS2" using p2-parameter-block
                         p2g-button-record
                         text-buffer
    end-call
```

Panels V2 の呼出し文です。

行13〜16:

```
    perform varying ndx1 from 30 by -1 until ndx1 = 1 or
```

```
                    rename-text(ndx1:1) not = " "
        continue
    end-perform
```

この perform 句は、ボタンの新しい名前が入っている rename-text から後続のスペースを取り除きます。

行17:

```
    move ndx1                to p2g-button-text-length
```

後続のスペースを取り除いたあと、ndx1 には、名前の実際の長さが入っています。p2g-button-text-length は、タイトルバッファの長さをいれる Panels V2 パラメータです。

行18:

```
    move pf-set-button-details to p2-function
```

この Panels V2 ファンクションは、プッシュボタンの属性を変更します。このファンクションの詳細については、『Panels V2 リファレンス』の「ファンクション」の章を参照してください。

行19**〜**22:

```
    call "PANELS2" using p2-parameter-block
                         p2g-button-record
                          rename-text
    end-call.
```

別の Panels V2 呼出しです。

警告： Panels V2 によってオブジェクトを作成または削除しないでください。Dialog System は、すべてを管理していることを前提としています。たとえば、Panels V2 がオブジェクトを作成した場合、そのハンドルは Dialog System には渡されません。したがって、そのオブジェクトに関するイベントは Dialog System を避けてしまいます。

13.5 サンプルプログラム

demo ディレクトリには、Dialog System と Panels V2 のインタフェースを示す詳しい例が、ほかにも2つあります。Clip は、Panels V2 のクリップボードに関する基本的な読み取り機能と書き込み機能をどのようにプログラムで使用するかを示しています。もう1つの Dsp2demo は、色の設定、ウィンドウのスクロールなどのクリップボード機能を示しています。

13.6 Panels V2 ユーザーイベント

Panels V2 ユーザーイベントを Dialog System で生成したり、受け取ることができます。詳細は、オンラインヘルプの POST-USER-EVENT ファンクションと GET-USER-EVENT-DATA ファンクションに関する説明を参照してください。最初の 32,000 個のイベントは、Micro Focus による使用のために予約済みです。

第14章 クライアント/サーバー結合の使い方

この章では、クライアント/サーバー結合の原理を説明し、ユーザーのプログラムを汎用クライアント/サーバー モジュールに結合する方法について説明します。

14.1 はじめに

クライアント/サーバー結合は、Micro Focus Common Communications Interface構成要素（CCI）を使って、「ユーザーの手間を煩わされずに」Dialog Systemフロントエンドとのクライアント/サーバーアーキテクチャを実現できます。クライアント/サーバー結合が提供するmfclientとmfserverという2つのモジュールのおかげで、ユーザーはアプリケーションに通信プログラムを組み込む必要がありません。

これらのモジュールは、構成ファイルに入っている情報に基づいて、モジュール間の通信とデータ転送を管理し、このデータを処理するためにリンクの両端でユーザーが定義したプログラムと対話することができます。mfclientは、ユーザーインタフェースを取り扱うユーザークライアントプログラム、mfserverは、データアクセスやビジネスロジックを取り扱うユーザーサーバープログラムを呼び出します。しかし、これらのユーザークライアントとサーバープログラムはユーザー定義で、ユーザーの要求に応じて何でもできます。

ユーザープログラムは、汎用クライアント/サーバーモジュールと対話するために、mfclisrv.cpyというCOPYファイルを使用しなければなりません。このCOPYファイルの説明はドキュメント*cscopy.txt*に入っています。*cscopy.txt*およびクライアント/サーバー結合ファイルの*csref.txt*と*csconfig.txt*は、インストールディレクトリのdocsディレクトリに入っています。

Dialog System CUSTOMERデモンストレーションに基づくサンプル2階層パラメータが、インストールディレクトリのDialogSystem¥demo¥csbindディレクトリに入っています。このサンプルはクライアント/サーバー結合の使い方を示します。このサンプルは、この章の各部で参照します。サンプルアプリケーションについて詳しくは、DialogSystem¥demo¥csbindディレクトリにインストールされた*csbind.txt*ファイルを参照してください。

14.2 クライアント/サーバー結合の内容

この節では、クライアント/サーバー結合を使って作成した2階層アプリケーションがどのように動作するかを説明します。

クライアント/サーバー結合は、サーバー側でmfserverの非専用コピーを実行することで可能になります。mfserverは、取り決めたサーバー名を使用してすべてのクライアントと通信します。mfserverモジュールの機能は、クライアントからの要求を受けて接続を確立したり終了したりするだけなので、最初のデフォルト値のまま使用できます。

このプロセスを図14-1に示します。図に示す情報の流れを以下に説明します。

1. ユーザーのクライアントプログラム（custint）がmfclientを呼び出し（CALL）ます。

2. mfclientは、初めて呼び出されたときに.cfgファイルから構成情報を読み込みます。

3. mfserverが接続要求を受け取ります。

4. mfserverは、各クライアントに対してセカンダリサーバーを生成します。サーバー名は内部生成されますが、もとのサーバー名に数値IDを付加したもの（例、mfserver01）となります。また、構成ファイルにサーバー名を指定しておくこともできます。

5. mfserverはmfclientにセカンダリサーバー名（mfserver01）を送り返し、対話を終了します。

6. mfclientはセカンダリサーバー（mfserver01）に接続し、LNK-PARAM-BLOCK（*クライアント/サーバー結合のCOPYファイル*の項を参照）を通して、構成ファイルから取得したパラメータを引き渡します。

7. mfserver01は、サーバープログラム（custdata）を呼び出し、Linkage Sectionを通してmfclientから受け取ったパラメータを渡します。

8. ユーザーが作成したサーバープログラム（custdata）は初めて呼び出されたので、アプリケーション初期化コードを実行して、プログラムを終了します。制御はmfserver01に戻ります。

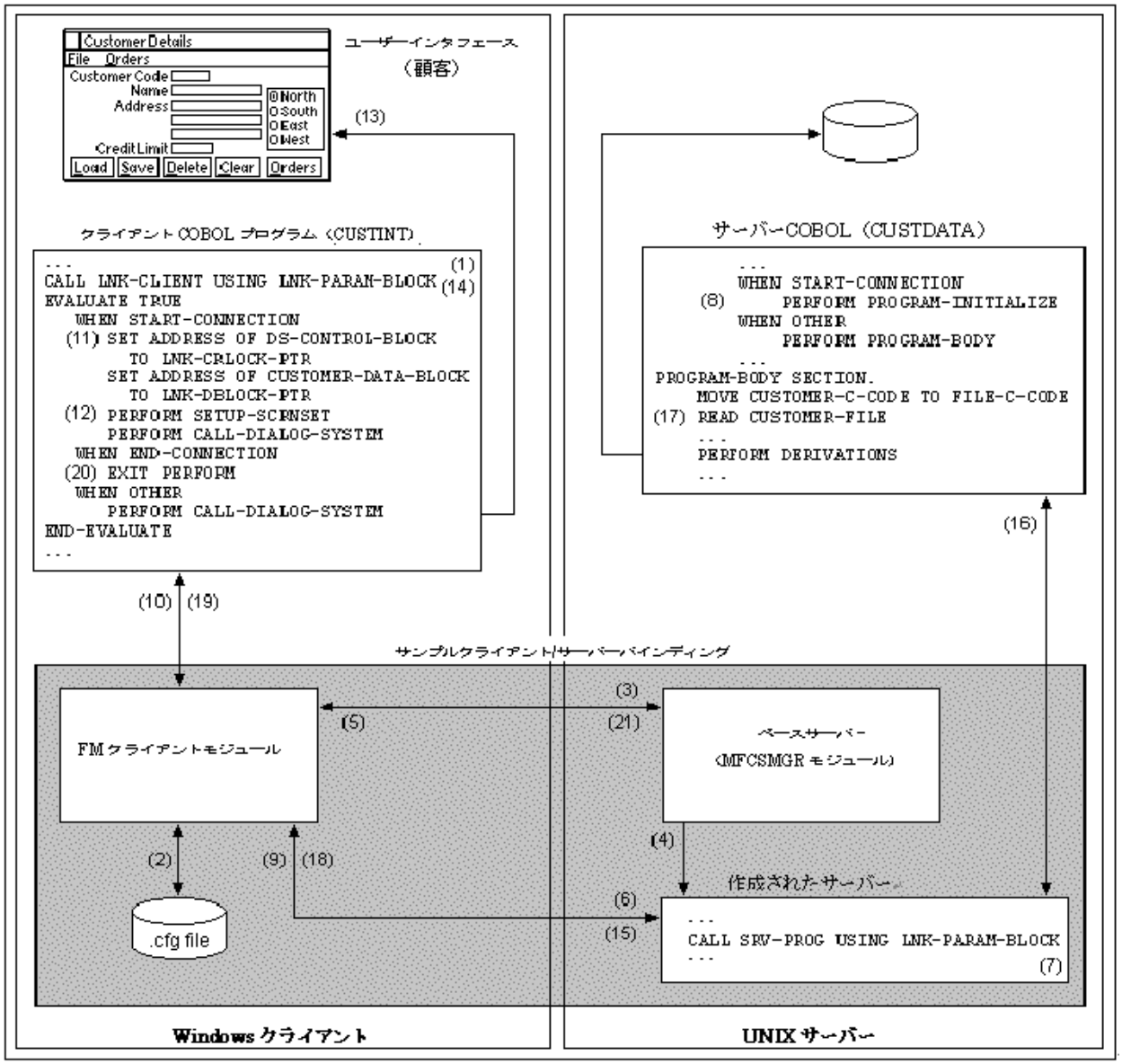


図 14-1 クライアント/サーバー結合

9. mfserver01はmfclientに制御を返します。

10. mfclientは接続が確立したことを確認し、ユーザークライアントプログラム（custint）に制御を渡します。

11. ユーザークライアントプログラム（custint）の中で、ユーザーインタフェースに関連するデータが、Linkage Sectionのmfclientによって割り当てられた領域にマッピングされます。

12. ユーザークライアントプログラム（custint）は、ユーザーインタフェースを呼び出し、スクリーンセットをアップし、Dialog Systemを呼び出し（CALL）します。

13. ユーザーはユーザーインタフェース（CUSTOMER）への入力を終了します。

14. ユーザークライアントプログラムは、mfclientを呼び出す（CALL）して、ユーザーが入力したデータ（例、顧客コード）やアプリケーションサーバープログラム（(custdata)）に必要なその他の情報をLinkage Sectionを通じて返します。

15. mfclientは、内部バッファを通じてmfserver01にデータを渡します。

16. mfserver01はユーザーサーバープログラム（custdata）を呼び出し（CALL）ます。

17. ユーザーサーバープログラム（custdata）は、適切なデータアクセスとビジネスロジックを実行し、Linkage Sectionを通じてmfserver01に結果を返します。

18. mfserver01はmfclientに制御を返します。

19. mfclientは、Linkage Sectionを通じてユーザークライアントプログラム（custint）にデータを送り返します。

20. ユーザークライアントプログラム（custint）は、データを表示し、新しいユーザー入力を受け入れるためにユーザーインタフェースを呼び出し（CALL）します。

    ユーザーがアプリケーションを終了するまで、ステップ14～20を繰り返します。

21. mfclientは、セカンダリサーバー（mfserver01）が終了したことをベースサーバー（mfserver）に知らせます。

    ベースサーバーとセコンダリサーバーを使用することによって、いくつかの問題が解決します。

    - アプリケーションごとに専用のサーバーを用意する必要がありません。セコンダリサーバーには、クライアント構成ファイルからクライアントごとの詳細が提供されます。ユーザーは引き続き複数のサーバーをセットアップできますが、その必要はありません。

    - データアクセスの衝突が起こりません。これは、各ユーザーが自分の実行単位を持つことができ、ファイルの状態が前回アクセスしたときと常に同じになるからです。

    - あるユーザーが時間的制約が厳しい要求を行った場合、そのユーザーだけに遅延が起こります。他のユーザーは引き続き最適応答を受けることができます。時間的制約の厳しい要求を処理すると、クライアントがインタフェースをロックアップすることがあります。これは、次のようにして解決できます。
      プログラムに要求idを返すような非同期要求を発行することができます。そして、返されるidを使って要求が終了したかを定期的にチェックします。非同期要求の例が、この章のサーバーアプリケーションをmfserverに接続するの項に書かれています。

14.3 プログラムを汎用モジュールと接続する

この項では、次の方法について説明します。

- ユーザープログラムを汎用クライアントおよびサーバーモジュールと接続する。

- 通信リンクを準備する。

14.3.1 クライアントアプリケーションをmfclientに接続する

この項では、クライアントプログラムをmfclientモジュールと接続する方法について説明します。このモジュールはサーバーとの通信を取り扱います。mfclientモジュールとmfserverモジュールは、mfclisrv.cpy COPYファイルに記述されたパラメータブロックを通じて互いに情報を渡します。

これらのモジュールは、同じパラメータブロックを使って、構成ファイルで呼び出すように要求された任意のユーザープログラムに情報を渡します。LNK-PARAM-BLOCKについて詳しくは、この章のサーバーアプリケーションをmfserverに接続するの項を参照してください。

ユーザーインタフェースを呼び出すクライアントプログラムと接続するには、mfclientにパラメータを渡すためのコードを追加しなければなりません。

クライアント/サーバー結合を使用するには、クライアントプログラムに次に示すコードを追加する必要があります。Dialog Systemに付属のユーザーインタフェースプログラム（custint.cbl）の一部分を次に示します。この部分は、クライアント/サーバー結合の方法を表しています。

```
$SET ANS85
 WORKING-STORAGE SECTION.
 COPY "MFCLISRV.CPY".
 LINKAGE SECTION.
* 次の2つのCOPYファイルは、インタフェースを呼び出す
* COBOLプログラムと通信するために、Dialog System
* ユーザーインタフェースによって使用されます。
 COPY "DS-CNTRL.V1".
 COPY "CUSTOMER.CPB".
 PROCEDURE DIVISION.
 CLIENT-CONTROL SECTION.
     PERFORM UNTIL END-CONNECTION
         CALL LNK-CLIENT USING LNK-PARAM-BLOCK
         EVALUATE TRUE
          WHEN START-CONNECTION
             SET ADDRESS OF DS-CONTROL-BLOCK TO LNK-CBLOCK-PTR
             SET ADDRESS OF CUSTOMER-DATA-BLOCK TO LNK-DBLOCK-PTR
             PERFORM SETUP-SCRNSET
```

```
            PERFORM CALL-DIALOG-SYSTEM
          WHEN END-CONNECTION
            EXIT PERFORM
          WHEN OTHER
            PERFORM CALL-DIALOG-SYSTEM
        END-EVALUATE
        IF CUSTOMER-EXIT-FLG-TRUE
          SET CLIENT-ENDING TO TRUE
        END-IF
    END-PERFORM.
CLIENT-CONTROL-END.
    STOP RUN.
```

クライアントカウントを可能にしたり、エラーメッセージ表示を自分で取り扱ったり、非同期要求を取り扱うためのコードを追加できます。

14.3.2 サーバーアプリケーションをmfserverに接続する

この項では、サーバープログラムを接続する方法について説明します。サーバープログラムは、クライアントとの通信を取り扱うmfserverモジュールに対してデータアクセスやビジネスロジックを実行します。mfclientモジュールとmfserverモジュールは、mfclisrv.cpy COPYファイルに説明されたパラメータブロックを通じて互いに情報を渡します。また、これらのモジュールは、同じパラメータブロックを使って、構成ファイルで呼び出すように要求された任意のユーザープログラムに情報を渡します。

次の説明は、既存のアプリケーションをサーバープログラムとして使用する場合は該当しません。このような方法でのクライアント/サーバー結合の使い方について詳しくは、この章のクライアント/サーバー結合アプリケーションの実行の項を参照してください。
クライアント/サーバー結合を使用するには、

- サーバープログラムのLinkage Sectionには、結合COPYファイルmfclisrv.cpy、およびクライアントプログラムとユーザーインタフェースの間で情報を渡すのに必要なCOPYファイルをインクルードする必要があります。
- Procedure Division見出しを変更して"LNK-PARAM-BLOCK"を含めます。これはmfserverからパラメータを渡します。
- mfclisrv.cpyでmfserverによって渡されるアドレスをユーザーインタフェースによって使用されるCOPYファイルと関連付けます。この例では、DS-CONTROL-BLOCKとSCREENSET-DATA-BLOCKが、ds-cntrl.cpyとcustomer.cpbで定義されたデータ構造です。LNK-PARAM-BLOCKについて詳しく

は、ディスク上のドキュメント*cscopy.txt*を参照します。

クライアント/サーバー結合の実行に使用されるサーバープログラム（custdata.cbl）の該当するコード部分を次に示します。

```
 LINKAGE SECTION.
 COPY "DS-CNTRL.V1".
 COPY "CUSTOMER.CPB".
 COPY "MFCLISRV.CPY".
 PROCEDURE DIVISION USING LNK-PARAM-BLOCK.
 CONTROLLING SECTION.
*----------------------------------------------------------------*
*  DIALOG SYSTEMのCOPYBOOKをLNK-PARAM-BLOCKで
*  予約された領域と関連付ける
*----------------------------------------------------------------*
    SET ADDRESS OF DS-CONTROL-BLOCK TO LNK-CBLOCK-PTR.
    SET ADDRESS OF CUSTOMER-DATA-BLOCK TO LNK-DBLOCK-PTR.
    EVALUATE TRUE
     WHEN START-CONNECTION
        PERFORM PROGRAM-INITIALIZE
     WHEN OTHER
        PERFORM PROGRAM-BODY
    END-EVALUATE.
    EXIT PROGRAM.
 PROGRAM-INITIALIZE SECTION.
    OPEN I-O CUSTOMER-FILE.
 PROGRAM-BODY SECTION.
    MOVE CUSTOMER-C-CODE TO FILE-C-CODE
    READ CUSTOMER-FILE
    ..........
    PERFORM DERIVATIONS
```

エラーメッセージの表示を取り扱うためのコードを追加できます。

14.3.3 通信リンクの準備

クライアント/サーバー結合は、構成ファイルの内容に基づいてクライアントプログラムとサーバープログラムの間の通信を制御します。このためユーザーが複雑な通信プログラムを作成する必要がありません。

デモンストレーションを実行するには、適切な構成（.cfg）ファイルを変更せずに使用します。自分のアプリケーションの場合、構成ファイルの内容をコピーし、変更しなければなりません。構成ファイルは、次のような主要情報を指定するためのパラメータが入ったASCIIテキストファイルです。

- mfserverによって呼び出されるユーザープログラムの名前
- サーバーと接続できるクライアントの最大数

クライアント/サーバー結合に関する構成ファイルの使い方について詳しくは、この章の 構成ファイルのエントリ の項を参照してください。

注意：

- ここでは、CCIを使った通信リンクがあらかじめ正しくインストールおよび構成され、クライアントマシンとサーバーマシンの両方で動作していることを前提としています。

14.4 クライアント/サーバー結合を使用するまえに

クライアント/サーバー結合は、既存のスタンドアロンアプリケーションまたはクライアント/サーバー対応アーキテクチャのアプリケーションといっしょに使用できます。既存のスタンドアロンアプリケーションの使い方について詳しくは、*クライアント/サーバー結合アプリケーションの実行*を参照してください。

いずれの場合も、アプリケーションには次のプログラムがあります（Dialog Systemに付属のデモンストレーションプログラムの例を示す）。

	2階層デモプログラム	スタンドアロンデモプログラム
ユーザーインタフェース	CUSTOMER.gs (GUI)	
インタフェースを呼び出すためのCOBOLコード	custint.cbl	usexsrv.cbl
データアクセスを実行し、ビジネスロジックを適用するためのCOBOLコード	custdata.cbl	customer.cbl

ユーザーのミドルウェアコードを作成するかわりにクライアント/サーバー結合を使用できます。これには、次の手順が必要です。

- mfclientモジュールとmfserverモジュールと通信接続の動作を制御するための構成（.cfg）ファイルを作成します。

  構成ファイルは、次のような主要情報を指定できるパラメータが入ったASCIIテキストファイルです。

  - 使用する通信プロトコル
  - mfserverによって呼び出すユーザープログラムの名前

  この構成ファイルの形式については、*クライアント/サーバー結合構成ファイル*で詳しく説明します。サンプル構成ファイル（custgui.cfgとcustomer.cfg）が、クライアント/サーバー結合デモンストレーションプログラムといっしょに用意されています。

  アプリケーションごとに別の構成ファイルを作成することができ、また1つの構成ファイルに複数のエントリを入れることもできます。複数のエントリを入れる場合は、ファイル内で各アプリケーション固有のタグを指定し、その後にそのアプリケーションのパラメータリストを続けます。クライアントプログラムでは、`lnk-tagname`にこのタグ名を指定しなければなりません。この方法を使うと、デフォルトの構成ファイルmfclisrv.cfg内に複数エントリを指定することで、コマンドラインが長くならずに済みます。また、この方法を使用して、1つのメニューから複数のアプリケーションを駆動することもできます。

- mfclientモジュールをリンケージパラメータといっしょに呼び出すためのコードをクライアントプログラムに追加します。これについては、*クライアントアプリケーションをmfclientに接続する*の項を参照してください。

- mfserverモジュールからリンケージパラメータといっしょにサーバープログラムを呼び出すためのコードをサーバープログラムに追加します。これについては、*サーバーアプリケーションをmfserverに接続する*の項を参照してください。

14.4.1 mfclisrv.cpy COPY**ファイル**

mfclientとmfserverの2つのモジュールは、mfclisrv.cpyというCOPYファイルに定義されたパラメータブロックを通して情報の受け渡しを行います。情報には、Dialog System の`screensetdata-block`や`ds-control-block`などが入っています。2つのモジュールは、構成ファイルで呼び出すように要求されたユーザープログラムに情報を渡すのにも、このパラメータブロックを使用します。mfclisrv.cpy COPYファイルは、mfclientとmfserverの2つのモジュールを使用するすべてのCOBOLプログラムでインクルードしなければなりません。

mfclisrv.cpy COPYファイルは、NetExpress基本インストールディレクトリのSOURCEディレクトリにインストールされています。

14.4.2 **クライアント/サーバー結合の構成ファイル**

この項では、クライアント/サーバー結合の構成ファイルについて説明します。この構成ファイルは、mfclientとmfserverの2つのモジュールの動作と通信リンクを制御します。

構成ファイルは、アプリケーションごとに1つまたは複数作成します。アプリケーションに必要な任意の数の構成ファイルを作成できますが、各ファイルは拡張子が.cfgでなければなりません。構成ファイルの名前を指定しない場合、デフォルトのmfclisrv.cfgという名前になります。構成ファイルは必須ではありませんが、このファイルがない場合、クライアント/サーバー結合といっしょに提供されるデモンストレーションプログラムがデフォルトとして実行されます。

クライアント/サーバーアプリケーションの設定や構成ファイルのエントリに誤りがあれば、それらを読み込んだプログラムはエラー内容を詳しく示すメッセージを画面に表示して終了します。

このようなエラーは、mfclisrv.logファイルにも記録されます。このファイルはカレントディレクトリに作成されるか、MFLOGDIR環境変数で指定されたディレクトリに作成されます。このため、端末が直接つながっていないサーバーでもメッセージを記録できます。

14.4.2.1 構成ファイルのエントリ

次にmfclisrv.cfg構成ファイルのエントリ一覧をアルファベット順に示します。構成ファイルに指定されていないエントリについてはデフォルト値が使用されます。

```
************************************************************
* Micro Focus - クライアント/サーバーモジュールの構成ファイル
************************************************************
```

[mf-clisrv]

cblksize=*nnnn* PIC X(4) COMP-X [default:0]

Dialog Systemの制御ブロックのサイズ。

clierrprog=*xxxxxx* PIC X(128) [default:none]

mfclientに代わって通信エラーを取り扱うプログラムの名前。mfclientエラーを呼び出し側プログラムで取り扱うようにするには、SAME と指定します。

commsapi=*xxxx* PIC X(4) [default:CCI]

通信API（有効値はCCIまたはNONE）。通信製品を一切使わずにPC単独でテストを行えるような2階層アプリケーションを作成するには、特殊エントリのNONEを指定します。

mfclientと通信要求がバイパスされ、データはmfclientとユーザー定義のサーバープログラムの間で直接送受信されます。

注意：ユーザー定義のサーバープログラムとして既存アプリケーションを使用している場合は、このオプションを指定できません。

compress=*nnn* PIC 9(3) [default:000]

圧縮ルーチン番号。番号は、使用する圧縮ルーチンの名前を示します。001ならばCBLDC001というルーチンが使用されます。0と指定するとデータ圧縮が行われません。

dblksize=*nnnn* PIC X(4) COMP-X [default:0]

ユーザーデータブロックのサイズ。Dialog Systemで結合モジュールを使用する場合、このエントリは、Dialog Systemによって生成されるデータブロックのサイズより大きくなります。スクリーンセットが使用するデータフィールドが変化し、COPYファイルが再生成された場合、新しいデータブロックのサイズに合わせてこのエントリを変更します。複数のスクリーンセットを使用する場合、このエントリは最も大きなスクリーンセットのデータブロックのサイズにします。

注意：このエントリが、実際に使用しているデータブロックのサイズより小さい場合、予期できない結果が生ずる可能性があります。

eblksize=*nnnn* PIC X(4) COMP-X [default:0]

オプションのDialog Systemイベントブロックのサイズ。

machinename= PIC X(34)

servernameエントリで指定したサーバーが存在するマシンの名前。これを指定することで、システムが定義された名前と一致する最初のサーバーを検索するのを避けられます。

maxtrans=*xxxx* PIC 99 [default:0]

転送パッケージの最大サイズをキロバイトで指定します。バッファ全体の大きさがこの値を超えると、システムはバッファを'maxtrans'サイズずつ分割して転送します。有効な値は1～62です。

midconfig=*xxxxx* PIC X(128) [default:none]

クロス階層ソリューションの一環として、mfclientがmfserverに呼び出されたときに使用される構成ファイルの名前。このエントリを指定すると、mfserverがルータのように機能し、ローカルなmfclientモジュールにデータを渡します。

mfclientモジュールはこの構成ファイルを使用して、別のサーバーを探して通信します。この方法を使用すると、マシンごとにプロトコルと通信APIを変えることができます。

protocol=*xxxxxx* PIC X(8) [default:CCITC32]

使用するCCIプロトコル。このエントリは、commsapiがCCIに設定されているときのみ有効です。次のCCIプロトコルを指定できます。

- Novel IPX（CCIIX32と指定）

- NetBEUI（CCINB32と指定）
- 動的データ交換（CCIDE32と指定）
- TCP/IP（CCITC32と指定）

commsapiが CCI（デフォルト）に設定されているときにプロトコルを指定しなかった場合、クライアント/サーバー結合でデフォルトのTCP/IP（CCITC32）が使用されることに注意してください。

scrntype=*xxxx* PIC X(4) [default:none]

複数のインタフェースタイプがサポートされている場合の必須インタフェース。キャラクタ制御ブロックを使ってGUIとキャラクタインタフェースの両方を実行している場合、scrntypeはクライアント/サーバー結合モジュールによって確認されません。

servername=*xxx* PIC X(14) [default:MFCLISRV]

通信で使用するサーバー名。

setenv=name value PIC X(148) [default:none]

どのサーバープログラムの実行よりもまえに設定する環境変数。フォーマットは次のようになっています。

variable name PIC X(20) variable value PIC X(128)

名前と値のフィールドは1つ以上の空白で区切ります。setenvエントリは9個まで指定できます。

srvanim=*x* PIC X(16) [default:N]

YまたはN。このパラメータをYにすると、サーバー上でユーザープログラムのアニメーション表示が可能になります。すなわち、mfserverを一旦停止して、COBSW=+A として再起動しなくても動的に設定されます。

UNIXシステムの場合は、x,*filename*と指定して、ユーザープログラムのクロスセッションアニメーションを可能にできます。詳しくは、*アプリケーションのアニメート*の項を参照してください。

srverrprog=*xxxxxx* PIC X(128) [default:none]

mfserverに代わって通信エラーを取り扱うプログラムの名前。mfserverエラーのためにsrvprogエントリで指定したのと同じものにするには、SAMEと指定します。

srvprog=*xxxxxx* PIC X(128) [default:custdata]

mfserverが呼び出すユーザー定義プログラムの名前。

srvtier=*xxxxxx* PIC X(128) [default:mfserver]

サーバー階層プログラムの名前。これが指定されると、mfserverはこのプログラムから呼び出されます。

subserver=*xxxx* PIC X(14) [default:none]

メインサーバーとの最初のコンタクトのあと、初期サーバー名に基づく名前の代わりに通信に使用されるサーバーのベース名。たとえば、デフォルトのサーバー名はMFCLISRVなのでそのままだとサブサーバー名はMFCLISRV00001、MFCLISRV00002、...となります。このパラメータをNEWSERVと指定すると、サブサーバー名はNEWSERV00001、NEWSERV00002、...となります。このパラメータを使用すると、ベースサーバー名を変えずにアプリケーション固有のサブサーバー名を使用できます。

timeout=*nnnnn* PIC X(4) COMP-5 [default: 120 secs)

システムのデフォルトタイムアウト値の代わりに使用する値。タイムアウト値は 1/10秒単位で指定します。すでに呼び出されているものについては、呼び出された時点で有効だったタイムアウト値がそのまま使用されます。

ublksize=*nnnn* PIC X(4) COMP-X [default:0]

オプションのユーザーデータブロックのサイズ。

useraudit=*x* PIC X [default:N]

クライアントの接続と切断の詳細をログに記録するかどうかを切り換えます。Yに設定した場合、次の詳細がクライアントとサーバーの両方に記録されます。

日付、時刻、接続/切断インジケータ、サーバー名、マシン名、プロトコル

14.4.2.2 構成ファイルで最低限必要なエントリ

構成ファイルで最低限必要なエントリは、次の要因によって異なります。

- アプリケーション（例、Dialog System）
- Dialog Systemのバージョン（Dialog Systemの制御ブロックのサイズが必要なため）
- Dialog Systemの機能（例、callouts）
- サーバーの数
- 通信プロトコル（例、CCITCP）

Dialogアプリケーションが既存のDialog Systemプログラムをサーバーとして使用する場合と使用しない場合について、

必要なエントリを次の表に示します。

	DSプログラムをサーバーとして使用しない	DSプログラムをサーバーとして使用する
dblksize	データブロックのサイズ	データブロックのサイズ
srvprog	mfserverがサーバー側処理を実行するために呼び出すプログラムの名前これは、サーバーマシン上のユーザープログラム（データアクセスとビジネスロジックを実行するCOBOLプログラム）です。	
srvtier		ベースサーバーが生成するプログラムの名前。デフォルトはmfserverです。これは、既存のDialog Systemプログラムをサーバーとして使用する場合にのみ変更します。この既存のプログラムも、データアクセスとビジネスロジックを実行するCOBOLコードを含んでいます。
Cblksize	Dialog Systemの制御ブロックのサイズ（ds-cntrl.r1 copybookを使用しない場合）。	Dialog Systemの制御ブロックのサイズ（ds-cntr.r1 copybookを使用しない場合）。
Protocol		使用するプロトコル。たとえば、TCPの場合はCCITC32。

複数サーバーを実行する場合は、servernameエントリを通じて使用されるサーバーの名前を指定しなければなりません。

未設定のエントリにはデフォルト値が使用されます。

TCP/IPを使用しない場合、プロトコルエントリを設定しなければなりません。

14.4.2.3 構成ファイルを置く場所

構成ファイルを置く場所はユーザーが指定できます。mfclientとmfserverの2つのプログラムは、構成ファイルを次の方法で見つけ出します。

1. MFCSCFG環境変数にファイル名が設定されているか調べます。
2. コマンド行の中にファイル名が指定されていれば、MFCSCFG環境変数で指定されたものより優先します。
3. MFCSCFGが設定されていないか、値がなく、しかもコマンド行でファイル名が指定されていない場合、デフォルト名の構成ファイルmfclisrv.cfgをカレントディレクトリから探します。

4. mfclisrv.cfgが見つからなければ、前の方で説明した構成ファイルの各エントリに対するデフォルト値が使用されます。

構成ファイル名をどの方法で指定する場合でも、場所と名前を合わせて128文字までの完全パス名で指定できます。

14.5 mfclientに対するユーザーのクライアントプログラムの接続

クライアント/サーバー結合を使ったプログラムを作成するには、クライアントとサーバーのプログラムが接続を正しく制御できるように次の示すようなコードを追加します。必要なコードはわずかです。コードには、動作の理解に役立つようにコメントが付いています。

```
WORKING-STORAGE SECTION.
COPY "mfclisrv.cpy".
LINKAGE SECTION.
COPY "DS-CNTRL.V1".
COPY "CUSTOMER.CPB".
PROCEDURE DIVISION.
Client-Control SECTION.
*  メインループは、サーバーとの接続が終了するまで
*  繰り返される
    PERFORM UNTIL End-Connection
* 'lnk-client'は名前'mfclient'を保持する。
*  初回はシステムを初期設定し、
*  サーバーとの接続を確立する
    CALL lnk-client USING lnk-param-block
    EVALUATE TRUE
      WHEN start-connection
*  'mfclient'によって割り当てられたアドレスをDSのコントロールブロックと
*  データブロックに指定することによって、それらをアクセス可能にする。
           SET ADDRESS OF ds-control-block TO lnkcblock-ptr
           SET ADDRESS OF customer-data-block TO lnkdblock-ptr
*  サーバーとの接続が確立したら、
*  ローカル初期設定を終了し、
*  Dialog Systemの最初の呼び出しを行う。
```

```
            PERFORM Setup-Scrnset
            PERFORM Call-Dialog-System
          WHEN end-connection
            EXIT PERFORM
          WHEN OTHER
            PERFORM Call-Dialog-System
        END-EVALUATE
```

* **ユーザーが画面上で終了フラグをセットしているかをチェックする**

```
        IF customer-exit-flg-true
          SET client-ending TO TRUE
        END-IF
      END-PERFORM
      STOP RUN.
```

注意：クライアントプログラムはリンケージにDialog System copybookを持っており、それを結合によって与えられるアドレスにマッピングします。このcopybookには、ほかにマッピングされていない3つの01レベル項目があります。これらは、Dialog Systemのバージョン番号とデータブロックのバージョン番号です。これらがマッピングされないのは、マッピングによって値がピックアップされないことと、そこに入っている必要があるからです。Dialog Systemのバージョン番号は、ほとんど変化しないのでプログラムにハードコーディングできます。しかし、データブロックのバージョン番号はときどき変化するので、プログラムに固定値をコーディングした場合、スクリーンセットが変更されるたびに更新しなければなりません。これを簡単にするために、結合で使用するサポートプログラムを使って、スクリーンセットのバージョン番号を抽出し、動的に設定することができます。このプログラムを使用するには、次の項目をWorking Storageに追加します。

```
01  ws-null           PIC X(4) COMP-X.
01  ws-scrnset-ver    PIC X(4) COMP-X.
01  ws-ret-stat       PIC X COMP-X VALUE 0.
```

スクリーンセットの詳細がセットアップされるProcedure Divisionに次のコードを追加します。使用するスクリーンセット名は、正しい拡張子を含んでいなければなりません。拡張子.rsに合わせて`ds-version-no`を3に設定します。

```
      MOVE "CUSTOMER.gs" TO ds-set-name
      MOVE 2 TO ds-version-no
```

```
     CALL "dsdblksz" USING
          ds-set-name
          ws-null
          ws-scrnset-ver
          ws-ret-stat
     END-CALL
     MOVE ws-scrnset-ver to ds-data-block-version-no
```

1つのアプリケーションのクライアント数を制御したりエラーメッセージの表示を自分のプログラムで行いたい場合は、ユーザープログラムの最初のEVALUATE文に次のようなコードを追加する必要があります。

```
     WHEN TOO-MANY-CLIENTS
        PERFORM OVER-CLIENT-LIMIT
     WHEN COMMS-ERROR
        PERFORM SHOW-ERROR
...
  OVER-CLIENT-LIMIT SECTION.
     DISPLAY SPACES AT 0101 WITH BACKGROUND-COLOR 7
       "MAXIMUM NUMBER OF CLIENTS EXCEEDED - SESSION ENDED"
       AT 1012 WITH FOREGROUND-COLOR 4
       SET CUSTOMER-EXIT-FLG-TRUE
           CLIENT-ENDING TO TRUE
     EXIT.
  SHOW-ERROR SECTION.
     DISPLAY LNK-ERROR-LOC AT 2201
        LNK-ERROR-MSG AT 2301 WITH SIZE LNK-ERROR-MSG-LEN.
  SHOW-ERROR-EXIT.
     EXIT.
```

非同期要求を処理する場合は、EVALUATE文に次のようなコードを追加します。

```
     WHEN START-CONNECTION
        PERFORM GET-USER-INPUT
        IF MAKE-ASYNC-REQUEST  <* USER ASYNCHRONOUS OPTION
          SET ASYNC-REQUEST TO TRUE
```

```
        END-IF
      WHEN ASYNC-OK
        SET TEST-ASYNC-RESULT TO TRUE
        PERFORM DELAY-LOOP
      WHEN ASYNC-INCOMPLETE
        DISPLAY "REQUEST STILL BEING PROCESSED " AT 1010
        PERFORM DELAY-LOOP
        SET TEST-ASYNC-RESULT TO TRUE
      WHEN RESULT-OK
        DISPLAY "REQUEST COMPLETED             " AT 1010
        PERFORM GET-USER-INPUT
      WHEN ASYNC-NOT-STARTED
      WHEN ASYNC-FAILED
        DISPLAY "ASYNCHRONOUS REQUEST FAILURE  " AT 1010
        PERFORM SHOW-ERROR
        PERFORM GET-USER-INPUT
      WHEN COMMS-ERROR
        PERFORM SHOW-ERROR
```

14.6 mfserverに対するユーザーのサーバープログラムの接続

```
WORKING-STORAGE SECTION.
LINKAGE SECTION.
    COPY "DS-CNTRL.V1".
    COPY "CUSTOMER.CPB".
    COPY "mfclisrv.cpy".
PROCEDURE DIVISION USING lnk-param-block.
Controlling SECTION.
*---------------------------------------------------------------*
*  Dialog SystemのcopybookをCS結合パラメータ
*  ブロック内の場所と関連付ける
*---------------------------------------------------------------*
    SET ADDRESS OF ds-control-block TO lnk-cblock-ptr.
```

```
SET ADDRESS OF customer-data-block TO lnk-dblock-ptr.
EVALUATE TRUE
  WHEN start-connection
     PERFORM Program-Initialize
  WHEN customer-exit-flg-true
     PERFORM Program-Terminate
  WHEN OTHER
     PERFORM Program-Body
END-EVALUATE.
EXIT PROGRAM.
```

エラーメッセージの表示を自分のプログラムで行う場合は、プログラムの最初のEVALUATE文に次のようなコードを追加します。

```
WHEN COMMS-ERROR
   PERFORM SHOW-ERROR
   SHOW-ERROR SECTION.
      DISPLAY LNK-ERROR-LOC AT 2201
         LNK-ERROR-MSG AT 2301 WITH SIZE LNK-ERROR-MSG-LEN.
   SHOW-ERROR-EXIT.
      EXIT.
```

14.7 クライアント/サーバー結合アプリケーションの実行

mfserverは.intコード形式で提供されているので、そのまま実行するか、.gntコードを生成するか、他のプログラムとリンクして実行可能モジュールを作成する（UNIXのみ）ことができます。

mfclientとmfserverの各モジュールは、それぞれクライアントとサーバーの各コンポーネントを準備して標準のCOBOLプログラムとして手動で実行します。

既存のスタンドアロンプログラムをサーバーとして実行する場合、その名前をsrvtier構成エントリの中でベースサーバーが実行すべきプログラムとして指定します（srvtier）。通信は、dsgrunとリネームされ、次のように作成されたmfserverモジュールのコピーによって取り扱われます。

- 非UNIXサーバーの場合、mfserver.intファイルをDSGRUN.intにコピーしなければなりません。
- UNIXの場合、プログラムを別に作成するかわりにmfserver.intファイルをDSGRUN.intとリンクできます。UNIXでは、次のエントリが入ったcobconfigファイルも作成し、

```
set program_search_order=3
```

このファイルを指し示すようにCOBCONFIG環境変数を設定しなければなりません。これによって、標準アプリケーションでDSGRUNが呼び出された場合、リネームされたmfserverモジュールが呼び出されます。

cobconfigファイルと環境変数について詳しくは、UNIX COBOLのマニュアルを参照してください。実行可能モジュールのリンクと構築について詳しくは、COBOL開発環境のマニュアルを参照してください。

既存のプログラムをサーバーとして使用することの最大の利点は、単一PCまたはPCネットワークで実行されるアプリケーションが利用でき、結合を使い、単一のユーザーインタフェースプログラムを用意するだけで、クライアント/サーバーアプリケーションとして展開できることです。

クライアント/サーバー結合を実行するには、サーバープログラムを起動してから、クライアントプログラムを起動します。

UNIXの場合は次のようなコマンドラインでサーバー プログラムを起動します。

```
cobrun mfserver [-b] [-p protocol] [-s server-name] [-v]
```

WindowsとOS/2の場合は次のようになります。

```
runw mfserver [-p protocol] [-s server-name] [-v]
```

ここで、

-b（UNIXのみ） は、サーバーをバックグラウンドプロセスとして実行します。コマンドラインの終わりにはアンパサンド（&）文字を付ける必要があります。サーバーからのエラー メッセージはログファイル（mfclisrv.log）に書き込まれます。

-p *protocol* 通信プログラムを指定します。

-s *server-name* デフォルトserver-nameのMFCLISRV以外のserver-nameを指定します。

-v mfclientのバージョン番号を表示します。

クライアントプログラムを起動するコマンドラインは次のようになります。

```
runw program-name [config-filename]
```

ここで、

program-name ユーザーインタフェースプログラムの名前を指定します。

config-filename 使用する構成ファイルの名前を指定します。

クライアントは、最初にMFCSCFG環境変数によって指定された名前、つぎにコマンド行で指定された名前を使って構成ファイルを探します。どちらの名前もない場合、mfclisrv.cfgという構成ファ

イルを探します（この場合、この名前のファイルを作成しておかなければなりません）。

これらおよびその他の構成ファイルエントリについて、ならびに構成ファイルの検索方法について詳しくは、*クライアント/サーバー結合の構成ファイル*の項を参照してください。

COBOLプログラムの実行について詳しくは、COBOLシステムのマニュアルを参照してください。

14.8 アプリケーションのアニメート

NetExpressの標準アニメート機能を使用して、クライアントプログラムをアニメートできます。

サーバープログラムをアニメーションにする場合は、構成ファイルでsrvanim=yというパラメータ設定を行います（*構成ファイルのエントリ*の項を参照）。

PCサーバーの場合、srvanim=yの設定になっていれば、クライアントがサーバーに接続すると自動的にAnimatorが起動します。

UNIXサーバーの場合は、mfserverを起動した端末でAnimatorが実行されるので、mfserverがフォアグラウンドモードで実行されていなければなりません。mfserverはバックグラウンドプロセスとして実行されるのが望ましいので、これは問題です。さらに同時に一人のユーザーしかmfserverからアニメーションを表示できないという制約があります。この問題を解決するには、srvanim=x,*filename*と設定します。ここで、*filename*はtouchコマンドで作成したファイルです。Animatorの出力を表示する端末では、COBANIM_2=animatorと設定してから、次のコマンドを実行します。

```
anim filename
```

標準入出力として使用したい端末では、構成ファイルにsrvanim=x,*filename*というパラメータ設定を追加してから、通常の方法でアプリケーションを実行します。

14.9 サーバーの管理

mfcsmgrプログラムを実行して必要なパラメータと値を渡すことで、サーバーの動作を一時的に変更できます。パラメータは次の2つのグループに分けられます。

- ロケーション

  このグループのパラメータは、ターゲットサーバーを指定するためのもので、m、p、sの設定があります。

- アクション

  これらのパラメータはターゲットサーバーの動作に関するもので、a、c、o、r、tの設定があります。パラメータo、r、tは単独で指定しなければなりません。これらを他のパラメータといっしょに指定すると、エラーメッセージが表示されます。

14.9.1 mfserverのシャットダウン

生成されたサーバーは、クライアントが正常終了するときにクライアントプログラムによって終了されます。

最初のサーバープログラムは、クライアントがなくなっても実行を続けるため、手動で終了しなければなりません。

14.9.2 許可パスワードの管理

アクティブなサーバーを誤ってシャットダウンしてしまうことがないように各サーバーにパスワードを割り当てることができます。このパスワードは、サーバーを終了したり、サーバーのパラメータを変更するときに入力しなければなりません。

14.9.3 最大クライアント数の設定

サーバーがサポートするクライアントの数はデフォルトで65535ですが、mfcsmgr機能を使って下限を設定することができます。指定した値はパスワードファイルに保存され、サーバーが起動するたびに使用されます。

14.9.4 サーバーオーバライドの実行

サーバーは、server-name、protocol、machine-nameというオーバライドオプションをサポートしています。サーバーは構成ファイルを使用しないので、オーバライドパラメータはmfcsmgrプログラムを使って指定します。

このプログラムを実行すると、指定された機能を処理するためにクライアントとサーバーの間で短いやり取りが起こります。

mfcsmgrのコマンドライン構文を次に示します。

WindowsまたはOS/2の場合

```
run mfcsmgr [-a] [-c nnnnn] [-d] [-i filename] [-m machine-name]
            [-p protocol] [-s server-name] [-o m,machinename]
            [-o p,protocol] [-o r] [-o s,servername] [-t] [-v]
```

UNIXの場合

```
cobrun mfcsmgr  [-a] [-c nnnnn] [-d] [-i filename] [-m machine-name]
                [-p protocol] [-s server-name] [-o m,machinename]
                [-o p,protocol] [-o r] [-o s,servername] [-t] [-v]
```

ここで、

-a	ターゲットサーバーの許可パスワードを変更します。
-c *nnnnn*	サーバーがサポート可能なクライアント数を設定します。
-d	クライアントの終了時に、ローカルなオーバライドファイルを削除します。
-i *filename*	クライアントの終了時に、指定のファイルをクライアントのローカルディレクトリにインストールします。
-m *machine-name*	ターゲットサーバーを実行しているマシン名を指定します。この指定は、複数のプラットフォームに同名のサーバーが存在する場合に必要になります。
-p *protocol*	使用する通信プロトコルを指定します（例、CCITCPやCCITC32）。
-s *server-name*	デフォルト（MFCLISRV）以外のサーバー名を指定します。
-o m, *machinename*	オーバライドを実行するマシン名を指定します。
-o p, *protocol*	オーバライドサーバーにアクセスします。
-o r	現在アクティブなオーバライドをリセットし、もとの設定に戻します。
-o s, *servername*	ターゲットサーバーへの接続をこのサーバーへの接続にオーバライドします。
-t	サーバーを終了します。
-v	mfclientのバージョン番号を表示します。

上記のフラグは-または/を付けて指定します。大文字小文字は区別しません。

14.10 高度なトピック

この項では、次の高度なトピックを取り扱います。

- 監査トレールの作成
- 構成ファイルの各エントリのオーバライド
- インライン構成機能の使い方
- 縮小データ転送機能
- サーバー制御のファイル管理機能

14.10.1 監査トレールの作成恒雄

クライアント/サーバー結合では、クライアントがサーバーに接続したときの日付と時刻を記録した監査トレールを

作成できます。この機能を使用するには、構成ファイルにuseraudit=yという設定を入れます。記録される情報の詳細は、この章の システムエラー/メッセージログ の項を参照して下さい。

14.10.2 構成ファイルの各エントリのオーバライド

各クライアントの起動時にクライアント/サーバーは、

1. 構成ファイルを読み込みます。
2. システムログファイルと同じ場所でmfcsovrd.cfgというファイルに関するクライアント/サーバー結合を探します。このファイルが見つかった場合、構成ファイルを使ってセットアップされたすべてのパラメータは、このファイルの内容によってオーバーラードされます。

   オーバライドファイルの形式は通常の構成ファイルとほぼ同じですが、override-cntrlというエントリが追加されています。このエントリはオーバライドの対象を示し、次のどちらかです。

   - サーバー名

     そのサーバーを使用するすべてのクライアントのパラメータがオーバライドファイルの内容に従って変更されます。

   - タグ名

     そのタグ名を使用するクライアントのパラメータだけが変更されます。

クライアントオーバライド機能は、サーバーが使用不可能になったのでアプリケーションを別のマシン上で実行しなければならない場合などに使用します。個々の構成ファイルを変更する方法もありますが、オーバライドファイルを使用すれば、1つのファイルだけですべてのアプリケーションの接続先を変更できます。

オーバライド機能はサーバーでも使用できますが、その場合はサーバーマシンが立ち上がって稼動していなければなりません。この場合もサーバー名またはタグ名でオーバライドできます。

オーバライドファイルが見つかると、そのことがログファイルに記録されます。同時にオーバライドの対象となるパラメータもすべて記録されます。

オーバライド機能を使用して、クライアントの接続先サーバーを変更する例を次に示します。

```
[OVERRIDE-CNTRL]
OVERRIDE=SERVERNAME
[OLDSERVER]
SERVERNAME=NEWSERVER
```

上の例では、[override-cntrl]セクションでオーバライドの対象とするサーバー名を指定（override=servername）し、そのサーバー名（[oldserver]）の下に新たな接続先サーバー名を指定（servername=newserver）しています。この場合、

ログファイルには次のように記録されます。

```
20/04/1997 11:01:02 Using Local File: mfcsovrd.cfg
Overriding Entries for Servername:OLDSERVER
servername=newserver
20/04/1997 11:01:02 Override Completed:
```

指定されたタグを使用する、すべてのクライアントのサーバーをオーバライドする例を次に示します。

```
[OVERRIDE-CNTRL]
OVERRIDE=TAGNAME
[MF-CLISRV]
SERVERNAME=NEWSERV
```

上の例では、[override=cntrl]セクションでオーバライドの対象とするタグ名を指定（override=tagname）し、そのタグ名（[mf-clisrv]）の下にオーバライドするパラメータ（この場合はサーバー名）を指定（servername=newserv）しています。この場合、ログファイルには次のように記録されます。

```
20/04/1997 11:04:02 Using Local File: mfcsovrd.cfg
Overriding Entries for Tagname:MF-CLISRV
servername=newserver
20/04/1997 11:04:02 Override Completed:
```

14.10.3 インライン構成機能の使い方

クライアント/サーバー結合の強力な機能の1つとして、構成ファイルで通信条件を制御できる点があります。ただし、エンドユーザーが構成ファイルのエントリを変更するだけでアプリケーションの動作が変わってしまうのは望ましくない場合があります。そこで、構成パラメータをすべてクライアントプログラム内で指定する方法があります。そうすると構成ファイルが一切使用されないため、エンドユーザーはアプリケーションの動作を変更できません。その場合もユーザーが複雑な通信コードを書く必要はなく、指定したいパラメータをクライアントプログラム内で簡単に指定できます。

パラメータの指定は、クライアントプログラムのmfclisrv.cpy COPYファイルの中で`load-inlinecfg`を設定することにより始まり、`end-inline-cfg`を設定することにより終わります。パラメータエントリは1つずつ`lnkerror-msg`にロードし、mfclientを呼び出して処理します。パラメータの設定は実際の処理ループに先だって行わなければなりません。インライン設定の例を次に示します。

```
WORKING-STORAGE SECTION.
01  ws-null           PIC X(4) COMP-X.
01  ws-scrnset-ver    PIC X(4) COMP-X.
```

```
01  ws-ret-stat       PIC X COMP-X VALUE 0.
COPY "mfclisrv.cpy".
01  dialog-system     PIC X(48).
LINKAGE SECTION.
COPY "DS-CNTRL.V1".
COPY "CUSTOMER.CPB".
PROCEDURE DIVISION.

Client-Control SECTION.
   SET load-inline-cfg TO TRUE
   MOVE "clierrprog=same" TO lnk-error-msg
   CALL lnk-client USING lnk-param-block

   MOVE "srverrprog=same" TO lnk-error-msg
   CALL lnk-client USING lnk-param-block

   MOVE "servername=mainserv" TO lnk-error-msg
   CALL lnk-client USING lnk-param-block

   SET end-inline-cfg TO TRUE
*  メインループは、サーバー端との接続が終わるまで
*  繰り返される
   PERFORM UNTIL End-Connection
*  'lnk-client'は名前'mfclient'を保持する。
*  初回はシステムを初期設定し、
*  サーバーとの接続を確立する
      CALL lnk-client USING lnk-param-block
      EVALUATE TRUE
         WHEN start-connection
   ........ The rest of the program is standard from this
   ............ point onwards  ............
```

外部の構成ファイルとインライン設定の両方を使用することもできます。そうすると、エンドユーザーも必要に応じてシステムをある程度制御でき、またクライアントプログラムも最終的な制御権を握っているという両方の利点が得られます。構成ファイルが先に処理され、次にインライン設定が処理されます。したがって、構成ファイルで不適切なパラメータ設定が行われてもインライン設定で上書きできます。この方法を使用するときは、インライン設定を`use-combined-cfg`と`end-inline-cfg`の2つの設定の間で行います。

```
WORKING-STORAGE SECTION.
01  ws-null           PIC X(4) COMP-X.
01  ws-scrnset-ver    PIC X(4) COMP-X.
01  ws-ret-stat       PIC X COMP-X VALUE 0.
COPY "mfclisrv.cpy".
01  dialog-system     PIC X(48).
LINKAGE SECTION.
COPY "DS-CNTRL.V1".
COPY "CUSTOMER.CPB".
PROCEDURE DIVISION.
Client-Control SECTION.
   SET use-combined-cfg TO TRUE
   CALL lnk-client USING lnk-param-block
   SET load-inline-cfg TO TRUE
   MOVE "servername=mainserv" TO lnk-error-msg
   CALL lnk-client USING lnk-param-block
   SET end-inline-cfg TO TRUE
*  メインループは、サーバー端との接続が終わるまで
*  繰り返される
   PERFORM UNTIL End-Connection
*  'lnk-client'は名前'mfclient'を保持する。
*  初回はシステムを初期設定し、
*  サーバーとの接続を確立する
      CALL lnk-client USING lnk-param-block
      EVALUATE TRUE
         WHEN start-connection
   ....... The rest of the program is standard from this
```

```
.......... point onwards  ............
```

14.10.4 縮小データ転送機能

縮小データ転送（RDT）は、ネットワークを通じて渡されるデータの量を制御したり、このデータを求める結果を得るのに必要最小限の大きさに制限するための手段を提供します。

クライアント/サーバー結合では、構成ファイルで指定された大きさのデータブロックを格納できるバッファが割り当てられます。クライアントプログラムとサーバープログラムの間で制御が移動するたびに、このサイズのバッファが送受信されます。そのため、ネットワークに不必要に大きな負荷がかかる場合があります。例えば、22Kのレコード領域を持つ24Kのバッファが割り当てられた場合を考えてみましょう。ファイルが10バイトのレコードキーを持つ場合、クライアントとサーバーの間でキーをやり取りするには10バイトのみ必要ですが、24Kのバッファが使用されます。実際にはデータサイズより1〜2バイト余分に必要ですが、それでもバッファ全体は大きすぎます。

RDT機能を使用するには、制御フラグ（use-rdt）と次の3つのパラメータをクライアントプログラムのmfclisrv.cpyで設定する必要があります。

lnk-usr-fcode　ユーザー機能インジケータ。ユーザーサーバープログラムに受信したデータの処理方法を示します。

Lnk-usr-retcode　バッファ先頭ポイント。4つのデータ領域のどれが転送開始ポイントか指定します。

- 1=Dialog Systemの制御ブロック
- 2=データブロック
- 3=Dialog Systemのイベントブロック
- 4=ユーザーデータブロック

11〜14も同じアドレス領域を示しますが、この場合はデータが圧縮されていることを示します。データ圧縮を使用するには構成ファイルで指定しておく必要があり、そうでないとデフォルトオプション (圧縮なし) になります。0と指定するとデータ転送が行われません。0でNULL操作を表せるので、IF文を多用してさまざまな判定をする必要がなくコードがシンプルになります。NULL操作は、ローカルに完結する処理のときにネットワークトラフィックをゼロにできるので、ネットワークの負荷軽減に大変有効です。

Lnk-data-length　転送するデータサイズ。

インデックスファイルに対してレコード (顧客詳細情報) の追加、削除、読み込みを行うアプリケーションを考えてみましょう。このインデックスファイルのレコードキーは顧客コードです。このアプリケーションには、顧客レコードの操作だけでなく顧客情報をインタフェース画面から消してしまうオプション（CLEAR）もあります。このサンプルアプリケーションのコードの一部を次に示します。`user-data-block`領域には、6バイトのレコードキーが入ります。クライアントとサーバーの間で顧客詳細情報レコードをやり取りするために、データブロック領域

（dblksize）を使用しますが、これがcustomer-data-blockです。customer-data-block内の6バイトのデータ項目customer-c-codeにレコードキーが入ります。

クライアント側のコードは次のようになります。

```
EVALUATE TRUE
    WHEN customer-load-flg-true
*   ユーザーが顧客コードを入力し、そのコードに関する
*   顧客詳細を読み取り、表示するためにインタフェースから
*   "LOAD"オプションを選択した。
    MOVE customer-c-code TO user-data-block
        SET use-rdt TO TRUE
        MOVE 1 TO lnk-usr-fcode
        MOVE 4 TO lnk-usr-retcode
        MOVE 6 TO lnk-data-length
    WHEN customer-del-flg-true
*   ユーザーが、ファイルから顧客レコードを削除するために、
*   顧客コードを入力し、DELETEオプションを選択した。
    MOVE customer-c-code TO user-data-block
        SET use-rdt TO TRUE
        MOVE 2 TO lnk-usr-fcode
        MOVE 4 TO lnk-usr-retcode
        MOVE 6 TO lnk-data-length
        INITIALIZE customer-data-block
    WHEN customer-clr-flg-true
*   ユーザーが、画面から現在の顧客詳細を消去するために
*   CLEARオプションを選択した。
    SET use-rdt TO TRUE
        MOVE 0 TO lnk-usr-retcode
        INITIALIZE customer-data-block
        PERFORM Set-Up-For-Refresh-Screen
END-EVALUATE
```

CLEARオプションが選択された場合、画面をクリアするのはローカルに完結する操作なので、サーバーに接続する

必要がなく NULL 操作を使用しています。RDT機能に対応するサーバー側のサンプルプログラムの一部を次に示します。クライアント/サーバー結合ではsend-via-rdtフラグがセットされるので、lnk-usr-fcodeをチェックしなければならないかわかります。サーバー側のコードは次のようになります。

```
WHEN send-via-rdt
    EVALUATE lnk-usr-fcode
        WHEN 1
*   LOAD機能の場合、サーバープログラムはデータファイルから
*   顧客詳細を読み取り、RDT機能を使用するのではなく、
*   データ領域customer-data-blockを使って、クライアントに
*   データを送り返す。RDTフラグがセットされていないかぎり、
*   クライアント/サーバー結合は、クライアントとサーバーの間で
*   常にデータ領域全体（構成ファイルのdblksizeによって定義される）
*   を受け渡しする。
        MOVE user-data-block TO customer-c-code
            SET customer-load-flg-true TO TRUE
            PERFORM... .rest of program... .
        WHEN 2
            MOVE user-data-block TO customer-c-code
            SET customer-del-flg-true TO TRUE
            PERFORM... .rest of program... .
    END-EVALUATE
```

14.10.5 サーバー制御のファイル管理機能

クライアント/サーバーソリューションにつきまとう問題の1つとして、クライアント数の増加に伴うクライアントプログラムなどの、更新の煩雑さがあります。

オーバライド機能（*構成ファイルの各エントリのオーバライド*の項を参照）を使用すると、各クライアントの構成ファイルを個別に書き換えなくても、メンテナンス中のサーバーから別のサーバーに変更できます。オーバライドはローカルでもリモートでも実行できますが、各クライアントのローカルなオーバライドファイルをインストールしたり削除したりする手間がかかります。

mfcsmgrプログラムを使用すると、-iまたは-dオプションを使用して、クライアントにオーバライドファイルをインストールまたは削除できます（*サーバーの管理*の項を参照）。インストール/削除はクライアントプログラムの終了時に行われます。

そこで、mfcsmgrプログラムで -iオプションを使用すればオーバライドファイルだけでなく任意のファイルをクライアントシステム（クライアントのローカルディレクトリ）にインストールできます。すなわち、この方法で新たなスクリーンセットや最新プログラムファイルをインストールすれば、管理センターから各クライアントに更新データを配布できます。セキュリティ上の理由から削除オプションはこれほど強化されておらず、削除できるのはオーバライドファイル（mfcsovrd.cfg）に限られます。

-iオプションを使用してインストールするファイルはサーバー上になければなりません。目的のファイルがmfserverを起動したディレクトリ内にある場合は、ファイル名だけを指定します。別のディレクトリにある場合は、完全パス名で指定する必要があります。例えば、/u/live/update/newprog.int、d:¥testprog.int、$LIVE/newtest.intはどれも有効ですが、$newfileという指定は無効です。

この機能を使用するのにプログラムの変更は必要ありません。クライアントにはファイルが転送されたことを示すメッセージが表示され、システムログファイルに詳しい情報が書き込まれます。

14.11 付属の顧客例の実行

Dialog Systemのクライアント/サーバー結合デモンストレーションを実行する方法については、DialogSystem¥demo¥csbindディレクトリにあるcsbind.txtを参照してください。

14.12 システムエラー/メッセージログ

結合のクライアントとサーバーの両方のモジュールで、エラーとメッセージのログが記録されます。すべてのエントリは、日付と時刻がスタンプされ、mfclisrv.logと呼ばれるファイルに記録されます。このファイルは、プログラムを起動したディレクトリか、MFLOGDIR環境変数で指定されたディレクトリに書き込まれます。このログを定期的にチェックして、システムが正しく動作しているかを確認してください。

14.13 クライアント/サーバー結合の制限事項

クライアント/サーバー結合には制限事項はほとんどありませんが、次の点に注意してください。

- Windows 95、Windows 98、Windows NT、UNIXプラットフォームでは、.intまたは.gnt形式のクライアント/サーバー結合モジュールを実行できます。また UNIXの場合は、クライアント/サーバー結合モジュールをユーザーが書いたアプリケーションプログラムとリンクして、実行可能オブジェクトを作成することもできます。

- サポートされるクライアントの数は、クライアント/サーバー結合による制限は受けませんが、サーバーの能力、ネットワークプロトコル、またはエンドユーザーの性能要求によって制限されることがあります。たとえば、UNIXの場合、mfserverが生成できるサブプロセスの数による制限があります。ユーザーがUNIXマシンにログオンするとき、限られた数のサブプロセスをサポートする一意のプロセスidが割り当てられます。クライアントがクライアント/サーバー結合を通じて接続すると、それはすべてベースサーバーのサブプロセスになります。もちろん、サブプロセスの許容数を変更したり、複数のベースサーバーを実行することに

よって、この制限を避けることができます。UNIXマシンが数百の直接ログインユーザーをサポートしている場合、1つのマシンで実行されるクライアント/サーバー結合は、サブプロセスの制限に関する問題が解決したときと同じ数のクライアントをサポートする必要があります。

- クライアント/サーバー結合には回復機能がありません。ネットワークがダウンするとデータが失われます。クライアント/サーバーのどちら側でも障害発生を検知してログに記録しますが、データは回復できません。接続の両側のユーザー プログラムを終了させるようなRTSエラーの場合も同様です。この点において、クライアント/サーバー結合は標準RTS以上のものは提供しません。

第15章 高度なトピック

この章では、Dialog System で利用できるより高度なシステム機能について説明します。

ここでは、以下の内容について取り扱います。

- 複数の解像度によるアプリケーションの実行
- Dialog System と代替エラーメッセージファイルにインタフェースをとる方法
- ファイル選択機能とのインタフェースを作成する方法.
- 実行時にメニュー項目を修正する方法
- 呼出しインタフェースの高度な使い方
- ヘルプの追加

15.1 複数の解像度で実行するアプリケーションの実装

本稼動で複数のデスクトップの解像度を使う場合、それに対応したアプリケーションを記述するために、現在の解像度にもとづいてウィンドウ、コントロール、フォントのマッピングを完全にサポートする、COBOLプログラムとスクリーンセットの変更を実装することができます。ここでは、実装に必要な3つの領域について説明します。

- スクリーンセットを複数の解像度に対応させる
- フォントのマッピングを可能にする
- COBOLを使ってDSFNTENV環境変数を正しく設定する

15.1.1 スクリーンセットを複数の解像度に対応させる

この機能は、"dsrtcfg"を呼び出す(CALLOUT)ことで可能になります。呼び出す際は、どの解像度でスクリーンセットが定義(DEFINED)されたのかをDialog Systemに伝える、9のフラグと1つの識別子を指定します。

Dialog Systemランタイムは、Panels V2の汎用座標を利用しています。これはクロス解像度のサポートおよび異なるグラフィックアダプタとの互換性の基礎となるものです。これを使わなければ、同じ解像度に設定されると思われる値をさまざまな座標値にすることができます。

定義プラットフォームに提供する識別子をチェックするため、Dialog Systemに確認プログラムが提供されています。このプログラムは、変更を実装する際に使う解像度の識別子について、詳しい説明が記載されたメッセージボックスを表示します。次のように実行します。

```
runw dsreschk
```

このプログラムは、次の内容を表示します。

- 現在の解像度のPanels V2汎用座標
- Dialog Systemランタイムの呼出しに渡される解像度の識別子

次のダイアログのコードを、SCREENSET-INITIALIZEDまたはウィンドウ作成の前に実行される他のダイアログテーブルに記述します。

```
CLEAR-CALLOUT-PARAMETERS $NULL

    CALLOUT-PARAMETER 1 CONFIG-FLAG $NULL
    CALLOUT-PARAMETER 2 CONFIG-VALUE $NULL
    MOVE 9 CONFIG-FLAG
    MOVE resolution id CONFIG-VALUE
    CALLOUT "dsrtcfg" 3 $PARMLIST
```

CONFIG-FLAGおよびCONFIG-VALUEは、C5 4バイトのデータフィールドが必要です。

一度実装すると、あらゆるウィンドウ、ダイアログボックスとその子コントロールは、現在の解像度に比例してサイズが変わります。ヘルプの「*複数の解像度とダイナミックなウィンドウのサイズ設定*」のトピックを参照してください。

15.1.2 フォントマッピングを可能にする

各スクリーンセットに複数の解像度をサポートするだけでなく、実行時のフォントが本稼動アプリケーションの解像度に調整されるように、作成するオブジェクトにフォントスタイルを割り当てる必要があります。

[編集]メニューの[フォント]を使ってフォントを指定すると、そのフォントのタイプフェースとスタイル名を指定することができます。スタイル名とは、指定されたタイプフェース、ポイントサイズ、属性をあらわすユーザー定義名です。このスタイル名によって、実行時の解像度に適したフォントサイズに、大きくまたは小さくすることができます。

フォントのタイプフェース、ポイントサイズ、属性を選択して、選択ボックスに必要なスタイル名を入力します。たとえば、My-Styleと入力します。

[適用]をクリックすると、選択したフォントスタイルが現在のオブジェクト(またはグループ)に適用されます。

[オプション]メニューの[リソースファイル]を使って、バイナリフォントサイドファイル(拡張子が.dfbのファイル)を指定すると、Dialog Systemは、実行時にバイナリフォントサイドファイルでスタイル名を調べ、そのフォントの詳細を使用中の解像度のプラットフォームに利用します。サイドファイルにスタイルがない場合は、デフォルトフォントが使われます。

1024*768解像度で作成されたスタイルを使って、それを640*480解像度に変換することを想定します。バイナリフォントサイドファイル(拡張子が.dfbのファイル)を指定して、スクリーンセットを保存します。スクリーンセットを保存すると、フォントサイドファイル(.dftファイル)のテキストバージョンが作成され(または追加され)、次の行が含まれます。

```
[NT]


FONT-RECORD
   STYLENAME My-Style
   ATTRIBUTES BITMAPPED, PROPORTIONAL
   POINTSIZE 12
   TYPEFACE "Roman"
END-RECORD
```

スクリーンセットおよびバイナリフォントサイドファイルを本稼動の環境に変換するには、.dftサイドファイルを編集し、各解像度で実行するための新しい定義を組み込む必要があります。

```
[RESOLUTION1]
FONT-RECORD
   STYLENAME My-Style
   ATTRIBUTES BITMAPPED, PROPORTIONAL
   POINTSIZE 12
   TYPEFACE "Roman"
END-RECORD

[RESOLUTION2]
FONT-RECORD
   STYLENAME My-Style
   ATTRIBUTES BITMAPPED, PROPORTIONAL
   POINTSIZE 8
   TYPEFACE "Roman"
END-RECORD
```

注:

- セクションマーカーNTは、定義の解像度に適するように変更されました。

- RESOLUTION2は、ファイルを編集して作成したものであり、使用するポイントサイズが小さくなっています。

新しいセクションマーカーと属性を追加したあとで、次のように、サイドファイルをバイナリフォーマットに変換します。

```
run dsfntgen /t appstyle.dft /b appstyle.dfb /c
```

これによってバイナリサイドファイルが作成されます。次に、環境変数を設定します。

```
set DSFNTENV=RESOLUTION2
```

Dialog Systemは、フォントスタイルの定義用に、新しいフォント属性を自動的に選択します。640*480解像度では、利用できる表示空間がより少ないために、小さいフォントが必要になり、DSFNTENVにRESOLUTION2が設定されます。

DSFNTENVが設定されないと、マッピングが実行されず、スクリーンセットに保存されたデフォルト値が使われます。スクリーンセットが提供されると、多くのスクリーンセットから構成されるアプリケーションは、1つのフォントサイドファイルを使うスタイル名の使用方法に一貫性を保つことができます。

15.1.3 COBOLを使ってDSFNTENV環境変数を設定する

COBOLプログラムの管理のもとで、必要なDSFNTENV環境変数を設定することができます。ここでは、どのように行うのかを説明しますが、まずはじめに、現在のアプリケーションが実行している解像度を決めることが必要になります。

次のコードを使って解像度を決めます。

```
* Determine resolution & set DSFNTENV accordingly
         MOVE LOW-VALUES               TO P2I-Initialization-Record
         MOVE P2I-Current-Environment  TO P2I-Environment
         MOVE 0                        TO P2I-Name-Length
         MOVE Pf-Initialize            TO P2-Function
         CALL "PANELS2" USING P2-Parameter-block
                              P2I-Initialization-Record
         END-CALL
         IF  P2-Status NOT = 0
            DISPLAY "Warning: Unable to start PANELS2. Error No = "
               P2-STATUS
```

```
            STOP RUN
        END-IF.
```

必要な変数とPanels V2インタフェースのレコードを取得するには、プログラムの作業場所節(Working-Storage section)に、pan2link.cpyおよびpan2err.cpyコピーファイルを含ませることが必要です。

次に、`P2I-Screen-Width`および`P2I-Screen-Height`に返される値をテストします。さらに、以下のコードとCOBOLの予約語を使って、環境変数を設定し、必要なフォントサイドファイルのセクションマーカーを指定します。

```
IF P2I-Screen-width = 640
          AND P2I-Screen-Height = 480
              DISPLAY "DSFNTENV" UPON ENVIRONMENT-NAME
              DISPLAY "RESOLUTION2" UPON ENVIRONMENT-VALUE
        END-IF
        IF P2I-Screen-width = 1024
         AND P2I-Screen-Height = 768
              DISPLAY "DSFNTENV" UPON ENVIRONMENT-NAME
              DISPLAY "RESOLUTION1" UPON ENVIRONMENT-VALUE
        END-IF
```

この環境変数は、Dialog Systemの最初の呼出しの前に設定する必要があり、COBOLの実行ユニットが終了するまで存在します。

Panels V2を呼び出す前に、次のコードを追加することによって、本稼動プラットフォームの解像度を改良する(たとえば、大きいフォントの800*600などに)ことができます。

```
ADD P2I-Generic-Coordinates TO P2I-Environment
```

これは、戻された座標を変更してDSRESCHKプログラムの戻り値と一致させます。それから、後続のIF文を調整して、サポートされた本稼動の解像度を反映させる必要があります。

これで、スクリーンセットに複数の解像度のサポートが実装され、すべてのスクリーンセットオブジェクトのフォントが実行時に再マップされ、現在の本稼動用解像度のプラットフォームを反映した正しい環境変数が設定されました。

よく設計されたウィンドウのレイアウトが提供されると、スクリーンセットはさまざまな解像度で完全に移植できるようになります。

15.2 Dialog System のエラーメッセージファイルの使い方

呼出しプログラムは、Dialog System のエラーメッセージファイルと表示機能を使って、エラーメッセージを表示す

ることができます。これは、Dialog System による実行には向いていないような複雑な妥当性検査が必要な場合などに便利です。

エラーメッセージは、通常はエラーメッセージファイルに格納されます。最も簡単な方法は、スクリーンセットが使用しているエラーメッセージファイルを使用することです。別の方法として、1つまたは複数のエラーメッセージファイルを使用することもできます。しかし、その場合は開く操作と閉じる操作を明示的に行う必要があります。

プログラムから Dialog System のエラーメッセージファイルを使用するには、実行時にプログラムで次の3つのことを行う必要があります。

- エラーファイルが開かれているかを確認します。
- エラーメッセージを抽出します。
- エラーメッセージとそれを表示するための指示を Dialog System に渡します。

次のコード部分は、このための手続きを示しています。

```
 1 working-storage section.
 2 ...
 3 01  error-file-linkage.
 4    03 short-file-name       pic x(8).
 5    03 file-access           pic xx.
 6    03 filler                pic x(6).
 7    03 errhan-code           pic xx.
 8
 9 01 error-file-data.
10    03 error-record-number   pic 9(4) comp.
11    03 error-record-contents pic x(76).
12 ...
13 procedure division.
14    ...
15    initialize ds-control-block
16    initialize data-block
17    move "N" to ds-control
18    move data-block-version-no to ds-data-block-version-no
19    move version-no to ds-version-no
```

```
20    move "custom" to ds-set-name
21    call "dsgrun" using ds-control-block
22                        data-block
23    ...
24    move "E" to ds-control
25    call "dsgrun" using ds-control-block
26                        data-block
27    ...
28    move "CUSTERR" to short-file-name
29    move "R" to file-access
30    move 101 to error-record-number
31    call "dserrhan&uot; using error-file-linkage
32                           error-file-data
33    ...
```

行3**〜**11:

Working-Storage Section でこれらのレコードを定義します。

```
 01  error-file-linkage.
     03 short-file-name       pic x(8).
     03 file-access           pic xx.
     03 filler                pic x(6).
     03 errhan-code           pic xx.


 01 error-file-data.
     03 error-record-number   pic 9(4) comp.
     03 error-record-contents pic x(76).
```

行15**〜**22:

Dialog System に対する最初の呼出しです。

```
    initialize ds-control-block
    initialize data-block
    move "N" to ds-control
```

```
    move data-block-version-no to ds-data-block-version-no
    move version-no to ds-version-no
    move "custom" to ds-set-name
    call "dsgrun" using ds-control-block
                        data-block
```

行24**〜**26:

スクリーンセットが使用しているエラーメッセージファイルを使う場合、Dialog System を通常の方法で、`ds-control` にパラメータ "E" をいれて呼出し、ファイルが開かれているかを確認しなければなりません。Dialog System は、ファイルを開いたプログラムにただちに制御を返し、`ds-control` をパラメータ "C"（`ds-continue`）に置き換えます。

```
    move "E" to ds-control
    call "dsgrun" using ds-control-block
                        data-block
```

`ds-err-file-open` は、`ds-control` に関してあらかじめ定義された値です。これは、制御ブロックで次のように定義されます。

```
       05 ds-err-file-open          pic x value "E".
```

したがって、行 24 は次のように変えることもできます。

```
    move ds-err-file-open to ds-control
```

行28:

エラーファイルの名前（拡張子 `.err` を含まない）をデータ項目 `short-file-name` に移動します。

```
    move "custerr" to short-file-name
```

行29:

"R" は読み取り（Read）を表します。

```
    move "R" to file-access
```

行30:

表示したいエラーメッセージの番号を移動します。

```
    move 101 to error-record-number
```

行31**〜**32:

この呼出しは、エラーメッセージファイルを読み取ります。呼出しが成功した場合、errhan-code を "OK" にし、error-record-contents にエラーメッセージを入れて戻ります。ファイルが見つからない場合、値 "NF"（Not Found）が errhan-code に書き込まれます。ファイルが使用中だったり、16 個を超えるファイルを開こうとした場合は、"FU"（File in Use）が errhan-code に書き込まれます。エラーファイルは、Dialog System によって自動的に閉じられるので、明示的に閉じる必要はありません。

```
call"dserrhan" using error-file-linkage
                     error-file-data
```

つぎにエラーメッセージをデータブロックのデータ項目に入れ、Dialog System が表示できるようにする必要があります。このためには、データ項目を引数として使い、メッセージボックスを表示する手続きを設定し、Dialog System を呼出してその手続きを要求するという方法があります。

たとえば、ERR-DISPLAY-MB というメッセージボックスがある場合、次のダイアログによって手続きを定義します。

```
DISPLAY-ERR-MSG
  INVOKE-MESSAGE-BOX ERR-DISPLAY-MB ERR-MSG-EF $REGISTER
```

そして、プログラムの中で Dialog System に戻るときに、次のような文を使って手続きを要求します。

```
move "display-err-msg" to ds-procedure
```

display-err-msg は手続きの名前、ds-procedure は制御ブロック内のデータ項目です。この手続きは、Dialog System に入るときに実行されます。

15.2.1 代替エラーメッセージファイルの使い方

スクリーンセットによって指定されたものと別のエラーメッセージファイルを使用するには、実行時にプログラムから次の4つのことを行う必要があります。

- エラーファイルを開きます。
- エラーメッセージを抽出します。
- エラーメッセージとそれを表示するための指示を Dialog System ランタイムシステムに渡します。
- スクリーンセットが使用していないエラーファイルが閉じていることを確認します。

代替メッセージファイルと Dialog System のエラーメッセージファイルを使用する場合の唯一の違いは、代替ファイルの場合、開く操作と閉じる操作を明示的に行わなければならないことです。

次のコード部分は、このための手続きを示しています。

```
 1 working-storage section.
 2 ...
 3 01  error-file-linkage.
 4    03 short-file-name        pic x(8).
 5    03 file-access            pic xx.
 6    03 filler                 pic x(6).
 7    03 errhan-code            pic xx.
 8
 9 01 error-file-data.
10    03 error-record-number    pic 9(4) comp.
11    03 error-record-contents  pic x(76).
12 ...
13 procedure division.
14    ...
15    move "ALTERR" to short-file-name
16    move "NI" to file-access
17    move "C:¥CUST¥ALTERR.ERR" to error-file-data
18    call "dserrhan" using error-file-linkage
19                          error-file-data
20    ...
21    move "ALTERR" to short-file-name
22    move "R" to file-access
23    move 101 to error-record-number
24    call "dserrhan" using error-file-linkage
25                          error-file-data
26    ...
27    move "ALTERR" to short-file-name
28    move "NC to file-access
29    call "dserrhan" using error-file-linkage
30                          error-file-data
31    ...
```

行3**〜**11:

ここでも、Working-Storage Section でレコードを定義する必要があります。

```
01  error-file-linkage.
    03 short-file-name       pic x(8).
    03 file-access           pic xx.
    03 filler                pic x(6).
    03 errhan-code           pic xx.


 01 error-file-data.
    03 error-record-number   pic 9(4) comp.
    03 error-record-contents pic x(76).
```

行15:

代替ファイルを識別します。

```
    move "ALTERR" to short-file-name
```

行16:

"NI" は、新しいファイルを開くためのファイルアクセスコードです。

```
    move "NI" to file-access
```

行17:

開くエラーファイルのパスと名前を指定します。

```
    move "C:¥CUST¥ALTERR.ERR" to error-file-data
```

行18**〜**19:

呼出し文によって、エラーファイルを読み取ります。呼出しが成功した場合、errhan-code を "OK" にし、error-record-contents にエラーメッセージを入れて戻ります。呼出しが失敗した場合、errhan-code は "NF"（Not Found）になります。ファイルがすでに開かれていたり、16 個を超えるファイルを開こうとした場合、errhan-code は "FU"（File in Use）になります。

```
    call "dserrhan" using error-file-linkage
                          error-file-data
```

行27:

閉じるファイルを指定します。

```
    move "ALTERR" to short-file-name
```

行28:

"NC" は、ファイルを閉じるためのコードです。

```
    move "NC to file-access
```

行29〜30:

この呼出しはファイルを閉じます。errhan は、ファイルを閉じ、プログラムに戻ります。

```
    call "dserrhan" using error-file-linkage
                          error-file-data
```

ここでエラーメッセージを表示する必要があります。具体的な手順については、*「Dialog System のエラーメッセージファイルの使い方」*の節を参照してください。

15.3 ファイル選択機能

ユーザーがファイルの名前を入力する必要がある場合（例えば、アプリケーションがレポートを生成し、ユーザーがレポートのファイル名を指定する場合）、現在システムにあるファイルを示すプロンプトを表示すると便利です。ユーザーは、システムの各ドライブやディレクトリを見て、既存のファイル名を選択して、新しい名前を入力することができます。

Dialog System では、スクリーンセット用の COPY ファイルを生成し、使用するファイル名を指定するときなどに、このファイル選択機能を使用します。

この節では、Dsdir DSX を使って、Dialog System アプリケーションからこの機能を提供する方法について説明します。DSX は、CALLOUT ダイアログファンクションを使って実行されるダイアログファンクションのことです。DSX は、オンラインヘルプを呼び出したり、ファイル選択機能を提供するなど、通常のさまざまなプログラミングタスクを実行するために用意されています。DSX の詳細については、オンラインヘルプの*「Dialog System 拡張機能」*の章を参照してください。

15.3.1 Dirdemo サンプルスクリーンセット

Dirdemo サンプルアプリケーションは、Dsdir DSX をアプリケーションの中で使用する方法を示しています。このアプリケーションを実行すると、Dsdir DSX によって利用できる機能を見ることができます。

Dirdemo は、Dialog System 定義ソフトウェアから直接実行します。このサンプルには、COBOL プログラムは付いていません。スクリーンセット dirdemo.gs をロードし、スクリーンセット Animator によって実行します。スクリー

ンセットの実行については、第 15 章「*すべてをまとめる*」の「*スクリーンセット Animator を使ったスクリーンセットのテスト*」の項を参照してください。

このサンプルスクリーンセットは以下の２つのプロンプトを表示します:

- ファイル名

  ファイル名フィールドにテキストを入力した場合、最初はそのファイル名に一致するファイルだけが表示されます。ファイル名の中で "*" や "?" のワイルドカード文字を使用することもできます。例えば、Dialog System のスクリーンセットの一覧を見るには、"*.gs" と入力します。このフィールドを空白にした場合、デフォルトの "*.*" が使用されます。

- ウィンドウタイトル

  ウィンドウタイトルフィールドの内容は、ファイル選択機能のタイトルとして表示されます。詳細は、オンラインヘルプの「*Dialog System 拡張機能*」の章にある Dsdir に関する説明を参照してください。

ファイル選択機能ウィンドウを表示するには、プルダウンメニューから次のいずれかのオプションを選択します。

開く	開くファイルを選択します。ユーザーは、既存のファイルリストの中からのみ選択できます。
保存	保存するファイルを選択します。ユーザーは、既存の任意のファイルを選択できます（また、新しいファイル名を指定することもできます）。
確認	ファイル名フィールドに入力されたファイルが実際に存在するかどうかを確認します。この機能を選択した場合、ファイル名フィールドでワイルドカード文字を使用することはできません。
終了	終了します。

このメニューには、ファイルを実際に開くオプションはありません。ユーザーがファイルを選択したあと、どの機能を使用するかを選ぶことができます。

15.3.2 Dirdemo のデータブロック

Dirdemo DSX を呼出すには、スクリーンセットのデータブロックの中で、スクリーンセットが使用する他のデータに加え、次の項目を定義しなければなりません。

```
DSDIR-PARAMS                 1
    DSDIR-FUNCTION           X    4.0
    DSDIR-RETURN-CODE        C    2.0
    DSDIR-FILENAME           X  256.0
```

ファイル選択ウィンドウに独自のタイトルを付けるには、次のスクリーンセットのデータブロックを追加する必要があります。

```
DSDIR-PARAMS2
   DSDIR-TITLE                X  256.0
```

これらのフィールドについては、オンラインヘルプの「*Dialog System 拡張機能*」の章を参照してください。

15.3.3 Dirdemo **のダイアログ**

Dirdemo サンプルは、ファイルを選択する必要がある場合、Dsdir DSX を呼出します。

このためのダイアログ（メインウィンドウに付属する）を次に示します。

```
1    DO-DSDIR-CALL
2      CLEAR-CALLOUT-PARAMETERS $NULL
3      CALLOUT-PARAMETER 1 DSDIR-PARAMS $NULL
4      CALLOUT-PARAMETER 2 DSDIR-PARAMS2 $NULL
5      CALLOUT "Dsdir" 0 $PARMLIST
6 ...
17   @OPEN-PD
18     MOVE "open" DSDIR-FUNCTION(1)
19     EXECUTE-PROCEDURE DO-DSDIR-CALL
20   @SAVE-PD
21     MOVE "save" DSDIR-FUNCTION(1)
22     EXECUTE-PROCEDURE DO-DSDIR-CALL
23   @CHECK-PD
24     MOVE "chek" DSDIR-FUNCTION(1)
25     EXECUTE-PROCEDURE DO-DSDIR-CALL
```

行1:

この手続きは、Dsdir DSX を呼出します。

```
   DO-DSDIR-CALL
```

行2**〜**4:

```
     CLEAR-CALLOUT-PARAMETERS $NULL
     CALLOUT-PARAMETER 1 DSDIR-PARAMS $NULL
```

```
CALLOUT-PARAMETER 2 DSDIR-PARAMS2 $NULL
```

CALLOUT に必要なパラメータを定義します。ファイル選択ウィンドウのタイトルを指定するには複数のパラメータが必要なので、CALLOUT を作成するまえにすべてのパラメータを定義しなければなりません。CLEAR-CALLOUT-PARAMETERS ファンクションは、新しいパラメータを定義するまえに、以前定義されたパラメータがあれば取り除くのに使用します。

行5:

この行は、前に定義されたパラメータを使って DSX を呼出します。ファイル選択ウィンドウが表示され、ユーザーはファイルを選択することができます。ユーザーがファイルを選択したか、ファイル選択をキャンセルした場合、スクリーンセットに制御を返します。

```
CALLOUT "Dsdir" 0 $PARMLIST
```

行17:

ユーザーが、プルダウンメニューから「開く」オプションを選択しました。

```
@OPEN-PD
```

行18:

この行は、ユーザーが開くファイルを選択できるように DSX に指示します。このモードでは、ユーザーは既存のファイルだけを選択することができます。

```
MOVE "open" DSDIR-FUNCTION(1)
```

行19:

DSX を呼出す手続きを実行します。

```
EXECUTE-PROCEDURE DO-DSDIR-CALL
```

行20〜22:

ユーザーは、プルダウンメニューから[保存]オプションを選択しました。このモードでは、ユーザーは任意のファイルを選択するか、新しいファイルを指定することができます。

```
@SAVE-PD
  MOVE "save" DSDIR-FUNCTION(1)
  EXECUTE-PROCEDURE DO-DSDIR-CALL
```

行23〜25:

ユーザーは、プルダウンメニューから[確認]オプションを選択しました。chek ファンクションは、与えられたファイル名の存在をチェックするように DSX に要求します。ファイル選択ウィンドウは表示されません。

```
@CHECK-PD
  MOVE "chek" DSDIR-FUNCTION(1)
  EXECUTE-PROCEDURE DO-DSDIR-CALL
```

15.4 実行時にメニュー項目を修正する

定義時にメニューを定義するだけでなく、実行時にメニュー項目を追加、削除または変更するダイアログを定義することができます。たとえば、最近開いたファイルのリストをメニューに追加したり、開いたウィンドウのリストをMDIアプリケーションに追加することができます。

既存のメニューバーは新しい項目を追加することしかできませんが、それ以外は、メニューバーを定義する際のあらゆるオプションを実行時に使うことができます。また、メニュー選択をコンテキストメニューに追加することはできません。

ADD-MENU-CHOICEダイアログファンクションを使って、メニューバーにメニュー選択を追加します。このファンクションのパラメータの1つに、メニュー項目のすべてのオプションを指定するテキスト文字があります。次のサンプルは、すべてのオプションが正しい場所で指定されていますが、実際には、このとおりにメニュー項目を定義しないでください。

```
GET-MENU-CHOICE-REFERENCE MAINWIN EDITMENU PARENT-REF
ADD-MENU-CHOICE PARENT-REF "~item>" $EVENT-DATA
GET-MENU-CHOICE-REFERENCE MAINWIN $EVENT-DATA PARENT-REF
ADD-MENU-CHOICE PARENT-REF "*~choice@item&ctrl+c" $EVENT-DATA
```

各パラメータを次に説明します。

PARENT-REF	このパラメータは、新しい項目が追加されるメニュー(またはサブメニュー)を指定します。このパラメータの値は、GET-MENU-CHOICE-REFERENCEファンクションを使って取得します。値が0になることはありません。
*	アスタリスクは、オフに設定(デフォルトは設定なし)されたチェックマークタイプにで作成されたメニュー項目を指定します。アスタリスクはテキストパラメータの最初の文字にします。最初の文字にしないと、メニューテキストの一部とみなされます。アスタリスクをメニューテキストの一部にするには、前にスペースを1つ追加します。
~	チルダは、メニュー選択のニーモニック文字として定義される次の文字を指定します。重複したニーモニック文字でメニュー項目を定義しても、警告されません(定義時にメニュー項目を定義する場合とは違います)。チルダ文字は、選択テキストのどこに指定してもかまいません。
>	より大きいは、テキストパラメータの最後の文字として指定され、メニュー項目が選択さ

	れると、選択項目のサブメニューが表示されます。メニュー選択がサブメニューを表示する場合は、チェックマークの状態およびショートカットキーの設定は無視されます。
choice	@(アット)までのテキストは、メニューバー(選択テキスト)に表示されるテキストを定義します。
@item	'@'は、次のテキスト(次の@、スペース、&または>まで)がメニューの名前であることを意味します。Dialog Systemの一般的なオブジェクト命名規則は、メニュー項目名にも適用されます。メニュー項目として同じ名前でスクリーンセットに定義された手続きが存在する場合、メニュー項目を選択すると、??で手続きが実行されます。メニュー項目名はオプションです。
&ctrl+c	アンパーサンドは、次のテキスト(次の@、スペース、&または>まで)がメニュー項目のショートカットキーであることを意味します。ショートカットキーは、メニューバー定義でメニュー選択を定義するときに使用する形式(CTCなど)、または、画面に表示されるショートカットキーの形式(たとえば、Ctrl+Cなど)の両方で指定することができます。

この機能を使ってメニューバー項目を作成すると、作成したメニューバー項目のIDが返されます。このIDをDISABLE-MENU-CHOICEなどのファンクションに使って、メニュー項目をさらに操作することができます。

DELETE-MENU-CHOICEファンクションを使って、メニュー項目を削除することができます。削除する項目は、メニューバー定義で定義したメニュー項目、またはADD-MENU-CHOICEファンクションを使って追加したメニュー項目です。

また、UPDATE-MENU-TEXTファンクションを使って、メニュー項目のテキストを更新することができます。

詳細については、関連するファンクションのヘルプの使用例が記載されているものを参照してください。

15.5 呼出しインタフェースの使い方

Dialog Systemの呼出しインタフェースは、呼出しを行うプログラムで使用され、Dialog Systemを使用したより高度な方法を提供します。たとえば、次のようなことができます。

- 複数のスクリーンセットを使う
- 同じスクリーンセットの複数インスタンスを使う

これらの機能を使うと、必要に応じて使用する論理的なコンポーネントにユーザーインタフェースを分割したり、同じスクリーンセットの複数のコピーを使用したり、あらゆるエラーメッセージを1つのファイルにまとめることができます。詳細については、「*スクリーンセットの使い方*」を参照してください。

15.6 ヘルプの追加

ここでは、Helpdemoスクリーンセットの詳細について検討します。Helpdemoスクリーンセットは、次のようなものです。

- DialogSystem¥demo¥helpdemoにあります。
- Dsonline Dialog System拡張機能を使います - これは、Windowsのヘルプを表示するDialog Systemの拡張機能です。
- Windowsのヘルプを利用して状況依存のヘルプを表示します。
- ヘルプシステムの全機能へアクセスすることができます。
- サンプルのスクリーンセットとして提供されています。

この節を読み進んでいくと、スクリーンセットを実行したくなります。

15.6.1 Helpdemoサンプルの実行

Dialog Systemの定義ソフトウェアから、直接Heopdemoサンプルプログラムを実行することができます。このサンプルにはCOBOLプログラムはありません。

スクリーンセットのhelpdemo.gsをロードし、スクリーンセットAnimatorから実行します。

スクリーンセットの実行に関する情報は、「*スクリーンセットの使い方*」の章の「*スクリーンセットのテスト*」の項を参照してください。

もう1つの方法として、Dsrunnerを利用してHelpdemoサンプルを実行することもできます。Dsrunnerに関する詳しい情報は、「*複数のスクリーンセット*」の章を参照してください。

15.6.2 Helpdemoデータブロック

Dsonline Dialog System拡張機能を呼び出すには、スクリーンセットが使用する他のデータのほかに、次の行をスクリーンセットのデータブロックに定義する必要があります。

```
DSONLINE-PARAMETER-BLOCK          1
    DSONLINE-FUNCTION           X   18.0
    DSONLINE-RETURN             C   2.0
    DSONLINE-HELP-FLAGS         C   2.0
    DSONLINE-HELP-CONTEXT       9   18.0
    DSONLINE-HELP-TOPIC         X  32.0
```

```
   DSONLINE-HELP-FILE          X  256.0
```

これらの各フィールドの意味は、ヘルプの「*Dialog Systemの拡張機能*」トピックのDsonline Dialog System拡張機能の説明に記載されています。

15.6.3 Helpdemoダイアログ

このHelpdemoサンプルは、いつでもヘルプ情報が必要なときに、Dsonline Dialog System拡張機能を呼び出します。ヘルプは特定のフィールドに対して要求されたり、ユーザーが[ヘルプ]メニューのオプションを選択したときに要求されます。

それを実行する(メインウィンドウに設定された)ダイアログは、次のようなものです。

```
1  F1
2     MOVE 1 DSONLINE-HELP-CONTEXT(1)
3     BRANCH-TO-PROCEDURE CONTEXT-HELP
4   @HELP-CONTENTS
5     MOVE "cont" DSONLINE-FUNCTION(1)
6     BRANCH-TO-PROCEDURE CALL-ON-LINE
7   @HELP-INDEX
8     MOVE "indx" DSONLINE-FUNCTION(1)
9     BRANCH-TO-PROCEDURE CALL-ON-LINE
10   @EXIT-F3
11     SET-EXIT-FLAG
12     RETC
13   CONTEXT-HELP
14     MOVE "ctxt" DSONLINE-FUNCTION(1)
15     BRANCH-TO-PROCEDURE CALL-ON-LINE
16   CALL-ON-LINE
17     MOVE "helpdemo.hlp" DSONLINE-HELP-FILE(1)
18     MOVE 0 DSONLINE-HELP-FLAGS(1)
19     CALLOUT "dsonline" 0 DSONLINE-PARAMETER-BLOCK
```

1行目:

```
F1
```

ユーザーがF1キーを押しました。このサンプルの各入力フィールドは、F1キーによって引き起こされるイベントに対応したコントロールダイアログをそれぞれ持っています。そのため、このダイアログがイベントを処理した場合は、

フォーカスがどの入力フィールドにも存在しないことを意味します。このイベントが処理されると、一般的なヘルプが提供されます。

2行目

```
MOVE 1 DSONLINE-HELP-CONTEXT(1)
```

表示されるコンテキストの番号を設定します。コンテキスト番号はオンラインヘルプファイルに定義されます。コンテキスト番号を指定しない場合は、オンラインヘルプファイルのバッファ(ohbld)がその番号を提供します。このサンプルでは、1は一般的なヘルプのコンテキスト番号です。

3行目

```
BRANCH-TO-PROCEDURE CONTEXT-HELP
```

ヘルプを表示する手続きへ分岐します。すべてのコンテキストヘルプが同じ方法で表示されるので、ダイアログの繰り返しを回避するために、手続きが使われます。

4行目:

```
@HELP-CONTENTS
```

ユーザーが[ヘルプ]メニューの[内容]を選択しました。

5行目

```
MOVE "cont" DSONLINE-FUNCTION(1)
```

Dsonline Dialog System拡張機能は、いくつかの動作を行います。何をするのかを伝えるには、ファンクション名を提供する必要があります(4文字)。`cont`ファンクションは、オンラインヘルプファイルの内容を表示するように要求します。

6行目

```
BRANCH-TO-PROCEDURE CALL-ON-LINE
```

Dialog System拡張機能を呼び出す手続きへ分岐します。

7-9行目:

```
@HELP-INDEX
MOVE "indx" DSONLINE-FUNCTION(1)
BRANCH-TO-PROCEDURE CALL-ON-LINE
```

ユーザーは[ヘルプ]メニューの[インデックス]を選択しました。**インデックス**ファンクションは、オンラインヘルプファイルを表示するインデックスを要求します。

10行目

```
  @EXIT-F3
```

ユーザーは[終了]メニューの[ファイル]を選択するか、F3キーを押しました。

11行目

```
    SET-EXIT-FLAG
```

終了フラグは、RETCが実行されたときに、Dsgrunが呼出しプログラムに返すものです。終了フラグは、スクリーンセットが処理を終えたことをDsrunnerに伝えます。

12行目

```
    RETC
```

呼出しプログラムへ戻ります(このサンプルでは、Dialog SystemまたはDsrunnerのいずれかです)。

13-15行目:

```
  CONTEXT-HELP
    MOVE "ctxt" DSONLINE-FUNCTION(1)
    BRANCH-TO-PROCEDURE CALL-ON-LINE
```

この手続きは、コンテキストヘルプを表示するために使われます。`ctxt`ファンクションはコンテキストヘルプを表示するよう要求します。`CALL-ON-LINE`手続きは、Dialog System拡張機能を呼び出すために使います。

16-19行目

```
  CALL-ON-LINE
    MOVE "helpdemo.hlp" DSONLINE-HELP-FILE(1)
    MOVE 0 DSONLINE-HELP-FLAGS(1)
    CALLOUT "dsonline" 0 DSONLINE-PARAMETER-BLOCK
```

この手続きはDsonline Dialog System拡張機能を呼び出します。これは、パラメータブロックにヘルプファイル名を設定し、オンラインヘルプシステムのフラグ設定をすべてオフにします(フラグに関する情報については、ヘルプの「*Dialog System拡張機能*」トピックに記載されているDsonline Dialog Systemを参照してください)。

15.6.4 入力フィールドのダイアログ

メインウィンドウのダイアログと同様に、ウィンドウの入力フィールドもそれぞれダイアログを持ちます。各入力フィールドには異なるコンテキスト番号を設定しますが、設定しない場合は、同じダイアログになります。

製品コードの入力フィールドの場合、ダイアログは次のようになります。

```
1  F1
2      MOVE 2 DSONLINE-HELP-CONTEXT(1)
```

```
3       BRANCH-TO-PROCEDURE CONTEXT-HELP
```

このダイアログは、メインウィンドウのF1キーのダイアログと似ていますが、別のコンテキスト番号を使っています。 2 は、この入力フィールドのヘルプに関するコンテキスト番号です。

15.7 詳細情報

スクリーンセットの使用をコントロールする方法、複数のスクリーンセットとスクリーンセットの複数インスタンスを利用する方法について詳しくは、「*スクリーンセットの使い方*」および「*複数のスクリーンセット*」の各章を参照してください。

「*複数のスクリーンセット*」にも呼出しインタフェースの詳細情報が記載されています。ヘルプの「*呼出しインタフェース*」のトピックでも、コントロールブロック、イベントブロック、データブロック、スクリーンセットAnimator、バージョンチェック、呼出しプログラムのDialog Systemへの戻り値に関する情報を提供しています。

第16章 Q & A

この章では、Micro Focus にお寄せいただいた中で、比較的多い質問とそれに対する回答を示します。

注：質問には、すべての環境に関するものと、特定の環境だけに当てはまるものがあります。特定の環境についての質問では、下にその環境を表示しています。

スクリーンセットから直接生成したデータブロックをチェッカーが拒否する。どこが間違っているのか?

データブロックにあるすべてのデータ項目に接頭辞としてスクリーンセット識別子が付くようにプログラムが設計されている場合、それに合わせてスクリーンセットを生成しなければなりません。

たとえば、Customer プログラムでは、データブロックのすべてのデータ項目に CUSTOMER という接頭辞が付きます。データブロックを作成し、データブロックにスクリーンセット識別子を付けるように指定しない場合、データ項目には CUSTOMER という接頭辞は付けられず、項目に対する参照はチェッカーによって拒否されます。

プログラムを再コンパイルせずにスクリーンセット Animator 機能を使用するには、どうしたらよいか?

2つの方法があります。

- Dialog System ランタイムシステムの名前を可変にします。たとえば、Customer デモプログラムは、次のレベル 01 データ項目を使用するように変更できます。

  ```
  01 dialog-system   pic x(8) value "dsgrun".
  ```

  これによって、Animator は `dialog-system` の値を "ds" に変更することができます。ds は、スクリーンセット Animator 機能が使用できる Dialog System ランタイムシステムです。

  一度このように行うと、以降のDialog Systemランタイムの呼出しはすべて、スクリーンセットAnimatorを利用することになります(`dialog-system`の値をdsgrunに戻した場合も)。

- `ds-control` の "T" と "O" のオプションを使用します。`ds-control` を "T" に設定するとスクリーンセット Animator がオンになります。`ds-control` を "O" に設定するとスクリーンセット Animator がオフになります。

  スクリーンセット Animator をオンにするには、

  1. Dialog System の CALL 文に区切り点を設定します。

2. Animator を使って、データ項目 `ds-control` の値を "T" に変更します。

3. Dsgrun に対する呼出しを Step 実行します。

   スクリーンセットがときどき点滅するのがわかります。"T" オプションによって、Dialog System はスクリーンセット Animator をオンにする内部呼出しを実行し、終了します。

4. カーソルを Dialog System の CALL 文に再設定します。

5. プログラムをもう一度 Zoom します。スクリーンセットのダイアログが見えるはずです。

スクリーンセット Animator をオフにするには、

1. データ項目 `ds-control` の値を "O" にします。

2. Dsgrun に対する呼出しを Step 実行します。

   スクリーンセット Animator が消滅します。"T" オプションのときと同様に Dialog System はスクリーンセット Animator をオフにする内部呼出しを実行し、終了します。

3. カーソルを Dialog System の CALL 文に再設定します。

4. プログラムを Zoom します。スクリーンセット Animator がオフになっています。

SHOW-WINDOW コマンドを使用するとダイアログボックスが表示されない。なぜか?

これは、ダイアログボックスがモーダルダイアログボックスの場合に起こります。SHOW-WINDOW は、画面上にこのタイプのダイアログボックスを描きません。モーダルダイアログボックスは、SET-FOCUS ファンクションによってフォーカスを設定した場合にのみ表示されます。SHOW-WINDOW が画面上に描くのは、ウィンドウやモードレスのダイアログボックスです。

フィールドがフォーカスを失うとすぐに妥当性検査(妥当性のチェック)を行うには、どうしたらよいか?

フィールドごとの妥当性検査を実装する際にもっとも一般的な問題は、アプリケーションが妥当性検査エラーによって無限ループになってしまうことです。たとえば、ある入力フィールドがフォーカスを失う(ルーズフォーカス)と妥当性検査が実行されますが、妥当性検査が不成功になると、フォーカスは移動元のオブジェクトに戻ります。しかし、フォーカスが戻ると、移動先のコントロールにルーズフォーカスが起こり、それによって妥当性検査が実行され無限ループになる可能性があります。

他の問題としては、フィールドの妥当性検査を必要としない場合が挙げられます。たとえば、ユーザーがダイアログボックスの[キャンセル]または[ヘルプ]ボタンをクリックする場合や、他のアプリケーションに切り替える場合などです。

フィールドごとの妥当性検査を実装する方法として、無限ループの問題を回避し、[キャンセル]および[ヘルプ]ボタ

ンを実装可能にするには、次のように行います。

1. データブロックにフラグを定義する

```
VALIDATION-ERROR-FOUND     C5    1.0
VALIDATION-ERROR-FIELD     C5    2.0
VALIDATION-ERROR-MESSAGE   C5   80.0
```

スクリーンセットにエラーメッセージフィールドがある場合は、VALIDATION-ERROR-MESSAGEを定義する代わりに、そのフィールドを使うことができます。そのようなフィールドがない場合は、エラーフィールドとしてVALIDATION-ERROR-MESSAGEフィールドを定義します。

2. 次のグローバルダイアログを追加する

```
REPORT-VALIDATION-ERROR
* Report a validation error
    INVOKE-MESSAGE-BOX VALIDATION-ERROR-MBOX VALIDATION-ERROR-MESSAGE
    $REGISTER
    SET-FOCUS VALIDATION-ERROR-FIELD
    TIMEOUT 1 CLEAR-VALIDATION-ERROR-TIMEOUT
  CLEAR-VALIDATION-ERROR-TIMEOUT
* Reset the validation error flag and TIMEOUT
    MOVE 0 VALIDATION-ERROR-FOUND
    TIMEOUT 0 $NULL
  DO-NOTHING
* Do-nothing procedure. Contains no functions.
```

REPORT-VALIDATION-ERROR手続きは、短いタイムアウトの後で実行されます。検出された妥当性検査エラーをレポートし、エラーが発生したフィールドにフォーカスを設定します。フォーカスイベントの処理がすべて終わるとすぐに、別のタイムアウトが実行され、CLEAR-VALIDATION-ERROR-TIMEOUTが実行されます。このイベントは、妥当性検査の不成功に関するフラグをクリアします。

3. SCREENSET-INITIALIZEDダイアログに次の行を追加する

```
MOVE 0 VALIDATION-ERROR-FOUND
```

これは、最初の妥当性検査エラーに用意されたVALIDATION-ERROR-FOUNDフラグをクリアします。

4. LOST-FOCUSダイアログを持ち、フォーカスの変更を引き起こすあらゆるオブジェクトに対して、LOST-FOCUSダイアログの先頭行に次の行を追加する

```
IF= VALIDATION-ERROR-FOUND 1 DO-NOTHING
```

5. 妥当性検査エラーによってフォーカスの変更を引き起こすダイアログで、フォーカスの変更を引き起こすダイアログの直前に、次のダイアログを追加する

```
MOVE $EVENT-DATA VALIDATION-ERROR-FIELD
    MOVE 1 VALIDATION-ERROR-FOUND
    TIMEOUT 1 REPORT-VALIDATION-ERROR
```

このダイアログのサンプルでは、妥当性検査が不成功に終わったオブジェクトのオブジェクトIDが、$EVENT-DATAに設定されると想定します。$EVENT-DATAには、VAL-ERRORが発生したときのオブジェクトIDが設定されます。

6. 妥当性検査エラーがレポートされる前にキャンセルする場合、エラーをキャンセルする場所に、次のダイアログを追加する

```
EXECUTE-PROCEDURE CLEAR-VALIDATION-ERROR-TIMEOUT
```

プッシュボタンで妥当性検査エラーをキャンセルする場合、ボタンのBUTTON-SELECTEDイベントではなく、GAINED-FOCUSイベントでエラーをキャンセルする必要があります。タイムアウトは、GAINED-FOCUSイベントとBUTTON-SELECTEDイベントの間で起りますが、これが妥当性検査エラーメッセージをトリガーするためです。

アプリケーションがフォーカスを失った場合も、妥当性検査エラーをキャンセルすることが重要になります。キャンセルしないと、ユーザーは他のアプリケーションに切り替えることができなくなるだけでなく、スクリーンセットAnimatorやNetExpress IDEの操作を妨害することにもなります。アプリケーションがフォーカスを失ったときに、妥当性検査エラーをキャンセルするには、ウィンドウにLOST-FOCUSダイアログを実装します。アプリケーションで他のウィンドウにフォーカスを切り替える場合も、この方法が適用されることに注目しましょう。

妥当性検査エラーがフォーカスの変更を引き起こすと、VALIDATION-ERROR-FOUNDフラグと非常に短いTIMEOUTが設定され、フラグを再度クリアします。TIMEOUTイベントは、処理されるイベントが存在するかぎり発生することはないので、必ずLOST-FOCUSイベントの後にTIMEOUTイベントが発生します。LOST-FOCUSイベントは、VALIDATION-ERROR-FOUNDフラグが設定されていることを認めると、フォーカスを失ったフィールドの妥当性検査を行いません(フォーカスの移動も起こりません)。

Dialog Systemランタイムエラーに関する詳しい情報を知りたい。どうしたらよいか？

アプリケーションでエラーが発生すると、Dialog Systemランタイムシステムはエラーコード(DS-ERROR-CODE)および2つのエラーに関する詳細(DS-ERROR-DETAILS-1 and DS-ERROR-DETAILS-2)を返します。これによって、アプリケーションの何が間違っているのかを判断することができます。

特に、エラーコード17に関しては、エラーに有効なあらゆる情報を得るために、いくつかのテーブルを調べることが必要になります。また、エラーのより詳しい情報を返すために、エラーの詳細を使用するエラーコードもありますが、それらを解読するにはマニュアルを参照しなければなりません。

Dialog Systemにはエラーをレポートするモジュール、dsexcept.dllがあります。このモジュールは、エラーコードを解読し、エラーコードから入手できない追加情報も提供してくれます。

dsexcept.dllをアクティブにするには、$COBDIR環境変数で指定されたディレクトリにこのファイルを置く必要があります。dsexcept.dllファイルは、Dialog Systemの¥binディレクトリで見つけることができます。

注:

- dsexcept.dllは、ランタイム形式のスクリーンセットの場合は、それほどたくさんの情報を提供しません。.

  これは、本稼動アプリケーションのユーザーが、スクリーンセットをデバッグしたり、リバースエンジニアリングするのを防ぐためです。

- スクリーンセットAnimatorでは、dsexcept.dllモジュールを使用できません。

スクリーンセットは実行時にどれだけのメモリを占有するか?

スクリーンセットが実行時に占有するメモリを正確に判断するのは不可能です。占有量は、利用可能な空きメモリの量を含む、いくつかの要因によって変化します。しかし、スクリーンセットが使用するメモリの量は、一般的にあまり大きくはありません。

すべてのスクリーンセットは仮想ファイルに保持されます。仮想ファイルは、メモリ要求が増えるとディスクにページアウトされます。各スクリーンセットが確実に占有するメモリは、COBOL ランタイムシステム（RTS）が仮想ファイルを管理するのに必要な 512 バイトのみです。

各仮想ファイルが開かれるたびに、512 バイトが必要です（64K バイトを超えている場合、さらに 512 バイトが必要です）。したがって、Dialog System が各スクリーンセットに1つの仮想ファイルを使用する場合、1つのスクリーンセットごとに少なくとも 512 バイトが必要です。

十分なメモリがある場合、RTS はできるかぎり多くの仮想ファイルをメモリ内に保持します。メモリの空きが少なくなると、RTS は仮想ファイルをディスクにページングします。

スクリーンセットによって占有されるメモリは変化しますが、極端に大きくなることはありません。

詳しい情報については、ヘルプの「*Dialog Systemの制約*」トピックを参照してください。

テキストフィールドに色を付けるには、どうしたらよいか？

Dialog Systemのテキストオブジェクトは、ウィンドウに描いた単純なテキストです。そのために、色を設定することはできません。しかし、入力フィールドオブジェクトを使うことによって、必要な効果を得ることはできます。入力フィールドの表示のみ属性を選択し、テキストの初期値(またはマスタフィールド値)を定義すると、通常の方法で色

を設定した場合と同じ結果になります。

リストボックスが含まれるスクリーンセットをもっと速く表示させたい。どうしたらよいか？

リストボックスがマスタフィールドグループに制限される場合、SET-DATA-GROUP-SIZEファンクションを使って、そのグループ内部で認識されたサイズを一時的に調整することによって、ロード時間を短縮することができます。これは、Dialog Systemにそのグループ配列の(n)項目だけを内部的に認識することを伝え－データがオブジェクトに挿入された数を超えることがないようにします。しかし、呼出し中のプログラムは、そのグループのあらゆるデータ項目にアクセスすることができ、ユーザーは同じファンクションを使って、適切に内部のサイズをリセットすることができます。

複数のスクリーンセットを使うと、アプリケーションが制御されたループに入るようだ。それはなぜか？

これは、ルータにもとづくアプリケーションで、DS-PUSH-SETおよびDS-USE-SETを組み合わせて使ったときに経験することがあります。具体的には、この現象はコード化されたダイアログスクリプトによるものとされており、アプリケーションが強制的にイベントベースのループに入ります。

この方法で複数のスクリーンセットを使う場合、次のダイアログを使って、前のスクリーンセットを再度アクティブにします。

```
OTHER-SCREENSET
        MOVE 1 OTHER-SET-EVENT-FLAG
        REPEAT-EVENT
        RETC
```

COBOLプログラムはDS-EVENT-SCREENSET-IDにもとづいて、イベントが発生したスクリーンセットを再ロードすることを決定します。

通常、エラー内容の説明は、DS-PROCEDUREに値を設定することによって実行されます。DS-PROCEDUREは、再ロードされたスクリーンセットのダイアログテーブルを実行します。実行された手続きは、独自のファンクションを実行し、しばしば元のスクリーンセットのイベントをさらに引き起こし、経験したような現象になります。

この現象の理由は、「*複数のスクリーンセット*」の章の「*イベントのシーケンス*」に説明されています。

OTHER-SCREENSETイベントに応じてスクリーンセットを再ロードする場合は、DS-PROCEDUREを設定しないようにします。

選択ボックスの自動挿入属性がオフに設定されている場合、ドロップダウンリストでリスト項目が重複するのはなぜか？

これは、主にコントロールを不適切に使うと起こります。

選択ボックスコントロールの適切な使用法は、あらかじめ定義されたリストから項目を1つ選択することです。必要な場合は、関連するマスタフィールドにユーザーが選択した項目を配置し、アプリケーションで適切に使うことができます。

ここで重要なことは、選択ボックスを作成するダイアログファンクションが、16進数の値"0A"で区切られたリスト項目を挿入するということです。マスタフィールドグループから選択ボックスに項目を挿入することはできません。

重複が起こるのは、関連するマスタフィールドのテキストに、リストに挿入された値以外のものが含まれており(つまりスペースで区切られている)、"0A"によって区切られていない場合です。重複が起こったように見えますが、項目が同じように区切られていないのです。

関連するマスタフィールドとREFRESH-OBJECTファンクションの不正な使い方が、この望ましくない動作を引き起こす主なメカニズムです。

コントロールのタブ順を制御したい。どうしたらよいか？

[コントロール]メニューオプションの[編集]を選択し、必要に応じて、表示されたリストの順番を並べ替えます。ヘルプの「*コントロール*」トピックを参照してください。

256色ビットマップリソースの.dllへのコンパイルが動作しない。

256色ビットマップは、分割ファイルとしてDialog Systemアプリケーションに含まれるよう指定しなければなりません。また、MFDSSW /B(n)環境変数を設定することも必要です。この環境変数はパレットの動作を決定するものであり、256色をサポートできるように設定する必要があります。

第17章 サンプルプログラム

この章には、次のオブジェクトに関するサンプルダイアログを紹介します。

- 入力フィールド
- プッシュボタン
- チェックボックス
- リストボックス
- スクロールバー
- タブコントロール

さらに次のサンプルプログラムもあります。

- Dsrnr
- Push-pop
- Custom1

17.1 入力フィールド

入力フィールドについての詳細は、*コントロールオブジェクト*の章を参照してください。この節では、次の操作に必要なダイアログについて述べます。

- 入力フィールドを妥当性検査する。
- 複数行の入力フィールドを編集する。
- 入力フィールドを持つスクロールバーを使用する。スクロールバーと関連付けられたイベントの項を参照。

17.1.1 入力フィールドの妥当性検査

入力フィールドは、VALIDATEファンクションによって妥当性検査されます。妥当性検査が失敗した場合、VAL-ERRORイベントが引き起こされます。入力フィールドは、次に示すように明示的に妥当性検査することができます。

```
VALIDATE AMOUNT-DEPOSITED-EF
```

親ウィンドウを妥当性検査することによって、入力フィールドを暗黙的に妥当性検査することもできます。次に例を示します。

```
VALIDATE DEPOSIT-WIN
```

フィールドが入っているウィンドウまたはダイアログボックスでユーザーがデータを受け入れた(たとえば､ユーザーがOKボタンまたはEnterキーを押した）ときに、フィールドを暗黙的に妥当性検査するとよいでしょう。この場合、ユーザーは情報を入力し、それを再確認してから、必要に応じて修正することができます。そして、そのあとに妥当性検査が行われます。妥当性検査が失敗した場合、エラーが生じたフィールドにフォーカスを戻すことができます。

```
1 BUTTON-SELECTED
2     VALIDATE $WINDOW
3     INVOKE-MESSAGE-BOX ERROR-MB "All fields OK" $REGISTER
4     RETC
5 VAL-ERROR
6     INVOKE-MESSAGE-BOX ERROR-MB ERROR-MSG-FIELD $REGISTER
7     SET-FOCUS $EVENT-DATA
```

行1**：**

```
 BUTTON-SELECTED
```

この例では、プッシュボタンを選択することによって妥当性検査を起動します。ウィンドウがフォーカスを失ったり、特定のフィールドがLOST-FOCUS イベントによってフォーカスを失った場合にも、妥当性検査が起動されます。たとえば、ユーザーがTabキーを使って別のコントロールに移動した場合、現在のコントロールのフォーカスが失われます。

行2**：**

```
    VALIDATE $WINDOW
```

この文は、現在のウィンドウにあるすべてのフィールドをあらかじめ設定された基準に従って妥当性検査します。エラーが検出された場合、VAL-ERRORイベントが引き起こされます。エラーが検出されない場合、Dialog Systemは通常どおり処理を続け、次のファンクションを実行します。

行3**〜**4**：**

```
    INVOKE-MESSAGE-BOX ERROR-MB "All fields OK" $REGISTER
    RETC
```

VAL-ERRORイベントは起こりませんでした。エラーが検出されず、プログラムに戻ることを示すメッセージボックス が起動されます。

行5**：**

```
 VAL-ERROR
```

VAL-ERROR イベントが検出されました。特殊レジスタ$EVENT-DATA は、妥当性検査が失敗した入力フィールドを識別します。

行6**：**

```
INVOKE-MESSAGE-BOX ERROR-MB ERROR-MSG-FIELD $REGISTER
```

メッセージボックスを使って、ユーザーにエラーを知らせます。2番目のパラメータERROR-MSG-FIELDには、この妥当性検査エラーに対して定義されたエラーメッセージが入っています。ERROR-MSG-FIELDは、データ定義内のエラーメッセージフィールドとして定義されます。

行7**：**

次のダイアログは、妥当性検査ステップのコーディング例を示しています。このダイアログはプッシュボタンに付属します。

```
SET-FOCUS $EVENT-DATA
```

妥当性検査に失敗したフィールドに入力フォーカスを戻します。

警告：LOST-FOCUSイベントを使うと、ユーザーがフィールドから離れようとしたときにそのフィールドを妥当性検査することができます。しかし、妥当性検査が失敗した場合、フォーカスがそのフィールド設定されているときは、ユーザーが別のアプリケーションにフォーカスを移動しようとしても、フォーカスが常にそのフィールドに戻ります。このような場合、ウィンドウ全体を妥当性検査する方がよいでしょう。

17.1.1.1 複雑なデータ妥当性検査

前の項で述べた手順では、簡単なデータ妥当性検査を取り扱いました。しかし、Dialog Systemではとても複雑な妥当性検査が必要な場合もあります。たとえば、信用限界金額の範囲は、顧客と会社の付き合いの年数によって決まることがよくあります。

このような複数フィールドにまたがった妥当性検査では、適切な介入が必要です。これは、スクリーンセットに入っていないデータが必要か、Dialog Systemに用意されている規則より複雑な妥当性検査が必要なためです、このようなチェックを行うには、呼出しプログラムに戻らなければなりません。

例については、*高度なトピック*の章を参照してください。

17.1.2 複数行入力フィールドの編集

アプリケーションプログラムまたはダイアログを使って、テキストを関連するデータ項目に移動することができます。

17.1.2.1 アプリケーションプログラムを使ったテキストの移動

アプリケーションプログラムの中でテキストを関連するデータ項目に移動する例とし、次の文があります。

```
move "Now is the time..." to large-entry-field
```

これは、文字列を複数行入力（MLE）と関連付けられたデータブロック項目に移動します。

改行を挿入するには、16進文字"0a"を使用します。たとえば、

```
move "line1" & x"0a" & "line2"
```

ただし、

`line1`と`line2`　は別々の行に表示されるテキストです。

`&`　文字列を連結します。

`x`　16進文字を表します。

`0a`　改行を示す16進文字です。

17.1.2.2 ダイアログを使ったテキストの移動

この場合、改行を挿入できません。たとえば、次の文をダイアログの中で使用します。

```
move "Now is the time..." large-entry-field
```

`Large-entry-field`は、MLEと関連付けられたデータ項目です。

17.2 プッシュボタン

この節では、次のプッシュボタンに関するサンプルダイアログを紹介します。

- Pauseプッシュボタン
- プッシュボタンと関連して動的に変化するビットマップ

17.2.1 Pauseプッシュボタンのためのダイアログ

```
1 BUTTON-SELECTED
2     DISABLE-OBJECT $CONTROL
3     ENABLE-OBJECT CONTINUE-PB
4     BRANCH-TO-PROCEDURE PAUSE-SELECTED
```

行1：

```
BUTTON-SELECTED
```

ユーザーが停止ボタンを選択すると、 BUTTON-SELECTED イベントが引き起こされます。このイベントは、プッシュボタンと関連付けられた一次イベントです。

行2：

```
DISABLE-OBJECT $CONTROL
```

$CONTROL は、現在選択されているコントロールを識別する特殊レジスタです。このファンクションは、現在のコントロール（ここでは、停止プッシュボタン）を使用禁止にします。これによって、ユーザーはこのボタンを使用できなくなります。

行3：

```
ENABLE-OBJECT CONTINUE-PB
```

この文は、継続プッシュボタンを使用可能にします。これによって、このボタンは選択できるようになります。CONTINUE-PBは、プッシュボタン属性ウィンドウにあるプッシュボタンに割り当てられたプッシュボタンの名前です。

行4：

```
BRANCH-TO-PROCEDURE PAUSE-SELECTED
```

PAUSE-SELECTED手続きのファンクションが実行されます。この手続きでは、フラグをセットし、呼出しプログラムに戻ることができます。

プッシュボタンの定義については、*コントロールオブジェクト*の章を参照してください。

17.2.2 プッシュボタンに指定されたビットマップを動的に変更するためのダイアログ

次のダイアログは、プッシュボタンに割り当てられたビットマップを動的に変更する方法を示しています。

注意：ビットマップを使用するには、Dialog Systemで利用可能にする必要があります。詳しくは*コントロールオブジェクト*の章を参照してください。

```
1 ...
2 @SAVE
3     BRANCH-TO-PROCEDURE SAVE-FUNCTION
```

```
4   SAVE-FUNCTION
5     SET-OBJECT-LABEL GENERIC-PB SAVE-STATES
6     SET-FOCUS GENERIC-WIN
7     ...
```

行2**〜**3**：**

```
 @SAVE
     BRANCH-TO-PROCEDURE SAVE-FUNCTION
```

ユーザーがメニューから保存オプションを選択します。保存機能を実行する手続きに分岐します。

行5**：**

```
    SET-OBJECT-LABEL GENERIC-PB SAVE-STATES
```

GENERIC-PB は、プッシュボタンに割り当てられた名前です。SAVE-STATESは、置き換えるビットマップの名前と3つのヌル文字（x"0A"）をいれるのに十分な大きなの英数字データ項目です。

行6**：**

```
    SET-FOCUS GENERIC-WIN
```

キーボードフォーカスをウィンドウに設定します。ウィンドウ内のすべてのオブジェクトが再生され、処理が続行されます。

次のような文を使ってデータブロックのsave-statesにデータを移動します。

```
    move "save-normal" & x"0A" & "save-disabled" & x"0A" &
                "save-depressed" & x"0A"  to save-states.
```

ただし、save-normal、save-disabled、save-depressedは、ビットマップの名前であり、プッシュボタンの状態によっていずれかが表示されます。名前を指定する順序は重要です。

1番目の名前は、プッシュボタンが"通常"状態のときに表示されるビットマップです。2番目の名前は、プッシュボタンが使用禁止になっているときに表示されるビットマップです。3番目の名前は、プッシュボタンが押されているときに表示されるビットマップです。

17.3 チェックボックス

チェックボックスについての詳細は、*コントロールオブジェクト*の章を参照してください。Objectsサンプルプログラムは、チェックボックスの使い方を示しています。

17.3.1 リスト項目の選択

この例では、リストから1つまたは複数の製品を選びます。そして表示プッシュボタンをクリックします、選択された項目がリストボックスに表示されます。図 17-1 を参照してください。

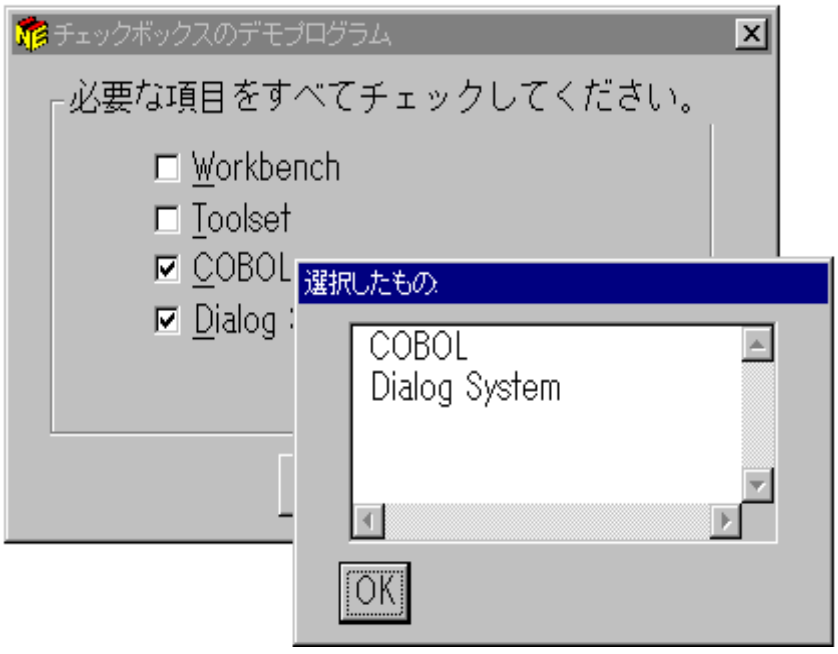


図 17-1 チェックボックスとリストボックス

このサンプルスクリーンセットのデータ定義は、次のようになっています。

```
1   WORKBENCH                   9        1.00
2   TOOLSET                     9        1.00
3   COBOL                       9        1.00
4   DIALOG-SYSTEM               9        1.00
5   PRODUCTS                        4
6     PRODUCT-DISPLAY           X       15.00
```

データ項目1から4は、対応するチェックボックスに関連付けられているデータ項目です。データ項目5と6は、選択された項目を表示するリストボックスに使用されます。

この例を制御するためのダイアログ（表示プッシュボタンに付属する）を次に示します。

```
 1 BUTTON-SELECTED
 2     MOVE "              " PRODUCT-DISPLAY(1)
 3     MOVE "              " PRODUCT-DISPLAY(2)
 4     MOVE "              " PRODUCT-DISPLAY(3)
 5     MOVE "              " PRODUCT-DISPLAY(4)
 6     MOVE 1 $REGISTER
```

```
 7    IF= WORKBENCH 0 CHECK-TOOLSET
 8    MOVE "Workbench" PRODUCT-DISPLAY($REGISTER)
 9    INCREMENT $REGISTER
10    BRANCH-TO-PROCEDURE CHECK-TOOLSET
11
12  CHECK-TOOLSET
13    IF= TOOLSET 0 CHECK-COBOL
14    MOVE "Toolset" PRODUCT-DISPLAY($REGISTER)
15    INCREMENT $REGISTER
16    BRANCH-TO-PROCEDURE CHECK-COBOL
17
18  CHECK-COBOL
19    IF= COBOL 0 CHECK-DIALOG-SYSTEM
20    MOVE "COBOL" PRODUCT-DISPLAY($REGISTER)
21    INCREMENT $REGISTER
22    BRANCH-TO-PROCEDURE CHECK-DIALOG-SYSTEM
23
24  CHECK-DIALOG-SYSTEM
25    IF= DIALOG-SYSTEM 0 DISPLAY-PRODUCTS-DB
26    MOVE "Dialog System" PRODUCT-DISPLAY($REGISTER)
27    BRANCH-TO-PROCEDURE DISPLAY-PRODUCTS-DB
28
29  DISPLAY-PRODUCTS-DB
30    REFRESH-OBJECT CHECKB-LB
31    SET-FOCUS CHECKB-DB
```

行1**：**

```
 BUTTON-SELECTED
```

ユーザーが表示プッシュボタンを選択します。

行2**〜**5**：**

```
    MOVE "             " PRODUCT-DISPLAY(1)
```

```
MOVE "            " PRODUCT-DISPLAY(2)
MOVE "            " PRODUCT-DISPLAY(3)
MOVE "            " PRODUCT-DISPLAY(4)
```

PRODUCT-DISPLAYグループ内の項目に空白文字列を移動することによって、リストボックスの項目を初期設定します。

行6**：**

```
MOVE 1 $REGISTER
```

インデックス（$REGISTER）を初期設定します。$REGISTERは、数値の格納に使用できる内部レジスタです。ここでは、リストボックスへのインデックスとして使用します。

行7**：**

```
IF= WORKBENCH 0 CHECK-TOOLSET
```

If Workbenchチェックボックスがチェックされていない場合、Toolsetチェックボックスがチェックされているかを調べるための手続きに分岐します。Workbenchチェックボックスがチェックされている場合、分岐せずに次のファンクションを続行します。

行8**〜**9**：**

```
MOVE "Workbench" PRODUCT-DISPLAY($REGISTER)
INCREMENT $REGISTER
```

ユーザーがWorkbenchチェックボックスを選択した場合、ここにききます。リストボックスのデータ項目にデータを書き込み、インデックスを増加させます。

行10**：**

```
BRANCH-TO-PROCEDURE CHECK-TOOLSET
```

Toolsetチェックボックスを調べる手続きに分岐します。

行12**〜**27**：**

```
CHECK-TOOLSET
  IF= TOOLSET 0 CHECK-COBOL
  MOVE "Toolset" PRODUCT-DISPLAY($REGISTER)
  INCREMENT $REGISTER
  BRANCH-TO-PROCEDURE CHECK-COBOL
```

```
CHECK-COBOL
  IF= COBOL 0 CHECK-DIALOG-SYSTEM
  MOVE "COBOL" PRODUCT-DISPLAY($REGISTER)
  INCREMENT $REGISTER
  BRANCH-TO-PROCEDURE CHECK-DIALOG-SYSTEM

CHECK-DIALOG-SYSTEM
  IF= DIALOG-SYSTEM 0 DISPLAY-PRODUCTS-DB
  MOVE "Dialog System" PRODUCT-DISPLAY($REGISTER)
  BRANCH-TO-PROCEDURE DISPLAY-PRODUCTS-DB
```

これは、他のチェックボックスと同じダイアログです。

行29**〜**31**：**

```
DISPLAY-PRODUCTS-DB
  REFRESH-OBJECT CHECKB-LB
  SET-FOCUS CHECKB-DB
```

選択された製品をリストボックスに表示します。

17.4 リストボックス

リストボックスの項目を更新するには、3つの方法があります。

- グループ項目を使用する。
- 実行時に項目を更新するダイアログを使用する。
- 区切り付き文字列を使用する。

17.4.1 グループ項目を使った項目の追加

リストボックスの各行がグループのオカレンス（要素）を示すように、リストボックスはグループ項目と関連付けることができます。グループ内の選択されたリストボックスが各行のフィールドを占有します。

Saledataサンプルアプリケーションは、この方法によるリストボックスへのアクセスを示しています。アプリケーションは、既存のファイルを使ってデータブロックのグループ項目（SALES-GROUP）に販売データセットをロードします。このデータはリストボックス（SALES-LB）に表示されます。項目を更新（挿入、変更、削除）したり、データセットを別の順序で表示することができます。

データブロックセクションは、次のダイアログで参照されるデータを定義します。

```
SALES-GROUP            100
  SALES-NAME       X       20.00
  SALES-REGION     X        4.00
  SALES-STATE      X        2.00
TMP-NAME           X       20.00
TMP-REGION         X        4.00
TMP-STATE          X        2.00
NUMBER-OF-RECORDS  9        3.00
```

次のダイアログは、リストボックスを取り扱います。

```
 1    SET-DATA-GROUP-SIZE SALES-GROUP NUMBER-OF-RECORDS
 2 ...
 3 ITEM-SELECTED
 4    MOVE $EVENT-DATA $REGISTER
 5 ...
 6  PROC-INSERT
 7    INSERT-OCCURRENCE SALES-GROUP $REGISTER
 8    MOVE TMP-NAME SALES-NAME($REGISTER)
 9    MOVE TMP-REGION SALES-REGION($REGISTER)
10    MOVE TMP-STATE SALES-STATE($REGISTER)
11    INCREMENT NUMBER-OF-RECORDS
12    INCREMENT $REGISTER
13    REFRESH-OBJECT SALES-LB
14    SET-LIST-ITEM-STATE SALES-LB 1 $REGISTER
15 ...
16  PROC-CHANGE
17    MOVE TMP-NAME SALES-NAME($REGISTER)
18    MOVE TMP-REGION SALES-REGION($REGISTER)
19    MOVE TMP-STATE SALES-STATE($REGISTER)
20    UPDATE-LIST-ITEM SALES-LB SALES-GROUP $REGISTER
21    SET-LIST-ITEM-STATE SALES-LB 1 $REGISTER
```

```
22 ...
23   PROC-DELETE
24     DELETE-OCCURRENCE SALES-GROUP $REGISTER
25     DECREMENT NUMBER-OF-RECORDS
26     DECREMENT $REGISTER
27     REFRESH-OBJECT SALES-LB
28     SET-LIST-ITEM SALES-LB 1 $REGISTER
```

行1**：**

```
    SET-DATA-GROUP-SIZE SALES-GROUP NUMBER-OF-RECORDS
```

この文は、データグループの内部サイズを定義します。これは、アクセス可能なグループの要素の数です。内部サイズの定義についての詳細は、ヘルプのSET-DATA-GROUP-SIZEファンクションに関する説明を参照してください。

行3**〜**4**：**

```
 ITEM-SELECTED
     MOVE $EVENT-DATA $REGISTER
```

選択したい項目が見つかるまで、リストボックスを拾い読みすることができます。項目（リストボックスの行）を選択されると、 ITEM-SELECTED イベントが引き起こされます。特殊イベントレジスタ$EVENT-DATAには、選択された項目の行番号が入っています。グループデータとリストボックスの両方で現在の場所を追跡するには、$REGISTERを使用します。

行6**：**

```
   PROC-INSERT
```

この手続きは、選択された項目によって現在占有されている位置に項目を挿入します。

行7**：**

```
     INSERT-OCCURRENCE SALES-GROUP $REGISTER
```

空白の要素をデータグループの指定された位置に挿入します。選択された項目とそれに続くすべての項目が、1行下に移動します。リストボックスは影響を受けません。このファンクションはデータグループを更新するだけです。

行8**〜**10**：**

```
     MOVE TMP-NAME SALES-NAME($REGISTER)
     MOVE TMP-REGION SALES-REGION($REGISTER)
     MOVE TMP-STATE SALES-STATE($REGISTER)
```

グループ項目を新しい情報で更新します。

行11：

```
INCREMENT NUMBER-OF-RECORDS
```

データグループ内のレコード数を増やします。

行12：

```
INCREMENT $REGISTER
```

ポインタを現在の行へ更新します。これによって、ポインタは挿入前と同じ行を指し示します。

行13：

```
REFRESH-OBJECT SALES-LB
```

グループ項目と関連付けられたリストボックスは、データの変更を反映するまえに再生しなければなりません。INSERT-OCCURRENCEファンクションとDELETE-OCCURRENCEファンクションは、リストボックスには影響を与えません。リストボックスに再生が必要な項目が1つしかない場合、UPDATE-LIST-ITEMファンクションが使用できます。しかし、挿入の場合、データグループ内の挿入行より後のすべての行が変更されます。したがって、REFRESH-OBJECTファンクションを使用した方がよいといえます。

行14：

```
SET-LIST-ITEM-STATE SALES-LB 1 $REGISTER
```

現在行の状態を"選択"にします。これは、挿入前に選択されていた行と同じです。

行16：

```
PROC-CHANGE
```

この手続きは、選択された項目の内容を変更します。

行17〜19：

```
MOVE TMP-NAME SALES-NAME($REGISTER)
MOVE TMP-REGION SALES-REGION($REGISTER)
MOVE TMP-STATE SALES-STATE($REGISTER)
```

グループ項目を新しい情報で更新します。

行20：

```
UPDATE-LIST-ITEM SALES-LB SALES-GROUP $REGISTER
```

リストボックスの1つの行だけが変更されるので、UPDATE-LIST-ITEMファンクションが使用できます。他のすべて

の行は同じなので、リストボックス全体を再生する必要はありません。

行21**：**

```
SET-LIST-ITEM-STATE SALES-LB 1 $REGISTER
```

現在の行の状態を"選択"にします。これは、変更前に選択された行と同じです。

行24**：**

```
DELETE-OCCURRENCE SALES-GROUP $REGISTER
```

データグループの選択された位置から要素を削除します。その後のすべての項目が1行だけ上に移動されます。

行25**：**

```
DECREMENT NUMBER-OF-RECORDS
```

データグループ内のレコード数を減らします。

行26**：**

```
DECREMENT $REGISTER
```

ポインタを現在の行に更新します。これによって、ポインタは変更前と同じ行を指し示します。

行27**：**

```
REFRESH-OBJECT SALES-LB
```

リストボックスを再生します。

行28**：**

```
SET-LIST-ITEM SALES-LB 1 $REGISTER
```

現在行の状態を"選択"にします。これは、削除された行の次の行と同じです。

17.4.2 ダイアログを使った項目の追加

リストボックスの項目は、ダイアログファンクションINSERT-LIST-ITEM、UPDATE-LIST-ITEM、DELETE-LIST-ITEMを使って管理することもできます。

たとえば、次のダイアログは、月名の省略形を`MONTH-LB`というリストボックスに追加する方法を示しています。

```
SCREENSET-INITIALIZED
   INSERT-LIST-ITEM MONTH-LB "Jan" 1
   INSERT-LIST-ITEM MONTH-LB "Feb" 2
```

```
    INSERT-LIST-ITEM MONTH-LB "Mar" 3
    INSERT-LIST-ITEM MONTH-LB "Apr" 4
    INSERT-LIST-ITEM MONTH-LB "May" 5
    INSERT-LIST-ITEM MONTH-LB "Jun" 6
    INSERT-LIST-ITEM MONTH-LB "Jul" 7
    INSERT-LIST-ITEM MONTH-LB "Aug" 8
    INSERT-LIST-ITEM MONTH-LB "Sep" 9
    INSERT-LIST-ITEM MONTH-LB "Oct" 10
    INSERT-LIST-ITEM MONTH-LB "Nov" 11
    INSERT-LIST-ITEM MONTH-LB "Dec" 12
```

これは、リストの選択項目が少なく、データブロックをできるだけ小さくしたい場合、リストボックスに項目を追加するよい方法です。この方法で小さなリストボックスに書き込むにはそれほど手間はかかりませんが、多少の時間は必要で、しかも検索すべきダイアログ文の数に従って増加します。ダイアログを検索する際の規則については、*ダイアログの使い方*の章の *Dialog System がイベントダイアログを探す方法*を参照してください。

別の方法として、次のダイアログを使用することもできます。

```
SCREENSET-INITIALIZED
    INSERT-LIST-ITEM MONTH-LB "Jan" 0
    INSERT-LIST-ITEM MONTH-LB "Feb" 0
    INSERT-LIST-ITEM MONTH-LB "Mar" 0
    INSERT-LIST-ITEM MONTH-LB "Apr" 0
    INSERT-LIST-ITEM MONTH-LB "May" 0
    INSERT-LIST-ITEM MONTH-LB "Jun" 0
    INSERT-LIST-ITEM MONTH-LB "Jul" 0
    INSERT-LIST-ITEM MONTH-LB "Aug" 0
    INSERT-LIST-ITEM MONTH-LB "Sep" 0
    INSERT-LIST-ITEM MONTH-LB "Oct" 0
    INSERT-LIST-ITEM MONTH-LB "Nov" 0
    INSERT-LIST-ITEM MONTH-LB "Dec" 0
```

3番目のパラメータの値`0`は、Dialog Systemに対して、リストの終わりにデータ項目を挿入するように指示します。

17.4.3 区切り付き文字列を使った項目の追加

リストボックスに少数の項目を追加する もう1つの方法として、プログラムから渡されたデータ項目といっしょに

INSERT-MANY-LIST-ITEMS ファンクションを使うことができます。たとえば、次のダイアログは、前と同じ月名の省略形をMONTH-LBに挿入します。

```
SCREENSET-INITIALIZED
    INSERT-MANY-LIST-ITEMS MONTH-LB MONTHS-STRING 48
```

各パラメータの意味を次に示します。

MONTH-LB	リストボックス属性ダイアログボックスで定義されたリストボックスの名前。
MONTHS-STRING	プログラムから渡されたデータ項目。個々のリスト項目はx"0A"によって区切られています。データ定義機能で次のように定義されています。 `MONTHS-STRING X 48`
48	リストボックスにコピーする文字数。このパラメータについての詳細は、ヘルプの*INSERT-MANY-LIST-ITEMS*トピックを参照してください。

プログラムのWorking-Storage Sectionで次のような文字列を定義します。

```
01 months.
   03 pic x(4) value "Jan" & x"0A".
   03 pic x(4) value "Feb" & x"0A".
   03 pic x(4) value "Mar" & x"0A".
   03 pic x(4) value "Apr" & x"0A".
   03 pic x(4) value "May" & x"0A".
   03 pic x(4) value "Jun" & x"0A".
   03 pic x(4) value "Jul" & x"0A".
   03 pic x(4) value "Aug" & x"0A".
   03 pic x(4) value "Sep" & x"0A".
   03 pic x(4) value "Oct" & x"0A".
   03 pic x(4) value "Nov" & x"0A".
   03 pic x(4) value "Dec" & x"0A".
```

次のような文によって、文字列をデータブロックに移動します。

```
move months to months-string
```

17.5 スクロールバー

この節では、次のことについて説明します。

- スクロールバーと関連づけられたイベント
- スクロールバーの属性

17.5.1 スクロールバーと関連づけられたイベント

スクロールバーには、SLIDER-MOVINGとSLIDER-RELEASEDの2つのイベントが関連付けられています。

SLIDER-MOVINGイベントは、スライダが新しい位置に移動されたときに起こります。

たとえば、次のダイアログを考えてみます。

```
1 SLIDER-MOVING
2     MOVE $EVENT-DATA COUNTER
3     REFRESH-OBJECT COUNTER-DISP
4 SLIDER-RELEASED
5     MOVE $EVENT-DATA COUNTER
6     REFRESH-OBJECT COUNTER-DISP
```

行1と3は、スライダの位置を入力フィールドCOUNTER-DISPに表示します。.

SLIDER-RELEASEDイベントは、スライダが解放されたときに起こります。ダイアログの行4〜6は、このイベントを使用し、さらにスライダの位置を表示します。以下にダイアログの各行を詳しく見てゆきます。

行1：

```
SLIDER-MOVING
```

スライダが新しい位置に移動されました。これによって、SLIDER-MOVINGイベントが引き起こされます。

行2：

```
MOVE $EVENT-DATA COUNTER
```

スライダを動かすと、特殊レジスタ$EVENT-DATA に新しいスライダ位置が書き込まれます。これは、最小値と最大値に対する相対的な位置です。COUNTERは、入力フィールドCOUNTER-DISPに付属するデータブロックのデータ項目です。

行3：

```
REFRESH-OBJECT COUNTER-DISP
```

入力フィールドは、新しいデータ値を反映できるように再生しなければなりません。

行4**：**

```
SLIDER-RELEASED
```

スライダが新しい位置で解放されました。これによって、SLIDER-RELEASEDイベントが引き起こされます。

行5**：**

```
MOVE $EVENT-DATA COUNTER
```

スライダが解放されたとき、$EVENT-DATAには新しいスライダ位置が入っています。ここでもCOUNTERは、入力フィールドCOUNTER-DISPに付属するデータブロックのデータ値です。

行6**：**

```
REFRESH-OBJECT COUNTER-DISP
```

入力フィールドを再生します。

サンプルアプリケーションObjectsにあるスクロールバーの例は、これと同じ入力フィールドの使い方を示しています。

17.5.2 スクロールバーの属性

スクロールバー属性ダイアログボックスを使うと、スライダ範囲、スライダ位置、スライダサイズなどのスクロールバーの属性にデフォルト値を割り当てることができます。これらの属性は、ダイアログによって変更できます。

たとえば、次のダイアログは、EMP-LIST-SBというスクロールバーの属性を変更します。

```
SET-SLIDER-RANGE EMP-LIST-SB 0 50
SET-SLIDER-POSITION EMP-LIST-SB 25
SET-SLIDER-SIZE EMP-LIST-SB 5
```

これらのファンクションについての詳細は、ヘルプの*「ダイアログの定義: ファンクション」*トピックを参照してください。

スクロールバーの定義については、ヘルプの*オブジェクトと属性*トピックを参照してください。

17.6 タブコントロール

ファンクションやDELETE-PAGE ファンクションを使うと、タブコントロールからページを挿入したり、削除することができます。これによって、タブコントロールを動的に管理できます。

たとえば、住所、交渉履歴、販売情報などの顧客情報を管理するアプリケーションを作成するとします。それぞれの

顧客について、この情報をタブコントロールに表示します。

図 17-2 を参照してください。

図 17-2 タブコントロールページの例

もし、1ページの販売情報では不足する場合、次のようにして2ページ目を追加することができます。

```
1    ADD-MORE-SALES-INFO
2      COPY-PAGE SALES-INFO-PAGE 0 2
3      SET-OBJECT-LABEL SALES-INFO-PAGE "Sales Information 2"
```

行1**：**

```
ADD-MORE-SALES-INFO
```

タブコントロールに販売情報ページを追加するための手続き。

行2**：**

```
    COPY-PAGE SALES-INFO-PAGE 0 2
```

COPY-PAGEはタブコントロールのページをコピーし、正しい位置に挿入します。SALES-INFO-PAGEは、そのページに割り当てられる名前です。2番目のパラメータ0は、コピーする位置を示します。0は最後、1は最初です。3番目のパラメータは、ページのインスタンス番号です。インスタンス番号は、ページに表示されるマスタフィールド付きの入力フィールドのより優先されます。しかし、タブコントロール内に作成したページは、明示的に削除しないかぎり（下を参照）常に表示されることを忘れないでください。

行3**：**

```
    SET-OBJECT-LABEL SALES-INFO-PAGE "Sales Information 2"
```

新しいページのタブ内のテキストを変更します。COPY-PAGEの後、SALES-INFO-PAGEは、もとのページではなく、新しいページを参照します。

タグコントロールからページを削除するには、DELETE-PAGEファンクションを使用します。たとえば、顧客が販売情報を持っていない場合、次のようにしてページを削除することができます。

```
BUTTON-SELECTED
    DELETE-PAGE SALES-INFO-PAGE
```

これらのファンクションについての詳細は、ヘルプを参照してください。

17.7 呼出しインタフェース

呼出しインタフェースについての詳細は、*スクリーンセットの使い方*の章を参照してください。この節では、次のことについて説明します。

- Dsrnrの使い方
- スクリーンセットのプッシュとポップ
- 複数のスクリーンセットインタフェースの使用

17.7.1 Dsrnrの使い方

Dsrnrは、Dialog Systemに付属のサンプルサブプログラムです。Dsrnrを起動するには、次の手順に従ってください。

1. Dsrunnerを起動します（詳しくは*複数のスクリーンセット*の章を参照）。

   runw dsrunner

2. Dsrunnerウィンドウの[ファイル]メニューから[スクリーンセットを開く]を選択します。

3. ファイルセレクタ（デモンストレーションディレクトリにある）が表示されたら、Dsrnrを選択します。

   Dsrnrのメインウィンドウが表示されます。Dsrnrインスタンスは、何回も開くことによって複数ロードできます（スタックには、最大32個のスクリーンセットを持つことができます）。サブプログラムは、Animatorによって実行することも可能です。

サンプルコードの主要部分を詳しく見て行きましょう。

```
14 linkage section.
15
```

```
16* スクリーンセットからのデータブロック
17 copy "dsrnr.cpb".
18
19* （オプション）dsrunnerの情報とdsイベントブロック
20 copy "dsrunner.cpy".
21 copy "dssysinf.cpy".
```

スクリーンセットデータにアクセスするには、データブロックのCOPYファイルをプログラムにコピーしなければなりません。オプションで、Dsrunnerとサブプログラムdsrunner.cpyの間のリンケージを表すdsrunnerCOPYファイルもコピーできます。Panels2イベント情報を処理したい場合、dssysinf.cpyもコピーする必要があります。これらは、サブプログラムのパラメータなので、linkage sectionにコピーします。

```
24 procedure division using data-block
25 dsrunner-info-block 26 ds-event-block.
```

data-blockパラメータを指定しなければなりません。他の2つのパラメータはオプションです。エラー通知が必要な場合、dsrunner-info-blockを使用します。Panels2とDialog Systemをいっしょに使用したい場合、ds-event-blockを使用します。

```
27 main section.
28* DSRUNNERから呼び出すのではなく、
29* コマンド行から実行した場合、メッセージを消す。
30 if (address of data-block = null)
31 display "This is a subprogram, and must be CALLed"
32* "(Preferably from DSRUNNER: できるだけDSRUNNERから実行してください)"と表示する
display "(Preferably from DSRUNNER)."
33 exit program
34 stop run
35 end-if
```

サブプログラムが呼び出されることとを確認します。これには、データブロックのアドレスがサブプログラムに渡されるかをチェックするのが簡単です。

```
37 move 0 to return-code
38
39* このデータブロックで呼び出されたのが初めてか
40* どうかを調べる。これには、データブロックの低値を
```

```
41* チェックする。DSRUNNERは、この値を使ってすべての
42* データブロックを初期設定する。
43 if (data-block = all low-values)
44* このスクリーンセット／Data-Blockで呼び出されたのが
45* 初めての場合、初期設定を行ってから、終了する。
46* スクリーンセットは、ここから戻るまでは
47* ロードされない（SCREENSET-INITIALIZEDイベント
48* も起こらない）。
49 perform initialisation
50 else
51* これが初めての呼び出しではない場合、
52* 通常の取り扱いを行う。
53 perform handle-screenset-request
54 end-if
55
56 continue.
57
58 exit-main.
59
60 exit program
61 stop run.
```

サブプログラムが新しいインスタンスとして呼び出されるたびに、データブロックを初期設定しなければなりません。行43は、このデータブロック（スクリーンセットの新しいインスタンス）によってサブプログラムが初めて呼び出されたのかどうかをチェックする方法を示しています。サブプログラムが初期設定を終了したら、Dsrunnerに戻らなければなりません。このサブプログラムに関するスクリーンセットの中でRETCが発生すると、Dsrunnerはユーザープログラムを呼び出します。

```
64 initialisation section.
65* pre-screensetの初期設定を行う。Perform any pre-screenset initialisation here.
66
67 initialize data-block
68 move dsrunner-screenset-instance to my-instance-no
```

```
69
70 continue.
```

行68は、サブプログラムがインスタンス番号を見つける方法を示しています。

```
73 handle-screenset-request section.
74* スクリーンセットがRETCを行うか、
75* DSGRUNエラーが起こった場合、ここで終了する。
76* 終了前にDSRUNNERがエラー取り扱いを指示する。
77
78* スクリンセットがエラーを起こしたために呼び出されたのか？
79 if (dsrunner-error-code not = 0)
80* その場合、エラーを取り扱う。
81 perform handle-dsgrun-error
82 exit section
83 end-if
84* 妥当性検査エラーのために呼び出されたのか？
85 if (dsrunner-validation-error-no not = 0)
86* その場合、エラーを取り扱う。
87 perform handle-validation-error
88 exit section
89 end-if
90
91* スクリーンセットからの通常のRETCでなければならない
92* ので、スクリーンセット要求をサービスするだけ。
93
94 move "successful" to program-string
95
96 evaluate reason-for-returning
97 when "+"
98 add program-value-1 to program-value-2
99 giving result-value
100
```

```
101 when "-"
102 subtract program-value-1 from program-value-2
103 giving result-value
104
105 when "*"
106 multiply program-value-1 by program-value-2
107 giving result-value
108
109 when "/"
110 divide program-value-1 by program-value-2
111 giving result-value
112 on size error
113 move "Bad result from divide" to program-string
114
115 when other
116 move "sorry, unsupported function" to program-string
117
118 end-evaluate
119
120 continue.
```

上のコードは、このサブプログラムに関するスクリーンセットの中でRETCダイアログに続くスクリーンセット要求を取り扱います。

行79と行85では、Dsrunnerエラーまたは何らかの妥当性検査エラーがないかをチェックします。エラーが起こった場合、適切に処理されます。エラーがない場合、スクリーンセット要求が取り扱われます。

```
123 handle-dsgrun-error section.
124* スクリーンセットでDSGRUNエラーが起こった場合、ここで終了する。
125* この例では、エラーメッセージ自身を表示することにした。
126* 起こったのが初めての場合、処理を続ける。そうでない場合、
myself. If this is the first time this has occurred, I
127* DSRUNNERによる終了を示すRETURN-CODEを設定する。
continue, otherwise I set the RETURN-CODE indicating that
```

```
128* return-codeがゼロの場合、
DSRUNNER should terminate me. If the return-code is zero
129* DSRUNNERは何もなかったかのように処理を続ける。
130
131 move dsrunner-error-code to error-number
132 move dsrunner-error-details-1 to error-details1
133 move dsrunner-error-details-2 to error-details2
134 display "dsrnr: "
135 "dsgrun error " error-number
136 ", " error-details1
137 ", " error-details2
138
139 if (handle-error-count = 0)
140 add 1 to handle-error-count
141 else
142* dsrunnerによって強制的に終了される。
143 move 1 to return-code
144 end-if
145
146 continue.
```

上のコードはdsrunnerエラーを取り扱います。

行143は、非ゼロの戻りコードを返すことによってDsrunnerを終了する方法を示しています。

```
149 handle-validation-error section.
150* スクリーンセットの妥当性検査エラーが起こった場合、
We end up here when a screenset validation error has
151* ここで終了する。ここでも上記と同じ規則が適用される。
152
153 move dsrunner-validation-error-no to error-number
154
155 display "dsrnr: validation error code ", error-number
156
```

```
157* dsrunnerによって強制的に終了される。
158 move 1 to return-code
159
160 continue.
```

上のコードは、ユーザー入力エラーを取り扱います。

17.7.2 プッシュポップサンプルプログラム

次のサンプルプログラムpush-pop.cblは、スクリーンセットのプッシュとポップの使い方を示します。

COBOLプログラム、スクリーンセット、関連するファイルは、サンプルディスクに入っています。これらのプログラムのコンパイルおよび実行する方法については、*スクリーンセットの使い方*の章を参照してください。

Push-popは、次の3つのスクリーンセットを使用します。

- ファイルマネージャスクリーンセット
- プリントマネージャスクリーンセット
- メインスクリーン。ここから他のスクリーンセットを呼び出す。

プログラムのリスト全体を次に示します。プッシュとポップに使われるファンクションについては、リストのあとで説明します。

```
1    $SET ANS85 MF
2
3    working-storage section.
4        copy "ds-cntrl.mf  ".
5        copy "pushmain.cpb ".
6        copy "filemgr.cpb  ".
7        copy "printmgr.cpb ".
8
9    01 new-screenset-name       pic x(12).
10
11   01 action                   pic 9.
12      78 load-file                    value 1.
13      78 load-print                   value 2.
14      78 exit-program                 value 3.
```

```
01 end-of-actions-flag     pic 9.
   88 end-of-actions               value 1.

procedure division.

main-process.
    perform program-initialize
     call "dsgrun" using ds-control-block,
                     pushmain-data-block
    perform process-actions until end-of-actions
stop run.

program-initialize.
    initialize ds-control-block
    initialize pushmain-data-block
    move ds-new-set to  ds-control
    move pushmain-data-block-version-no to
       ds-data-block-version-no
    move pushmain-version-no to ds-version-no
    move "pushmain" to ds-set-name
    move zero to end-of-actions-flag.

process-actions.
    evaluate true
      when pushmain-action = load-file
        move "filemgr" to ds-set-name
        move ds-push-set to ds-control
        move 1 to ds-control-param
        initialize filemgr-data-block
        move filemgr-data-block-version-no to
          ds-data-block-version-no
        move filemgr-version-no to ds-version-no
```

```
47          call "dsgrun" using ds-control-block,
48                              filemgr-data-block
49          perform file-mgr-work
50
51        when pushmain-action = load-print
52          move "printmgr" to ds-set-name
53          move ds-push-set to ds-control
54          move 1 to ds-control-param
55          initialize printmgr-data-block
56          move printmgr-data-block-version-no to
57            ds-data-block-version-no
58          move printmgr-version-no to ds-version-no
59         call "dsgrun" using ds-control-block,
60                              printmgr-data-block
61          perform print-mgr-work
62          when pushmain-action = exit-program
63          move 1 to end-of-actions-flag
64      end-evaluate.
65
66  file-mgr-work.
67          move ds-quit-set to ds-control
68          call "dsgrun" using ds-control-block,
69            filemgr-data-block
70
71          move ds-continue to ds-control
72          call "dsgrun" using ds-control-block,
73                              pushmain-data-block.
74
75  print-mgr-work.
76          move ds-quit-set to ds-control
77          call "dsgrun" using ds-control-block,
78            printmgr-data-block
```

```
79          move ds-continue to ds-control
80          call "dsgrun" using ds-control-block,
81                              pushmain-data-block.
```

このコードの機能を次に詳しく説明します。

行1**〜**7**：**

```
1  $SET ANS85 MF
2
3  working-storage section.
4      copy "ds-cntrl.mf  ".
5      copy "pushmain.cpb ".
6      copy "filemgr.cpb  ".
7      copy "printmgr.cpb ".
```

プログラムの最初のセクションは、適切な制御ブロックCOPYファイルと、使用される各スクリーンセットについて生成されたCOPYファイルをコピーします。

行9**：**

```
9  01 new-screenset-name    pic x(12).
```

つぎにnew-screenset-nameのPICTURE文字列を定義します。

行11**〜**16**：**

```
11   01 action                  pic 9.
12      78 load-file                    value 1.
13      78 load-print                   value 2.
14      78 exit-program                 value 3.
15   01 end-of-actions-flag     pic 9.
16      88 end-of-actions               value 1.
```

特定のスクリーンセットをロードする操作の値を宣言します。

行20**〜**25**：**

```
20   main-process.
21      perform program-initialize
22      call "dsgrun" using ds-control-block,
```

```
23                      pushmain-data-block
24      perform process-actions until end-of-actions
25      stop run.
```

プログラムの最初の手続きはmain-processです。これは、ルーチンprogram-initializationを実行し、ds-control-blockとpushmainスクリーンセットのためのデータブロックを使ってDsgrunを呼び出します。そして、end-of-actionsを受け取るまでprocess-actionsを実行してから、stop-runが起こります。

行27**〜**35**：**

```
27   program-initialize.
28      initialize ds-control-block
29      initialize pushmain-data-block
30      move ds-new-set to ds-control
31      move pushmain-data-block-version-no to
32        ds-data-block-version-no
33      move pushmain-version-no to ds-version-no
34      move "pushmain" to ds-set-name
35      move zero to end-of-actions-flag.
```

program-initialize手続きは、制御ブロックとデータブロックを初期設定し、スクリーンセットpushmainのために適切な値を移動します。行30で、ds-controlに値Nをいれていることに注意してください。この値は、Dsgrunが呼び出されていない状態でスクリーンセットが起動した場合に使用します。

行37**〜**39**：**

```
37   process-actions.
38      evaluate true
39        when pushmain-action = load-file
```

このセクションは、新しいスクリーンセットをロードすべきかどうかを決めるための評価を実行します。pushmain-actionの値が変化するまで、メインスクリーンセットがロードされており、フォーカスが設定されます。pushmain-actionの値がload-fileになると、ファイルマネージャのためのスクリーンセットがロードされます。

行40**〜**41**：**

```
40   move "filemgr" to ds-set-name
41   move ds-push-set to ds-control
```

pushmain-actionがload-fileの場合、

- スクリーンセット名filemgrがds-set-nameに移動します。
- ds-push-setの値がds-controlに移動します。

注意：ds-push-setは、ds-cntrl.mf内のlevel-78データ項目で、定義値はSです。

行43**〜**49**：**

```
43   initialize filemgr-data-block
44   move filemgr-data-block-version-no to
45     ds-data-block-version-no
46   move filemgr-version-no to ds-version-no
47   call "dsgrun" using ds-control-block,
48                       filemgr-data-block
49   perform file-mgr-work
```

このセクションでは、制御ブロックとデータブロックを初期設定し、バージョン情報をチェックします。そして、ファイルマネージャスクリーンセットを使ってDsgrunを呼び出します。

行51**〜**61**：**

```
51   when pushmain-action = load-print
52   move "printmgr" to ds-set-name
53   move ds-push-set to ds-control
54   move 1 to ds-control-param
55   initialize printmgr-data-block
56   move printmgr-data-block-version-no to
57     ds-data-block-version-no
58   move printmgr-version-no to ds-version-no
59   call "dsgrun" using ds-control-block,
60                       printmgr-data-block
61   perform print-mgr-work
```

このセクションは、pushmain-actionが2になった場合、プリントマネージャについて同じファンクションを実行します。

行62**〜**63**：**

```
62   when pushmain-action = exit-program
63   move 1 to end-of-actions-flag
```

pushmain-actionがexit-programの場合、値1がend-of-actions-flagに移動し、プログラムが終了します。

行66**〜**69**：**

```
66   file-mgr-work.
67   move ds-quit-set to ds-control
68   call "dsgrun" using ds-control-block,
69                       filemgr-data-block
```

ファイル管理ファンクションは、呼出しプログラムではなく、Dialog Systemによって実行されます。ファイル管理が終了すると、アクティブなスクリーンセットが閉じ、新しいスクリーンセットがスタックからポップされます。これは、ds-quit-setをds-controlに移動することによって行われます。Dsgrunが呼び出されると、ds-control-blockが使用され、filemgr-data-blockは無視します。

行71**〜**73**：**

```
71   move ds-continue to ds-control
72   call "dsgrun" using ds-control-block,
73                       pushmain-data-block.
```

ds-continueをds-controlに移動し、スクリーンセットスタックからスクリーンセットpushmainがポップされます。そして、もとの位置から処理が継続されます。

行75**〜**81**：**

```
75   print-mgr-work.
76   move ds-quit-set to ds-control
77   call "dsgrun" using ds-control-block,
78                       printmgr-data-block
79   move ds-continue to ds-control
80   call "dsgrun" using ds-control-block,
81                       pushmain-data-block.
```

プリント管理ファンクションについて、同じ操作が繰り返されます。

17.7.2.1 Custom1サンプルプログラム

custom1.cblは、1つのスクリーンセットの複数のインスタンスの使い方を示します。これは長いプログラムなので、複数インスタンスの使用に関係あるセクションだけを示します。COBOLプログラム、スクリーンセット、関連するファイルは、サンプルディスクに入っています。

このプログラムは、Custom1というメインスクリーンセットと、データグループ内の項目にマップされるCustom2と呼ばれる第2のスクリーンセットの複数のインスタンスを使用します。

行45**〜**46**：**

```
45   78 main-ss-name               value "custom1".
46   78 instance-ss-name           value "custom2".
```

Working-Storage Sectionの中のメインスクリーンセットの名前とインスタンススクリーンセットの名前のためのPICTURE文字列を設定します。

行48**〜**51**：**

```
48   copy "ds-cntrl.mf  ".
49   copy "custom1.cpb  ".
50   copy "custom2.cpb  ".
51   copy "dssysinf.cpy ".
```

いくつかのCOPYファイルをプログラムのworking storage sectionにコピーします。dssysinf.cpyがコピーされることに注意してください。このCOPYファイルは、複数のインスタンスを使用する場合は必ず必要です。

行53**〜**54**：**

```
53   01 instance-table           value all x"00".
54      03 group-record-no       pic 9(2) comp-x occurs 32.
55   01 group-index              pic 9(2) comp-x value 0.
```

データグループをスクリーンセットのフィールドにマップするためのインスタンスを設定します。

行57**〜**61**：**

```
57   78  refresh-text-and-data-proc   value "p255".
58   78  dialog-system                value "dsgrun".
59
60     77  array-ind              pic 9(4) comp.
61     77  display-error-no       pic 9(4).
```

いくつかの値を宣言します。ダイアログ手続きp255は、テキストとデータを再生するのに使用します。

行63**〜**64**：**

```
63   01 main-screenset-id           pic x(8).
64   01 instance-screenset-id       pic x(8).
```

main-screenset-idとinstance-screenset-idのためのPICTURE文字列を定義します。

行66**：**

```
66   01 temp-word                   pic 9(4) comp-x.
```

temp-wordPICTURE文字列は、アクティブなデータブロックを示す値をいれておくために使用します。これは、実際にはファンクションコードです。これは、プログラムの後の方で、temp-wordの値を評価し、適切な操作を実行します。

行103**：**

```
103   move data-block-ptr(1:2) to temp-word(1:2)
```

複数のスクリーンセットをいつ使用するかを知るには、どのスクリーンセットのデータブロックがアクティブかを調べます。このコードは、データブロックの最初の2バイトをtemp-wordに移動します。temp-wordは、ファンクションコードとして使用されます。

行104**〜**143**：**

```
104   evaluate temp-word
105
106   when 1
107     perform set-up-for-ss-change
108     move x"0000" to data-block-ptr(1:2)
109
110   when 2
111      perform poss-invoke-new-instance
112      move x"0000" to data-block-ptr(1:2)
113
114   when 3
115     perform close-instance
116     move x"0000" to data-block-ptr(1:2)
```

```
117
118   when 4
119     perform update-details
120     move x"0000" to data-block-ptr(1:2)
121
122   when 5
123     perform close-all-instances
124     move x"0000" to data-block-ptr(1:2)
...
141   end-evaluate
142     perform clear-flags
143     perform call-dialog-system.
```

temp-wordの値を評価し、適切な操作を実行します。評価が終わると、すべてのフラグをクリアし、Dialog Systemを呼び出します。

行271**〜**287**：**

```
271   call-dialog-system section.
272
273   call dialog-system using ds-control-block,
274                        data-block-ptr
275                        ds-event-block
276   if (ignore-error = 0)
277     if not ds-no-error
278       move ds-error-code to display-error-no
279       move ds-error-details-1 to display-error-details1
280       move ds-error-details-2 to display-error-details2
281       display "ds error no: display-error-no
282          ", " display-error-details1
283          ", " display-error-details2
284          "Screenset is " ds-set-name
285     end-if
286   end-if
```

```
287   .
```

ds-control-block、data-block-ptr（特定のスクリーンセットインスタンスのデータブロックへのポインタ）、ds-event-blockを使って、Dialog Systemを呼び出します。Dialog Systemを呼び出せない場合、エラーを表示します。

行284**〜**297**：**

```
294   set-up-for-ss-change section.
295     move ds-event-screenset-id to ds-set-name
296     move ds-use-instance-set to ds-control
297     move ds-event-screenset-instance-no to
298                     ds-screenset-instance
```

OTHER-SCREENSETイベントの結果、新しいスクリーンセットに移動するために、Dsgrunにパラメータを移動します。ds-use-instance-setの代わりに、最初のスクリーンセットに行って,ds-use-setを移動するかどうかをチェックすることができます。スクリーンセットが1つしかない場合も、最初のスクリーンセットのインスタンス番号は返されます。

行319**〜**329**：**

```
319   poss-invoke-new-instance section.
320
321     set not-found to true
322     move 0 to group-index
323     perform until found or group-index = 10
324
325     add 1 to group-index
326     if group-record-no(group-index) =
327                 customer-index-of-interest
328       set found to true
329     end-if
```

必要なグループ要素がインスタンス化されているかをチェックします。

行331**〜**341**：**

```
331   if found
332     move ds-use-instance-set to ds-control
```

```
333     move instance-screenset-id to ds-set-name
334     move group-index to ds-screenset-instance
335     move group-record-no(ds-screenset-instance)
336                          to group-index
337
338     move "show-yourself" to ds-procedure
339     move customer-group-001-item(group-index) to
340                             redef-block
341     set address of data-block-ptr to address of data-block
```

グループ要素が見つかった場合、それにフォーカスを移動します。

行344**〜**348**：**

```
344     move ds-push-set to ds-control
345     move instance-ss-name to ds-set-name
346     move data-block-version-no to ds-data-block-version-no
347     move version-no to ds-version-no
348     move ds-screen-noclear to ds-control-param
```

グリープ要素が見つからない場合、新しいスクリーンセットの要素を作成します。

行353**〜**361**：**

```
353     move 1 to ds-clear-dialog
354     move "init-proc" to s-procedure
355
356     move customer-index-of-interest to group-index
357     move customer-group-001-item(group-index) to
358                             redef-block
359     set address of data-block-ptr to address of data-block
360
361     perform call-dialog-system
```

最初のウィンドウを示すために、手続き`init-proc`を`ds-procedure`に移動します。制御が戻ってきたら、データブロックポインタのアドレスを設定します。

行365**〜**369**：**

```
365    move ds-screenset-id to instance-screenset-id
366    move group-index to
367    group-record-no(ds-screenset-instance)
368  end-if
369     .
```

screenset-idと、このスクリーンセットインスタンスが取り扱うラインググループを格納します。

行370**〜**376**：**

```
370  close-instance section.
371
372    move ds-quit-set to ds-control
373    move 0 to group-record-no(ds-screenset-instance)
374  .
```

特定のスクリーンセットインスタンスを閉じます。

行377**〜**389**：**

```
377  update-details section.
...
382     move group-record-no(ds-screenset-instance) to
383                              group-index
384     move group-index to customer-index-of-interest
385     move redef-block to customer-group-001-item(group-index)
...
389     move 0 to group-record-no(ds-screenset-instance)
```

スクリーンセットが正しいグループ要素を更新できるように、スクリーンセットインスタンスからメインスクリーンセットに情報をコピーし、`customer-index-of-interest`を設定します。つぎに`group-record-no`に0を移動することによって、インスタンステーブルの値を消去します。

行394**〜**401**：**

```
394  perform derivations
395    move ds-use-set to ds-control
```

```
396    move main-screenset-id to ds-set-name
397    move "refresh-proc" to  ds-procedure
398
399    set address of data-block-ptr to
400      address of customer-data-block
401   .
```

メインスクリーンセットを再びインスタンス化し、リストボックスを再生します。

行408**〜**422**：**

```
404   close-all-instances section.
...
408    move 0 to group-index
409    move 1 to ignore-error
410    perform 10 times
411
412    add 1 to group-index
413    if group-record-no(group-index) not = 0
414      move instance-screenset-id to ds-set-name
415      move group-index to ds-screenset-instance
416      move ds-use-instance-set to ds-control
417      move "terminate-proc" to ds-procedure
418      perform call-dialog-system
419       move 0 to group-record-no(group-index)
420    end-if
421
422    end-perform
```

すべてのアクティブなスクリーンセットインスタンスを閉じます。

行427**〜**428**：**

```
427    set address of data-block-ptr to
428      address of customer-data-block
```

すべてのアクティブなスクリーンセットインスタンスを閉じたあと、メインスクリーンセットを再びインスタンス化

し、データブロックポインタを設定します。

第18章 チュートリアル－サンプルスクリーンセットの作成

この章では、*Dialog Systemの概要*の章で説明したDialog Systemを使ってアプリケーションを作成します。

このチュートリアルでは、あなたがロードレースのディレクタであると仮定します。人々は、各種新聞やスポーツ雑誌に掲載された申込書を送ることによってレースに参加でいます。参加のエントリを処理し、各参加者に特定の走行番号を割り当てるようなシステムのインタフェースを設計する必要があります。また、広告媒体ごとに反応を記録して、各媒体の効果を調べることにします。

インタフェースの最初の画面では、氏名、住所、年齢、性別、競技クラブ、広告コードなどの詳細を収集します。

レースエントリシステムのためのスクリーンセットを作成するには、まず、Dialog Systemを通常の方法で起動します。

18.1 サンプルデータ定義

サンプルスクリーンセット用のデータモデルは、あらかじめ用意されているので作成する必要はありません。この節では、モデルに沿ってデータを定義する方法について説明します。

18.1.1 データブロックの定義

データブロックを定義し、オブジェクトとして設定する表示フィールドのためにマスタフィールドを作成します。マスタフィールドには、ユーザーが入力フィールドに入力するデータが書き込まれます。

データブロックを定義するには、

1. スクリーンセットメニューからデータブロックを選択します。

   データ定義ウィンドウが表示されます。

2. オプションメニューからプロンプトモードを選択します。

   Data Typeダイアログボックスが表示されます。

3. フィールドを選択し、OKをクリックします。

   フィールドの詳細ダイアログボックスが表示されます。

4. NAMEと入力し、英数字を選択してから、整数部に15を入力します。

   Nameフィールドが定義されます。これは、次のステップで定義するD-NAMEというオブジェクトのためのマスタフィールドになります。

5. OKをクリックして、次の入力の空白行を作成します。

上の手順を繰り返して、次の各フィールドを定義します。

フィールド名	タイプとサイズ	用途
MALE	9(1)	ラジオボタンの状態を保存するためのフラグフィールド
ADDRESS	X(100)	D-ADDRESSのマスタフィールド
CLUB	X(30)	D-CLUBのマスタフィールド
AGE	X(3)	年齢リストボックスに使用
CODE	X(3)	D-CODEのマスタフィールド
FLAG-GROUP		1の反復で始まるグループ。フラグを保持するのに使用
EXIT-FLG	9(1)	ユーザーが終了を選択したことを示すフラグ
SAVE-FLG	9(1)	ユーザーが保存を選択したことを示すフラグ
CLR-FLG	9(1)	ユーザーがクリアを選択したことを示すフラグ

これで、サンプルプログラムのデータブロックができあがりました。データ定義ウィンドウは、図 18-1 に示すようになります。

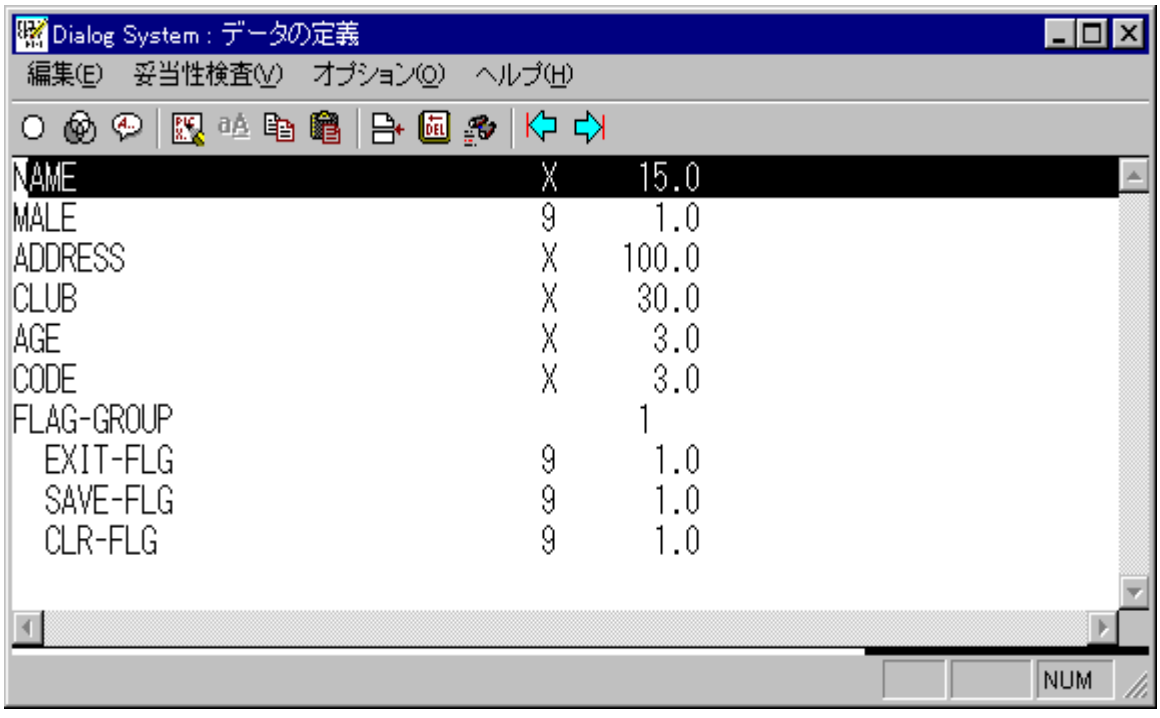


図 18-1 データ定義

18.1.2 サンプルウィンドウのオブジェクトの作成

サンプルスクリーンセットには、各種のコントロールオブジェクトと1つのメッセージが入ったウィンドウがあります。コントロールオブジェクトを配置するためのウィンドウを定義しないと、それらにアクセスできません。したがって、まずウィンドウを定義する必要があります。

オブジェクトの定義を始めるまえに、作成したオブジェクトにラベルを付けられるように、属性の自動設定をオフにします。

1. メインメニューバーのオプションメニューから含めるを選択します。
2. ドロップダウンメニューから属性の自動設定を選択します。

一次ウィンドウを定義するには、

1. オブジェクトツールバーにある一次ウィンドウアイコンを選ぶか、オブジェクトメニューから一次ウィンドウを選択します。
2. ウィンドウボックスの左上の角がデスクトップの左上近くにくるように、マウスでウィンドウを移動します。
3. マウスをクリックして位置を固定します。
4. マウスを右下に移動してウィンドウを拡大します。

   ウィンドウは、Dialog Systemウィンドウより大きくします。
5. マウスをクリックしてサイズを固定します。
6. 属性ダイアログボックスが表示されたら、MAINという名前と、ロードレース競技者一覧というタイトルを付けます。
7. オプションタブを選択し、メニューの選択を解除します。
8. ダイアログボックスの残りの項目は、デフォルト属性値のままにしておきます。

   画面は、図 18-2 のようになります。

図 18-2 Mainウィンドウの定義

18.1.3 サンプルコントロールオブジェクトの作成

次の手順で下のリストにあるコントロールを定義します。

1. [オブジェクト]ツールバーまたは[オブジェクト]メニューからコントロールを選択します。
2. コントロールを一次ウィンドウに配置します。図 18-3 のようになります。

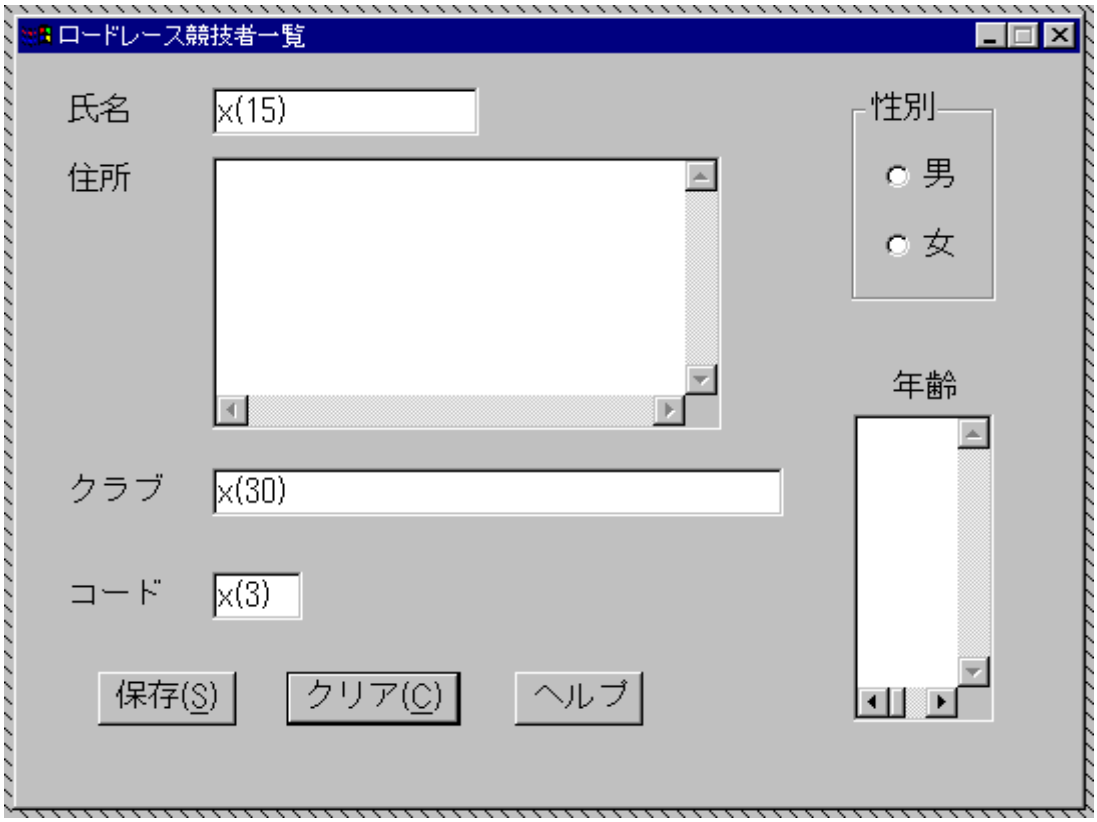


図 18-3 コントロールの追加

3. それぞれの属性ダイアログボックスで指示に従って属性を変更します。

   他の属性はデフォルト値のままにしておきます。

テキスト	競技者の氏名フィールドのラベル。氏名と入力します。
入力フィールド	競技者の氏名をいれます。NameにはD-NAME、PICTUREにはX(15)、マスターにはNAMEと入力します。
テキスト	競技者の住所フィールドのラベル。住所と入力します。
複数行入力フィールド	競技者の住所をいれます。氏名にはD-ADDRESS、長さには120、マスターにはADDRESSと入力します。
テキスト	競技者のクラブフィールドのラベル。クラブを指定します。
入力フィールド	競技者の競技クラブをいれます。名前にはD-CLUB、PICTUREにはX(30)、マスターにはCLUBと入力します。
ラジオボタン	競技者の性別を示します。テキストに男と入力し、初期状態を使用可能に設定します。

ラジオボタン	競技者の性別を示します。テキストに女と入力し、初期状態を使用可能に設定します。
グループボックス	2つのラジオボタンの上に配置。テキストに性別と入力します。
テキスト	年齢層フィールドのラベル。年齢と入力します。
リストボックス	競技者の年齢層を示します。初期テキスト定義済みを選択します。リストテキストをクリックし、10-18、19-35、36-40、41-45、46-50、51-55、56-60、60+のように年齢層を1行に1つずつ入力します。水平スクロールバーチェックボックスの選択を解除します。
テキスト	広告コードフィールドのラベル。コードを指定します。
入力フィールド	広告コードをいれます。名前にはD-CODE、PICTUREにはX(3)、マスターにはCODEを指定します。
プッシュボタン	競技者データベースにデータを入力するためのボタン。テキストには"保存"を指定し、省略時設定を使用可能にし、初期状態を使用可能に設定します。
プッシュボタン	現在の入力を消去するためのボタン。テキストにクリアを指定し、初期状態を使用可能に設定します。
プッシュボタン	ヘルプを表示するためのボタン。テキストにヘルプを指定し、初期状態を使用可能に設定します。

コントロール（および必要に応じて一次ウィンドウ）の位置とサイズを調整し、ウィンドウの見栄えを整えます。

ラジオボタンが正しく動作するためには、コントロールグループにいれる必要があります。同様にプッシュボタンもコントロールグループにする必要があります。

コントロールグループを定義するには、

1. 編集を選択します。
2. コントロールグループの定義を選択します。
3. 表示されたボックスの左上の角の位置を調整します。
4. ボックスをクリックし、必要なコントロールがすべて囲まれるように拡大します。
5. もう一度クリックします。

18.1.4 メッセージボックスの作成

ユーザーがヘルプを押したときにヘルプメッセージを表示されるように、メッセージボックスを作成します。

1. オブジェクトツールバーまたはオブジェクトメニューからメッセージボックスオブジェクトを選択します。

   属性ダイアログボックスがすぐに表示されます。

2. 名前にHELP-MSGを指定します。

3. 見出しにヘルプを指定します。

4. テキストにウィンドウに関する適切なヘルプテキストを入力します。

5. アイコンドロップダウンリストから情報を選択します。

これで、サンプルスクリーンセットのオブジェクト定義が終わりました。

18.1.5 スクリーンセットの保存

オブジェクトの定義が終わった段階でスクリーンセットを保存しておくとよいでしょう。サンプルスクリーンセットを保存するには、

1. このスクリーンセットをいま初めて保存するので、名前を付けて保存を選択します。

   スクリーンセットを保存するファイル名とディレクトリを入力するためのダイアログボックスが表示されます。

2. ファイル名にはentries.gsを指定します。

これ以後のサンプルスクリーンセットの保存では、上書き保存を使用することができます。別のバージョンとして保存したい場合、名前を付けて保存を使って新しい名前を指定します。

18.1.6 テスト

デスクトップレイアウトの見栄えに問題がないか、フィールドが正しく設定されているかなどを確認するために、スクリーンセットは、ダイアログなしでもをテストすることができます。サンプルスクリーンセットをテストするには、

1. ファイルメニューからデバッグを選択します。

   スクリーンセットAnimatorウィンドウが表示されます。

2. 実行メニューから実行を選択します。

   サンプルスクリーンセットは図 18-4 のようになります。

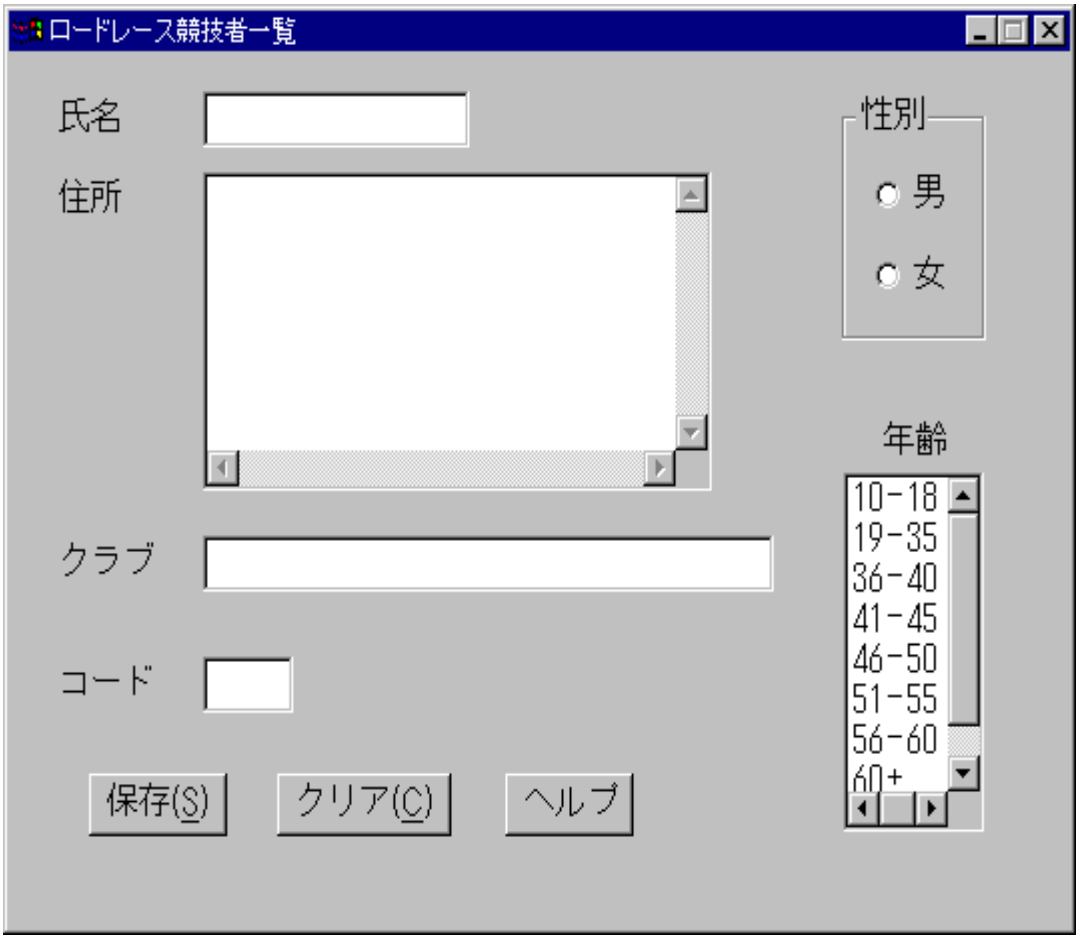


図 18-4 スクリーンセットAnimatorからのスクリーンセットの実行

フィールドにデータを入力してみます。名前フィールドに15字を超える文字を入力しようとすると、警報音が鳴ります。

ラジオボタンはコントロールグループとしてまとめられているために、男または女のいずれかだけが選択できま、リストボックスからは1つの年齢層だけが選択できることに注意してください。

3. スクリーンAnimatorウィンドウに戻るためにEscapeキーを押します。

   "RETC has just been executed"ダイアログボックスが表示されます。

4. Interruptをクリックします。

5. 実行メニューから終了を選択してDialog Systemのメインウィンドウに戻ります。

サンプルスクリーンセットのダイアログを定義すると、スクリーンセットAnimatorがもっと使いやすくなります。

18.1.7 ダイアログの定義

サンプルスクリーンセットを完成するには、ダイアログを定義しなければなりません。サンプルスクリーンセットのオブジェクトとグローバルダイアログを定義する方法を次の各項で説明します。

18.1.7.1 サンプルオブジェクトのダイアログの定義

オブジェクトのダイアログを定義するには、

1. 保存プッシュボタンオブジェクトを選択します。
2. 編集メニューからオブジェクトダイアログを選択します。

   ダイアログ定義ウィンドウが表示されます。このウィンドウには、BUTTON-SELECTEDというデフォルトイベントがすでに表示されています。
3. オプションメニューからプロンプトモードを選択します。

   データを定義するときと同様に、ダイアログの項目を尋ねるプロンプトが表示されます。
4. カーソルキーを使って1つずつ下に降ります。

   入力したい行の種類を選択するダイアログボックスが表示されます。
5. 注釈を選択し、ダイアログの説明を入力します。

   たとえば、"サンプルスクリーンセットでは、入力データベース内のレコードの作成または変更を取り扱う手続きに制御を渡す"と入力します。
6. 保存が押されたときに動作するを指定します。
7. ファンクションを選択します。
8. スクロールリストからSET-FLAGを選択します。

   パラメータダイアログボックスが表示されます。
9. パラメータ1に対してセットされるフラグの名前を指定します。.

   パラメータ2と3は、このファンクションでは必要ないためこれらのスペースは淡色表示になっています。
10. ユーザーが保存を押したときにSaveフラグがセットされるように、パラメータ1に対してSAVE-FLG(1)を入力します。
11. ファンクションを選択します。
12. スクロールリストからRETCを選択します。

    RETCにはパラメータがありません。このファンクションは、呼出しプログラムに制御を返します。このスクリーンセットを使用するCOBOLプログラムは、Saveフラグをチェックし、フラグがセットされている場合、画面の内容を保存します。

サンプルスクリーンセットの他のオブジェクトにも同様にダイアログを付けることができます。

コンテキストメニューを使用すると、作業しやすくなります。ダイアログ行を右クリックすると、コンテキストメニューが表示されます。コンテキストメニューについての詳細は、*ウィンドウオブジェクト*の章を参照してください。

残りのオブジェクトのダイアログについては、次の各項を参照してください。

18.1.7.1.1 **クリアボタン**

ダイアログ：

```
BUTTON-SELECTED
    SET-FLAG CLR-FLG(1)
    RETC
```

結果：

ユーザーがクリアを押した場合、Dialog SystemはClearフラグをセットし、呼出しプログラムに制御を返します。呼出しプログラムは、入力フィールドを消去します（結果として、入力が取り消されます）。

18.1.7.1.2 Help**ボタン**

ダイアログ：

```
BUTTON-SELECTED
    INVOKE-MESSAGE-BOX HELP-MSG $NULL $EVENT-DATA
```

結果：

ユーザーがヘルプを押した場合、Dialog Systemは作成されたメッセージボックスを表示されます。これにより、ユーザーはヘルプテキストを読むことができます。

18.1.7.1.3 **男ラジオボタン**

ダイアログ：

```
BUTTON-SELECTED
    SET-FLAG MALE
```

結果：

ユーザーが男を押した場合、Dialog SystemはMALEフラグを1に設定します。これは、男性の競技者を表します。

18.1.7.1.4 女ラジオボタン

ダイアログ：

```
BUTTON-SELECTED
    CLEAR-FLAG MALE
```

結果：

ユーザーが女を押した場合、Dialog SystemはMALEフラグを0に設定します。これは、女性の競技者を表します。

18.1.7.1.5 年齢層リストボックス

ダイアログ：

```
ITEM-SELECTED
    RETRIEVE-LIST-ITEM $CONTROL AGE $EVENT-DATA
```

結果：

ユーザーがリストから年齢層を選択した場合、Dialog Systemは、選択されたリスト項目をAGEというフィールドにいれます。

18.1.7.2 サンプルグローバルダイアログ定義

つぎにスクリーンセットのグローバルダイアログを定義します。これには、スクリーンセットメニューからグローバルダイアログを選択します。

デフォルトのグローバルダイアログは次のようになっています。

```
ESC
    RETC
CLOSED-WINDOW
    RETC
```

次のように2つの行を追加します。

```
ESC
    SET-FLAG EXIT-FLG(1)
    RETC
CLOSED-WINDOW
    SET-FLAG EXIT-FLG(1)
    RETC
```

このダイアログは、ユーザーがEscapeキーを押したか、システムメニューを使ってウィンドウを閉じた場合、終了フラグをセットし、アプリケーションプログラム（呼び出しプログラム）に制御を返します。

次のダイアログを追加します。

```
REFRESH-DATA
  REFRESH-OBJECT MAIN
```

このダイアログは、呼出しプログラムからDialog Systemに対してメインウィンドウをリセットするようにという指示があった場合に、そのように実行します。アプリケーションと呼出しインタフェースとの対話については、*スクリーンセットの使い方*の章を参照してください。

次のダイアログを追加します。

```
SCREENSET-INITIALIZED
  SET-FLAG MALE
  SET-BUTTON-STATE RB1 1
```

このダイアログは、1番目のラジオボタン（男）をデフォルトとしてセットします。ラジオボタンのグループがを入力する場合、いずれかのラジオボタンを最初にセットしておく必要があります。

ダイアログの使い方についての詳細は、*ダイアログの使い方*の章を参照してください。

18.1.8 スクリーンセットの再テスト

サンプルスクリーンセットを再び保存してから、もう一度実行してみます。各フィールドにデータを入力したり、ラジオボタンやリスト項目を選択してみます。今回は、変更のあとに保存を押すと、スクリーンセットAnimatorウィンドウが再び表示されます。

18.1.9 スクリーンセットの変更

サンプルスクリーンセットの再テストしたあと、さらに変更したい部分（スクリーンレイアウトを改善するなど）が出てくるかもしれません。この場合、この章で述べた手順を繰り返して、スクリーンセットを修正することができます。

18.1.10 まとめ

この章では、次のことを行いました。

- サンプルスクリーンセットの作成
- データの定義
- オブジェクトの定義

- スクリーンセットの保存
- スクリーンセットのテスト
- スクリーンセットへのダイアログの追加
- 再テスト
- 必要なステップの繰り返しによる修正

最後にサンプルスクリーンセットにあるユーザーインタフェースを利用するCOBOLプログラムを作成する必要があります。DialogSystem¥demo¥entriesには、entriesx.gsというサンプルスクリーンセットのデモンストレーションバージョンが入っています。

18.2 次の章の紹介

次の章*チュートリアル−サンプルスクリーンセットの使い方*では、サンプルスクリーンセットのためのCOBOLプログラムを作成する際に、注意すべき点について説明します。ここでは、ユーザー入力を読み取り、それをユーザーの指示に従って格納または消去する、ごく簡単なプログラムのサンプルコーディングを示します。

第19章 チュートリアル－サンプルスクリーンセットの使い方

*チュートリアル－サンプルスクリーンセットの作成*の章では、サンプルアプリケーションのためのユーザーインタフェースが入ったEntriesスクリーンセットの作成方法について説明しました。この章では、Entriesスクリーンセットを使用するアプリケーションプログラムを作成する方法を次のステップに分けてで説明します。

1. スクリーンセットからCOBOLのCOPYファイルを生成します。
2. Dialog Systemランタイムへの呼び出しを含んだCOBOLアプリケーションプログラムを作成します。
3. COBOLプログラムをデバッグとアニメートを行います。
4. アプリケーションをパッケージ化します。

19.1 データブロックCOPYファイルの生成

データブロックCOPYファイルには、実行時に呼出しプログラムからDialog Systemに渡されるデータブロックの定義が入っています。COPYファイルは、呼出しプログラムにいれておかなければなりません。このCOPYファイルには、バージョンチェック情報も入っています。

Dialog Systemを使うと、スクリーンセットからCOPYファイルを生成したり、COPYファイルの生成方法に関するオプションを設定することができます。

サンプルスクリーンセットのCOPYファイルを作成するには、スクリーンセットの構成オプションを設定してから、COPYファイルを生成します。

1. オプションメニューから構成を選択します。
2. ポップアップメニューからスクリーンセットを選択します。
3. スクリーンセット識別子にEntryという名前を入力します。
4. スクリーンセット識別子を先頭に付けるチェックボックスがオンになっていることを確認します。

   これによって、データブロックにあるすべてのデータ名の前に"entry"という文字列が付けられます。サンプルプログラムでは、この接頭辞付きのデータ名を使用します。
5. EnterキーまたはOKを押して、このダイアログボックスの他のデフォルトをそのまま使用します。
6. ファイルメニューから生成を選択します。

7. 表示されたドロップダウンメニューからデータブロックCOPYファイルを選択します。
8. COPYファイルに使用するファイルの名前として entries.cpbを入力します。
9. Enterキーを押します。

   いま設定したCOPYファイルオプションを使って、サンプルスクリーンセット用のCOPYファイルが生成されます。

19.1.1 オプションの選択とCOPYファイルの生成

サンプルCOPYファイルはすでにできあがっています。

19.2 COBOLアプリケーションプログラムの作成

前の章で作成したサンプルスクリーンセットを使用するCOBOLプログラムを次に示します。このプログラムは、entries.cblというデモンストレーションプログラムとしてDialog Systemソフトウェアに含まれています。

要件やプログラミングスタイルによっては、これとは異なるプログラム構造になることがあります。

```
 1 $set ans85
 2  identification division.


 3  program-id. race-entries.


 4  environment division.


 5  input-output section.
 6  file-control.
 7      select entry-file assign "entries.dat"
 8      access is sequential.


 9  data division.


10  file section.
11  fd  entry-file.
12  01  entry-record.
13      03  file-name            pic x(15).
```

```
14      03  file-male            pic 9.
15      03  file-address         pic x(100).
16      03  file-club            pic x(30).
17      03  file-code            pic x(3).

18  working-storage section.
19      copy "ds-cntrl.v1".
20      copy "entries.cpb".

21  78  refresh-text-and-data-proc value 255
22  77  display-error-no         pic 9(4).

23  procedure division.

24  main-process section.
25      perform program-initialize
26      perform program-body until entry-exit-flg-true
27      perform program-terminate.

28  program-initialize section.
29      initialize entry-data-block
30      initialize ds-control-block
31      move entry-data-block-version-no
32                              to ds-data-block-version-no
33      move entry-version-no to ds-version-no
34      open output entry-file
35     perform load-screenset.

36  program-body section.
37  * （フラグで）返されたユーザー操作を処理する。
38  * フラグをクリアし、Dialog Systemを再び呼び出す。
```

```
39      evaluate true
40        when entry-save-flg-true
41          perform save-record
42        when entry-clr-flg-true
43          perform clear-record
44      end-evaluate
45      perform clear-flags
46      perform call-dialog-system.

47  program-terminate section.
48      close entry-file
49      stop run.

50  save-record section.
51  * すべてのテキストフィールドに値が入っている場合、
52  * 現在の詳細をファイルに保存する。

53      if (entry-name    < > spaces) and
54         (entry-address < > spaces) and
55         (entry-club    < > spaces) and
56         (entry-code    < > spaces)
57             move entry-name    to file-name
58             move entry-male    to file-male
59             move entry-address to file-address
60             move entry-club    to file-club
61             move entry-code    to file-code
62             write entry-record
63      end-if.

64  clear-record section.
```

```
65 * データブロックを初期設定して、現在の詳細を消去する。

66      initialize entry-record
67      initialize entry-data-block
68      perform set-up-for-refresh-screen.

69  clear-flags section.
70      initialize entry-flag-group.

71  set-up-for-refresh-screen section.
72 * Dialog Systemが次回呼び出されたときに手続きP225
73 * （グローバルダイアログ内）を強制的に実行する。この
74 * 手続きは、データブロックの値を使って
75 * メインウィンドウを再生するだけである。

76      move "refresh-data" to ds-procedure.

77  load-screenset section.
78 * 使用するスクリーンセットを指定し、Dialog Systemを呼び出す。

79      move ds-new-set to ds-control
80      move "entries"  to ds-set-name
81      perform call-dialog-system.

82  call-dialog-system section.
83      call "dsgrun" using ds-control-block,
84                          entry-data-block.
85      if not ds-no-error
86          move ds-error-code to display-error-no
87          display "ds error no:   " display-error-no
88         perform program-terminate
```

```
89     end-if.
```

行1**～**22**：**

```
1  $set ans85
2  identification division.

3  program-id. race-entries.

4  environment division.

5  input-output section.
6  file-control.
7      select entry-file assign "entries.dat"
8      access is sequential.

9  data division.

10 file section.
11 fd  entry-file.
12 01  entry-record.
13     03  file-name           pic x(15).
14     03  file-male           pic 9.
15     03  file-address        pic x(100).
16     03  file-club           pic x(30).
17     03  file-code           pic x(3).

18 working-storage section.
19     copy "ds-cntrl.v1".
20     copy "entries.cpb".

21 78  refresh-text-and-data-proc value 255
22 77  display-error-no           pic 9(4).
```

これらの行は、ユーザーによる入力を格納するためのレコードを設定します。

行23**〜**27**：**

```
23  procedure division.


24  main-process section.
25      perform program-initialize
26      perform program-body until entry-exit-flg-true
27      perform program-terminate.
```

これらの行はプログラム全体を定義し、ユーザーがEscapeキーを押したり、システムメニューを使ってウィンドウを閉じたときに、プログラムが終了するようにします。この部分は、スクリーンセットのExitフラグをセットします。フラグの設定は、処理のためにプログラムに渡されます。

行28**〜**35**：**

```
28  program-initialize section.
29      initialize entry-data-block
30      initialize ds-control-block
31      move entry-data-block-version-no
32                            to ds-data-block-version-no
33      move entry-version-no to ds-version-no
34      open output entry-file
35      perform load-screenset.
```

データブロックCOPYファイルには、ユーザーデータだけではなく、Dialog Systemがチェックすべきスクリーンセットのバージョン番号がいくつか入っています。呼出しプログラムは、Dialog Systemのランタイムシステムを呼び出すまえに、これらのバージョン番号をコントロールブロック内のデータ項目にコピーしなければなりません。

呼出しプログラムを作成する場合、*copy "ds-cntrl.mf"*という文を使って、プログラムのWorking-Storage SectionにCOPYファイルをコピーしなければなりません。ANSI-85準拠COBOLを使っている場合、COPYファイルds-cntrl.ansを使用する必要があります。

また、コントロールブロックには、スクリーンセットの名前やDialog Systemの動作を制御するその他の情報が入っているかを確認する必要があります。

行36**〜**46**：**

```
36  program-body section.
```

```
37 * （フラグで）返されたユーザー操作を処理する。
38 * フラグをクリアし、Dialog Systemを再び呼び出す。

39      evaluate true
40        when entry-save-flg-true
41          perform save-record
42        when entry-clr-flg-true
43          perform clear-record
44      end-evaluate
45      perform clear-flags
46      perform call-dialog-system.
```

Dialog Systemでは、プログラムがユーザーの操作を指示するのではなく、ユーザーがプログラムの次の動作を決めることができます。Dialog Systemから返されるデータブロックのフラグは、ユーザーの操作に対応する値に設定され、プログラムはこれに合わせて次の動作を決めます。

プログラムは、次のようにさまざまな方法で応答することができます。

- 格納された情報の変更

  `save-record` sectionで行っています。

- データベースから追加情報を検索する。

  このアプリケーションでは、行っていません。

- 結果またはエラーメッセージの表示

  `call-dialog-system` sectionで行っています（プログラムがDialog Systemを呼び出すときにエラーが起こった場合）。

- アプリケーションの使い方に関する支援

  このアプリケーションはシンプルで、すべてのヘルプはDialog Systemのメッセージボックスによって取り扱われます。別の方法として、Helpフラグがセットされたときにヘルプを提供するためのコードをプログラムにいれておくことができます。

- ユーザーが入力した情報の妥当性検査

  `save-record` sectionでは、空のレコードが保存されないように最小限の妥当性検査を行っています。入力段階でプログラムとDialog System自身の両方でもっと複雑な妥当性検査を行うことも可能です。

- ユーザーによる追加入力の要求

  レコードを保存または消去したあとに、Dialog Systemに戻ります。

行47**〜**49**：**

```
47  program-terminate section.
48      close entry-file
49      stop run.
```

これらの行は、エラーが起こったり、ユーザーがEscapeキーを押すか、システムメニューを使ってウィンドウを閉じたときに、プログラムを終了します。

行50**〜**63**：**

```
50  save-record section.
51 * すべてのテキストフィールドに値が入っている場合、
52 * 現在の詳細をファイルに保存する。

53      if (entry-name    < > spaces) and
54         (entry-address < > spaces) and
55         (entry-club    < > spaces) and
56         (entry-code    < > spaces)
57            move entry-name    to file-name
58            move entry-male    to file-male
59            move entry-address to file-address
60            move entry-club    to file-club
61            move entry-code    to file-code
62            write entry-record
63      end-if.
```

これらの行は、ユーザーが保存を押したときにユーザーの入力を保存します。このボタンを押すとSaveフラグがセットされます。このフラグは、プログラムの`program-body` sectionでテストされます。レコードが空の場合、入力は保存されません。

行64**〜**68**：**

```
64  clear-record section.
```

```
65 * データブロックを初期設定して、現在の詳細を消去する。

66      initialize entry-record
67      initialize entry-data-block
68      perform set-up-for-refresh-screen.
```

これらの行は、ユーザーがクリアを押したときに現在の入力を画面とデータブロックから消去します。このボタンを押すとClearフラグがセットされます。このフラグは、program-body sectionでテストされます。

行69〜76：

```
69  clear-flags section.
70      initialize entry-flag-group.
71  set-up-for-refresh-screen section.

72 * Dialog Systemが次回呼び出されたときに手続きP225
73 * （グローバルダイアログ内）を強制的に実行する。この
74 * 手続きは、データブロックの値を使って
75 * メインウィンドウを再生するだけである。

76      move "refresh-data" to ds-procedure.
```

これらの行は、フラグをクリアし、次のユーザーの入力に備えて画面を再生するようにDialog Systemに指示します。

行77〜89：

```
77  load-screenset section.
78 * 使用するスクリーンセットを指定し、Dialog Systemを呼び出す。

79      move ds-new-set to ds-control
80      move "entries"  to ds-set-name
81      perform call-dialog-system.

82  call-dialog-system section.
83      call "dsgrun" using ds-control-block,
84                          entry-data-block.
```

```
85     if not ds-no-error
86         move ds-error-code to display-error-no
87         display "ds error no:   " display-error-no
88         perform program-terminate
89     end-if.
```

これらの行は、正しいスクリーンセットをロードし、Dialog Systemを呼び出します。さらに2番目のセクションでは、呼び出しエラーをチェックし、エラーが生じている場合、プログラムを終了します。

19.3 COBOLプログラムのデバッグとアニメート

COBOLシステムでは、環境を編集、デバッグ、アニメートできます。

アプリケーションをデバッグしているとき、各プログラムのソースコードが別々のウィンドウに表示されます。コードをアニメートする場合、それぞれの文を実行するたびに、ソースの各行が順番に強調表示され、その文の効果がわかります。プログラムを実行する速さをコントロールしたり、実行を中断して、データ項目をチェックしたり、変更することができます。詳しくは、ヘルプのデバッグトピックを参照してください。+

19.4 アプリケーションのパッケージ化

アプリケーションを完成するには、いくつかの細かい作業を行わなければなりません。

Dialog Systemアプリケーションを構築するには、NetExpressからProject機能を使用します。詳しくは、ヘルプのアプリケーションのビルドトピックを参照してください。

ヘルプのコンパイルトピックでは、実稼動アプリケーションを準備するために次に行うべきことを説明しています。

テストが終わったら、最終製品を組み立てることができます。最終製品はディスクにコピーしたり、顧客に送付したり、他のマシンにロードし、アプリケーションとして実行することができます。

最終製品は、アプリケーションの大きさによって次のいずれかまたは両方から構成されます。

- 実行可能モジュール－業界標準の.exeおよびdllファイル形式です。
- ランタイムサポートファイル－ファイル入出力やメモリ管理を行うファイル。

第20章 チュートリアル－ステータスバーの追加とカスタマイズ

このチュートリアルでは、Dialog Systemに付属のサンプルCustomerスクリーンセットにステータスバーコントロールプログラムを追加します。同じステップを使って、任意のスクリーンセットにあるどんなウィンドウにもステータスバーを追加できます。スクリーンセットに他のコントロールプログラムを追加するときも同じ方法が使用できます。ここでは、次の手順について説明します。

- スクリーンセットにステータスバーを追加する。
- スクリーンセットを実行する。
- ステータスバーを操作する。
- ステータスバーに関するイベントを登録する。

このチュートリアルを始めるまえに次のことを行ってください。

- *独自コントロールのプログラミング*の章を読む。

ユーザーコントロールオブジェクトとユーザーコントロールプログラムの解説は、次の場所にもあります。

- ヘルプの*オブジェクトと属性*

  ユーザーコントロールに関するトピックでは、ユーザーコントロールの作成方法と属性の設定方法について説明しています。

- ヘルプの*コントロールプログラム*

  このトピックでは、コントロールプログラムと利用可能な機能について詳しく説明しています。

- COBOLソースコード自身

20.1 準備作業

1. 以下のセッションでCustomerスクリーンセットを使用して、それにいろいろな変更を加えます。念のためCustomer.gsのバックアップを取っておくことをお勧めします。
2. NetExpressを起動します。
3. DialogSystem¥demo¥customerディレクトリ下の Customer.appを開きます。
4. 右ペイン内の Customer.gs を右クリックして、コンテキストメニュー中の[編集]を選択します。

これで Dialog System が起動され、顧客情報ウィンドウの編集が可能となります。

20.2 スクリーンセットへのステータスバーの追加

コントロールプログラムを使用するまえに、スクリーンセットのデータブロックで共通データ領域を定義しなければなりません。この領域は、実行時にスクリーンセットとコントロールプログラムの間で情報を渡すのに使用されます。

20.2.1 データ項目の定義

スクリーンセットで定義するそれぞれのユーザーコントロールは、関連するデータブロック項目（マスタフィールド）を持っていなければなりません。データ項目は、作成されるコントロールのクラスライブラリオブジェクト参照を保持するのに使用されるので、*OBJ-REF*として定義しなければなりません。データ項目は、ユーザーコントロールを追加するまえに定義しなければなりません。スクリーンセットでオブジェクト参照を定義していない場合、Dialog Systemではユーザーコントロールを定義することができません。

ステータスバーのオブジェクト参照を保持するスクリーンセットのデータブロックに次のデータ項目を含めます。

```
MAIN-WINDOW-SBAR-OBJREF         OBJ-REF
```

スクリーンセットのデータブロックに次のようなFUNCTION-DATAのデータ定義を含めます。

```
FUNCTION-DATA                      1
  WINDOW-HANDLE                C5    4.0
  OBJECT-REFERENCE             OBJ-REF
  CALL-FUNCTION                X    30.0
  NUMERIC-VALUE                C5    4.0
  NUMERIC-VALUE2               C5    4.0
  SIZE-WIDTH                   C5    4.0
  SIZE-HEIGHT                  C5    4.0
  POSITION-X                   C5    4.0
  POSITION-Y                   C5    4.0+
  IO-TEXT-BUFFER               X   256.0
  IO-TEXT-BUFFER2              X   256.0
```

FUNCTION-DATA定義は、DialogSystem¥sourceサブディレクトリにあるfuncdata.impファイルからインポートすることもできます。[ファイル]/[インポート]/[スクリーンセット]を選択し、現在のスクリーンセットが上書きされるという旨の確認ダイアログに対して[はい]をクリックします。ここで、[ファイル]ボタンをクリックしfuncdata.impを選択してダブルクリックします。[インポート]ボタンをクリックしてインポートを開始します。

FUNCTION-DATAはすべてのコントロールプログラムに共通なので、コントロールプログラムを使用する各スクリー

ンセットで1度定義するだけで十分です。

20.2.2 ステータスバーの定義

必要なデータを定義したら、ユーザーコントロールを定義します。

1. ステータスバーを追加したいウィンドウを選択します。この例では、MAIN-WINDOWウィンドウです。
2. [オブジェクト]メニューから[ユーザーコントロール]を選択します。

   - または -

   [オブジェクト]ツールバーからユーザーコントロールを選択します。
3. ユーザーコントロールの位置とサイズを調整します。

   ステータスバーは常にウィンドウの一番下に表示されるので、ステータスバーコントロールプログラムは、ユーザーコントロールの位置とサイズを使用しません。したがって、ステータスバーのユーザーコントロールはウィンドウの一番下に沿って配置することをお勧めします。
4. ユーザーコントロールの属性ダイアログボックスで次の設定を行います。

   - ユーザーコントロールに名前を付けます。MAIN-WINDOW-STATUS-BARのように、できるだけコントロールの意味がわかる名前にします。
   - MAIN-WINDOW-SBAR-OBJREFをマスタフィールド名として指定します。
   - SBAR2をユーザーコントロールプログラムの名前として指定します。
   - ドロップダウンリストからStatus Barを選択します。
   - プログラムを現在のプロジェクトに追加をチェックしておきます。
   - ユーザーコントロールプログラムsbar2.cblを生成するために、[生成]をクリックします。

     メッセージボックスが現れ、生成されるコントロールプログラムで使用するためのデータブロックエントリを追加する必要があることを指示します。これはすでに行ってあるので [OK]をクリックします。

     生成されたプログラムが Customerプロジェクトに追加されますので、バックグラウンドでこれを見ることができます。
5. [OK]をクリックし、スクリーンセットを保存します。
6. NetExpressに戻ります。Customer.gsを右クリックしてコンテキストメニューから[COPYファイル生成]を選択します。

7. NetExpressの[プロジェクト]メニューにある[すべてをリビルド]を使って、コントロールプログラムをコンパイルします。実行可能プログラムが生成されます。

20.3 スクリーンセットの実行

スクリーンセットに関するすべての手順が終わったら、それを（忘れずに保存してから）実行し、ユーザーコントロールの動作を確認することができます。

この時点でステータスバーは何も実用的なことを行いません。時計とキー状態は更新されません。

ステータスバーの更新が正しく動作するには、ダイアログを追加する必要があります。

20.4 ステータスバーの操作

それぞれのユーザーコントロールは、そのコントロールプログラムであらかじめ定義された機能を使って操作できます。FUNCTION-NAMEやFUNCTION-DATA内の他のパラメータを設定することによって、コントロールに対して、関連するデータの再生、削除、更新などの操作を実行できます。

次の各項では、ステータスバーに表示される次の情報の状態を操作するためのコードをインプリメントします。

- Insert/Caps/NumLockのキー状態情報
- 時計時刻
- ウィンドウ／ステータスバー部分のサイズ変更
- マウスオーバーヒントのテキスト

20.4.1 時計時刻とキー状態の管理

時計とキーの状態情報を正確に保つには、ステータスバーを定期的に再生する必要があります。これには、Dialog Systemのタイムアウト機能を使用します。

20.4.1.1 タイムアウト機能の使い方

タイムアウトを設定するには、ステータスバーがあるウィンドウ（ここではMAIN-WINDOW）にWINDOW-CREATEDイベントを追加する必要があります。

- 顧客情報ウィンドウを選択し、オブジェクトダイアログへ行きます。
- イベントツールバーボタンをクリックし、WINDOW-CREATED を選択します。

続いて次のダイアログ行を付け加えます。

```
TIMEOUT 25 REFRESH-STATUS-BAR
```

このダイアログは、1/4秒ごとにREFRESH-STATUS-BAR手続き（あとで定義します）を実行します。

注意：スクリーンセットAnimatorを使ってアプリケーションをデバッグしようとすると、スクリーンセットAnimatorがREFRESH-STATUS-BAR手続きをループ実行しているように見えます。これは、実行中のREFRESH-STATUS-BARの最後の行から次のタイムアウトイベントが引き起こされるとき（1/4秒後）までの間にアプリケーションと対話するのが難しいからです。このためスクリーンセットAnimatorを使ってデバッグする場合、タイムアウト値を調整するか、この行を完全に取り除く方がよいでしょう。

つぎにタイムアウトイベントを処理するためのダイアログを追加する必要があります。通常、タイムアウトイベントが起こったときに引き起こされる手続きは、グローバルダイアログに配置します。ダイアログをオブジェクトに配置した場合、そのオブジェクトがフォーカスを失ったとき、Dialog Systemランタイムは、タイムアウト手続きを見つけられなくなることがあります（この場合、ランタイムエラー8が起こります）。

次のダイアログは、タイムアウトイベントが起こったときに（ステータスバーコントロールプログラムのREFRESH-OBJECT機能を使って）ステータスバーを更新します。このダイアログが正しく動作するためには、グローバルダイアログに配置しなければなりません。

```
REFRESH-STATUS-BAR
  MOVE "REFRESH-OBJECT" CALL-FUNCTION(1)
  SET OBJECT-REFERENCE(1) MAIN-WINDOW-SBAR-OBJREF
  CALLOUT "SBAR2" 0 $NULL
```

20.4.2 ウィンドウ／ステータスバー部分のサイズ変更

つぎにウィンドウをサイズ変更したときにステータスバーが正しくサイズ変更されるためのダイアログを追加する必要があります。スタータスバーコントロールプログラムは、ウィンドウをサイズ変更、最大化、元のサイズに戻すときに呼び出す必要があるRESIZE機能を持っています。

この機能は、ウィンドウを最小化するときに呼び出す必要はありません。これは、ウィンドウを最小化したときはステータスバーが見えなくなるからです。

注意：ウィンドウをサイズ変更するには、Size Borderを正しく設定しなければなりません。Customerの例では、MAIN-WINDOWウィンドウはこの属性セットを持っていないので、自分で設定する必要があります。

ウィンドウをサイズ変更したときにステータスバーがサイズ変更されるように、ステータスバーを含んだウィンドウに次のダイアログを追加します。

```
WINDOW-SIZED
    BRANCH-TO-PROCEDURE RESIZE-PROCEDURE

WINDOW-RESTORED
    BRANCH-TO-PROCEDURE RESIZE-PROCEDURE

WINDOW-MAXIMIZED
    BRANCH-TO-PROCEDURE RESIZE-PROCEDURE

  RESIZE-PROCEDURE
    MOVE "RESIZE" CALL-FUNCTION(1)
    SET OBJECT-REFERENCE(1) MAIN-WINDOW-SBAR-OBJREF
    CALLOUT "SBAR2" 0 $NULL
```

20.4.3 マウスオーバーヒントテキストの追加

完全に動作するステータスバーにするには、マウスがウィンドウ内のオブジェクトの上に移動したときに、にステータスバーに表示されるテキストを更新する必要があります。これには、次のようにします。

- ステータスバーが入ったウィンドウのMOUSE-OVERイベントを（SET-PROPERTYダイアログ機能を使って）使用可能にします。
    - WINDOW-CREATEDイベントに次のダイアログ行を追加します。

      ```
      SET-PROPERTY MAIN-WINDOW "MOUSE-OVER" 1
      ```

      このダイアログは、MOUSE-OVERイベントが必要な他のすべてのウィンドウに追加する必要があります。
- MOUSE-OVERイベント（それが生成されたときに）に対応して、ステータスバー上にテキストを設定するためのダイアログを追加します、通常、MOUSE-OVERテキストはステータスバーの左端（セクション番号1）に表示されます。
    - グローバルダイアログに次のダイアログ手続きを追加します。

      ```
      DISPLAY-HINT-TEXT
        MOVE "UPDATE-SECTION-TEXT" CALL-FUNCTION(1)
        MOVE 1 NUMERIC-VALUE(1)
        SET OBJECT-REFERENCE(1) MAIN-WINDOW-SBAR-OBJREF
      ```

```
CALLOUT "SBAR2" 0 $NULL
```

- ステータステキストを更新する必要があるウィンドウの各オブジェクトに次のようなダイアログを追加することによって、オブジェクト上でMOUSE-OVERイベントが起こったときにステータスラインを更新します。

```
MOUSE-OVER
  MOVE "Status text" IO-TEXT-BUFFER(1)
  BRANCH-TO-PROCEDURE DISPLAY-HINT-TEXT
```

*Status text*は、ステータスバーに表示したいテキストに置き換えます。

たとえば、CUSTOMERスクリーンセットのMAIN-WINDOW上にあるLOADプッシュボタンの場合、*Status Text*を*Load Record*に置き換えます。

スクリーンセットを実行すると、CUSTOMERスクリーンセットに次のようなステータスバーが表示されます。

- ステータスバーのセクション1にMOUSE-OVERテキストが表示されます。
- ステータスバーのセクション2、3、4にINSERT、CAPS LOCK、NUM LOCKキーのキー状態が表示されます。
- ステータスバーのセクション5に現在時刻が表示されます。

スクリーンセットを見終わったら、スクリーンセットを保存し Dialog Systemを終了します。

次の節では、ステータスバーに関するイベントを登録する方法について説明します。

20.5 ステータスバーに関するイベントの登録

ステータスバーコントロールプログラムには、ユーザーがステータスバーのいずれかのセクションの上でマウスをクリックしたときに生成されるイベントを取り扱うためのコードがあります。別のイベントを追加するためのステータスバーコントロールプログラムをカスタマイズする方法については、この章のステータスバーのカスタマイズの節を参照してください。

クラスライブラリは、*コールバック*によってアプリケーションにイベントを送ります。たとえば、ステータスバーコントロールプログラムは、SBar2Button1Downという入口点をクラスライブラリといっしょに登録します。これによって、ユーザーがステータスバーの上で左マウスボタンをクリックすると、入口点Sbar2Button1Downにあるコートが実行されます。

コールバック登録は、生成されたプログラムのCreate Entrypoint内のコントロールが完全に作成されたあとに行われます。コードを提供する必要があるすべてのイベントについて、このセクションにコールバックを登録する必要があります。

イベントが登録されると、そのイベントに関連したコールバックコードは、データベース内のデータを更新して、Dialog Systemスクリーンセットに戻すことができます。コントロールプログラムとDialog Systemスクリーンセットの

間でデータを渡す方法については、ヘルプの*コントロールプログラム*トピックを参照してください。

これで、任意のDialog Systemスクリーンセットにステータスバーコントロールプログラムを追加するための手順が完了しました。

同じような手順を使って、任意のスクリーンセットに任意のコントロールプログラムを追加し、ダイアログを個々の要件に合わせてカスタマイズすることができます。

20.6 ステータスバーコントロールプログラムのカスタマイズ

ここでは､ユーザーがクロックセクションをダブルクリックしたときに､必要に応じてイベントを処理するためのコールバックを受け取るための機能を追加する方法について説明します。

定義時に生成したステータスバーコントロールプログラムは「そのまま」使用でき、任意数のウィンドウ上の多数のコントロールを作成および管理できるように作られています。また、独自の機能を追加することもできます。

注意：ステータスバーが入った各スクリーンセットは、それ専用に生成されたステータスバーコントロールプログラムを使用しなければなりません。これは、生成された各ステータスバーコントロールプログラムが、生成対象となるスクリーンセットに固有のコードを含んでいるからです。

ステータスバーにセクションを追加したい場合があります。例として、この節ではステータスバーコントロールプログラムにイベントを追加します。

この例の目的から、sbar2.cblというコントロールプログラムが生成されているものとし、それをカスタマイズします。

どんな場合も、カスタマイズしたプログラムは実行時に$COBDIR上で利用可能になっていなければなりません。

20.6.1 新しいイベントのためのコールバックの登録

ステータスバーにleft-mouse-button-double-clickイベントを追加するには、次の2つのことを行う必要があります。

- **新しいイベントを登録するために、`Register-Callbacks Section`にコードを追加します。**
- **新しいイベントを処理するために、コントロールプログラムにコードを追加します。**

新しいイベントに関するコールバックを登録するためのコードは、`left-mouse-button-down`イベントのためのコールバックを登録するためのコードとよく似ています。新しいイベントに関するコールバック登録コードを追加する前に、新しいイベントのコードがどのように動作するかについて説明します。

```
- MOVE ProgramID & "Button1Down " TO MessageName
```

注意：ProgramID定数は、このプログラムの入口点の命名に使用されます。これによって、ほかのコントロールプログラムが重複名をロードするのを防ぐことができます。

コールバックを受け取る入口点の名前（ここではButton1Down）をMessageNameに格納します。

```
- INVOKE EntryCallback "new" USING MessageName RETURNING aCallback
```

上で指定した入口点名を使って新しいコールバックオブジェクトを作成します。

```
- MOVE p2ce-Button1Down TO i
```

イベント番号（p2ce-Button1Down）を変数Iに格納します。すべてのイベント番号は、NetExpressシステムのsource¥guicl¥p2cevent.cpyというファイルにリストされています。

```
INVOKE aStatusBar "setEvent" USING i aCallback
```

入口点（ProgramID & "Button1Down"）へのコールバックを起こすために、イベント（p2ce-Button1Down）を設定します。

```
INVOKE aCallback "finalize" RETURNING aCallback
```

コールバックオブジェクトはもう必要ないので、削除してかまいません（仕上げ）。

ここで、新しいイベントのコールバックを登録するためのコードを追加することができます。これには、コントロールプログラムのRegister-Callbacks Sectionにコードを追加します。

Left-mouse-button-double-click登録コードを追加するには、

1. **NetExpressをクリックします。**

2. **sbar2.cblを右クリックし、コンテキストメニューから「編集」を選択します。**

3. **Left-mouse-button-downイベントに関する登録コードの直後で、注釈より前に次のコードを挿入します。**

```
MOVE ProgramID & "DblClk1 " TO MessageName

INVOKE EntryCallback "new" USING MessageName RETURNING aCallback

MOVE p2ce-Button1DblClk TO i

INVOKE aStatusBar "setEvent" USING i aCallback

INVOKE aCallback "finalize" RETURNING aCallback
```

20.6.2 Left-mouse-button-double-clickイベントの追加

Left-mouse-button-double-clickイベントは、left-mouse-button-downイベントとよく似ているので、left-mouse-button-down sectionにある既存のコードをコピーし、イベントを処理する新しいセクションを追加します。両方のイベントで異なるコードは、セクションと入口点の名前、ユーザーイベント番号だけです。

新しいイベントを処理するためのコードを追加するには、上のようにsbar2.cblを編集のために開き、

1. Left-mouse-button-down sectionにあるすべてのコードを複製します。新しいセクションの名前をLEFT-MOUSE-BUTTON-DBLCLKにします。
2. LEFT-MOUSE-BUTTON-DBLCLK Sectionの入口点の名前をButton1DownからDblClk1へ変更します。

3. LEFT-MOUSE-BUTTON-DBLCLK Sectionのユーザーイベント番号を34590から34591へ変更します。
4. sbar2.cblを保存します。
5. プロジェクトを再構築します。

注意：アプリケーションで使われるコントロールプログラムが使用および生成するユーザーイベントは、重複が起こらないようにメモしておきます。ユーザーイベントの重複使用は不可能ではありませんが、1つのユーザーイベントを同じウィンドウ上で2つの異なる目的に使用した場合、プログラムが正しく動作しないことがあります。

これらの変更が終わったら、ダイアログを追加し、カスタマイズしたステータスバーコントロールに付け加えられた新しいイベントを使用することができます。

- 例として、NetExpressでcustomer.gsを右クリックし、コンテキストメニューから「編集」を選択します。
- Customer Detailsウィンドウを右クリックし、コンテキストメニューから「ダイアログ」を選択します。
- 新しいイベントを追加し、ステータスバーがダブルクリックされたときにマシンがビープ音を鳴らすようにするには、次のコードを入力します。

```
USER-EVENT
    XIF= $EVENT-DATA 34591 SBAR2-DOUBLE-CLICK
 SBAR2-DOUBLE-CLICK
    BEEP
```

- スクリーンセットを保存し、実行します。
- ステータスバーをダブルクリックしてビープ音が鳴るかをテストします。

次の章では、メニューバーとツールバーを追加およびカスタマイズする方法について説明します。

第21章 チュートリアル－メニューバーとツールバーの追加とカスタマイズ

このチュートリアルでは、Dialog Systemに付属のサンプルCustomerスクリーンセットにメニューバーとツールバーのコントロールプログラムを追加する手順について説明します。メニューバーの追加手順は、前のチュートリアルで説明したステータスバーの追加手順とほとんど同じです。

どのスクリーンセットのどんなウィンドウにメニューバーやツールバーを追加する場合も、手順は同じです。他のコントロールプログラムをスクリーンセットに追加する場合も原理は同じです。ここでは、次の操作を習います。

- メニューバーとツールバーをスクリーンセットに追加する。
- スクリーンセットを実行する。
- メニューバーとツールバーを操作する。
- メニューバーとツールバーに関するイベントを登録する。

このチュートリアルを始めるまえに次の準備作業を行っておいてください。

- *独自コントロールのプログラミング*の章を読む。

ユーザーコントロールオブジェクトとコントロールプログラムについては、次の文書にも説明があります。

- ヘルプの*オブジェクトと属性*

  ユーザーコントロールに関するトピックでは、ユーザーコントロールを作成し、その属性を定義する方法を説明しています。

- ヘルプの*コントロールプログラム*

  このセクションのトピックでは、コントロールプログラムと利用可能な機能について詳しく説明しています。

- COBOLのソースコード

ツールバーコントロールプログラムの使い方に関するデモンストレーションがスクリーンセットtbards.gsに入っています。このデモンストレーションに関する説明は、DialogSystem¥demo¥tbardsサブディレクトリのtbards.txtファイルに入っています。

21.1 設定

1. Customerスクリーンセットを使用します。このスクリーンセットは大幅に変更するので、作業を始めるまえに

customer.gsのバックアップを取っておくことをお勧めします。

2. NetExpressを起動します。

3. DialogSystem¥demo¥customerサブディレクトリにあるcustomer.appを開きます。

4. 左ペインにあるcustomer.gsを右クリックし、コンテキストメニューから編集を選択します。

   Dialog Systemが起動し、Customer Detailsウィンドウが表示されます。

21.2 メニューバーとツールバーの追加のスクリーンセットへの

クラスライブラリのメニューやツールバーを使用するときに注意すべき最も重要な特徴として、各ツールバーのボタンがメニュー項目と関連付けられていることがあります。ツールバーのボタンを選択すると、関連付けられたメニュー項目に関するクラスライブラリによってメニューイベントが生成されます。同様にメニュー項目を使用可能、使用禁止、チェックまたはチェック解除にすると、関連するツールバーのボタンも同じ状態になります。

コントロールプログラムを使用するまえに、スクリーンセットのデータブロックで共通データ領域を定義しておかなければなりません。この領域は、実行時にスクリーンセットとコントロールプログラムとの間で情報を渡すのに使用されます。

21.2.1 データ項目の定義

スクリーンセットで定義する各ユーザーコントロールには、データブロック項目（マスタフィールド）が関連付けられていなければなりません。データ項目は、作成されたコントロールのクラスファイルオブジェクト参照の保持に使用するときのように、OBJ-REFとして定義しなければなりません。データ項目は、ユーザーコントロールを追加するまえに定義しなければなりません。スクリーンセットの中でオブジェクト参照が定義されていない場合、Dialog Systemでユーザーコントロールを定義できません。

メニューバーとツールバーのオブジェクト参照を保持するには、スクリーンセットのデータブロックに次のデータ項目を挿入します。

```
MYTOOLBAR           OBJ-REF
```

前の章*チュートリアル―ステータスバーの追加とカスタマイズ*を完了している場合、スクリーンセットのデータブロックにFUNCTION-DATAに関する次のデータ定義を挿入する必要はありません。

```
FUNCTION-DATA                       1
  WINDOW-HANDLE                 C5    4.0
  OBJECT-REFERENCE              OBJ-REF
  CALL-FUNCTION                 X    30.0
  NUMERIC-VALUE                 C5    4.0
```

```
  NUMERIC-VALUE2              C5    4.0
  SIZE-WIDTH                  C5    4.0
  SIZE-HEIGHT                 C5    4.0
  POSITION-X                  C5    4.0
  POSITION-Y                  C5    4.0
  IO-TEXT-BUFFER              X   256.0
  IO-TEXT-BUFFER2             X   256.0
```

FUNCTION-DATAはすべてのコントロールプログラムについて共通なので、コントロールプログラムを使用する各スクリーンセットで一度定義するだけで十分です。

FUNCTION-DATAの定義をいれる必要がある場合、DialogSystem¥sourceサブディレクトリにあるfuncdata.impファイルからインポートできます。[ファイル]/[インポート]/[スクリーンセット]を選択し、現在ロードされているスクリーンセットを上書きしてもかまわないのでOKをクリックします。[ファイル]ボタンをクリックし、funcdata.impをダブルクリックします。[インポート]ボタン、そして[OK]、[閉じる]をクリックします。

さらに同様の方法で次のデータ定義を

```
TBAR-PARMS                        1
     MENU-INDEX             9     2.0
     CALLBACK-ENTRY-NAME    X    32.0
     ACCEL-FLAGS            9     3.0
     ACCEL-KEY              9     3.0
     MENU-TEXT              X   256.0
     MENU-HINT-TEXT         X   256.0
     RESOURCE-FILE          X   256.0
     RESOURCE-ID            9     5.0
     TOOL-TIP-TEXT          X   256.0
     INSERT-BUTTON-BEFORE   9     2.0
     MSG-BOX-TEXT           X   256.0
```

DialogSystem¥sourceサブディレクトリにあるtbardata.impファイルからインポートする必要があります。

MYTOOLBAR　作成されたツールバーのオブジェクト参照を格納するのに使用されます。このデータ項目は、下のステップで定義するツールユーザーコントロールのためのマスタフィールドになります。

TBAR-PARMS　ツールバー機能を使用するときにメニュー項目とツールバーボタンに関する情報をコントロールプログラムに渡すのに使われる汎用グループです。

クラスライブラリを使用可能にするには、データブロックの中でCONFIG-FLAGとCONFIG-VALUEをC5 4.0項目として定義し、スクリーンセットのグローバルダイアログにあるSCREENSET-INITIALIZEDイベントの始めに次のダイアログを追加します。

```
CLEAR-CALLOUT-PARAMETERS $NULL
CALLOUT-PARAMETER 1 CONFIG-FLAG $NULL
CALLOUT-PARAMETER 2 CONFIG-VALUE $NULL
MOVE 15 CONFIG-FLAG
MOVE 1 CONFIG-VALUE
CALLOUT "dsrtcfg" 3 $PARMLIST
```

このダイアログは、SCREENSET-INITIALIZEDイベント内の他のダイアログより前に配置しなければなりません。

21.2.2 メニューバーとツールバーの定義

必要なデータを定義したら、ユーザーコントロールを定義します。

1. メニューバーとツールバーを追加する顧客情報ウィンドウ（MAIN-WINDOW）を選択します。
2. 現在ウィンドウ上からDialog Systemのメニューをすべて取り除きます。これには、「ウィンドウの属性」ダイアログボックスの「メニュー」オプションのチェックを解除します。
3. オブジェクトメニューからユーザーコントロールを選択します。
   - または -
   オブジェクトツールバーからユーザーコントロールを選択します。
4. ユーザーコントロールをウィンドウ上部のメニューとツールバーを表示したい位置に配置し、サイズを調整します。
5. 「ユーザーコントロールの属性」ダイアログボックスの次の項目を設定します。
   - ユーザーコントロールの名前を選択します。ここでは、MAIN-WINDOW-TOOLBARにします。
   - マスタフィールド名を指定します。これは、前に定義したデータブロック名と一致していなければならないので、MYTOOLBARにします。
   - ユーザーコントロールプログラムの名前としてTBAR2を指定します。
   - 「コントロールタイプ」ドロップダウンリストからツールバーを選択します。
   - 「プログラムを現在のプロジェクトに追加」にチェックマークを付けます。
   - [生成]をクリックして、ユーザーコントロールプログラムtbar2.cblを生成します。

生成されたコントロールプログラムが使用する項目をデータブロックにいれる必要があることを示すメッセージが表示されます。この作業は前のステップで行ったので、[OK]をクリックして操作を続けます。

生成されたプログラムがバックグラウンドでCustomerプロジェクトに追加されるのがわかります。

6. OKをクリックしてスクリーンセットを保存します。

7. ツールバーのGenerate copyfileボタンを選択し、customer.cpbに上書きを了解します。

8. NetExpressのプロジェクトメニューにある[すべてをリビルド]を使って、生成されたプログラムをコンパイルします。プログラムの実行可能バージョンが作られます。

21.3 スクリーンセットの実行

上記の手順がすべて完了したら、ユーザーコントロールの動作を見るためにスクリーンセットを実行することができます。

この時点で、メニューバーとツールバーは特に何も行いません。

メニューバーとツールバーが正しく動作するためのダイアログを追加する必要があります。

TBAR2は、ツールバーコントロールプログラムのカスタマイズバージョンの名前です。これがメニューに使用するコントロールプログラムです。

ツールバーコントロールのカスタマイズバージョンが呼び出されるようにするには、スクリーンセットのグローバルダイアログの終わりに次のダイアログを追加します。

```
CALLOUT-TBAR
    CLEAR-CALLOUT-PARAMETERS $NULL
    CALLOUT-PARAMETER 1 FUNCTION-DATA $NULL
    CALLOUT-PARAMETER 2 TBAR-PARMS $NULL
    CALLOUT "TBAR2" 0 $PARMLIST
```

前の章が終わっていない場合、Object DialogにWINDOW-CREATEDイベントを追加する必要があります。前の章が終わっている場合、WINDOW-CREATEDイベントに次のダイアログを追加します。

```
INVOKE MYTOOLBAR "show" $NULL
```

スクリーンセットを保存し、閉じます。

21.4 メニュー構造の定義

独自のメニューを作成する例として、tbar2defn.cpyに定義されたデータ構造を見てください。それには、左ペインに

あるtbar2defn.cpyを右クリックし、編集を選択します。NetExpressの検索、定義検索メニュー項目を選択し、データ項目"mData"を探します。

21.4.1 既存のメニュー構造

mDataテーブルは、メニューの内容と構造（階層）を定義します。1つのメニューレコードの例であるFileを次に示します。

```
 01 mData.
 78 mDataStart             value NEXT.


***>>> Start USER MENU DATA <<<***
*>-------------------------------------------------------------------
*  メニュー項目のオブジェクト参照
     03   object reference.

*  メニュー項目のタイプ
     03   pic x comp-5        value typeHasChildren.

*  アクセレレータキーの定義（詳しくはkeys.cpyを参照）
     03   pic x(4) comp-5      value noAcceleratorKeyDefined.

*  メニュー項目が選択されたときに起動するCALLBACKの名前
     03   pic x(mCallbackSize) value SPACES.

*  ヌルで終わるメニューテキストとオプションの
*  ヌルで終わるメニューヒントテキスト
*  文字列全体の終わりにさらにもう1つのヌルを付ける
     03   pic x(mStringSize)   value z"&File" & x"00".
*>-------------------------------------------------------------------
     03   object reference.
     03   pic x comp-5        value typeLastSibling.
     03   pic x comp-5        value KS-Virtual-Key.
```

```
           03   pic x comp-5        value OVK-F3.
           03   pic xx.

      *> メニュー/ツールボタンが選択されたときに呼び出す入口点。これは、すべての
      *> 選択項目について同じにする－このプログラムのOn-Item-Selected SECTIONを参照
           03   pic x(mCallbackSize) value '"'&ProgramID & 'OnItemSelected"'.
           03   pic x(mStringSize)   value z"E&xit" &
                                      z"Quit this application" & x"00".
      *>-----------------------------------------------------------------
```

テーブルの各要素は、メニュー構造内の項目を定義し、次の5つの部分からできています。

1. 作成されるメニュー項目のオブジェクト参照
2. メニュー項目のタイプ（メニュー項目タイプの定義については、tbar2defn.cpyファイルを参照）
3. アクセレレータキー（使用する場合）
4. メニュー項目または関連するツールバーボタンが選択されたときに実行するCALLBACK入口点の名前
5. オプションのメニュー項目テキストとメニューヒントテキスト

1番目のメニューレコードは、ウィンドウに最初に追加するメニュー項目を記述しています。最初のメニュー項目は次のいずれかでなければなりません。

- メニュー項目（typeSibling）
- サブメニュー（typeHasChildren）

サブメニューを追加するたびに、次のいずれかの項目が見つかるまで、後続のすべてのメニュー項目レコードがサブメニューに追加されます。

- typeLastSibling
- typeLastSiblingIsSeparator
- typeLastSiblingHasChildren
- typeHasChildren

最初の2つのタイプの場合、後続の項目は親サブメニューまたはウィンドウに追加されます。最後の2つのタイプの場合、後続の項目は新しいサブメニューに追加されます。

これを簡単に説明すると次のようになります。

- 通常のメニュー項目（サブメニューまたはセパレータではない）を追加するには、typeSiblingを使用します
- セパレータを追加するには、typeSeparatorを使用します。
- 新しいサブメニューを追加するには、typeHasChildrenを使用します。
- 通常のメニュー項目がサブメニューの最後の項目である（サブメニューまたはセパレータではない）場合、typeLastSiblingを使用します。
- セパレータがサブメニューの最後の項目である場合、typeLastSiblingIsSeparatorを使用します。
- 新しいサブメニューがサブメニューの最後の項目である場合、typeLastSiblingHasChildrenを使用します。

21.4.2 新しいメニューオプションの追加

例として、編集とオプションという2つの新しいメニューオプションを追加します。 編集には、Select allというメニュー項目があります。オプションには、 カスタマイズというメニュー項目があります。このメニュー項目は、フォントが入ったサブメニューを開きます。

tbar2defn.cpyで定義されたメニューを変更するには、既存のメニューレコードを例として使用し、メニュー構造を変更します。

1. mDataテーブル構造のSTART USER MENU DATA行とEND USER MENU DATA行の間のコードをそのすぐ下にコピーします。そして、コピーの方を編集します。
2. コードの最初のブロックの最後の行にあるメニュー項目テキストを「ファイル」から編集に変更します。
3. アクセレレータキーをF3からF6に変更します。
4. メニュー項目のテキストを「終了」から「すべて選択」、メニューヒントのテキストを「このｱﾌﾟﾘｹｰｼｮﾝを終了」から「すべて選択」に変更します。他の部分はそのままにしておきます。
5. 2番目のメニューオプションに使用されるコード行をそのすぐ下にコピーします。そして、コピーの方を編集します。
6. メニュー項目のテキストを「編集」から「オプション」に変更します。
7. メニュー項目のタイプをtypeLastSiblingHasChildrenに変更します。
8. アクセレレータキーをOVK-F6からNoAcceleratorKeyDefinedに変更します。
9. メニュー項目のテキストをSelect allからCustomize、メニューヒントのテキストをSelect allからCustomizeに変更します。他の部分はそのままにしておきます。
10. 3番目のメニューオプションに使用されるコードの半分の行だけをそのすぐ下にコピーします。

11. メニュー項目のタイプをtypeLastSiblingに変更します。

12. アクセレレータキーをF7からF9に変更します。

13. メニュー項目のテキストを「カスタマイズ」から「フォント」、メニューヒントのテキストを「カスタマイズ」から「フォント」に変更します。他の部分はそのままにしておきます。

14. プロジェクトメニューからすべてをリビルドを選択します。

15. NetExpressのアニメート、実行メニュー項目を使って、アプリケーションを実行します。

この時点では、追加したメニューオプションが表示されますが、選択しても何も起こりません。

21.5 ツールバー構造の定義

独自のツールバーを作成する例として、tbar2defn.cpyに定義されたデータ構造を見てください。それには、tbar2.cblの中でNetExpressの検索、Locate Definitionメニュー項目を選択し、データ項目"mData"を探します。

21.5.1 既存のツールバー構造

bDataテーブル構造は、ツールバーに追加するボタンの順序付きリストを定義しています。ボタンは、リストされた順序でツールバーに追加されます。

1つのボタンレコードの例を次に示します。

```
  01 bData.
  78 bDataStart value NEXT.
*>-----------------------------------------------------------------
*  ボタンオブジェクトの参照
   03 object reference.

*  ボタンと関連付けるメニュー項目インデックス、このボタンを
*  セパレータにする場合はゼロ。このインデックスは、mDataテーブル内の
*  インデックスを参照する。
   03 pic x comp-5 value 2. *> "Exit" menu item

*  ボタンビットマップのリソース識別子
   03 pic x(4) comp-5 value IDB-EXIT.
```

```
*  マウスがボタンの上に来たときに表示するツールチップ
   03 pic x(bStringSize) value z"Quit this application".
*>-----------------------------------------------------------------
```

テーブルの各要素は、ツールバーのボタンを定義し、次の4つの部分からできています。

1. 作成されるツールバーボタンのオブジェクト参照
2. ボタンを関連づるメニュー項目の（mDataテーブルにある）インデックス（セパレータの場合はゼロ）
3. ボタンに使用するリソースファイル内のビットマップのリソース識別子
4. マウスがボタンの上に来たときに表示するツールチップ。

21.5.2 新しいツールバーボタンの追加

例として、すべてを選択とフォントという新しいツールバーボタンを追加します。

tbar2defn.cpyで定義されたツールバーを変更するには、既存のボタンを例として使用し、ボタン構造を変更します。

1. bDataテーブル構造のSTART USER BUTTON DATA行とEND USER BUTTON DATA行の間にあるコードをそのすぐ下にコピーします。そして、コピーの方を編集します。
2. メニュー項目のインデックスを2から4に変更します。必要に応じて、コメントも修正します。
3. ボタンビットマップのリソース識別子をIDB-EXITからIDB-BLUEに変更します。
4. ツールチップを"終了"から"すべてを選択"に変更します。
5. 2番目のツールバーボタンに使用されているコードの行をそのすぐ下にコピーします。
6. メニュー項目のインデックスを4から7に変更します。必要に応じて、コメントも修正します。
7. ボタンビットマップのリソース識別子をIDB-BLUEからIDB-REDに変更します。
8. ツールチップを"すべてを選択"から"フォント"に変更します。
9. プロジェクトメニューから[すべてをリビルド]を選択します。
10. スクリーンセットを実行します。

    作成した2つのボタンがツールバーのメニュー項目の下に表示されます。

    ここでも、ツールバーのボタンを選択しても何も起こりません。

21.6 メニューバーとツールバーのカスタマイズ

前の節では、tbresid.cpyファイルにある既存のビットマップボタンを追加しました。この節では、新しいボタンビットマップを作成します。

1. 新しいリソースファイルをプロジェクトに追加し、ツールバーが使用するビットマップリソースを作成します。

    - NetExpressのコマンドプロンプトで、DialogSystem¥demo¥tbardsサブディレクトリにあるビットマップ（*.bmpファイル）とmfres.hファイルをDialogSystem¥demo¥customerサブディレクトリにコピーします。tbar.rcをコピーし、tbres.rcに名前を変更します。
    - NetExpressの中でプロジェクトウィンドウを右クリックし、ソースプールへファイルを追加を選択します。
    - tbres.rcを選択します。
    - tbres.rcを右クリックし、選択されたファイルをパッケージ化、動的リンクライブラリ(DLL)を選択し、[作成]をクリックして、左ペインに追加します。
    - 左ペインでtbres.rcを右クリックし、コンパイル、編集を選択します。
    - ツリー構造にある"ﾋﾞｯﾄﾏｯﾌﾟ@を右クリックし、コンテキストメニューから新規作成を選択します。
    - Resource Identifierフィールドに"IDB_BUTTON1"と入力します。
    - Resource Filenameフィールドに"selectall.bmp"と入力します。
    - ID Valueフィールドに220と入力し、OKをクリックします。
    - Image Editorメニューからファイル、新規作成、ビットマップを選択し、OKをクリックします。
    - ビットマップの幅と高さとして16を入力し、OKをクリックします。
    - ツールバーボタンが使用可能になったときに表示するビットマップを描きます。
    - 画像をselectall.bmpという名前で保存し、Image Editorを閉じます。
    - 同じ手順を"IDB_BUTTON2"についても実行します。ただし、ID Valueは221、画像の名前はcustfont.bmpにします。これによって、両方のビットマップリソース（およびIDB_識別子）がリソースファイル内で定義されます。
    - ファイル、名前をつけて保存を選択し、このリソーススクリプトをtbres.rcという名前で保存します。
    - tbres.rcを閉じます。

- tbar2defn.cpyを右クリックし、コンテキストメニューから編集を選択します。
- resourceDllNameをtbres.dllに変更します。
- 次の行の下に、

  ```
  copy "tbresid.cpy"
  ```

  作成したビットマップを追加します。これには、次の行を入力します。

  ```
  78 IDB-Button1          value 220.
  78 IDB-Button2          value 221.
  ```

- tbar2defn.cpyを保存し、閉じます。
- プロジェクトメニューからすべての従属関係の更新を選択します。
- プロジェクトを再構築します。tbar2.cblで必要なリソース識別子定数が入ったmfres.cpyが作成されます。 再構築の際にエラーが表示されるかもしれませんが、ここでは無視します。

2. メニューとツールバーの構造を定義するために、COPYファイルを変更します。

   メニュー項目および関連するツールバーボタンを提供するtbar2defn.cpyの中でメニューとツールバーの構造を検討します。このファイルには、メニューとツールバーの外見とイベントに対する応答を定義するデータ構造が入っています。

   COPYファイルの前半はメニューを定義しています。各ツールバーボタンには、メニュー項目が関連付けられています。まず、メニュー項目の定義を見ます。

   - 左ペインでtbar2.cblを右クリックし、編集を選択します。
   - NetExpressの検索、Locate Definitionメニュー項目を選択し、データ項目"mData."を探します。
   - この章の*メニュー構造の定義*で説明した各メニュー項目を定義するmDataグループ構造をよく見ます。構造を詳しく記述しています。
   - データ項目"bData."を探します。
   - この章の*ツールバー構造の定義*で説明した各ツールバー項目を定義するbDataデータ構造をよく見ます。

   ボタン定義レコードには、まえにリソースファイル内で定義したIDB_リソース識別子への参照が入っていることを確認します。具体的には、IDB-Button1とIDB-Button2です。

   tbar2.cblを保存し、閉じます。

3. 次のようにして、メニュー選択に対して応答するためのダイアログをスクリーンセットに追加します。

- 既存のアプリケーションにツールバーを追加する場合（今回の例）、
    - ツールバーの親ウィンドウに次のダイアログを追加します。

      ```
      USER-EVENT
          XIF= $EVENT-DATA 37000 @EXISTING-DIALOG-MENU-PROCEDURE
      ```

      実行手続きを独自のメニューダイアログテーブルに置き換えます。ここでは、TEST-MENU-CHOICEを使用するので、次のように入力します。

      ```
      USER-EVENT
        XIF= $EVENT-DATA 37000 TEST-MENU-CHOICE
      ```

      上の行のすぐ下に次のダイアログを追加します。

      ```
      TEST-MENU-CHOICE
        IF= NUMERIC-VALUE(1) 2 @EXIT
        IF= NUMERIC-VALUE(1) 4 SELECT-ALL
        IF= NUMERIC-VALUE(1) 7 FONTS
      ```

      ダイアログの終わりに次の行を追加します。

      ```
      SELECT-ALL
        BEEP
      FONTS
        BEEP
        BEEP
      ```

      既存のメニュー項目を置き換えるために、クラスライブラリメニューを追加する場合、メニューイベントの取り扱いに使用されていた手続き名（先頭に@記号が付いている）に分岐するだけでかまいません。別の方法として、クラスライブラリメニューイベントが起こったとき、ツールバーのCALLBACKコードからサブプログラムにCOBOL CALLを行うこともできます。

- 新しいアプリケーションを開発している場合、
    - ツールバーの親ウィンドウに関するUSER-EVENTダイアログに次のダイアログを追加します。

      ```
      USER-EVENT
          XIF= $EVENT-DATA 37000 DIALOG-PROCEDURE
      DIALOG-PROCEDURE
      ```

* **ダイアログファンクション処理－メッセージボックスを表示する**

ダイアログファンクションを使ってクラスファイルメニューイベントを取り扱う代わりに、ツールバーのCALLBACKコードから既存のプログラムに対して、そのメニュー機能を実行するためのCOBOL CALLを行うこともできます。

注：例では、データブロックの一部（FUNCTION-DATAとTBAR-PARMS）だけを使って、ツールバーコントロールプログラムを呼び出しています。CALLBACK処理にほかのデータが必要な場合、CALLOUTとルールバープログラムがデータブロック全体を使用できるようにする必要があります。上で述べたグループの使用を変更せずにすみます。

スクリーンセットを保存し、NetExpress IDEに切り換え、プロジェクトをリビルドします。

[アニメート]、[実行]を選択し、スクリーンセットをテストします。[編集]メニューまたはツールバーボタンからすべてを選択を選ぶと、コーディングしたダイアログ手続きが実行されます。オプション、カスタマイズメニューのフォントも試してみます。

第22章 チュートリアルーActiveXコントロールの追加

この章では、スクリーンセットにActiveXコントロールを追加する方法について説明します。この章を読むまえに、*独自コントロールのプログラミング*の章と前のチュートリアルを読んでおいてください。

次の各項では、あらかじめ用意された時計ActiveXコントロールを使って、スクリーンセットを変更する手順について説明します。説明では、まえに作成したカスタマイズステータスバースクリーンセットを使用します。

22.1 スクリーンセットの変更

このチュートリアルの目的は、ステータスバー上の時刻をダブルクリックしたときに用意されたActiveX時計コントロールが表示されるようにすることです。ActiveXコントロールの使い方についてはあとで述べることとし、ここでは、とりあえず次のことを行います。

1. スクリーンセット内に次のオブジェクトを定義します。
    - ActiveX時計を表示するダイアログボックス。このオブジェクトには、CLOCK-DIALOGという名前を付けます。
    - アラーム時刻を設定するためのダイアログボックス。このオブジェクトには、SET-ALARM-DIALOGという名前を付けます。
    - アラーム時刻になったときにメッセージを表示するためのメッセージボックス。このオブジェクトには、ALARM-GONE-OFF-MBOXという名前を付けます。
2. ダイアログボックスとメッセージボックスに関するすべての属性をデフォルトとして設定します。
3. 親を$WINDOWとして定義します。
4. このスクリーンセットに関するData BlockにPIC 9(2)データ項目、ALARM-HOURS、ALARM-MINUTESを追加します。
5. CLOCK-DIALOGダイアログボックスで、
    - テキストを"Set Alarm"に設定したプッシュボタンを作成します。
    - Set Alarmボタンに次のダイアログを追加します。

```
BUTTON-SELECTED
   MOVE " " IO-TEXT-BUFFER2(1)
```

```
SET-FOCUS SET-ALARM-DIALOG
```

- テキストを"Close"に設定したプッシュボタンを作成します。
- Closeボタンに次のダイアログを追加します。このボタンをクリックすると、CLOCK-DIALOGが隠れます。

```
BUTTON-SELECTED
    UNSHOW-WINDOW CLOCK-DIALOG $NULL
```

CLOCK-DIALOGダイアログボックスは削除しないでください。削除すると、アラームが動作しなくなります。

6. SET-ALARM-DIALOGで、

- アラーム時刻の「時」をいれておく入力フィールドを作成します。

  これをALARM-HOURSマスタフィールドとリンクします。

- アラーム時刻の「分」をいれておく入力フィールドを作成します。

  これをALARM-MINUTESマスタフィールドとリンクします。

- アラームが引き起こされたときに表示するメッセージをいれておく入力フィールドを作成します。

  この入力フィールドのマスタフィールドはIO-TEXT-BUFFER2にします。

- テキストを"OK"に設定したプッシュボタンを作成します。OKボタンは、次のダイアログを持っています。

```
BUTTON-SELECTED
    BRANCH-TO-PROCEDURE SET-ALARM
```

- テキストを"キャンセル"に設定したプッシュボタンを作成します。
- ユーザーがキャンセルをクリックしたときにSET-ALARM-DIALOGダイアログボックスが削除されるように、次のダイアログを追加します。

```
BUTTON-SELECTED
    DELETE-WINDOW SET-ALARM-DIALOG $NULL
```

- SET-ALARM-DIALOGダイアログボックス自身に次のダイアログを追加します。

```
CR
    BRANCH-TO-PROCEDURE SET-ALARM
```

```
    ESC
        DELETE-WINDOW SET-ALARM-DIALOG $NULL
      SET-ALARM
```

*** あとでここにダイアログを追加する。**

これらの手順が済んだら、CLOCK-DIALOGダイアログボックスを表示するために、前のチュートリアルで追加したleft-mouse-button-double-clickイベントを使用するダイアログを付け加えることができます。

Left-mouse-button-double-clickイベントコールバックでは、コールバックが引き起こされたときにユーザーイベント34591を送るためのコードを追加します。また、追加したコードは、double-clickイベントが起こったステータスバー上のセクション番号をNUMERIC-VALUE(1)データ項目に配置します。

この情報によって、double-clickユーザーイベントを処理するために、MAIN-WINDOWダイアログテーブルに次のダイアログを追加できます。

```
USER-EVENT
    XIF= $EVENT-DATA 34591 DOUBLE-CLICK-EVENT
  DOUBLE-CLICK-EVENT
    IF= NUMERIC-VALUE(1) 5 DOUBLE-CLICK-ON-CLOCK
  DOUBLE-CLICK-ON-CLOCK
    SET-FOCUS CLOCK-DIALOG
```

このダイアログは、MAIN-WINDOWウィンドウがイベント番号34591を受け取り、NUMERIC-VALUE(1)が5（ステータスバーの第5セクションには時計がある）に設定された場合、CLOCK-DIALOGダイアログボックスが表示されるようにします。

22.2 Dialog Systemの時計ActiveXコントロールの使い方

この機能を利用したデモンストレーションがocxds.gsスクリーンセットに用意されています。ドキュメントはocxds.txtファイルに入っています。

チュートリアルに従うと、スクリーンセット内にCLOCK-DIALOGが作成されます。これで、このダイアログボックスにあらかじめ用意されたDialog Systemの時計ActiveXコントロールを追加することができます。

スクリーンセットに次のデータ定義を追加します。

```
CLOCK-DIALOG-ACTIVEX-OBJREF       OBJ-REF
ACTIVEX-PARAMETERS                     1
  PARM-NAME                   X     30.0
  P1                          C5     4.0
```

```
P2                              C5    4.0
P3                              OBJ-REF
```

最初のデータ項目は、ActiveXのオブジェクト参照を格納するのに使用されます。指定されたマスタフィールドグループは、ActiveXコントロールプログラムを呼び出すときに2番目のパラメータとして使用されます。

つぎにCLOCK-DIALOGダイアログボックスにActiveXコントロールを配置します。

1. [オブジェクト]、ユーザーコントロールを選択するか、オブジェクトツールバーにあるユーザーコントロールボタンをクリックします。
2. CLOCK-DIALOGダイアログボックス上のユーザーコントロールの位置とサイズを調整します。

   ActiveXコントロールの属性ダイアログボックスが開きます。
3. ActiveXコントロールの名前を指定し、マスタフィールドをCLOCK-DIALOG-ACTIVEX-OBJREFに設定し、プログラム名をocxctrlとします。
4. Selectプッシュボタンをクリックします。利用可能なActiveXコントロールがリストされます。Dialog System Clock Objectを選びます。
5. ActiveXコントロールに関する属性を設定する必要があります。それぞれのActiveXコントロールは、カスタマイズ可能な属性を持っています。この属性によって、コントロールの実行時の動作が決まります。

   [属性]ボタンをクリックします。ダイアログボックスが表示され、リストか、ActiveXページダイアログ（利用可能な場合のみ）を表示するDialogプッシュボタンを使って、デフォルト属性を構成することができます。

   Dialog Systemの時計ActiveXの場合、利用可能な属性は背景ビットマップと時計のアナログ／ディジタル切り替えです。
6. ActiveX属性ダイアログボックスのOKをクリックし、ユーザーコントロールの属性ダイアログボックスのOKをクリックします。コントロールが作成され、属性がスクリーンセットに格納されます。

注意：Dialog Systemのエクスポート機能は、ActiveX属性をエクスポートしません。したがって、ActiveXユーザーコントロールをエクスポートおよびインポートする場合、属性を再設定する必要があります。これは、ActiveXユーザーコントロールオブジェクトを複製したり（コントロールキーを押しながらオブジェクトをドラッグすることによって）、ActiveXユーザーコントロールをコピーしたり、貼り付ける場合も同じです。

ActiveXユーザーコントロールができあがったら、ステータスバーの場合と同様にREGISTER-CALLBACKSファンクションを使ってコールバックを登録する必要があります。これには、CLOCK-DIALOGダイアログボックスに次のダイアログを追加します。

```
WINDOW-CREATED
    MOVE "REGISTER-CALLBACKS" CALL-FUNCTION(1)
    MOVE-OBJECT-HANDLE CLOCK-DIALOG WINDOW-HANDLE(1)
    SET OBJECT-REFERENCE(1) CLOCK-DIALOG-ActiveX-OBJREF
    CALLOUT "ActiveXCTRL" 0 FUNCTION-DATA
```

また、アラーム時刻になったことを示すユーザーイベントを取り扱うためのダイアログを追加する必要があります。同じCLOCK-DIALOGに関するダイアログテーブルに、次のダイアログを追加します。

```
USER-EVENT
    IF= $EVENT-DATA 34570 ALARMGONEOFF


  ALARMGONEOFF
    INVOKE-MESSAGE-BOX ALARM-GONE-OFF-MBOX IO-TEXT-BUFFER2(1) $REGISTER
```

最後にSET-ALARM-DIALOGダイアログボックスに関するダイアログにまえに追加したSET-ALARM手続きを完成する必要があります。この手続きは、アラームをセットするためにActiveXコントロールのSetAlarmメソッドを呼び出します。

```
  SET-ALARM
    SET-MOUSE-SHAPE SET-ALARM-DIALOG "SYS-WAIT"
    MOVE ALARM-HOURS P1(1)
    MOVE ALARM-MINUTES P2(1)


    NULL-TERMINATE IO-TEXT-BUFFER2(1)
    CLEAR-CALLOUT-PARAMETERS $NULL
    CALLOUT-PARAMETER 1 IO-TEXT-BUFFER2(1) $null
    CALLOUT-PARAMETER 8 P3(1) $NULL
    INVOKE "chararry" "withValue" $PARMLIST
```

このダイアログは、Messageテキスト（IO-TEXT-BUFFER2(1)内）をCharacterArrayインスタンスへのオブジェクト参照として渡します。P3(1)変数のためのファンクションNULL-TERMINATEとINVOKEの使い方に注意してください。これらのファンクションについての詳細は、ヘルプを参照してください。次のダイアログを追加します。

```
MOVE "INVOKE-ActiveX-METHOD" CALL-FUNCTION(1)
    SET OBJECT-REFERENCE(1) CLOCK-DIALOG-ActiveX-OBJREF
    MOVE "SetAlarm" PARM-NAME(1)
```

```
CLEAR-CALLOUT-PARAMETERS $NULL
CALLOUT-PARAMETER 1 FUNCTION-DATA $NULL
CALLOUT-PARAMETER 2 ActiveX-PARAMETERS $NULL
CALLOUT "ocxctrl" 0 $PARMLIST

CLEAR-CALLOUT-PARAMETERS $NULL
CALLOUT-PARAMETER 8 P3(1) $NULL
INVOKE P3(1) "finalize" $PARMLIST

DELETE-WINDOW SET-ALARM-DIALOG $NULL
```

これで、Dialog Systemの時計ActiveXコントロールがスクリーンセットに追加されました。スクリーンセットを保存し、変更結果を見るために実行してみます。

第23章 チュートリアルービットマップを使ったマウスポインタの変更

*ウィンドウオブジェクト*と*コントロールオブジェクト*の章では、ビットマップグラフィックスの選択方法について述べました。この章では、次のことについて説明します。

- マウスポインタを変更する。
- ビットマップに関する機能を提供する。

23.1 マウスポインタの変更

マウスポインタの形状を変更するには、SET-MOUSE-SHAPEファンクションを使用します。たとえば、操作が完了するまで待つようにユーザーに伝えるには、マウスポインタを砂時計または時計に変更します。

SET-MOUSE-SHAPEファンクションについての詳細は、ヘルプの*「ダイアログの定義: ファンクション」*を参照してください。

23.1.1 Moudemoサンプルスクリーンセット

Moudemoサンプルプログラム（サンプルディスクに入っている）は、マウスポインタをその下にあるオブジェクトに従って変更します。

- 静的ラジオボタンを選択した場合、3種類のオブジェクト（入力フィールド、リスト、ビットマップ）にマウスポインタを移動しても何も起こりません。
- 動的ポインタラジオボタンを選択した場合、各オブジェクトにマウスポインタを移動すると、その形状が変化します。

カーソル形状がテキストバーに変化するようなテキスト入力用の入力フィールドや、各行を選択および強調表示できるリストボックスもあります。

Moudemoスクリーンセットをテストするには、

1. Dialog Systemを起動します。
2. スクリーンセットmoudemo.gsをロードします。
3. スクリーンセットAnimatorを呼び出すためにファイルメニューから実行を選択します。
4. スクリーンセットAnimatorの値はデフォルトのままにします。

5. Enterキーを押します。

図 23-1 に示すような画面が表示されます。

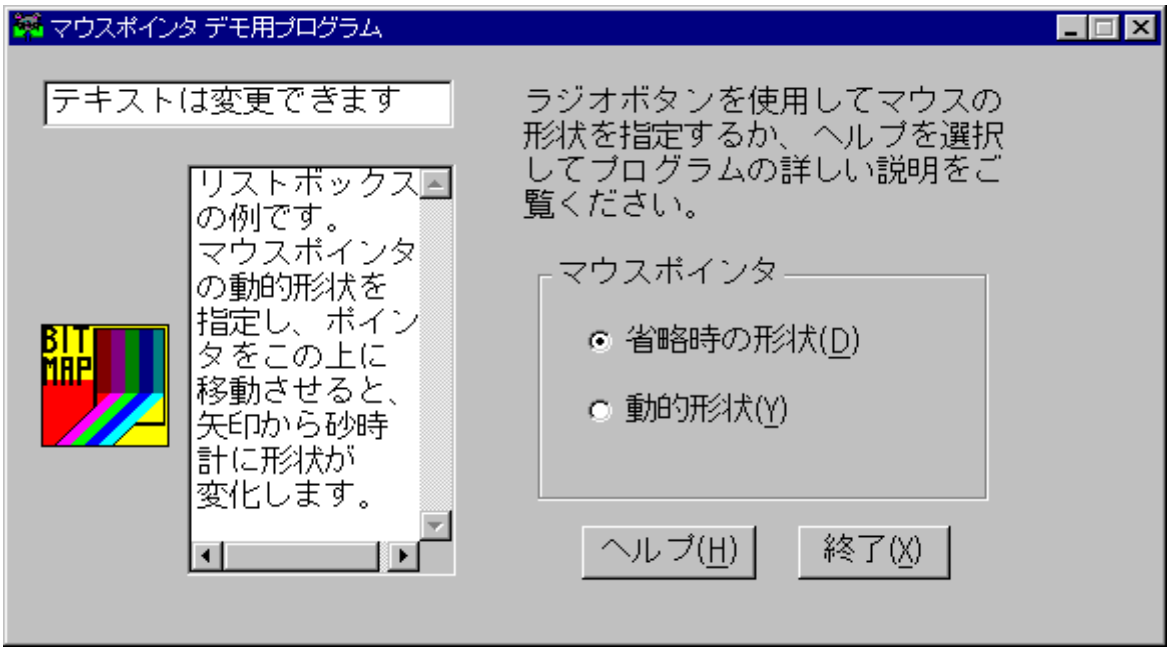


図 23-1 Moudemo画面

ユーザーがEscapeキーを押したり、メインウィンドウを閉じたとき、グローバルダイアログを使って呼出しプログラムに制御を返すことができます。

```
ESC
    RETC
CLOSED-WINDOW
    RETC
```

スクリーンセットを初期設定すると、静的ラジオボタンがオンに設定され、入力フィールドにテキスト文字列が配置されます。マウスポインタのデフォルト形状がSYS-Arrowに設定されます。

```
SCREENSET-INITIALIZED
    SET-BUTTON-STATE DEFAULT-RB 1
    MOVE "You can change this test text!" TEST-TEXT-DATA
    REFRESH-OBJECT TESTTEXT-EFLD
    SET-MOUSE-SHAPE MOUSEDEMO-WIN "SYS-Arrow"
```

終了ボタンを選択した場合、終了メッセージボックスが表示され、$REGISTERの値がチェックされます。$REGISTERが1の場合、OKが選択されており、ユーザーが終了しようとしていること示します。

```
BUTTON-SELECTED
    INVOKE-MESSAGE-BOX EXIT-MSG $NULL $REGISTER
    IF= $REGISTER 1 EXITAPP
  EXITAPP
     RETC
```

必要に応じて各オブジェクトにさらにダイアログボックスを付けます。静的ラジオボタン（スクリーンセットの開始時のデフォルト）の場合、メインウィンドウ全体にSYS-Arrow形状が使用されます。

```
BUTTON-SELECTED
    SET-MOUSE-SHAPE MOUSEDEMO-WIN "SYS-Arrow"
```

動的ラジオボタンが選択されている場合、マウスポインタの形状は、その下にあるオブジェクトによって変化します。SYS-Moveは操作環境に用意された形状です。このスクリーンセットではさらにpencil-ptrが作成されます。

```
BUTTON-SELECTED
    SET-MOUSE-SHAPE TESTTEXT-EFLD "SYS-Move"
    SET-MOUSE-SHAPE TESTMAP-BMP "pencil-ptr"
```

ヘルプボタンを選択すると、ヘルプダイアログが表示されます。

```
BUTTON-SELECTED
    SET-FOCUS HELP-DIAG
```

ヘルプウィンドウのOKをクリックすると、画面からウィンドウが消え、制御がメインウィンドウに戻ります。

```
BUTTON-SELECTED
    DELETE-WINDOW HELP-DIAG MOUSEDEMO-WIN
```

リストボックスや入力フィールドには、ダイアログは付けられません。

23.1.1.1 サイドファイルの変更

サンプルをインストールすると、moudemoサンプルのための資源サイドファイル（moudemo.icn）がロードされます。したがって、マウスポインタを追加するためにds.icnファイルを編集する必要はありません。

この節では標準のds.icnファイルにマウスポインタとビットマップを追加する方法について説明します。

このスクリーンセットで使用するためのマウスポインタpencil-ptrが作成され、DLLファイルに配置されています。適切なセクション見出しの下にds.icnファイル項目を作成しなければなりません。

```
pencil-ptr       : dssamw32.dll 002
```

マウスポインタはDLLファイルに配置しなければなりません。具体的な手順については、ヘルプの*Bitmaps, Icons and Mouse Pointers*トピックを参照してください。

また、ビットマップの名前と場所を宣言する新しいビットマップの項目を追加しなければなりません。これは、スクリーンセットと同じディレクトリでなくてもかまいません。Moudemoサンプルスクリーンセットでは、このビットマップをcolorful.bmpと呼んでいます。これに対応するds.icnの項目（適切なセクション見出しの下にある）は、次のようになります。

```
colorful : dssamw32.dll 003
```

23.2 ビットマップのプログラミング

Dialog Systemでは、ビットマップは、ユーザーが選択できるコントロールです。この点で、ビットマップはラジオボタンと同様に考えることができます。ビットマップを選択したあとに何が起こるかは、ダイアログのコーディングによって決まります。

ビットマップはグラフィック画像によって表現されるので、ユーザーにとって、より意味があるものになります。いくつかのビットマップが付けられたウィンドウを図 23-2 に示します。

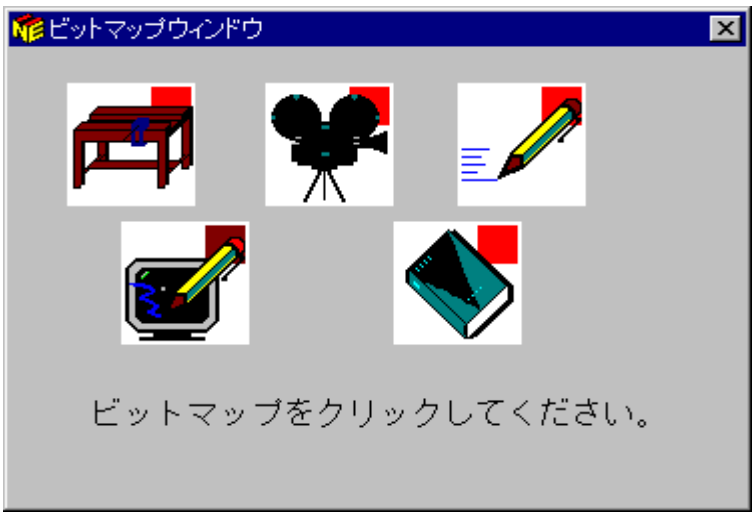


図 23-2 ビットマップが取り付けられたウィンドウ

ビットマップについての理解を助けるために、次の例には、ユーザーが実行できるファンクションを示す5つのビットマップがあります。ユーザーがビットマップを選択したら、呼出しプログラムに戻り、ビットマップを示すファンクションを実行したいとします。

これらのビットマップコントロールには、WBBMP、ANIMBMP、HELPBMP、SCREENBMP、EDITBMPという名前が付いています。各ビットマップは、内部識別子であるハンドルを持っています。オブジェクトに割り当てられるハンドルは、データブロックで定義されていなければなりません。

この例のハンドル（実際には、すべてのハンドル）は、次のように定義されています。

```
WB-HANDLE               C       4.00
ANIM-HANDLE             C       4.00
```

```
HELP-HANDLE             C       4.00
SCREEN-HANDLE           C       4.00
EDIT-HANDLE             C       4.00
```

最初にビットマップのハンドルを保存しなければなりません。最もよい方法は、グローバルダイアログでSCREENSET-INITIALIZEDファンクションを使用することです。

```
SCREENSET-INITIALIZED
    MOVE-OBJECT-HANDLE ANIMBMP ANIM-HANDLE
    MOVE-OBJECT-HANDLE EDITBMP EDIT-HANDLE
    MOVE-OBJECT-HANDLE WBBMP WB-HANDLE
    MOVE-OBJECT-HANDLE HELPBMP HELP-HANDLE
    MOVE-OBJECT-HANDLE SCREENBMP SCREEN-HANDLE
```

MOVE-OBJECT-HANDLEファンクションは、ビットマップ（ANIMBMP）のハンドルを数値フィールド（ANIM-HANDLE）に移動します。

BITMAP-EVENT は、ビットマップに関連付けられる唯一のイベントです。ユーザーがビットマップ上でマウスをクリックすると、イベントが引き起こされ、ビットマップのハンドルが$EVENT-DATAに格納されます。

次のダイアログは、ビットマップが表示されるウィンドウに付属されます。

```
BITMAP-EVENT
    IF= $EVENT-DATA ANIM-HANDLE SET-ANIM
    IF= $EVENT-DATA EDIT-HANDLE SET-EDIT
    IF= $EVENT-DATA WB-HANDLE SET-WB
    IF= $EVENT-DATA HELP-HANDLE SET-HELP
    IF= $EVENT-DATA SCREEN-HANDLE SET-SCREEN
  SET-ANIM
    MOVE 1 FUNCTION
    RETC
  SET-EDIT
    MOVE 2 FUNCTION
    RETC
  SET-WB
    MOVE 3 FUNCTION
    RETC
```

```
  SET-HELP
    MOVE 4 FUNCTION
    RETC
  SET-SCREEN
    MOVE 5 FUNCTION
    RETC
```

したがって、ユーザーがビットマップをクリックすると、次のことが起こります。

- ビットマップイベントが引き起こされ、ビットマップのハンドルが$EVENT-DATAに格納されます。
- $EVENT-DATAが、まえに格納されたビットマップのハンドルと比較されます。
- FUNCTIONに値を設定する手続きが実行されます。
- 制御がプログラムに返されます。

プログラムは、FUNCTIONの値に基づいて適切なファンクションを実行します。

Dialog Systemのインタフェースへのビットマップの追加については、ヘルプの*ビットマップ*、*アイコン*、*マウスポインタ*トピックを参照してください。

付録A: フォントと色の指定

次のような目的のために、複数のフォントと色を使用することができます。:

- 組織の標準に準拠するために、インタフェースの表示を変更する。例えばすべての アプリケーションに特定の色設定を使用している場合など。
- ユーザーの好みに合わせてインタフェースの表示を変更する。
- 特定のコンポーネントに対するユーザーの注意を引くための視覚的な目安を使って、 インタフェースのコンポーネントを強調する。たとえば、次のようなことができます:
    - デフォルト値を別のフォントで表示する。
    - 妥当性検査エラーが起こったフィールドの背景色を変更する。
    - 太字を使って [削除] ボタンを強調する。

異なるフォントを使用すると、インタフェースを大幅に向上することができます。 しかし、使用するフォントの種類が多すぎると、画面が煩雑になります。

A.1 フォントの設定

フォントは視覚的特性が共通している文字の集合です。別のフォントを使用することによって、 インタフェースの表示を診やすくすることができます。たとえばモノスペースフォントを使って インタフェースの各部をきれいに揃えることができます。

また、異なるフォントを使って、インタフェースの特定の部分にユーザーの注意を引くことが できます。例えば、あるテキスト文字を周囲の文字とは異なるフォントにすることによって 強調表示することができます。次のテキストは、この機能を示しています:

テキストを読みやすくするために **フォントの設定** を変えます。

インタフェースの特定のコンポーネントを強調表示するもうひとつの手段には、テキスト のサイズ（ポイントサイズ）があります。前の例では強調したい文字は大きなポイントサイズ （たとえば10ポイントの代わりに12ポイント）にすることができます。これもユーザーの注意を その部分にひきつけるのに役立ちます。

フォントは定義時にだけ設定できます。Dialog System には、フォント定義を動的に変更 するための機能はありません。しかし、Panels V2 を使ってフォントを動的に変更すること ができます。 Dialog System プログラムから Panels V2 を呼び出す方法については、*Panles V2 の使い方* の章を参照して下さい。

フォントの設定方法については、ヘルプの *Dialog Systemの概要* のトピック を参照して下さい。

A.2 色の設定

オブジェクトの前景色と背景色は、定義時に [編集] メニューの [色の設定] メニュー項目を使って変更するか、SET-COLOR ファンクションを使って 動的に変更することができます。

たとえば、次のダイアログ部分は、妥当性検査エラーが起こったフィールドの色を 'RED'（背景色）上に 'WHITE'（前景色）で表示するものです。Dialog System が妥当性エラーを検出した場合、エラーが起こったフィールドの識別子 が $EVENT-DATA に書きこまれることを思い出してください。

```
VAL-ERROR
    SET-COLOR $EVENT-DATA 'WHITE' 'RED'
    SET-FOCUS $EVENT-DATA
```

すべての環境で利用可能な複数の汎用色があります。また、各環境には、選択可能な 追加色のリストがあります。クロスプラットフォームファイルを作成する場合、使用する 色がすべての環境でサポートされているかを確認して下さい。

オブジェクトの色をデフォルト色に戻すには、前景色と背景色の値を $NULL にします。 :

```
SET-COLOR DELETE-PB $NULL $NULL
```

SET-COLOR ファンクションの詳細と、利用可能な色の一覧については、ヘルプの *「ダイアログの定義: ファンクション」* のトピックを参照して下さい。

索引