Micro Focus Net Express®

Dialog System ユーザガイド

第 9 版
2003 年 5 月

このユーザガイドでは、グラフィカルユーザインターフェイスを紹介するとともに、Dialog System の活用方法について説明します。 ここでは、主なメニュー項目の解説、簡単なスクリーンセットの作成方法と使用方法、重要な機能の紹介、および、アプリケーション開発における一般的な手順のデモンストレーションを説明します。 このユーザガイドでは、より高度な機能についても説明しています。 例、サンプルプログラム、Dialog System の機能を最大限に活用するためのヒントが豊富に用意されています。

グラフィカルユーザインターフェイス

第 1 章 : グラフィカルユーザインターフェイス

ここでは、グラフィカルユーザインターフェイス (GUI) を紹介し、その使用方法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、インターフェイスを使用するプログラムの作成開始前に、インターフェイスの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。

- アプリケーションプログラムやメインプログラムロジックに依存しないユーザインターフェイスを作成できる。
- アプリケーションプログラムに影響を与えずに、ユーザインターフェイスを定義および変更できる。
- 同一のプログラムに対して複数の異なるユーザインターフェイスを作成できる。
- COBOL プログラムのサポートなしで、インターフェイスをプロトタイプ化およびテストできます。

  そのため、Dialog System 内のみで、プログラムとは無関係に、データの入力、出力の受信、ウィンドウ間の移動などを実行できます。
- 実行時のユーザとの対話は、すべて、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処理されます。

  そのため、アプリケーションプログラムのサイズを縮小して簡単に維持できます。
- Dialog System では、プログラムとユーザインターフェイスの間の通信にコーディング済みデータ構造を使用して、プログラムの作成を簡便化します。
- Dialog System は、異なる環境の間で移植できます。

用語

Dialog System のマニュアルで使用されている用語については、ヘルプの『用語集』を参照してください。 用語集では、1 つの項目に対して 2 つの用語がある場合に、その両方について説明し、Dialog System で使用する用語を明記しています。

アプリケーションに GUI を使用する目的

GUI の目的は、次のような特徴をもったインターフェイスを提供することです。

- 使用方法を簡単に習得できる。

- 1 度習得するのみでよい (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 を使用すると、アプリケーションによって提供される機能を理解することよりも、作業自体に集中できます。 GUI は、直感的であり、実験的な操作も行えます。 また、操作が誤ってもかまいません。 つまり、ユーザが独自の方法とペースで使用方法を習得できます。

GUI をしばらく使用していなかったユーザが再び使用する場合でも、使用方法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。

文字ベースのアプリケーションインターフェイスでは、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、ユーザは目的の機能まで各階層を移動しなければなりません。 ショートカットキーを使用する方法もありますが、記憶するのが困難です。

GUI アプリケーションでは、実行する作業に直接的にアプローチできます。 ユーザは、オブジェクトを選択してから操作を選択します。 そのため、いくつかのメニュー階層を飛ばすことができます。 マウスのポイント操作とクリック操作によって、インターフェイス内を簡単に移動して必要な作業を実行できます。

GUI では、関連する領域とのリンクにより、コンテキストヘルプを使用できます。 そのため、階層ヘルプツリー内の移動など、文字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、作業を実行するために必要な背景的な知識を簡単に得ることができます。

Dialog System の役割

Dialog System は、ユーザインターフェイスのライフサイクルの 5 つの段階でユーザを支援します。

- ユーザインターフェイスの定義

  Dialog System を使用して、アプリケーションで使用するデータと画面上の表示状態を定義します。 この定義には、ユーザがデータを入力する方法やインターフェイ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 のテスト機能を使用し、プログラムを実行した場合を想定して、データ入力やインターフェイス内の移動を実行します。 また、出力のシミュレートや戻り値の監視を実行します。

- 呼び出し側プログラムの統合

  インターフェイスを詳しくテストした後で、Dialog System を使用して、データ構造やランタイムインターフェイスの詳細を記述したコピーファイルを作成します。

- アプリケーションの実行

  Dialog System をランタイムモードで使用し、呼び出し側プログラムに制御が戻るまでの間にユーザとの対話を処理してスクリーンセットを使用します。

- アプリケーションの維持

  Dialog System を使用して、呼び出し側プログラムに影響を与えることなく、ユーザインターフェイスを変更およびカスタマイズします。 また、ユーザインターフェイスを変更しないで、呼び出し側プログラムを変更できます。 両方を変更するのは、ユーザインターフェイスとプログラムの間で受け渡すデータ項目を変更する必要があるときのみです。

GUI システムの使用方法

Dialog System は、マウスで使用するように設計されています。 マウスを使用して、画面上のオブジェクトの選択、移動、およびサイズ変更を行うことができます。 これらの操作は、キーボードを使用し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場合に使用します。

次の各項では、GUI 環境に慣れるために、マウス、ウィンドウ、コントロールオブジェクトの使用方法について説明します。

マウス操作

マウスによって、画面上の位置を示すポインタを操作します。 画面上には、ウィンドウ内の位置を示す選択カーソルも表示されます。

マウスポインタは、ウィンドウの境界を超えて動かすことができ、形状が変化します。たとえば、テキストフィールドでは、マウスポインタが I 字形で表示され、テキストが追加される位置を正確に示します。 サイズ変更可能なウィンドウの隅では両矢印で表示されます。処理が完了するまでの間は時計 (環境によって文字盤のある時計や砂時計) で表示されます。 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できる操作を判断する手がかりとなります。

マウス操作を行う前に、マウスポインタを画面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する場合は、ポインタをそのオプションに移動します。

マウスを使用して、画面上の各種ボタンを操作したり、オブジェクトや項目を選択したり、画面上の情報をスクロールしたりできます。

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放します。

  項目またはオブジェクトを選択したり、プッシュボタンやオプションボタンを押したり、チェックボックスやチェックマーク選択を切り替えたりします。

- クリックアンドドラッグする (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または項目上に移動し、マウスボタンを放します。

  複数の項目を選択するためにその領域をボックスで囲んだり、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たりします。

- ダブルクリックする - マウスボタンを押して放す操作を 2 回繰り返します。

  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。

- 放す - 押していたマウスボタンを放します。

  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たりします。

移動や選択など、Dialog System に関係するその他のボタン動作を構成できます。 特定の動作を関連付けて、Dialog System で使用することができます。Dialog System には、このための各種の標準パターンがあります。 使用するパターンは、オペレーティングシステム環境、使い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。

ウィンドウとメニュー

デスクトップとは、画面上の作業領域です。 ウィンドウは、画面の全体または一部を表すオブジェクトです。 画面の端を越えたデスクトップ上にもウィンドウを作成できます。 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、一次ウィンドウはデスクトップの子にあたります。 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できます。

すべての一次ウィンドウに、二次ウィンドウと呼ばれる子ウィンドウを作成できます。 これらの二次ウィンドウにも他の二次ウィンドウを作成できます。

一次ウィンドウと二次ウィンドウについては、『ウィンドウオブジェクト』の章を参照してください。

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。

サブメニューは、必要な選択項目をクリックすると、メニューバーからプルダウンできます。 サブメニューがプルダウンされている間に別の項目をクリックした場合は、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、項目が表示されないことがあります (たとえば、配置先の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)。 淡色表示されている項目は使用不能です。 淡色表示された項目をクリックした場合は、警告音が鳴りますが、その他には何も起こりません。

メニュー項目では、次の操作を実行できます。

- 別のプルダウンメニューを表示します。 別のプルダウンメニューがある場合は、項目に続いて矢印が表示されます。
- 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。 項目に続いて省略記号 (...) が表示されます。
- チェックマークされた選択項目を表示します。 オンとオフの 2 つの状態があります。 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、チェックマークが項目の先頭に表示されます。 このような項目は、トグルとも呼ばれます。
- 操作を直接実行します。 この機能を識別するための情報は表示されません。

メニュー項目を選択すると、上記のリストのどれか 1 つの操作が実行され、対応する情報 (省略記号やチェックマークなど) が表示されます。

ウィンドウの操作

ここでは、マウスとアイコン (通常のウィンドウの一部) を使用して、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。

各構成要素を示したウィンドウを図 1-1 に示します。

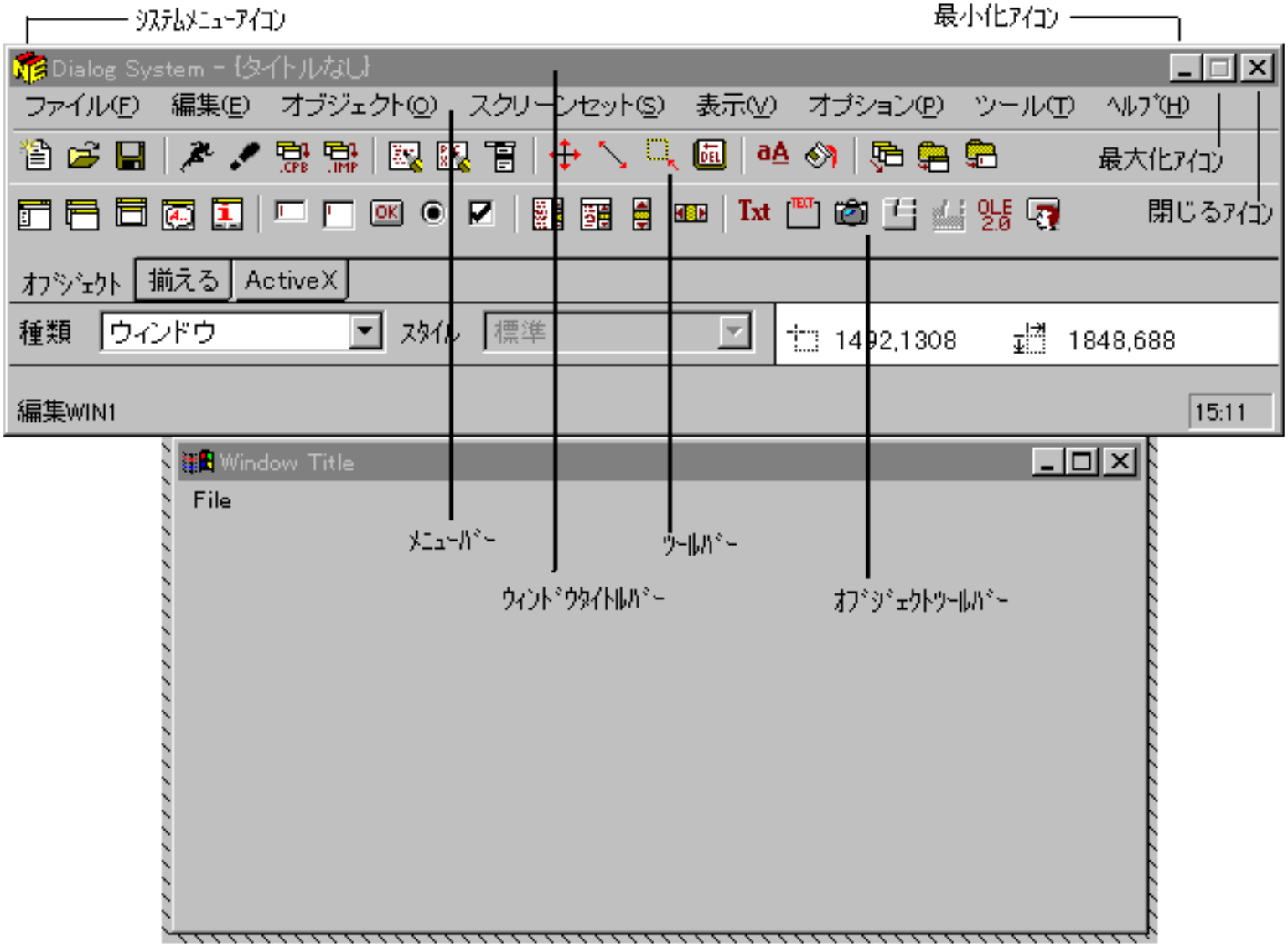

図 1-1： ウィンドウの構成要素

一次ウィンドウを使用していない場合は、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。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 (『マウス操作』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したりできます。 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クリックアンドドラッグし、マウスボタンを放します。 このときにクリックするボタンは、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隅の境界に移動してから、クリックアンドドラッグします。 マウスポインタは、ウィンドウのサイズを変更するのに適当な位置にある場合に形状が変化します。 ウィンドウの上下左右または隅をドラッグすると、ウィンドウの形が変化します。 ウィンドウのサイズと形が適切な状態になったときに、マウスボタンを放します。

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、移動するウィンドウの一部をクリックします (タイトルバーのアイコンをクリックすると、アイコンがアクティブになります)。

ウィンドウが他のウィンドウの下に隠れて表示されない場合は、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) のタイトルバーには、色が付いています。

ウィンドウ操作の詳細については、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。

ダイアログボックス

ダイアログボックスは、ユーザ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の場合は、サイズを変更したり、メニューバーを設定したりできません。 ダイアログボックスには、Dialog System 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) を追加できます。 ダイアログボックスは、他のウィンドウや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をそのウィンドウの子と呼び、そのウィンドウをダイアログボックスの親と呼びます)。

ダイアログボックスは、ユーザ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、ユーザがプッシュボタンをクリックした場合に Dialog System がユーザの操作を受け付けるように設定します (たとえば、リストからの選択を受け付けるための [OK] ボタン)。 ユーザがプッシュボタンをクリックした場合に Dialog System がユーザの操作を無視するように設定することもできます (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場合によく使用されます。 ダイアログボックスによって、ユーザ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したりできます。

メッセージボックス

メッセージボックスは、ユーザにメッセージを伝えるために画面上に表示されるウィンドウです。 メッセージボックスには、メッセージを表すためのテキストとグラフィック、メッセージに対して応答するためのプッシュボタンが表示されます。 メッセージボックスを閉じるには、ユーザがプッシュボタンをクリックする必要があります。

メッセージボックスを使用して、誤ったデータが入力された場合に警告を表示したり、破壊的な操作 (オブジェクトの削除など) を実行してよいかどうかをユーザに確認したりできます。

コントロール

コントロールは、ウィンドウまたはダイアログボックスに追加して、ユーザがアプリケーションと対話できるようにするための Dialog System のオブジェクトです。 コントロールオブジェクトを配置できるのは、ウィンドウ内またはダイアログボックス内のみです。 Dialog System には、エント

リフィールド、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の詳細については、『コントロールオブジェクト』の章とヘルプを参照してください。

選択オブジェクト

GUI システムでは、ユーザがオブジェクトとそのオブジェクトに適用する操作を選択することによって、アプリケーションを操作できます。

Dialog System には、次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適用できます。

- 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てクリックします。
- マウスで複数のオブジェクトを選択するには、マウスボタンの 1 つが選択操作用に設定されている必要があります (『マウス操作』を参照)。

  シフトキーを押したまま、各オブジェクトをクリックすると、複数のオブジェクトを選択できるので、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用してオブジェクトを選択するには、[編集] メニューの [領域の選択] または [すべて選択] を使用します。 [領域を選択] 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 では、次の選択方法も使用できます。これら方法は、特定のオブジェクトのみに適用できます。

- メニュー項目を選択するには、その項目にマウスポインタを移動してクリックするか、または、矢印キーを使用して項目を選択して 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て、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は、デフォルト操作が実行されます。 他の項目をクリックすると、最初の選択が取り消され、新しい項目が選択されます。
- 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て、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は、デフォルト操作が実行されます。 リストボックスは、1 つの項目のみ、複数の項目、または、複数の隣接する項目を選択するように設定できます。
- オプションボタンの項目を選択するには、ボタンをクリックします。 選択されたボタン内には、黒い円が表示されます。 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は、前のボタンの選択が解除されます。 (コントロールグループ内で選択できるオプションボタンは、1 つのみです。)
- 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。 選択状

態が切り替わります。選択されている場合にクリックすると、選択が解除されます。選択されていない場合にクリックすると、選択されます。 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、実際に押されたように表示されます。ボタンを押すと操作が始まります。

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 (縦スクロール用) または下端 (横スクロール用) にスクロールバーが表示されます。 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表示されます。

- 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 画面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体をクリックします。 クリックした方向にスクロールされます。
- 一度に複数の画面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、マウスを使用して、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ときに、マウスボタンを放します。

詳細情報

次の章『Dialog System の概要』では、Dialog System を使用するための基本概念について説明します。

グラフィカルユーザインターフェイス
データ定義とスクリーンセットの作成

第 2 章 : Dialog System の概要

前の章では、グラフィカルユーザインターフェイス (GUI) の利点について説明しました。 ここでは、Dialog System を使用して COBOL アプリケーションの GUI を作成するための基本概念について説明します。 説明する内容は、次のとおりです。

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

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インターフェイスの独立

  プログラムが実行時にユーザと対話する必要がある場合に Dialog System が呼び出されて、処理されます。 このような独立性によって、プログラムの構造を改善できます。

- プログラムからのユーザインターフェイスの独立

  同一のプログラムに対して複数の異なるユーザインターフェイスを作成できます。

- グラフィカルオブジェクトの定義の簡便性

  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ック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設定やスタック解除などのオブジェクトの動作を管理します。 定義時と実行時の画面処理も Dialog System によって実行されます。

- 入力、出力、およびウィンドウ間の移動の完全な処理

  ダイアログと呼ばれる簡単な命令セットを使用して、ユーザ入出力の処理やユーザに返す画面内容などを Dialog System に指定できます。

- データブロックの自動生成

Dialog System は、呼び出し側プログラムにデータブロックの定義を渡すことができます。このデータブロック定義は、実行時に Dialog System とのインターフェイスを確立するために必要です。

- 妥当性検査機能

  Dialog System では、ほとんどの入力データを妥当性検査し、必要に応じてユーザにエラーメッセージを表示できます。 呼び出し側プログラムも、必要に応じてこれらのエラーメッセージを使用できます。

- プロトタイプ化とテスト

  Dialog System は、呼び出し側プログラムと同様にインターフェイスを実行できます。そのため、信頼性が高い結果が得られます。 インターフェイスが完成するまで待つ必要はありません。インターフェイスの開発中に一部分をテストできます。

Dialog System の機能の概要

Dialog System を使用すると、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、このインターフェイスとユーザの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たりできます。 Dialog System では、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 - ユーザインターフェイスに表示する項目を作成し、適合させます。
- Dsgrun プログラム - インターフェイスおよびアプリケーションプログラムの両方と通信し、インターフェイスの構成要素の実行時動作を制御します。

ここでは、定義ソフトウェアの概要、および、定義ソフトウェアを使用してユーザインターフェイ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。

- インターフェイスとその要素の設計
- ダイアログを使用してインターフェイスの要素をアクティブ化する方法
- データブロックを使用して各構成部分を呼び出し側プログラムとリンクする方法

インターフェイスの設計

GUI を使用する 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。 このモデルでは、ユーザインターフェイスで取得するデータとアプリケーションからユーザに提示するデータを定義します。

このデータモデルは、呼び出し側プログラムに必ず渡す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ための条件を指定する必要があります。

ユーザインターフェイスを作成する場合には、次の点に注意してください。

ユーザが制御できる：	ユーザがインタフェースを制御できるようにします。 ユーザがアプリケーションの一連の動作を決定します。
使いやすい：	ユーザが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るようにする必要があります。 情報は、順序立てて整理し、読みやすく表示します。
一貫性がある：	ユーザインターフェイスが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性をもつようにします。 たとえば、ウィンドウのメニュー項目 [終了] を必ず [ファイル] メニューの最後のオプションに指定します。また、削除操作を行う場合は、必ず確認が必要です。
便利である：	ユーザには、システム内での位置、実行した操作の内容、次に実行する操作などを明確でわかりやすくフィードバックします。 ユーザにとっては、インターフェイスでデータエントリ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便利です。
誤操作に対応する：	ユーザの誤操作を簡単に元に戻せるようにします。 エラーメッセージで、エラーの内容、原因、解決方法などについて説明します。
効率的である：	システムの応答時間は短くします。
すべての設定が正しい：	たとえば、選択項目のリストがある場合は、すべての項目が有効かつ選択可能であるようにします。
論理的である：	たとえば、メニュー項目は論理的に配置します。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。

ユーザがどのようにしてデータ (ユーザインターフェイス) と対話するかについても考慮する必要があります。

簡単なメニュー選択やデータエントリフィールドのみを配置したユーザインターフェイスから、メニューバー、オプションボタンなどのコントロールを設定した本格的なウィンドウまで、さまざまな形態のユーザインターフェイスを作成できます。

インターフェイスの内容は、次のような多くの要素によって決定されます。

- ユーザの種類と習熟度
- 利用可能なハードウェアとソフトウェアのリソース
- 実行すべきタスクの複雑さ

ただし、どの場合についても次の点が共通しています。

- ユーザが操作の流れを決定する。
- インターフェイスを制御するのは、プログラムではなく、ユーザである。
- ユーザとコンピュータとの対話は、簡単で自然にする。

ユーザインターフェイスに対して定義する基本的な視覚的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、プッシュボタンなど) は、画面内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。

- ウィンドウオブジェクト
- コントロールオブジェクト

ウィンドウオブジェクト

ウィンドウオブジェクトは、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は、定義時でも実行時でも同様に表示されます。

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。 ウィンドウオブジェクトは、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用してサイズを変更できます。 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。

コントロールオブジェクト

その他のオブジェクトは、すべてコントロールオブジェクトです。 コントロールオブジェクトは、ウィンドウに表示されます。エントリフィールド、プッシュボタン、オプションボタン、チェックボックス、リストボックスなどがあります。

Dialog System のデフォルトの範囲に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に、Dialog System では、専用のコントロールプログラムを生成できます。

次の 2 種類のコントロールを定義できます。

- ユーザコントロール

  ユーザコントロールは、Dialog System では描画できないオブジェクトのコンテナまたはアウトラインを定義します。

  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 GUI 内でのコントロールの役割を簡単に識別できます。

- 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実行時に作成および操作するには、プログラムを作成する必要があります。 このコントロールを選択すると、実行時と同じように表示されます。

これらのコントロールについては、『コントロールオブジェクト』と『カスタム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。

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には、背景色などの属性が設定されています。 属性を変更して、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の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] を選択します。 この場合は、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は表示されません。 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、または、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は、オブジェクトの「属性」ダイアログボックスがすぐに表示され、属性を設定できます。 ここでダイアログボックスの [キャンセル] をクリックすると、オブジェクトが削除されます。 [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用し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、またはユーザコントロールを定義する場合は、 [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。

オブジェクトの操作

オブジェクトを操作するには、そのオブジェクト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として選択されています。

オブジェクトへの命名

すべてのオブジェクトに共通の属性は、オブジェクト名です。 オブジェクト名が必要なオブジェクトと、オブジェクト名をオプションで指定するオブジェクトがあります。 オブジェクトには、一意な名前を指定する必要があります。

COBOL プログラム内の変数に関する命名規則を作成することをお勧めします。 一貫性のある命名規則を使用すると、どの開発者でもアプリケーションを容易に理解できます。

また、ユーザインターフェイス内のオブジェクトについても同様の命名規則を作成することをお勧めします。

簡単な命名規則の例として、次のようにいくつかの項目を連結する方法もあります。

windowname-objectinfo-objecttype

各項目の意味は、次のとおりです。

windowname	オブジェクトを配置するウィンドウの名前。オブジェクト自体がウィンドウである場合は、そのウィンドウの名前。
objectinfo	オブジェクトに関する識別情報。 ウィンドウ上のオブジェクトを一意に識別する情報を指定します (プッシュボタンに表示されるテキストなど)。
objecttype	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」、プッシュボタンは「pb」、ダイアログボックスは「db」など。

この規則に従うと、新入社員に関する情報を集めるウィンドウには、NEW-EMPLOYEE-WIN という名前を付けることができます。 また、このウィンドウのプッシュボタンには、NEW-EMPLOYEE-OK-PB という名前を付けることができます。 給与情報の入力に使用するエントリフィールドには、NEW-EMPLOYEE-SALARY-EF という名前を付けることができます。

ダイアログの使用方法

ユーザインターフェイスは、グラフィック表示のみではありません。 完成仕様には、ユーザとコンピュータの対話方法、および、ユーザインターフェイスソフトウェアとアプリケーションソフトウェアとの対話方法も記述します。

表示状態の定義が完了した後で、ユーザとマシンとの実行時の対話を定義する必要があります。 この対話はダイアログと呼ばれます。 ダイアログは、イベントと関数で構成されます。 イベントが発生すると、そのイベントに関連付けられた関数が実行されます。 イベントは、キーボードのキーが押されたり、メニュー項目やオブジェクトが選択されたりすることによって発生します。

たとえば、BUTTON-SELECTED という Dialog System のイベントは、ユーザが (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発生します。 選択されたボタンが [入力] の場合は、CREATE-WINDOW という関数 (ユーザが他の情報を入力するための新しいウィンドウを作成する関数) を関連付けることもできます。

Dialog System を使用することによって、ユーザ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できます。

ダイアログ文については、『ダイアログの使用方法』の章とヘルプトピックの『Dialog System の概要』を参照してください。

イベント

イベントは、ユーザインターフェイス内での変化を表し、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発生します。 イベントを発生させる操作の例は、次のとおりです。

- ユーザがキーを押した。
- マウスポインタがプッシュ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発生した。
- ユーザ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、本質的に同じです。 ただし、便宜上、次の 3 種類に分けられます。

- 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場合に発生します ([ファイル] メニューから [終了] を選択した場合など)。
- オブジェクトイベント - ウィンドウ、プッシュボタン、オプションボタン、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場合に発生します。
- キーボードイベント - キーボードのキーが押された場合に発生します。

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の順に検索します。 イベントは、ダイアログ定義を作成するときに定義します。 イベントは、イベントに関連付けられる関数によって定義されます。

関数

関数は、Dialog System が処理を実行するための命令です。 関数はイベントと関連付けられており、次のような場合に動作します。

- 関数が一覧表示されているイベントが発生した場合
- 関数が一覧表示されている手続きが実行された場合

Dialog System には、関数が多数あります。 スクリーンセットの移動に固有の関数 (SET-FOCUS、INVOKE-MESSAGE-BOX など) や 他のプログラム言語の命令に似た関数 (データ項目間のデータを転記する MOVE) などがあります。

各関数の詳細については、ヘルプトピックの『ダイアログ文：関数』を参照してください。

手続き

手続きとは、任意の名前が付いた関数セットで、サブルーチンとも呼ばれます。 手続きは、イベントと同様に見なすことができます。

データブロックとスクリーンセットの使用方法

Dialog System では、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、インターフェイ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。 ファイル名は、filename.gs の形式で指定します。 これには、エントリフィールドのようにインターフェイスで必要な画面上のオブジェクトが含まれます。

インターフェイスを実行すると、エントリフィールドなどのオブジェクトに、インターフェイスとアプリケーションプログラムの間で受け渡すデータが入力されます。 インターフェイスとアプリケーションプログラムの間でデータを受け渡すには、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付けます。 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。 関連付けるには、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存され、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 を呼び出すときに、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、次の要素が保存されます。

- データブロック

  すべての入出力に関するデータ定義です。

- 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付けられた動作

  これらの動作は、ダイアログイベントおよび関数と呼ばれ、ユーザが定義します。

Dialog System を使用したアプリケーションの作成手順

次の手順で Dialog System を起動します。

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

Dialog System を使用してアプリケーションを作成する手順は、次のとおりです。

1. データ定義と妥当性検査をします。

   『データ定義とスクリーンセットの作成』の章にある『データ定義の作成手順』とヘルプトピックの『データ定義と妥当性検査』を参照してください。
2. ウィンドウ、ダイアログボックス、および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 (エントリフィールド、テキストフィールド、オプションボタンなど) 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』、および『ダイアログの使用方法』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、定義処理の各段階が終わるたびに保存することをお勧めします。 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。

   その後でサンプルスクリーンセットを保存する場合は、[上書き保存] を使用できます。 別のバージョンのスクリーンセットを試す場合は、[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. スクリーンセットをテストします。

   スクリーンセット Animator を使用して、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 Animator』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再度テストします。

   ヘルプの『スクリーンセット Animator』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。

   前の手順に戻ります。

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『データ定義の作成手順』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用して、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用方法』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。

    ヘルプトピックの『デバッグ』を参照してください。
12. アプリケーションをパッケージ化します。

    ヘルプトピックの『コンパイルとリンク』を参照してください。

この順序に厳密に従う必要はありません。 データを定義する前にオブジェクトを定義することも可能です。ただし、ダイアログに指定するオブジェクトを定義する前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。

Dialog System の概要　ウィンドウオブジェクト

第 3 章 : データ定義とスクリーンセットの作成

ここでは、Dialog System を使用してアプリケーションを作成するための手順について説明します。

データモデルの設計

『Dialog System の概要』の章で説明したとおり、Dialog System アプリケーションを開発するための第一歩は、データ定義を作成するときに利用するデータモデルを設計することです。 このデータモデルは、インターフェイスを実行したときに、スクリーンセットが呼び出し側プログラムに渡す基本的なデータ項目を定義したものです。

データ定義は、呼び出し側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、ユーザインターフェイスで取得するデータを指定します。 このデータは、スクリーンセットが呼び出し側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基準も決定します。

データ定義と妥当性検査

開発サイクルのこの段階では、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、次のようなデータ特性があります。

- データ形式 (整数、文字列、計算用など)
- データサイズ
- 妥当性検査条件
- データ項目間の関係

モデルを作成すると、Dialog System とユーザプログラムの間で渡すデータを定義できます。

スクリーンセット全体に関する設計上の注意事項がいくつかあります。 詳細については、『スクリーンセットの使用方法』の章にある『スクリーンセットの操作の制御』を参照してください。

データ定義

インターフェイスとユーザのアプリケーションプログラムの間でデータを渡すには、エント

リフィールドなどのいくつかのオブジェクトを、ユーザの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。 これをデータ定義と呼びます。

マスタフィールドには、ユーザがエントリフィールドに入力したデータが格納されます。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。

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、すべてのユーザが簡単にデータを入力できます。 プロンプトモードと非プロンプトモードの相違は、次のとおりです。

- プロンプトモードでは、入力する行の種類に応じて、注釈、フィールド、グループのはじめ、またはグループの終わりプロンプトが表示されます。
- 非プロンプトモードで入力する場合は、行の入力が終わると、入力項目がフォーマットされます。 入力項目は、空白文字で区切る必要があります。

注釈

データ定義に注釈を付けることをお勧めします。アスタリスク文字で始まる行は、どれも注釈として処理され、妥当性検査や再フォーマットは実行されません。 たとえば、次のように記述します。

* これは注釈行です

データ定義の作成手順

次の手順に従って、データ定義を作成します。

- データ定義に使用するデータ項目を指定する
- データ形式コードを割り当てる
- データグループを使用する
- データの属性を調べる
- 妥当性検査基準を指定し、エラーメッセージを関連付ける

データブロック

データブロックには、Dialog System の「データの定義」ウィンドウで定義するデータ項目を指定します。 各スクリーンセットには固有のデータブロックがあり、コピーファイルとして呼び出し側プログラムに組み込まれます。

実行時には、Dialog System を呼び出す前に、呼び出し側プログラムによってこのレコードにユーザデータが転記されます。 Dialog System によってデータブロックに配置されたユーザ入力は、制御が戻ったときに、プログラムで使用されます。

関連するデータ項目を集団内に配置して、リストボックスなどのオブジェクトにマップできます。

「データの定義」ウィンドウのデータブロックは、表示、定義、および編集できます。
「データの定義」ウィンドウを表示するには、次の操作を実行します。

- 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例を、次に示します。

フィールド名	形式	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		

データブロックの形式は、Dialog System によって制御されます。

- プロンプトモードで入力された項目は、自動的にフォーマットされます。
- 非プロンプトモードで入力された項目は、行の入力が終わると、フォーマットされます。
- 行の中の各項目は、空白文字で区切ります。

エラーが発生すると、行の再入力要求する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、固定長であるため、長さを入力する必要はありません。

データ定義は、スクリーンセット用に 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 コピーファイルは、呼び出し側プログラムと Dialog System の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、呼び出し側プログラムは、Dialog System を呼び出す前に、データブロックにデータを転記します。 Dialog System は、制御を呼び出し側プログラムに戻す前に、データブロックにデータを転記できます。

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、スクリーンセットのデータ項目と関連するフィールド番号について記述します。

コピーファイルは、COBOL レコードを定義したファイルです。このファイルは、コンパイル時に呼び出し側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。 このファイルを変更する場合は、適合するようにプログラムを変更する必要があります。

プログラムを確実に適合させるために、チェック機構が使用されます。 データブロックコピーファイルの形式は、次のとおりです。

- データブロックは、COBOL プログラムの作業場所節 (Working-Storage Section) のレベル-01 レコードに対応します。
- 単一のデータ項目は、レコードのレベル-03 データ項目に対応します。
- データ集団は、レベル-03 集団データ項目に対応します。
- データ集団項目は、レベル-05 データ項目に対応します。

データ項目

データ定義で定義されたすべてのデータ項目は、スクリーンセットにグローバルに適用され、スクリーンセットダイアログで自由に参照できます。

標準的なデータ項目は、エントリフィールドまたはリスト項目にマップされます。 データ項目は、形式によって分類され、集団に配置したり、リストボックスなどのオブジェクトにマップしたりできます。 また、データ項目をフラグとして設定して、ダイアログから参照したり、呼び出し側プログラムに値を渡したりすることができます。

データブロックの各データ項目には、次の要素を指定する必要があります。

- データ項目名 (長さは最大 30 バイト)。
- 呼び出し側プログラムでデータ形式を判断するためのデータ形式コード。 データ形式コードは、次のどれかです。

X	英数字。
9	整数部と小数部を含む数字 (最大 18 桁)。 小数部の桁数は、9 桁以内にする必要があります。
A	英字。
S	符号付き数字。
C	計算用 (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)。 計算用データ項目は、どのエントリフィールドにも、表示または設定できません。
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、どのエントリフィールドにも、表示または設定できません。
N	DBCS(N)。 2 バイト文字集合。
G	DBCS(G)。 2 バイト文字集合。
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します。

- データ項目のサイズ。 この項目の意味は、選択したデータ形式によって、次のように変化します。

データ形式コード	サイズフィールドの意味
X または A	文字数。
9 または S	整数部の桁数と小数部の桁数。

C または C5	格納するバイト数。 このサイズによって、格納できる数の範囲を次のように指定します。 1.00 ～ 255 2.00 ～ 65,535 4.00 ～ 4,294,967,295 8.00 ～ 18,446,744,073,709,551,615 Dialog System では、値の下位 18 桁のみが処理されることがあります。
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、固定長です。 オブジェクト参照のサイズを指定できません。

次の例は、数字データ項目と符号付き数字データ項目に入力できる最大値と最小値の範囲を表しています。

```
LARGEST-VALUE            9     18.0
SMALLEST VALUE           S     0.9
```

次の例は、整数部が 3 桁で小数部が 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
```

データ項目の表示状態は、該当するエントリフィールドの PICUTURE 文字列によって制御されます。

次の例は、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
```

データ集団の使用方法

データ集団は、同じ種類のデータ項目の集合が、複数にわたって発生したり、繰り返されたりする場合に使用します。 データ集団のデータ項目は、データブロックおよび呼び出し側プログラムの添字によって参照されます。 呼び出し側プログラムでは、データ集団が、隣接するデータ項目として処理されるので、各データ項目にアクセスする場合には、添字が指標として使用され、集団内の相対的な位置が検索されます。 1 回のみ繰り返すようにデータ集団を定義した場合は、呼び出し側プログラム内ではデータ項目に添字は使用されませ

ん (Dialog System では添字を使用する必要があります)。

集団の入力は、集団名と繰り返し回数で構成されます。たとえば、次のようになります。

```
GROUP-1                150
```

ある集団に属するデータ項目は、集団名の下に一覧表示され、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を変更するには、データ項目を削除してから再定義するか、または、「データの定義」ウィンドウの [編集] メニューから [コピー]、[貼り付け]、および [削除] を実行します。

依存関係

依存関係とは、データ項目、オブジェクト、およびダイアログの関係を表す用語です。 依存関係は、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場合
- 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場合

名前付きオブジェクトの依存関係を問い合わせるには、メインウィンドウの [表示] メニューから [依存関係] を選択するか、または、「データの定義」ウィンドウの [オプション] メニューから [依存関係の表示] を選択します。

Customer サンプルスクリーンセットの依存関係を表示した「依存関係の照会」ダイアログボックスを図 3-1 に示します。

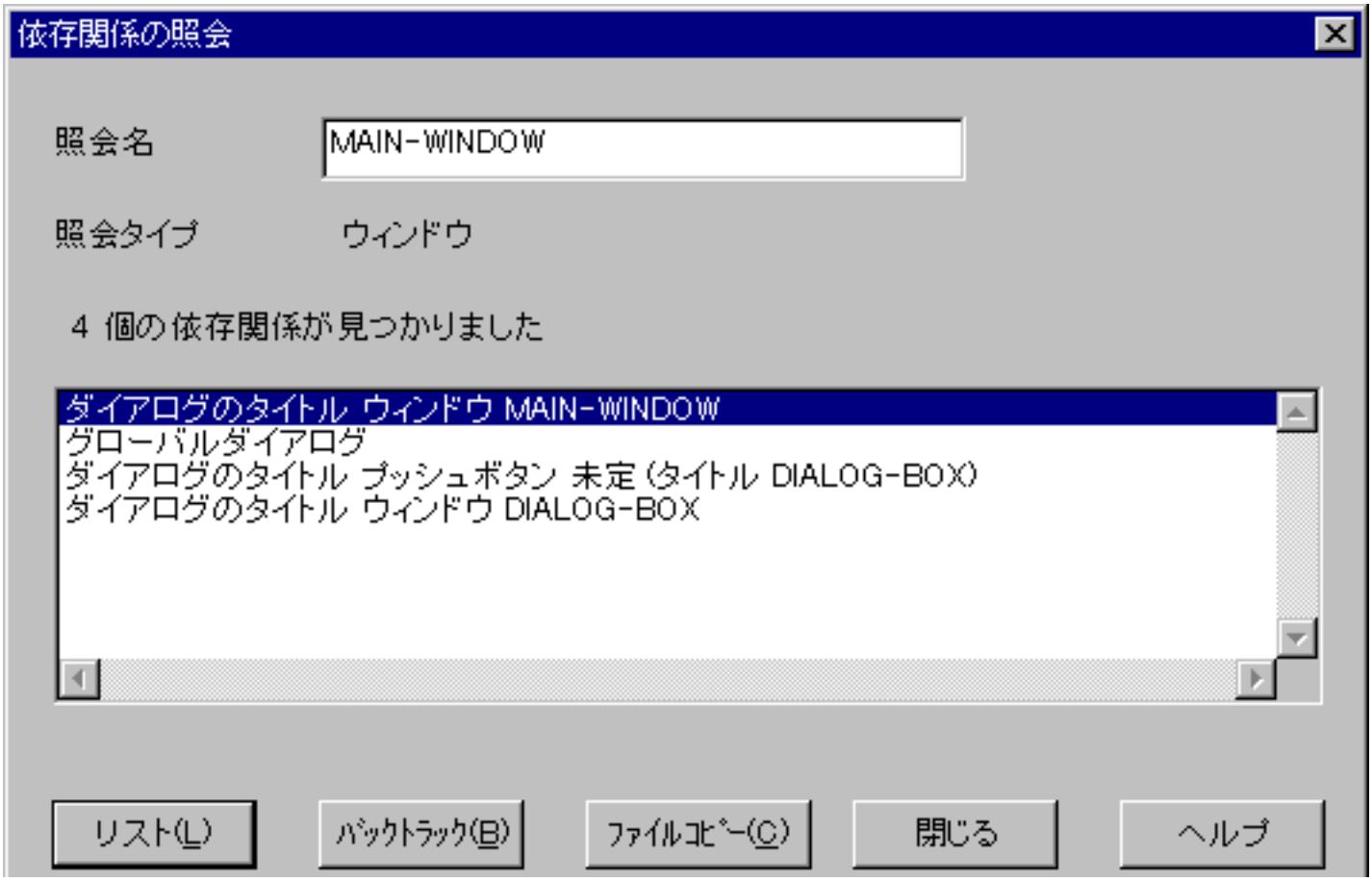

図 3-1：「依存関係の照会」ダイアログボックス

「データの定義」ウィンドウから依存関係を照会すると、「データの定義」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依存関係がダイアログボックスに表示されます。 メインウィンドウから依存関係を照会すると、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依存関係がダイアログボックスに表示されます。

「依存関係の照会」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依存関係を一覧表示できます。 関連する行をハイライトして、[リスト] をクリックします。 この機能によって、必要に応じて依存関係を連鎖的に追跡できます。

ユーザデータの妥当性検査

Dialog System では、ユーザがエントリフィールドに入力したデータが、指定した条件に一致するかどうかを妥当性検査できます。

入力データが妥当性検査されるのは、ダイアログに VALIDATE 関数を指定した場合のみです。 たとえば、メインウィンドウのすべてのエントリフィールドを妥当性検査するには、次のように指定します。

```
VALIDATE MAIN-WINDOW
```

メインウィンドウに選択ボックスを追加した場合は、この関数によって選択ボックスも妥当

性検査されます。 単一のエントリフィールドまたは選択ボックスのデータ入力を妥当性検査することもできます。

妥当性検査が失敗した場合は、VAL-ERROR イベントが生成されます。 VAL-ERROR イベントは、さまざまな方法で使用できます。 たとえば、妥当性検査が失敗したフィールドに入力フォーカスを戻すように設定できます。 この場合は、$EVENT-DATA レジスタには、妥当性検査が失敗したエントリフィールドの ID が格納されます。

```
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用できます。 メッセージボックスには、1 つのメッセージのみを表示するように定義することも、該当するエラーメッセージをメッセージボックスに転記してさまざまなメッセージを表示できるように定義できます。

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には、ERR-MSG の内容が表示されます。 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、ユーザがどのボタンをクリックしたか ([OK]、[キャンセル] など) を示すコードを格納します。

妥当性検査条件

次の妥当性検査条件を組み合わせて指定できます。

範囲 / 表	指定した値 (ゼロを含む) の範囲内または範囲外にデータがあることを妥当性検査します。 必要に応じていくつかの範囲を指定できます。 指定できる値は、数字または英数字です。 2 バイトの範囲を指定できません。
日付	データが指定した形式の有効な日付であることを妥当性検査します。 DBCS データ項目にはデータ妥当性検査を実行できません。
NULL	データが NULL または NULL 以外 (ゼロまたは空白文字) であることを妥当性検査します。 NULL は、DBCS の空白文字、英字フィールドまたは英数字フィールド、および数字フィールドのゼロです。 データ項目がすべて NULL の場合は、妥当性検査が失敗するように選択できます。

ユーザ定義	VALIDATE 関数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。 プログラム名の最大長は、パスを含めて 256 文字です。

Customer サンプルスクリーンセットでは、次の 3 つのデータ項目が妥当性検査されます。

- C-LIMIT - 1000 ～ 5000の範囲 / 表の妥当性検査
- C-AREA - N、S、E、W の値の範囲 / 表の妥当性検査
- C-ORD-DT - DDMMYY 形式の日付の妥当性検査

VALIDATE 関数と VAL-ERROR イベントについては、ヘルプトピックの『ダイアログ文：関数』と『ダイアログ文：イベント』を参照してください。

エラーメッセージの定義

データ妥当性検査のオプションとして、妥当性検査が失敗した場合に、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エラーメッセージを格納するオプションを指定できます。 エラーメッセージの使用方法は、条件によって異なります。

エラーメッセージの定義には、次の規則が適用されます。

- 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。
- エラーメッセージフィールドに一度に格納できるエラーメッセージの文字列は 1 つのみです。 エラーメッセージフィールドの既存の内容は、次のエラーメッセージによって上書きされます。
- スクリーンセットに一度に関連付けられるエラーメッセージファイルは 1 つのみです。

エラーメッセージを妥当性検査処理に関連付けるには、「データの定義」ウィンドウで妥当性検査する項目をハイライトします。 [妥当性検査] メニューを使用して、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するか、または、既存のエラーファイルを読み込むには、[ファイル] をクリックします。

Customer サンプルスクリーンセットでは、customer.err エラーファイル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用します。

- 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用します。

- 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用します。

新しいエラーメッセージを指定するには、[エラーメッセージ番号] に番号を、[エラーメッセージテキスト] 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処理に関連付けるには、メッセージをハイライトして、[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番号は、「エラーメッセージ番号」に表示されます。

説明したエラーメッセージのリスト内のデータ項目に対するデータ入力にエラーが検出された場合は、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] を選択すると、「エラーメッセージ定義」ダイアログボックスを使用できます。 このダイアログボックスでは、エラーメッセージとエラーメッセージファイルを編集できますが、妥当性検査処理にメッセージを関連付けできません。

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、ヘルプトピックの『データ定義と妥当性検査』を参照してください。

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では、アプリケーションに適切なオブジェクトを選択します。オブジェクトは、Dialog System のビルド単位です。詳細については、『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。

詳細情報

データ定義については、ヘルプトピックの『データ定義と妥当性検査』を参照してください。特に、データ集団の使用方法、データ集団の内部サイズ、生成されたデータブロックの

内容について説明しています。

データ定義とスクリーンセットの作成 コントロールオブジェクト

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インターフェースの設計に必要な第一歩について説明しました。 ここでは、ユーザが作成するアプリケーションに適合したウィンドウオブジェクトについて説明します。

オブジェクトは、Dialog System のビルド単位です。 オブジェクトには、2 種類あります。『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、ここで説明するウィンドウオブジェクトです。ここでは、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。
- 別のウィンドウ定義方法。
- メニューバーオプションの操作方法。

ウィンドウは、ユーザインターフェイ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、柔軟性に富んでいます。 ユーザ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め、すべてのオブジェクトをウィンドウに配置できます。

画面のレイアウトに使用するオブジェクトの定義

簡単なメニュー選択やデータエントリフィールドのみを配置したユーザインターフェイスから、メニューバー、オプションボタンなどのコントロールを設定した本格的なウィンドウまで、さまざまな形態のユーザインターフェイスを作成できます。

インターフェイスの内容は、次のような多くの要素によって決定されます。

- ユーザの種類と習熟度
- 利用可能なハードウェアとソフトウェアのリソース
- 実行すべきタスクの複雑さ

ただし、どの場合についても次の点が共通しています。

- ユーザが操作の流れを決定する。
- インターフェイスを制御するのは、プログラムではなく、ユーザである。
- ユーザとコンピュータとの対話は、簡単で自然にする。

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

ウィンドウに対して選択する属性は、ウィンドウに表示する内容、アプリケーションに対するユーザからの応答方法によって異なります。 選択できる属性は、次のとおりです。

タイトルバー	ウィンドウの最上部。ウィンドウのタイトルが表示されます。図 4-1 に示すウィンドウのタイトルは、一次ウィンドウです。
最小化アイコンと最大化アイコン	ウィンドウのタイトルバーの右端にあります。これらのアイコンを使用すると、ウィンドウを簡単に最小化または最大化したり、ウィンドウを元のサイズに復元できます。

境界	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、境界なしで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に提供するオプションがあります。図 4-1 で選択できる項目は、[関数] です。 GUI には、通常、各メニューバー項目に対し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) を設定します。
スクロールバー	ウィンドウに表示されている情報の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用すると、表示される情報を調整できます。 水平スクロールバーは、ウィンドウの下端にあり、横方向の表示を調整します。 垂直スクロールバーは、ウィンドウの右端にあり、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に、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、コントロールが表示されます。また、ユーザはほとんどのアプリケーション操作をここで行います。

デスクトップ

デスクトップとは、画面上の作業領域です。 デスクトップ上にウィンドウを作成できます。 インターフェイスには、多数のウィンドウを設定できます (ヘルプトピックの『Dialog System の制限』を参照してください)。ただし、インターフェイスを最初に実行したときに表示されるウィンドウは 1 つのみです。 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できます。

- 一次ウィンドウ
- 二次ウィンドウ

一次ウィンドウ

最初に作成したウィンドウを一次ウィンドウと呼びます。他のウィンドウが上位に存在しないため、一次ウィンドウは、デスクトップの子にあたります (デスクトップは一次ウィンド

ウの親です)。 一次ウィンドウは、他のどのウィンドウにも依存しないので、他のウィンドウでの操作は一次ウィンドウに影響しません。 複数の一次ウィンドウを作成できます。

二次ウィンドウ

すべての一次ウィンドウに、二次ウィンドウと呼ばれる子ウィンドウを作成できます。 これらの二次ウィンドウにも他の二次ウィンドウを作成できます。 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。 二次ウィンドウでの操作の多くは、一次ウィンドウの範囲によって決定されます。 二次ウィンドウを表示すると、その二次ウィンドウに関連付けられた一次ウィンドウも表示されます。 ウィンドウ間の関係は、ユーザ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、社員に関するアプリケーションには、一次ウィンドウとして新入社員ウィンドウを作成し、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を設定できます。

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。

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、その一次ウィンドウに関連付けられた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ともに移動し、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に表示されなくなります。 一次ウィンドウを復元すると、すべての二次ウィンドウも、元の位置に復元されます。

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 関数を使用して、他のウィンドウの子に変更できます。 この関数については、『ダイアログの使用方法』の章にある『デフォルトの親ウィンドウの変更』とヘルプトピックの『ダイアログ文：関数』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、情報を表示して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とウィンドウ』では、ダイアログボックスと二次

ウィンドウの使い分けに関するヒントを説明します。

クリッピング

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表示されない場合に、「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。

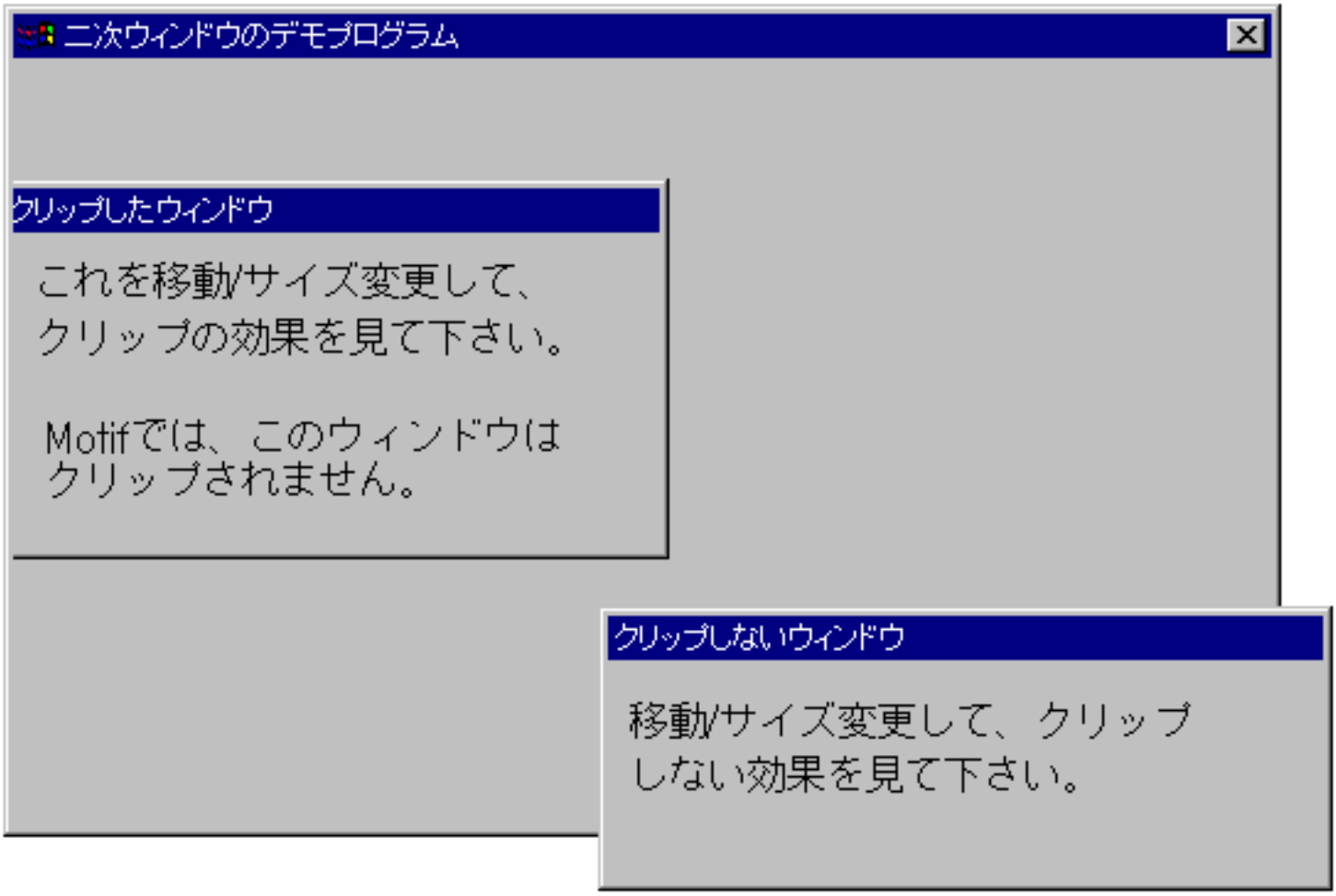

図 4-2： クリップされたウィンドウとクリップされていないウィンドウ

たとえば、画面を特定の視覚的レイアウトに設定したい場合に、二次ウィンドウをクリップします。 たとえば、ユーザが見慣れている用紙形式に近いウィンドウを表示する場合は、クリップされた二次ウィンドウを使用すると、視覚的に似せることができます。

一方、クリップされていない二次ウィンドウには柔軟性があります。 ユーザは、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。

Objects サンプルアプリケーションは、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。

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。 その後で、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隅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隅を任意の位置に置き、その後でマウスをクリックして固定します。 次に、ウィンドウの右下隅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。

クリップされた子ウィンドウの子としてダイアログボックスを指定すると、ダイアログボックス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と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と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ていない子ウィンドウをもつスクリーンセットを描く場合には、制限があります。 この場合は、子ウィンドウを通常の方法で新しい位置に正しく移動できません。まず、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。 移動後にクリップ属性を削除できます。

ウィンドウのサイズには制約はなく、Dialog System の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。

「ウィンドウの属性」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、『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、「ウィンドウの属性」ダイアログボックスでは、ウィンドウの属性を選択できます。

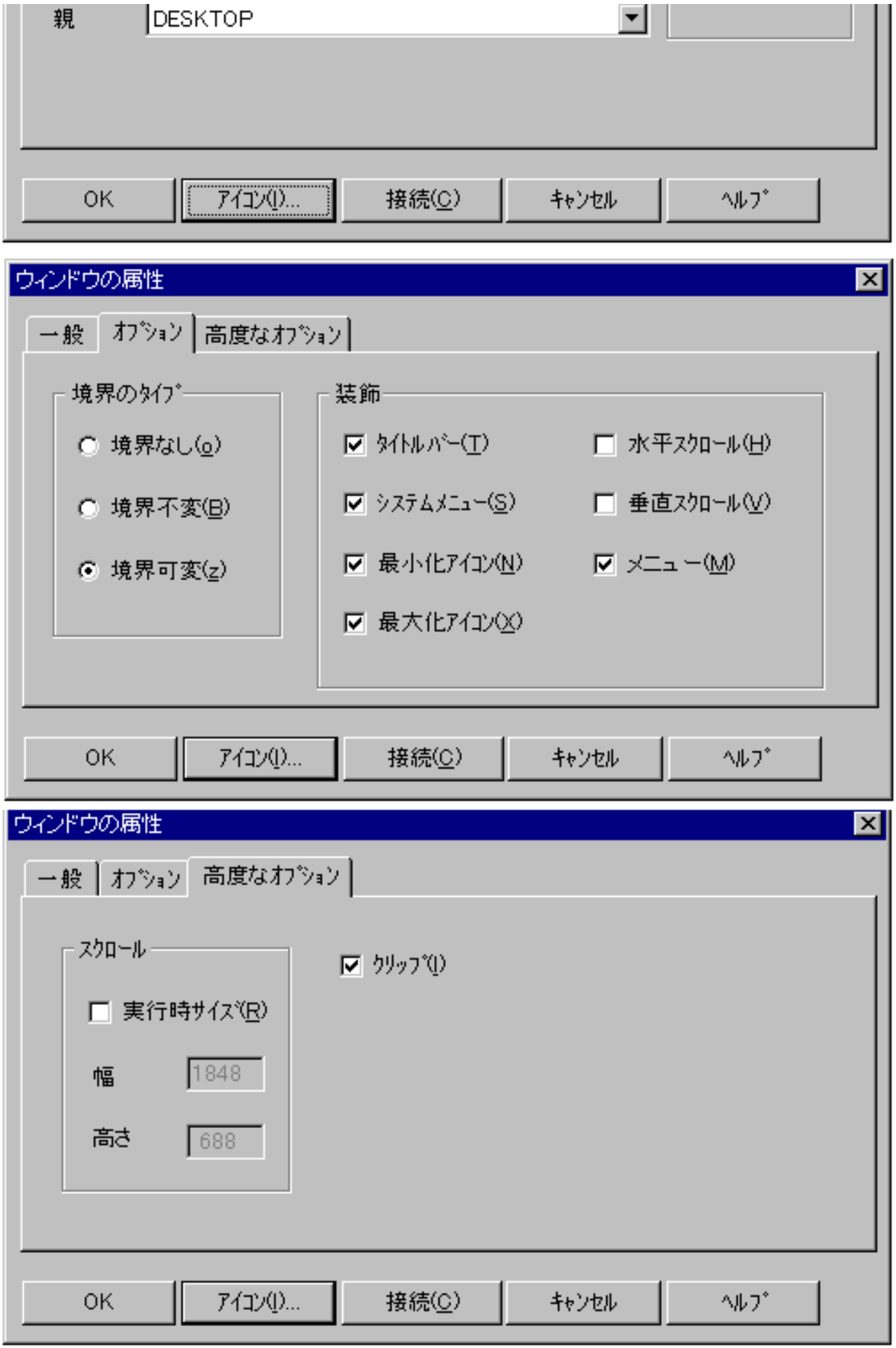

図 4-3： 「ウィンドウの属性」ダイアログボックス

これらの属性については、ヘルプを参照してください。

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、ダイアログを使用してウィンドウを操作できます。 詳細については、『ダイアログの使用方法』の章にある『ウィンドウダイアログ』を参照してください。

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ために、最もよく利用されます。 この情報は、テキストフィールド、データエントリフィールド、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または取得されます (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ません。また、メニューバーを設定できません。
- 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、ユーザがプッシュボタンをクリックした場合に Dialog System がユーザの操作を受け付けるように設定します (たとえば、リストからの選択を受け付けるための [OK] ボタン)。 ユーザがプッシュボタンをクリックした場合に Dialog System がユーザの操作を無視するように設定することもできます (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)。
- 同じダイアログを使用する複数のダイアログボックスを 1 つのウィンドウとして操作できます。

図 4-4 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。

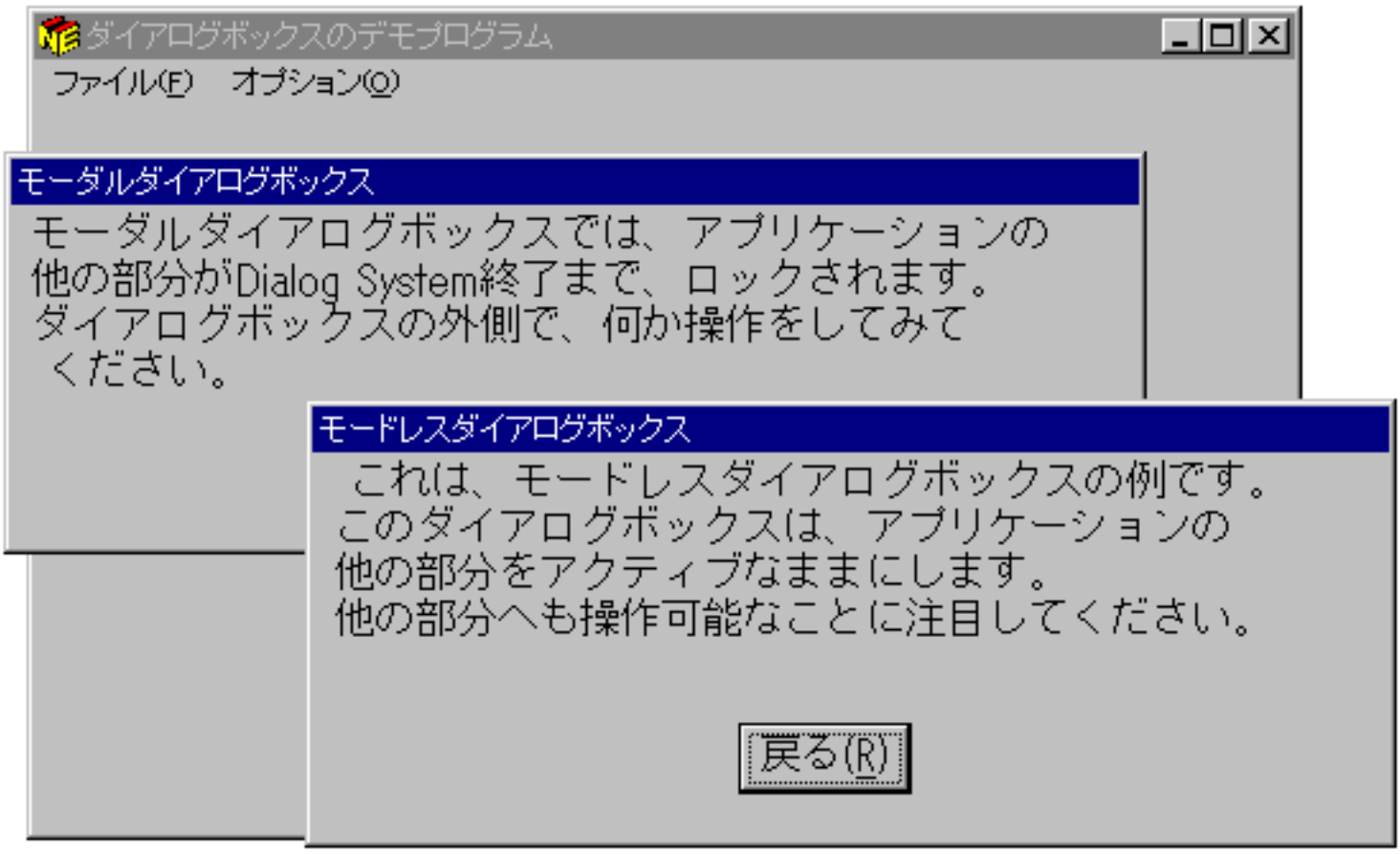

図 4-4： 標準的なダイアログボックス

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、モードレスにできます。 その中のどれを選択するかは、ダイアログボックスに対するユーザの対応方法によって決まります。

親モーダルのダイアログボックスを使用すると、ユーザ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を操作できません。 つまり、親モーダルの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できません。 ただし、他の一次ウィンドウ、および、そのアプリケーション外のオブジェクトにはアクセスできます。

アプリケーションモーダルのダイアログボックスを使用すると、ユーザ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の部分とアプリケーション内のすべてのオブジェクトを操作できません。

アプリケーションモーダルのダイアログボックスは、ユーザによるファイル名入力を必要とする場合などに使用します。 アプリケーションは、ユーザがファイル名を入力する (または操作を取り消す) まで、先に進むことができません。

Dialog System では、モーダルが設定された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が設定されたダイアログボックスです (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] を選択します)。

モードレスダイアログボックスを使用する場合は、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ためにすぐに必要とは限りません。

モードレスダイアログボックスでは、ユーザがアプリケーションと最も柔軟に対話できるので、モードレスダイアログボックスをお勧めします。 ただし、前述のように、アプリケーションモーダルの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、アプリケーションモーダルの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を表しています。 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ください。

ダイアログボックスとウィンドウ

ここまでの説明のとおり、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。 実際に、ダイアログボックスは、特殊なウィンドウといえます。 『グラフィカルユーザインターフェイス』の章にある『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ダイアログボックスとウィンドウのどちらを使用するかを選択することがあります。 ダイアログボックスとウィンドウの使い分け基準は、次のとおりです。

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用します。

- アプリケーションでモーダル入力が必要な場合。 たとえば、ユーザがアプリケーションにデータを入力する必要があり、ユーザが応じるまでは処理を続行できない場合は、モーダルが設定されたダイアログボックスを使用します。

- 提示または収集する情報量が少なく、メニューバーが不要で、ダイアログボックス内に情報が収まる場合。

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用します。

- ユーザがアプリケーションのメニューバーの項目リストから項目を選択する場合。

- 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、ユーザがデータやコントロール全体を表示するときに、ウィンドウのサイズを変えたり、スクロールしたりする必要がある場合。

- オブジェクトに子ウィンドウが必要な場合。 ウィンドウを使用すると、アプリケーションをより論理的に設計できます。

メッセージボックス

メッセージボックスは、ユーザにメッセージを提示するために利用されます。 メッセージボックスに配置できるのは、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、ユーザが操作を選択したり、メッセージボックスを削除したりするためのボタンのみです。 通常、これらのメッセージは、「ファイルが見つかりません」のようなイベントの結果として、アプリケーションによって表示されます。

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。

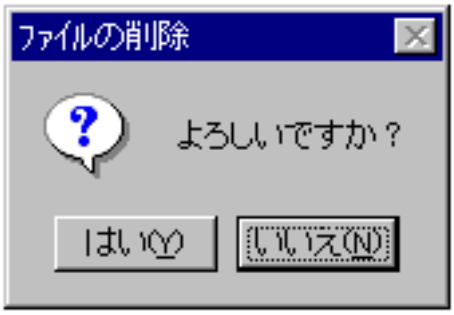

図 4-5： 標準的なメッセージボックス

Dialog System では、メッセージの性質に応じて、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。 アイコンに違いがあるため、各種類を識別できます。

注意	応答してもしなくてもかまわないような状況をユーザに通知します。 たとえば、ユーザにプリンタの用紙切れを知らせることは、ユーザの現在の作業には影響しません。
情報	ユーザに情報を伝えます。 ユーザが入力する必要があるのは、「メッセージを読んだ」という確認のみです。 たとえば、製品情報メッセージは、情報メッセージです。
警告	望ましくない結果が起こりうる状況であることを通知します。 たとえば、ユーザ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できます。
質問	ユーザの応答が必要な状況を表します。 たとえば、ユーザが [ファイルの削除] オプションを選択すると、削除を実行してもよいかどうかをユーザに確認するメッセージが表示されます。
致命的	ユーザ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ことを通知します。 たとえば、ユーザがファイルの最大数を開いている場合に、操作を続行すると、システムが停止します。

メッセージボックスには、ユーザ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、次のような定義済みのプッシュボタンの組み合せ (6 種類) を使用できます。

- [OK]
- [OK] と [キャンセル]
- [再試行] と [キャンセル]
- [中止]、[再試行]、および [無視]
- [はい] と [いいえ]
- [はい]、[いいえ]、および [キャンセル]

たとえば、次のように使用します。

- [OK] ボタンは、すべての情報メッセージで、ユーザがメッセージを受け取ったことを確認するためのみに使用します。
- [はい] と [いいえ] ボタンの組み合せは、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使用します。

ユーザがダイアログで選択したボタンは判断可能です。 INVOKE-MESSAGE-BOX 関数のコードに、クリックされたボタンを識別するコードが一覧表示されています。 ヘルプトピックの『ダイアログ文：関数』を参照してください。

たとえば、ユーザがメニューから [ファイルの削除] を選択したことを想定します。メッセージボックス (上記の図 4-5) が起動し、ユーザがファイルを削除してもよいかを確認できます。

メニュー

メニューは、3 種類あります。

- 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

メニューバー

メニューバーは、ウィンドウの上部に位置するコマンドのリストです。通常のメニューバーでは、あるコマンドを選択すると、次に選択する項目が表示されるので、正確には、メニューバーは、コマンドグループのリストです。

メニューバーの定義については、ヘルプトピックの『オブジェクトと属性』を参照してください。

メニュー項目では、次の操作を実行できます。

- 別のプルダウンメニューを表示します。 別のプルダウンメニューがある場合は、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します。 項目に続いて省略記号 (...) が表示されます。

- チェックマークされた選択項目を表示します。 オンとオフの 2 つの状態があります。 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、チェックマークが項目の先頭に表示されます。 このような項目は、トグルとも呼ばれます。

- 操作を直接実行します。 この機能を識別するための情報は表示されません。

メニュー項目を選択すると、上記のリストのどれか 1 つの操作が実行され、対応する情報 (省略記号やチェックマークなど) が表示されます。

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操作を実行できます。図 4-6 は、標準的なプルダウンメニューを示しています。

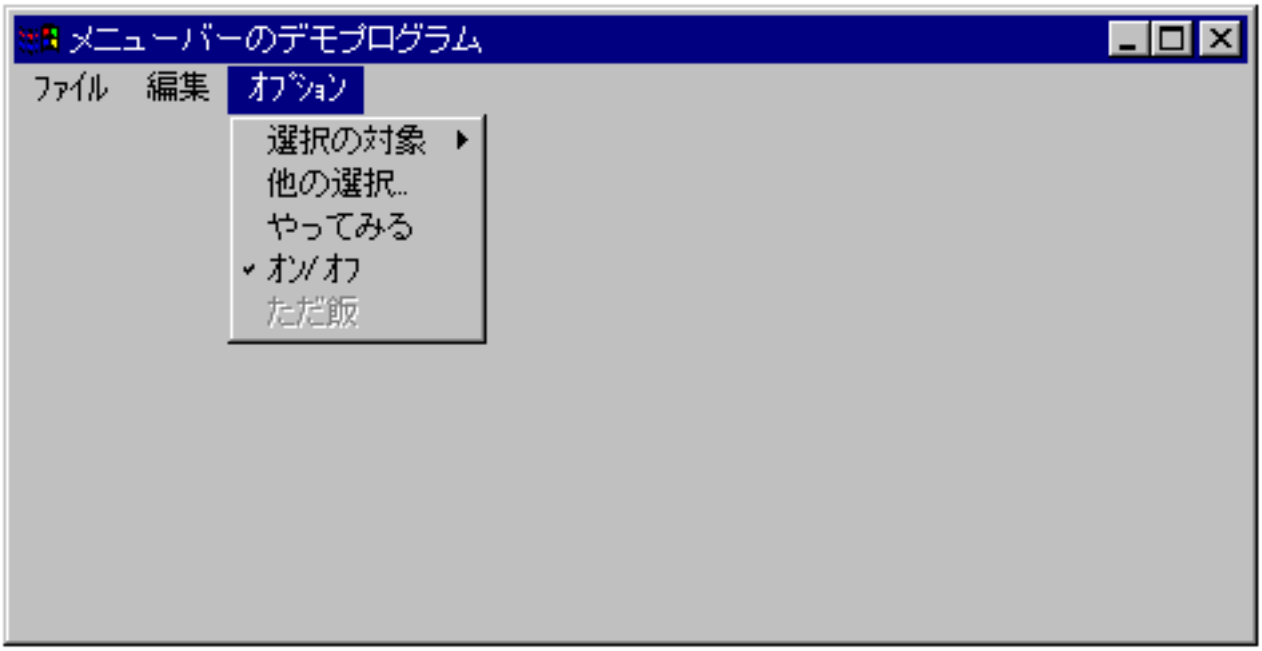

図 4-6： 標準的なプルダウンメニュー

項目を選択した後の操作には、いくつかの種類があります。 通常、メニュー項目には、操作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、そのマークと関連付けられた操作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、ユーザが選択できるメニュー項目をさらに表示します。
チェックマーク	メニュー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には、その項目は有効であることを示します。 オフの場合には、その項目は無効です。 チェックマークがオフの場合は、すぐに実行され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	そのメニュー項目を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の選択やオプションボタンの選択など、より複雑な選択項目を表示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	操作をすぐに実行します。

メニューバーダイアログの例については、『ダイアログの使用方法』の章にある『メニューバーダイアログ』と『項目の有効化と無効化』を参照してください。

ヘルプトピックの『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。

コンテキストメニュー

コンテキストメニューを使用すると、ユーザが Dialog System の作業領域で早く簡単に操作を実行できます。 画面上を右クリックすると、クリックした項目に適したメニュー項目が表示されます。 たとえば、メインの定義ウィンドウのプッシュボタンオブジェクトと、「ダイアログの定義」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は、次のとおりです。

1. [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。

2. ダイアログの任意の行を右クリックします。

   Dialog System では、図 4-7 のようなコンテキストメニューを表示します。

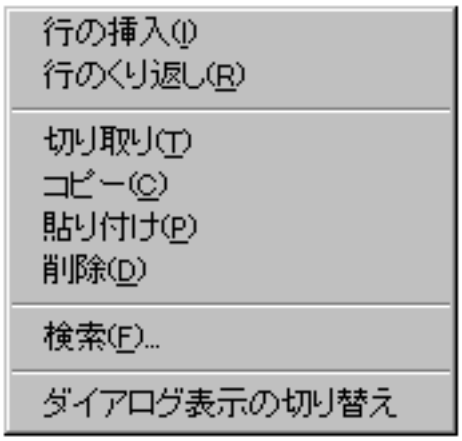

図 4-7： 汎用ダイアログのコンテキストメニュー

アイコンの設定

アイコンとは、小さい絵記号のことであり、ウィンドウを最小化したときに表示されます。 ウィンドウまたはダイアログボックスを作成する場合は、最小化したときに使用するアイコンを設定できます。

アイコンをウィンドウに設定するには、次の操作を実行します。

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。

   「ウィンドウの属性」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、「アイコンの選択」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： 「アイコンの選択」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、「ウィンドウの属性」ダイアログボックスに戻ります。

   最小化のアイコンを確実に選択するようにしてください。その他のアイコンは使用されません。
6. [OK] をクリックすると、「オブジェクトの定義」ウィンドウに戻ります。

ウィンドウオブジェクト　ダイアログの使用方法

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説明しました。 ここでは、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用方法
- コントロールのグループ化
- コントロールの整列

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、ウィンドウとダイアログボックスで使用できます。 コントロールオブジェクトは、ウィンドウまたはダイアログボックスの外側に指定することはできません。そのため、ウィンドウまたはダイアログボックスを定義してから、コントロールオブジェクトを定義する必要があります。 詳細については、『ウィンドウオブジェクト』の章を参照してください。

コントロールでは、その種類に応じて次の操作を実行できます。

- データの入力 (たとえば、ユーザが値のリストから値を選択します)
- 異なる表示間の移動
- データの表示

ここで説明するコントロールは、次のとおりです。

コントロール	用途
テキストフィールドとエントリフィールド	データ項目の読み込みと入力を行うフィールドやラベル用のテキストが表示されるフィールドです。
プッシュボタン	ユーザが操作を選択して、すぐに実行できるようにします。
オプションボタン	グループ化した場合に相互排他的に選択する項目 (グループ内で 1 つしか選択できない) をまとめてユーザに表示できます。
チェックボックス	相互排他的ではない選択項目 (複数を選択できる) のセットです。ユーザは、一部またはすべての項目を選択できます。項目を選択しないことも可能です。
リストボックス	項目のリストです。ユーザは、1 つまたは複数の項目をリストから選択できます。
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、選択した項目をエントリフィールドに配置します。
スクロールバー	エントリフィールドなどの他のコントロールとともに使用した場合に、ユーザが特定の値まで「スクロール」できます。
グループボックス	ユーザが見やすいように、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区切り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。
OLE2 コントロール	外部のオブジェクト (スプレッドシートなど) を呼び出すことができるウィンドウを表示します。

ユーザコントロール	Dialog System 定義ソフトウェアでは作成できないオブジェクトを設定します。ユーザコントロールの使用方法は、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も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。

テキストフィールドとエントリフィールド

最も基本的なコントロールは、エントリフィールドです。 ここでは、次の内容について説明します。

- テキストフィールド、単一行エントリフィールド、および複数行エントリフィールドの特徴
- 他のコントロールとともにエントリフィールドを使用する方法

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、ユーザは、表示されている情報や入力する情報を理解できます。 テキストフィールドはラベルと見なすことができます。テキストフィールドの使用例を図 5-1 に示します。

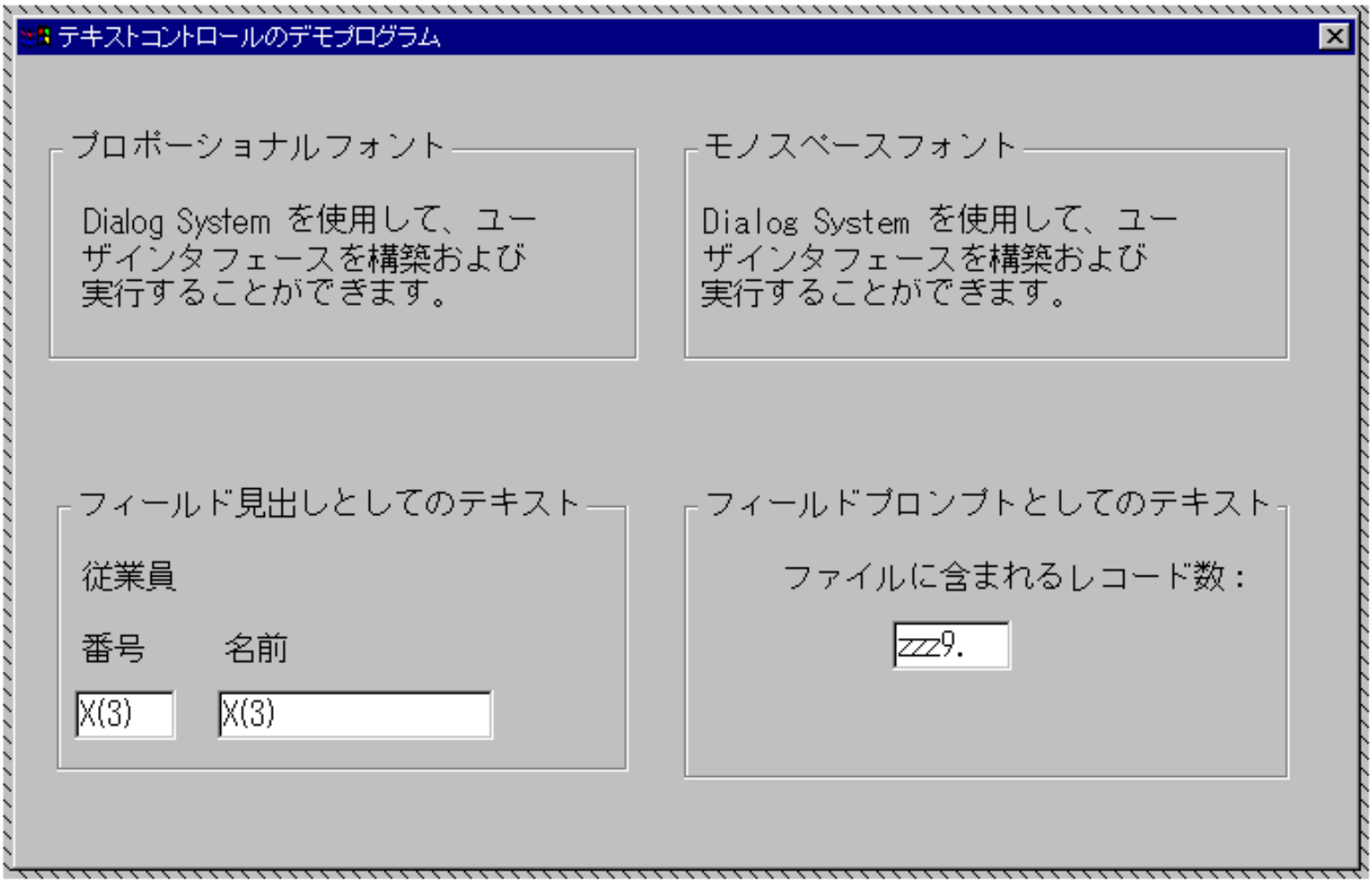

図 5-1： テキストフィールドの例

テキストフィールドは、次のように使用します。

- フィールドや列の見出し
- 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、イベントや関数を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は、表示専用エントリフィールドを使用します。 詳細については、『表示専用エントリフィールド』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は、ヘルプトピックの『オブジェクトと属性』を参照してください。

エントリフィールドからの入力情報の取得

エントリフィールドには実行時にデータ項目の内容が表示され、ユーザがその内容を編集できます。 エントリデータ項目に必要な値が不明な場合に使用します。 この場合は、ユーザが必要な情報を入力する必要があります。

エントリフィールドをデータ項目に関連付けます。 ユーザがエントリフィールドを変更すると、ユーザが追加で操作をしなくても、その結果がすぐにデータ項目に反映されます。 データ項目のエントリが変更された場合は、その変更が反映される前に、エントリフィールドを最新表示する必要があります。エントリフィールドは、次のような場合に最新表示されます。

- 実行時にエントリフィールドの親ウィンドウが作成された場合
- エントリフィールド、または、エントリフィールドの親ウィンドウが、ダイアログ関数 REFRESH-OBJECT によって明示的に最新表示された場合
- 入力フォーカスが設定された場合

エントリフィールドの親ウィンドウが SHOW-WINDOW 関数によって表示されている場合は、エントリフィールドは最新表示されません。

数字エントリフィールドにフォーカスを設定した場合に、関連付けられたマスタフィールドに有効な数字データが入力されていないと、その値はゼロに設定されます。

Dialog System には、単一行エントリと複数行エントリ (MLE) の 2 種類のエントリフィールドがあります。

単一行エントリフィールド

単一行エントリフィールドは、最も簡単なエントリフィールドで、1 行のテキストのみを表示または収集できます。

たとえば、Customer サンプルでは、顧客コード、名前、住所、クレジットの上限などの情報をユーザが入力する必要があります。 これらの情報は、エントリフィールドから入力します。

単一行エントリフィールドの例を図 5-2 に示します。

図 5-2： エントリフィールドの例

他のコントロールとともにエントリフィールドを使用する方法

エントリフィールドを他のコントロールとともに使用すると、ユーザ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は、エントリフィールドとともにスクロールバーを使用すると、ユーザは「スクロール」によって適切な値を選択できます。

エントリ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。

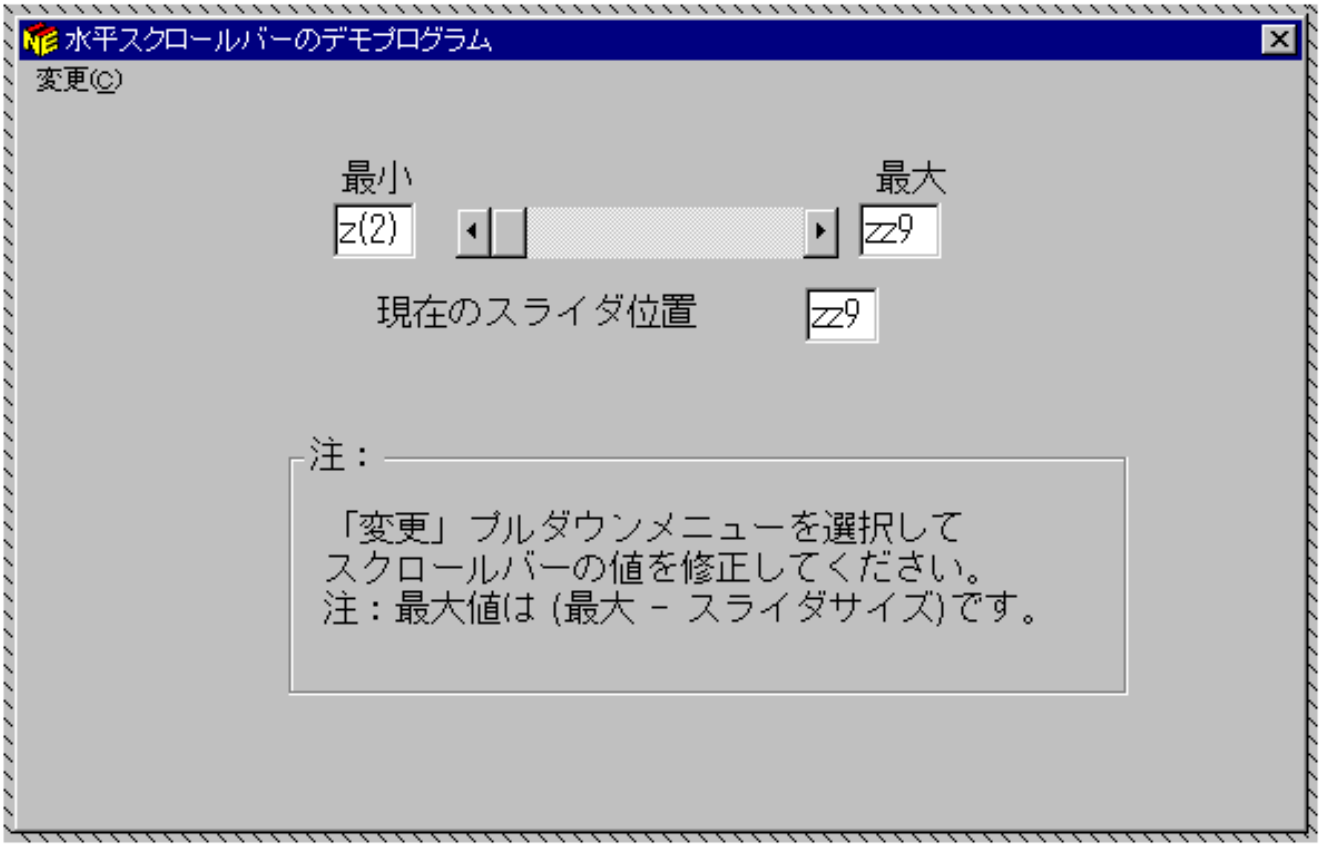

図 5-3： エントリフィールドとスクロールバーの使用

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。

ヘルプトピックの『データ定義と妥当性検査』では、データ定義を使用した妥当性検査の設定方法について説明しています。

表示専用エントリフィールド

エントリフィールドによっては、表示専用にする必要がある場合があります。 たとえば、従業員の住所情報を更新するためのダイアログボックスを作成すると仮定します。 従業員の氏名などのフィールドは、ユーザが閲覧してもかまいませんが、更新されると困ることがあります。

表示専用エントリフィールドのもう 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定していない場合もあります。 この場合は、プロンプトを表示専用エントリフィールドにしてから、実行時に正しいテキストを指定することができます。

動的プロンプトの作成方法を説明するために、ここでは、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エントリフィールドがあると仮定します。 プログラムでは、次のような文を使用して項目に入力します。

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

generic-prompt と関連付けられたエントリフィールドが最新表示されると、新しいプロンプトが表示されます。

「表示専用」は、単一行エントリフィールドの属性です。 エントリフィールドを表示専用に設定する方法については、ヘルプトピックの『オブジェクトと属性』を参照してください。

自動スワイプ

自動スワイプは、ユーザがフィールドに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エントリフィールドの属性です。 ユーザがフィールドに何らかのデータを入力する場合は、新しいデータを入力する前にそのフィールド内がクリアされます。

ユーザがデフォルト値を使用しないことがわかっている場合は、自動スワイプを使用して、そのデフォルト値をクリアできます。 たとえば、エントリフィールドを使用してファイル名を入力する場合は、filename.ext のようなわかりやすい文字列をデフォルト値にします。 このように指定すると、必要なファイル名の形式がわかりやすくなります。ただし、ファイルにデフォルトのファイル名が必要な場合はほとんどありません。

ユーザ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エントリが始まります。

SET-AUTOSWIPE-STATE 関数を使用すると、エントリフィールドの自動スワイプ状態を変更できます。 上記の例では、ユーザが有効なファイル名を入力した場合は、SET-AUTOSWIPE-STATE をオフに設定できます。 その場合は、ユーザがこのフィールドにタブ移動しても、データの入力前にフィールドのデフォルト値がクリアされません。 詳細については、ヘルプトピックの『ダイアログ文：関数』を参照してください。

複数行エントリフィールド

複数行エントリフィールド (MLE) では、単一行エントリ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できます。 このコントロールが最も便利なのは、製品に関する膨大な説明などの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も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。 左側の MLE にはワードラップオプションが設定されており、右側の MLE には設定されていません。

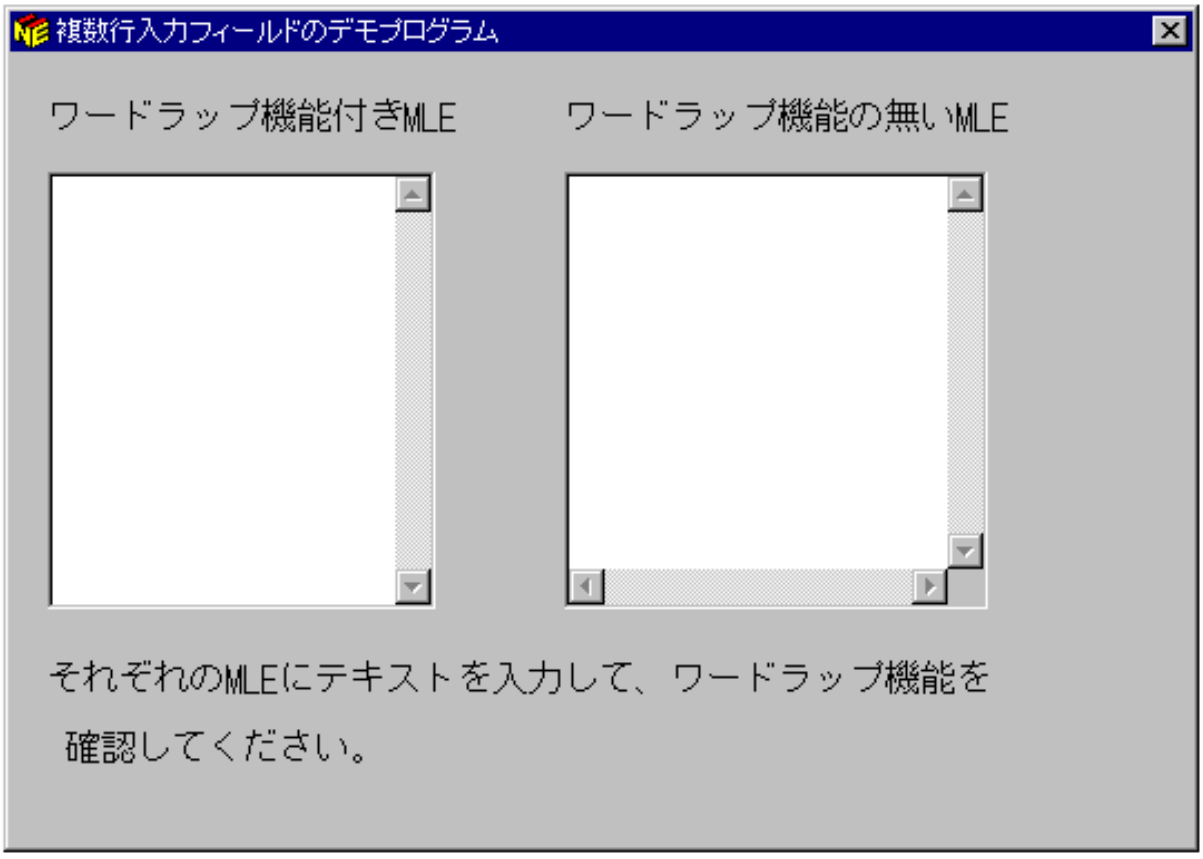

図 5-4： 複数行エントリフィールド

MLE に関連付けるデータ項目には、英数字、DBCS 文字、改行文字 (ANSI の 10 進数 10、x"0A") を入力できます。改行文字は、予期と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用すると、右または下の境界を超えて、文字を表示できます。

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 の相違が示されています。

MLE の編集

MLE にテキストを入力したり、MLE 内のテキストを編集したりするには、次の操作を実行します。

- アプリケーションプログラム内の関連付けられたデータ項目にテキストを転記します。
- ダイアログを使用してテキストをデータ項目に転記します。 この場合は、改行文字を挿入できません。
- クリップボードから MLE にテキストを直接読み込みます。
- ユーザが MLE 内で情報を直接編集できるようにします。

MLE の最新表示

実行時に MLE のデータ項目を変更した場合は、MLE を最新表示する必要があります。 MLE は、次のような場合に最新表示されます。

- 実行時に MLE の親ウィンドウが作成された場合
- MLE、または、MLE の親ウィンドウが、ダイアログ関数 REFRESH-OBJECT によって明示的に最新表示された場合

ユーザが MLE を変更すると、ユーザが追加で操作をしなくても、その結果がすぐにデータ項目に反映されます。

プッシュボタン

プッシュボタンは、ユーザが操作を選択するためのボタンです。 プッシュボタン内のテキストによって示される操作がすぐに実行されます。プッシュボタンが設定された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

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は、「プッシュボタンの属性」ウィンドウを使用して、プッシュボタンにテキストのかわりにビットマップのセットを設定できます。 プッシュボタンの 3種類の状態 (通常の状態、無効な状態、押し下げた状態) について、ビットマップを定義できます。

通常の状態のビットマップは、定義する必要があります。 押し下げた状態または無効な状態のビットマップを定義しない場合は、通常の状態のビットマップが使用されます。

実行時にビットマップを設定または変更する場合については、 『マウスポインタのビットマップの変更』の章を参照してください。

オプションボタン

プッシュボタンとは異なり、オプションボタンは、グループ化した場合に相互排他的に選択する項目 (グループ内で 1 つしか選択できない) をまとめてユーザに表示できます。オプションボタンの使用例を図 5-6 に示します。この例では、ユーザはどれかの請求方法を選択する必要があります。

図 5-6： 標準的なオプションボタン

オプションボタンをグループ化すると、グループ内で一度に選択できるボタンが 1 つに限定されます。 ユーザが別のボタンを選択すると、前に選択されていたボタンの選択が解除されます。 コントロールグループの定義方法については、『コントロールのグループ化』を参照してください。

オプション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、オプションボタン「VISA」には、次のようなダイアログを設定できます。

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

BILL-TO-VISA 手続きには、必要な請求用の関数が含まれています。

注：オプションボタンの選択を解除するには、コントロールグループ内の他のオプションボタンを選択する必要があります。

オプションボタンの定義については、ヘルプトピックの『オブジェクトと属性』を参照してください。

チェックボックス

チェックボックスは、ユーザが相互排他的ではない項目を選択できることを除き、オプションボタンとよく似ています。たとえば、図 5-7 に示すように、「該当する項目をすべて選択する」ような状況では、チェックボックスを使用できます。

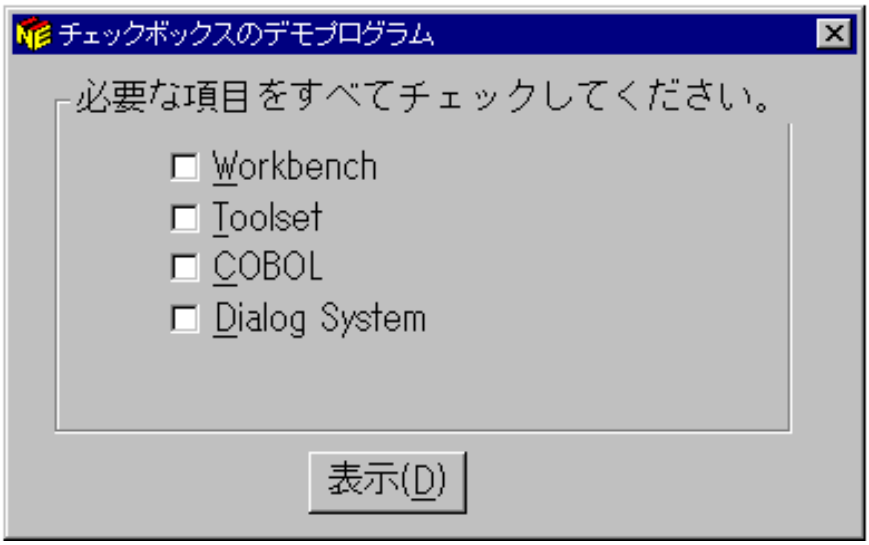

図 5-7： チェックボックスのグループ

チェックボックスには、 数字データ項目を設定できます。 チェックボックスを設定した場合は、そのチェックボックスに関連付けられたデータ項目が 1 に設定されます。 チェックボックスを設定していない場合は、データ項目が 0 に設定されます。

詳細については、『サンプルプログラム』の章を参照してください。

リストボックス

リストボックスを使用すると、ユーザが 1 つまたは複数の項目を選択できるリストが表示されます。 リストボックスは、次の要素で構成されます。

- 項目のリストを含むウィンドウ
- 表示されていない部分の情報を表示するためのスクロールバー (1 つ以上)

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 - リストから 1 つの項目のみを選択できます。
- 複数選択リストボックス - 任意の数 (リストボックスに表示されている項目の数を上限とする) の項目を選択できます。

- 範囲選択リストボックス - 任意の数 (リストボックスに表示されている項目の数を上限とする) の隣接する項目を選択できます。

Objects 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が示されています。

リストボックスの例を図 5-8 に示します。

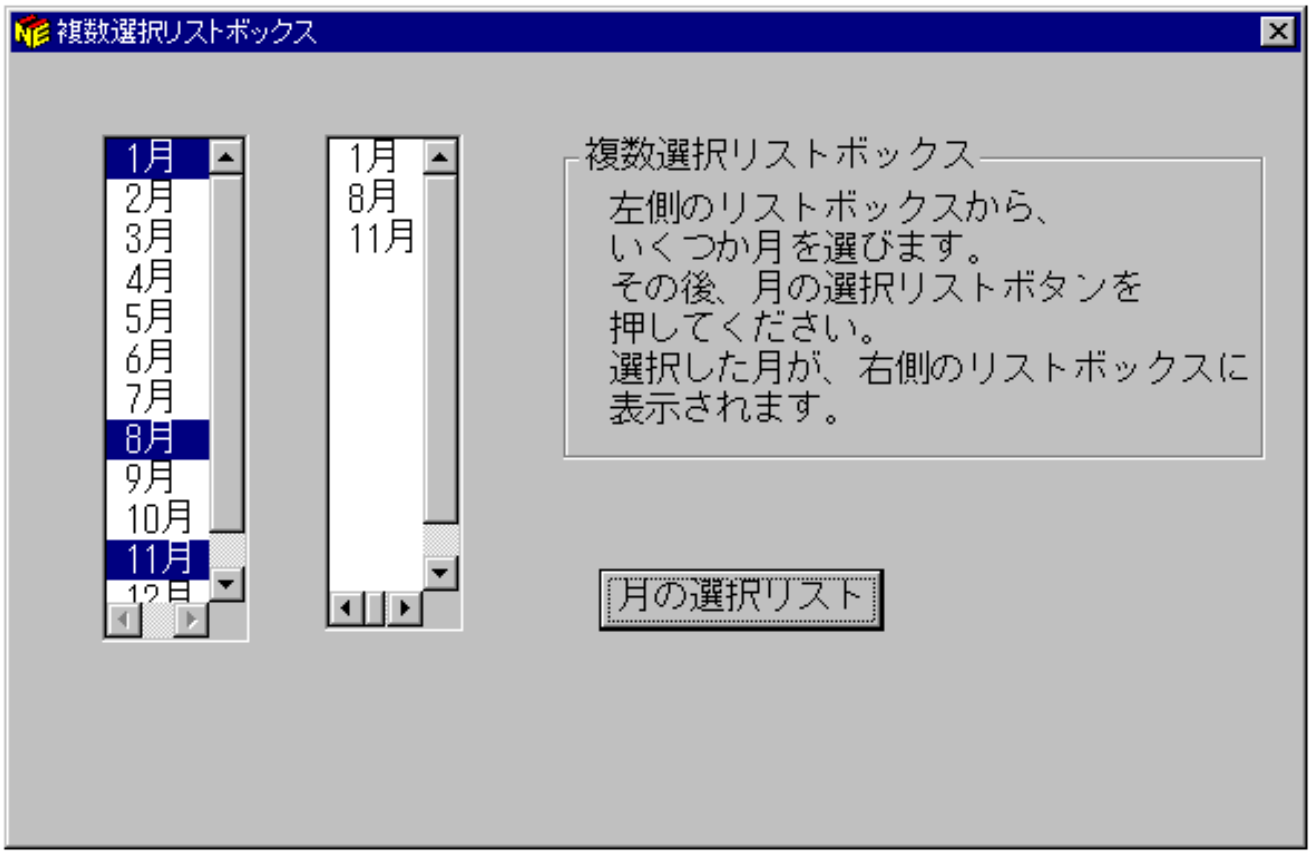

図 5-8： リストボックスの例

リストボックスの定義については、ヘルプトピックの『オブジェクトと属性』を参照してください。

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集団項目を使用する。

  リストボックスの各行が集団のオカレンス (要素) を示すように、 リストボックスを集団項目と関連付けることができます。 集団内で選択された項目が各行のフィールドになります。

  この方法は、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。
- リストボックスの属性を定義する場合に定義機能を使用する。

  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できます。 これらの項目は、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。 その後の処理は、他のリスト項目と同様に行われます。 たとえば、これらの項目を変更したり、削除したりできます。項目間に他の項目を挿入することもできます。

この方法は、項目数が少ないリストに項目を追加する場合に便利です。 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、ヘルプトピックの『オブジェクトと属性』を参照してください。

- 実行時にダイアログで INSERT-LIST-ITEM 関数を使用する。

  この方法は、リストボックスに項目を追加する際に、リストの選択項目が少なく、データブロックを小さくしたいときに便利です。 小さいリストボックスに項目を設定する場合はあまり時間がかかりませんが、実行するダイアログ文を追加する場合は多少時間がかかります。
- 区切られた文字列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が少なくなります。
  - Dialog System 以外でもリストを維持できます。 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維持し、実行時にプログラムで読み込むことができます。 この方法では、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、データに比較的小さい変更を加えることができます。

  プログラムと Dialog System の間で渡されるデータブロックのサイズは大きくなりますが、データ項目をリストに 1 つず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。

選択ボックス

選択ボックスは、スクロール可能な項目リストを表示して、エントリフィールドに選択項目を入力するためのコントロールです。 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。

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

選択ボックスの種類:

単純 (固定)	エントリフィールドとリストボックスで構成されます。 ユーザは、エントリフィールドにデータを入力するか、または、リストから項目を選択できます。選択した項目はエントリフィールドに配置されます。
ドロップダウン	右端の矢印ボタンと表示されないリストボックスが設定されたエントリフィールドで構成されています。 ユーザが矢印ボタンをクリックすると、ドロップダウンリストボックスが表示されます。 ユーザは、エントリフィールドにデータを手入力するか、または、ドロップダウンリストから項目を選択してエントリフィールドに値を指定します。 ユーザが項目を選択するか、または、別の場所をクリックすると、リストボックスが表示されなくなります。
ドロップダウンリスト	右端の矢印ボタンと表示されないリストボックスが設定された表示専用エントリフィールドで構成されています。 ユーザは、表示専用フィールドにはデータを入力できません。リストから項目を選択する必要があります。 表示専用エントリフィールドは、現在の選択のみを表示するために縮小されたリストボックスと見なすことができます。

3 種類の選択ボックスは、まったく同じ方法で定義できます。「選択ボックスの属性」ダイアログボックスで必要な種類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別に追加するには、次の操作を実行します。

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、隠れるコントロールの前で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、または、[編集] メニューの [コントロール] を使用してコントロールの順序を変えます。 この操作によって、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。

エントリフィールド

選択ボックス内のエントリフィールドは、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、エントリフィールドにデフォルトの選択項目を表すテキストを表示します。

エントリフィールドの最新表示

実行時にエントリフィールドのマスタフィールドの内容を表示するには、選択ボックスを最新表示する必要があります。 選択ボックスは、次の場合に最新表示されます。

- 実行時に選択ボックスの親ウィンドウが作成された場合
- REFRESH-OBJECT ダイアログ関数によって、選択ボックスまたは選択ボックスの親ウィンドウが明示的に最新表示された場合

SHOW-WINDOW 関数を使用して選択ボックスを表示すると、最新表示されません。

エントリフィールドの変更

ユーザが手入力して、または、リストボックスの項目を選択して、エントリフィールドを変更すると、ユーザが追加で操作をしなくても、その結果がすぐにマスタフィールドに反映されます。

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。 その後の処理は、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、項目間に他の項目を挿入したりできます。
- INSERT-LIST-ITEM (または INSERT-MANY-LIST-ITEMS) および DELETE-LIST-ITEM ダイアログ関数を使用して、実行時に項目を挿入したり、削除したりできます。

  リストボックスに項目を追加するこの方法 (および他の方法) については、『サンプルプログラム』の章にある『ダイアログを使用した項目の追加』を参照してください。

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用すると、表示される情報を調整できます。 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、エントリフィールドでスクロールバーを使用すると、ユーザは「スクロール」によって値を設定できます。 Objects サンプルアプリケーションには、スクロールバーのこのような使用方法が示されています。

水平スクロールバーの例を図 5-10 に示します。

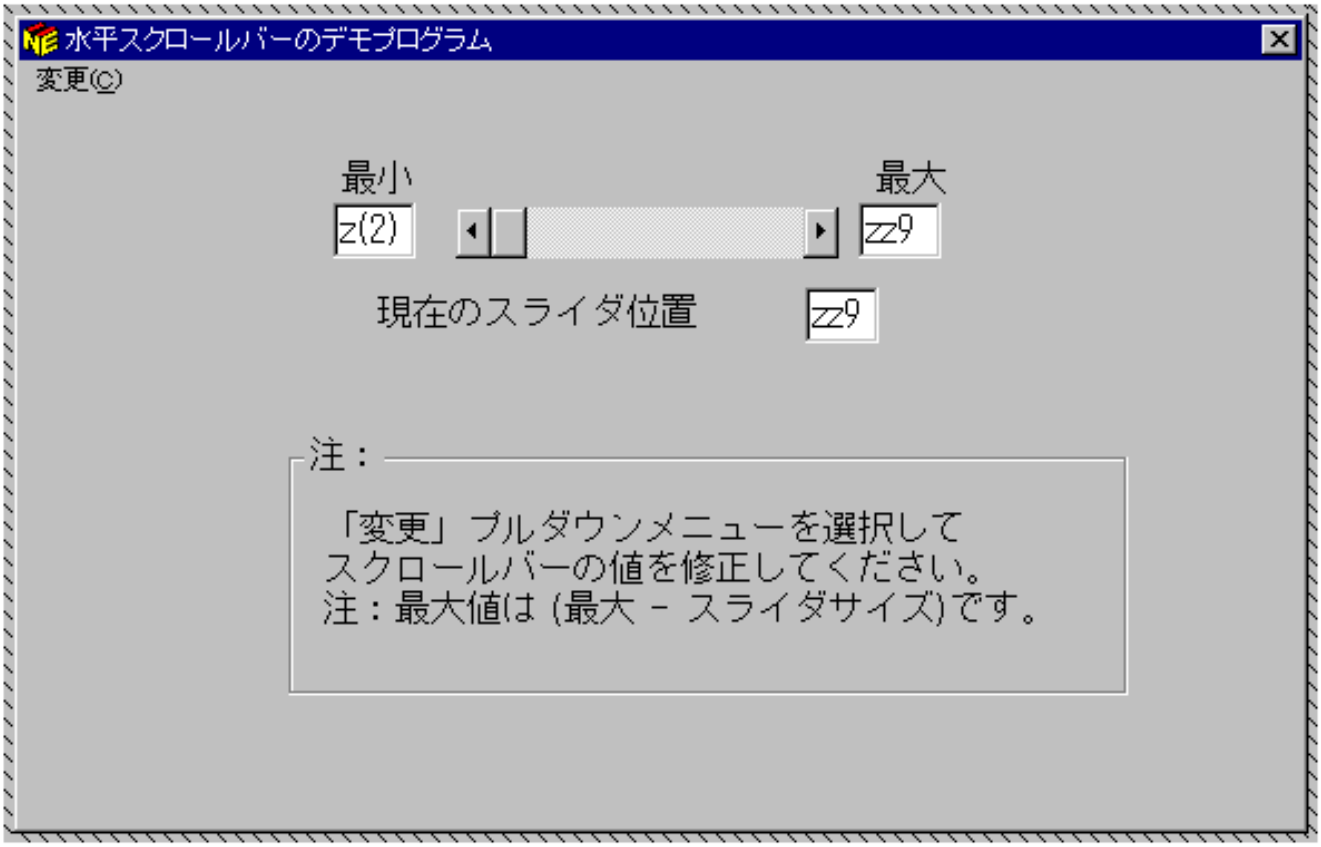

図 5-10： 水平スクロールバー

「スクロールバーの属性」ダイアログボックスでは、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。

コード例については、『チュートリアル - サンプルスクリーンセットの作成』と『チュートリアル - サンプルスクリーンセットの使用方法』の章を参照してください。 スクロールバーの定義については、ヘルプトピックの『オブジェクトと属性』を参照してください。

グループボックス

グループボックスは、コントロールまたはコントロールグループを囲んで見やすくするためのグラフィックフレームです。 グループボックスによって、複数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。

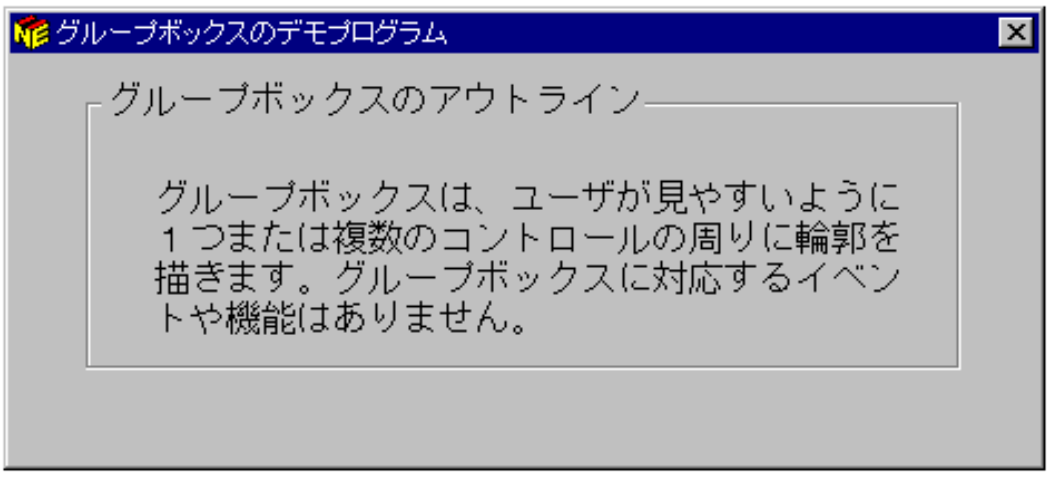

図 5-11： 標準的なグループボックス

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関数を関連付けることはできません。

グループボックスの定義については、ヘルプトピックの『オブジェクトと属性』を参照してください。

タブコントロール

タブコントロールは、関連する情報を見やすく、また、使いやすく編成できます。図 5-12 は、タブコントロールの例を示しています。

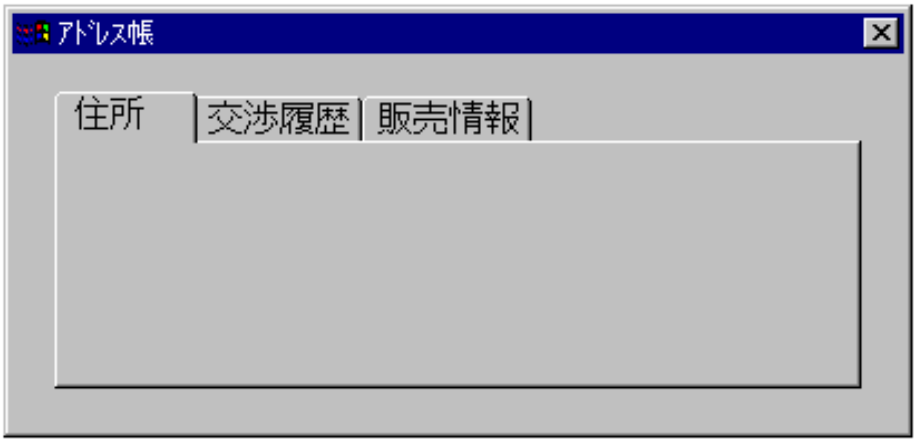

図 5-12： 標準的なタブコントロール

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て 1 つのコントロールを作成します。各ページは、コントロールの一辺 (通常は最上部) に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設定できます。 すべてのタブが 1 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ように表示するか、または、タブがすべて見えるように複数行にわたってタブを折り返して表示します。

タブコントロールで表示する領域は、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き、Dialog System のどのコントロールオブジェクトでも設定できます。

コード例については、『サンプルプログラム』の章を参照してください。

OLE2 コントロール

OLE2 コントロールは、内部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。

たとえば、ペイントブラシアプリケーションで図をデザインして編集し、この図を OLE2 ウィンドウに読み込むことができます。

ユーザコントロール

ユーザコントロールは、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します。

ユーザコントロールのイベントと関数は、そのオブジェクトに関連付けられたコントロールプログラムに実装されます。 通常、コントロールプログラムでは、クラスライブラリを使用して、オブジェクトを作成および維持します。

ユーザコントロールを作成および操作するには、定義時にコントロールプログラムを生成します。 コントロールプログラムを使用する利点は、Dialog System やランタイムソフトウェアに統合できるのみではなく、ユーザがオブジェクトを定義できる点です。

ユーザ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、次のような利点があります。

- 実行時にオブジェクトを自動的に作成して表示できます。
- オブジェクトのタイプに応じたさまざまな解像度の画面をサポートします。また、子ウィンドウおよびガジェットのサイズ変更と移動が可能です。 詳細については、『高度なトピック』の章とヘルプトピックの『ダイナミックなウィンドウのサイズ設定』を参照してください。
- オブジェクトの位置とサイズは、Dialog System の他のオブジェクトと同じ方法で、簡単に定義および修正できます。
- オブジェクトの順番は、Dialog System の他のオブジェクトと同じ方法で、再指定できます。 そのため、実行時にタブの順序を定義できます。 ユーザは、Dialog System の既存のオブジェクトとユーザ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できます。
- すべての標準的な整列機能をユーザコントロールに適用できるので、Dialog System の他のオブジェクトと整列できます。 オブジェクトの整列については、『コントロールの整列』を参照してください。

ユーザコントロールオブジェクトの作成と操作については、『カスタム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。

ActiveX コントロール

ActiveX コントロールは、サードパーティのベンダによって提供され、Dialog System アプリケーションに統合できます。 前述のユーザコントロールの利点は、スクリーンセットで ActiveX コントロールを使用する場合にも適用されます。 ActiveX コントロールを使用した場合は、他に次の利点があります。

- ActiveX コントロールを定義時に描画できます。
- スクリーンセットの初期属性を保存するための、コントロールの属性ページダイアログを表示できます。
- Dialog System プログラミングアシスタントを使用して、実行時のコントロールの表示状態、および、属性、メソッド、イベントなどによるコントロールの動作を操作するコードを記述する場合に便利です。 詳細については、『カスタム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用するコントロールについては、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、ベンダの使用許諾契約書に従う必要があります。

コントロールのグループ化

エントリフィールド、オプションボタン、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は、ユーザは、Tab キーを使用して、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できます。

場合によっては、複数のコントロールが論理グループ (オプションボタンのセットなど) を構成しており、ユーザがグループ内のコントロールすべてに注目するか、または、すべてを無視することがあります。 そのため、連続して Tab キーを押すと、フォーカスが論理グループから次のグループまたはコントロールに移動するよう

にすると便利です。 グループ内のコントロール間を移動するには、カーソルキーを使用できます (この動作は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。

複数のオプションボタンのどれか 1 つのみを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る必要があります。 どれか 1 つのオプションボタンを選択すると、他のオプションボタンの選択が解除されます。

通常、ユーザ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を移動させます。 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、カーソルキーを使用して、操作に適した順序でフォーカスがグループ内のコントロール間を移動するように、各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使用不能になっている場合は、Tab キーでそのグループに移動できません。

各ウィンドウ内または各ダイアログボックス内でコントロールとグループの順序を指定できます。 具体的な手順については、ヘルプトピックの『Dialog System の概要』を参照してください。

コントロールの整列

Dialog System には、グラフィック環境内でグラフィックオブジェクトを移動またはサイズ変更する場合に、グラフィック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、整列ツールバーから実行できます。このツールバーは、[整列] タブページに表示されます。 整列ツールバーについては、ヘルプトピックの『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は、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おおよその位置に設定し、整列ツールバーのボタンをクリックすると、間隔やサイズを調整できます。

ほとんどの整列操作では、整列機能を組み合わせて使用する必要があります。 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は、次のような手順を実行します。

1. プッシュボタンを必要な位置に配置します。
2. [編集] メニューの [領域の選択] を選択するか、または、[領域の選択] ツールバーオプションを使用して、オブジェクトを囲みます。 この操作によって、すべてのオブジェクトがグループとして処理されます。
3. [下揃え] ボタンをクリックします。
4. [オブジェクトの幅を等しくする] ボタンをクリックします。
5. [オブジェクトの高さを等しくする] 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、別の方法でボタンを整列させます。

1. [横に並べて表示] を選択します。

   間隔は設定されません。
2. 次の操作によって、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設定操作を行います。

   この操作では、コントロールの間隔が広がります。

4 つのプッシュボタンが正しい順序で並んでいないとします。 これを正しい順序に並べ替えるには、[並べ替え] ボタンを使用します。 たとえば、[OK]、[挿入]、[キャンセル]、[ヘルプ] という 4 つのプッシュボタンを [ヘルプ]、[キャンセル]、[挿入]、[OK] という順序に並べると仮定します。

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。 操作方法は、次のとおりです。

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も繰り返します。
3. 整列ツールバーの [オブジェクトの並べ替え] をクリックします。

   オブジェクトが [ヘルプ]、[キャンセル]、[挿入]、[OK] の順に並びます。

各ボタンは、選択した順番にグループ内で入れ替わります。 デフォルトの入替え順序は、左から右です。 つまり、この例では、最初に [OK] ボタンが、次に [挿入] が、3 番目に [キャンセル] が、4 番目に [ヘルプ] が入れ替わります。

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は、次のとおりです。

1. 前の説明と同様に、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移動します。
3. [縦に並べて表示] 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、コントロール間の間隔を広げます。

   コントロールがウィンドウ内の適切な位置に配置し、必要に応じて [オブジェクトの再配置] ボタンを使用して順序を変更します。

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 のコントロールオブジェクトの例を表示できます。

ビットマップの使用方法

ここでは、Dialog System インターフェイスでビットマップグラフィックを使用する方法について説明します。 Dialog System では、ユーザインターフェイスで次のオブジェクトについて使用するビットマップを選択できます。

ピットマップオブジェクト	ユーザが選択す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いビットマップ。『ウィンドウオブジェクト』の章にある『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のビットマップ。マウスやシステムのさまざまな状態を表します。 『チュートリアル - マウスポインタのビットマップの変更』の章を参照してください。
プッシュボタン	ボタンを装飾したり、さまざまな状態のボタンを表すためのビットマップ。

Dialog System には、Dialog System で使用するオブジェクトのビットマップと、作業環境の標準ビットマップが用意されています。 ユーザが、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、および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、ヘルプトピックの『ビットマップ、アイコン、マウスポインタ』を参照してください。

ビットマップの定義

ビットマップは、ウィンドウ内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。 ビットマップは、主に装飾用に使用します。ビットマップによって、ボタンに注意を引き付けたり、ウィンドウ内やダイアログボックス内にロゴなどの項目を配置したりします。 ビットマップをクリックすると、BITMAP-EVENT が発生し、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。 この操作によって、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できます。 詳細については、ヘルプトピックの『コントロールの使い方』と『ビットマップ、アイコン、マウスポインタ』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配置するには、次の操作を実行します。

1. メインウィンドウの [オブジェクト] メニューから [ビットマップ] を選択します。
   または、
   [オブジェクト] ツールバーのビットマップアイコンをクリックします。

   図 5-13 に示すとおり、「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。

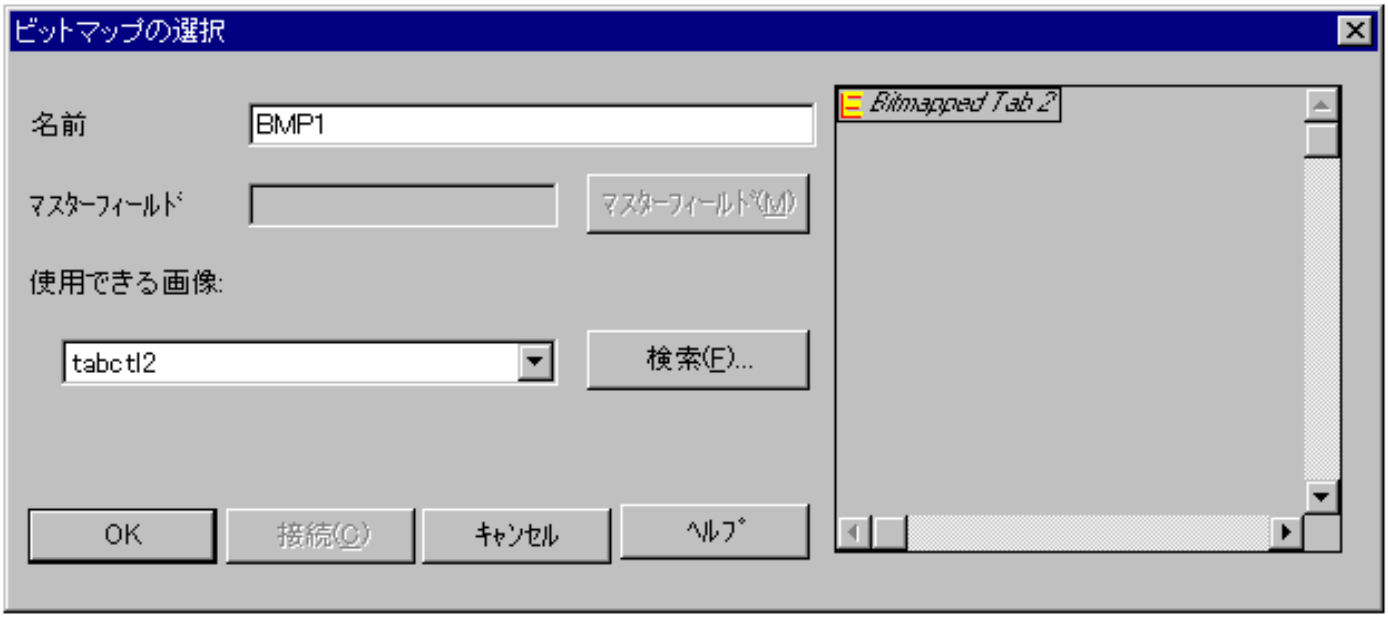

図 5-13： 「ビットマップの選択」ダイアログボックス

2. 「名前」フィールドに名前を入力する。
   または、
   「使用できる画像」フィールドからビットマップを選択します。

3. 選択した後で [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られ、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. マウスを使用してビットマップを配置してクリックします。

   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] を使用すると、リスト内のビットマップの後に、他のコントロールを追加できます。 [コントロール] を選択しても、ウィンドウが非表示または再表示されるまで、表示には影響がありません。

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の表示状態が用意されています。 ビットマップ化されたプッシュボタンを使用して、プッシュボタンの表示状態を変更できます。

たとえば、プッシュボタンをグラフィックシンボルにしたり、プッシュボタンの表示状態を状況によって変えたりする場合があります。

ビットマップをボタンと関連付けるには、次の操作を実行します。

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「プッシュボタンの属性」ダイアログボックスが表示されます。

図 5-14： 「プッシュボタンの属性」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、使用可能なオプションが表示されます。

図 5-15： 「プッシュボタンの属性」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」オプションを使用できるようになります。 オプションには、3 つの選択項目があります。

- 「通常」
- 「押下」

- 「使用禁止」

4. 「ボタン状態ビットマップ」の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が、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場合は、ユーザは独自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、ネイティブの Win32 ツールバーを実装できます。

第 6 章 : ダイアログの使用方法

ここまでは、スクリーンセットの視覚的な面について説明してきました。 以後では、ダイアログを使用してインターフェイスを実行する方法について説明します。 ここでは、ダイアログの強力な機能について説明します。

- ダイアログとそのレベル
- イベント、関数、および手続き
- 特殊レジスタ
- ダイアログのイベントと関数の重要な例

ダイアログの機能

ダイアログは、イベントをトラップして、発生したイベントに対応するためのコードです。このコードは、画面には表示されません。簡単なイベントと応答の例は、「ユーザが [次のウィンドウ] というボタンを押した場合に、新しいウィンドウにフォーカスを設定する」ような場合です。 この操作を実行するコードは、ダイアログを使用して作成します。このダイアログにアプリケーションプログラムがアクセスします。

ダイアログの構成要素は、次のとおりです。

- 発生する可能性があり、応答する必要があるイベントの名前
- 各イベントが発生したときに実行する命令 (関数と呼ばれる)
- オプション。呼び出し可能な関数のリストを含む手続き (サブルーチンと呼ばれる)
- オプション。ダイアログを読みやすくするための注釈

イベントに対するダイアログの末尾には、イベントまたは手続きの名前を指定します。 たとえば、次のダイアログがウィンドウに設定されていると想定します。

```
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
```

関数 SET-EXIT-FLAG と RETC は、CLOSED-WINDOW イベントに関連付けられます。 つまり RETC は、CLOSED-WINDOW イベントに関連付けられた最後の関数です。 関数 MOVE 1 ACTION、RETC、および SET-FOCUS EMPLOYEE-WIN は、F1 イベントに関連付けられます。

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、注釈を使用できます。 注釈を指定すると、次のような利点があります。

- 作成したダイアログで、混乱する可能性のある文を、わかりやすく説明できます。
- 状況に応じて、実行可能なダイアログ文の行を実行しないようにできます。
- 読みやすくするために、イベントと手続きを分けることができます。

注釈行の先頭には、アスタリスク (*) を指定します。

次のサンプルダイアログは、注釈を使用した例を示しています。

```
*********************
 SCREENSET-INITIALIZED
*
* 各ビットマップのハンドルを保存する
*
     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
*********************
```

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、またはインターフェイス全般に対して設定します。 これら 3 つのクラスのオブジェクトに設定するダイアログには、基本的な相違はありません。 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には、次のクラスがあります。

- 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

コントロールダイアログ

このダイアログは、テキスト、ユーザコントロール、AcriveX コントロール、およびグループボックスを除く、すべてのコントロールに設定できます。

たとえば、次のダイアログは、プッシュボタンに設定してウィンドウのタイトルを変更するためのダイアログです。

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

ウィンドウダイアログ

このダイアログは、すべてのウィンドウ、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できます。

たとえば、次のダイアログは、ウィンドウに設定して、ユーザが選択した [閉じる] オプションを処理するためのダイアログです。

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

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設定できます。

たとえば、次のダイアログは、リストボックスの最初の数項目を初期化するためのダイアログです。

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

ダイアログ文の配置位置

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配置します。
- 手続きが 1 つのオブジェクトのみに関係する場合は、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直接設定できます。

ダイアログの構造を工夫すると、効率的に実行できます。また、維持とデバッグもより簡単になります。 ダイアログを記述する場合は、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ことをお勧めします。

ダイアログの種類

ダイアログには、次の 3 つの種類があります。

- イベント
- 手続き名
- 関数

イベント

スクリーンセットに渡すイベントは、明確に定義する必要があります。 定義されていないイベントは無視されます。 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が 255 に制限されています。

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発生した。
- ユーザがスクロールバーのスライダを動かした。

Dialog System によるイベントダイアログの検索方法

Dialog System では、イベントが発生すると、コントロールダイアログ内、そのコントロールを含むウィンドウのダイアログ内、および、グローバル (スクリーンセット) ダイアログ内の順で、イベントが検索されます。

注：手続きにも同じ規則が適用されます。 コントロールのダイアログで手続き名を呼び出した場合に、その手続き名が検出されないと、ウィンドウダイアログ内、グローバルダイアログ内の順で手続きが検索されます。

これらの 3 つのダイアログにイベントが一覧表示されていない場合は、コントロールレベル、ウィンドウレベル、グローバルレベルの順に ANY-OTHER-EVENT が検索されます。

ANY-OTHER-EVENT が検出されない場合は、イベントに対する動作は実行されません。

Dialog System でイベントが検出されると、そのイベントに設定された関数が 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限り、イベントに対して一覧表示された関数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中断されます。 Dialog System では、一度に 1 つのイベントしか処理できないので、後のイベントがキューに置かれ、必要に応じて後で処理されます。 Dialog System では、各ダイアログテーブルのイベントの数が 255 に制限されています。

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、イベントの発生を認識して処理します。 通常は、呼び出し側プログラムが Panels V2 からイベントの情報を受け取ることはありません。 ただし、dssysinf.cpy コピーファイル内の ds-event-block によって、これらのイベントを呼び出し側プログラムに戻すことができます。 このファイルを呼び出し側プログラムの作業場所節 (Working-Storage Section) にコピーして、Dialog System の呼び出し内に指定する必要があります。 この操作を実行するのは、画面イベントが Dialog System で明確に定義されていないような場合です。

イベントの詳細については、ヘルプトピックの『ダイアログ文：イベント』を参照してください。

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。

関数

関数は、Dialog System で処理を行うための命令です。 関数は、次のような場合に動作します。

- 関数が一覧表示されているイベントが発生した場合
- 関数が一覧表示されている手続きが実行された場合

各イベントまたは各手続きに対して入力できる関数の数は、関数とイベントのバッファのサイズ (2 KB) と、ダイアログテーブルの最大サイズ (64 KB) によって制限されます。

最大 2048 個の関数を入力できますが、実際の最大数は関数によって異なります。平均的な長さの関数の最大数は、通常は 341 個です。これより多くの関数が必要な場合は、特別な関数を含む手続きを呼び出すことができます。このような手続きの例は、次のとおりです。

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

function-340 ～ function-344 は、ヘルプに一覧表示されている Dialog System の関数です。

Dialog System には、関数が多数あります。 スクリーンセットの移動に固有の関数 (SET-FOCUS、INVOKE-MESSAGE-BOX など) や 他のプログラム言語の命令に似た関数 (データ項目間のデータを転記する MOVE) などがあります。

各関数の詳細については、ヘルプトピックの『ダイアログ文：関数』を参照してください。

手続き

手続きとは、任意の名前が付いた関数セットで、サブルーチンとも呼ばれます。手続きは、イベントと同様に見なすことができます。つまり、EXECUTE-PROCEDURE または BRANCH-TO-PROCEDURE 関数が実行されると、イベントが「発生」します。

たとえば、次のダイアログは、手続きを使用した「ループ」のコーディング方法を示してい

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
   ...
```

特殊レジスタ

Dialog System には、次のように、パラメータとして使用できるレジスタと変数があります。

$REGISTER	汎用の内部レジスタ。 配列への指標として使用したり、数値を一時的に格納するために使用したりできます。
$NULL	デフォルト値を与えたり、実際のパラメータが必要ないときに他のパラメータ間を区切ったりするためのパラメータ。
$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 関数とともに使用できます。 F3 CLEAR-OBJECT $CONTROL ユーザが F3 キーを押すと、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 関数とともに使用して、現在のウィンドウをクリアできます。 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 イベントとともに使用できます。 妥当性検査エラーが検出された場合は、VAL-ERROR イベントが起動され、エラーが発生し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、エラーが発生したフィールドにユーザがフォーカスを設定し、データを修正できます。 このような $EVENT-DATA の使用方法については、付録『フォントと色の指定』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、ヘルプトピックの『ダイアログ文：イベント』を参照してください。
$INSTANCE	特定のタブコントロールページを参照するためのインスタンス番号。

ダイアログの重要なイベントと関数

Dialog System には、スクリーンセットの動作を制御するためのイベントと関数が豊富にあります。 ここでは、これらを起動する方法、 メニューから選択する方法、 プログラムに戻る方法、およびスクリーンセットに戻ると何が起こるかについて説明します。

以後では、基本的な関数とその関数に適したダイアログのサンプルを示します。

スクリーンセットの初期化

スクリーンセットを初めて作成する場合は、スクリーンセットのデフォルトの状態 (オプションボタンの状態、データ集団のサイズ、デフォルト値、使用可能にするメニューオプションなど) を設定することがあります。この操作を実行するには、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
```

1 行目：

```
SCREENSET-INITIALIZED
```

このイベントは、スクリーンセットが初めて入力されたとき (または呼び出し側プログラムが Dialog System に対して新しいスクリーンセットを要求した場合) に発生します。

2 ～ 3 行目：

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

SET-BUTTON-STATE では、チェックボックス、プッシュボタン、または、オプションボタンの状態を設定します。 これら 2 つの文によって、OK-BUTTON をオンに、CANCEL-BUTTON をオフに設定します。

4 ～ 5 行目：

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

ENABLE-OBJECT 関数によって [名前を付けて保存] (SAVE-AS) プッシュボタンを使用できるようにします。また、DISABLE-OBJECT 関数によって、[上書き保存] (SAVE) プッシュボタンを使用できなくします。 たとえば、ユーザが [名前を付けて保存] オプションを使用して新しいファイルにファイル名を指定する場合などに、これらの関数を使用できます。

6 行目：

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

SET-DATA-GROUP-SIZE 関数では、データ集団の内部の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。

7 行目：

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

この文では、エントリフィールドのデフォルト値を設定します。

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すると、ダイアログを使用してウィンドウを作成および初期化できます。

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。

このダイアログ文では、ウィンドウを初期化して表示できるようにします。 画面上の変化はありませんが、ウィンドウがバックグラウンドで作成されます。 作成されたウィンドウは、SHOW-WINDOW ダイアログ文を使用して表示できます。

ウィンドウの初期化は、あまり複雑な処理ではありませんが、若干の時間がかかります。 初期化の影響を避けるためには、ウィンドウが必要になる前に作成し、表示する準備ができたら、SHOW-WINDOW によって表示します。

最初のウィンドウの表示

スクリーンセットの起動時に最初に実行する関数の 1 つが、アプリケーションのメインウインドウとダイアログボックスを表示する関数です。 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。

[スクリーンセット] メニューの [最初のウィンドウ] を選択すると、定義ソフトウェアによって、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は、ダイアログを追加する必要はありません。

ただし、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りする場合があります。 たとえば、定義時にメインウインドウが不明な場合は、実行時に選択します。 このような場合は、SET-FOCUS 関数、SHOW-WINDOW 関数、または、SHOW-WINDOW 関数を使用します。SHOW-WINDOW を使用する場合は、次のように入力して、まず、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

その後で、SHOW-WINDOW 関数を入力します。 SET-FIRST-WINDOW 関数を使用する場合は、この操作は不要です。

ウィンドウの表示

SHOW-WINDOW 関数を使用すると、ウィンドウを表示できます。ただし、ウィンドウにフォーカスが設定されません (オブジェクトをキーボードやマウスで操作すると、オブジェクトに入力フォーカス (またはフォーカス) が設定されます)。

ウィンドウがまだ作成されていない場合は、SHOW-WINDOW 関数によってウィンドウが自動的に作成されます。

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、ユーザが F3 キーを押したときに発生するイベントです。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は、その親ウィンドウも表示されます。

ウィンドウの非表示

UNSHOW-WINDOW 関数では、ウィンドウ、すべての子ウィンドウ、およびコントロールを非表示にします。 ウィンドウは、存在しなくなったわけではないので、再作成する必要はありません。

UNSHOW-WINDOW を使用すると、クライアント領域を整理できます。

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、および、コントロールが、非表示にす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 番目のパラメータには、フォーカスを受け取るウィンドウを指定します。

たとえば、次のように記述します。

```
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。

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は、『スクリーンセットの使用方法』の章を参照してください。

デフォルトの親ウィンドウの変更

SET-DESKTOP-WINDOW 関数では、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として指定します。

たとえば、WIN1 (および WIN1-HAND ハンドル)、WIN2、および 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。 この場合に次の関数を実行す

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。

この関数は、現在のスクリーンセットから関数を選択したときに、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などに使用できます。 この場合に、新しいスクリーンセットの最初のウィンドウを最初のスクリーンセットの現在のウィンドウの子にすると仮定します。

この操作を実行するには、最初のスクリーンセットで次の関数を使用します。

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

次に、プログラム内で 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使用方法については、『複数のスクリーンセット』の章を参照してください。

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、次の関数を使用します。

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

注：作成済みのウィンドウは変更できません。 この関数は、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適用されます。

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。 ただし、ウィンドウを削除する場合は、そのウィンドウのインスタンス、すべての子ウィンドウ、そのウィンドウに関連付けられているすべてのコントロールが削除されま

す。 このウィンドウの他のインスタンスを参照する場合は、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のように、そのセッション内の後の処理でウィンドウが不必要になる場合を除き、DELETE-WINDOW は使用しないで UNSHOW-WINDOW を使用してください。

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は、『スクリーンセットの使用方法』の章を参照してください。

ウィンドウへのフォーカスの設定

SET-FOCUS ダイアログでは、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。

この関数では、必要に応じて、フォーカスの設定前にウィンドウが作成および表示されます。 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、フォーカスを受け取るウィンドウです。

ウィンドウの移動

MOVE-WINDOW 関数では、ウィンドウを現在の位置から新しい位置へ移動して、画面上でそのウィンドウを再編成できます。 たとえば、いくつかの値を監視および表示するために作成したウィンドウは、他のウィンドウが表示されたときに非表示にする関数ではなく、MOVE-WINDOW を使用して、移動できます。

構文例は、次のとおりです。

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

$WINDOW は現在のウィンドウを示します。2 は「右」方向、5 はウィンドウを移動するシステム単位数を示します。 これらのパラメータについては、ヘルプの MOVE-WINDOW に関する説明を参照してください。

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は、1 つのウィンドウに対して SET-OBJECT-LABEL 関数を使用して、ウィンドウのタイトルのみを変更できます。構文例は、次のとおりです。

```
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は、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。 ただし、ウィンドウが表示されている場合は、変更されたタイトルがすぐに表示されます。

ウィンドウを閉じる

メニューには、[閉じる] というオプションがあります。 [閉じる] 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。このオプションの選択時に、別の操作 (アプリケーションの終了など) を実行することもできます。この操作には、CLOSE-WINDOW イベントを使用します。 次にダイアログの例を示します。

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

このダイアログ文では、「閉じる」イベントを検出し、ユーザがユーザインターフェイスの終了を選択したことを示すフラグを設定して、呼び出し側プログラムに返します。 その後で、プログラムでファイルを閉じ、アプリケーションの終了に必要なその他プロセスを実行できます。

ボタンを押す

BUTTON-SELECTED 関数は、プッシュボタン、チェックボックス、または、オプション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場合

やチェックボックスを選択した場合に発生します。

たとえば、次のように記述します。

```
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 イベントが発生すると、そのイベントに関連付けられているすべての関数が実行されます。

ボタンの状態の設定と取得

SET-BUTTON-STATE および GET-BUTTON-STATE 関数は、ボタンとチェックボックスの状態を制御する場合に使用します。 スクリーンセットを初めて入力し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化』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取得することもできます。 たとえば、次のダイアログでは、呼び出し側プログラムで実行された操作に従ってオプション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
    ...
```

マスターフィールドに 1 または 0 の値を転記すると、チェックボックスの状態を制御できます。 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は、check-credit に 1 または 0 を転記すると、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に状態の変化を表示するには、そのオブジェクトを更新する必要があります。

メニューバーダイアログ

ユーザがプルダウンメニューの操作を選択した場合に、それに応答するダイアログを定義することもできます。 ただし、ダイアログを操作項目に追加する前に、操作に名前を付ける必要があります。 操作への命名については、ヘルプトピックの『オブジェクトと属性』を参照してください。

次のダイアログは、Saledata サンプルアプリケーションの一部です。この中の INSERT-CHOICE、CHANGE-CHOICE、および、DELETE-CHOICE は、メニューバーの操作に関連付けられた名前です。ダイアログ内の選択項目の前に指定された「@」は、その項目がメニュー項目であることを表しています。

```
@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
```

ユーザがメニューバーの [変更] を選択すると、@CHANGE-CHOICE イベントが起動され、このイベントのすべての関数が実行されます。

項目の有効化と無効化

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、新規ファイルであり、まだ名前が付けられていないため、[上書き保存] よりも [名前を付けて保存] (ファイル名の指定が必要) が強制的に選択されるようにします。 このようなダイアログのコード例は、次のとおりです。

```
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    ...
```

2 行目：

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

XIF は、条件関数です。 FILE-SAVE-FLAG は、データブロックのデータ項目です。 値 0 は、ファイルが保存されていないことを表します。 DISABLE-SAVE は、実行する手続きです。 この文では、ファイルがまだ保存されていない場合は、DISABLE-SAVE 手続きを実行します。

4 行目：

```
    DISABLE-SAVE
```

実行する手続きを開始します。

5 行目：

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

この文では、[上書き保存] 項目を使用不可にします。

6 行目：

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

この文では、[名前を付けて保存] を選択可能にします。

項目の選択

メニューバー項目は、ユーザが項目を選択したときにの動作を定義したものです。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]、および [貼り付け] オプションのコード例です。

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

注：カラム 1 の「@」は、メニュー項目であることを示します。

メニューバー項目を操作するためのダイアログ (メニューバー項目を使用可能および使用不能にする方法を含む) については、『ウィンドウオブジェクト』の章にある『メニューバー』を参照してください。

入力の妥当性検査

VALIDATE 関数を使用すると、ウィンドウに設定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の妥当性検査ができます。 たとえば、次のように記述します。

VALIDATE SALARY-RANGE-EF

SALARY-RANGE-EF エントリフィールドのみを妥当性検査します。

次の文

VALIDATE SALARY-DETAILS-WIN

では、SALARY-DETAILS-WIN ウィンドウのすべてのフィールドが妥当性検査されます。

エラーが検出された場合は、VAL-ERROR イベントが発生します。 次のダイアログは、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ド例です。

```
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
```

構文の詳細については、『サンプルプログラム』の章にある『エントリフィールドの妥当性検査』を参照してください。

手続きの使用方法

一般に、手続きは共通の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方法は、少なくとも 2 種類あります ([キャンセル] ボタンをクリックするか、または、Esc キーを押す)。 キャンセル機能をコーディングする場合に最もよい方法は、[キャンセル] ボタンが選択された場合も、Esc 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、手続きを起動する方法が 2 種類あります (手続きへの分岐 (COBOL の GOTO 文と同様) と手続きの実行 (COBOL の PERFORM 文と同様))。 次のダイアログは、これらの相違を表しています。

```
F1
   BRANCH-TO-PROCEDURE CLEAR-OK
   RETC

F2
   EXECUTE-PROCEDURE CLEAR-OK
   RETC
```

最初の文 (ユーザが F1 キーを選択した場合) では、CLEAR-OK 手続きに制御が渡されます。 次の操作は、CLEAR-OK 内の関数によって異なります。 RETC 関数は実行されません。

ユーザが F2 キーを選択した場合は、CLEAR-OK 内の関数が実行された後で、EXECUTE-PROCEDURE 文の次の文である RETC 関数に制御が戻ります。

Dialog System には、いくつかの分岐関数も用意されています。 使用する関数は、手続きに分岐するか、または、手続きを実行するかによって異なります。

たとえば、IF= 関数では、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= 関数では、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。

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

呼び出し側プログラムに制御を戻す方法

スクリーンセットの実行中にプログラムに戻る必要がある場合があります。 たとえば、データベースから別のデータを取得する場合、ファイルを更新する場合、複雑な妥当性検査を行う場合などです。RETC 関数を使用すると、呼び出し側プログラムに制御を戻すことができます。

たとえば、次のダイアログは、ファイル削除関数を実行するために呼び出し側プログラムに戻ります。

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

呼び出し側プログラムからの制御の回復

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できる動作の一部を次に示します。

- ウィンドウを最新表示する (プログラムで関連するデータ項目が変更され、その結果をユーザに対して表示する場合など)。
- スクリーンセットの状態をリセットする (プログラムでファイルを保存した後に、[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する場合など)。
- オブジェクトを最新表示する (データベース内に表示する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 (プログラムの中で複雑な妥当性検査を行う必要があり、その結果を表示する場合など)。

次のダイアログでは、呼び出し側プログラムから制御が戻った後に、SALES-LIST リストボックスを最新表示して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
```

Windows オペレーティングシステムでトラップされたイベント

Dialog System に制御が移る前に、Windows OS によってトラップされるイベントがあります。

たとえば、標準の Windows システムメニューには、いくつかのアクセラレータキーがあり、メニュー上の関数にすばやくアクセスできます。 たとえば、Alt+F4 アクセラレータキーを押すと、アクティブなウィンドウとそれに関連付けられたすべてのウィンドウが画面から消去されます。 Alt+F4 キーを押すと発生するイベント (AF4) のダイアログを定義する

と、そのイベントは、Windows によってトラップされ、Dialog System には到達しません。

そのため、Alt+F4 キーを押す操作に対して定義された次のダイアログは、Dialog System で検出されないので、実行されません。

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

サンプルプログラム

ウインドウオブジェクト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含まれています。 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、および、エラーファイルで構成されています。 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で表示するオブジェクトを [オブジェクト] メニューから選択します。

プログラムのコンパイル、アニメート、および実行については、COBOL システムのヘルプを参照してください。

実行時にメニューバーの項目から動的に操作する場合は、例として、Dialog System のデモンストレーションディレクトリの Dynmenu デモンストレーションを参照してください。

サンプルダイアログ

ここで説明したダイアログよりも複雑なダイアログもあります。 次のダイアログは、Dialog System の定義機能の一部です (Dialog System 自体も 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設定し、マウスの構成を再定義するためのダイアログです。

```
...
 @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
* ユーザ定義の操作
    EXECUTE-PROCEDURE ENABLE-CHOICES
```

```
    SET-BUTTON-STATE USER-DEFINED-RB 1
    SET-FOCUS USER-DEFINED-RB
* DS2.1 マウスの動作
  DS21
    EXECUTE-PROCEDURE DISABLE-CHOICES
    SET-BUTTON-STATE DS21-RB 1
    SET-FOCUS DS21-RB
* CUA の動作
  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 内で処理されることに注意してください。 COBOL プログラムには制御が戻りません。

ダイアログの使用方法 | Windows GUI アプリケーションの新規作成ウィザード

第 7 章 : スクリーンセットの使用方法

ここまでは、データ定義とアプリケーションのユーザインターフェイスを作成する方法について説明しました。 ここでは、次の内容について説明します。

- 呼び出しインターフェイス、および、呼び出しインターフェイスによってスクリーンセットと COBOL プログラムとの通信を可能にする方法
- インターフェイ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の考察事項

呼び出しインターフェイス

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方法
2. 呼び出しインターフェイスの構造と使用方法
3. COBOL アプリケーションプログラムの作成方法
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
5. アプリケーションのパッケージ化

データブロックのコピーファイルの生成

データブロックのコピーファイルには、実行時に呼び出し側プログラムから Dialog System に渡されるデータブロックの定義が記述されています。 コピーファイルは、呼び出し側プログラムに指定する必要があります。 このコピーファイルには、バージョン確認情報も記述されています。

スクリーンセットからコピーファイルを生成するには、次の操作を実行します。

- 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構成オプションを選択します。

  定義ソフトウェアを構成して、さまざまな方法でコピーファイルを作成できます。 必要なコピーファイルの種類は、Micro Focus COBOL と ANSI COBOL 標準のどちらを使用して呼び出し側プログラムを記述しているかによって異なります。 コピーファイルのオプションを構成する方法については、ヘルプトピックの『Dialog System の構成』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COPYファイル] を選択します。

  表示されたダイアログボックスでコピーファイルのパスとファイル名を選択します。 Dialog System のデフォルトのコピーファイル名は、screenset-name.cpb です。
- 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開き、生成中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。

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場合は、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 文字を超える場合は、30 文字に切り捨てられます。

  データ項目名が切り捨てられた場合は、項目名全体を引用符で囲んで注釈としてコピーファイルの項目の横に表示されます。
- 「スクリーンセット ID の接頭辞が付けられたフィールド」構成オプションを選択している場合は、データブロックのすべてのデータ名の先頭にスクリーンセット ID が設定されます。 データ項目名は、この接頭辞を含めて 30 文字以内にする必要があります。

  このオプションは、次のように変更できます。

  - 特定のスクリーンセットを変更するには、[オプション] メニューから [構成]、[スクリーンセット] の順に選択して、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を変更するには、ds.cfg 構成ファイルを編集するか、[オプション] メニューから [構成]、[デフォルト] の順に選択します。

- 名前が切り捨てられた結果、複数のデータ項目名が重複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。

  このエラーを修正するには、「データの定義」ウィンドウでデータ項目名を変更し、コピーファイルを再生成します。

注：次のようにコマンド行を指定すると、データブロックのコピーファイルを生成できます。

dswin/g screenset-name

このように指定すると、コピーファイルが生成され、終了します。

コピーファイルの生成については、ヘルプトピックの『Dialog System の概要』、『呼び出しインターフェイス』、および『データ定義と妥当性検査』を参照してください。

呼び出しインターフェイスの構造

ここでは、最も簡単に Dialog System を呼び出す方法について説明します。 Dialog System に不慣れな場合でも、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できます。

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

- 制御ブロック (制御ブロックコピーファイルの DS-CONTROL-BLOCK に相当) は、呼び出し側プログラムと Dialog System の間で制御データを受け渡すために使用されます。

- データブロック (データブロックコピーファイルの DATA-BLOCK に相当) は、呼び出し側プログラムと Dialog System の間でユーザデータを受け渡すために使用されます。

スクリーンセットを使用するには、Dialog System の制御ブロックとデータブロックを使用して、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 の基本的な呼び出しは、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、最初に現在の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。

スクリーンセットの構成ダイアログボックスの「スクリーンセット識別子を先頭に付ける」を選択した状態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が接頭辞として設定されます。 そのため、この接頭辞を Dialog System に対する呼び出しで指定する必要があります。たとえば、DialogSystem\demo インストールディレクトリ内の entries.cbl サンプルプログラムでは、「entry」というスクリーンセット ID をもつスクリーンセットが使用されているので、次のように呼び出しています。

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

Dialog System を最初に呼び出した後で、呼び出しが失敗した場合に使用する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 プログラム内で Dialog System を実際に呼び出す位置は、重要ではありません。 必要な場合にいつでもプログラムから Dialog System を呼び出せるように、Dialog System の呼び出しを別のルーチンとして作成することをお勧めします。

制御ブロックに指定してスクリーンセットの実行方法を制御するためのオプションについては、ヘルプトピックの『呼び出しインターフェイス』を参照してください。

呼び出しインターフェイスをさらに高度に使用する方法については、『高度なトピック』と『複数のスクリーンセット』の章を参照してください。

スクリーンセットの操作の制御

呼び出し側プログラムでスクリーンセットの処理を制御する方法については、スクリーンセットと呼び出し側プログラムを設計する際に詳細に検討する必要があります。 その場合は、次の点に注意してください。

- 関数の分割方法、および、関数を複数のスクリーンセットにまとめるかどうか。
- セキュリティ上の理由から、データへのアクセス権に応じてユーザをグループ化し、そのユーザグループごとにスクリーンセットを作成するかどうか。
- 1 名のユーザが、データの表示、比較、または、編集を同時に実行できるように、スクリーンセットのインスタンスを複数使用できるようにするかどうか。

Dialog System の呼び出しインターフェイスを使用して画面を制御するための基本的な情報については、ヘルプトピックの『呼び出しインターフェイス』を参照してください。

複数のスクリーンセットの使用

複数のスクリーンセットを使用するには、スクリーンセットスタックにプッシュしてポップします。 原則として、FIFO (先入れ後出し) の方式をとっています。 スクリーンセットのプッシュとポップは、次の場合に便利です。

- 特定の機能に使用するスクリーンセットが不用になった場合に、このスクリーンセットを画面に表示させないことができます。
- 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してポップ (または使用) できます。
- 使用していないウィンドウや入力フォーカスがないウィンドウを画面に表示しないで、画面を見やすくできます。

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するための前提条件はありません。 すべてのスクリーンセットやスクリーンセットのオカレンスをプッシュしてポップできます。 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くには、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 に格納します。 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。 スクリーンセットをスクリーンセットスタックからポップするには、次のどちらかを使用します。

- ds-quit-set は、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 は、既存のスクリーンセットを閉じずに、指定し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。

詳細については、『複数のスクリーンセット』の章を参照してください。

同一のスクリーンセットから複数のインスタンスを使用する方法

Dialog System では、同じスクリーンセットの複数のインスタンスを、呼び出し側プログラムで使用できます。 この機能は、各集団項目が同じ形式のデータ集団を操作するときに役立ちます。 同じスクリーンセットのインスタンスを複数使用すると、必要に応じて、1 つのスクリーンセットで集団項目を表示、比較、または、更新できます。

同じスクリーンセットのインスタンスを複数使用する場合は、プログラムで次の処理を実行する必要があります。

- スクリーンセットのインスタンスの数をトレースする。
- スクリーンセットのインスタンスについて適切なデータブロックを Dsgrun に渡す。

複数のインスタンスを使用している場合に、最初にスクリーンセットが「N」または「S」の呼び出しで起動されると、インスタンスの値が制御ブロックの ds-screenset-instance フィールドに割り当てられて配置されます。

インスタンスの値は、特定のスクリーンセットのインスタンスに一意なものです。 インスタンスの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の値をトレースする必要があります。

dssysinf.cpy ファイル内の ds-event-screenset-id と ds-event-screenset-instance-no を調べると、アクティブなインスタンスをトレースできます。このインスタンス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。

dssysinf.cpy の詳細については、『Panels V2 の使用方法』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をロードするには、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用してスクリーンセット Animator からスクリーンセットを実行する場合は、スクリーンセットのインスタンスを複数使用できません。 アプリケーションから Dsgrun を呼び出す場合は、複数のインスタンスがサポートされます。

詳細については、『複数のスクリーンセット』の章を参照してください。

COBOL アプリケーションプログラムの作成

データブロック コピーファイルには、ユーザデータのみでなく、Dialog System がスクリーンセットから呼び出されたときに確認するスクリーンセットのバージョン番号も含まれます。 バージョン番号を確認するには、Dialog System のランタイムシステムを呼び出す前に、呼び出し側プログラムでこれらのバージョン番号を制御ブロック内のデータ項目にコピーする必要があります。

制御ブロック

制御ブロックは、呼び出し側プログラムと Dialog System の間で制御情報やデータを受け渡すために使用されます。

制御ブロックは、次の要素で構成されます。

- データブロック、制御ブロック、および、スクリーンセットのバージョンを確認するバージョン確認項目
- Dialog System に関数の実行を命令する入力フィールド。

  スクリーンセットを制御するための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、次の 3 つのバージョンの制御ブロックコピーファイルが組み込まれています。

- ds-cntrl.ans

  ANSI-85 準拠 COBOL で使用します。
- ds-cntrl.mf

  Micro Focus 準拠 COBOL (COMP-5) で使用します。

- dscntrlx.mf

  Micro Focus 準拠 COBOL (COMP-X) で使用します。 3 を VERSION-NO ではなく DS-VERSION-NO へ転記して使用します。

呼び出し側プログラムを作成する場合は、次の文を使用して、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 COBOL を使用する場合は、ds-cntrl.ans コピーファイルを使用する必要があります。

データブロックフィールドの情報と制御ブロックは、Dialog System と呼び出し側プログラムの間で制御が移るときに受け渡されます。

また、スクリーンセットの名前や Dialog System の動作を制御する情報が制御ブロックに指定されていることを確認する必要があります。

Dialog System では、プログラムがユーザの操作を指定するのではなく、ユーザがプログラムで次に実行する機能を決めることができます。 Dialog System からデータブロックに返されたフラグには、ユーザの操作に対応した値が設定され、プログラムの次の動作を指定します。

プログラムは、次のようにさまざまな方法で対応できます。

- 保存された情報を修正します。
- データベースから追加情報を検索します。
- 結果またはエラーメッセージを表示します。
- アプリケーションの使用を支援します。

  簡単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全体を処理できます。 ヘルプを処理する別の方法としては、ヘルプフラグが設定されたときにヘルプを表示するためのコードをプログラムに記述する方法もあります。

- ユーザが入力した情報を妥当性検査します。
- ユーザからの追加入力を要求します。

  レコードを保存または消去した後に、Dialog System に戻ります。

(『チュートリアル - サンプルスクリーンセットの作成』の章で作成される) サンプルスクリーンセットを使用する COBOL プログラム entries.cbl については、『チュートリアル - サンプルスクリーンセットの使用方法』の章を参照してください。 このプログラムは、デモンストレーショ

ンプログラムとして Dialog System ソフトウェアに組み込まれています。

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

Net Express は、編集、デバッグ、および、アニメートを実行するための統合的な環境です。

アプリケーションのデバッグ中に、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。 コードのアニメート中は、各文が実行されるに従って、ソースの各行が次々にハイライトされ、各文の結果が表示されます。 プログラムの実行速度を制御したり、実行を中断してデータ項目を確認および変更したりできます。 詳細については、ヘルプトピックの『デバッグ』を参照してください。

プログラムをテストした後で、プログラムのスクリーンセットを個別に変更できます。 スクリーンセットとプログラムが条件に合うまで、この操作を続けることができます。

スクリーンセットのテスト

Dialog System の主要な利点は、スクリーンセットや COBOL プログラムの完成前でも、インターフェイ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一部をプロトタイプ化して、概要を把握したり、試行したりできます。

インターフェイスの開発期間中にプロトタイプを作成すると、多くの作業を迅速に処理することができて便利です。 作業中のシステムモデルを短時間で構築し、実際に運用した場合のシステムの状態を開発者やユーザが端末で確認できます。 そのため、迅速で費用をかけずにシステムを変更できます。

ダイアログを指定しないでスクリーンセットをテストして、デスクトップのレイアウトを整えたり、フィールドを正しく設定したりできます。

スクリーンセット Animator の使用方法

Dialog System のスクリーンセット Animator は、Dialog System に組み込まれたユーティリティです。作成したスクリーンセットのテストとデバックを実行できます。 スクリーンセット Animator を使用すると、スクリーンセットを使用するプログラムを作成する前にスクリーンセットのルックアンドフィールと機能をテストできます。 スクリーンセット Animator は、次の操作に使用できます。

- オブジェクト、ダイアログ、および、データブロックを完全に機能させて、個別にスクリーンセットを実行できます。
- ユーザ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て、操作を実行できます。その後で、必要な回数だけ戻って戻り値を検査できます。
- ダイアログイベントと関数の実行をトレースできます。
- 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て、スクリーンセットを

再び実行できます。

- スクリーンセット Animator を使用して呼び出し側プログラムを実行し、データ、および、スクリーンセットと呼び出し側プログラムの間で受け渡される制御情報を確認できます。

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。

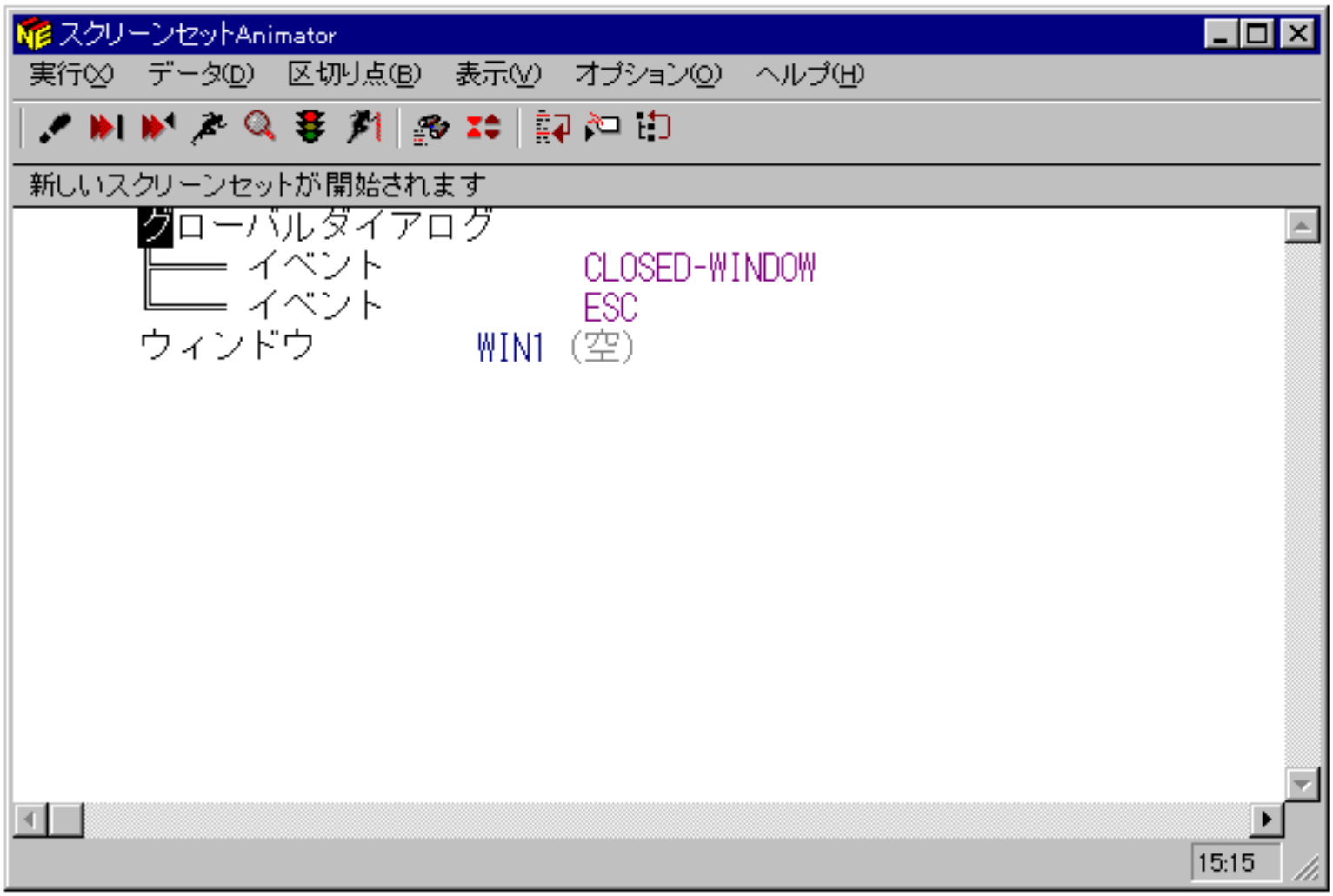

図 7-1： 「スクリーンセット Animator」ウィンドウ

この図の詳細については、ヘルプトピックの『スクリーンセット Animator』を参照してください。

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] メニューの [実行] を選択します。

スクリーンセットをテストする場合は、変更した値やデフォルト値を使用してスクリーンセットの主な機能を確認する必要があります。

呼び出し側プログラムに制御を戻す位置までスクリーンセットの処理が進むと、スクリーンセット Animator に戻り、制御ブロックと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。

データブロックには、Dsgrun に渡されたデータの現在の値が表示されます。 この時点で、スクリーンセット Animator に表示される情報を変更し、スクリーンセットの別の処理方法をテストできます。 データブロックを変更するには、[データ] メニューの [検査] 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の項目を表示し、[選択] プッシュボタンでデータブロックの項目を変更します。

必要に応じて値を変更してスクリーンセットを再実行し、呼び出し側プログラムの動作をシミュレートできます。

また、必要に応じてスクリーンセットを変更し、そのインターフェイスを十分に再テストできます。

詳細については、ヘルプの『スクリーンセット Animator』を参照してください。

LOST-FOCUS イベントと GAINED-FOCUS イベントのデバッグ

GAINED-FOCUS イベントまたは LOST-FOCUS イベントの結果として実行されたダイアログ関数または COBOL コードをスクリーンセット Animator または Net Express IDE でステップ実行すると、デバッガとアプリケーションの間の対話によって、アプリケーション内でさらにフォーカスが移動することがあります。 このフォーカスの移動は、ダイアログ関数または COBOL コードの行をステップ実行するデバッガに設定されたフォーカスによって発生します。 このフォーカスの移動は、通常の予測される動作ですが、LOST-FOCUS イベントと GAINED-FOCUS イベントをデバッグする場合に注意してください。

エントリフィールドが 2 つしか設定されていないアプリケーションウィンドウを例として説明します。 最初のフィールドにフォーカスがある場合に Tab キーを押すと、フォーカスが次のフィールドに移動します。 最初のフィールドには、LOST-FOCUS イベントが定義されており、このイベントが発生すると、COBOL 呼び出し側プログラムに制御が戻ります。 DSGRUN を最後に呼び出した行の後に COBOL のブレークポイントを設定した場合は、そのブレークポイントに達すると、デバッグ中のアプリケーションから Net Express IDE にフォーカスが移動します。 このフォーカスの移動によって、2 つ目のエントリフィールドに対して LOST-FOCUS イベントが発生します (COBOL のブレークポイントに達するまで、2 つ目のエントリフィールドにはフォーカスが設定されています)。 2 つ目のエントリフィールドにも LOST-FOCUS イベント用のダイアログが設定されていると、制御が DSGRUN に戻った途端にこのダイアログが実行されます。 COBOL のブレークポイントに到達しない場合は、2 つ目のフィールドの LOST-FOCUS イベントに対するダイアログ関数は実行されません。

ダイアログの定義

ユーザインターフェイスは、グラフィック表示のみではありません。 完成した仕様には、ユーザとコンピュータの対話方法、および、ユーザインターフェイスソフトウェアとアプリケーションソフトウェアとの対話方法も記述されています。

表示状態の定義が完了した後で、ユーザとマシンとの実行時の対話を定義する必要があります。 この対話はダイアログと呼ばれます。 ダイアログは、イベントと関数で構成されます。 イ

ベントが発生すると、そのイベントに関連付けられた関数が実行されます。 イベントは、キーボードのキーが押されたり、メニュー項目やオブジェクトが選択されたりすることによって発生します。

たとえば、BUTTON-SELECTED のような Dialog System のイベントは、ユーザが (マウスまたはキーボードを使用して) プッシュボタンを選択したときに発生します。 選択されたボタンが [入力] の場合は、CREATE-WINDOW という関数 (ユーザが他の情報を入力するための新しいウィンドウを作成する関数) を関連付けることもできます。

Dialog System を使用することによって、ユーザ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できます。 コントロールダイアログは、各コントロールに影響します。ウィンドウダイアログは、各ウィンドウまたはダイアログボックスのすべてのコントロールに影響します。また、グローバルダイアログは、すべてのウィンドウとオブジェクトに影響します。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の順に検索します。

ダイアログの文については、『ダイアログの使用方法』の章とヘルプトピックの『Dialog System の概要』を参照してください。

スクリーンセットの再テスト

スクリーンセットを再び保存して、インターフェイスを再実行します。 フィールドにデータを入力し、インターフェイスのオプションボタン、リスト項目などのオブジェクトを選択します。

スクリーンセットの変更

スクリーンセットの再テスト後に、スクリーンセットを変更できます (画面のレイアウトの改良など)。 ここで説明した手順を繰り返して、スクリーンセットを十分に設計してください。

スクリーンセットが完成した後で、スクリーンセットを含むユーザインターフェイスを使用する COBOL プログラムを作成する必要があります。

Windows GUI アプリケーションの新規作成ウィザードでは、必要に応じて、自動的に COBOL プログラムを作成できます。 詳細については、『Windows GUI アプリケーションの新規作成ウィザード』の章を参照してください。 『サンプルプログラム』の章では、簡単なプログラムを生成するためのサンプルコードを紹介しています。このプログラムでは、ユーザの入力を読み取り、ユーザの指示に応じて、保存または消去します。

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、さまざまな付随作業を行う必要があります。

Net Express から提供されるプロジェクト機能を使用して、Dialog System アプリケーションをビルドします。 詳細については、ヘルプトピックの『アプリケーションのビルド』を参照してください。

ヘルプトピックの『コンパイル』では、アプリケーションを運用するための準備について説明しています。

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。 完成した製品をディスクにコピーしてエンドユーザに送付し、他のマシンにロードしてアプリケーションとして実行できます。

最終製品は、アプリケーションのサイズによって、次の一方または両方で構成されます。

- 実行可能モジュール。 これらのファイルは、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。 これらのファイルは、ファイル入出力やメモリ管理などの機能を実行します。

ヘルプの追加

Dialog System 拡張機能を使用して、Dialog System のスクリーンセットから直接 Windows ヘルプを表示できます。 Dialog System 拡張機能は、CALLOUT ダイアログ関数を使用して実装したダイアログ関数を表します。 Dialog System 拡張機能は、Windows ヘルプの表示やファイル選択機能の実行などの多くの定型的なプログラム作業を実行するための機能です。 Dialog System 拡張機能の詳細については、ヘルプトピックの『Dialog System 拡張機能』を参照してください。

Helpdemo スクリーンセットでは、Dsonline Dialog System 拡張機能を使用しています。これは、Windows のヘルプを表示するための Dialog System の拡張機能です。 Helpdemo スクリーンセットの詳細については、『チュートリアル - サンプルスクリーンセットの作成』の章にある『ヘルプの追加』を参照してください。

アプリケーションの最適化

COBOL プログラムを最適化するためのヒントについては、ヘルプトピックの『効率的なプログラミング』を参照してください。

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。説明する内容は、次のとおりです。

- COBDIR 環境変数を使用して、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 ではなく、UNSHOW-WINDOW を使用する
- オブジェクト名を最小化する

- 実行時の保存形式を使用する
- ds-no-name-info フラグを設定する

ディレクトリの検索の制限

Path 文を実行すると、 必要なプログラムまたはファイルが現在のディレクトリにない場合に、指定されたドライブとそのディレクトリがオペレーティングシステムによって検索されます。 Path では、検索先の候補を指定した順序で検索パスが定義されます。

開発環境では、Windows のシステムレジストリが COBOL ランタイムによって使用され、Dialog System インストール先と現在の Net Express プロジェクトの位置が識別されます。

アプリケーションが完成し、運用の準備ができたときに、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を検索先候補の先頭付近に指定します。

イベントダイアログの検索

『ダイアログの使用方法』の章では、Dialog System でイベントと手続きを検索するための次の 3 つのレベル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

プッシュボタンなどのコントロールでイベントが発生したときに、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が検索されます。 イベントが検出されると、その下に指定された関数が実行されます。

イベントが検出されない場合は、コントロールを含むウィンドウに設定されたダイアログでイベントが検索されます。 ウィンドウに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が検索されます。

次の点に注意してください。

- ダイアログと手続きは、下位のレベルに指定する。
- グローバルダイアログは、共通のダイアログと手続のために予約しておく。
- 最も一般的なイベントは、イベントテーブルの上位に指定する。

ダイアログの数が増えるほど、実行速度は遅くなります。 また、グローバルレベルに定義するイベントや手続きの数が増えるほど、実行速度が遅くなります。 グローバルダイアログは一般的なルーチンをまとめて指定する場合に便利ですが、下位のレベルでイベントが検出されない

たびに検索されます。

UNSHOW-WINDOW と DELETE-WINDOW

UNSHOW-WINDOW 関数と DELETE-WINDOW 関数は似ています。 どちらの関数も、指定されたウィンドウやダイアログボックスを非表示にして、入力フォーカスを別の場所に設定します。

DELETE-WINDOW 関数では、ウィンドウとそのコントロールを削除します。 そのウィンドウを再度表示するには、明示的または暗黙的に再度作成する必要があります (ウィンドウが作成されていない場合は、SHOW-WINDOW 関数と SET-FOCUS 関数を実行すると、ウィンドウが暗黙的に作成されます)。 ダイアログボックスとウィンドウ (およびダイアログボックスとウィンドウに定義されたすべてのオブジェクト) を作成するには、多少時間がかかります。

UNSHOW-WINDOW 関数を実行した場合は、ウィンドウを作成したまま残しておくため、再度すぐに表示できます。

そのため、ウィンドウまたはダイアログボックスを非表示にして再度表示する場合は、UNSHOW-WINDOW 関数を使用します。 システムログオンウィンドウなどのように、ウィンドウまたはダイアログを終了する場合は、リソースを解放するために DELETE-WINDOW 関数を使用します。

オブジェクトへの最小限の命名

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限り、名前を付ける必要はありません。 ダイアログで参照されるオブジェクトのみに名前を付けることによって、記憶域をやや解放し、Dialog System によるオブジェクトの検索時間を減らすことができます。

実行時の保存形式

Dialog System アプリケーションを配布する場合は、実行形式で保存オプションを使用することをお勧めします。 このオプションによって、スクリーンセットを実行するために十分な情報量を保存できます。ただし、スクリーンセットを検索して編集するための情報は保存されません。

このオプションは、セキュリティ上の問題点に対応しており、アプリケーションに必要なディスク容量を減らすこともできます。 スクリーンセットのサイズは、元のサイズの 3 分の 1 ほどに削減できます。

Dialog System では、あまり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を設定してスクリーンセットを保存すると、その後でスクリーンセッ

トを変更できません。 そのため、編集できるスクリーンセットのコピーを保存しておいてください。

---

ds-no-name-info の使用

制御ブロックに ds-no-name-info フラグが設定されている場合は、 Dialog System が呼び出し側プログラムに戻るときに、スクリーンセットのヒープ領域が読み込まれず、ds-object-name と ds-window-name の値を取得できません。 これら 2 つのフィールドのどちらも使用していない場合は、初期化時にこのフラグを真に設定します。 たとえば、次のように記述します。

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

このように指定すると、RETC 関数の後で Dialog System からアプリケーションに戻るときに、わずかに速く終了するようになります。

スクリーンセット Animator では、スクリーンセットに保存された名前情報が必要なので、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 を使用できなくなります。

詳細情報

呼び出しインターフェイスのより高度な使用方法 (複数のスクリーンセットや同じスクリーンセットの複数のインスタンスの使用方法など) 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 以外の環境から Net Express にアプリケーションを移行する場合は、環境間の注意点に関する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。

スクリーンセットの使用方法 データアクセス

第 8 章 : Windows GUI アプリケーションの新規作成ウィザード

ここまでは、Dialog System の基本的な要素を紹介しました。 ここでは、新しい Windows GUI アプリケーションの新規作成ウィザードがもつ次のような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が設定されたスクリーンセットをすばやく簡単に作成できます。
- インストールされたデータベースに対して容易にアクセスおよびクエリーを実行できます。

  「データアクセス」オプションを選択し、ウィザードで SQL クエリーを定義するのみで実行できます。

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のみでなく、ユーザが要求した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：Windows GUI アプリケーションの新規作成ウィザードが出力するプログラムとファイルは、ユーザ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。 これらのプログラムとファイルは、すべての状況に対応できる汎用コードではありませんが、基本機能の使用方法の習得に役立ちます。 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』オンラインマニュアルの『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用方法が説明されています。

ウィザードの起動

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

- Net Express IDE で [ファイル] メニューの [新規作成]、[Dialog System スクリーンセット] を順に選択して、直接起動します。
- IDE で [ファイル] メニューから [新規作成]、[プロジェクト] を順に選択して、直接起動します。 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 で [ファイル] メニューの [新規作成] を選択します。

注：ウィザードを IDE から開く場合にプロジェクトが開いていないと、新しいプロジェクトが作成されます。 この場合は、プロジェクトの名前と位置を指定する必要があります。

ウィザードの使用方法

ここでは、ウィザードの各手順の機能を簡単に説明します。

手順 1：スクリーンセット名

このウィザードでは、新しいスクリーンセットの名前が提示されます。 このスクリーンセット名は変更できます。 拡張子 .gs を追加する必要はありません。この拡張子は自動的に追加されます。

手順 2：インターフェイスタイプ

ここでは、新しいスクリーンセットのインターフェイスを 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は、インターフェイスがもつ、子 MDI の数を指定します。 SDI では、必要な一次ウィンドウの数を指定します。

手順 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の機能を選択できます。すべての必要な機能について、適切なデータブロック、ダイアログ、および、コントロールプログラムを生成できます。ウィザードのこの手順を図 8-1 に示します。

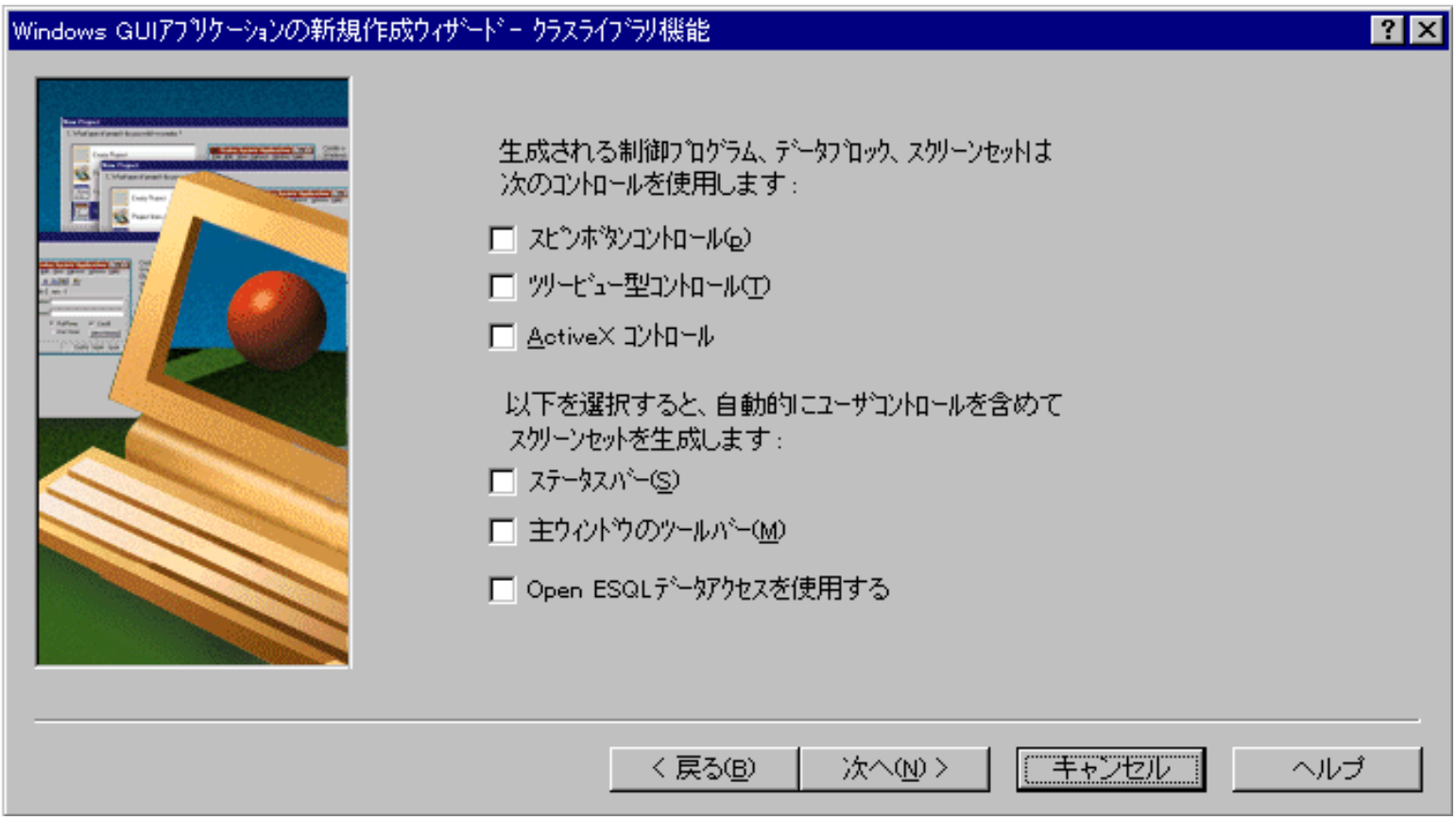

図 8-1： クラスライブラリ機能

「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 ODBC データソースにアクセスし、次の手順で説明するクエリーを設定できます。 このオプションを選択しない場合は、次の手順をスキップし、手順 5 に進んでください。

「OpenESQL データアクセスを使用する」を選択した場合は、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。

手順 4：クエリーの定義

手順 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は、この手順をスキップし、手順 5 に進んでください。

注：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は、ウィザードのこの手順を実行する前に ODBC ドライバをインストールする必要があります。

この手順では、次の操作を実行できます。

- 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ペインに一覧表示する。

  インストールされているデータソースへのアクセスには、OpenESQL アシスタントの技術が使用されます。

  データソースをまったく設定していない場合は、Net Express のインストール時に自動的に設定されるサンプルのデータソースのどれかを利用できます。 その中の Net Express 4.0 Sample 2 は、sample.mdb という Microsoft Access のサンプルデータベースです。これは、Net Express 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。

- 表を選択する。

  データベース名をダブルクリックすると、そのデータベースに含まれる表のリストが表示されます。表をダブルクリックして、リストから表を選択してください。
- 列を表示する。

  表を選択すると、表は列に拡大されます。 列を選択するには、列をダブルクリックします。 列を選択すると、右側のクエリーペインに EXEC SQL 文が追加されます。
- 複数の表を選択する。

  次の表をダブルクリックすると、「クエリーに追加する表」ダイアログボックスに表結合方法が表示されます。 別の表結合方法を使用する場合は、[いいえ] をクリックして [検索条件] タブを選択します。
- クエリーを選択する。

  列を選択したら、ツールバーの [クエリーの実行] をクリックして、クエリーを実行できます。クエリー結果を表示して、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：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は、自動的に 1 つ選択されます。

この手順の画面を図 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クエリーをビルドおよびテストします。 この画面は、IDE の [ツール] メニューの [OpenESQL アシスタントを表示] 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 の使用方法については、『データベースアクセス』オンラインマニュアルの『OpenESQL』の章を参照してください。

手順 5：拡張機能

この手順では、スクリーンセットに必要な Dialog System の拡張機能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設定します。 それぞれの拡張機能によって要件が異なります。

手順 6：Dialog System の実行時コンフィグレーションオプション

これら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については、[ヘルプ] をクリックしてください。

手順 7：COBOL プログラムの生成

「スケルトン COBOL プログラムを生成」を選択すると、スケルトン COBOL プログラムが生成されます。このプログラムは、dsgrun を呼び出し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の名前が付いていますが、名前を変更できます。

この手順の画面を図 8-3 に示します。

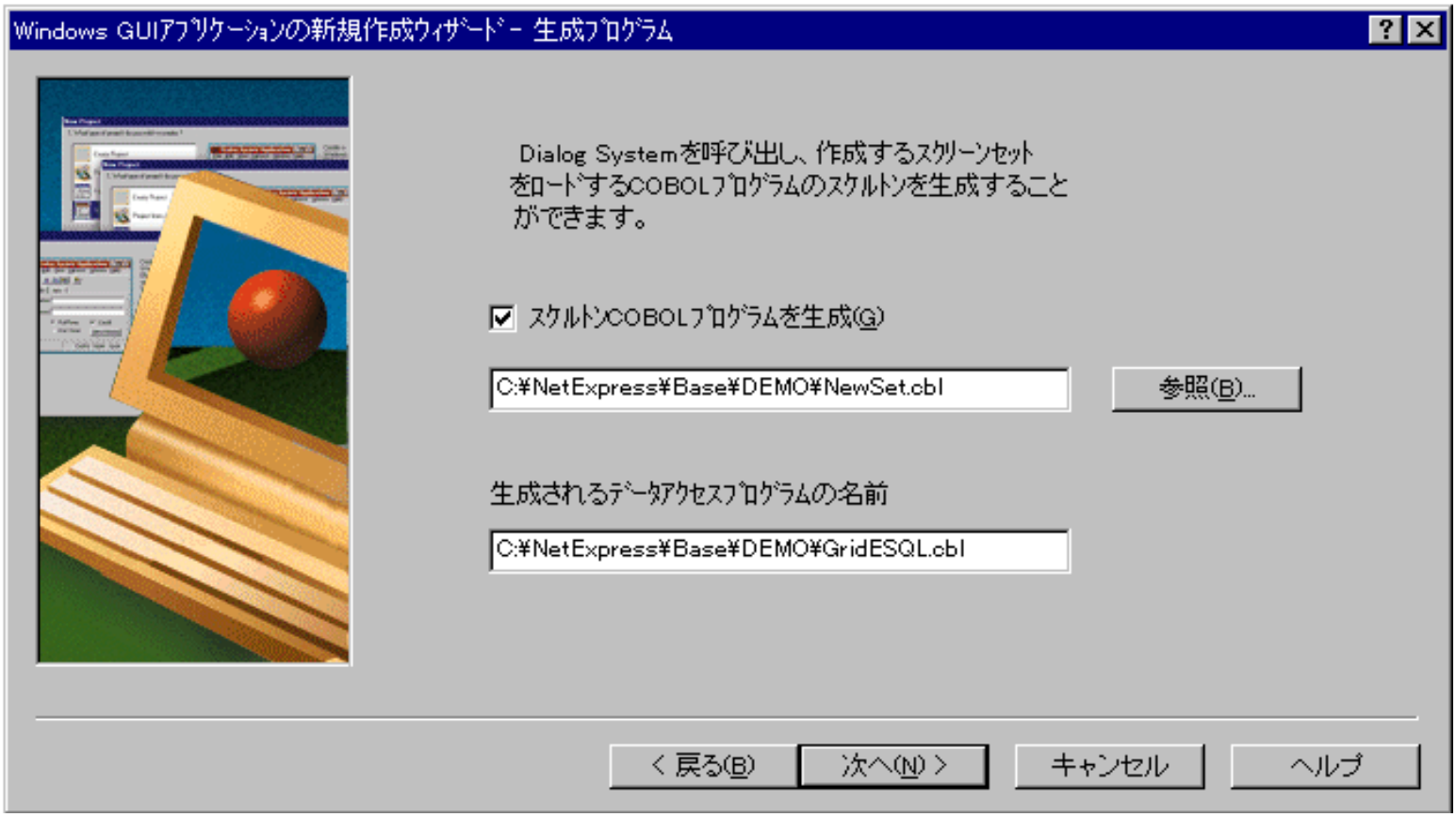

図 8-3： プログラムの生成

手順 3 で「OpenESQL データアクセスを使用する」を選択した場合は、「生成されるデータアクセスプログラムの名前」の項目が表示されます。

手順 8：選択したオプションの確認

この手順では、これまでの手順で選択したオプションを確認します。 前の手順に戻って、選択を変更できます。 選択内容を確認すると、スクリーンセットとプログラムが作成されます。

ウィザードからの出力

Windows GUI アプリケーションの新規作成ウィザードで作成されるファイルは、選択するオプションによって異なります。作成されるファイルには、次のどれかです。

- 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。手順 4 で「グリッド」を選択した場合に、グリッドコントロールを作成および管理します。
- LBoxESQL.cbl。手順 4 で「リストボックス」を選択した場合に、リストコントロールを作成および管理します。
- ツールバーコントロールプログラム (拡張子 .cbl)

アプリケーションの実行

Windows GUI アプリケーションの新規作成ウィザードで作成されるファイルを使用して、データベースの詳細を表示、クエリー、または変更するには、次の操作を実行します。

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、プロジェクトをリビルドします。

2. IDE のツールバーにある [実行] をクリックするか、または、[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。

詳細情報

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、データを操作できます。 詳細については、『データアクセス』の章を参照してください。

Windows GUI アプリケーションの新規作成ウィザード
カスタムコントロールのプログラミング

第 9 章 : データアクセス

前の章では、新しい Windows GUI アプリケーションの新規作成ウィザードによって、インストール済みのデータベースに対するアクセスやクエリーを簡単に実行できることを説明しました。 ここでは、データの操作方法について説明します。

注：Windows GUI アプリケーションの新規作成ウィザードが出力するプログラムとファイルは、ユーザが独自のデータアクセスアプリケーションを開発するための出発点となります。 これらのプログラムとファイルは、すべての状況に対応できる汎用コードではありませんが、基本機能の使用方法の習得に役立ちます。 これらのコードを必要に応じて (通常はより複雑なコードに) 変更できます。

『入門書』オンラインマニュアルの『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用方法が説明されています。

Windows GUI アプリケーションの新規作成ウィザード

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、『Windows GUI アプリケーションの新規作成ウィザード』の章にある『手順 3：クラスライブラリ機能』の「OpenESQL データアクセスを使用する」オプションを選択しておく必要があります。

インストールされたデータベースへのアクセス

アプリケーションのビルド後にクエリー結果を表示する方法は、『Windows GUI アプリケーションの新規作成ウィザード』の章にある手順 4 で選択したオプションに応じて 2 種類あります。

- 「リストボックス」

  または、

- 「グリッド」

これらのビューは機能的には同じですが、インターフェイスが異なります。 グリッドビューは、わかりやすい表示方法です。リストボックスビューは、Windows の使用経験が豊富なユーザに適しています。

グリッドビューでクエリーを実行すると、そのクエリーに対するすべてのデータがデータベースからメモリに読み込まれます。 メモリから一度に読み込まれた行のみが画面にページとして表示されます。画面をスクロールすると、メモリから行が取得され、表示中のページが更新されます。

単独の表データベースクエリーのグリッドビューの例を図 9-1 に示します。

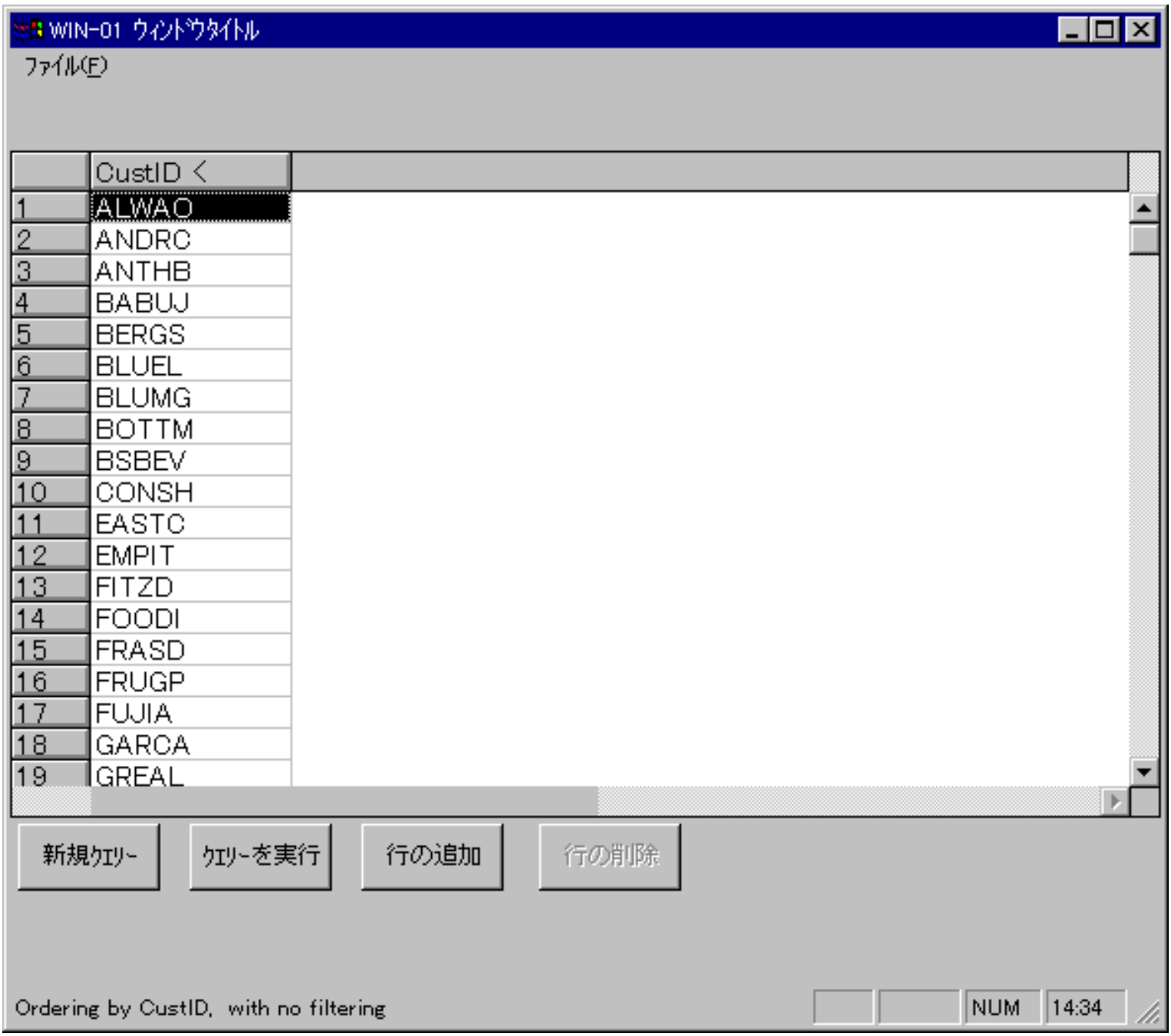

図 9-1： 単独の表データベースクエリーのグリッドビュー

表結合データベースクエリーのグリッドビューの例を図 9-2 に示します。

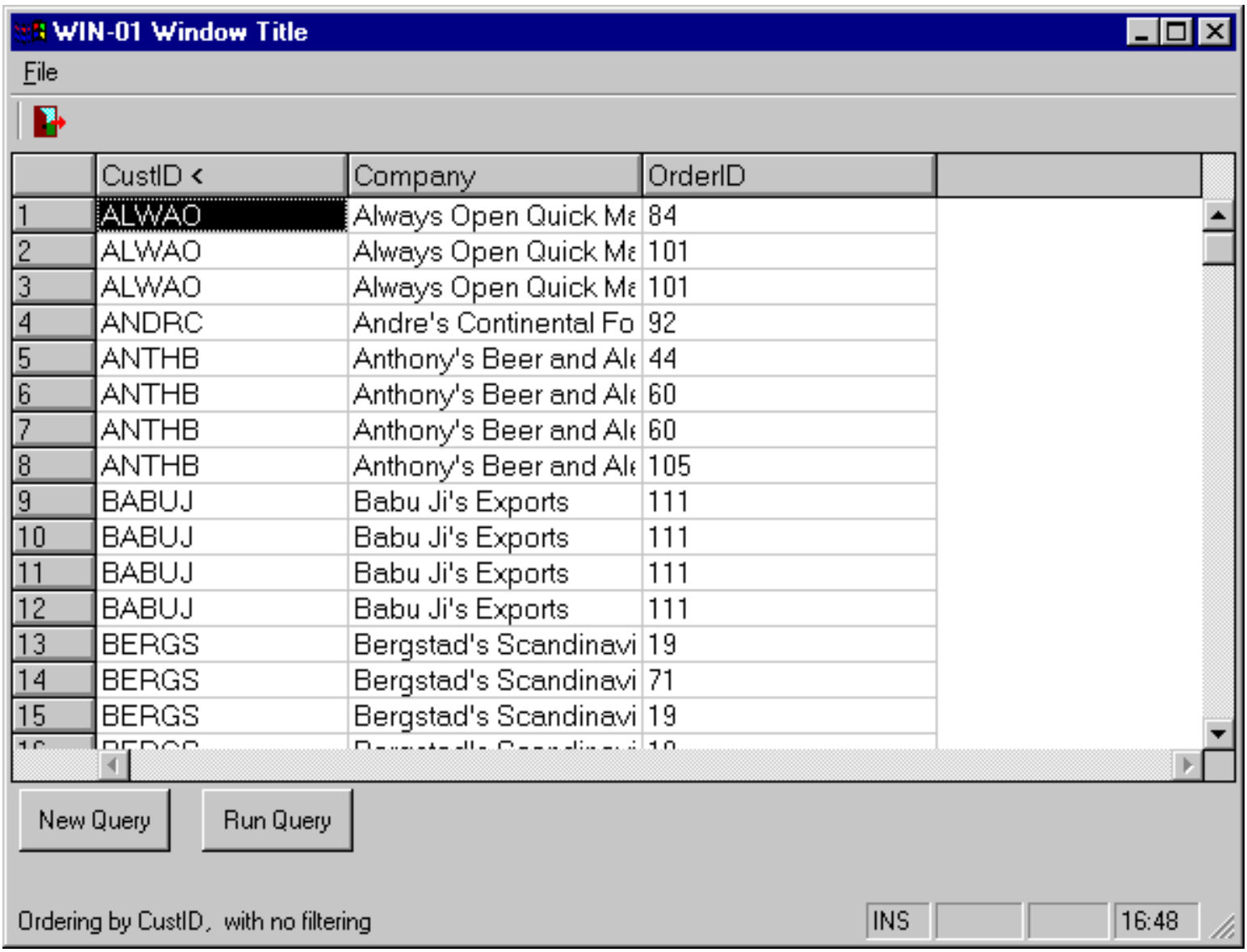

図 9-2： 表結合データベースクエリーのグリッドビュー

単独の表データベースクエリーのリストボックスビューの例を図 9-3 に示します。

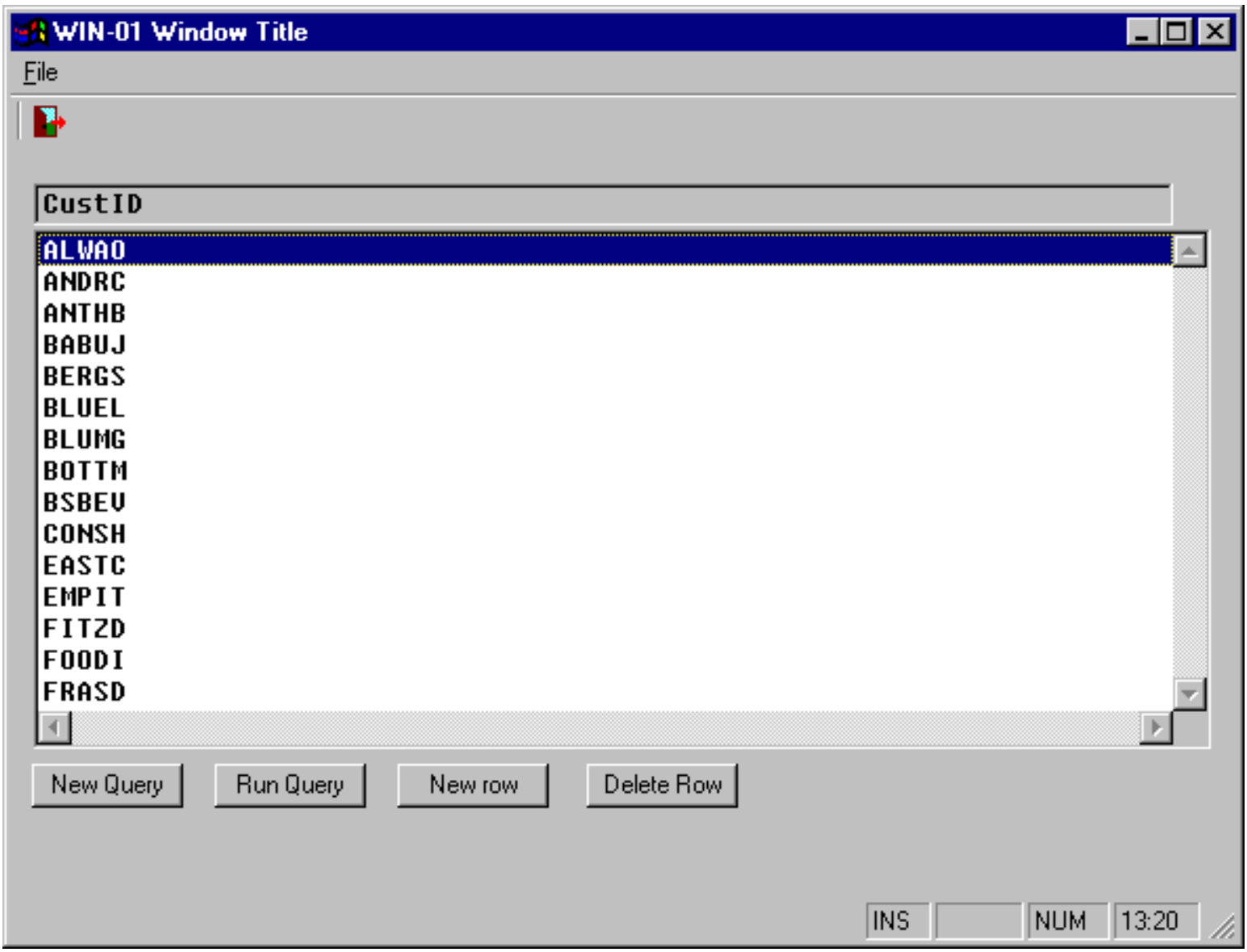

図 9-3： 単独の表データベースクエリーのリストボックスビュー

表結合データベースクエリーのリストボックスビューの例を図 9-4 に示します。

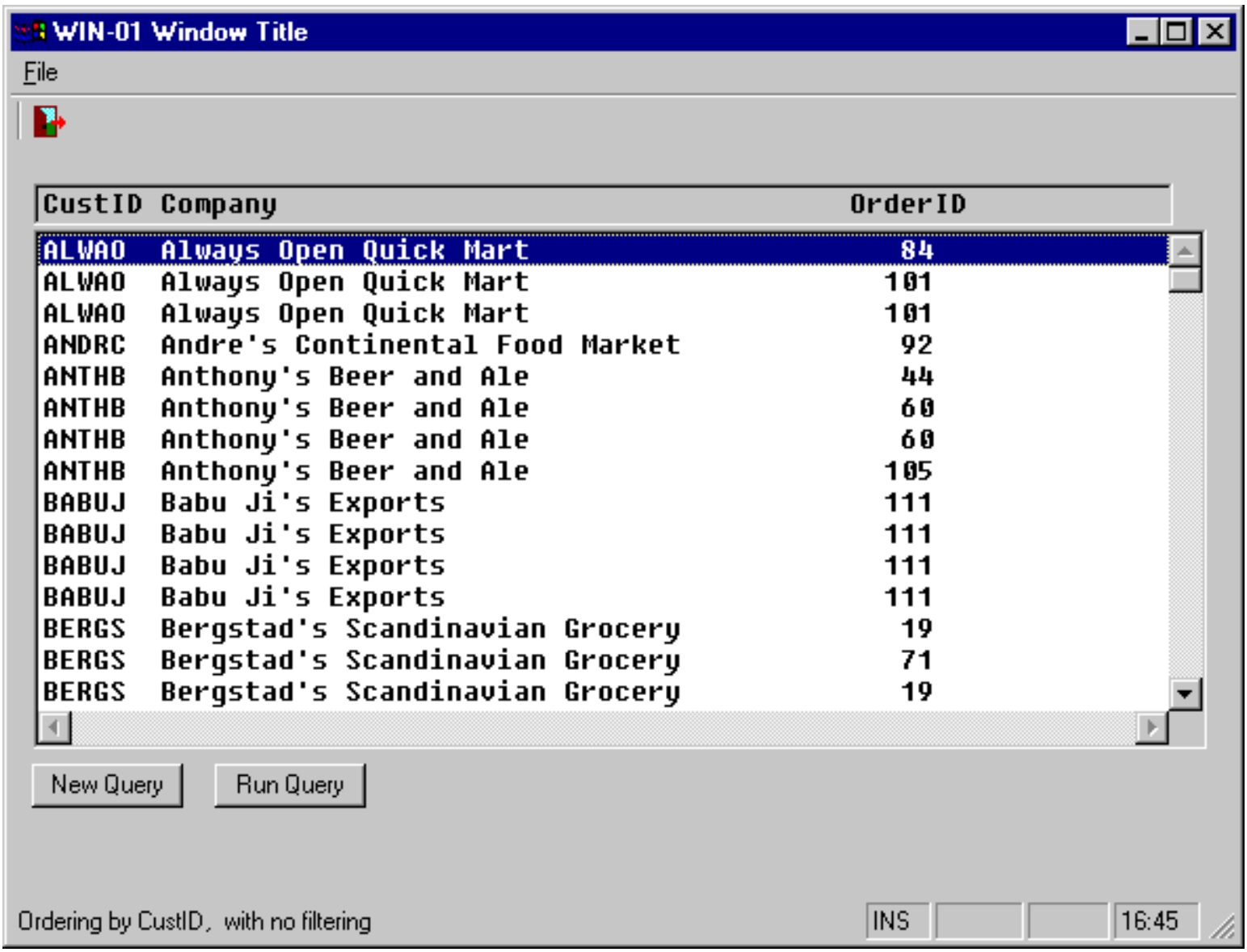

図 9-4： 表結合データベースクエリーのリストボックスビュー

画面の一番下のステータスバーには、マウスが指すオブジェクトに関する情報が表示されます。 マウスを画面の他の部分に移動すると、選択した機能の種類がステータスバーに表示されます。 メッセージボックスには、エラーに関する情報 (キーフィールドを編集しようとした場合など) も表示されます。

データの操作

上記の図のとおり、表結合クエリービューにはボタンが 2 つしかありません ([クエリーの新規作成] と [クエリーの実行])。 これは、表結合クエリーの出力が読み取り専用であるためです。 データ操作に関する次の説明は、単独の表データベースクエリーのみに関するものです。

実行できる操作は、次のとおりです。

- 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

データを編集する

	グリッドビュー	リストボックスビュー
フィールド内のデータを変更します。	フィールドを選択し、変更内容を指定します。キーフィールドは変更できません。 ソートされた列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、開いたダイアログボックスの中で変更を指定します。 ソートされた列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。
次の列に移動します。	マウスまたは左右の矢印キーを使用します。	
フィールド内のデータを削除します。	フィールドを選択し、Delete キーを押します。	変更する行を選択し、開いたダイアログボックスの中で変更を指定します。 ソートされた列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。
画面を最新表示します。	[クエリーの実行] をクリックします。	[クエリーの実行] をクリックします。

注：1 つのクエリーを一度に複数開くことができます。そのため、ディスクからクエリーが最新の状態に更新されてから、データに対する編集がエコーされます。 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。

新しい行を挿入する

	グリッドビュー	リストボックスビュー
新しい行を挿入します。	[新規行] をクリックします。 空のグリッドの最も上にある行に新しいデータを入力します。	[新規行] をクリックすると、キーフィールドを除く各列に関するエントリフィールドをもつダイアログボックスが表示されます。 どのエントリフィールドもデータの妥当性検査は実行されません。
挿入します。	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。	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。
クエリーの新しい行を表示します。	[クエリーの実行] をクリックします。	Enter キーを押します。

注：新しい行がすぐに表示されないことがあります。 考えられる原因は、次のとおりです。

- このデータが現在のクエリーによって除外されている可能性があります。 新しい行を表示するには、新しいクエリーを選択する必要があります。
- 新しい行が現在のビューの外部にある可能性があります。新しい行を表示するには、その行の位置までスクロールする必要があります。

行を削除する

グリッドビューとリストボックスビューのどちらでも、データ行を削除するには、削除する行を選択して、[行の削除] をクリックします。

データの表示

実行できる操作は、次のとおりです。

- 新しいクエリーを定義して、データを検索する
- 列ごとにデータをソートする

データを検索する

	グリッドビュー	リストボックスビュー
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. 検索するフィールドをハイライトします。 3. 検索条件を入力します。 4. フィールドを右クリックして、ポップアップメニューを表示します。 5. 論理演算子 (=、>、<、および !=) を選択します。 デフォルト値は「フィルタなし」です。 6. [クエリーの実行] をクリック	1. [クエリーの新規作成] をクリックします。 ダイアログボックスが開き、アプリケーションの各フィールドに関するエントリフィールドが表示されます。 2. 検索条件を入力します。 3. フィールドを右クリックして、ポップアップメニューを表示します。 4. 論理演算子 (=、>、<、および !=) を選択します。 デフォルト値は「フィルタなし」です。

	して、検索結果を表示します。	5. Enter キーを押して、検索結果を表示します。

データをソートする

	グリッドビュー	リストボックスビュー
列内のデータをソートします。	列ヘッダーバーをクリックします。次の項目が表示されます。 ● 「<」は、データが昇順にソートされることを表します。 ● 「>」は、データが降順にソートされることを表します。	リストボックスを右クリックします。 ソートする列を選択するためのポップアップメニューが表示されます。

データアクセス　複数のスクリーンセット

第 10 章 : カスタムコントロールのプログラミング

ここでは、ユーザコントロールや 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。 具体的には、次の内容を説明します。

- コントロールプログラム

  ここでは、「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、ユーザインターフェイスのオブジェクトやコントロールを作成したり、操作したりするための COBOL 原始プログラムです。

- ActiveX コントロール

- ユーザコントロール

コントロールプログラム

コントロールプログラムを使用すると、オブジェクトをそのオブジェクトに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最新表示、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させ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、汎用性をもつように設計されています。 そのため、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、再利用できます。 ただし、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スピンボタンコントロールは、変更しないで自由に利用できます。
- ツールバーコントロールとツリービューコントロールは変更しないで使用できますが、個々の要件に合わせてプログラムへのデータインターフェイスを変える必要があります。
- ActiveX コントロールは、実装状態に依存します。

  これらのバージョンを個々の要件に合わせて適合させる必要があります。 ただし、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、『ユーザコントロール』を参照してください。

コントロールプログラムは、Net Express のクラスライブラリを呼び出して実行します。

ユーザコントロールの実装アーキテクチャ

次の図は、Dialog System とクラスライブラリを使用してユーザコントロールを実装するためのアーキテクチャを表した図です。

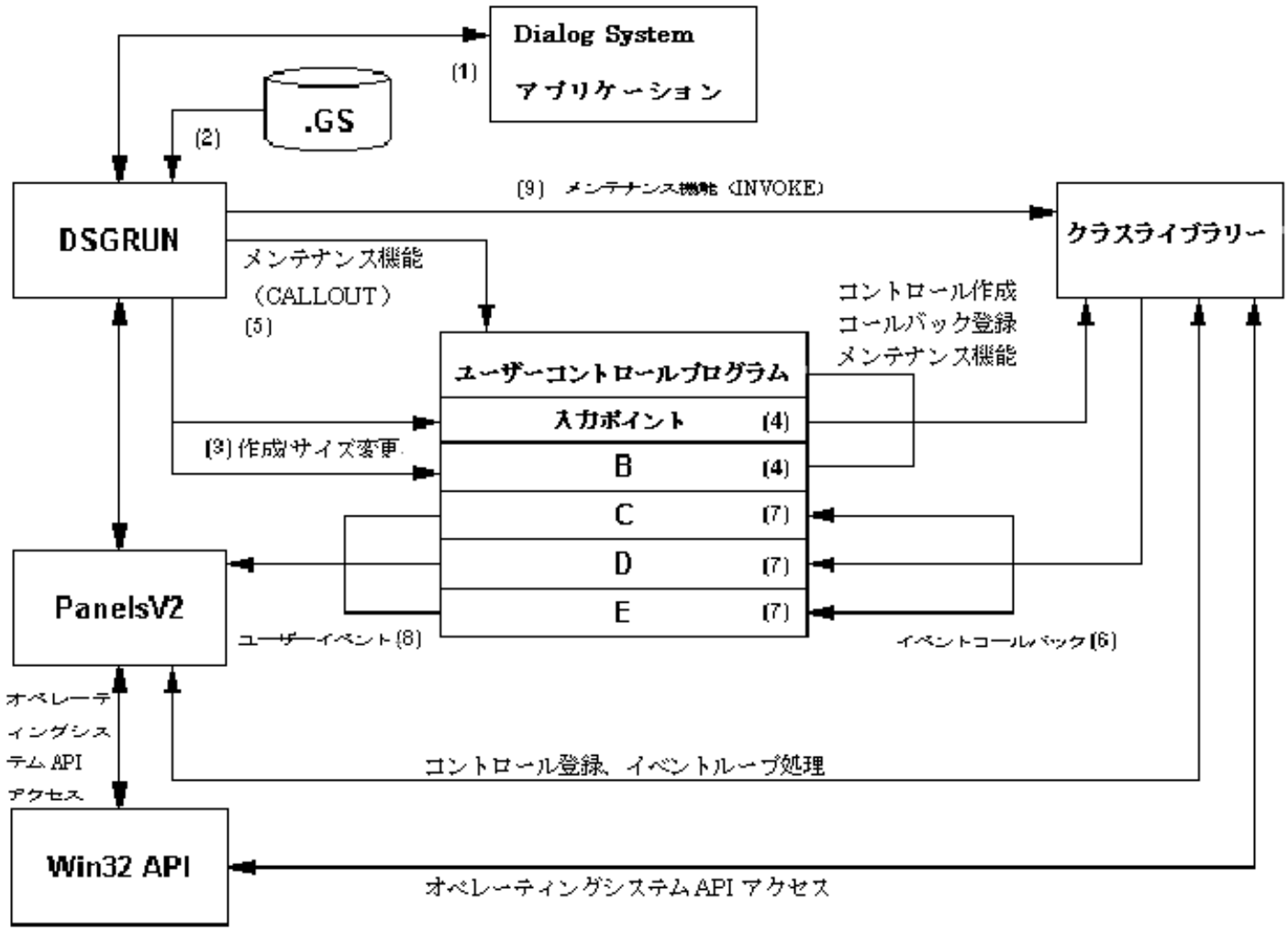

図 10-1： コントロールの実装アーキテクチャ

ActiveX コントロールまたはユーザコントロールを作成する場合は、次の操作を実行します。

1. ウィンドウ上のユーザコントロールをスクリーンセット内で描画し、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で 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. ActiveX コントロールとユーザコントロールを定義したウィンドウが Dialog System で作成されると、そのユーザコントロールに関連付けられたプログラム内のエントリポイントが呼び出されます。

4. エントリポイントのコードで、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ために必要なすべてのタスクを実行します。

   コントロール作成手順の一部として、生成されたコントロールプログラム内のコードによって、プログラムに関連するイベントのコールバックを登録できます。 これによって、プログラムのエントリポイントと定義済みのシステムイベントのオカレンスが関連付けられます。

5. 定義済みのシステムイベントが発生すると、指定されたエントリポイントのコードが実行されます。

6. エントリポイントのコードで、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. エントリポイントのコードで、Panels V2 プログラムにメッセージを返します。 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できます。

8. コントロールは、ユーザ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、または、Dialog System の INVOKE 関数を通じてクラスライブラリによって直接操作できます。

ActiveX コントロール

『コントロールオブジェクト』の章で説明したとおり、ActiveX コントロールは、サードパーティのベンダから提供されますが、Dialog System 内に統合できます。 インターフェイスで ActiveX コントロールを使用する場合は、条件に合わせてカスタマイズする必要があります。

ActiveX コントロールの属性

ActiveX には、設計時に次の 3 種類の属性を設定できます。

- Dialog System の属性

  ActiveX コントロールを選択するときに定義する必要がある属性です。

  [編集] メニューから [属性] を選択するか、または、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。 このページには、一般属性リスト内の一般属性も含まれます。 すべての ActiveX コントロールに属性シートが設定されているわけではありません。

通常は、デフォルトの属性は、作成するアプリケーションに適合しません。 たとえば、スプレッドシートの ActiveX コントロールに対するデフォルトの属性は、2 列× 2 行です。 一方、Dialog System の時計の ActiveX コントロールのようにデフォルトの属性をそのまま使用できるものもあります。

設定する属性を決定するには、ActiveX コントロールのマニュアルを参照してください。 変更した属性データは、スクリーンセットに格納されます。

ActiveX コントロールのカスタマイズ

ActiveX コントロールをカスタマイズするには、次の手順を実行します。

- 必要な ActiveX コントロールを選択します
- そのコントロールに対して Dialog System の属性を定義します
- プログラミングアシスタントを使用して ActiveX コントロールをカスタマイズします

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、次の操作を実行します。

1. [ファイル] メニューの [インポート] を選択します。
2. サブプルダウンメニューの [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 コントロールが表示されたリストボックスから、必要な 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 コントロールを次に使用する場合は、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 10-2 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 10-2： 「ActiveXコントロールの属性」ダイアログボックス

ActiveX コントロールの属性の定義

「ActiveX コントロールの属性」ダイアログボックスで次の操作を実行します。

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. Net Express プロジェクトが開いており、Net Express 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] をクリックして、COBOL 原始プログラムを作成します。この原始プログラムは、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、開いたプロジェクトに追加されます。

   このプログラムが、スクリーンセット内の ActiveX コントロールに対する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。

プログラミングアシスタントを使用した ActiveX コントロールプログラムのカスタマイズ

カスタマイズした ActiveX コントロールのプログラム例が Net Express\DialogSystem\Demo\Activex\Custgrid フォルダに保存されています。 この custgrid.txt ファイルには、デモンストレーションプログラムの実行方法が記述されています。

ActiveX コントロールを実行するためのコントロールプログラムをカスタマイズするには、次の操作を実行します。

- ActiveX コントロールによって発生するイベントのイベントハンドラを登録します。
- 必要なイベントを処理するためのコードを実装します。
- 実行時の属性を設定して 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、ActiveX コントロールでサポートされるメソッド、属性、およびイベントを検索する必要があります。 プログラミングアシスタントでは、そのグラフィック環境を使用してプログラムに必要な定義済みの機能を抽出できるので、作業を簡便化できます。 プログラミングアシスタントでは、次の処理を実行するためのコードを使用できます。

- 属性を取得または設定する。

  ActiveX コントロールには、属性または特性 (背景色など) が設定されており、これらの読み取りや変更が可能です。

  - 属性の現在状態を検出するには、GET property name 文を使用します。
  - 属性を変更するには、SET property name 文を使用します。

- メソッドを呼び出します。

  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の関数セットです。 これらの定義済みの関数では、コントロールに対して特定の動作を実行するように命令します。その場合に、パラメータを送信する、値を返す、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します。

  イベントは、ActiveX コントロールから提供され、動作が実行されたことを示します。 イベントでは、イベントの詳細を記述したパラメータ情報を送信できます。 この情報は、ActiveX コントロールを使用しているアプリケーションプログラムで受け取られ、認識されます。

  コールバックを登録すると、コントロールでイベントが発生したときに COBOL コードのブロックが実行されます。 たとえば、ActiveX グリッド行を削除すると、MessageName に記述されたプログラムのエントリポイントが呼び出されます。 エントリポイントのコードは、コントロールプログラムのソースファイルに含まれています。

  次のコード例では、ユーザが ActiveX のグリッドデータ行を削除した場合にコールバックが実行されるように登録します。

  ActiveX コントロールで生成されるイベントの名前は、次のとおりです。

  MOVE z"RowDeleted " TO MessageName

  イベントの発生時に実行されるエントリポイントの名前は、次のとおりです。

  MOVE ProgramID & z"DeletedRow" TO CodeName

  コールバックを次のように登録します。

  INVOKE anActiveX "SetNamedEventToEntry" USING MessageName

  CodeName

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、次の操作を実行します。

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 10-3 に示すように、「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。

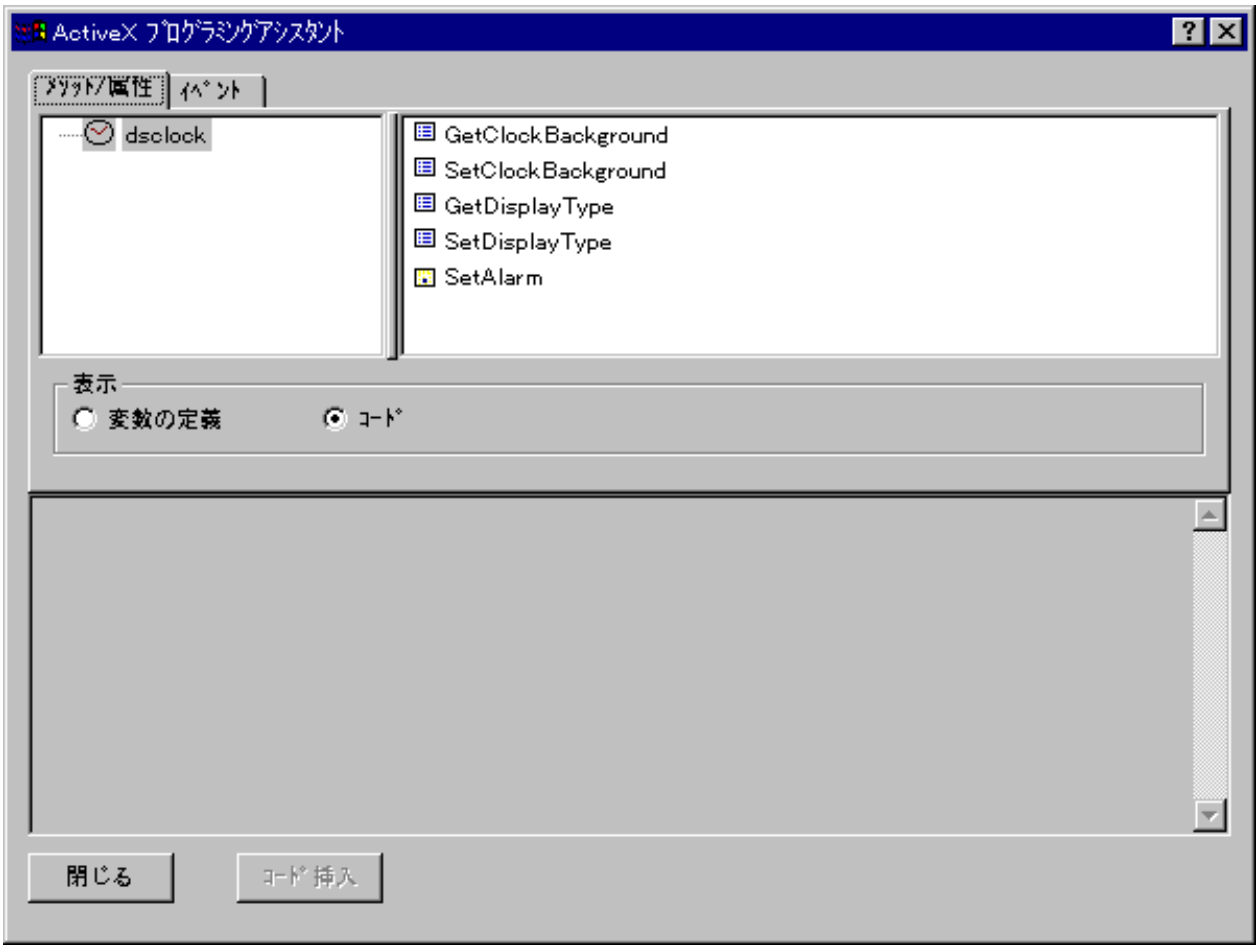

図 10-3： 「ActiveX プログラミングアシスタント」ダイアログボックス

このダイアログボックスには、2 つのタブページが表示されます。このタブページから、開いている COBOL プログラムにメソッドと属性またはイベントに関連したコードを直接挿入できます。

メソッドと属性

[メソッド / 属性] 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。

左側のペインには、ActiveX コントロールとその実行時のオブジェクト階層が表示されます。 右側のペインには、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。

右側のペイン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部のペインに表示されます。

実行時に ActiveX を作成するプログラムにこのコードを挿入するには、次の操作を実行します。

1. Net Express 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のプログラムロジックに完全に依存するためです。 コードは、コントロールの初期化、イベントのコールバック、または、簡単な

実行時の操作に使用できます。

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、選択した関数に必要なすべてのパラメータと戻り値が含まれています。

サブオブジェクト

ActiveX コントロールには、サブオブジェクトを関連付けることができます。各サブオブジェクトには、独自のメソッドや属性があり、これらを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、左側のペインで ActiveX コントロールに結合された「+」記号をクリックします。

一般にサブオブジェクトには、イベントが関連付けられていません。 これらのサブオブジェクトは、ActiveX コントロール内でグローバルに処理されます。

イベント

[イベント] タブを選択すると、図 10-4 に示すような画面が表示されます。

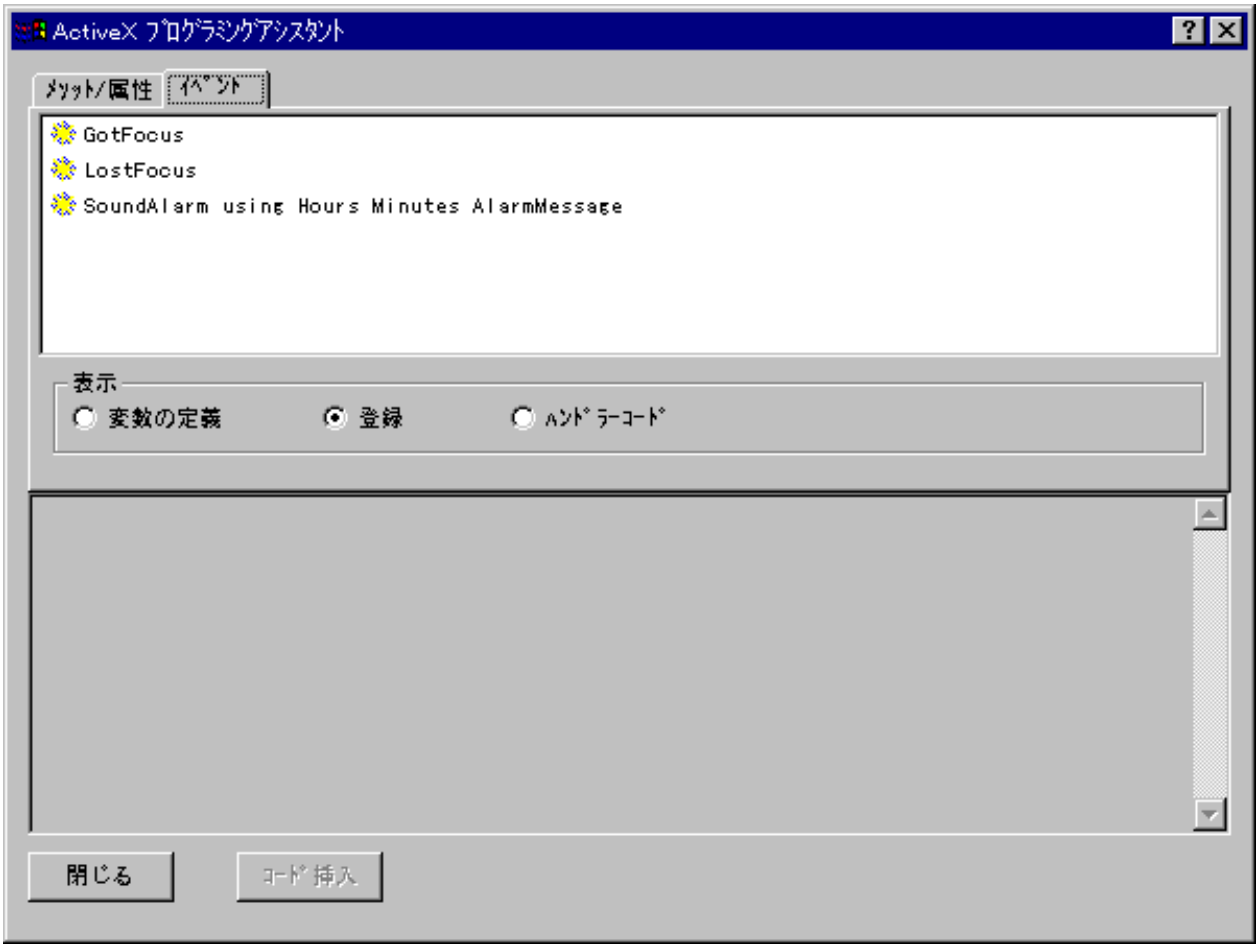

図 10-4：「ActiveX プログラミングアシスタント」ダイアログボックス - イベント

ダイアログボックスの上部には、イベント名が一覧表示されます。 各イベント名の横には、そのイベントから ActiveX コントロールプログラムに渡される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、下部のペインには、上部のペイン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。 オプションボタンには、次のような機能があります。

変数

[変数の定義] ボタンを選択すると、上部のペインで選択したイベントと関連付けられた変数が表示されます。 これら

の変数は、

- ActiveX コントロールから渡されるパラメータの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に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 に挿入する必要があります。

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ため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、次の 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エントリポイントを指定します。
- 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通知します。

Program ID+On+event name という名前のエントリポイントが生成されます。 たとえば、BeforeDeleteRow という名前のイベントに対しては、Program IDOnBeforeDeleteRow という名前のエントリポイントが生成されます。

注：複数のコントロールプログラム間でエントリポイントを一意に識別するために、生成されるエントリポイント名の前にはプログラム ID が設定されます。たとえば、複数の ActiveX コントロール内に同じイベント名がある場合は、エントリポイントにプログラム ID が設定されていないと、OLE クラスライブラリから呼び出すコントロールプログラムを判断できません。

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 節に挿入する必要があります。

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、次のように処理されます。

- イベントハンドラコードが、ダイアログボックスの下部のペインに表示されます。
- 「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ている場合は、イベントに対して次のコードが表示されます。

  - 一般的なプログラム命令の注釈
  - 変更可能な各パラメータについて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。

注釈にイベントハンドラコードを挿入します。

* ここにイベントハンドラコードを追加します。

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含まれています。

このコードを挿入するには、次の操作を実行します。

1. Net Express 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化、イベントのコールバック、または、簡単な実行時の操作に使用できます。
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、選択した関数に必要なすべてのパラメータと戻り値が含まれています。

まとめ

ActiveX コントロール用のプログラミングアシスタントを使用すると、ActiveX コントロールを理解してコードを作成するための作業を大幅に削減できます。

ユーザコントロール

インターフェイスでユーザコントロールを使用する場合は、条件に合わせてカスタマイズする必要があります。 ユーザコントロールは、次の手順でカスタマイズします。

- ユーザコントロールを作成し、Dialog System の属性を定義します。
- 必要な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します。
- コントロールプログラムをカスタマイズします。

ユーザコントロールの指定

ユーザコントロールオブジェクトを使用して、Net Express 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に追加できます。

ユーザコントロールを作成するには、次の手順を実行します。

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. ユーザコントロールを追加するウィンドウを選択します。

3. [オブジェクト] メニューの [ユーザコントロール] を選択するか、または、[オブジェクト] ツールバーのユーザコントロールをクリックします。

4. ユーザコントロールの位置とサイズを調整します。

   図 10-5 に示すような「ユーザコントロール」ダイアログボックスが表示されます。

図 10-5：「ユーザコントロールの属性」ダイアログボックス

5. 「ユーザ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. ユーザコントロールに名前を付けます。 MAIN-WINDOW-STATUS-BAR のように、コントロールの意味がわかる名前にします。

    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. ユーザコントロールプログラムの名前を指定します。

        Net Express 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ると、変更して必要な機能を設定できます。
    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. Net Express プロジェクトが開いており、Net Express 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] をクリックして、COBOL 原始プログラムを作成します。この原始プログラムは、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、開いたプロジェクトに追加されます。

        このプログラムがスクリーンセット内のユーザコントロール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。

ユーザコントロールのタイプ

どの場合でも、DialogSystem\Source ディレクトリの funcdata.imp ファイルに一覧表示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] メニューで [インポート]、[スクリーンセット] の順に選択し、開いたダイアログボックスでインポートするファイルとして funcdata.imp を選択します。

次では、生成されたプログラムをスクリーンセットとともに使用するための変更について詳しく説明します。

スピンボタン

COBOL コードを変更する必要はありません。 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実装するのみです。 その結果、ダイアログコードによって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から渡される新しい値に更新されます。

例:

```
USER-EVENT
    XIF=$EVENT-DATA 34580 UPDATE-MASTER
    UPDATE-MASTER
    * ユーザコントロールプログラムによって NUMERIC-VALUE フィールド
    * が更新される。 このコードでは必要なフィールドを更新する。
    MOVE NUMERIC-VALUE(1) MY-NUMERIC-FIELD
```

ステータスバー

COBOL コードを変更する必要はありません。 次の要素を実装するのみです。詳細については、このマニュアルの後の章を参照してください。

- ステータスバーのヒントテキスト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。詳細については、ヘルプを参照してください。
- 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。これらのイベントは、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。
- 通常の CALLOUT を有効にして、トグルキー (挿入 / 上書き、CAPS、Num Lock) の状態と現在のシステム時刻を最新の状態にするための 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』の章を参照してください。

ツリービュー

生成し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は、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適用します。

ツリービューコントロールプログラムを使用するには、作成したツリーコントロールに挿入するデータを ATVIEW-PARMS データブロックグループに適用します。 詳細については、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含まれています。 これらの例は、必要に応じてカスタマイズできます。

ツールバー

作成したツールバーコントロールプログラムを使用する場合は、まず、tbardata.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適用します。

ツールバーコントロールプログラムを使用するには、プログラムで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。カスタマイズするには、使用する構造体を定義する WORKING-STORAGE コピーファイルを 1 つ編集します。

次に、ユーザによるメニュー項目やボタンの選択に対して応答し、必要な動作を実行する (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ダイアログの実行など) ためのコードを作成します。

ユーザ定義

ユーザコントロールオブジェクトを作成する場合は、必要な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ルトンコントロールプログラムを作成できます。

まとめ

作成したプログラムは、変更しないで正常にコンパイルおよび実行できます。 アプリケーションで必要な追加機能を実行できるようにコードを変更できます。また、ここまでの説明のとおり、作成し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 インターフェイスを通じて、すべてのクラスライブラリコントロールにアクセスできるフルセットのコントロールプログラムを作成することです。 このプロセスは、今後にリリースされる Net Express の Dialog System で完成する予定です。

第 11 章 : 複数のスクリーンセット

ここでは、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。 Dsrunner の使用方法、および、呼び出しインターフェイスを使用してアプリケーションで複数のスクリーンセットを維持する方法について説明します。

複数のスクリーンセットを使用する場合は、次のどちらかの方法で Dialog System を呼び出します。

- Dsrunner を使用する

  この方法では、特に複数のスクリーンセットとモジュールを使用する Dialog System アプリケーションを開発している場合に、Dialog System を最も速く簡単に呼び出すことができます。
- Router プログラムと呼び出しインターフェイスを使用する (詳細については、以後で説明します)

Dsrunner

Dsrunner は、ほとんどのアプリケーションで使用できます。 Dsrunner を使用すると、Dialog System を最も簡単に呼び出せます。 また、複数のスクリーンセットと複数の副プログラムモジュールを使用するモジュール形式の Dialog System アプリケーションを開発できます。 ユーザはビジネスロジックとサポートする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 プログラムは、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できます。 Dsrunner は、必要に応じて副プログラムやスクリーンセットの切り換えを処理します。 そのため、複数のスクリーンセットや複数の副プログラムを切り換えるためのコードを作成する必要はありません。

Dsrunner のアーキテクチャ

Dsrunner は、アプリケーションを副プログラムとして実行するプログラムです。 Dsrunner を使用してスクリーンセットを起動すると、スクリーンセットのロード、および、関連する副プログラムの呼び出しも実行されます。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できます。ただし、データブロックを正しく設定しておく必要があります。

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は、プログラムとスクリーンセットの切り換え方法を指定し、イベントの発生時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する必要があります。 Dialog System には、切り替え方法を指定した 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、以後で説明します。

Dsrunner は、Router プログラムで実行されるすべての作業に加え、他の作業も実行されます。

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副プログラムも含まれています。 メイン (Dsrunner) スクリーンセットからアプリケーションの最初のウィンドウが表示されます。Dsrunner プログラムからは、他のスクリーンセットや副プログラムをロードするための関数が提供されます。

各スクリーンセットには、(オプションで) 副プログラムを関連付けることができます。 スクリーンセットと副プログラムを関連付けるには、両方に同じ名前 (ファイル拡張子を除く) を付けます。 たとえば、dsrnr.gs というスクリーンセットに関連付ける副プログラムには dsrnr.cbl という名前を付けます。

スクリーンセット、および、関連付けた副プログラムを起動すると、次の処理が実行されます。

1. Dsrunner によってスクリーンセットのデータブロックにメモリが割り当てられ、LOW-VALUES に初期化されます。

2. スクリーンセットを Dialog System にロードする前に副プログラムを呼び出します。

   これによって、必要な初期化が実行されます。
3. 初期化が終わると、副プログラムで EXIT PROGRAM が実行されます。

4. EXIT PROGRAM によって、Dsrunner からスクリーンセットがロードされます。

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。

   スクリーンセットで RETC 命令を実行した場合は、Dsrunner に戻ります。

6. 次に、Dsrunner の関数が要求されているかどうかが確認されます (有効な SIGNATURE とファンクションコードが指定されているかがチェックされます)。

   - 関数が要求されている場合は、Dsrunner でその関数が実行され、スクリーンセットに戻ります。

   - 関数が要求されていない場合 (通常の場合) は、Dsrunner で副プログラムが呼

び出されます。

7. 副プログラムは、次のように動作します。

    1. 要求された関数を実行します。

    2. データブロックを更新します。

    3. EXIT PROGRAM を実行して、Dsrunner に制御を戻します。

    Dsrunner からスクリーンセットに制御が戻ります。

パラメータ

副プログラムを呼び出す場合は、4 つのパラメータを渡します。これらのパラメータは、プログラムの Linkage Section に指定する必要があります。 これらのパラメータは、副プログラムのパラメータなので、プログラムの Working-Storage Section に指定しないでください。

4 つのパラメータは、次のとおりです。

- screenset-data-block (必須)

  screenset.cpb で提供されます。

  screenset-data-block は、スクリーンセットに対して提供されるデータブロックです。 副プログラムを初めて呼び出したときに、このデータブロックが LOW-VALUES に初期化されます。 Dsrunner では、データブロックのバージョン番号が確認されません。 そのため、スクリーンセットのデータブロックと副プログラムのデータブロックが同期するように、十分注意してください。
- dsrunner-info-block (オプション。無視できます。)

  dsrunner.cpy で提供されます。

- ds-event-block (オプション。無視できます。)

  dssysinf.cpy で提供されます。 このパラメータは、Dsgrun から返るパラメータとまったく同じです。
- ds-control-block (オプション。無視できます。)

  ds-cntrl.cpy で提供されます。Dsrunner が Dsgrun を呼び出す場合に使用されます。このパラメータには、Dsrunner が Dsgrun を呼び出すための値が格納されます。また、Dsrunner に戻るときに Dsgrun から返される値も格納されます。この値は、エラーコード、ウィンドウ名、オブジェクト名などです。 ds-control-block の詳細につ

いては、『スクリーンセットの使用方法』の章にある『制御ブロック』を参照してください。

副プログラムで ds-control-block の ds-clear-dialog フィールドと ds-procedure フィールドを修正すると、スクリーンセットの実行方法を変更できます。 他の入力フィールドは、Dsrunner の制御下にあり、変更できません。

Dsrunner のスクリーンセット

ユーザが特別な操作を実行しなくても、Dsrunner でスクリーンセットを実行できます。ただし、[オプション] メニューの [構成]、[スクリーンセット] の順に設定して、スクリーンセット識別子を定義する必要があります。 Dsrunner がスクリーンセットを切り換えるには、スクリーンセット識別子が必要です。

Dsrunner のスクリーンセットは、Dialog System に組み込まれており、dsrunner.gs が DialogSystem\bin サブディレクトリに保存されています。 このスクリーンセットは、変更可能なテンプレートです。また、ユーザが独自のスクリーンセットを作成できます。この場合は、データブロックを正しく設定し、必要なグローバルダイアログを作成する必要があります。

データブロックヘッダー

Dsrunner スクリーンセットのデータブロックの先頭には、次のフィールドを記述する必要があります。 これらのフィールドには、制御情報とディスパッチ情報が保存されます。

```
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

  シグネチャです。 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

  ファンクションコードです。
- DSRUNNER-RETURN-CODE

  戻りコードです。
- DSRUNNER-PARAM-NUMERIC と DSRUNNER-PARAM-STRING

数字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、起動することが可能です。 ただし、他のアプリケーションを起動できません。

共有メモリバッファを使用する場合は、DSRUNNER-DATA-ITEMS 集団の直後に次のフィールドを挿入する必要があります。

```
SHARED-MEMORY-BUFFER
  YOUR-DATA              any format or size
```

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予約しておく必要があります。 スクリーンセットのデータブロックの最初の部分に dsrunner.imp というファイルを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で次の処理を実行するための主要な文が指定されています。

- Dsrunner のシグナチャを設定する。
- 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条件

Dsrunner とともに使用するスクリーンセットは、次の条件を満たしている必要があります。

- ロードするスクリーンセットの一意のスクリーンセット識別子が指定されていること。
- 次のダイアログを使用してスクリーンセットおよび副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、どのイベントにも設定できます。 副プログラムを閉じるには、副プログラムに対して独自の終了フラグを設定し、RETC を実行してから、Dsrunner に対して終了フラグを設定し、RETC を実行します。
- 複数のスクリーンセットを処理する場合は、次のグローバルダイアログが指定されていること。

  ```
  OTHER-SCREENSET
    REPEAT-EVENT
  ```

RETC

Dsrunner のグローバルダイアログ

Dsrunner に組み込まれた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発生す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ついてスタックされます)。 RETC によって、Dsrunner が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。 このダイアログは、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)
```

このダイアログでは、このスクリーンセットのデータブロックを Dsrunner に対して設定することを示すシグナチャを設定します。 データブロックを設定しない場合は、Dsrunner ですべてのファンクションコードが無視され、RETC によって、スクリーンセットに関連付けられた副プログラムのみが呼び出されます。

```
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ための要求に対して、SET-EXIT-FLAG が実行されます。 この文によって、Dsrunner が RETC の後に終了します。副プログラム内で終了処理を実行する場合は、独自の終了フラグを設定し、RETC を実行してから、Dsrunner に対して終了フラグを設定し、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 に記録される
   MOVE "lnch" DSRUNNER-FUNCTION-CODE(1)
   RETC
```

OPEN-SCREENSET-ERROR

このダイアログは、Dsrunner の関数の実行方法を表しています。 この例では、ファイルリクエスタを示し、ファイル名を取得しています。 その後で、スクリーンセットを起動するための関数を実行しています。

Dsrunner のプログラムと関数

Dsrunner プログラムのデータブロックでは、最初のいくつかのフィールドに特定の関数が指定されています。 指定されている関数の例は、次のとおりです。

- 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
- デバッグするための Dialog System のトレースモードの切り替え
- 共有メモリ領域の作成と使用

関数の一覧表示については、ヘルプトピックの『Dsrunner の関数』を参照してください。

Dsrunner の関数の使用

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 を実行する前に、Dsrunner のファンクションコードを Dsrunner のデータブロック内に設定します。 たとえば、次のように記述します。

```
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。 この関数では、スクリーンセットインスタンスが返ります。これは、SAVED-SS-INSTANCE として保存されます。

コマンド行を使用し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 [screenset-name /l /d screenset-name]

詳細は、次のとおりです。

screenset-name は、最初にロードするスクリーンセットの名前です。 このスクリーンセット名は、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」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 によって、スクリーンセット Animator がすぐに使用できるようになります。つまり、スクリーンセット Animator は、最初のスクリーンセットにあるダイアログの最初の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化できるように、Dsrunner は、.obj 形式と .gnt 形式の両方で提供されます。

Net Express IDE でのスクリーンセットの起動

スクリーンセットは、Net Express IDE で Dsrunner から起動することもできます。

1. [アニメート] メニューの [設定] を選択します。
2. 「アニメート開始位置」に「dsrunner」を指定します。
3. 「コマンド行パラメータ」に上記のオプション (/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「デバッグ / 編集」ウィンドウにロードされます。

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、最初のスクリーンセット名をコマンド行ではなく、プログラム内で指定できます。 この機能は、アプリケーションで独自のコマンド行引数を使用する場合に便利です。 また、ライブラリを開く場合など、アプリケーション固有の初期化を実行する場合にも便利です。

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、次の操作を実行します。

1. Dsrunner ウィンドウの [ファイル] メニューから [開く] を選択します。
2. スクリーンセットを選択して [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、関連付けられたプログラムが実行されます。

この手順を繰り返し、別のスクリーンセットを選択して実行できます。

アプリケーションの起動

ここでは、Dsrunner のアーキテクチャと、Dsrunner を効率的に使用する方法について説明します。 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。 組み込みのスクリーンセットは、一例です。 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 で次の処理が実行されます。

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 によって、スクリーンセットが正しい (つまりスクリーンセット識別子が指定されている) かがチェックされます。

    - スクリーンセットが正しくない場合は、そのことを示すメッセージボックスが表示されます。

    - スクリーンセットが正しい場合は、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確保され、LOW-VALUES に初期化されます。

3. スクリーンセット名と同じルートファイル名 (およびパス) をもつ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は、そのことを示す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化などの必要な初期化を実行するために、副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

        この情報は dsrunner.cpy で提供され、次の情報が含まれています。

        - screenset-id

        - ds-session-id

        - screenset-instance-number
        - エラーコード
    - Ds-Event-Block

5. 副プログラムが初期化されると、戻りコード「ゼロ」で Dsrunner に戻る必要があります。
    - ゼロ以外の戻りコードで戻った場合は、Dsrunner によってメッセージボックスが開き、エラーメッセージが表示されます。
    - ゼロの戻りコードで戻った場合は、Dsrunner によってスクリーンセットがロードされます (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 を呼び出した場合に、エラーが発生すると、副プログラムが呼び出されます。このときは、Dsrunner の情報ブロックに error-codes が指定されます。
6. スクリーンセットで RETC を実行すると、副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、スクリーンセットのダイアログに SET-EXIT-FLAG を指定する必要があります。

サンプル副プログラムの実行

Dsrnr は、Dialog System に付属するサンプル副プログラムです。 このプログラムの起動方法とサンプルコードの主要な項目については、『サンプルプログラム』の章を参照してください。

複数のスクリーンセットと Router プログラム

Dialog System のランタイムシステムは、次のスクリーンセットを使用できるようにプログラムできます。

- 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

これらの機能を使用すると、次の操作が可能です。

- ユーザインターフェイ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側プログラムを設計する場合は、次の点に注意して、スクリーンセットの処理方法を詳細に検討する必要があります。

- 関数の分割方法、および、関数を複数のスクリーンセットにまとめるかどうか。
- セキュリティ上の理由から、データへのアクセス権に応じてユーザをグループ化し、そのユーザグループごとにスクリーンセットを作成するかどうか。
- 1 名のユーザが、データの表示、比較、または、編集を同時に実行できるように、スクリーンセットのインスタンスを複数使用できるようにするかどうか。

Dialog System の呼び出しインターフェイスを使用して画面を制御するための基本的な情報については、ヘルプトピックの『呼び出しインターフェイス』を参照してください。

複数のスクリーンセットの使用

複数のスクリーンセットを使用するには、スクリーンセットスタックにプッシュしてポップします。 原則として、FIFO (先入れ後出し) の方式をとっています。スクリーンセットのプッシュとポップは、次の場合に便利です。

- 特定の機能に使用するスクリーンセットが不用になった場合に、このスクリーンセットを画面に表示させないことができます。
- 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してポップ (または使用) できます。
- 使用していないウィンドウや入力フォーカスがないウィンドウを画面に表示しないで、画面を見やすくできます。

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するための前提条件はありません。すべてのスクリーンセットやスクリーンセットのオカレンスをプッシュしてポップできます。 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。

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は、構成要素ごとにモジュールを作成し、各モジュールに独自のスクリーンセットを関連付けることをお勧めします。各構成要素に、専用のインターフェイスを設定すると、独立した COBOL プログラムになります。

たとえば、メインのデータエントリ構成要素と 2 つのユーティリティ構成要素 (一方は印刷関数、他方は管理関数を処理) をもつアプリケーションを構築していると仮定します。

スクリーンセットをスクリーンセットスタックに配置することによって、1 つのアプリケーション内で複数のスクリーンセットを使用できます。 このスタックは、プログラムを呼び出すときに使用す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指定して、Dialog System を呼び出すのみです。 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。

複数のプログラムにそれぞれスクリーンセットが関連付けられている場合は、イベントが発生し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 Router サンプルプログラムは、複数のプログラムとそれぞれに関連付けられたスクリーンセットを含むアプリケーションの構築方法を示しています。

用語と概念

Router プログラムを参照する前に、いくつかの用語と概念を理解する必要があります。

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は、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ックオブジェクトと、アクティブなスクリーンセットによって表示されるグラフィックオブジェクトの間には、表示上の相違はありません。 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。 一度にイベントを受け取ることができるスクリーンセットは、1 つのみです。

一般に複数のスクリーンセットを使用する場合は、Dialog System を呼び出し、ds-control に ds-use-set を設定して、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です。

他のスクリーンセットのイベント

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発生した場合は、現在のスクリーンセットで OTHER-SCREENSET イベントという特殊なイベントが発生します。 このイベントは、別のスクリーンセットに通知する必要があるイベントが発生したことを現在のスクリーンセットに通知するのみです。 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) 見つからない場合は、何も起こりません。 別のスクリーンセットに対して発生したイベントを現在のスクリーンセットで検出できるように、OTHER-SCREENSET を指定する必要があります。 このようなイベントが検出された場合は、適切なスクリーンセットをアクティブにします。 最も一般的な方法としては、OTHER-SCREENSET イベントが発生したことを示すフラグを設定し、呼び出し側プログラムに戻って、スクリーンセットを変更します。

アクティブではないスクリーンセットについて発生したすべてのイベントは、OTHER-SCREENSET イベントとしてアクティブなスクリーンセットに戻されます。

正しいスクリーンセットを識別するには、Dialog System を呼び出して、ds-event-block を指定します。 Dialog System が呼び出し側プログラムに戻ったときは、イベントが発生したスクリーンセットのスクリーンセット識別子が ds-event-screenset-id に格納されます。 このスクリーンセット識別子は、[オプション] メニューの [構成]、[スクリーンセット] の順に選択して、指定します。 スクリーンセット識別子は、指定しなくてもかまいません。ただし、指定しない場合は、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できません。

正しいスクリーンセットがアクティブで、OTHER-SCREENSET イベントに REPEAT-EVENT ダイアログ関数を指定した場合は、元のイベントが繰り返されます。

注：繰り返されるイベントが OTHER-SCREENSET でない場合は、このイベントは、正しいスクリーンセットがアクティブな場合に発生するイベントです。

Router を使用した複数のスクリーンセットのサンプルアプリケーション

Router を使用して Dialog System で複数のスクリーンセットを処理する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保存されています。 この例では、メインプログラム Programa が副プログラム Programb と Programc を呼び出します。 各プログラムには独自のスクリーンセットがあります。

4 番目のプログラムの Router では、ルーティング関数を処理します。 このプログラムの唯一の目的は、次に実行するプログラム (およびスクリーンセット) を決定し、そのプログラムを呼び出すことです。

アプリケーションの構造を図 11-1 に示します。

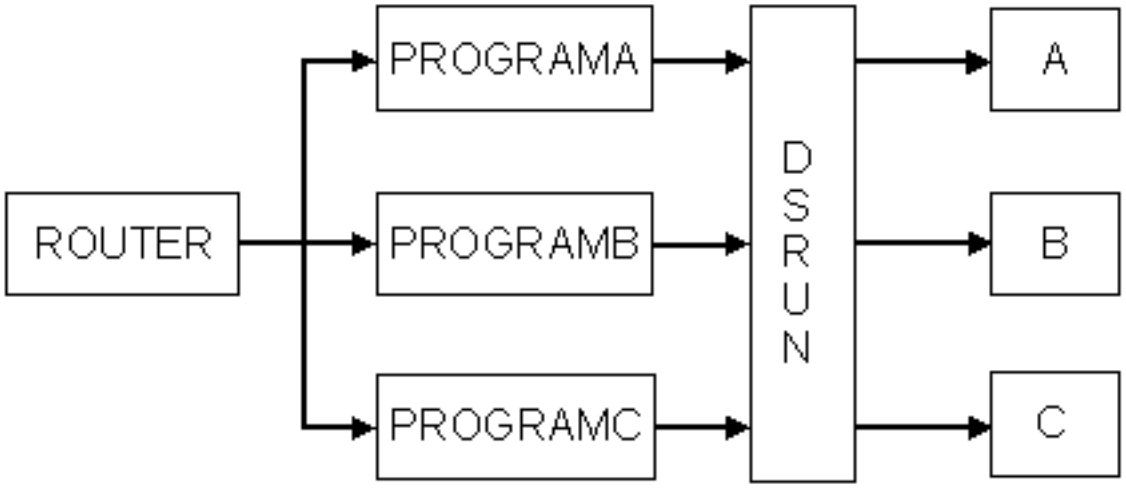

図 11-1： Router アプリケーションの構造

スクリーンセットの複数のインスタンスを使用する方法

複数のスクリーンセットを使用できるのみでなく、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同様に使用します。 実際の相違は、スクリーンセットの識別です。 ここを読む前に、『複数のプログラムとスクリーンセットの使用』を参照してください。

スクリーンセットの複数のインスタンスは、次のような場合に使用します。

- 1 つのウィンドウとそのウィンドウに関連付けられたコントロールを含むスクリーンセットを使用する場合
- 各集団項目が同じ形式のデータ集団を操作する場合。 同じスクリーンセットのインスタンスを複数使用すると、必要に応じて、1 つのスクリーンセットで集団項目を表示、比較、または、更新できます。

同じスクリーンセットのインスタンスを複数使用する場合は、プログラムで次の処理を実行する必要があります。

- スクリーンセットのインスタンスの数をトレースする
- スクリーンセットのインスタンスについて適切なデータブロックを Dsgrun に渡す

複数のインスタンスを使用する場合は、まず「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、次の手順を実行します。

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 に ds-push-set を指定します。

3. Dialog System が戻ると、ds-screenset-instance 制御ブロックフィールドに割り当てられたインスタンスの値が配置されます。

   インスタンスの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のみです。

インスタンスの値は、特定のスクリーンセットのインスタンスに一意なものです。 インスタンスの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の値をトレースする必要があります。

アクティブなインスタンスの値のトレース

dssysinf.cpy ファイル内の ds-event-screenset-id と ds-event-screenset-instance-no を調べると、アクティブなインスタンスをトレースできます。このインスタンス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。 dssysinf.cpy の詳細については、『Panels V2 の使用方法』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、次のように操作します。

- 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、次のように操作します。

- ds-event-screenset-instance を調べます。 このパラメータには、適切なインスタンスの値が格納されています。

特定のインスタンス値で Dialog System を呼び出すには、次のように操作します。

1. ds-event-screenset-instance-no を ds-instance に転記します。

2. ds-event-screenset-id を ds-set-name に転記して、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：

- 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] の順に選択して、スクリーンセット識別子をスクリーンセットの名前に設定する必要があります。

- 定義ソフトウェアを使用してスクリーンセット Animator からスクリーンセットを実行する場合は、スクリーンセットのインスタンスを複数使用できません。 アプリケーションから Dsgrun を呼び出す場合は、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 Animator の詳細については、ヘルプの『スクリーンセット Animator』を参照してください。

正しいデータブロックの使用方法

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は、データブロックの複数のコピーを用意する必要があります。 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は、次のようなコードを使用するのが最も簡単です。

  copy "program.cpb"

  replacing data-block-a by data-block-b
- データブロックのヒープまたはスタックを実装します。

  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くには、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 によって、値「S」が ds-control に格納されます。 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。

  スクリーンセットをスクリーンセットスタックからポップするには、次のどちらかを使用します。

  - 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

  既存のスクリーンセットを閉じずに、指定し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。

詳細については、ヘルプトピックの『スクリーンセット Animator』を参照してください。

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ーフェイスの使用を示す 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に関するデモンストレーションです。
- 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用方法を示しています。

これらのサンプルコードの主要な項目については、『サンプルプログラム』の章を参照してください。

Router プログラム

Router プログラムのロジックを説明する前に、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コピーファイルを説明します。

次のコードは、コピーファイル router.cpy を示しています。このファイルには、プログラム名と 2 つのフラグが指定されています。

program-name には、次に呼び出すプログラムの名前が指定されています。

cancel-on-return は、Router に戻る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) でキャンセルが要求された場合は、Router で exit-on-return が設定され、メイン実行ループが終了します。

```
 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 を呼び出すと起動します。 Programa で、特定の関数を処理するために副プログラムが必要と判断されると、副プログラムの名前が program-name に書き込まれ、終了します。 次に Router で program-name 内のプログラムが呼び出されます。

```
 29 main-section.
 30
 31*   すぐに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して起動する
 35    move main-program to program-name
 36
 37*   終了が要求されるまで、program-name 内のプログラムを呼び出す
 38    perform until exit-on-return
 39
 40*    呼び出したプログラムを記録する
 41     move program-name to dispatched-program
 42
 43*    Program-Name 内の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*      最後のプログラムでキャンセルが要求された場合は、キャンセルする
 47       set cancel-on-return to false
 48       cancel dispatched-program
 49       if dispatched-program not = main-program
 50*        副プログラムがキャンセルされた場合は、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*        メインプログラムでキャンセルが要求された場合は、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する場合は、プログラムが Router に戻る前に、cancel-on-return を設定する必要があります。 main-program に指定されたプログラムでキャンセルが要求された場合は、Router がメインプログラムをキャンセルした後に終了します。

メインプログラム

ここでは、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 つの副プログラム (Programb と Programc) はほとんど同じです。

31 ～ 44 行目：

```
 31 procedure division using program-control.
 32
 33 main-section.
 34     if new-instance
 35*    初めて呼び出した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*    初期化が終了したら、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 main-section では、スクリーンセットの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は、プログラムでそのインスタンスを使用します。作成されていない場合は、プログラムでインスタンスを作成します。

インスタンスが作成された後で、Dialog System を呼び出してそのインスタンスを表示します。 program-name 内のプログラムの名前が Programa である限り、Dialog System を呼び出します。

program-name 内のプログラム名が Programa ではない場合は、プログラムを終了し、Router に戻ります。 Router で、program-name 内のプログラムを呼び出します。

46 ～ 60 行目：

```
 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。

62 ～ 66 行目：

```
 62 use-set-instance.
 63*    スタック内の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*    呼び出し側プログラム
66     move this-program-name to ds-set-name.
```

作成されたスクリーンセットのインスタンスを使用するには、Dialog System にスクリーンセット名を通知し、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。

67 ～ 101 行目：

```
67 ...
68 call-ds.
69*    標準の初期化と呼び出し
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設定されている場合は、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は、このフラグが設定される
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は、このフラグが設定される
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*    このプログラム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 の呼び出しコードです。この呼び出しには、3 つのパラメータとその後に評価を指定しています。 評価では、スクリーンセットダイアログで設定し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 の終了、スクリーンセットとプログラムの切り換え、副プログラムの呼び出し、または、イベントの処理のどれかが実行されます。

このコードで重要なのは、OTHER-SET-TRUE フラグが設定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、イベントが発生したスクリーンセットのスクリーンセット識別子 ([オプション] メニューから [構成]、[スクリーンセット] の順に選択して設定) を ds-event-screenset-id に書き込みます。

この例ではスクリーンセットとそのスクリーンセットに関連付けられたプログラムに同じファイル名を使用しているので、Programa で正しいスクリーンセット識別子が検出されると、プログラム名も検出されることになります。 スクリーンセットとそのスクリーンセットに関連付けられたプログラムの名前が異なる場合は、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 で ds-screenset-name にスクリーンセット識別子を書き込み、終了します。 Router で、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。

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処理するためのスクリーンセットダイアログを示しています。 ここに一覧表示されたダイアログは、Programa スクリーンセットの一部ですが、他のスクリーンセットについても同じです。

Dialog System では、OTHER-SCREENSET を使用して、現在ロードされているスクリーンセット以外のスクリーンセットでイベントが発生したことを検出します。この例では、この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 フラグが設定され、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通知されます。

次に REPEAT-EVENT 関数が実行されます。 REPEAT-EVENT は、Dialog System に対して、次回の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 の後で、RETC によってスクリーンセットから呼び出し側プログラムに戻ります。

呼び出し側プログラム (ここでは Programa) でスクリーンセットのフラグをチェックできます。 フラグが設定されている場合は、プログラムによって ds-set-name 内にスクリーンセット識別子が書き込まれ (スクリーンセット名とプログラム名が同じであることを仮定しています。)、Router に戻ります。 Router で、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。 REPEAT-EVENT 関数は、Programa スクリーンセットによって実行されるため、OTHER-SCREENSET イベントの原因となったイベントが繰り返されます。

注：繰り返されるイベントが OTHER-SCREENSET でない場合は、このイベントは、正しいスクリーンセットがアクティブな場合に発生するイベントです。

これらの関数の詳細については、ヘルプトピックの『ダイアログ文：関数』を参照してください。

イベントの発生順序

ここまでは、スクリーンセット切り換えの全体的なプロセスと主要なイベントについて説明しました。 ただし、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。 通常は、制御が適切なプログラムに渡る前に、いくつかのプログラムが切り換えられます。

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。 あるグラフィックオブジェクトから別のグラフィックオブジェクトへ入力フォーカスが移動すると、2 つのイベントが発生します。 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、フォーカスが設定されるオブジェクトは 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は、そのオブジェクトが設定されたウィンドウに関する LOST-FOCUS イベントも発生します。 同様に、フォーカスが設定されるオブジェクトがコントロールオブジェクトの場合は、そのオブジェクトが設定されたウィンドウに関する GAINED-FOCUS イベントも発生します。

次の表は、「A」のウィンドウから「B」のウィンドウにフォーカスの移動があった場合に実際に発生するイベントの発生順序を示しています。

システムイベント	アクティブなスクリーンセット	イベントが発生したスクリーンセット	DS イベント
Mouse-button-1-down	A	B	OTHER-SCREENSET
Lost-focus(「A」ウィンドウ)	B	A	OTHER-SCREENSET
Gained-focus(「B」ウィンドウ)	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 から呼び出し側プログラムに戻ります。 次に、Dialog System と正しいスクリーンセット (ds-event-screenset-id によって識別) が呼び出されます。

「A」ウィンドウにフォーカスがあるときに「B」ウィンドウでマウスボタンを押すと、この表に示す順序でイベントが発生します。

「B」ウィンドウ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。 その結果、スクリーンセット「A」がスクリーンセット「B」に切り換えられます。

次のシステムイベントは、「A」ウィンドウがフォーカスを失うイベントです。 表を見るとわかるように、このイベントは「B」への切り換え後に発生します。これによって、「A」で OTHER-SCREENSET イベントが発生します。

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。 マウスボタンの解放イベントは、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すると、イベントの順序が問題になることはありません。これは、スクリーンセットのスワップが何度発生しても、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：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 関数を使用します。

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は、正しいスクリーンセットで繰り返す必要があります。 ただし、通常は、イベントが必要ないため、繰り返す必要がありません。

フォーカスが windowA の buttonA にあるときに、windowB の buttonB をクリックすると、次のイベントシーケンスが発生します。

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

この順序は、REPEAT-EVENT が必要でないことを示しています。Button-Clicked イベントによって BUTTON-SELECTED イベントが発生する前に、windowB の Gained-Focus が

ScreensetB をロードしているからです。

フォーカスの設定

Dialog System を呼び出し、ds-control に ds-use-set を指定して、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。 ユーザが (新しいウィンドウまたはコントロールを選択) スクリーンセットを切り換えた場合は、ユーザの操作によって GAINED-FOCUS イベントが発生するため、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発生する場合は、Dialog System を呼び出す前に ds-procedure にダイアログ手続き名を指定する必要があります。 SET-FOCUS ダイアログ関数を使用して、適切なウィンドウにフォーカスを設定します。

詳細情報

呼び出しインターフェイスの詳細については、ヘルプトピックの『呼び出しインターフェイス』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制御ブロック、データブロック、スクリーンセット Animator の使用方法、バージョン確認、呼び出し側プログラムが Dialog System に返す値について説明しています。

複数のスクリーンセット　Panels V2 の使用方法

第 12 章 : 異なるプラットフォームへの移行

このバージョンの Dialog System は、主に Windows 95 および Windows NT 向けです。

ここでは、これらの環境の主な相違について説明し、複数の環境で使用するスクリーンセットを処理するための指針を示します。

対応する以前のバージョンの Dialog System がある場合は、システムで適切な Dialog System ランタイムソフトウェアが使用されるので (呼び出し側プログラムは使用されません)、スクリーンセットを他の環境で実行できます。 一部のオブジェクト定義は、異なる環境間で移植できません。Dialog System では、移植性に関する警告がオプションで表示されます。 [オプション] メニューの [含める] から [移植性に関する警告] を選択すると、移植できないオブジェクトを作成しようとした場合に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。環境は絶えず変化しているので、Dialog System で移植に関する違反を検出できないことがあります。

環境間の相違

サポートされている各環境では、多くのオブジェクトやコントロールが表面的にはほとんど同様に表示されます。 プレゼンテーションマネージャと Windows の間で最も異なって表示されるのは、オプションボタンやチェックボックスなどのオブジェクトです。

Dialog System で作成されるオブジェクトは、現在の環境のオブジェクトと同様に表示されます。 プレゼンテーションマネージャで作成したアプリケーションでは、オプションボタンは楕円で囲まれて表示されます。このボタンを選択すると、楕円内に色が表示されます。 スクリーンセットを環境間で移動すると、オブジェクトの表示状態が変化します。

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固定できます。

オブジェクトやメニュー項目には、特定の環境でしか利用できないものがあります。たとえば、 OLE2 オブジェクトは、Windows でしか利用できません。

アプリケーションを移植するには、特定の環境に固有の関数を使用しないで、アプリケーションを実行する環境に共通の関数や機能を使用すると、すべての環境でアプリケーションが同様に表示および動作します。 アプリケーションを移植する必要ない場合は、特定の環境のみを対象に設計し、その環境に固有の機能を使用できます。

たとえば、32 ビットの Windows のインターフェイスを利用する場合は、COBOL システム

のクラスライブラリを使用して、Dialog System のインターフェイスを拡張できます。 ただし、そのインターフェイスを以前のバージョンの Dialog System に移植できません。

デスクトップモード

環境間の大きな相違の 1 つに Dialog System の起動方法があります。

プレゼンテーションマネージャ：
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。 デスクトップモードに切り替えた場合を除き、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。Windows では、メニューを使用して、クリップされたウィンドウを作成できないためです。 最初に作成するウィンドウが Dialog System ウィンドウ内にある場合は、このウィンドウがクリップされるので、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 でウィンドウ上のメニューバーを表示するには、デスクトップモードを使用する必要があります。 ただし、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。

グラフィック環境と 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、16 ビットバージョンの Dialog System 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の作成方法については、16 ビット製品のマニュアルを参照してください。

移植に関する指針

ここで示す指針は、主にグラフィカルインターフェイ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参照してください。

- 異なる解像度間の移植性を確保する場合は、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。 そのため、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。 その結果、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。

ただし、Dialog System では、実行時用に複数の解像度がサポートされています。 ヘルプトピックの『Dsgrun の呼び出し』と『呼び出しインターフェイス』、および『高度なトピック』の章を参照してください。

- 使用できる画面の領域に注意する必要があります。 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表示できますが、25 行の画面では、標準 VGA の解像度より多くの情報を表示できません。

- 環境によっては使用できないフォントがあります。 すべてのグラフィック環境で必ず使用できるフォントは、デフォルトのフォント、システムのモノスペースフォント、およびシステムのプロポーショナルフォントのみです (GUI エミュレーションでは、複数のフォントがサポートされません。このモードでは、システムのモノスペースフォントを使用します)。

- システムの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します。 ボタンは、まず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関係なく、ボタンテキスト全体が表示されます。

  ただし、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。 そのため、VGA では狭い間隔で横方向に配置されたボタンが、XGA では重なって表示されることがあります。 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。 ボタンテキストはクリップされることがあります。 (システムの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書体に変更すると正しく表示される場合があります。

  デフォルトのプッシュボタンの境界は、ボタンが定義された領域を囲むように描かれます。 環境によっては、この境界が他より強調されます。 この問題を解決するには、オブジェクトの間隔を十分にとります。

- 必ず、(「ボタンの属性」ダイアログボックスの) 「テキスト揃え」を使用してください。 ボタングループごとに、「テキスト揃え」を使用するか、または使用しないかを統一してください。 つまり、この属性をグループ内のすべてのボタンに対して選択する。または、まったく選択しないかのどちらかに統一します。

  たとえば、同じテキストを含む 2 つのプッシュボタンがあり、 どちらも、最初は「テキスト揃え」が選択されていると仮定します。 一方のボタンの「テキスト揃え」の選択を解除しても、両方のボタンのサイズに相違はありません。 両方とも同じサイズです。 ただし、この定義を保存して別の環境でロードすると、ほとんどの場合は、サイズが異なって表示されます。 実際に、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。

- フォントの移植性に注意します。 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。 スタイル名をもつフォントは、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できます。 このようなマップを行う場合は、Dialog System でマップ情報が保存されたバイナリファイルが検索されます。 このファイルをフォントサイドファイルと呼びます。 詳細については、ヘルプトピックの『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性を確保するには、「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。 このオプションを ds.cfg にデフォルトとして設定できます。 詳細については、ヘルプトピックの『Dialog System の概要』を参照してください。

異種環境に関するその他の注意点

すべて (3 種類) のグラフィック環境で使用するスクリーンセットを作成する場合は、[オプション] メニューの [含める] から [移植性に関する警告] を選択します。この選択によって、特定のタイプのオブジェクトを作成する場合に、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、(その警告を無視して) そのオブジェクトの作成を続けることができます。ただし、警告で指摘された環境でスクリーンセットを実行しようとすると、期待どおりに表示されません。 ほとんどの場合は、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直すことができます。

特に次の点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でメニューバーを表示できません。 プレゼンテーションマネージャではメニューバーを表示できます。
- Windows では、親ウィンドウが表示されない場合は、クリップされないウィンドウも表示されません。 ダイアログを作成する際は、このことに注意する必要があります。
- Windows では、クリップされないウィンドウにタイトルバーを表示する必要があります。
- Windows では、中央揃えと右揃えを使用できません。
- Windows では、プッシュボタンとスクロールバーに色を設定しても効果がありません。
- Windows では、タイトルバー付きのウィンドウに境界を表示する必要があります。
- 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エントリフィールドは、Windows 3.0 ではサポートされません

が、Windows 3.1 ではサポートされます。

- Windows では、境界がないプッシュボタンはサポートされません。

下位互換性の問題

次の 2 つの注意点があります。

- ノートブック
- コンテナ

ノートブック

タブコントロールオブジェクトは、以前のバージョンの Dialog System のノートブックコントロールオブジェクトに換わるオブジェクトです。 タブコントロールの機能は、ノートブックと似ています。ただし、ノートブックコントロールで使用できた次の機能は、タブコントロールでは使用できません。

- 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、変更できません。
- ステータス行のテキスト

  タブコントロールにはステータス行はありません。
- タブコントロールの形

  タブコントロールのタブは常に長方形です。

Windows 95 の基本リリースでは、タブの方向がサポートされません。 ただし、Microsoft Internet Explorer 3.0 以上には、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

れているので、タブの方向がサポートされます。 また、ノートブックと異なり、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は、正しく表示されないことがあります。

Dialog System では、以前のバージョンの Dialog System で作成されたノートブックオブジェクトの属性が適用されます。 ただし、新しいバージョンでサポートされなくなった属性は、定義ソフトウェアから編集できません。

コンテナ

Dialog System でコンテナを実装する場合は、リストビューと呼ばれる Windows コントロールが使用されます。 このコントロールでは、コンテナオブジェクトの機能のサブセットを使用できます。

リストビューコントロールの主な制約は、「リストビューの最初の列に指定できるのはアイコンのみで、他の列に指定できるのはテキストのみである」という点です。

また、次のコンテナの機能は廃止されました。

- 総合タイトル。

  リストビューには、総合タイトルを設定できません。 総合タイトルを指定しても、無視されます。
- リストビューの列のフォーマット。Windows によって定義されます。

  列ヘッダーまたはデータのフォーマットを制御するためのフラグ (整列、区切り文字、列幅の設定など) は、すべて無視されます。
- 差分イベント。差分イベントはサポートされません。

  差分イベントの設定は無視され、差分イベントは生成されません。
- 列の非表示。作成した列は、必ず表示されます。

  Dialog System 拡張機能「Dscnr」の Sc 関数および Hc 関数は無効です。
- フォント。フォントは設定できません。

新しい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を使用することをお勧めします。このオブジェクトを使用すると、リストビューオブジェクトを全体的に制御できます。

互換性の一覧表

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 および 95(16 ビット)	Windows NT および 95(32 ビット)
3D オブジェクト	×	○	×	○	○
ビットマップ	○	○	○	○	○
チェック ボックス	○	○	○	○	○
色	○(9)	○(9)	×	○(9)	○(9)
コンテナ	○	○	×	○	×(11)
ダイアログ ボックス	○	○	○	○	○
エントリフィールド	○	○	○	○	○
グループボックス	○	○	○	○	○
フォント	○(8)	○(8)	×	○(8)	○(8)
アイコン / ビットマップ ボタン	○	○	×	○	○
リストボックス	○	○	○	○	○
リストビュー	×	×	×	×	○
メニューバー	○	○	○	○	○
メッセージ ボックス	○	○(4)	○	○(4)	○(4)
複数行エントリフィールド	○	○(5)	○	○	○
一次ウィンドウ	○(1)	○(1)	○(1)	○(1)	○(1)
プッシュボタン	○	○(6)	○	○(6)	○(6)
オプションボタン	○	○	○	○	○
スクロールバー	○	○	○	○	○
(クリップされた) 二次ウィンドウ	○	○(3)	○	○(3)	○(3)

(クリップされていない) 二次ウィンドウ	○	○(2)	○	○(2)	○(2)
選択ボックス	○	○	○	○	○
タブコントロール	○	○	×	○	×(10)
タブコントロールページ	×	×	×	×	○(10)
テキスト	○	○	○(7)	○	○
ユーザコントロール	×	×	×	×	○

注：

- 「システム位置」は無効です。
- 黒、青、茶、シアン、緑、マゼンタ、赤、白、黄の各色は、すべての環境で共通です。

異なるプラットフォームへの移行 クライアント / サーバ結合の使用方法

第 13 章 : Panels V2 の使用方法

Dialog System は、Panels Version 2 (Panels V2) という技術に基づいて設計されています。

Dialog System で Panels V2 を使用すると、次の利点があります。

- Dialog System のみの場合と比べ、オブジェクトを細かく制御できます。

  たとえば、Panels V2 の呼び出しを使用すると、システムメニューをオフにできます。
- Dialog System でサポートされない Panels V2 の機能にアクセスできます。

  たとえば、定義機能ソフトウェアのラバーバンド技術は、Dialog System にはありませんが、Panels V2 では利用できます。

ここでは、Panels V2 を使用して Dialog System アプリケーションを拡張する方法について説明します。

Panels V2 の呼び出し

Panels V2 の呼び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
```

Panels V2 が Pan2win を呼び出し、Pan2win が適切な Windows API 呼び出しを実行します。

Panels V2 が提供する関数セットは、使用する環境に関係なくすべて同じです。ただし、各環境では表示機能がサポートされ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。

Dialog System と Panels V2 のイベント

Panels V2 への呼び出しが指定された標準的な Dialog System アプリケーションを図 13-1 に示します。

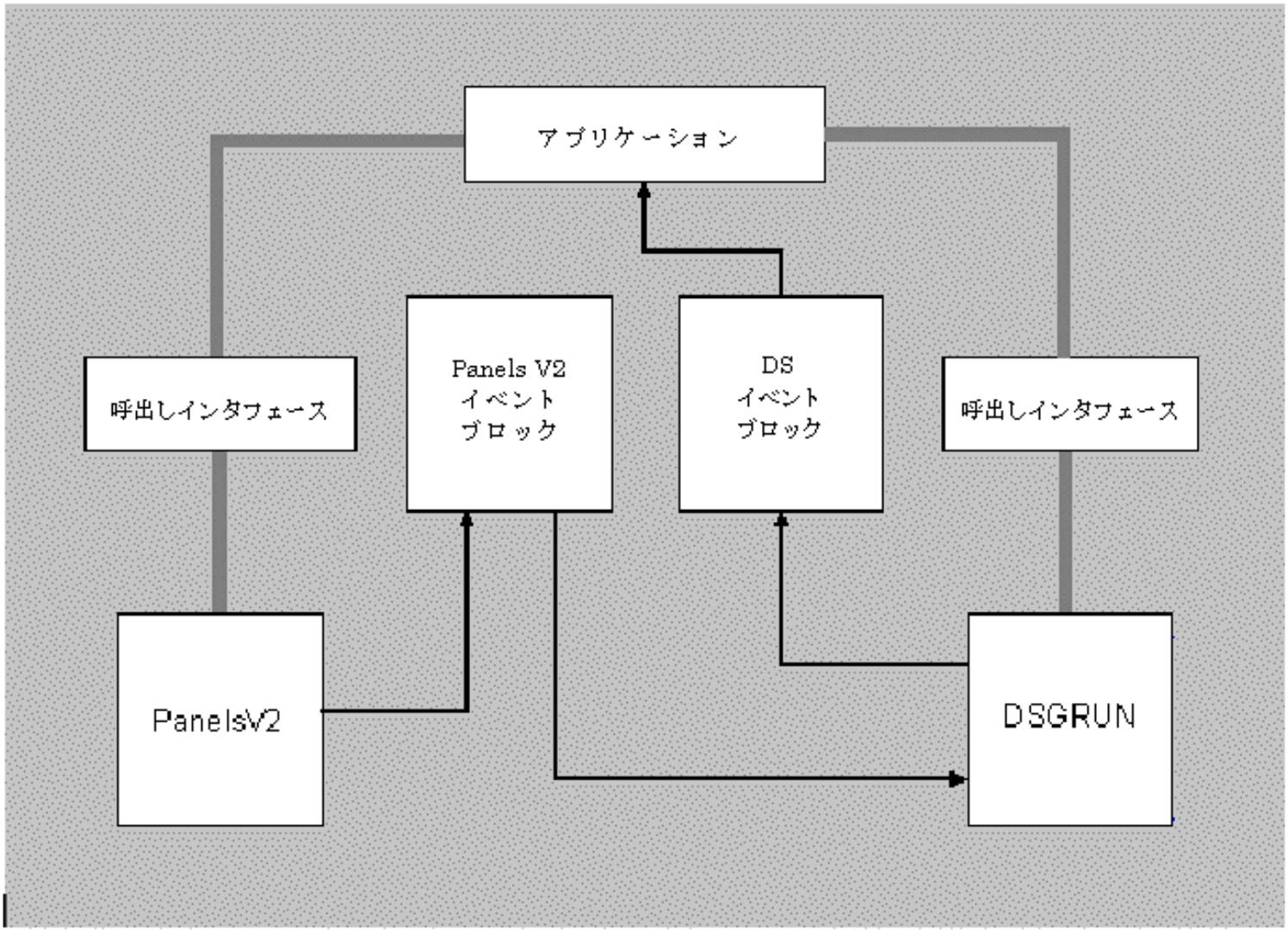

図 13-1： Dialog System と Panels V2 のイベント

イベントの処理方法に注意してください。Panels V2 とアプリケーションのイベントは、直接関係ありません。イベントが発生すると、Panels V2 でそのイベントが検出され、Dialog System (Dsgrun) を通じてイベント情報が返ります。 Dialog System のイベントブロック (dssysinf.cpy) を通じて Dialog System からそのイベント情報がプログラムに返ります。 Dialog System とアプリケーションのどちらでどのようにイベントを処理するかを決定するには、ダイアログを使用します。

コピーファイル

プログラムの Working-Storage Section には、コピーファイルを指定する必要があります。 たとえば、次のように記述します。

```
 working-storage section.

     copy "ds-cntrl.mf".
     copy "clip.cpb".
     copy "pan2link.cpy".
     copy "dssysinf.cpy".
```

最初の 2 つは、すでに説明した 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、ヘルプトピックの『呼び出しインターフェイス』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。

Panels V2 コピーファイル (pan2link.cpy)

COBOL プログラムには、pan2link.cpy コピーファイルも指定する必要があります。

このファイルには、次の内容が記述されています。

- レコード定義
- 定義済みのレベル-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、このファイル内の定義によって、アプリケーションの Panels V2 部分の記述を大幅に簡略化できます。

このファイルを参照して、内容を把握してください。

Dialog System のイベントブロック (dssysinf.cpy)

このファイルには、Dialog System のイベントブロックの定義が記述されています。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 の重要なフィールドは、ds-descendant と ds-ancestor です。Dialog System でオブジェクトが作成されると、ハンドルというオブジェクトに固有の識別子が返ります。オブジェクトでイベントが発生すると、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が Panels V2 から Dialog System に返ります。 これら 2 つのハンドルが ds-descendant と ds-ancestor で、イベントが発生したときに書き込まれます。

注：Dialog System V2.1 以降では、dssysinf.cpy の形式が変更されました。dssysinf.cpy を使用した Dialog System V2.1 のプログラムがある場合は、再コンパイルする必要があります。

Dialog System と Panels V2 アプリケーションの作成

Dialog System と Panels V2 のアプリケーションでは、次の手順を守る必要があります。

- Dialog System と Panels V2 が連動するための動作を確立します。
- Dialog System のオブジェクトを Panels V2 に指定します。
- Panels V2 の関数を実行します。

これらの手順について説明するために、以後では、プッシュボタンの名前を変更する簡単な Panels V2 関数の例を使用します。プッシュボタン名の変更は、ダイアログ関数 (SET-OBJECT-LABEL) を使用して実行することもできますが、この例では、Panels V2 の呼び出しインターフェイスの主な関数を使用します。

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 を初期化します。

```
call "dsgrun" using ds-control-block
                    data-block
                    ds-event-block
```

Dialog System から返される項目の 1 つに ds-session-id (制御ブロック内に格納) 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 と通信する場合に Dialog System と連動するには、この識別子を Panels V2 に通知する必要があります。 この操作には、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。

この文では、Dialog System と Panels V2 の間に必要な通信リンクが確立されます。その結果、Panels V2 呼び出しを直接実行できます。

Panels V2 への Dialog System オブジェクトの指定

MOVE-OBJECT-HANDLE 関数では、オブジェクトのハンドルをデータブロックの数字データ項目に保存できます。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
```

この文では、RENAME-PB というプッシュボタンのハンドルが数字項目 PB-HAND に格納され、RENAME-WIN というウィンドウのハンドルが WIND-HAND に格納されます。 RENAME-WIN と WIND-HAND は、データ定義で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 (および Panels V2) が Dialog System のオブジェクトにアクセスできます。

Panels V2 関数の実行

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.
```

1 行目：

```
 rename-button section.
```

この節は、Panels V2 の呼び出しインターフェイスです。

3 ～ 5 行目：

```
    initialize p2-parameter-block
```

```
    initialize p2g-button-record
    move 250                to p2g-button-text-length
```

Panels V2 パラメータを初期化します。

6 行目：

```
    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。

7 行目：

```
    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。 詳細については、『Dialog System と Panels V2 の通信の確立』を参照してください。

8 行目：

```
    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 関数です。 ここでは、タイトルを変更するのみで、残りの機能はすべて変更しません。

9 ～ 12 行目：

```
    call "PANELS2" using p2-parameter-block
                         p2g-button-record
                         text-buffer
    end-call
```

Panels V2 の呼び出し文です。

13 ～ 16 行目：

```
    perform varying ndx1 from 30 by -1 until ndx1 = 1 or
                    rename-text(ndx1:1) not = " "
       continue
    end-perform
```

この perform 文によって、ボタンの新しい名前が入っている rename-text から後続の空白文字を削除します。

17 行目：

```
    move ndx1               to p2g-button-text-length
```

後続の空白文字を削除した後は、ndx1 に名前の実際の長さが格納されます。p2g-button-text-length は、タイトルバッファの長さを格納する Panels V2 のパラメータです。

18 行目：

```
    move pf-set-button-details to p2-function
```

この Panels V2 関数では、プッシュボタンの属性を変更します。

19 ～ 22 行目：

```
call "PANELS2" using p2-parameter-block
                     p2g-button-record
                      rename-text
end-call.
```

別の Panels V2 呼び出しです。

警告：Panels V2 で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、Dialog System 内ですべてが管理されることを前提としています。たとえば、Panels V2 でオブジェクトを作成した場合は、そのハンドルは Dialog System に渡されません。 そのため、そのオブジェクトに関するイベントは Dialog System と連動しません。

サンプルプログラム

demo ディレクトリには、Dialog System と Panels V2 のインターフェイスを示す詳しい例が、他に 2 つあります。Clip は、Panels V2 のクリップボードの基本的な読み取り機能と書き込み機能を Dialog System アプリケーションでプログラムする方法を示しています。 Dsp2demo は、クリップボード機能、色の設定、および、ウィンドウのスクロールを示しています。

Panels V2 ユーザイベント

Panels V2 のユーザイベントを Dialog System で生成したり、受け取ったりできます。 詳細は、ヘルプの POST-USER-EVENT 関数と GET-USER-EVENT-DATA 関数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 用に予約されています。

Panels V2 の使用方法 高度なトピック

第 14 章 : クライアント / サーバ結合の使用方法

ここでは、クライアント / サーバ結合の動作と、ユーザのプログラムを汎用クライアント / サーバモジュールに結合する方法について説明します。

はじめに

クライアント / サーバ結合を実行すると、Micro Focus Common Communications Interface 構成要素 (CCI) によって簡単に Dialog System をフロントエンドとするクライアント / サーバアーキテクチャを実現できます。 クライアント / サーバ結合では、mfclient と mfserver という 2 つのモジュールを使用するので、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。 この 2 つのモジュールはあらゆるアプリケーションで利用できる汎用モジュールです。

これらのモジュールでは、構成ファイル内の情報に基づいて、モジュール間の通信とデータ転送が管理されます。また、リンクの両端でユーザ定義のプログラムと対話してこのデータが処理されます。mfclient は、ユーザインターフェイスを処理するユーザクライアントプログラムから呼び出されます。mfserver は、データアクセスやビジネスロジックを処理するユーザサーバプログラムを呼び出します。 ただし、これらのユーザクライアントとサーバプログラムはユーザが定義するので、どのような操作でも実行できます。

ユーザプログラムでは、mfclisrv.cpy コピーファイル (『mfclisrv.cpy コピーファイル』を参照) を使用して汎用クライアント / サーバモジュールと対話する必要があります。

Dialog System CUSTOMER デモンストレーションに基づく 2 階層のサンプルアプリケーションが、インストールディレクトリの DialogSystem\demo\csbind ディレクトリに格納されています。 このサンプルアプリケーションでは、クライアント / サーバ結合の使用方法が説明されています。このサンプルは、この全体で参照します。 サンプルアプリケーションの詳細については、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。

クライアント / サーバ結合の動作

ここでは、クライアント / サーバ結合を使用して作成した 2 階層の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 / サーバ結合を実行するには、サーバ側で mfserver の非専用コピーを実行します。mfserver は、取り決めたサーバ名を使用してすべてのクライアントと通信します。 この 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のみなので、最初のデフォルト値のまま使用できます。

このプロセスについては、次の図 (図 14-1) を参照してください。 この図で説明する情報の流れは、次のとおりです。

1. ユーザのクライアントプログラム (custint) が mfclient を呼び出します。
2. mfclient を最初に呼び出したときに .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 が各クライアントに対してセカンダリサーバを生成します。 サーバ名は、内部的に生成されます。サーバ名は、ベースサーバ名に数字 ID を追加したもの (mfserver01 など) となります。 また、構成ファイルにサーバ名を指定しておくこともできます。
5. mfserver がセカンダリサーバ名 (mfserver01) を mfclient に送り返し、この対話が終了します。
6. mfclient が、セカンダリサーバ (mfserver01) に接続し、LNK-PARAM-BLOCK を通じて構成ファイルから取得したパラメータを渡します。 『クライアント / サーバ結合の構成ファイル』を参照してください。
7. mfserver01 がユーザのサーバプログラム (custdata) を呼び出し、Linkage Section を通じて mfclient から受け取っ

たパラメータを渡します。

8. ユーザが作成したサーバプログラム (custdata) が最初に呼び出された時点で、アプリケーション初期化コードが実行され、プログラムが終了します。 制御が mfserver01 に戻ります。

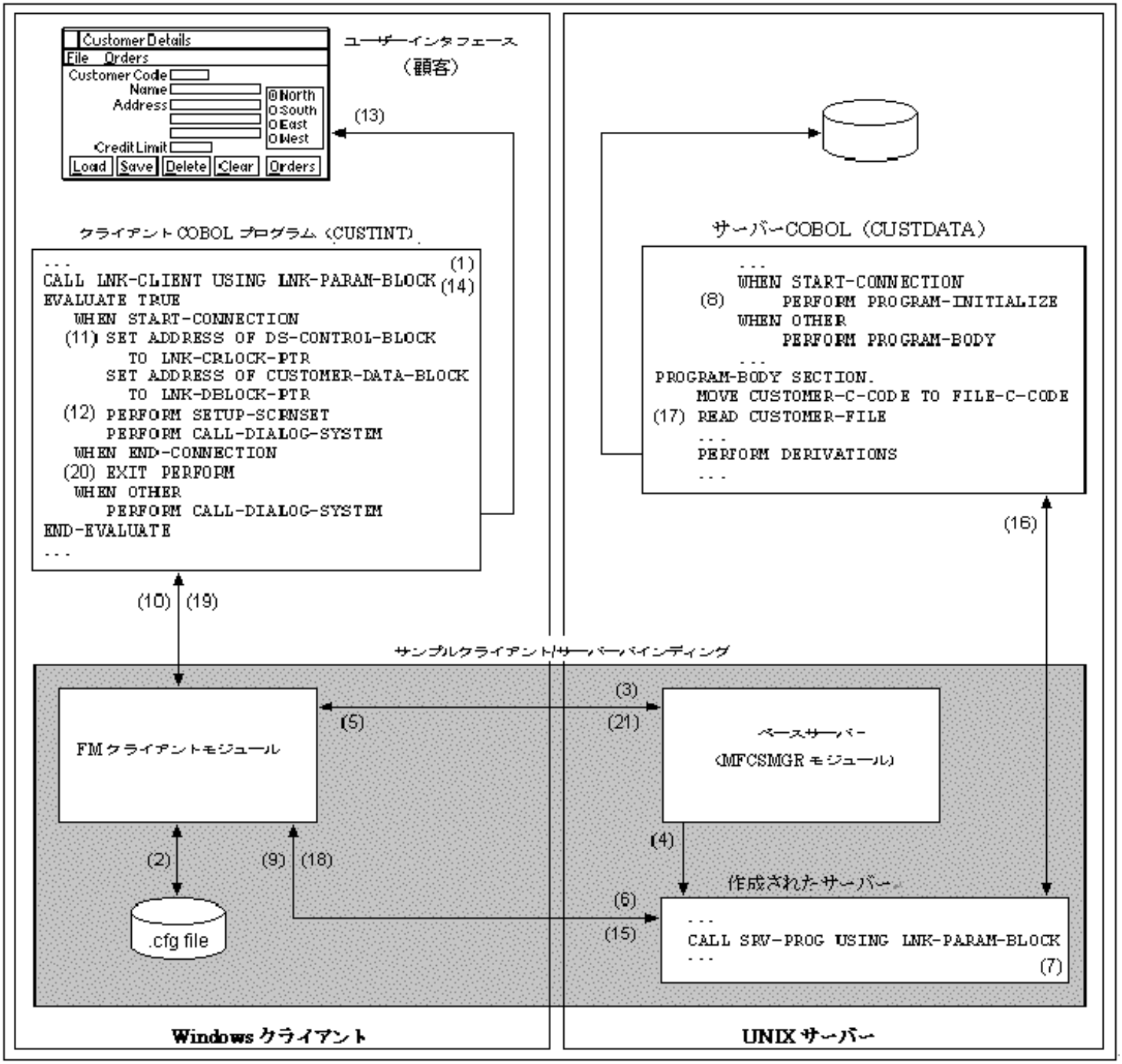

図 14-1： クライアント / サーバ結合

9. mfserver01 は、mfclient に制御を戻します。

10. mfclient が、接続が確立したことを確認し、ユーザのクライアントプログラム (custint) に制御を戻します。

11. ユーザのクライアントプログラム (custint) で、ユーザインターフェイスに関連付けられたデータが、Linkage Section 内の 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. ユーザのクライアントプログラム (custint) は、ユーザインターフェイスを呼び出し、スクリーンセットを設定し、Dialog System を呼び出します。

13. ユーザがユーザインターフェイス (CUSTOMER) に入力します。

14. ユーザのクライアントプログラムが再度 mfclient を呼び出して、ユーザが入力したデータ (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 がユーザのサーバプログラム (custdata) を呼び出します。

17. ユーザのサーバプログラム (custdata) が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 に制御を戻します。

19. mfclient は、Linkage Section を通じてユーザのクライアントプログラム (custint) にデータを送り返します。

20. ユーザのクライアントプログラム (custint) は、データを表示し、新しいユーザ入力を受け付けるためにユーザインターフェイスを呼び出します。

    ユーザがアプリケーションを終了するまで、手順 14 ～ 20 を繰り返します。

21. セカンダリサーバ (mfserver01) が終了したことを、mfclient がベースサーバ (mfserver) に通知します。

    ベースサーバとセカンダリサーバを使用することによって、次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を用意する必要がありません。 セカンダリサーバには、クライアントのクライアント構成ファイルから詳細が提供されます。 ユーザは複数のサーバを設定できますが、その必要はありません。

    - データアクセスの競合が発生しません。ユーザが各自の実行単位を使用できるので、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ためです。

    - 処理に時間がかかる要求をユーザが実行した場合に 処理が遅延するのは、そのユーザのみです。 他のユーザは、引き続き最適な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。 この状態を解決するには、次の操作を実行します。

      プログラムに要求 id を返すような非同期要求を実行します。 次に、返された id を使用して要求が完了したことを定期的にチェックします。 非同期要求の例については、『クライアントプログラムと mfclient の接続』を参照してください。

ユーザプログラムと汎用モジュールの接続

ここでは、次の方法について説明します。

- ユーザプログラムを汎用クライアントモジュールおよび汎用サーバモジュールと接続する
- 通信リンクを準備する

ユーザのクライアントアプリケーションと mfclient の接続

ここでは、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との通信を処理します。 mfclient モジュールと mfserver モジュールは、mfclisrv.cpy コピー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。

また、これらのモジュールは、構成ファイルで呼び出すように要求されているユーザプログラムに情報を渡す場合にも、同じパラメータブロックを使用します。 LNK-PARAM-BLOCK の詳細については、『サーバアプリケーションと mfserver の接続』を参照してください。

ユーザインターフェイ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する必要があります。

クライアント / サーバ結合を使用するには、クライアントプログラムに次のようなコードを追加する必要があります。このコードは、Dialog System に付属しているユーザインターフェイスプログラム (custint.cbl) の一部です。このコードは、クライアント / サーバ結合の使用方法を表しています。

```
$SET ANS85
 WORKING-STORAGE SECTION.
 COPY "MFCLISRV.CPY".
 LINKAGE SECTION.
* 次の 2 つの コピーファイルは、インターフェイスを呼び出す
* COBOL プログラムと通信するために、Dialog System の
* ユーザインターフェイスで使用されます
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.
```

コードを追加して、クライアントカウント、ユーザによるエラーメッセージ表示の処理、非同期要求の処理などを実行できます。 詳細については、『クライアントプログラムと mfclient の接続』と『サーバプログラムと mfserver の接続』を参照してください。

サーバアプリケーションと mfserver の接続

ここでは、サーバプログラムを接続する方法について説明します。サーバプログラムは、クライアントとの通信を処理する 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 コピー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。 また、これらのモジュールは、構成ファイルで呼び出すように要求されているユーザプログラムに情報を渡す場合にも、同じパラメータブロックを使用します。

次の説明は、既存のアプリケーションをサーバ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 / サーバ結合の使用方法については、『クライアント / サーバ結合アプリケーションの実行』を参照してください。

クライアント / サーバ結合を使用するには、次の条件を満たす必要があります。

- サーバプログラムの Linkage Section に、mfclisrv.cpy 結合コピーファイル、およびクライアントプログラムとユ

ーザインターフェイスの間で情報を渡すのに必要なコピーファイルを指定する必要があります。

- 手続き部の見出しを変更して「LNK-PARAM-BLOCK」を含めます。「LNK-PARAM-BLOCK」は、mfserver からパラメータを渡します。

- mfclisrv.cpy の mfserver から渡されたアドレスとユーザインターフェイスで使用するコピーファイルを関連付けます。この例の DS-CONTROL-BLOCK と SCREENSET-DATA-BLOCK は、ds-cntrl.cpy と customer.cpb で定義したデータ構造体です。 LNK-PARAM-BLOCK の詳細については、『サーバプログラムと mfserver の接続』を参照してください。

次のコードは、クライアント / サーバ結合の実行に使用するサーバプログラム (custdata.cbl) の該当部分です。

```
 LINKAGE SECTION.
 COPY "DS-CNTRL.V1".
 COPY "CUSTOMER.CPB".
 COPY "MFCLISRV.CPY".
 PROCEDURE DIVISION USING LNK-PARAM-BLOCK.
 CONTROLLING SECTION.
*-------------------------------------------------------*
*  DIALOG SYSTEM のコピーブックを 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処理するためのコードを追加できます。 詳細については、『サーバプログラムと mfserver の接続』を参照してください。

通信リンクの準備

クライアント / サーバ結合は、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プログラムの間の通信を制御します。 そのため、ユーザ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、適切な構成ファイル (.cfg) を変更しないで使用します。 ユーザアプリケーションの場合は、構成ファイルの内容をコピーして変更する必要があります。 構成ファイルは、次のような主要な情報を指定するためのパラメータが記述された ASCII テキストファイルです。

- mfserver によって呼び出すユーザプログラムの名前

- サーバと接続できるクライアントの最大数

クライアント / サーバ結合での構成ファイルの使用方法については、『クライアント / サーバ結合の構成ファイル』を参照してください。

注：

- ここでは、CCI を使用し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ており、クライアントマシンとサーバマシンの両方で動作していることを前提としています。
- Micro Focus の CCI でサポートされるベンダのネットワークソフトウェアを使用する必要があります。 サポートされるベンダのソフトウェアについては、Micro Focus の Web サイトの製品情報または Micro Focus のソフトウェアのマニュアルを参照してください。または、Micro Focus の営業担当者に照会してください。

クライアント / サーバ結合の使用前の注意

クライアント / サーバ結合は、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 / サーバアーキテクチャのアプリケーションと使用できます。 既存のスタンドアロンアプリケーションの使用方法については、『クライアント / サーバ結合アプリケーションの実行』を参照してください。

どちらの場合も、アプリケーションには次のプログラムを記述する必要があります (Dialog System に付属のデモンストレーションプログラムは、次のとおりです)。

	2 階層デモプログラム	スタンドアロンデモプログラム
ユーザインターフェイス	CUSTOMER.gs (GUI)	
インターフェイスを呼び出す 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 / サーバ結合を使用できます。 その場合は、次の手順を実行する必要があります。

- mfclient モジュールと mfserver モジュールの動作と通信接続を制御するための構成ファイル (.cfg) を作成します。

  構成ファイルは、次のような主要な情報を指定するためのパラメータが記述された 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 / サーバ結合の構成ファイル』を参照してください。 サンプルの構成ファイル (custgui.cfg と customer.cfg) は、クライアント / サーバ結合デモンストレーションプログラムとともに組み込まれています。

  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も、1 つの構成ファイルに複数のエントリを指定することもできます。 複数のエントリを使用する場合は、各エントリに固有のタグを指定し、その後にそのタグに関連するパラメータのリストを続けます。 クライアントインターフェイスプログラムでは、lnk-tagname にこのタグ名を指定する必要があります。 この方法を使用して、mfclisrv.cfg ファイル (他の名前を指定しない場合にこのファイル内のエントリが検索されます) 内に複数のエントリを指定すると、コマンド行を短くできます。 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 モジュールと連絡パラメータを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。 詳細については、『クライアントアプリケーションと mfclient の接続』を参照してください。
- mfserver モジュールから連絡パラメータとともにサーバプログラムを呼び出すためのコードをサーバプログラム

に追加します。 詳細については、『サーバアプリケーションと mfserver の接続』を参照してください。

mfclisrv.cpy コピーファイル

mfclient モジュールと mfserver モジュールは、mfclisrv.cpy コピー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。 この情報には、Dialog System の screensetdata-block と ds-control-block のアドレスなどが含まれます。 これらのモジュールでは、構成ファイルで呼び出すように指定されているユーザプログラムに情報を渡す場合にも、このパラメータブロックを使用します。 mfclisrv.cpy コピーファイルは、mfclient モジュールと mfserver モジュールを使用するすべての COBOL プログラムに指定する必要があります。

mfclisrv.cpy コピーファイルは、Net Express の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。

クライアント / サーバ結合の構成ファイル

ここでは、クライアント / サーバ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 モジュールと mfserver 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数の構成ファイルを作成できますが、各ファイルの拡張子を .cfg にする必要があります。 構成ファイル名を特に指定しない場合は、デフォルトの構成ファイル名 mfclisrv.cfg が使用されます。 構成ファイルが必要ないために指定しない場合は、クライアント / サーバ結合の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ると、それらを読み込んだプログラムが終了し、無効なエントリを表すメッセージが画面に表示されます。

このようなエラーは、mfclisrv.log ファイルにも記録されます。このファイルは現在のディレクトリに作成されるか、または、MFLOGDIR 環境変数で指定されたディレクトリに作成されます。 そのため、端末が直接つながっていないサーバでもエラーメッセージを記録できます。

構成ファイルのエントリ

mfclisrv.cfg 構成ファイルのエントリは、次のとおりです。 エントリをアルファベット順に説明します。 構成ファイルに指定し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 のかわりに通信エラーを処理するプログラムの名前。 呼び出し側プログラムで mfclient
                        エラーを処理するには、「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)。 通信製品を一切使用しないで 1 台の PC でテストを行えるような 2 階層のアプリケーションを作成するには、特殊なエントリ「NONE」を指定します。

データは、mfserver と通信要求をバイパスして、mfclient とユーザのサーバプログラムの間で直接送受信されます。

注:ユーザのサーバ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 の場合は、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は、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以上になります。 スクリーンセットで使用するデータフィールドが変更され、コピーファイルが再生成された場合は、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は、このエントリを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：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は、予期し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が存在するマシンの名前。 このエントリを指定すると、定義した名前と一致する最初のサーバが検索されるのを防ぐことができます。

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」のサイズに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]

階層間ソリューションの一環として mfserver によって呼び出された mfclientで使用す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、ローカルの 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、別のサーバを検索して通信します。 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 を変更できます。

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に有効です。 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) に設定されている場合にプロトコルを指定しないと、クライアント / サーバ結合でデフォルトの TCP/IP (CCITC32) が使用されます。

scrntype=xxxx PIC X(4) [default:none]

複数のインターフェイスタイプがサポートされている場合 (文字制御ブロックを使用して GUI とキャラクタインターフェイスの両方を実行している場合) に必要なインターフェイス番号。 scrntype はクライアント / サーバ結合モジュールによって妥当性検査されません。

servername=xxx PIC X(14) [default:MFCLISRV]

通信で使用するサーバ名。

setenv=name value PIC X(148) [default:none]

すべてのサーバプログラムを実行する前に設定する環境変数。 形式は、次のとおりです。

variable name PIC X(20) variable value PIC X(128)

名前と値のフィールドは 1 つ以上の空白文字で区切ります。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、サーバ上でユーザプログラムをアニメートできます。 そのため、mfserver をいったん停止して、COBSW=+A によっ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 と指定すると、ユーザプログラムのクロスセッションをアニメートできます。 詳細については、『アプリケーションのアニメート』を参照してください。

srverrprog=xxxxxx PIC X(128) [default:none]

mfserver にかわって通信エラーを処理するプログラムの名前。 「SAME」と指定すると、mfserver のエラーのための srvprog エントリで指定した名前が使用されます。

srvprog=xxxxxx PIC X(128) [default:custdata]

mfserver が呼び出すユーザプログラムの名前。

srvtier=xxxxxx PIC X(128) [default:mfserver]

サーバ階層プログラムの名前。 このエントリを指定すると、mfserver がこのプログラムから呼び出されます。

subserver=xxxx PIC X(14) [default:none]

メインサーバとの最初の通信後に、最初のサーバ名に基づく名前のかわりに通信に使用されるサーバの基本名。 たとえば、デフォルトのサーバ名は MFCLISRV なので、サブサーバ名は 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、ベースサーバ名を変えずにアプリケーション固有のサブサーバ名を使用できます。

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かわりに使用する値。 タイムアウト値は 1/10 秒単位で指定します。 実行済みの呼び出しについては、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 に設定した場合は、次の詳細がクライアントとサーバの両方に記録されます。

日付、時刻、「接続 / 切断」インジケータ、サーバ名、マシン名、プロトコル

構成ファイルに最低限必要なエントリ

構成ファイルに最低限必要なエントリの数は、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 callout)
- サーバの数
- 通信プロトコル (例 CCITCP)

ダイアログアプリケーションが既存の Dialog System プログラムをサーバとして使用する場合と使用しない場合に必要なエントリを次の表に示します。

	Dialog System プログラムをサーバとして使用しない場合	Dialog System プログラムをサーバとして使用する場合
dblksize	データブロックのサイズ。	データブロックのサイズ。
srvprog	サーバ側の処理を行うために mfserver が呼び出すプログラムの名前。 これは、サーバマシン上のユーザプログラム (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 コピーブックを使用しない場合)。	Dialog System の制御ブロックのサイズ (ds-cntr.r1 コピーブックを使用しない場合)。
Protocol		使用するプロトコル。 たとえば、TCP の場合は CCITC32。

複数のサーバを実行する場合は、servername エントリを通じて使用されるサーバの名前を指定する必要があります。

設定しないエントリには、デフォルト値が使用されます。

TCP/IP を使用しない場合は、プロトコルエントリを設定する必要があります。

構成ファイルの検索

構成ファイルの検索先を指定できます。 mfclient プログラムと mfserver プログラムでは、構成ファイルが次の方法で検索されます。

1. MFCSCFG 環境変数にファイル名が設定されているかどうかが検索されます。

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値より優先されます。

3. MFCSCFG が設定されていない場合または値が指定されていない場合に、コマンド行にファイル名が指定されていないと、デフォルトの mfclisrv.cfg が現在のディレクトリで検索されます。

4. mfclisrv.cfg が検出されない場合は、説明済みの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、位置と名前を合わせて 128 文字までの完全パス名で指定できます。

クライアントプログラムと mfclient の接続

クライアント / サーバ結合を使用したプログラムを作成するには、クライアントとサーバのプログラムが接続を正しく制御できるように、次の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注釈が付いています。

```
 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、サーバとの接続が終了するまで
* 繰り返される

     PERFORM UNTIL End-Connection

* 「lnk-client」の値は「mfclient」である。
* 初回はシステムを初期設定し、
* サーバ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 の制御ブロックと
* ユーザのデータブロックに指定してアクセスを
* 可能にする。

             SET ADDRESS OF ds-control-block TO lnkcblock-ptr
             SET ADDRESS OF customer-data-block
                                 TO lnkdblock-ptr

* サーバとの接続が確立したら、
* ローカルの初期設定を終了し、
* 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

*  ユーザが画面上で終了フラグを設定しているかどうかを
*  確認する

        IF customer-exit-flg-true
           SET client-ending TO TRUE
        END-IF
     END-PERFORM
     STOP RUN.
```

注：クライアントプログラムの連絡節に指定された Dialog System コピーブックは、結合によって指定されるアドレスにマッピングされます。 このコピーブックには、マッピングされていない 01 レベルの項目が他に 3 つあります。 この項目は、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。 これらの項目がマッピングされないのは、マッピングによって値が選択されないため、および、必要な値が格納されているためです。 Dialog System のバージョン番号は、ほとんど変更されないので、プログラムでハードコーディングできます。 ただし、データブロックのバージョン番号は変更されることがあるので、プログラムに固定値を指定すると、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。 この操作を簡便化するために、結合で使用するサポートプログラムを使用して、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を設定する 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には、拡張子を正しく指定する必要があります。 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
```

アプリケーションを実行するクライアントの数を制御する場合。または、エラーメッセージの表示を独自に処理する場合は、ユーザ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  <* ユーザ非同期オプション
           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
```

サーバプログラムと mfserver 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 のコピーブックを CS 結合パラメータ
*  ブロック内の位置と関連付ける
*-----------------------------------------------------------*

    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ユーザプログラムで実行する場合は、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.
```

クライアント / サーバ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、または、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 モジュールと mfserver モジュールは、標準の COBOL プログラムとして実行します。クライアントとサーバの各構成要素は手動で起動します。

既存のスタンドアロンプログラムをサーバとして実行する場合は、その名前を srvtier 構成エントリに、ベースサーバが実行するプログラムとして指定します (srvtier)。 mfserver のコピーが次のように作成され、名前が dsgrun に変更されます。このコピーによって通信が処理されます。

- UNIX 以外のサーバの場合は、mfserver.int ファイルを DSGRUN.int にコピーしなければなりません。
- UNIX の場合は、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 にリンクできます。UNIX では、cobconfig ファイルを作成し、次のエントリを指定します。

  set program_search_order=3

  さらに、COBCONFIG 環境変数でこのファイルを指定します。 その結果、スタンドアロンアプリケーションで DSGRUN が呼び出された場合に、名前が変更された 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ビルドについては、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として使用する最大の利点は、1 台の PC または PC ネットワークで実行するアプリケーションを利用できること、および、結合を使用して、単一のユーザインターフェイスプログラムを用意するのみで、クライアント / サーバ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 / サーバ結合を実行するには、サーバ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 でサーバプログラムを起動するコマンド行は、次のとおりです。

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]

詳細は、次のとおりです。

-b (UNIX のみ)	サーバをバックグラウンドプロセスとして実行します。コマンド行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。 サーバからのエラーメッセージは、mfclisrv.log ログファイルに書き込まれます。
-p protocol	通信プロトコルを指定します。
-s server-name	デフォルト (MFCLISRV) 以外のサーバ名を指定します。
-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行は、次のとおりです。

runw program-name [config-filename]

詳細は、次のとおりです。

program-name	ユーザインターフェイスプログラム名を指定します。
config-filename	使用する構成ファイル名を指定します。 クライアントでは、まず MFCSCFG 環境変数によって指定された名前、次にコマンド行で指定された名前で、構成ファイルが検索されます。 どちらの名前も指定されていない場合は、mfclisrv.cfg 構成ファイルが検索されます (この場合は、この名前のファイルを作成する必要があります)。 構成ファイルのエントリ、または、構成ファイルの検索方法については、『クライアント / サーバ結合の構成ファイル』を参照してください。

COBOL プログラムの実行については、COBOL システムのマニュアルを参照してください。

アプリケーションのアニメート

Net Express 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。

サーバプログラムをアニメートする場合は、構成ファイルパラメータの srvanim=y を設定します (『構成ファイルのエントリ』を参照)。

PC サーバでは、srvanim=y が設定されている場合にクライアントがサーバに接続すると、自動的に Animator が起動します。

UNIX サーバの場合は、mfserver を起動した端末で Animator が実行されるので、mfserver をフォアグラウンドモードで実行する必要があります。 ただし、mfserver は、バックグラウンドプロセスとして実行する方が望ましいので、問題が発生します。 また、一度に mfserver をアニメートできるのは、1 名のユーザのみです。 この問題を解決するには、srvanim=x,filename と設定します。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

標準の入出力に使用する端末では、構成ファイルに srvanim=x,filename 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。

クライアントの管理

mfcsmgr プログラムを実行して必要なパラメータと値を渡すと、サーバの動作を変更できます。 パラメータは、次の 2 つのグループに分けられます。

- 位置

  このグループのパラメータは、ターゲットサーバを指定するためのパラメータで、m、p、および s として設定し

ます。
- 動作

  これらのパラメータは、ターゲットサーバの動作を制御するためのパラメータで、 a、c、o、r、および t として設定します。パラメータ o、r、および t は、単独で指定する必要があります。これらのパラメータを組み合わせて指定すると、エラーメッセージが表示されます。

mfserver のシャットダウン

生成されたサーバは、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。

最初のサーバプログラムは、クライアントが終了しても動作し続けるため、手動で終了しなければなりません。

認証パスワードの管理

アクティブなサーバ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、サーバを終了する前か、または、サーバのパラメータを変更する前に入力する必要があります。

クライアントの最大数の設定

サーバがサポートするクライアントの数はデフォルトでは 65535 ですが、mfcsmgr の機能を使用してクライアントの最大数を減らすことができます。 指定した値はパスワードファイルに保存され、サーバが起動するたびに使用されます。

優先サーバの設定

サーバは、server-name、protocol、machine-name という優先オプションをサポートしています。 サーバでは構成ファイルが使用されないので、優先パラメータは mfcsmgr プログラムを使用して指定します。

このプログラムを実行すると、指定した機能を処理するためにクライアントとサーバの間で短い情報交換が発生します。

mfcsmgr のコマンド行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]
```

詳細は、次のとおりです。

-a	ターゲットサーバの認証パスワードを変更します。
-c nnnnn	サーバでサポートするクライアントの最大数を設定します。
-d	クライアントが存在する場合にローカルの優先ファイルを削除します。

-i filename	クライアントが存在する場合に、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) 以外のサーバ名を指定します。
-o m, machinename	優先サーバを実行しているマシンの名前を指定します。
-o p, protocol	優先サーバにアクセスします。
-o r	アクティブな優先パラメータをリセットして、 元の設定に戻します。
-o s, servername	ターゲットサーバへの接続をこのサーバへの接続に変更します。
-t	サーバを終了します。
-v	mfclient のバージョン番号を表示します。

上記のフラグには、「-」または「/」を付けて指定します。大文字と小文字は区別されません。

高度なトピック

ここでは、次の高度なトピックを説明します。

- 監査トレールの作成
- 構成ファイルのエントリの無効化
- インライン構成機能の使用方法
- 縮小データ転送機能
- サーバ制御のファイル管理機能

監査トレールの作成

クライアント / サーバ結合では、クライアントがサーバ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」を設定します。 監査トレールの情報は、『システムエラー / メッセージログ』で説明するシステムログファイルに記録されます。

構成ファイルのエントリの無効化

各クライアントの起動時に、クライアント / サーバでは次の処理が実行されます。

1. 構成ファイルが読み込まれます。
2. システムログファイルと同じ位置で mfcsovrd.cfg ファイル内のクライアント / サーバ結合が検索されます。 このファイルが検出されると、構成ファイルを使用して設定されていたすべてのパラメータよりも、このファイルの設定が優先されます。

   優先ファイルの形式は通常の構成ファイルと同じですが、override-cntrl というエントリが追加されています。 このエントリによって、優先させるエントリの内容を指定します。指定できる対象は、次のとおりです。

   - サーバ名

     サーバ名を指定した場合は、そのサーバを使用するすべてのクライアントのパラメータが優先ファイルの内容に従って変更されます。

- タグ名

  タグ名を指定した場合は、そのタグ名を使用するクライアントのパラメータのみが変更されます。

クライアントの優先機能は、サーバが使用不可能になったために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こともできます。ただし、1 つの優先ファイルですべてのアプリケーションの接続先を変更できます。

優先機能はサーバでも使用できますが、その場合はサーバマシンが起動および稼動している必要があります。 この場合も、サーバ名またはタグ名を優先させることができます。

優先ファイルが検出されると、エントリがログファイルに記録されます。また、優先する対象となるパラメータもすべて記録されます。

優先機能を使用してクライアントの接続先サーバを変更する例を次に示します。

```
[OVERRIDE-CNTRL]
OVERRIDE=SERVERNAME
[OLDSERVER]
SERVERNAME=NEWSERVER
```

この例では、[override-cntrl] セクションで優先する対象となるサーバ名を指定 (override=servername) し、そのサーバ名 ([oldserver]) の下に新たな接続先サーバ名を指定 (servername=newserver) しています。 この場合は、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を優先設定する例を次に示します。

```
[OVERRIDE-CNTRL]
OVERRIDE=TAGNAME
[MF-CLISRV]
SERVERNAME=NEWSERV
```

この例では、[override=cntrl] セクションで優先する対象となるタグ名を指定 (override=tagname) し、そのタグ名 ([mf-clisrv]) の下に優先するパラメータ (この場合はサーバ名) を指定 (servername=newserv) しています。 この場合は、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:
```

インライン構成機能の使用方法

クライアント / サーバ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、エンドユーザが構成ファイルのエントリを変更するのみでアプリケーションの動作が変更されるので、 問題が発生することがあります。 そのため、すべての構成パラメータをクライアントプログラム内で指定する方法があります。 この方法では、構成ファイルが一切使用されないため、エンドユーザはアプリケーションの動作を変更できません。 ユーザが複雑な通信コードを作成する必要はなく、パラメータをクライアントプログラム内で簡単に指定するのみなので、便利です。

パラメータを指定する場合は、クライアントプログラムの mfclisrv.cpy コピーファイルの中で、まず load-inlinecfg を設定し、最後に end-inline-cfg を設定します。 各パラメータエントリは、lnkerror-msg フィールドにロードされ、mfclient が呼び出されて処理されます。 パラメータは、実際の処理ループを開始する前に指定する必要があります。 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、サーバとの接続が終了するまで
*  繰り返される

     PERFORM UNTIL End-Connection

*  「lnk-client」の値は「mfclient」である。
*  初回はシステムを初期設定し、
*  サーバ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構成機能の両方を使用できます。 この方法では、エンドユーザが必要に応じてシステムを制御でき、最終的な制御がクライアントプログラムによって実行されるので、便利です。 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。 そのため、構成ファイルに不適切なパラメータが指定されていても、インライン設定が優先されます。 この方法を使用するときは、まず use-combined-cfg を設定し、最後に end-inline-cfg を設定します。

```
 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 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、サーバとの接続が終了するまで
* 繰り返される

    PERFORM UNTIL End-Connection

* 「lnk-client」の値は「mfclient」である。
* 初回はシステムを初期設定し、
* サーバ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  .....
```

縮小データ転送機能

縮小データ転送 (RDT) 機能を使用すると、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、処理に必要な最小限のデータに制御したりできます。

クライアント / サーバ結合では、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、ネットワークに許容範囲を超える負荷がかかる場合があります。 たとえば、22 KB のレコード領域をもつ 24 KB のバッファが割り当てられた場合を想定します。 これらのレコードを格納するファイルで 10 バイトのレコードキーを使用する場合に、クライアントとサーバの間でキーを送受信すると、必要なバイト数は 10 バイトのみであるにもかかわらず、24 KB 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、その点を考慮しても、バッファ全体に比べて必要なデータが小さすぎます。

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 に設定する必要があります。

lnk-usr-fcode	ユーザファンクション番号。 ユーザのサーバプログラムに対して、受信したデータの処理方法を通知します。

Lnk-usr-retcode　バッファの開始位置。 4 つのデータ領域のどれが転送開始位置であるかを指定します。

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転送時に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ます。指定しない場合は、デフォルトのオプション (圧縮なし) が適用されます。 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 文を多用してコードの流れを変更する必要がなく、コードがシンプルになります。 NULL 操作は、ローカルで特定の機能を処理する場合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　転送するデータのサイズ。

索引ファイルに対してレコード (顧客情報) の追加、削除、および読み込みを行うアプリケーションを想定します。 顧客情報のレコードキーは、顧客コードです。 このアプリケーションには、すべての顧客情報をインターフェイス画面から消去するオプション (CLEAR) も指定されていると仮定します。 このような条件を満たすアプリケーションは、この製品でインストールされています (顧客サンプルプログラム)。 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格納されます。このアプリケーションでは、クライアントとサーバの間で顧客詳細情報レコードをやり取りするためのデータブロック領域 (dblksize) を使用しますが、これを customer-data-block と呼びます。 customer-data-block 内の 6 バイトのデータ項目 customer-c-code にレコード キーが格納されます。

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
* 顧客詳細を読み取って表示するためにインターフェイスから
* 「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

* ユーザが、顧客コードを入力し、ファイルから顧客レコードを削除するために、
* 「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

* ユーザが、画面から現在の顧客情報を消去するために
* 「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」オプションでは、NULL 操作を使用しています。これは、レコードの消去操作がローカルで実行されるので、サーバに接続する必要がないためです。 RDT 機能を実行するためのサーバプログラムの一部を次に示します。クライアント / サーバ結合では、「send-via-rdt」フラグが設定されるので、lnk-usr-fcode の値をチェックできます。 サーバ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」機能を実行すると、サーバプログラムによって
*  データファイルから顧客情報が読み取られ、
*  データが customer-data-block データ領域に格納されて
*  クライアントに返る。RDT 機能は使用されない。
*  RDT フラグが設定されている場合を除き、
*  クライアント / サーバ結合は、クライアントとサーバの間で
*  常にデータ領域全体 (構成ファイルの dblksize で定義)
*  を送受信する。

           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
```

サーバ制御のファイル管理機能

クライアント / サーバソリューションの問題の 1 つに、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。

優先機能 (『構成ファイルのエントリの無効化』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、クライアントアプリケーションの接続先を維持中のサーバから別のサーバに変更できます。 優先機能は、ローカルでもリモートでも実行できますが、各クライアントでローカルに優先ファイルをインストールしたり削除したりする場合は、時間がかかります。

mfcsmgr プログラムを使用すると、優先ファイルのインストール (-i オプションを指定) や削除 (-d オプションを指定) を実行できます。 『サーバの管理』を参照してください。 インストールまたは削除は、クライアントプログラムの終了時に実行されます。

mfcsmgr プログラムで -i オプションを使用すると、優先ファイルのみでなく、どのタイプのファイルでも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。 つまり、この方法で新しいスクリーンセットやプログラムファイルをインストールすると、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いません。削除できるのは優先ファイル (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、サーバ上に保存する必要があります。 インストールするファイルが mfserver を起動したディレクトリ内にある場合は、ファイル名のみを指定します。 別のディレクトリにある場合は、完全パス名で指定し、そのパスからファイル名を抽出できるようにする必要があります。 たとえば、/u/live/update/newprog.int、d:\testprog.int、$LIVE/newtest.int はどれも有効ですが、$newfile は無効です。

これらの機能を使用するためにプログラムを変更する必要はありません。 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。

付属の顧客例の実行

Dialog System のクライアント / サーバ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 ディレクトリの csbind.txt ファイルを参照してください。

システムエラー / メッセージログ

結合のクライアントモジュールとサーバモジュールで、エラーとメッセージのログが記録されます。 すべてのエントリには、日付と時刻がスタンプされ、mfclisrv.log 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、または、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ことを確認してください。

クライアント / サーバの制約

クライアント / サーバ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 NT、および 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 / サーバ結合モジュールを実行できます。 また、UNIX プラットフォームの場合は、クライアント / サーバ結合モジュールをユーザが作成したアプリケーションプログラムとリンクさせて、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 / サーバ結合によって制限されることはありません。ただし、サーバの能力、ネットワークプロトコル、またはエンドユーザの性能条件によって制限されることがあります。 たとえば、UNIX では、mfserver が生成するサブプロセスの数によって制限が決まります。 ユーザが UNIX マシンにログオンするときに割り当てられる一意のプロセス ID では、限られた数のサブプロセスしかサポートされません。 クライアント / サーバ結合を通じて接続したクライアントは、すべてベースサーバのサブプロセスになります。 ただし、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を実行することによって、この制限を回避できます。 UNIX マシンが直接ログインできるユーザを数百件サポートするように構成されている場合は、サブプロセスの制限に関する問題が解決したときに、1 つのマシンで実行されるクライアント / サーバ結合で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 / サーバの両端で障害の発生を検知されログに記録されますが、データは回復できません。 接続の両端のユーザ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。 この場合は、クライアント / サーバ結合では、標準 RTS 以上の機能は提供されません。

第 15 章 : 高度なトピック

ここでは、Dialog System で利用できる、より高度なシステム機能について扱います。

ここでは、次の内容について説明します。

- 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とのインターフェイスを作成する方法
- ファイル選択機能とのインターフェイスを作成する方法
- 実行時にメニュー項目を修正する方法
- 呼び出しインターフェイスの高度な使用方法
- ヘルプの追加

複数の解像度で実行するアプリケーションの実装

運用で複数のデスクトップの解像度を使用する場合は、それに対応したアプリケーションを記述するために、現在の解像度に基づいてウィンドウ、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 を使用して DSFNTENV 環境変数を正しく設定する

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、フラグ「9」と識別子を指定します。

Dialog System ランタイムシステムでは、Panels V2 の汎用座標を使用します。そのため、

クロス解像度がサポートされ、異なるグラフィックアダプタとの互換性が確保されます。この座標以外では解像度設定が同じに見える複数の値を、異なる座標値として区別できます。

定義プラットフォームに提供する識別子を検証するために、Dialog System には検証プログラムが提供されています。このプログラムによって、変更の適用に必要な解像度識別子に関するメッセージ ボックスが表示されます。 このプログラムは、次のように実行します。

runw dsreschk

その結果、次の内容が表示されます。

- 現在の解像度の Panels V2 汎用座標
- Dialog System ランタイム呼び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、すべてのウィンドウ、ダイアログボックス、およびその子コントロールは、現在の解像度に比例してサイズ変更されます。 ヘルプトピックの『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』を参照してください。

フォントのマッピングを可能にする

各スクリーンセットで複数の解像度をサポートするのみでなく、作成するオブジェクトにフォントスタイルを割り当て、実行時のフォントが運用アプリケーションの解像度に合わせて調整されるようにする必要があります。

[編集] メニューの [フォント] を使用してフォントを指定すると、そのフォントの書体とスタイル名を指定できます。 スタイル名とは、指定した書体、ポイントサイズ、および属性を表すユーザ定義の名前です。 このスタイル名によって、フォントを実行時の解像度に適したサイズに変更できます。

フォントの書体、ポイントサイズ、および属性を選択して、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 (またはオブジェクトグループ) 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] を使用して、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、実行時に Dialog System によってバイナリフォントサイドファイル内のスタイル名が検索され、そのフォントの情報が、使用中の解像度のプラットフォームに使用されます。 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用されます。

解像度 1024*768 で作成されたスタイルを使用し、解像度 640*480 に変換する場合を想定します。 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 (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運用環境に合わせて変換するには、.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。 そのため、このファイルを次のように記述します。

```
[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 が解像度の定義に合わせて変更されています。
- 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後で、次のように、サイドファイルをバイナリ形式に変換します。

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 では、そのフォントスタイル定義に対して、新しいフォント属性が自動的に選択されます。 640*480 未満の解像度では、DSFNTENV が RESOLUTION2 に設定されます。この場合は、使用可能な表示空間が減るため、より小さいフォントが必要になります。

DSFNTENV を設定しない場合は、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用されます。 複数のスクリーンセットで構成されているアプリケーションでも、これらのスクリーンセットでのスタイル名の使用方法が一貫している場合は、フォントサイドファイルを 1 つ使用するのみでかまいません。

COBOL を使用して 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 環境変数を設定できます。ここでは、設定方法を説明します。まず、現在のアプリケーションが実行している解像度を判断する必要があります。

次のコードを使用して解像度を調べます。

```
* 解像度の検出 & DSFNTENV の設定
    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.
```

プログラムの作業場所節に pan2link.cpy および pan2err.cpy コピーファイルを記述し、必要な変数と Panels V2 インターフェイスレコードを取得します。

次に、P2I-Screen-Width と P2I-Screen-Height に返される値をテストします。さらに、次のコードと COBOL の予約語を使用して、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 の最初の呼び出しの前に設定する必要があります。また、この環境変数は、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、運用プラットフォームの解像度を改良する (800*600 - 大きいフォントなど) ことができます。

```
ADD P2I-Generic-Coordinates TO P2I-Environment
```

このコードによって、返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。 次に、後続の IF 文を調整して、サポートされた運用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ます。また、現在の運用環境の解像度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す。

ウィンドウのレイアウトが適切に設計されている場合は、スクリーンセットをさまざまな解像度に完全に対応させることができます。

Dialog System のエラーメッセージファイルハンドラの使用方法

Dialog System のエラーメッセージファイルハンドラと表示機能を使用して、呼び出し側プログラムでエラーメッセージを表示できます。 この機能は、Dialog System では処理できない複雑な妥当性検査処理が必要な場合に便利です。

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、スクリーンセットで使用中のエラーメッセージファイルを使用する方法です。 または、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。ただし、その場合は、エラーメッセージファイルを「開く」操作と「閉じる」操作を明示的に行う必要があります。

Dialog System のエラーメッセージファイルハンドラを実行時に使用するには、そのプログラムで次の処理を行う必要があります。

- エラーファイルが開かれていることを確認する。
- エラーメッセージを抽出する。
- エラーメッセージとエラーメッセージを表示するための命令を Dialog System に渡す。

これらの操作を実行するには、次のコードのように記述します。

```
 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" using error-file-linkage
32                          error-file-data
33    ...
```

3 ～ 11 行目：

```
 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).
```

作業場所節でこれらのレコードを定義します。

15 ～ 22 行目：

```
    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
```

Dialog System に対する最初の呼び出しです。

24 ～ 26 行目：

```
    move "E" to ds-control
    call "dsgrun" using ds-control-block
```

```
                data-block
```

スクリーンセットで使用中のエラーメッセージファイルを使用する場合は、Dialog System を通常の方法で呼び出し、ファイルが開かれていることを確認する必要があります。この呼び出しでは、ds-control にパラメータ「E」を指定する必要があります。 Dialog System によって、ファイルを開いたプログラムにただちに制御が戻り、ds-control のパラメータが「C」(ds-continue) に置き換えられます。

ds-err-file-open は、ds-control に対してあらかじめ定義された値です。 この値は、制御ブロックで次のように定義します。

```
    05 ds-err-file-open          pic x value "E".
```

つまり、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
```

28 行目：

```
    move "custerr" to short-file-name
```

エラーファイルの名前 (拡張子 .err を除く) をデータ項目 short-file-name に転記します。

29 行目：

```
    move "R" to file-access
```

「R」は、読み取りを表します。

30 行目：

```
    move 101 to error-record-number
```

表示するエラーメッセージの番号を転記します。

31 ～ 32 行目：

```
    call"dserrhan" using error-file-linkage
                         error-file-data
```

この呼び出しでは、エラーメッセージファイルを読み取ります。 呼び出しが成功すると、「OK」が errhan-code に、エラーメッセージが error-record-contents に格納されて返ります。 ファイルが見つからない場合は、値「NF」(Not Found) が errhan-code に書き込まれます。 ファイルが使用中の場合、または 17 ファイル以上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。

続いてエラーメッセージをデータブロックのデータ項目に格納し、Dialog System が表示できるようにする必要があります。 この操作を実行する 1 つの方法として、データ項目を引数として使用し、メッセージボックスを表示する手続きを設定した後で、Dialog System を呼び出してその手続きを要求する方法があります。

たとえば、ERR-DISPLAY-MB というメッセージボックスがある場合は、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
```

さらに、プログラムで Dialog System に戻るときに、次のような文を使用して手続きを要求します。

```
    move "display-err-msg" to ds-procedure
```

display-err-msg は、手続きの名前です。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 のエントリ時に実行されます。

代替エラーメッセージファイルの使用方法

スクリーンセットで指定したエラーメッセージファイル以外を使用するには、そのプログラムで実行時に次の処理を行う必要があります。

- エラーファイルを開く。
- エラーメッセージを抽出する。
- エラーメッセージとエラーメッセージを表示するための命令を Dialog System ランタイムシステムに渡す。
- スクリーンセットで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する。

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相違点は、代替ファイルの場合には、「開く」操作と「閉じる」操作を明示的に行う必要があることです。

これらの操作を実行するには、次のコードのように記述します。

```
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     ...
```

3 ～ 11 行目：

```
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).
```

ここでも、作業場所節でレコードを定義する必要があります。

15 行目：

```
    move "ALTERR" to short-file-name
```

代替ファイルを識別します。

16 行目：

```
    move "NI" to file-access
```

「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。

17 行目：

```
    move "C:\CUST\ALTERR.ERR" to error-file-data
```

開くエラーファイルのパスと名前を指定します。

18 ～ 19 行目：

```
    call "dserrhan" using error-file-linkage
                          error-file-data
```

呼び出し文によって、エラーファイルを読み取ります。 呼び出しが成功すると、「OK」が errhan-code に、エラーメッセージが error-record-contents に格納されて返ります。 呼び出しが失敗すると、errhan-code は「NF」(Not Found) になります。 ファイルがすでに開かれている場合、または 17 以上のファイルを開こうとした場合は、errhan-code が「FU」(File in Use) になります。

27 行目：

```
    move "ALTERR" to short-file-name
```

閉じるファイルを指定します。

28 行目：

```
    move "NC to file-access
```

「NC」は、ファイルを閉じるためのコードです。

29 ～ 30 行目：

```
    call "dserrhan" using error-file-linkage
                          error-file-data
```

この呼び出しでは、ファイルが閉じます。errhan によって、ファイルが閉じ、プログラムに戻ります。

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用方法』を参照してください。

ファイル選択機能とのインターフェイスを作成する方法

ユーザがファイルの名前を入力する必要がある場合は、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。 ユーザは、システムのドライブやディレクトリを参照し、既存のファイル名を選択して、新しい名前を入力できます。

Dialog System では、スクリーンセット用の コピーブックを生成し、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。

ここでは、Dialog System の拡張機能 Dsdir DSX を使用して、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。 Dialog System Extensions (DSX) は、CALLOUT ダイアログ機能を使用して実行されるダイアログ機能です。 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したりするなど、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の機能です。 DSX の詳細については、ヘルプトピックの『Dialog System 拡張機能』を参照してください。

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 で使用できる機能を表示できます。

Dirdemo は、Dialog System 定義ソフトウェアから直接実行できます。 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 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、『スクリーンセットの使用方法』の章を参照してください。 または、Dsrunner を使用して Dirdemo サンプルを実行することもできます。 Dsrunner の詳細については、『複数のスクリーンセット』の章を参照してください。

このプログラムでは、次の 2 つのプロンプトが表示されます。

- 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は、最初はそのファイル名に一致するファイルのみが表示されます。 ファイル名にワイルドカード文字「＊」と「?」を使用できます。 たとえば、Dialog System のスクリーンセットをすべて表示するには、*.gs と入力します。 このフィールドを空白のままにした場合は、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。 詳細については、ヘルプトピックの『Dialog System 拡張機能』にある Dsdir の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ど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。 選択できるファイルは、既存のファイルのみです。
上書き保存	保存するファイルを選択します。 既存の任意のファイルを選択できます (または、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は、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。 ファイルの選択後に、使用する機能を選択します。

Dirdemo のデータブロック

Dsdir DSX を呼び出すには、スクリーンセットのデータブロック内で、スクリーンセットに使用する他のデータに加え、次の項目を定義する必要があります。

```
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、スクリーンセットのデータブロックに次のコード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、ヘルプトピックの『Dialog System 拡張機能』にある Dsdir DSX の説明を参照してください。

Dirdemo のダイアログ

Dirdemo サンプルでは、ファイルを選択する必要があるときに Dsdir DSX を呼び出しま

す。

この操作を実行するためのダイアログ (メインウィンドウに設定) は、次のとおりです。

```
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
```

1 行目：

```
DO-DSDIR-CALL
```

この手続きでは、Dsdir DSX が呼び出されます。

2 ～ 4 行目：

```
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 を指定する前にすべてのパラメータを定義する必要があります。 CLEAR-CALLOUT-PARAMETERS 関数は、新しいパラメータを定義する前に、以前定義されたパラメータが削除されていることを確認するために使用します。

5 行目：

```
CALLOUT "Dsdir" 0 $PARMLIST
```

この行では、以前に定義されたパラメータを使用して DSX を呼び出します。 ファイル選択ウィンドウが表示され、ユーザはファイルを選択できます。 ユーザがファイルを選択した場合、または、ファイル選択をキャンセルした場合は、スクリーンセットに制御が戻りま

す。

17 行目：

```
  @OPEN-PD
```

ユーザがプルダウンメニューから [開く] オプションを選択したことを表します。

18 行目：

```
    MOVE "open" DSDIR-FUNCTION(1)
```

この行では、ユーザが開くファイルを選択できるように、DSX に対してパラメータを設定します。 このモードでは、ユーザは既存のファイルのみを選択できます。

19 行目：

```
    EXECUTE-PROCEDURE DO-DSDIR-CALL
```

DSX を呼び出す手続きを実行します。

20 ～ 22 行目：

```
  @SAVE-PD
    MOVE "save" DSDIR-FUNCTION(1)
    EXECUTE-PROCEDURE DO-DSDIR-CALL
```

ユーザがプルダウンメニューから [上書き保存] オプションを選択したことを表します。 このモードでは、ユーザは任意のファイルを選択するか、または、新しいファイルを指定できます。

23 ～ 25 行目：

```
  @CHECK-PD
    MOVE "chek" DSDIR-FUNCTION(1)
    EXECUTE-PROCEDURE DO-DSDIR-CALL
```

ユーザがプルダウンメニューから [確認] オプションを選択したことを表します。 chek 関数は、指定さ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。

実行時のメニュー項目変更

定義時にメニュー項目を定義するのみでなく、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したりできます。

既存のメニューバーには新しい項目しか追加できませんが、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用できます。また、メニュー項目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 ダイアログ関数を使用して、メニューバーにメニュー項目を追加します。 この関数のパラメータの 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列があります。次のサンプルのオプションは、すべて正しい位置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
```

詳細は、次のとおりです。

PARENT-REF	このパラメータでは、新しい項目を追加するメニュー (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 関数を使用して取得します。 値が 0 になることはありません。
*	アスタリスクを指定すると、作成されたメニュー項目にチェックマークが表示されない状態で設定されます (デフォルトではチェックマークなし)。 アスタリスクは、テキストパラメータの先頭に指定します。先頭に指定しない場合は、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、前に空白文字を 1 つ追加します。
~	チルダを指定すると、その次に指定した文字がメニュー項目のニーモニック文字として定義されます。 メニュー項目を定義する場合にニーモニック文字が重複していても、警告が表示されません (定義時のメニュー項目定義と異なります)。 チルダ文字は、メニュー項目テキストのどこに指定してもかまいません。
>	記号「より大きい」をテキストパラメータの最後の文字として指定すると、メニュー項目の選択時にそのサブメニュー項目が表示されます。 メニュー項目のサブメニューを表示するには、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) の直前までのテキストは、メニューバー (選択テキスト) に表示されます。

@item	「@」を指定すると、次のテキスト (次の @、空白文字、&、または > の直前まで) がメニュー項目名として処理されます。 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。 メニュー項目と同じ名前の手続きがスクリーンセットに存在する場合にメニュー項目を選択すると、その名前の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	アンパサンドを指定すると、次のテキスト (次の @、空白文字、&、または > の直前まで) がメニュー項目のショートカットキーになります。 ショートカットキーを指定する場合は、メニューバー定義でメニュー項目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 (Ctrl+C など) を使用できます。

この関数を使用し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 などの関数に使用すると、メニュー項目をさらに操作できます。

DELETE-MENU-CHOICE 関数を使用すると、メニュー項目を削除できます。 削除できる項目は、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 関数を使用して追加したメニュー項目です。

また、UPDATE-MENU-TEXT 関数を使用して、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、ヘルプで、関連する関数の説明 (関数の使用例など) を参照してください。

呼び出しインターフェイスの使用方法

呼び出し側プログラムで Dialog System の呼び出しインターフェイスを使用すると、Dialog System をより高度な方法で操作できます。 高度な操作の例：

- 複数のスクリーンセットを使用する。
- 同じスクリーンセットについて複数のインスタンスを使用する。

これらの機能を使用すると、必要に応じて論理的な構成要素にユーザインターフェイスを分割したり、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たりできます。 詳細については、『スクリーンセットの使用方法』の章を参照してください。

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説明します。Helpdemo スクリーンセット：

- DialogSystem\demo\helpdemo に格納されています。
- Dsonline Dialog System 拡張機能を使用します。この機能は、Windows のヘルプを表示するための Dialog System の拡張機能です。
- Windows のヘルプを使用してコンテキストヘルプを表示します。
- ヘルプシステムのすべての機能にアクセスできます。
- サンプルのスクリーンセットです。

この説明の参照中に、このスクリーンセットを実行することもできます。

Helpdemo サンプルの実行

Dialog System 定義ソフトウェアから直接 Helpdemo サンプルプログラムを実行できます。このサンプルには COBOL プログラムはありません。

helpdemo.gs スクリーンセットをロードし、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用方法』の章にある『スクリーンセットのテスト』を参照してください。

また、Dsrunner を使用して Helpdemo サンプルを実行することもできます。 Dsrunner の詳細については、『複数のスクリーンセット』の章を参照してください。

Helpdemo のデータブロック

Dsonline DSX を呼び出すには、次の行とスクリーンセットで使用するその他のデータ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については、ヘルプトピックの『Dialog System の拡張機能』にある Dsonline DSX の説明を参照してください。

Helpdemo のダイアログ

Helpdemo サンプルでは、ヘルプ情報が必要なときに Dsonline DSX を呼び出します。 特定のフィールドに対してヘルプを呼び出します。また、ユーザが [ヘルプ] メニューのオプションを選択した場合にも呼び出します。

この操作を実行するためのダイアログ (メインウィンドウに設定) は、次のとおりです。

```
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

ユーザが F1 キーを押したことを表します。 このサンプルの各エントリフィールドには、ユーザが F1 キーを押した場合に発生するイベントに対する制御ダイアログが設定されています。そのため、このダイアログでイベントが処理された場合は、フォーカスがどのエントリ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、一般的なヘルプが表示されます。

2 行目：

```
   MOVE 1 DSONLINE-HELP-CONTEXT(1)
```

表示す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

ビルダ (ohbld) によって番号が設定されます。 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、手続きが使用されます。

4 行目：

```
  @HELP-CONTENTS
```

ユーザが [ヘルプ] メニューの [目次] を選択したことを表します。

5 行目：

```
    MOVE "cont" DSONLINE-FUNCTION(1)
```

Dsonline DSX には数種類の動作があります。 動作を指定するには、関数名を指定する必要があります (4 文字)。 cont 関数は、オンラインヘルプファイルの目次を表示するための関数です。

6 行目：

```
    BRANCH-TO-PROCEDURE CALL-ON-LINE
```

DSX 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
```

ユーザが [ヘルプ] メニューの [索引] を選択したことを表します。 indx 関数は、オンラインヘルプファイルの索引を表示するための関数です。

10 行目：

```
  @EXIT-F3
```

ユーザが [ファイル] メニューの [終了] を選択したこと、または、F3 キーを押したことを表

します。

11 行目：

```
   SET-EXIT-FLAG
```

終了フラグは、RETC の実行時に、Dsgrun から呼び出し側プログラムに返されるフラグです。 終了フラグによって、スクリーンセットでの処理が終了したことが Dsrunner に通知されます。

12 行目：

```
   RETC
```

呼び出し側プログラム (このサンプルでは、Dialog System または Dsrunner) に戻ります。

13 ～ 15 行目：

```
 CONTEXT-HELP
   MOVE "ctxt" DSONLINE-FUNCTION(1)
   BRANCH-TO-PROCEDURE CALL-ON-LINE
```

この手続きでは、コンテキストヘルプが表示されます。 ctxt 関数は、コンテキストヘルプを表示するための関数です。 CALL-ON-LINE 手続きでは、DSX を呼び出します。

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
```

この手続きでは、Dsonline DSX を呼び出します。 この手続きでは、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 (フラグの詳細については、ヘルプトピックの『Dialog System 拡張機能』にある Dsonline DSX の説明を参照してください)。

エントリ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、ウィンドウの各エントリフィールドにもダイアログが設定されます。 各エントリ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。

「Product Code」エントリフィールドのダイアログは、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が使用されています。 2 は、このエントリフィールドのヘルプに関するコンテキスト番号です。

詳細情報

スクリーンセットの使用を制御する方法、および複数のスクリーンセットとスクリーンセットの複数のインスタンスを利用する方法については、『スクリーンセットの使用方法』と『複数のスクリーンセット』の章を参照してください。

『複数のスクリーンセット』の章にも呼び出しインターフェイスの詳細情報が記載されています。 ヘルプトピックの『呼び出しインターフェイス』でも、制御ブロックの情報 (イベントブロック、データブロック、スクリーンセット Animator、バージョン確認、呼び出し側プログラムから Dialog System への戻り値などの情報) を説明しています。

高度なトピック　サンプルプログラム

第 16 章 : Q & A

ここでは、サポート窓口に頻繁にお寄せいただいた質問とその回答を説明します。

注：質問は、すべての環境に関連する場合と特定の環境に固有の場合があります。 質問の対象となる環境は、左側の余白に明記されています。

スクリーンセットから直接生成したデータブロックがチェッカーによってエラーになります。 どこに問題があるのですか？

データブロック内のすべてのデータ項目の先頭にスクリーンセット ID を設定するプログラムを作成した場合は、それに合わせてスクリーンセットを生成する必要があります。

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、データブロックに接頭辞としてスクリーンセット ID を設定するように指定しない場合は、データ項目に Customer という接頭辞は設定されず、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しないでスクリーンセット Animator 機能を使用するには、どうしたらよいですか？

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 デモプログラムを変更して、次のレベル-01 データ項目を使用します。

  ```
  01 dialog-system    pic x(8) value "dsgrun".
  ```

  これによって、アプリケーションのデバッグ中に dialog-system の値を「ds」に変更できます。「ds」は、スクリーンセット Animator 機能を使用するための Dialog System ランタイムシステムです。

  一度この操作を実行すると、その後の Dialog System ランタイムの呼び出しでは、必ずスクリーンセット Animator が使用されます (dialog-system の値を dsgrun に戻した場合を含む)。

- 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、次の操作を実行します。

1. Dialog System の CALL 文にブレークポイントを設定します。
2. データ項目 ds-control の値を「T」に変更します。
3. Dsgrun に対する呼び出しをステップ実行します。

   「T」オプションによって、Dialog System でスクリーンセット Animator 機能をオンにする内部呼び出しが実行され、終了します。
4. カーソルを Dialog System の CALL 文に再設定します。
5. プログラムを再度ズームします。 その後でダイアログの行が実行されると、スクリーンセット Animator が表示されます。

スクリーンセット Animator をオフにするには、次の操作を実行します。

1. データ項目 ds-control の値を「O」に変更します。
2. Dsgrun に対する呼び出しをステップ実行します。

   スクリーンセット Animator が表示されなくなります。「T」オプションと同様に、「O」オプションによって、Dialog System でスクリーンセット Animator をオフにする内部呼び出しが実行され、終了します。
3. カーソルを Dialog System の CALL 文に再設定します。
4. プログラムをズームします。 スクリーンセット Animator がオフになります。

SHOW-WINDOW 関数を使用するとダイアログボックスが表示されないのは、なぜですか？

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 関数では、画面上にこのタイプのダイアログボックスが描画されません。 モーダルダイアログボックスは、SET-FOCUS 関数を使用してフォーカスを設定した場合にのみ表示されます。 SHOW-WINDOW で画面上に描画されるのは、ウィンドウやモードレスダイアログボックスです。

フィールドからフォーカスが外れたときに、すぐに妥当性検査を実行するには、どうしたらよいですか ？

フィールドごとの妥当性検査を実装する際の最も一般的な問題は、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が発生することです。 たとえば、エントリフィールドか

らフォーカスが外れて妥当性検査が実行された場合に、その妥当性検査に失敗すると、フォーカスは元のオブジェクトに戻ります。 その結果、別のコントロールからフォーカスが外れるので、そのコントロールが妥当性検査されると、無限ループが発生する可能性があります。

また、フィールドの妥当性検査を必要としない場合も問題になります。 たとえば、ユーザ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。

このような場合は、フィールドごとに妥当性検査を実装すると、無限ループの問題を回避し、[キャンセル] および [ヘルプ] ボタンを使用できます。具体的には、次のように操作します。

1. 次のようにデータブロックにフラグを定義します。

   ```
   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を定義してある場合は、VALIDATION-ERROR-MESSAGE を定義するかわりに、そのフィールドを使用できます。 エラーメッセージフィールドを定義していない場合は、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します。

   ```
   REPORT-VALIDATION-ERROR
   * 妥当性検査エラーをレポートする
       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
   * 妥当性検査エラーフラグと TIMEOUT をリセットする
       MOVE 0 VALIDATION-ERROR-FOUND
       TIMEOUT 0 $NULL
     DO-NOTHING
   * Do-nothing 手続きには、関数は含まれません。
   ```

   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設定され、CLEAR-VALIDATION-ERROR-TIMEOUT が実行されます。 このイベントでは、妥当性検査エラーに関するフラグがクリアされます。
3. SCREENSET-INITIALIZED ダイアログに次の行を追加します。

MOVE 0 VALIDATION-ERROR-FOUND

この行によって、最初の妥当性検査エラー用に用意された VALIDATION-ERROR-FOUND フラグがクリアされます。

4. LOST-FOCUS ダイアログが設定され、フォーカスの変更が発生する可能性があるオブジェクトに対して、LOST-FOCUS ダイアログの先頭の行に次の行を追加します。

   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が発生するすべてのダイアログについて、フォーカスの変更を発生させるそのダイアログの直前に、次のダイアログを追加します。

   MOVE $EVENT-DATA VALIDATION-ERROR-FIELD
   MOVE 1 VALIDATION-ERROR-FOUND
   TIMEOUT 1 REPORT-VALIDATION-ERROR

   このダイアログ例では、妥当性検査エラーが発生し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 が発生したときにオブジェクト 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は、エラーをキャンセルする場所に、次のダイアログを追加します。

   EXECUTE-PROCEDURE CLEAR-VALIDATION-ERROR-TIMEOUT

   プッシュボタンの妥当性検査エラーをキャンセルする場合は、ボタンの BUTTON-SELECTED イベントではなく、GAINED-FOCUS イベントの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 イベントの間で発生し、妥当性検査エラーメッセージが表示されます。

   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、ユーザが他のアプリケーションに切り替えられなくなり、スクリーンセット Animator や Net Express 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 ダイアログを設定します。 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、この方法を適用できます。

妥当性検査エラーによってフォーカスが変更されると、VALIDATION-ERROR-FOUND フラグと、このフラグを再度クリアするための短い TIMEOUT が設定されます。 TIMEOUT イベントは、処理するイベントが存在する限り発生しないので、必ず LOST-FOCUS イベントの後に発生します。 LOST-FOCUS イベントで VALIDATION-ERROR-FOUND フラグが設定されていることが検知されると、フォーカスを失ったフィールドは妥当性検査されません (フォーカスの移動も発生しません)。

Dialog System ランタイムエラーの詳細を知るには、どのようにしたらよいですか？

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からエラーコード (DS-ERROR-CODE) および 2 つのエラー詳細情報 (DS-ERROR-DETAILS-1 と DS-ERROR-DETAILS-2) が返ります。これによって、アプリケーションの問題点を判断できます。

ただし、エラーコード 17 については、すべてのエラー情報を得るために、複数のテーブルを調べる必要があります。 また、エラーの詳細を使用してエラー情報を返すエラーコードの場合は、マニュアルを参照して解読する必要があります。

Dialog System には、エラーをレポートするモジュール dsexcept.dll が組み込まれています。このモジュールは、エラーコードを解読し、エラーコードのみではわからない追加情報も表示します。

dsexcept.dll をアクティブにするには、$COBDIR 環境変数で指定したディレクトリにこのファイルを配置する必要があります。 dsexcept.dll ファイルは、Dialog System の bin ディレクトリに保存されています。

注：

- dsexcept.dll では、ランタイム形式のスクリーンセットに関する情報は多く表示されません。

  これは、運用アプリケーションのユーザが、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したり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しますか ？

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などの複数の要因によって変化します。 スクリーンセットに使用されるメモリの量は、通常はあまり大きくありません。

すべてのスクリーンセットは、仮想ファイルに格納されます。仮想ファイルは、メモリ要求が増えると、ディスクにページングされます。 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ために必要な 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 (64 KB を超えている場合は、さらに 512 バイトが必要です）。 つまり、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は、スクリーンセットごとに 512 バイト以上が必要です。

十分なメモリがある場合は、RTS は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の量は変化しますが、極端に大きくなることはありません。

詳細については、ヘルプトピックの『Dialog System の制限』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、どうしたらよいですか？

Dialog System のテキストオブジェクトは、ウィンドウに描かれた単純なテキストです。そのため、色を設定できません。 ただし、エントリフィールドオブジェクトを使用して、色を設定できます。 エントリフィールドの表示専用属性を選択し、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、色を設定できます。この場合の色は、通常の方法で設定できる色に限定されます。

リストボックスが含まれるスクリーンセットをより速く表示するには、どのようにしたらよいですか？

リストボックスをマスタフィールドグループに設定している場合は、SET-DATA-GROUP-SIZE 関数を使用すると、そのグループで内部的に認識されたサイズを一時的に調整して、ロード時間を短縮できます。 この関数では、Dialog System に対して、そのグループ配列の (n) 項目のみを内部的に認識するように通知し、データがオブジェクトに挿入された数を超えないようにします。 ただし、呼び出し側プログラムは、そのグループのすべてのデータ項目にアクセスできます。必要に応じてこの関数を使用して、内部のサイズをリセットすることもできます。

複数のスクリーンセットを使用すると、制御されたループがアプリケーションで発生するのは、なぜですか ？

「Router」に基づくアプリケーションで DS-PUSH-SET と DS-USE-SET を組み合わせて使用すると、制御されたループが発生することがあります。 通常、このループは、コード化されたダイアログスクリプトに特有で、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用する場合は、次のダイアログを使用して、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 に基づいて、イベントが発生したスクリーンセットの再ロードを決定します。

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。 手続きが実行されると、指定された関数が実行され、元のスクリーンセットのイベントがさらに発生することがあります。この場合に、制御されたループが発生します。

この現象の理由については、『複数のスクリーンセット』の章にある『イベントの発生順序』を参照してください。

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 を設定しないでください。

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に、ドロップダウンリストのリスト項目が重複することがあるのはなぜですか ？

この状態は、主にコントロールの使用方法が不適切な場合に発生します。

選択ボックスコントロールを適切に使用するには、あらかじめ定義されたリストから項目を 1 つ選択します。 必要な場合は、ユーザが選択した項目を関連するマスタフィールドに配置し、アプリケーションで適切に使用できます。

重要な点は、選択ボックスを作成するダイアログ関数によって、16 進数の値「0A」で区切られたリスト項目が挿入されるということです。 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。

重複が発生するのは、関連付けられた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値 (区切り文字としての空白文字) が含まれており、X"0A" によって区切られていない場合です。 重複しているように表示されますが、実際には、項目が同じように区切られていません。

関連付けられたマスタフィールドが正しく作成されてない場合と REFRESH-OBJECT 関数によって、このような動作が発生します。

コントロールのタブ順を制御するには、どのようにしたらよいですか ？

[編集] メニューの [コントロール] を使用すると、必要に応じて、表示されたリストの順番を並べ替えます。 ヘルプトピックの『コントロール』を参照してください。

256 色のビットマップリソースを .dll にコンパイルできないのは、なぜですか ？

256 色のビットマップは、独立ファイルとして Dialog System アプリケーションに指定する必要があります。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

ここでは、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。

- エントリフィールド
- 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

また、次のサンプルプログラムも紹介します。

- Dsrnr
- Push-pop
- Custom1

エントリフィールド

エントリフィールドの詳細については、『コントロールオブジェクト』の章を参照してください。 ここでは、次の操作に必要なダイアログについて説明します。

- エントリフィールドの妥当性検査
- 複数行エントリフィールドの編集
- エントリフィールドとスクロールバーの使用 (『スクロールバーと関連付けられたイベント』を参照)

エントリフィールドの妥当性検査

エントリフィールドは、VALIDATE 関数を使用して妥当性検査します。 妥当性検査が失敗した場合は、VAL-ERROR イベントが発生します。 エントリフィールドは、次のように明示的に妥当性検査できます。

VALIDATE AMOUNT-DEPOSITED-EF

親ウィンドウを妥当性検査することによって、エントリフィールドを暗黙的に妥当性検査できます。 たとえば、次のように記述します。

VALIDATE DEPOSIT-WIN

フィールドが設定されたウィンドウまたはダイアログボックスのデータをユーザが承認した場合 (ユーザが [OK] ボタンまたは Enter キーを押した場合など) に、フィールドを暗黙的に妥当性検査することをお勧めします。 ユーザは、情報を入力し、見直してから、妥当性検査前に必要に応じて修正できます。 妥当性検査が失敗した場合は、エラーが発生し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
```

1 行目：

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処理が起動します。ウィンドウや特定のフィールドがフォーカスを失った場合に LOST-FOCUS イベントによって妥当性検査処理を起動することもできます。 たとえば、ユーザが Tab キーを使用して別のコントロールに移動した場合は、現在のコントロールのフォーカスが失われます。

2 行目：

VALIDATE $WINDOW

この文では、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は、VAL-ERROR イベントが発生します。 エラーが検出されない場合は、Dialog System の処理が通常どおりに続行され、次の関数が実行されます。

3 ～ 4 行目：

```
    INVOKE-MESSAGE-BOX ERROR-MB "All fields OK" $REGISTER
    RETC
```

VAL-ERROR イベントが発生しない場合のダイアログです。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。

5 行目：

```
 VAL-ERROR
```

VAL-ERROR イベントが検出されました。特殊レジスタ $EVENT-DATA には、妥当性検査に失敗したエントリフィールドが格納されます。

6 行目：

```
    INVOKE-MESSAGE-BOX ERROR-MB ERROR-MSG-FIELD $REGISTER
```

メッセージボックスを使用して、ユーザ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格納されます。 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します。

7 行目：

次のダイアログは、妥当性検査手順のコーディング例です。 このダイアログはプッシュボタンに設定します。

```
    SET-FOCUS $EVENT-DATA
```

妥当性検査に失敗したエントリフィールドに入力フォーカスを戻します。

警告：LOST-FOCUS イベントを使用すると、ユーザがフィールドから移動しようとしたときに、そのフィールドを妥当性検査できます。 フォーカスがそのフィールド設定されている場合に妥当性検査が失敗すると、ユーザ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。 このような場合は、ウィンドウ全体を妥当性検査することをお勧めします。

複雑なデータの妥当性検査

前で述べた手順では、簡単なデータの妥当性検査について説明しました。 ただし、Dialog

System では処理できない複雑な妥当性検査が必要な場合もあります。 たとえば、クレジットの限度額の範囲は、顧客と会社の取引年数によって決まることがあります。

このように、複数のフィールドにまたがった妥当性検査では、スクリーンセットに含まれないデータが必要な場合や Dialog System の妥当性検査規則より複雑な妥当性検査が必要な場合があるので、Dialog System から制御を移す必要があります。このようなチェックを行うには、呼び出し側プログラムに戻る必要があります。

例については、『高度なトピック』の章を参照してください。

複数行エントリフィールドの編集

アプリケーションプログラムまたはダイアログを使用して、関連付けたデータ項目にテキストを転記できます。

アプリケーションプログラムを使用したテキストの転記

アプリケーションプログラムで関連付けたデータ項目にテキストを転記する場合は、次のように記述します。

```
move "Now is the time..." to large-entry-field
```

この文では、複数行入力 (MLE) フィールドに関連付けられたデータブロック項目に文字列を転記します。

改行文字を挿入するには、16 進文字「0a」を使用します。 たとえば、次のように記述します。

```
move "line1" & x"0a" & "line2"
```

詳細は、次のとおりです。

line1 と line2	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	改行文字を示す 16 進文字です。

ダイアログを使用したテキストの転記

この場合は、改行文字を挿入できません。 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。

プッシュボタン

ここでは、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。

- [一時停止] プッシュボタン
- プッシュボタンと関連付けられた動的に変化するビットマップ

[一時停止] プッシュボタンのダイアログ

```
1 BUTTON-SELECTED
2     DISABLE-OBJECT $CONTROL
3     ENABLE-OBJECT CONTINUE-PB
4     BRANCH-TO-PROCEDURE PAUSE-SELECTED
```

1 行目：

    BUTTON-SELECTED

ユーザが [一時停止] ボタンを選択すると、BUTTON-SELECTED イベントが発生します。 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。

2 行目：

    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ための特殊なレジスタです。 この関数によって、現在のコントロール (ここでは [一時停止] プッシュボタン) を使用不能にします。 これによって、ユーザはこのボタンを使用できなくなります。

3 行目：

    ENABLE-OBJECT CONTINUE-PB

この文によって、[継続] プッシュボタンを使用可能にします。 ユーザは、このボタンを選択できるようになります。 CONTINUE-PB は、「プッシュボタンの属性」ウィンドウ内のプッシュボタンに割り当てられたプッシュボタン名です。

4 行目：

    BRANCH-TO-PROCEDURE PAUSE-SELECTED

PAUSE-SELECTED 手続きの関数が実行されます。 この手続きでは、フラグを設定し、呼び出し側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。

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：ビットマップを使用するには、あらかじめ Dialog System で使用できるようにする必要があります。 詳細については、『コントロールオブジェクト』の章を参照してください。

```
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     ...
```

2 ～ 3 行目：

```
 @SAVE
     BRANCH-TO-PROCEDURE SAVE-FUNCTION
```

ユーザがメニューから [上書き保存] オプションを選択したことを表すダイアログです。上書き保存機能を実行する手続きに分岐します。

5 行目：

```
     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 NULL 文字 (x"0A") を十分に格納できる大きさの英数字データ項目です。

6 行目：

```
SET-FOCUS GENERIC-WIN
```

キーボードの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最新表示され、処理が続行されます。

プログラムに次のような文を指定してデータブロックの save-states にデータを転記します。

```
move "save-normal" & x"0A" & "save-disabled" & x"0A" &
            "save-depressed" & x"0A"  to save-states.
```

save-normal、save-disabled、および save-depressed は、ビットマップの名前です。プッシュボタンの状態によってど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」の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、プッシュボタンが無効な場合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。

チェックボックス

チェックボックスの詳細については、『コントロールオブジェクト』の章を参照してください。 Objects サンプルプログラムでは、チェックボックスの使用例を紹介しています。

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択し、[表示] プッシュボタンをクリックして、選択した項目をリストボックスに表示する方法を説明します。図 17-1 を参照してください。

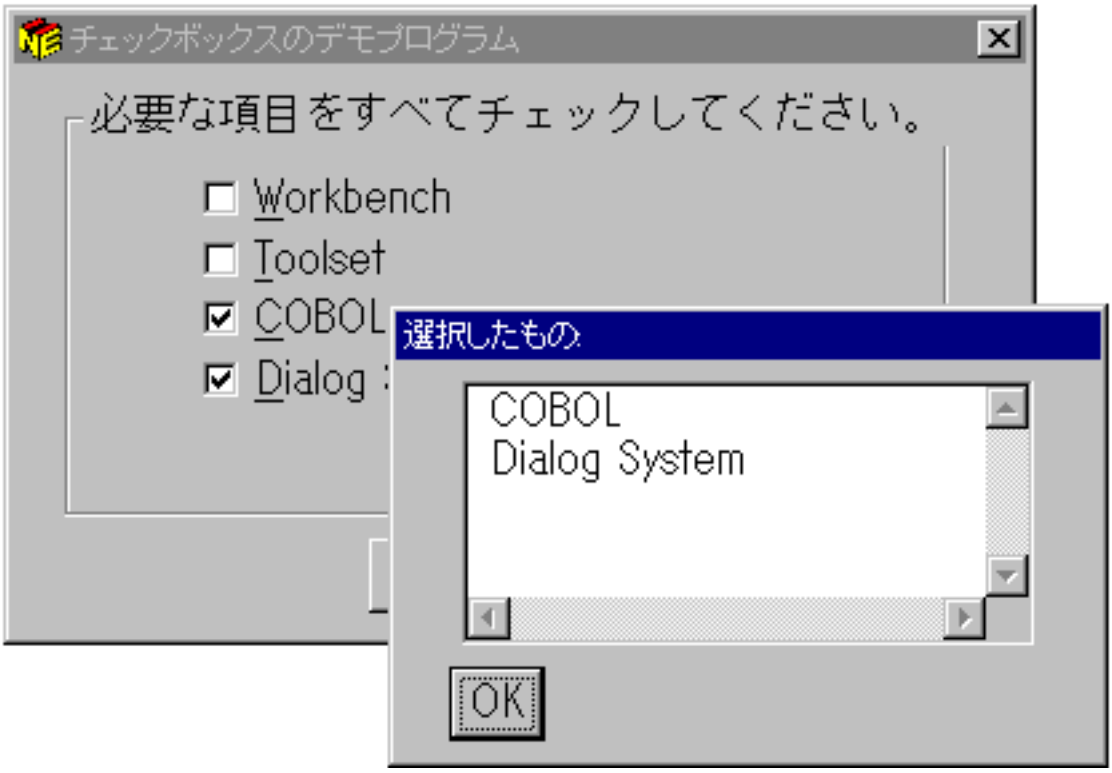

図 17-1： チェックボックスとリストボックス

このサンプルスクリーンセットのデータ定義は、次のとおりです。

```
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 ～ 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] プッシュボタンに設定) 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
```

1 行目：

```
 BUTTON-SELECTED
```

ユーザが [表示] プッシュボタンを選択したことを表す文です。

2 ～ 5 行目：

```
    MOVE "          " PRODUCT-DISPLAY(1)
    MOVE "          " PRODUCT-DISPLAY(2)
    MOVE "          " PRODUCT-DISPLAY(3)
    MOVE "          " PRODUCT-DISPLAY(4)
```

PRODUCT-DISPLAY 集団内の項目に空白文字列を転記することによって、リストボックスの項目を初期化します。

6 行目：

```
    MOVE 1 $REGISTER
```

指標 ($REGISTER) を初期化します。 $REGISTER は、数値を格納できる内部レジスタです。 ここでは、リストボックスへの指標として使用します。

7 行目：

```
    IF= WORKBENCH 0 CHECK-TOOLSET
```

「Workbench」チェックボックスがチェックされていない場合は、「Toolset」チェックボックスがチェックされているかどう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は、分岐しないで次の関数を実行します。

8 ～ 9 行目：

```
    MOVE "Workbench" PRODUCT-DISPLAY($REGISTER)
    INCREMENT $REGISTER
```

ユーザが「Workbench」チェックボックスを選択した場合に、この文が処理されます。リストボックスのデータ項目にデータを書き込み、指標の値を増加します。

10 行目：

```
    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。

12 ～ 27 行目：

```
  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
```

このダイアログは、他のチェックボックスのダイアログと似ています。

29 ～ 31 行目：

```
  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。

リストボックス

リストボックスの項目を更新するには、次の 3 つの方法があります。

- 集団項目を使用する
- 実行時に項目を更新するダイアログを使用する
- 区切られた文字列を使用する

集団項目を使用した項目の追加

リストボックスを集団項目と関連付けて、リストボックスの各行が集団のオカレンスを示すようにできます。 集団内で選択された項目が各行のフィールドになります。

Saledata サンプルアプリケーションは、リストボックスへのアクセス方法を示しています。 アプリケーションでは、既存のファイルを使用してデータブロックの集団項目 (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したりできます。

このデータブロック節では、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
```

次のダイアログでは、リストボックスを処理します。

```
 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
```

1 行目：

```
   SET-DATA-GROUP-SIZE SALES-GROUP NUMBER-OF-RECORDS
```

この文では、データ集団の内部のサイズを定義します。 アクセス可能な集団のオカレンスの数を指定します。 内部サイズの定義については、ヘルプの SET-DATA-GROUP-SIZE 関数に関する説明を参照してください。

3 ～ 4 行目：

```
 ITEM-SELECTED
    MOVE $EVENT-DATA $REGISTER
```

選択したい項目が見つかるまで、リストボックス内を参照できます。項目 (リストボックスの行) が選択されると、 ITEM-SELECTED イベントが発生します。 特殊イベントレジスタ $EVENT-DATA には、選択された項目の行番号が格納されています。 集団データとリストボックスの両方で現在の位置をトレースするには、$REGISTER を使用します。

6 行目：

PROC-INSERT

この手続きでは、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。

7 行目：

```
INSERT-OCCURRENCE SALES-GROUP $REGISTER
```

空白のオカレンスをデータ集団内の指定された位置に挿入します。 選択された項目とその項目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。この関数では、データ集団を更新するのみです。

8 ～ 10 行目：

```
MOVE TMP-NAME SALES-NAME($REGISTER)
MOVE TMP-REGION SALES-REGION($REGISTER)
MOVE TMP-STATE SALES-STATE($REGISTER)
```

集団項目を新しい情報で更新します。

11 行目：

```
INCREMENT NUMBER-OF-RECORDS
```

データ集団内のレコード数を増加します。

12 行目：

```
INCREMENT $REGISTER
```

ポインタを現在の行に更新します。 これによって、ポインタは挿入前と同じ行を指し示します。

13 行目：

```
REFRESH-OBJECT SALES-LB
```

データの変更を反映する前に、集団項目と関連付けられたリストボックスを最新表示する必要があります。 INSERT-OCCURRENCE 関数と DELETE-OCCURRENCE 関数は、リストボックスには影響を与えません。 最新表示する必要がある項目がリストボックス内に 1 つしかない場合は、UPDATE-LIST-ITEM 関数を使用できます。 ただし、挿入する場合は、データ集団内に挿入した行より後のすべての行が変更されます。 そのため、REFRESH-OBJECT 関数を使用することをお勧めします。

14 行目：

```
SET-LIST-ITEM-STATE SALES-LB 1 $REGISTER
```

現在の行の状態を「選択」にします。 この行は、挿入前に選択されていた行と同じです。

16 行目：

```
PROC-CHANGE
```

この手続きでは、選択された項目の内容を変更します。

17 ～ 19 行目：

```
MOVE TMP-NAME SALES-NAME($REGISTER)
MOVE TMP-REGION SALES-REGION($REGISTER)
MOVE TMP-STATE SALES-STATE($REGISTER)
```

集団項目を新しい情報で更新します。

20 行目：

```
UPDATE-LIST-ITEM SALES-LB SALES-GROUP $REGISTER
```

変更するのはリストボックス内の 1 行のみなので、UPDATE-LIST-ITEM 関数を使用できます。 他のすべての行は変更されていないので、リストボックス全体を最新表示する必要はありません。

21 行目：

```
SET-LIST-ITEM-STATE SALES-LB 1 $REGISTER
```

現在の行の状態を「選択」に変更します。 この行は、変更前に選択された行と同じです。

24 行目：

```
DELETE-OCCURRENCE SALES-GROUP $REGISTER
```

オカレンスをデータ集団内の指定された位置から削除します。 それ以降のすべての項目が 1 行のみ上に移動します。

25 行目：

```
DECREMENT NUMBER-OF-RECORDS
```

データ集団内のレコード数を減少します。

26 行目：

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。

27 行目：

REFRESH-OBJECT SALES-LB

リストボックスを最新表示します。

28 行目：

SET-LIST-ITEM SALES-LB 1 $REGISTER

現在の行の状態を「選択」に変更します。 この行は、削除された行の次の行と同じです。

ダイアログを使用した項目の追加

ダイアログ関数 INSERT-LIST-ITEM、UPDATE-LIST-ITEM、および DELETE-LIST-ITEM を使用して、実行時にリストボックス内の項目を管理できます。.

次のダイアログは、月の省略形を 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
```

リストの選択項目が少ない場合に、データブロックを小さくするには、この方法でリスト

ボックスに項目を追加することをお勧めします。 この方法で小さなリストボックスに項目を追加する場合は、あまり時間がかかりません。ただし、多少の時間は必要で、かつ、検索するダイアログ文の数が増加します。 ダイアログを検索する際の規則については、『ダイアログの使用方法』の章にある『Dialog System によるイベントダイアログの検索方法』を参照してください。

または、次のダイアログも使用できます。

```
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 によって、Dialog System でリストの末尾にデータ項目が挿入されます。

区切られた文字列を使用した項目の追加

リストボックスに少数の項目を追加する別の方法として、プログラムから渡されたデータ項目と INSERT-MANY-LIST-ITEMS 関数を使用できます。 たとえば、次のダイアログを指定した場合も、同じ月の省略形を MONTH-LB に挿入できます。

```
SCREENSET-INITIALIZED
    INSERT-MANY-LIST-ITEMS MONTH-LB MONTHS-STRING 48
```

パラメータの詳細は、次のとおりです。

MONTH-LB	「リストボックスの属性」ダイアログボックスで定義し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。各リスト項目は、x"0A" によって区切られています。 このデータ項目は、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。 このパラメータの詳細については、ヘルプトピックの『INSERT-MANY-LIST-ITEMS』を参照してください。

プログラムの作業場所節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転記します。

```
move months to months-string
```

スクロールバー

ここでは、次の内容について説明します。

- スクロールバーと関連付けられたイベント
- スクロールバーの属性

スクロールバーと関連付けられたイベント

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したときに発生します。

この機能は、次のようなダイアログで指定します。

```
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
```

1 行目と 3 行目では、スライダの位置をエントリフィールド COUNTER-DISP に表示します。

SLIDER-RELEASED イベントは、スライダが放されたときに発生します。 ダイアログの 4 ～ 6 行目では、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。 ダイアログの詳細は、次のとおりです。

1 行目：

```
    SLIDER-MOVING
```

スライダが新しい位置に移動したことを表します。SLIDER-MOVING イベントが発生します。

2 行目：

```
    MOVE $EVENT-DATA COUNTER
```

スライダを動かすと、特殊レジスタ $EVENT-DATA にスライダの新しい位置が格納されます。 これは、最小値と最大値に対する相対的な位置です。 COUNTER は、エントリフィールド COUNTER-DISP に設定されたデータブロックのデータ項目です。

3 行目：

```
    REFRESH-OBJECT COUNTER-DISP
```

新しいデータ値を反映できるようにエントリフィールドを最新表示する必要があります。

4 行目：

```
    SLIDER-RELEASED
```

スライダが新しい位置で放されました。SLIDER-RELEASED イベントが発生します。

5 行目：

```
    MOVE $EVENT-DATA COUNTER
```

スライダが放されると、$EVENT-DATA に新しいスライダ位置が格納されます。 ここでも

COUNTER は、エントリフィールド COUNTER-DISP に設定されたデータブロックのデータ項目です。

6 行目：

```
REFRESH-OBJECT COUNTER-DISP
```

エントリフィールドを最新表示します。

Objects サンプルアプリケーション内のスクロールバーの例では、この方法でエントリフィールドを使用しています。

スクロールバーの属性

「スクロールバーの属性」ダイアログボックスでは、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、次のダイアログでは、スクロールバーの EMP-LIST-SB という属性を変更します。

```
SET-SLIDER-RANGE EMP-LIST-SB 0 50
SET-SLIDER-POSITION EMP-LIST-SB 25
SET-SLIDER-SIZE EMP-LIST-SB 5
```

これらの関数については、ヘルプトピックの『ダイアログ文：関数』を参照してください。

スクロールバーの定義については、ヘルプトピックの『オブジェクトと属性』を参照してください。

タブコントロール

COPY-PAGE 関数や DELETE-PAGE 関数を使用すると、タブコントロールからページを挿入したり、削除したりできます。 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

1 ページの販売情報では不足する場合は、次のように 2 ページ目を追加できます。

```
1    ADD-MORE-SALES-INFO
2      COPY-PAGE SALES-INFO-PAGE 0 2
3      SET-OBJECT-LABEL SALES-INFO-PAGE
                              "Sales Information 2"
```

1 行目：

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。

2 行目：

```
    COPY-PAGE SALES-INFO-PAGE 0 2
```

COPY-PAGE では、タブコントロールページをコピーし、タブコントロールの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 は末尾、1 は先頭を表します。 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エントリフィールドより優先されます。 ただし、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限り常に表示されます (次の説明を参照)。

3 行目：

```
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 の後の SALES-INFO-PAGE は、元のページではなく、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 関数を使用します。 たとえば、顧客の販売情報がない場合は、次のようにしてページを削除できます。

```
BUTTON-SELECTED
   DELETE-PAGE SALES-INFO-PAGE
```

これらの関数については、ヘルプを参照してください。

呼び出しインターフェイス

呼び出しインターフェイスについては、『スクリーンセットの使用方法』の章を参照してください。 ここでは、次の内容について説明します。

- Dsrnr の使用方法
- 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スタンスの使用方法

Dsrnr の使用方法

Dsrnr は、Dialog System に組み込まれたサンプル副プログラムです。 Dsrnr を起動するには、次の手順を実行します。

1. Dsrunner を起動します (詳細については、『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. ファイル選択ウィンドウが表示されたら、Dsrnr (デモンストレーションディレクトリにある) を選択します。

   Dsrnr のメインウィンドウが表示されます。 Dsrnr を何回も開くと、複数のインスタ

ンスをロードできます (スタックには、最大 32 個のスクリーンセットをロードできます)。 副プログラムは、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説明します。

```
 14 linkage section.
 15
 16* スクリーンセットのデータブロック
 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、データブロックのコピーファイルをプログラムにコピーする必要があります。 また、Dsrunner と副プログラム、 dsrunner.cpy の間のリンクを表す dsrunner コピーファイルをコピーすることもできます。 Panels2 イベント情報を処理したい場合は、dssysinf.cpy もコピーする必要があります。 これらのファイルは、副プログラムのパラメータなので、連絡節にコピーする必要があります。

```
 24 procedure division using data-block
 25                         dsrunner-info-block
 26                         ds-event-block.
```

data-block パラメータを指定する必要があります。 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は、dsrunner-info-block を使用します。 Panels V2 と Dialog System をともに使用する場合は、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は、メッセージを表示する。
 30     if (address of data-block = null)
 31        display "このプログラムは、副プログラムであり、"
 32        display "(DSRUNNER から) 呼び出される必要があります。"
 33        exit program
 34        stop run
 35     end-if
```

副プログラムが呼び出されることを確認します。確認するには、データブロックのアドレスが副プログラムに渡されるかどうかを調べます。

```
 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。 その場合は、データブロックで low-values を
41* 調べる。この値は、DSRUNNER によって
42* すべてのデータブロックとともに初期化される。
43     if (data-block = all low-values)
44* このスクリーンセットとデータブロックで呼び出されたのは、
45* これが初めてである。初期化を行い
46* 終了する。ここから戻るまでは、スクリーンセットがロードされず、
47* SCREENSET-INITIALIZED イベントも
48* 発生しない。
49         perform initialisation
50     else
51* これが初めての呼び出しでない場合は、
52* 通常のとおり処理する。
53         perform handle-screenset-request
54     end-if
55
56     continue.
57
58 exit-main.
59
60     exit program
61     stop run.
```

副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化する必要があります。 43 行目は、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) を使用して副プログラムを呼び出したのが初めてかどうかを調べる方法を示しています。 副プログラムが初期化を終了したら、Dsrunner に戻る必要があります。 この副プログラムに関連付けられたスクリーンセット内で RETC が発生すると、Dsrunner はユーザプログラムを呼び出します。

```
64 initialisation section.
65* スクリーンセットを使用する前に初期化する。
66
67     initialize data-block
68     move dsrunner-screenset-instance to my-instance-no
69
70 continue.
```

68 行目は、副プログラムでインスタンス番号を検索する方法を示しています。

```
73 handle-screenset-request section.
74* スクリーンセットで RETC が実行されるか、または、
75* DSGRUN エラーが発生した場合は、ここで終了する。終了前に、
76* DSRUNNER によってこのプログラムが呼び出され、エラー処理が実行される。
77
78* スクリンセットでエラーが発生したために呼び出されたのかを調べる。
```

```
79     if (dsrunner-error-code not = 0)
80* その場合は、エラーを処理する。
81         perform handle-dsgrun-error
82         exit section
83     end-if
84* 妥当性検査エラーが原因で呼び出されたのかを調べる。
85     if (dsrunner-validation-error-no not = 0)
86* その場合は、エラーを処理する。
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 である必要があるので、
92* スクリーンセット要求を処理する。
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"
114                                to program-string
115     when other
116       move "sorry, unsupported function"
117                                to program-string
118    end-evaluate
119
120    continue.
```

このコードでは、この副プログラムに関連付けられたスクリーンセット内で RETC ダイアログ関数に続くスクリーンセット要求を処理します。

79 行目と 85 行目では、Dsrunner エラーまたは妥当性検査エラーがないかを調べます。 エラーが発生している場合は、エラー処理されます。エラーがない場合は、スクリーンセット要求が処理されます。

```
123 handle-dsgrun-error section.
124* スクリーンセットで DSGRUN エラーが発生した場合は、ここで終了する。
125* この例では、エラーメッセージを表示する。
126* これが始めて発生した場合、
127* 処理を続行する。初めてでない場合は、
128* RETURN-CODE を設定し、DSRUNNER によってこのダイアログを終了する。
129* return-code がゼロの場合は、DSRUNNER は、
130* 問題がない場合と同様に処理を続行する。
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.
```

このコードでは、dsrunner エラーを処理します。

143 行目は、ゼロ以外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発生した場合は、
151* ここで終了する。 ここでも上記のダイアログ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: 妥当性検査エラーコード" 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.
```

このコードでは、ユーザ入力エラーを処理します。

Push-pop 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用方法を示しています。

COBOL プログラム、スクリーンセット、および関連するファイルは、サンプルディスクに収録されています。 これらのプログラムのコンパイルと実行については、『スクリーンセットの使用方法』の章を参照してください。

Push-pop では、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します)

プログラム全体を次に示します。 プッシュとポップに使用する関数については、プログラムの後で説明します。

```
 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.
15  01 end-of-actions-flag      pic 9.
16     88 end-of-actions              value 1.
17
18  procedure division.
19
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.
26
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.
36
37  process-actions.
38      evaluate true
39        when pushmain-action = load-file
40          move "filemgr" to ds-set-name
41          move ds-push-set to ds-control
42          move 1 to ds-control-param
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
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。

1 ～ 7 行目：

```
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節では、適切な制御ブロックのコピーファイルと、使用される各スクリーンセットについて生成されたコピーファイルをコピーします。

9 行目：

```
9  01 new-screenset-name    pic x(12).
```

次に、new-screenset-name の PICTURE 文字列を定義します。

11 ～ 16 行目：

```
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。

20 ～ 25 行目：

```
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。 この手続きでは、ルーチン program-initialization を実行し、ds-control-block と pushmain スクリーンセットのデータブロックを使用して、Dsgrun を呼び出します。 end-of-actions が渡されて stop-run が発生するまで process-actions が実行されます。

27 ～ 35 行目：

```
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 手続きでは、制御ブロックとデータブロックを初期化し、スクリーンセット pushmain の適切な値を転記します。 30 行目では、ds-control に値「N」が格納されます。 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を起動した場合に使用します。

37 ～ 39 行目：

```
37   process-actions.
38       evaluate true
39         when pushmain-action = load-file
```

この節では、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。 メインスクリーンセットがロードされており、pushmain-action の値が変化するまで、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スクリーンセットがロードされます。

40 ～ 41 行目：

```
40   move "filemgr" to ds-set-name
41   move ds-push-set to ds-control
```

pushmain-action の値が load-file の場合は、

- スクリーンセット名 filemgr が ds-set-name に転記されます。
- ds-push-set の値が ds-control に転記されます。

注：ds-push-set は、ds-cntrl.mf 内のレベル-78 データ項目で、定義された値は「S」です。

---

43 ～ 49 行目：

```
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
```

ここでは、制御ブロックとデータブロックを初期化し、バージョン情報を確認します。 その後で、ファイルマネージャスクリーンセットを使用して Dsgrun を呼び出します。

51 ～ 61 行目：

```
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
```

ここでは、pushmain-action が 2 になった場合に、プリントマネージャに対して同じ関数を実行します。

62 ～ 63 行目：

```
62   when pushmain-action = exit-program
63   move 1 to end-of-actions-flag
```

pushmain-action の値が exit-program の場合は、値 1 が end-of-actions-flag に転記され、プログラムが終了します。

66 ～ 69 行目：

```
66   file-mgr-work.
67   move ds-quit-set to ds-control
68   call "dsgrun" using ds-control-block,
69                       filemgr-data-block
```

ファイル管理関数は、呼び出し側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。 この操作は、ds-quit-set を ds-control に転記することによって実行されます。 Dsgrun が呼び出されると、ds-control-block が使用され、filemgr-data-block が無視されます。

71 ～ 73 行目：

```
71   move ds-continue to ds-control
72   call "dsgrun" using ds-control-block,
73                       pushmain-data-block.
```

ds-continue が ds-control に転記され、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。 元の位置から処理が継続されます。

75 ～ 81 行目：

```
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.
```

プリント管理関数について、同じ操作が繰り返されます。

Custom1 サンプルプログラム

custom1.cbl では、同じスクリーンセットの複数のインスタンスの使用方法を説明します。

これは長いプログラムなので、複数のインスタンスの使用に関係する部分のみを説明します。 COBOL プログラム、スクリーンセット、および関連するファイルは、サンプルディスクに収録されています。

このプログラムでは、Custom1 というメインスクリーンセットと 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。Custom2 スクリーンセットは、データ集団内の項目にマップされます。

45 ～ 46 行目：

```
45   78 main-ss-name              value "custom1".
46   78 instance-ss-name           value "custom2".
```

プログラム内の作業場所節に、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を表す PICTURE 文字列を設定します。

48 ～ 51 行目：

```
48   copy "ds-cntrl.mf  ".
49   copy "custom1.cpb  ".
50   copy "custom2.cpb  ".
51   copy "dssysinf.cpy ".
```

いくつかのコピーファイルをプログラムの作業場所節にコピーします。 dssysinf.cpy がコピーされることに注意してください。 このコピーファイルは、複数のインスタンスを使用する場合に必要です。

53 ～ 54 行目：

```
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.
```

データ集団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。

57 ～ 61 行目：

```
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最新表示す

るために使用します。

63 ～ 64 行目：

```
63   01 main-screenset-id            pic x(8).
64   01 instance-screenset-id         pic x(8).
```

main-screenset-id と instance-screenset-id の PICTURE 文字列を定義します。

66 行目：

```
66   01 temp-word                    pic 9(4) comp-x.
```

temp-word PICTURE 文字列には、アクティブなデータブロックを示す値を格納します。 この値は、ファンクションコードです。 この値はプログラムの後半で適用され、temp-word の値が評価されて、適切な処理が実行されます。

103 行目：

```
103   move data-block-ptr(1:2) to temp-word(1:2)
```

複数のスクリーンセットを使用する場合の問題点は、どのスクリーンセットのデータブロックがアクティブであるかを判断することです。 このコードでは、データブロックの最初の 2 バイトを temp-word に転記します。temp-word は、ファンクションコードとして使用されます。

104 ～ 143 行目：

```
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。

271 ～ 287 行目：

```
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)、および ds-event-block を使用して、Dialog System を呼び出します。 Dialog System を呼び出せない場合は、エラーが表示されます。

284 ～ 297 行目：

```
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
```

Dsgrun にパラメータを転記すると、OTHER-SCREENSET イベントが発生した場合に新しい

スクリーンセットに移動します。 最初のスクリーンセットに移動して、ds-use-instance-set のかわりに ds-use-set を転記するかどうかを確認できます。 スタック上にスクリーンセットのインスタンスが 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が返されます。

319 ～ 329 行目：

```
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
```

必要な集団のオカレンスがインスタンス化されているかを確認します。

331 ～ 341 行目：

```
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
```

集団のオカレンスが見つかった場合は、そのオカレンスにフォーカスを移動します。

344 ～ 348 行目：

```
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
```

集団のオカレンスが検出されない場合は、新しいスクリーンセットのオカレンスを作成します。

353 ～ 361 行目：

```
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
360                             to address of data-block
361     perform call-dialog-system
```

最初のウィンドウを表示するために、手続き init-proc を ds-procedure に転記します。 制御が戻ったときに、データブロックポインタのアドレスを設定します。

365 ～ 369 行目：

```
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、および、このスクリーンセットインスタンスが処理する集団の行を格納します。

370 ～ 376 行目：

```
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。

377 ～ 389 行目：

```
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)
```

スクリーンセットの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 を設定して、スクリーンセットが正しい集団オカレンスを更新できるようにします。 次に group-record-no に 0 を転記することによって、インスタンステーブルの値をクリアします。

394 ～ 401 行目：

```
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、リストボックスを最新表示します。

408 ～ 422 行目：

```
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。

427 ～ 428 行目：

```
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後で、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

ここでは、Dialog System を使用して、『Dialog System の概要』の章で説明した手順でアプリケーションを作成します。

ここでは、各種の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参加できるロードレースを管理する場合を想定します。 参加申込書を処理し、各参加者に一意の番号を割り当てるようなシステムのインターフェイ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。

インターフェイ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。

データ定義のサンプル

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。 ここでは、このモデルを使用してデータを定義する方法について説明します。

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用にマスタフィールドを作成します。 各エントリフィールドにマスタフィールドを関連付け、このマスタフィールドにエントリデータを保存します。

データブロックを定義するには、次の操作を実行します。

1. [スクリーンセット] メニューの [データブロック] を選択します。

   「データの定義」ウィンドウが表示されます。

2. [オプション] メニューの [プロンプトモード] を選択します。

   「データ型」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、「整数」に「15」を入力します。

   これらの操作によって「Name」フィールドが定義されます。このフィールドは、次の手順で定義する「D-NAME」というオブジェクトのマスタフィールドになります。
5. [OK] をクリックして、次のエントリを入力するための空白行を作成します。

こ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	形式とサイズ	説明
MALE	9(1)	オプション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で使用
CODE	X(3)	D-CODE のマスタフィールド
FLAG-GROUP		グループのはじめ。グループ反復 1。 フラグを保存するために使用。
EXIT-FLG	9(1)	ユーザが [終了] を選択したことを示すフラグ
SAVE-FLG	9(1)	ユーザが [保存] を選択したことを示すフラグ
CLR-FLG	9(1)	ユーザが [クリア] を選択したことを示すフラグ

サンプルプログラム用のデータブロックの定義は、ここまでです。「データの定義」ウィンドウは、図 18-1 のように表示されます。

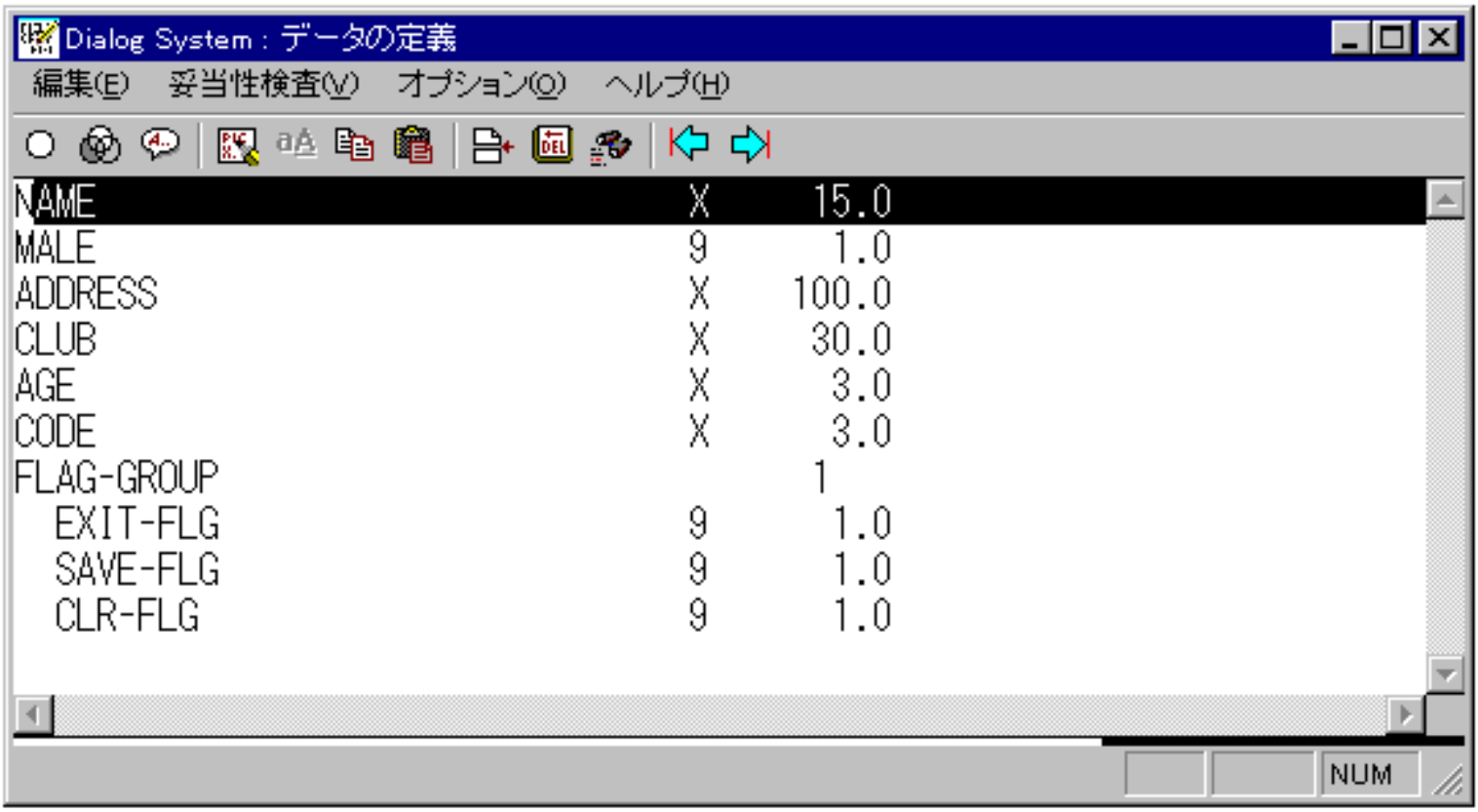

図 18-1：「データの定義」ウィンドウ

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 つのメッセージボックスが設定され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、コントロールオブジェクトにアクセスできないので、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前に、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] メニューで [含める] を選択します。

2. ドロップダウンメニューの [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、次の操作を実行します。

1. [オブジェクト] ツールバーの一次ウィンドウアイコンを選択するか、または、[オブジェクト] メニューの [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隅がデスクトップの左上付近にくるように、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。

   このウィンドウを Dialog System ウィンドウより大きく表示します。

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] タブをクリックして、[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。

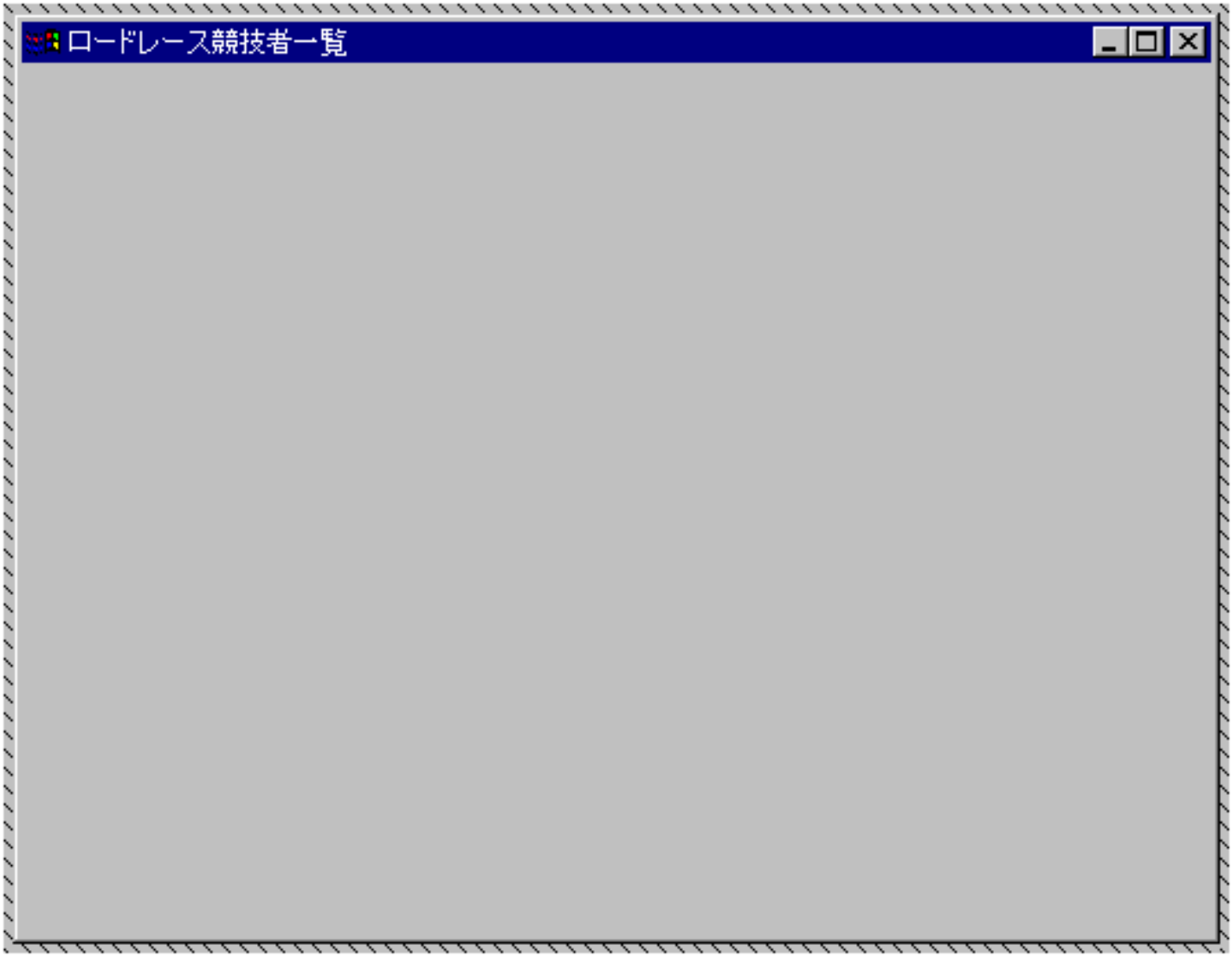

図 18-2： メインウィンドウの定義

サンプルコントロールオブジェクトの作成

次の手順で、次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. 図 18-3 のようにコントロールを一次ウィンドウに配置します。

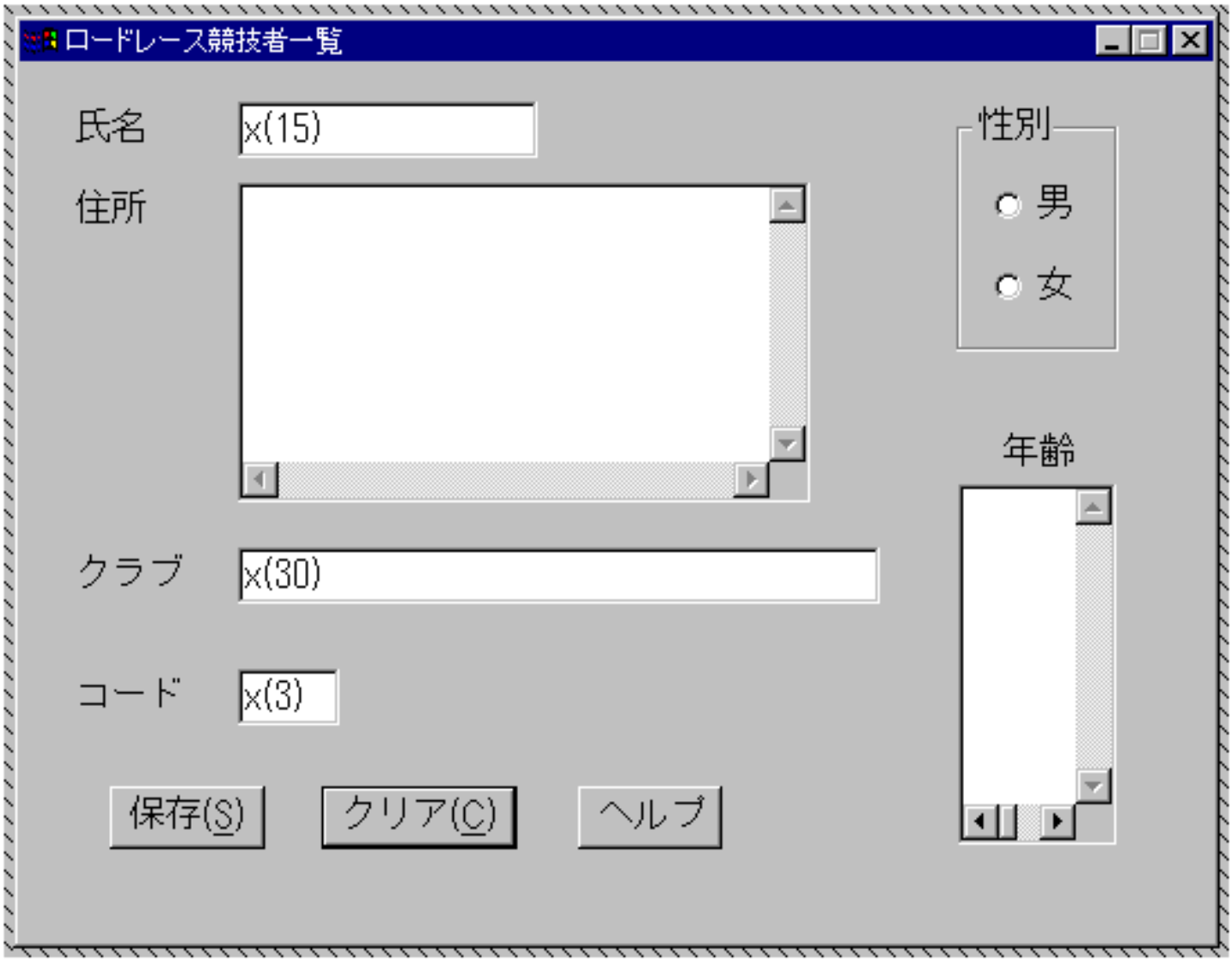

図 18-3： コントロールの追加

3. コントロールの「属性」ダイアログボックスで次のように属性を変更します。

   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	競技者の氏名を格納するためのフィールド。「名前」には「D-NAME」、「PICTURE 文字列」には「X(15)」、「マスターフィールド」には「NAME」を指定します。
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	競技者の住所を格納するためのフィールド。「名前」には「D-ADDRESS」、「長さ」には「120」、「マスターフィールド」には「ADDRESS」を指定します。
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格納するためのフィールド。「名前」には「D-CLUB」、「PICTURE文字列」には「X(30)」、「マスターフィールド」には「CLUB」を指定します。

オプションボタン	競技者の性別を表示するためのフィールド。「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。
オプションボタン	競技者の性別を表示するためのフィールド。「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 つのオプションボタンの上に配置します。「テキスト」に「性別」と入力します。
テキスト	年齢フィールドのラベル。「年齢」と入力します。
リストボックス	競技者の年齢を表示するためのフィールド。「初期テキスト定義済み」を選択します。[リストテキスト] をクリックして、年齢を 1 行に 1 つずつ指定します (「10-18」、「19-35」、「36-40」、「41-45」、「46-50」、「51-55」、「56-60」、「60+」)。「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	広告コードを格納するためのフィールド。「名前」には「D-CODE」、「PICTURE 文字列」には「X(3)」、「マスターフィールド」には「CODE」と入力します。
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「保存」を指定し、「デフォルトボタン」を有効にして、初期状態を「使用可能」に設定します。
プッシュボタン	現在の入力をクリアするためのボタン。「テキスト」に「クリア」と入力し、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」と入力し、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、ウィンドウの表示状態を整えます。

オプションボタンを正しく動作させるためには、コントロールグループに含める必要があります。 同様にプッシュボタンもコントロールグループに含める必要があります。

コントロールグループを定義するには、次の操作を実行します。

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隅を適切な場所に置きます。
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. 再度クリックします。

メッセージボックスの作成

ユーザが [ヘルプ] ボタンをクリックしたときに、ヘルプメッセージを表示するための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」と入力します。
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。

サンプルスクリーンセットのオブジェクトの定義は、ここまでです。

スクリーンセットの保存

定義手順の各段階が終わるたびにスクリーンセットを保存することをお勧めします。 サンプルスクリーンセットを保存するには、次の手順を実行します。

1. このスクリーンセットは、まだ保存されていないので、[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ためのファイル名とディレクトリを入力する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。

一度サンプルスクリーンセットを保存すると、その後は、[上書き保存] を選択し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作するには、[名前を付けて保存] を使用して、新しい名前を付けて別のバージョンを作成します。

テスト

ダイアログを作成しなくてもスクリーンセットをテストして、デスクトップレイアウトの表示状態に問題がないか、フィールドが正しく設定されているかなどを確認できます。 サンプルスクリーンセットをテストするには、次の操作を実行します。

1. [ファイル] メニューの [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] メニューの [実行] を選択します。

   サンプルスクリーンセットは、図 18-4 のように表示されます。

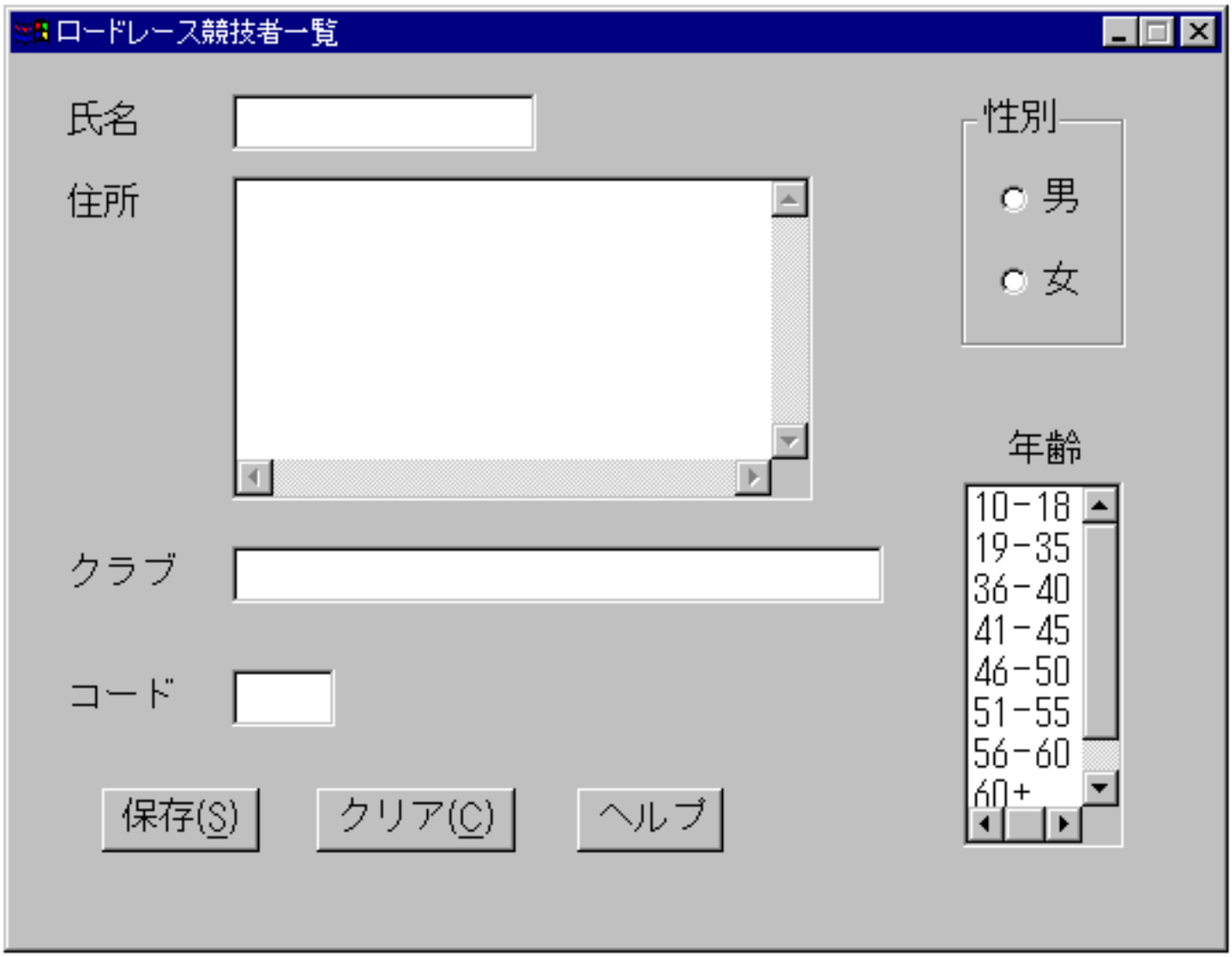

   図 18-4： スクリーンセット Animator からのスクリーンセットの実行

   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 16 文字以上を入力しようとすると、警告音が鳴ります。

   オプションボタンは、コントロールグループとしてまとめられているので、選択できるボタンは「男」または「女」のどちらかです。リストボックスから選択できる年齢は、1 つのみです。
3. Esc キーを押して、スクリーンセット Animator ウィンドウに戻ります。

   「RETC が今、実行されました」というタイトルのダイアログボックスが開きます。
4. [中断] 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 をより便利に使用できます。

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、ダイアログを定義する必要があります。 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に説明します。

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、次の操作を実行します。

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] メニューの [オブジェクトダイアログ] を選択します。

   「ダイアログの定義」ウィンドウが表示されます。 この「ダイアログの定義」ウィンドウには、BUTTON-SELECTED というデフォルトのイベントがすでに表示されています。
3. [オプション] メニューの [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エントリするためのプロンプトが表示されます。
4. カーソルキーを使用して 1 行ずつ移動します。

   入力する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、「Dialog System からエントリデータベース内のレコードの作成または変更を処理する手続きに制御を渡す」と入力します。
6. [保存] ボタンが押された場合の動作を指定します。
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」に設定するフラグの名前を指定します。

   「パラメータ 2」と「パラメータ 3」は、この関数では必要ないので、これらの領域は淡色表示されています。
10. ユーザが [保存] をクリックしたときに保存フラグが設定されるように、「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。 この関数では、呼び出し側プログラムに制御が戻ります。 このスクリーンセットを使用する COBOL プログラムでは、保存フラグをチェックし、フラグが設定されている場合に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設定できます。

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。 ダイアログの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の詳細については、『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、次を参照してください。

[クリア] ボタン

ダイアログ：

```
BUTTON-SELECTED
    SET-FLAG CLR-FLG(1)
    RETC
```

結果：

ユーザが [クリア] を押した場合は、クリアフラグが設定され、呼び出し側プログラムに制御が戻ります。 呼び出し側プログラムによって、入力フィールドがクリアされます (その結果、エントリが取り消されます)。

[ヘルプ] ボタン

ダイアログ：

```
BUTTON-SELECTED
    INVOKE-MESSAGE-BOX HELP-MSG $NULL $EVENT-DATA
```

結果：

ユーザが [ヘルプ] を押した場合は、作成したメッセージボックスが表示されます。 これにより、適切なヘルプテキストが表示されます。

「男」オプションボタン

ダイアログ：

```
BUTTON-SELECTED
    SET-FLAG MALE
```

結果：

ユーザが「男」オプションボタンを押した場合は、男性フラグに 1 が設定されます。これは、男性の競技者を表します。

「女」オプションボタン

ダイアログ：

```
BUTTON-SELECTED
    CLEAR-FLAG MALE
```

結果：

ユーザが「女」オプションボタンを押した場合は、男性フラグが 0 に設定されます。これは、女性の競技者を表します。

「年齢」リストボックス

ダイアログ：

```
ITEM-SELECTED
    RETRIEVE-LIST-ITEM $CONTROL AGE $EVENT-DATA
```

結果：

ユーザがリストから年齢層を選択した場合に、選択されたリスト項目が「AGE」というフィールドに格納されます。

サンプルグローバルダイアログの定義

[スクリーンセット] メニューの [グローバルダイアログ] を選択して、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。

デフォルトのグローバルダイアログは、次のとおりです。

```
ESC
    RETC
CLOSED-WINDOW
    RETC
```

次のように 2 行を追加します。

```
ESC
    SET-FLAG EXIT-FLG(1)
    RETC
CLOSED-WINDOW
    SET-FLAG EXIT-FLG(1)
    RETC
```

このダイアログでは、ユーザが Esc キーを押すか、または、システムメニューを使用して、ウィンドウを閉じた場合に、終了フラグが設定され、アプリケーションプログラム (呼び出し側プログラム) に制御が戻ります。

次のダイアログを追加します。

```
  REFRESH-DATA
    REFRESH-OBJECT MAIN
```

このダイアログでは、呼び出し側プログラムから Dialog System に対してメインウィンドウの最新表示要求があった場合に、その処理が実行されます。 アプリケーションと呼び出し側インターフェイスとの対話については、『スクリーンセットの使用方法』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
```

このダイアログでは、1 番目のオプションボタン (男) をデフォルトとして設定します。 オプションボタンのグループを入力する場合は、どれかのオプションボタンを最初に設定しておく必要があります。

ダイアログの使用方法については、『ダイアログの使用方法』の章を参照してください。

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、オプションボタンやリスト項目を選択したりします。 変更後に [上書き保存] 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。

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後に、さらに変更できます (スクリーンレイアウトの改良など)。 ここで述べた手順を繰り返して、スクリーンセットを十分に設計してください。

まとめ

ここでは、次の作業を行いました。

- 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
- 必要な手順の繰り返しによる設計

最後にサンプルスクリーンセットにあるユーザインターフェイスを利用する COBOL プログラムを作成する必要があります。 DialogSystem\demo\entries ディレクトリには、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保存されています。

詳細情報

次の『チュートリアル - サンプルスクリーンセットの使用方法』の章では、サンプルスクリーンセット用の COBOL プログラムを作成する際の注意点について説明します。 次では、ユーザ入力を読み取り、ユーザの指示に従って保存またはクリアする、簡単なプログラムのサンプルコードを使用します。

チュートリアル-サンプルスクリーンセットの作成
チュートリアル-ステータスバーの追加とカスタマイズ

第 19 章 : チュートリアル-サンプルスクリーンセットの使用方法

『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用のユーザインターフェイスを含むエントリスクリーンセットの作成方法について説明しました。 ここでは、エントリ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、次の手順に分けて説明します。

1. スクリーンセットから COBOL のコピー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 プログラムの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。

データブロックのコピーファイルの生成

データブロックのコピーファイルには、実行時に呼び出し側プログラムから Dialog System に渡されるデータブロックの定義が記述されています。 コピーファイルは、呼び出し側プログラムに指定する必要があります。 このコピーファイルには、バージョン確認情報も記述されています。

Dialog System を使用すると、スクリーンセットからコピーファイルを生成したり、コピーファイルの生成方法に関するオプションを設定したりできます。

サンプルスクリーンセットのコピー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、コピー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先頭に「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、これらの接頭辞付きのデータ名を使用します。

5. Enter キーを押すか、または、[OK] をクリックして、このダイアログボックスの他のデフォルト値をそのまま使用します。

6. [ファイル] メニューの [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. コピー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。

   設定したコピーファイルオプションによって、サンプルスクリーンセット用のコピーファイルが生成されます。

オプションの選択とコピーファイルの生成

サンプルコピーファイルは、すでに生成しました。

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、デモンストレーションプログラムとして Dialog System ソフトウェアに組み込まれています。 ファイル名は、entries.cbl です。

条件やプログラミングスタイルによっては、別の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 * (フラグに) 返されたユーザ操作を処理する。
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格納されている場合は、
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化して、現在の詳細をクリアする。

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 が次に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。
74 * この手続きでは、データブロックの値によって
75 * メインウィンドウが最新表示される。

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.
```

1 ～ 22 行目：

```
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).
```

これらの行では、ユーザによるエントリを格納するためのレコードを設定します。

23 ～ 27 行目：

```
23  procedure division.

24  main-process section.
```

```
25      perform program-initialize
26      perform program-body until entry-exit-flg-true
27      perform program-terminate.
```

これらの行では、プログラム全体の構造を定義し、ユーザが Esc キーを押すか、または、システムメニューを使用して、ウィンドウを閉じたときに、プログラムが終了します。 これらの動作では、スクリーンセットに終了フラグが設定されます。 フラグの設定は、処理のためにプログラムに渡されます。

28 ～ 35 行目：

```
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.
```

データブロックのコピーファイルには、ユーザデータのみではなく、Dialog System が呼び出されたときに確認するスクリーンセットのバージョン番号も記述されています。 バージョン番号を確認するには、Dialog System のランタイムシステムを呼び出す前に、呼び出し側プログラムでこれらのバージョン番号を制御ブロック内のデータ項目にコピーする必要があります。

呼び出し側プログラムを作成する場合は、copy "ds-cntrl.mf" という文を使用して、プログラムの作業場所節にコピーファイルをコピーする必要があります。 ANSI-85 準拠の COBOL を使用している場合は、コピーファイル ds-cntrl.ans を使用する必要があります。

また、スクリーンセット名と Dialog System の動作を制御する情報が、制御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。

36 ～ 46 行目：

```
36  program-body section.
37 * (フラグに) 返されたユーザ操作を処理する。
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の操作を指示するのではなく、ユーザがプログラムの次の動作を決めることができます。 データブロック内に設定されたフラグには、ユーザの操作に対応する値が Dialog System から返されます。この値に基づいて、プログラムで次の動作が決定されます。

プログラムは、次のようにさまざまな方法で対応できます。

- 保存された情報を修正します。

  save-record 節で実行されています。

- データベースから追加情報を検索します。

  このアプリケーションでは、実行されません。

- 結果またはエラーメッセージを表示します。

  call-dialog-system 節で実行されています (プログラムから Dialog System を呼び出すときにエラーが発生した場合)。

- アプリケーションの使用を支援します。

  このアプリケーションは、単純なので、すべてのヘルプは Dialog System のメッセージボックスによって処理されます。 ヘルプを処理する別の方法としては、ヘルプフラグが設定されたときにヘルプを表示するためのコードをプログラムに記述する方法もあります。

- ユーザが入力した情報を妥当性検査します。

  save-record 節では、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実行しています。 入力段階でプログラムと Dialog System の両方で、より複雑な妥当性検査を実行することも可能です。

- ユーザからの追加入力を要求します。

  レコードを保存または消去した後に、Dialog System に戻ります。

47 ～ 49 行目：

```
47  program-terminate section.
48      close entry-file
```

```
49      stop run.
```

これらの行では、エラーが発生した場合、または、ユーザが Esc キーを押すか、システムメニューを使用してウィンドウを閉じた場合に、プログラムを終了します。

50 ～ 63 行目：

```
50  save-record section.
51 * すべてのテキストフィールドに値が格納されている場合は、
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.
```

これらの行では、ユーザが [上書き保存] を押したときにユーザの入力を保存します。 このボタンを押すと、保存フラグが設定されます。このフラグは、プログラムの program-body 節でテストされます。 レコードが空の場合、入力は保存されません。

64 ～ 68 行目：

```
64  clear-record section.
65 * データブロックを初期化して、現在の詳細をクリアする。

66      initialize entry-record
67      initialize entry-data-block
68      perform set-up-for-refresh-screen.
```

これらの行では、ユーザが [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クリアします。 このボタンを押すと、クリアフラグが設定されます。このフラグは、program-body 節でテストされます。

69～76 行目：

```
69  clear-flags section.
70      initialize entry-flag-group.
```

```
71  set-up-for-refresh-screen section.

72 * Dialog System が次に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。
74 * この手続きでは、データブロックの値によって
75 * メインウィンドウが最新表示される。

76      move "refresh-data" to ds-procedure.
```

これらの行では、フラグをクリアし、次のユーザの入力に備えて Dialog System によって画面を最新表示します。

77 ～ 89 行目：

```
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.
```

これらの行では、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 番目の節では、呼び出しエラーをチェックし、エラーが発生している場合は、プログラムを終了します。

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、および、アニメートできます。

アプリケーションのデバッグ中に、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。 コードのアニメート中は、各文が実行されるに従って、ソースの各行が次々にハイライトされ、各文の結果が表示されます。 プログラムの実行速度を制御したり、実行を中断してデータ項目を確認および変更したりできます。 詳細については、ヘルプトピックの『デバッグ』を参照してください。

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、さまざまな付随作業を行う必要があります。

Net Express から提供されるプロジェクト機能を使用して、Dialog System アプリケーションをビルドします。 詳細については、ヘルプトピックの『アプリケーションのビルド』を参照してください。

ヘルプトピックの『コンパイル』では、アプリケーションを運用するための準備について説明しています。

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。 最終製品は、フロッピーディスクにコピーしたり、顧客に送付し、他のマシンにロードして、アプリケーションとして実行できます。

最終製品は、アプリケーションのサイズによって、次の一方または両方で構成されます。

- 実行可能モジュール。 これらのファイルは、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。 これらのファイルは、ファイル入出力やメモリ管理などの機能を実行します。

チュートリアル-サンプルスクリーンセットの使用方法
チュートリアル-メニュー/ツールバーの追加とカスタマイズ

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

ここでは、Dialog System に付属する Customer サンプル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方法を説明します。 どのスクリーンセットのどのウィンドウにも同じ手順で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を使用できます。 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前には、次の準備が必要です。

- 『カスタムコントロールのプログラミング』の章をすべて読んでください。

ユーザ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、次の資料も参照してください。

- ヘルプの『オブジェクトと属性』

  ユーザコントロールに関するトピックでは、ユーザ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

ステータスバーのコントロールプログラムの使用例は、sbards.gs スクリーンセットに含まれています。また、DialogSystem\demo\sbards サブディレクトリの sbards.txt ファイルに説明が記述されています。

設定

1. この作業では、Customer スクリーンセットに多数の変更を加えるので、作業を始める前に customer.gs のバックアップを作成してください。

2. Net Express を起動します。

3. DialogSystem\demo\customer サブディレクトリの customer.app を開きます。

4. 左側のペインで customer.gs を右クリックし、コンテキストメニューの [編集] を選択します。

   Dialog System が起動し、「顧客情報」ウィンドウが表示されます。

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前に、スクリーンセットのデータブロックで共通のデータ領域を定義します。 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受け渡すために使用されます。

データ項目の定義

スクリーンセットで定義する各ステータスバーには、データブロック項目 (マスターフィールド) を関連付ける必要があります。 このデータ項目には、作成したコントロールのクラスライブラリオブジェクト参照を格納するので、OBJ-REF として定義します。 データ項目は、ステータスバーを追加する前に定義する必要があります。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は、Dialog System でステータスバーを定義できません。

ステータスバーのオブジェクト参照を格納するスクリーンセットのデータブロックに、次のデータ項目を指定します。

```
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 のデータ定義を指定します。

```
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できます。 [ファイル]、[インポート]、[スクリーンセット] の順に選択して、[OK] をクリックし、現在ロードされているスクリーンセットの上書きを許可します。 [ファイル] ボタンをクリックして、funcdata.imp をダブルクリックします。[インポート] ボタン、[OK]、[閉じる] の順にクリックします。

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一度定義するのみで十分です。

ステータスバーの定義

必要なデータを定義した後でステータスバーを定義します。

1. ステータスバーを追加する「顧客情報」ウィンドウ (MAIN-WINDOW) を選択します。

2. [プログラム済みコントロール] ツールバーの [ステータスバー] を選択します。

3. ステータスバーの位置とサイズを調整します。

   ステータスバーは、描画時の位置に関係なく、ウィンドウの下部に配置されます。
4. 「ステータスバー属性」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. ステータスバーに名前を付けます。 MAIN-WINDOW-STATUS-BAR のように、コントロールの意味がわかる名前にします。

    2. マスタフィールド名として MAIN-WINDOW-SBAR-OBJREF を指定します。

    3. ステータスバーコントロールプログラムの名前として SBAR2 を指定します。

    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] をクリックして、ステータスバーコントロールプログラム sbar2.cbl を生成します。

       メッセージボックスが開き、生成されたコントロールプログラムで必要なデータブロック内の項目が確認されます。 必要な項目は、前の手順で確認済みなので、[OK] をクリックして、作業を続けます。

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ます。

5. [OK] をクリックしてスクリーンセットを保存し、閉じます。

6. Net Express に戻り、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. Net Express の [プロジェクト] メニューの [すべてをリビルド] を使用して、生成されたコントロールプログラムをコンパイルし、このプログラムの実行可能バージョンを作成します。

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完了した後で、スクリーンセットを実行して、ユーザコントロールの動作を確認できます。

この段階では、ステータスバーが実質的に機能しません。ステータスバー上の時計とキーの状態は更新されません。

ステータスバーを正しく更新するためのダイアログを追加する必要があります。

ステータスバーの操作

各ユーザコントロールは、そのコントロールプログラムであらかじめ定義された関数を使用して操作できます。 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、関連付けられたデータの最新表示、削除、更新などの操作を実行できます。

以後では、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実装します。

- 時計の時刻
- Insert/Caps/NumLock キーの状態
- ウィンドウとステータスバーのサイズ変更
- マウスヒントテキスト

時計の時刻とキー状態の管理

時計とキーの状態情報を正確に管理するには、ステータスバーを定期的に最新表示する必要

があります。 この操作には、Dialog System のタイムアウト機能を使用します。

タイムアウト機能の使用方法

タイムアウトを設定するには、ステータスバーを含むウィンドウ (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。

1. 「顧客情報」ウィンドウを選択して、[オブジェクトダイアログ] に進みます。
2. [イベント] ツールバーボタンをクリックして、WINDOW-CREATED を選択します。
3. 次のダイアログを追加します。

   TIMEOUT 25 REFRESH-STATUS-BAR

   このダイアログによって、1/4 秒ごとに REFRESH-STATUS-BAR 手続き (後で定義します) を実行します。

   注：スクリーンセット Animator を使用し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 で REFRESH-STATUS-BAR 手続きがループ実行されているように表示されます。 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発生するとき (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ためです。 このため、スクリーンセット Animator を使用してデバッグする場合は、タイムアウト値を調整するか、または、この行をまとめて削除します。

   ---

4. 次に、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。 通常は、タイムアウトイベントの発生時に実行する手続きを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に、そのオブジェクトがフォーカスを失うと、Dialog System ランタイムでタイムアウト手続きが検出されなくなることがあります (この場合は、ランタイムエラー 8 が発生します)。

   次のダイアログでは、タイムアウトイベントが発生したときに (ステータスバーコントロールプログラムの REFRESH-OBJECT 関数を使用) ステータスバーが更新されます。 このダイアログを正しく実行するためには、グローバルダイアログに配置する必要があります。

   ```
   REFRESH-STATUS-BAR
     MOVE "REFRESH-OBJECT" CALL-FUNCTION(1)
     SET OBJECT-REFERENCE(1) MAIN-WINDOW-SBAR-OBJREF
   ```

```
CALLOUT "SBAR2" 0 $NULL
```

ウィンドウとステータスバーのサイズ変更

ウィンドウのサイズを変更したときに、ステータスバーのサイズも正しく変更す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には、ウィンドウのサイズ変更、最大化、または、復元を実行する場合に呼び出す RESIZE 関数があります。

この関数は、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すると、ステータスバーが表示されなくなるためです。

注：ウィンドウのサイズを変更するには、「境界可変」属性を設定する必要があります。

ツールバーの [オブジェクトダイアログ] を選択し、ウィンドウのサイズを変更したときにステータスバーのサイズも変更されるように、ステータスバーを含む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
```

マウスヒントテキストの追加

ステータスバーを完全に機能させるには、マウスでウィンドウ内のオブジェクト上を移動するときに、ステータスバーに表示されるテキストが更新されるようにする必要があります。 この操作は、次のように実行します。

1. ステータスバーを含むウィンドウの MOUSE-OVER イベントを (SET-PROPERTY ダイアログ関数を使用) 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。

2. ステータスバーにテキストを設定するためのダイアログを追加して、MOUSE-OVER イベントに応答できるようにします。 MOUSE-OVER テキストは、通常、ステータスバーの左端のセクション (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
```

3. ステータステキストを更新す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して、オブジェクト上で MOUSE-OVER イベントが発生するたびにステータス行が更新されるようにします。

```
MOUSE-OVER
  MOVE "Status text" IO-TEXT-BUFFER(1)
  BRANCH-TO-PROCEDURE DISPLAY-HINT-TEXT
```

   このダイアログ内の「Status text」の部分に、ステータスバーに表示するテキストを指定します。

   たとえば、CUSTOMER スクリーンセットの MAIN-WINDOW 上にある [ロード] プッシュボタンの場合は、「Status Text」の部分に「レコードをロードします」を指定します。

スクリーンセットを実行すると、CUSTOMER スクリーンセットのステータスバーが次のように表示されます。

- 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 に現在の時刻が表示されます。

スクリーンセットの表示状態を確認した後で [中止] をクリックし、スクリーンセットを保存し、閉じます。

次では、ステータスバーに関するイベントの登録方法について説明します。

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、ユーザがステータスバーのセクションをマウスでクリックしたときに生成されるイベントを処理するためのコードがあります。 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して別のイベントを追加する方法については、『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』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通知します。 たとえば、ステータスバーコントロールプログラムは、SBar2Button1Down というエントリポイントをクラスライブラリとともに登録します。これによって、ユーザがステータスバーを左マウスボタンでクリックした場合に、エントリポイント Sbar2Button1Down にあるコードが実行されます。

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後に実行されます。 コードを指定する必要があるすべてのイベントについて、「Register-callbacks」節にコールバックを登録する必要があります。

イベントが生成されると、そのイベントに関連付けられたコールバックコードによって、データブロック内のデータが更新され、Dialog System スクリーンセットに戻ることができます。 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受け渡す方法については、ヘルプトピックの『コントロールプログラム』を参照してください。

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は、ここまでです。

同様の手順を使用して、スクリーンセットにコントロールプログラムを追加し、ダイアログを必要に応じてカスタマイズできます。

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、ステータスバーに機能を追加して、ユーザが時計セクションをダブルクリックした場合に、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方法について説明します。

定義時に作成したステータスバーコントロールプログラムをそのまま使用できます。また、このコントロールプログラムは、必要なウィンドウで多数のコントロールを作成および管理できるようにコード化されます。 また、次のような独自の機能を追加することもできます。

注：ステータスバーを含む各スクリーンセットには、そのスクリーンセットに対して固有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する必要があります。 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に、そのスクリーンセットに固有のコードが含まれているためです。

たとえば、ステータスバーにさらにセクションを追加したい場合を考えます。 このセクションに、イベントを追加するようにステータスバーコントロールプログラムを変更する方法を示すとします。

その場合に、sbar2.cbl というコントロールプログラムを生成してカスタマイズすると仮定します。

カスタマイズしたプログラムは、実行時に $COBDIR で必ず使用できるようにします。

新しいイベントのためのコールバックの登録

Left-mouse-button-double-click イベントをステータスバーに追加するには、次の 2 つの操作を実行します。

- 新しいイベントを登録するためのコードを Register-Callbacks 節に追加します。
- 新しいイベントを処理するためのコントロールプログラムを追加します。

新しいイベントに関するコールバックを登録するコードは、left-mouse-button-down イベントに関するコールバックを登録するコードとよく似ています。 新しいイベントの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の動作について説明します。

-

  MOVE ProgramID & "Button1Down " TO MessageName

  注：Program ID 定数は、このプログラムでエントリポイントの命名に使用されます。 この定数によって、複数のコントロールプログラムで重複した名前がロードされるのを防ぐことができます。

  コールバックを受け取るエントリポイントの名前 (ここでは、Button1Down) を

MessageName に格納します。

- INVOKE EntryCallback "new" USING MessageName
  RETURNING aCallback

  指定したエントリポイント名を使用して、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、Net Express システムの source\guicl\p2cevent.cpy ファイルに一覧表示されています。

- INVOKE aStatusBar "setEvent" USING i aCallback

  エントリポイント (ProgramID と「Button1Down」) にコールバックするためのイベント (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不要なので、削除できます (完成)。

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を追加します。 この場合は、コントロールプログラムの Register-Callbacks 節にコードを追加します。

left-mouse-button-double-click の登録コードを追加するには、次の操作を実行します。

1. Net Express 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の [編集] を選択します。
3. left-mouse-button-down イベントの登録コードと注釈の間に、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
   ```

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 イベントと類似しているので、left-mouse-button-down 節に既存のコードをコピーして、イベントを処理する節を追加します。 イベントによって異なるコードは、この節、エントリポイント名、および、ユーザイベント番号です。

コードの追加

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、前の説明のとおり、sbar2.cbl を開いて、次のように編集します。

1. left-mouse-button-down 節内のコードをすべて複製します。 新しい節に LEFT-MOUSE-BUTTON-DBLCLK という名前を付けます。

2. LEFT-MOUSE-BUTTON-DBLCLK 節のエントリポイント名を Button1Down から DblClk1 に変更します。

3. LEFT-MOUSE-BUTTON-DBLCLK 節のユーザイベント番号を 34590 から 34591 に変更します。

4. sbar2.cbl を保存します。

5. プロジェクトをリビルドします。

注：アプリケーションで使用するコントロールプログラムによって使用および生成されるユーザイベントが重複しないように、各ユーザイベント名を記録しておいてください。 ユーザイベントが重複していても警告が表示されません。ただし、同一のユーザイベントを同一のウィンドウで複数の目的に使用すると、プログラムが期待どおりに動作しません。

ダイアログの追加

この段階で、ダイアログを追加して、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して追加した新しいイベントを使用できます。

1. Net Express の customer.gs を右クリックして、コンテキストメニューの [編集] を選択します。

2. 「顧客情報」ウィンドウを右クリックして、コンテキストメニューの [ダイアログ] を選択します。

3. 新しいイベントを追加して、ステータスバーをダブルクリックした場合に警告音が鳴

るようにするには、次のコードを指定します。

```
USER-EVENT
   XIF= $EVENT-DATA 34591 SBAR2-DOUBLE-CLICK
 SBAR2-DOUBLE-CLICK
   BEEP
```

4. スクリーンセットを保存して実行します。

5. ステータスバーをダブルクリックして、警告音をテストします。

次の章では、メニューバーとツールバーの追加とカスタマイズについて説明します。

チュートリアル-ステータスバーの追加とカスタマイズ | チュートリアル-ActiveX コントロールの追加

第 21 章 : チュートリアル-メニュー/ツールバーの追加とカスタマイズ

ここでは、Dialog System に付属する Customer サンプル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方法を説明します。 メニューバーの追加手順は、前の章で説明したステータスバーの追加手順とほぼ同じです。

どのスクリーンセットのどのウィンドウにも同じ手順でメニューバーとツール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を使用できます。 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メニューバーとツールバーを追加する
- スクリーンセットを実行する
- メニューの構造を定義する
- ツールバーの構造を定義する
- ツールバーをカスタマイズする

このチュートリアルを始める前には、次の準備が必要です。

- 『カスタムコントロールのプログラミング』の章をすべて読んでください。

ユーザ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、次の資料も参照してください。

- ヘルプの『オブジェクトと属性』

  ユーザコントロールに関するトピックでは、ユーザ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

ツールバーのコントロールプログラムの使用例は、tbards.gs スクリーンセットに含まれて

います。また、DialogSystem\demo\tbards サブディレクトリの tbards.txt ファイルに説明が記述されています。

設定

1. この作業では、Customer スクリーンセットに多数の変更を加えるので、作業を始める前に customer.gs のバックアップを作成してください。

2. Net Express を起動します。

3. DialogSystem\demo\customer サブディレクトリの customer.app を開きます。

4. 左側のペインで customer.gs を右クリックし、コンテキストメニューの [編集] を選択します。

   Dialog System が起動し、「顧客情報」ウィンドウが表示されます。

スクリーンセットへのメニューバーとツールバーの追加

クラスライブラリのメニューとツールバーを使用する場合に最も重要な点は、各ツールバーボタンにメニュー項目が関連付けられている点です。 ツールバーボタンを選択すると、関連付けられたメニュー項目に対するメニューイベントがクラスライブラリによって生成されます。 同様に、メニュー項目の使用可能、使用不能、チェック、または、チェックの解除を実行すると、関連付けられたツールバーボタンに対するイベントが発生します。

コントロールプログラムを使用する前に、スクリーンセットのデータブロックで共通のデータ領域を定義します。 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受け渡すために使用されます。

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コントロールには、データブロック項目 (マスターフィールド) を関連付ける必要があります。 このデータ項目には、作成したコントロールのクラスライブラリオブジェクト参照を格納するので、OBJ-REF として定義します。 データ項目は、ユーザコントロールを追加する前に定義する必要があります。 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は、Dialog System でユーザ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格納するスクリーンセットのデータブロックに、次のデータ項目を指定します。

MYTOOLBAR　　OBJ-REF

『チュートリアル - ステータスバーの追加とカスタマイズ』の章の作業を完了している場合は、スクリーンセットのデータブロックに次の FUNCTION-DATA のデータ定義を指定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のみで十分です。

FUNCTION-DATA 定義を指定する必要がある場合は、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できます。 [ファイル]、[インポート]、[スクリーンセット] の順に選択して、[OK] をクリックし、現在ロードされているスクリーンセットの上書きを許可します。 [ファイル] ボタンをクリックして、funcdata.imp をダブルクリックします。[インポート] ボタン、[OK]、[閉じる] の順にクリックします。

次の定義をインポートします。

```
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
```

この定義は、前の説明のとおり、DialogSystem\source サブディレクトリの tbardata.imp ファイルから同様にインポートします。

MYTOOLBAR	作成したツールバーのオブジェクト参照を格納します。 このデータ項目は、次の手順で定義するツールバーユーザコントロールのマスタフィールドになります。
TBAR-PARMS	ツールバーの機能を使用した場合に、メニュー項目とツールバーボタンの情報をコントロールプログラムに渡すための汎用グループです。

クラスライブラリを有効にするには、データブロックで 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内にある SCREENSET-INITIALIZED イベントの先頭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 イベント内のどのダイアログよりも前に配置する必要があります。

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後でユーザ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. [プログラム済みコントロール] ツールバーの [ツールバー] を選択します。

3. メニューとツールバーを表示するウィンドウの最上部で、ツールバーの位置とサイズを調整します。

4. 「ツールバーの属性」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. ツールバーに名前を付けます。 ここでは、MAIN-WINDOW-TOOLBAR という名前を付けます。

    2. マスタフィールド名を指定します。 この名前は、前に定義したデータブロックの名前 MYTOOLBAR と一致する必要があります。

    3. ツールバ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。

    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] をクリックして、ツールバーコントロールプログラム tbar2.cbl を生成します。

   クラスライブラリのメニュー構造を作成するかどうかを確認するためのメッセージボックスが表示されます。 [はい] をクリックします。

   メッセージボックスが開き、生成されたコントロールプログラムで必要なデータブロック内の項目が確認されます。 必要な項目は、前の手順で確認済みなので、[OK] をクリックして、作業を続けます。

   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ます。

5. [OK] をクリックしてスクリーンセットを保存します。

6. ツールバーの [COPY ファイル生成] ボタンを選択し、customer.cpb ファイルの上書きを承認します。

7. Net Express の [プロジェクト] メニューの [すべてをリビルド] を使用して、生成されたコントロールプログラムをコンパイルし、このプログラムの実行可能バージョンを作成します。

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完了した後で、スクリーンセットを実行して、ユーザコントロールの動作を確認できます。

この段階では、メニューバーとツールバーが実質的に機能しません。

メニューバーとツールバーを正しく更新するためのダイアログを追加する必要があります。

TBAR2 は、カスタマイズしたバージョンのツールバーコントロールプログラムの名前です。 ここでは、TBAR2 がメニューのコントロールプログラムになります。

カスタマイズしたバージョンのツールバーコントロールを呼び出すには、スクリーンセットのグローバルダイアログの末尾に次のダイアログを追加します。

```
CALLOUT-TBAR
   CALLOUT "TBAR2" 0 $NULL
```

前の章の作業を完了していない場合は、オブジェクトのダイアログに WINDOW-CREATED イベントを追加する必要があります。前の章の作業を完了している場合は、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ます。

メニューの構造の定義

[編集] メニューの [メニューバー] を選択するか、または、ツールバーの [編集] メニューバーアイコンを選択します。 「メニューバーの定義」ダイアログボックスが表示されます。

DSGRUN ではなく、コントロールプログラムによって作成されたメニューを使用するので、[オプション] を選択して「クラスライブラリと使う」をチェックします。 この操作によって、メニューの作成に DSGRUN が使用されません。

新しいメニューオプションの追加

ここでは、既存の [注文] メニューオプションを挟むように、2 つのメニュー項目 [編集] と [オプション] を追加します。 [編集] メニューには、メニュー項目 [すべて選択] を 1 つのみ設定します。 [オプション] メニューには、メニュー項目 [カスタマイズ] を設定します。このメニュー項目を選択した場合に [フォント] サブメニューが開くように設定します。

1. [注文] をクリックし、[編集]、[前に挿入] の順に選択します。

2. 「メニューバーの選択の詳細」ダイアログボックスの「選択名」フィールドに「EDIT」と入力し、「選択テキスト」フィールドに「編集」と入力します。 他のフィールドは変更しないで、[OK] をクリックします。

   新しいメニュー項目 [編集] とサブメニュー項目 [編集] が追加されます。

3. [編集] サブメニュー項目をクリックし、[編集]、[変更] の順に選択します。

4. 「プルダウンの選択の詳細」ダイアログボックスの「選択テキスト」フィールドと「クラスライブラリヒントテキスト」フィールドに「すべて選択」を入力します。 ショートカットキープルダウンメニューの F6 キーを選択します。 [OK] をクリックします。

5. 手順 1 と同様に [注文] をクリックし、[編集]、[後に挿入] の順に選択します。

6. 「メニューバーの選択の詳細」ダイアログボックスの「選択名」フィールドに「OPTIONS」と入力し、「選択テキスト」フィールドに「オプション」と入力します。 [OK] をクリックします。

   新しいメニュー項目 [オプション] とサブメニュー項目 [オプション] が追加されます。

7. [オプション] サブメニュー項目をクリックし、[編集]、[変更] の順に選択します。

8. 「プルダウンの選択の詳細」ダイアログボックスの「選択テキスト」フィールドと「クラスライブラリヒントテキスト」フィールドに「カスタマイズ」を入力します。 [OK] をクリックします。

9. [カスタマイズ] をクリックし、[編集]、[サブプルダウン] の順に選択します。

   [カスタマイズ] にサブメニュー選択 [カスタマイズ] が表示されます。

10. このサブメニュー選択をクリックし、[編集]、[変更] の順に選択します。

11. 「プルダウンの選択の詳細」ダイアログボックスの「選択テキスト」フィールドと「クラスライブラリヒントテキスト」フィールドに「フォント」を入力します。 ショートカットキープルダウンメニューの F9 キーを選択します。 [OK] をクリックします。

12. 「メニューバーの定義」ダイアログボックスを閉じ、スクリーンセットを保存します。

13. ツールバーをダブルクリックして、「ユーザコントロールの属性」ダイアログボックスを開きます。 「コントロールタイプ」プルダウンから「ツールバー」を選択し、[生成] をクリックします。

    クラスライブラリのメニュー構造を作成するかどうかを確認するためのメッセージボックスが表示されます。 [はい] をクリックします。

    ツールバープログラム全体を上書きするかどうかを確認するためのメッセージボックスが表示されます。 [いいえ] をクリックすると、メニュー定義のみを作成し、ツールバーへの変更に影響しません。

    メッセージボックスが開き、生成されたコントロールプログラムで必要なデータブロック内の項目が確認されます。 必要な項目は、前の手順で確認済みなので、[OK] をクリックして、作業を続けます。

    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ます。

14. [キャンセル] をクリックします。

15. [プロジェクト] メニューの [すべてをリビルド] を選択します。

16. Net Express の [アニメート] メニューの [実行] メニュー項目を使用して、アプリケー

ションを実行します。

この段階では、追加したメニューオプションが表示されますが、これらのメニューオプションを選択しても機能しません。

ツールバーの構造の定義

tbar2.cbl で定義したデータ構造体を例に、ツールバーの作成方法を説明します。このデータ構造体を表示するには、tbar2.cbl から Net Express の [検索] メニュー、[定義検索] メニュー選択の順に選択し、データ項目「bData」を検索します。

既存のツールバーの構造

「bData」表構造では、ツールバーに追加するボタンを定義し、順序を設定します。 ボタンは、リスト内の順序どおりにツールバーに追加されます。

1 つのボタンレコードの例は、次のとおりです。

```
*>------------------------------------------------------
*  ボタンオブジェクト参照
   03 object reference.

*  ボタンに関連付けるメニュー項目指標、または、
*  ゼロ (ボタンが区切り記号の場合)。 この指標は、
*  mData 表内の指標である。
   03 pic x comp-5 value 2. *> [終了] メニュー項目

*  このボタンビットマップのリソース ID。
   03 pic x(4) comp-5 value IDB-EXIT.

*  マウスでボタンをポイントすると、ツールのヒントが表示される。
   03 pic x(bStringSize) value z"Quit this application".
*>------------------------------------------------------
```

表の各要素によってツールバーのボタンを定義します。各要素は、次の 4 つの部分で構成されます。

1. 作成したツールバーボタンのオブジェクト参照
2. ボタンに関連付けるメニュー項目の指標 (mData 表内) またはゼロ (区切り記号の場合)
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 ID
4. マウスでボタンをポイントしたときに表示されるツールのヒント

生成されたデフォルトのツールバー構造では、区切り記号、および、メニュー指標 2 に関

連付けられたボタンが指定されています。ただし、ここでは、Dialog System の既存のメニューからメニュー構造を作成したので、[ファイル] メニューの [終了] メニュー項目の指標を 7 にする必要があります。2 つ目のボタンを正しいメニュー指標に関連付けるには、2 を 7 に変更してください。

新しいツールバーボタンの追加

ここでは、[すべて選択] と [フォント] に対して 2 つのツールバーボタンを追加します。

tbar2.cbl で定義されたツールバーを変更するには、既存のボタンレコードを使用してボタン構造を変更します。

1. [終了] ボタンを表す bData 表構造の 2 つ目の要素を、そのすぐ下に複製します。 この複製したコードを編集します。

2. メニュー項目指標を 7 から 9 に変更します。また、必要に応じて、注釈を変更します。

3. ボタンビットマップのリソース ID を IDB-EXIT から IDB-BLUE に変更します。

4. ツールのヒントを「終了」から「すべて選択」に変更します。

5. 2 つ目のツールバーボタンに使用するコードの行を、そのすぐ下に複製します。

6. メニュー項目指標を 9 から 14 に変更します。また、必要に応じて、注釈を変更します。

7. ボタンビットマップのリソース ID を IDB-BLUE から IDB-RED に変更します。

8. ツールのヒントを「すべて選択」から「フォント」に変更します。

9. [プロジェクト] メニューの [すべてをリビルド] をクリックします。

10. スクリーンセットを実行します。

    ツールバーのメニュー項目の下に、作成した 2 つのボタンが表示されます。

    この場合も、ツールバーのボタンを選択しても機能しません。

ツールバーのカスタマイズ

前の説明では、tbresid.cpy ファイルから既存のビットマップを追加しました。 ここでは、新しいボタンビットマップを作成します。

リソースファイルの設定

プロジェクトに新しいリソースファイルを追加して、ツールバーに使用するビットマップリソースを作成します。

1. Net Express のコマンドプロンプトで、ビットマップ (*.bmp ファイル) と mfres.h ファイルを DialogSystem\demo\CommonControls\Toolbar サブディレクトリから DialogSystem\demo\customer サブディレクトリにコピーします。 tbar.rc をコピーして、名前を tbres.rc に変更します。

2. Net Express のプロジェクトウィンドウを右クリックして、[ソースプールへファイルを追加...] を選択します。

3. tbres.rc を選択します。

4. 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)] の順に選択します。次に、[作成] をクリックして、このファイルを左側のペインに追加します。

5. 左側のペインで tbres.rc を右クリックし、[コンパイル]、 [編集] の順に選択します。

6. ツリー構造の「ビットマップ」を右クリックして、コンテキストメニューの [新規作成] を選択します。

7. 「リソース識別子」フィールドに「IDB_BUTTON1」と入力します。

8. 「リソースファイル名」フィールドに「selectall.bmp」と入力します。

9. 「ID 値」フィールドに 220 と入力し、[OK] をクリックします。

10. イメージエディタのメニューで [ファイル]、[新規作成]、[ビットマップ] の順に選択し、[OK] をクリックします。

11. 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。

12. ツールバーのボタンが使用可能な場合に表示するビットマップを描画します。

13. 作成したビットマップを selectall.bmp として保存して、イメージエディタを閉じます。

14. もう 1 つのビットマップに、リソース識別子「IDB_BUTTON2」、リソースファイル名 custfont.bmp、および ID 値 221 を指定して、両方のビットマップリソース (および IDB_ 識別子) をリソースファイル .lp で定義できるようにします。

15. [ファイル] メニューの [名前を付けて保存] を選択し、このリソーススクリプトを tbres.rc として保存します。

16. tbres.rc を閉じます。

17. tbar2.cbl を右クリックし、コンテキストメニューの [編集] を選択します。

18. resourceDllName を tbres.dll に変更します。

19. 次の行の下に新しく作成したビットマップを追加します。

    copy "tbresid.cpy"

    ビットマップを追加するためのコードは、次のとおりです。

    ```
    78 IDB-Button1          value 220.
    78 IDB-Button2          value 221.
    ```

20. tbar2.cbl を保存し、閉じます。

21. [プロジェクト] メニューの [すべての従属関係の更新] を選択します。

22. プロジェクトをリビルドします。 tbar2.cbl で必要なリソース識別子定数を含む mfres.cpy が作成されます。 この段階では、リビルドについて表示されたエラーをすべて無視します。

コピーファイルの設定

コピーファイルを変更して、ツールバー構造を定義します。

メニュー項目、および、その項目に関連付けられたツールバーボタンを含む tbar2.cbl のツールバー構造を確認します。 このファイルには、ツールバーの表示状態とイベントへの応答を定義するデータ構造体が含まれます。

ツールバーの定義方法は、次のとおりです。

1. 左側のペインで tbar2.cbl を右クリックし、[編集] を選択します。

2. Net Express の [検索] メニュー、[定義検索] メニュー選択を順に選択し、データ項目「bData」を検索します。

3. 『ツールバーの構造の定義』で説明したとおり、bData グループ構造によってツールバーの各項目が定義されます。

リソースファイルに定義した IDB_ リソース識別子 (ここでは IDB-Button1 と IDB-Button2) への参照がボタン定義データレコードに格納されることを確認してください。

tbar2.cbl を保存し、閉じます。

スクリーンセットへのダイアログの追加

次のように、ダイアログをスクリーンセットに追加して、メニュー選択に応答します。

ツールバーを既存のアプリケーションに追加する場合 (ここの例など) は、次の手順を実行します。

1. 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
   ```

2. 実行した手続きをカスタマイズしたメニューダイアログテーブルに置き換えます。この例では、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
   ```

3. こ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) 7 @EXIT
     IF= NUMERIC-VALUE(1) 9 SELECT-ALL
     IF= NUMERIC-VALUE(1) 14 FONTS
   ```

4. ダイアログの末尾に次の行を追加します。

   ```
   SELECT-ALL
     BEEP
   FONTS
     BEEP
     BEEP
   ```

   クラスライブラリメニュー項目を追加して、既存のメニュー項目を置換する場合は、メニューイベントの処理に使用した手続き名 (先頭に @ が設定されています) に分岐します。 または、クラスライブラリのメニューイベントの後で、ツールバーのコールバックコードから副プログラムを COBOL の CALL 文で呼び出します。

新しいアプリケーションを作成している場合は、次の手順を実行します。

1. ツールバーの親ウィンドウに対する 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

   *  ダイアログ関数の処理中 - メッセージボックスが表示される
   ```

   クラスライブラリのメニューイベントを処理するためのダイアログ関数を使用するかわりに、ツールバーのコールバックコードで COBOL の CALL 文を使用して既存のプログラムを呼び出し、メニュー関数を実行できます。

スクリーンセットを保存し、Net Express IDE に戻って、プロジェクトをリビルドします。

[アニメート] メニューの [実行] を選択して、スクリーンセットをテストします。 [編集] メニューの [すべて選択] を選択するか、または、[すべて選択] ツールバーボタンをクリックすると、コード化したダイアログ手続きが実行されます。 [オプション] メニューの [カスタマイズ] から [フォント] についてもテストします。

チュートリアル-メニュー/ツールバーの追加とカスタマイズ
チュートリアル-マウスポインタのビットマップの変更

第 22 章 : チュートリアル-ActiveX コントロールの追加

ここでは、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。 このチュートリアルを始める前に、『カスタム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。

以後では、あらかじめ用意された時計の ActiveX コントロールを使用して、スクリーンセットを変更する手順について説明します。

スクリーンセットの変更

ここでは、ステータスバー上の時刻をダブルクリックしたときに、組み込みの時計の ActiveX コントロールが表示されるように設定します。 ActiveX コントロールの使用方法については後で説明します。ここでは、Net Express の customer.gs を開き、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 の時計を表示するダイアログボックス
- アラーム時刻を設定するためのダイアログボックス
- 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

1. ActiveX の時計を設定するために、Customer スクリーンセットの約 3 分の 1 のサイズのダイアログボックスを作成して、配置します。
2. このオブジェクトに CLOCK-DIALOG という名前を付けます。
3. ダイアログボックスに関するすべての属性をデフォルトに設定します。
4. 親を $WINDOW として定義します。
5. 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を作成します。
6. このオブジェクトに ALARM-GONE-OFF-MBOX という名前を付けます。
7. ダイアログボックスに関するすべての属性をデフォルトに設定します。

8. 親を $WINDOW として定義します。

9. このスクリーンセットに関するデータブロックに PIC 9(2) データ項目 (ALARM-HOURS と ALARM-MINUTES) を追加します。

10. アラーム時刻を設定するために、Customer スクリーンセットの約 3 分の 1 のサイズのダイアログボックスを作成して、配置します。

11. このオブジェクトに SET-ALARM-DIALOG という名前を付けます。

12. ダイアログボックスに関するすべての属性をデフォルトに設定します。

13. 親を $WINDOW として定義します。

14. CLOCK-DIALOG ダイアログボックスで次の操作を実行します。

    1. プッシュボタンを作成し、テキストを「アラーム設定」に設定します。

    2. [アラーム設定] 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
```

    3. プッシュボタンを作成し、テキストを「閉じる」に設定します。

    4. [閉じる] ボタンに次のダイアログを追加し、このボタンをクリックすると、CLOCK-DIALOG が表示されないように設定します。

```
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。

15. SET-ALARM-DIALOG で次の操作を実行します。

    1. アラーム時刻の「時間」を入力するエントリフィールドを作成します。

       これを ALARM-HOURS マスターフィールドとリンクします。

    2. アラーム時刻の「分」を入力するエントリフィールドを作成します。

       これを ALARM-MINUTES マスターフィールドとリンクします。

3. アラームが発生したときに表示するメッセージを入力するエントリフィールドを作成します。

   このエントリフィールドのマスターフィールドを IO-TEXT-BUFFER2 にします。

4. プッシュボタンを作成し、テキストを「OK」に設定します。 [OK] ボタンには、次のダイアログを指定します。

   ```
   BUTTON-SELECTED
      BRANCH-TO-PROCEDURE SET-ALARM
   ```

5. プッシュボタンを作成し、テキストを「キャンセル」に設定します。 ユーザが [キャンセル] ボタン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、[キャンセル] ボタンに次のダイアログを指定します。

   ```
   BUTTON-SELECTED
      DELETE-WINDOW SET-ALARM-DIALOG $NULL
   ```

6. SET-ALARM-DIALOG ダイアログボックス自体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
   * 後でここにダイアログを追加する。
   ```

これらの手順が済んだ後で 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。 『チュートリアル - ステータスバーの追加とカスタマイズ』の章にある『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』を参照してください。

left-mouse-button-double-click イベントコールバックには、コールバックが実行されたときにユーザイベント 34591 を送信するためのコードを追加しました。 また、追加したコードでは、double-click イベントが発生し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
```

このダイアログでは、MAIN-WINDOW ウィンドウがユーザイベント番号 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設定されています) に設定された場合に、CLOCK-DIALOG ダイアログボックスが表示されるようにします。

Dialog System の時計 ActiveX コントロールの使用方法

この機能を適用したデモンストレーションが clock.gs スクリーンセット内に組み込まれています。clockds.txt ファイルに、説明が記述されています。

チュートリアルに従うと、スクリーンセット内に CLOCK-DIALOG ダイアログボックスが作成されます。 これで、このダイアログボックスに Dialog System の時計 ActiveX コントロールを追加できます。

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
```

最後の 4 項目にはインデントを挿入して、これらの項目が集団項目であることを表す必要があります。

最初のデータ項目は、ActiveX のオブジェクト参照を格納するために使用されます。 指定し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。

次に 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. [ファイル]、[インポート]、[ActiveX コントロール...] の順に選択します。

2. Dialog System Clock を選択します。

   コントロールが ActiveX ツールバーに追加されます。このツールバーからこのコントロールを選択して Customer スクリーンセットのステータスバーに配置します。

   ActiveX コントロールを配置すると、「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 に設定して、プログラム名を CLOCKCTRL に設定します。

4. [生成] をクリックして、ActiveX コントロールプログラムを生成します。

   [OK] をクリックします。
5. 次に、ActiveX コントロールの属性を設定します。 各 ActiveX コントロールには、そのコントロールの実行時の動作を指定するための属性が設定されており、属性を変更できます。

   コントロールが描画されている場合は、「属性」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、コントロールに対して設定できる属性が表示されます。 属性の値は、このリストで変更できます。または、[コントロール] コンテキストメニューを使用して、ActiveX の属性ページのコンテキストメニューで選択した属性をすべて表示できます。

   Dialog System の時計 ActiveX に対して設定できる属性は、背景のビットマップと時計の表示方法 (アナログまたはデジタル) です。

ユーザイベントを処理するために、アラーム時刻になったことを表すダイアログを追加する必要があります。 そのためには、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。

```
 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追加した SET-ALARM 手続きを完成させる必要があります。 この手続きは、アラームを設定するために ActiveX コントロールの SetAlarm メソッドを呼び出します。 SET-ALARM の下に次のダイアログを追加します。

```
    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) 変数に対する関数 NULL-TERMINATE と INVOKE の使用方法に注意してください。 これらの関数の詳細については、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て実行し、変更結果を確認してください。

チュートリアル-ActiveX コントロールの追加 フォントと色の指定

第 23 章 : チュートリアル-マウスポインタのビットマップの変更

『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の選択方法について説明しました。 ここでは、次の内容について説明します。

- マウスポインタを変更する
- ビットマップに関数機能を提供する

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 関数を使用します。 たとえば、ユーザに対して操作が完了するまで待機するように表示するには、マウスポインタを時計 (環境によっては文字盤のある時計や砂時計) に変更します。

SET-MOUSE-SHAPE 関数の詳細については、ヘルプトピックの『ダイアログ文：関数』を参照してください。

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収録) では、マウスポインタが指すオブジェクトに応じてマウスポインタの形状が変化します。

- 「省略時の形状」オプションボタンを選択した場合は、3 種類のオブジェクト (入力フィールド、リスト、および、ビットマップ) にマウスポインタを移動してもマウスポインタの形状は変化しません。
- 「動的形状」オプションボタンを選択した場合は、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。

カーソルの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、各行を選択およびハイライト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、次の操作を実行します。

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. [ファイル] メニューの [実行] を選択して、スクリーンセット Animator を呼び出します。
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。

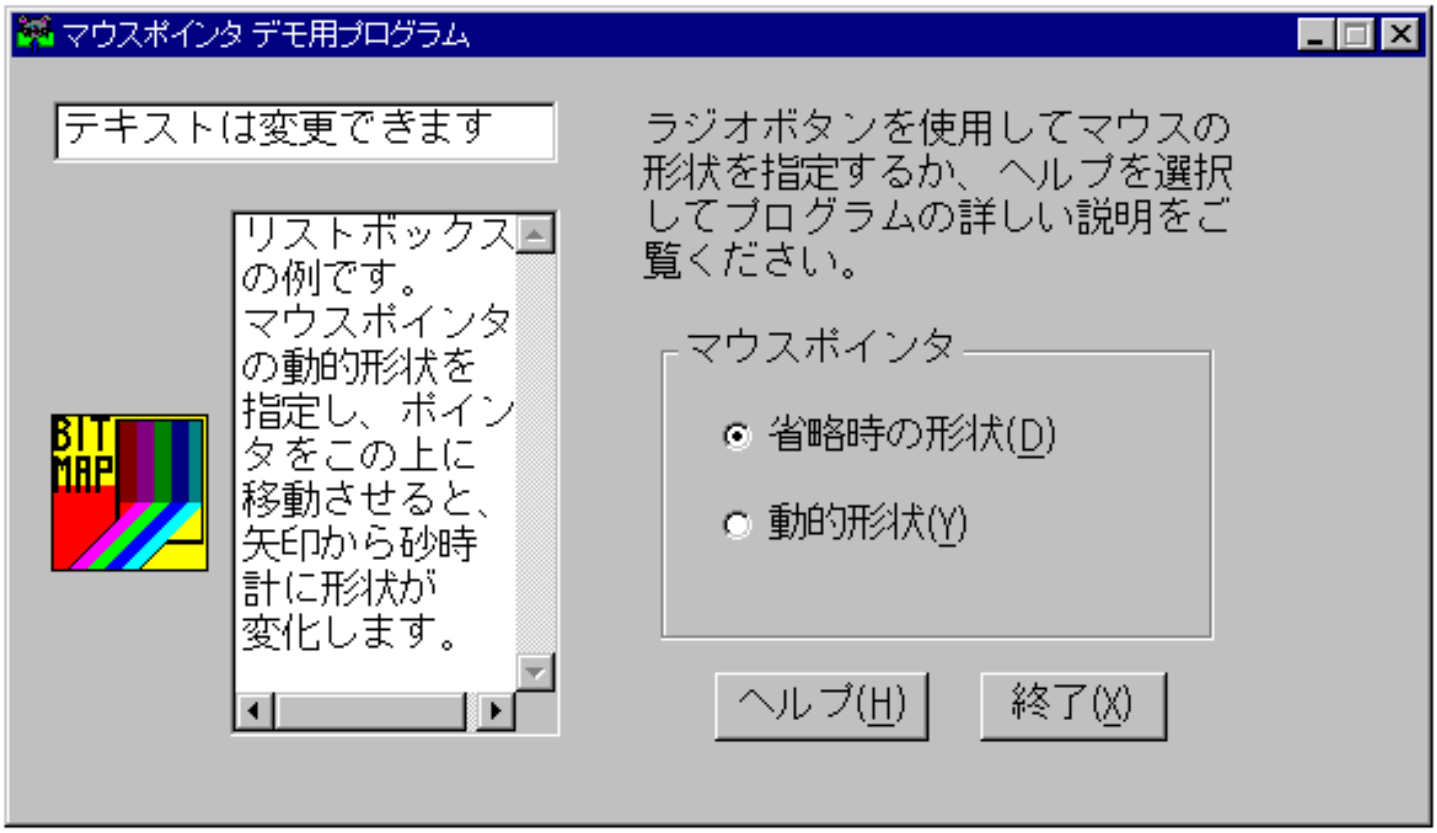

図 23-1： 「Moudemo」の画面

ユーザが Esc キーを押したり、メインウィンドウを閉じたりしたときに、グローバルダイアログによって呼び出し側プログラムに制御が戻ります。

```
ESC
   RETC
CLOSED-WINDOW
   RETC
```

スクリーンセットを初期化すると、「省略時の形状」オプション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の形状が 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は、「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 の場合は、[OK] が選択されており、ユーザ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設定します。「省略時の形状」オプションボタン (スクリーンセットの起動時のデフォルト) を選択した場合は、メインウィンドウ全体に SYS-Arrow の形状が使用されます。

```
BUTTON-SELECTED
   SET-MOUSE-SHAPE MOUSEDEMO-WIN "SYS-Arrow"
```

「動的形状」オプションボタンが選択されている場合は、マウスポインタが指すオブジェクトに応じてマウスポインタの形状が変化します。 SYS-Move は、オペレーティングシステムで用意されています。pencil-ptr は、このスクリーンセット用に特別に用意されています。

```
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、ダイアログを設定できません。

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 サンプル用のリソースサイドファイル (moudemo.icn) がロードされます。 そのため、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。

ここでは、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。 そのため、適切なセクションヘッダーの下にある ds.icn ファイルに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 ファイルに配置する必要があります。 具体的な手順については、ヘルプトピックの『ビットマップ、アイコン、マウス』を参照してください。

また、ビットマップの名前と位置を宣言する新しいビットマップのエントリを追加する必要があります。 ビットマップファイルの位置は、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 と呼びます。 ds.icn の該当するセクションヘッダーの下には、次の項目を記述します。

colorful : dssamw32.dll 003

ビットマップのプログラミング

Dialog System では、ユーザがビットマップをコントロールとして選択できます。 その意味では、ビットマップはオプションボタンと同様にみなすことができます。 ビットマップを選択した後の動作は、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、グラフィック画像なのでわかりやすく表現できます。いくつかのビットマップが設定されたウィンドウを図 23-2 に示します。

図 23-2： ビットマップを設定したウィンドウ

ビットマップについてわかりやすく説明するために、次の例では、ユーザが実行できる関数を示す 5 つのビットマップを使用します。 ユーザがビットマップを選択した場合に、呼び出し側プログラムに戻り、ビットマップで表される関数を実行します。

これらのビットマップコントロールには、WBBMP、ANIMBMP、HELPBMP、SCREENBMP、および EDITBMP という名前が付いています。各ビットマップには、内部識別子であるハンドルがあります。 オブジェクトに割り当てるハンドルは、データブロックで定義する必要があります。

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する必要があります。 最もよい方法は、グローバルダイアログで SCREENSET-INITIALIZED 関数を使用する方法です。

```
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 関数では、ビットマップ (ANIMBMP) のハンドルが数字フィールド (ANIM-HANDLE) に転記されます。

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、イベントが発生し、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設定します。

```
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
```

ユーザがビットマップをクリックすると、次のように処理されます。

- ビットマップイベントが発生し、ビットマップのハンドルが $EVENT-DATA に格納されます。
- $EVENT-DATA が、格納済みの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戻ります。

このプログラムでは、FUNCTION の値に基づいて適切な関数が実行されます。

Dialog System のインターフェイスへのビットマップの追加については、ヘルプトピックの『ビットマップ、アイコン、マウスポインタ』を参照してください。

チュートリアル-マウスポインタのビットマップの変更

付録 A 章 : フォントと色の指定

ここでは、異なるフォントと色の使用方法を説明します。 次のような目的でインターフェイスの表示状態を変更できます。

- 組織の標準に準拠するため。 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。
- ユーザの好みに合わせる。
- 特定の構成要素にユーザの注意を引きつける。 たとえば、次のように変更できます。
    - デフォルト値を別のフォントでハイライトする。
    - 妥当性検査エラーが発生したフィールドの背景色を変更する。
    - 太字を使用して [削除] ボタンを強調する。

異なるフォントと色を使用すると、インターフェイスを大幅に見やすくできます。 ただし、使用するフォントの種類が多すぎると、画面が見にくくなります。そのため、1 画面に使用する色を 3 色までにすることをお勧めします。

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、インターフェイスを見やすくできます。 たとえば、モノスペースフォントを使用すると、インターフェイスの各部分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用して、インターフェイスの特定の部分にユーザの注意を引き付けることができます。 たとえば、テキスト文字列内のパラメータを周囲の文字とは異なるフォントにすることによってハイライトできます。 次のテキストは、この機能を示しています。

ダイアログ内の large-entry-field は MLE に関連付けられたデータ項目です。

インターフェイスの特定の構成要素をハイライトするもう 1 つの方法としては、テキストのサイズ (ポイントサイズ) を変更します。 前の例の場合は、ファイル名を表す文字を大きなポイントサイズ (たとえば 10 ポイントのかわりに 12 ポイントを使用) にできます。 この方法もユーザの注意をファイル名に引き付けるのに役立ちます。

フォントは、定義時のみに設定できます。 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。 ただし、Panels V2 を使用してフォントを動的に変更できます。 Dialog System プログラムから Panels V2 を呼び出す方法については、『Panels V2 の使用方法』の章を参照してください。

フォントの設定方法については、ヘルプトピックの『Dialog System の概要』を参照してください。

色の設定

オブジェクトの前景色と背景色は、定義時に変更できます。[編集] メニューの [色の設定] を使用して変更する方法、[色の編集] ツールバーアイコンをクリックして変更する方法、または、SET-COLOR 関数を使用して動的に変更する方法があります。

たとえば、次のダイアログでは、妥当性検査エラーが発生したフィールドの色を「RED (背景色)」と「WHITE (前景色)」で表示するように変更します。 Dialog System で妥当性検査エラーが検出された場合は、エラーが発生したフィールドの識別子が $EVENT-DATA に書き込まれます。

```
VAL-ERROR
   SET-COLOR $EVENT-DATA 'WHITE' 'RED'
   SET-FOCUS $EVENT-DATA
```

汎用色はすべての環境で利用できます。 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で実行するアプリケーションを作成する場合は、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ください。

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 関数の詳細と利用可能な色の一覧については、ヘルプトピックの『ダイアログ文：関数』を参照してください。
