Sun WorkShop Visual ユーザーズ ガイド

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303
U.S.A. 650-960-1300

Part No. 806-4848-01
2000 年 6 月 Revision A

原典： *Sun WorkShop Visual User's Guide*
Part No: 806-3574-10
Revision A

製品名の変更について

Sun は新しい開発製品戦略の一環として、Sun の開発ツール群の製品名を Sun WorkShop™ から Forte™ Developer に変更いたしました。製品自体の内容に変更はなく、従来通りの高品質をお届けいたします。

これまでの Sun の主力製品である基本プログラミングツールに、Forte Fusion™ や Forte™ for Java™ といった Forte 開発ツールの得意とする、マルチプラットフォームおよびビジネスアプリケーション実装の機能を盛り込むことで、より広範囲できめ細かな製品ラインが完成されました。

WorkShop 5.0 で使用されていた名称と、Forte Developer 6 で使用される新しい名称の対応については、以下の表をご覧ください。

旧名称	新名称
Sun Visual WorkShop™ C++	Forte™ C++ Enterprise Edition 6
Sun Visual WorkShop™ C++ Personal Edition	Forte™ C++ Personal Edition 6
Sun Performance WorkShop™ Fortran	Forte™ for High Performance Computing 6
Sun Performance WorkShop™ Fortran Personal Edition	Forte™ Fortran Desktop Edition 6
Sun WorkShop Professional™ C	Forte™ C 6
Sun WorkShop™ University Edition	Forte™ Developer University Edition 6

製品名の変更に加えて、次の 2 つの製品について大きな変更があります。

- Forte for High Performance Computing には Sun Performance WorkShop Fortran に含まれていたすべてのツール、および C++ コンパイラが含まれます。したがって、High Performance Computing のユーザーは開発用に 1 つの製品だけを購入すれば済むことになります。
- Forte Fortran Desktop Edition は以前の Sun Performance WorkShop Personal Edition と同じです。ただし、この製品に含まれる Fortran コンパイラでは、自動並列化されたコード、および明示的な指令に基づいた並列コードは生成できません。この機能は Forte for High Performance Computing に含まれる Fortran コンパイラでは使用できます。

Sun の開発製品を引き続きご利用いただきましてありがとうございます。今後もみなさまのご要望にお応えする製品をお届けできるよう努力してまいります。

目次

図目次

表目次

第1章

概要

Sun™ WorkShop™ Visual の紹介

Sun WorkShop Visual は、標準 OSF/Motif ツールキットのウィジェットを組み合わせて使用することによって、グラフィカルユーザーインタフェース (GUI) を構築する対話型ツールです。Sun WorkShop Visual を使用すると、アイコンをクリックすることにより、画面上で素早く簡単にウィジェット階層を作成することができます。作成した結果 (デザイン) は、ウィジェット階層を表わすツリー構造とダイナミックディスプレイ (デザイン中のインタフェースの模擬実装。最終的にできる動作と外観の確認に使用します) の 2 通りの方法で画面に表示されます。対話型編集機能を使用すると、階層のブラウズ、個々のウィジェットのカット&ペースト、および属性の設定を行うことができます。ユーザーがウィジェット階層を編集するに従ってダイナミックディスプレイが変更されるため、編集操作による効果を即座に確認することができます。

デザインの作成が完了すると、Sun WorkShop Visual はインタフェースに必要なコードファイルを生成します。Sun WorkShop Visual によって生成されたコードは、インタフェースのプロトタイプとして、修正を行うことなくコンパイル、リンク、実行することができます。インタフェースのプロトタイプをアプリケーションコードへ接続するには、連結コードを作成します。Sun WorkShop Visual には、連結コードのテンプレートとして使用できるファイルが用意されています。

Sun WorkShop Visual で作成したインタフェースをアプリケーションに接続するには、コールバック関数を特定のウィジェットに関連付けます。たとえば、ユーザーがインタフェース上の特定のプッシュボタンをクリックした際に、特定の関数が呼び出されるように指定することができます。コールバック関数により、アプリケーションはインタフェースからのユーザーイベントを受け取って処理することができるようになります。

Sun WorkShop Visual には、Sun WorkShop Visual 再現機能、Sun WorkShop Visual 捕獲機能、AppGuru デザイナーというツールが用意されています。これらのツールを使用すると、アプリケーションのデザインをより効率的かつ効果的に行うことができます。AppGuru デザイナーで、作成するアプリケーションの基本構成要素をあらかじめ決定し、Sun WorkShop Visual 再現機能および Sun WorkShop Visual 捕獲機能で、作成した Motif アプリケーションの内容更新と動作確認を行うことができます。Sun WorkShop Visual 再現機能および AppGuru デザイナーの詳細は、481 ページの第 13 章「デザインツール」 を、Sun WorkShop Visual 捕獲機能の詳細は、499 ページの第 14 章「Sun WorkShop Visual 再現機能」を参照してください。

Sun WorkShop Visual では、スマートコードコールバック機能を使用して、ユーザーがデザインした thin クライアント / サーバーアプリケーションを構築できます。さらに、クライアントアプリケーションでインターネットにアクセスできるようにすることも可能です。詳細は以下の章を参照してください。

- 549 ページの第 15 章「グループ」では、スマートコードのブロックを構築する方法を説明します。
- 559 ページの第 16 章「取得と設定用のスマートコード」では、スマートコードの基本原則を説明します。
- 577 ページの第 17 章「thin クライアント用スマートコード」 では、チュートリアルを使用しながら、ユーザーがデザインした thin クライアントとサーバーを作成する方法を説明します。
- 615 ページの第 18 章「インターネット用スマートコード」 では、インターネットにアクセスするために必要なコードと構造体を Sun WorkShop Visual が自動生成する仕組みを説明します。

Sun WorkShop Visual を拡張して、他の X ツールキットのウィジェットを使用することができます。719 ページの第 23 章「ユーザー定義ウィジェット」では、Sun WorkShop Visual を拡張して別のウィジェットを組み込む方法を説明します。

Sun WorkShop Visual は通常、標準 OSF/Motif および X ツールキットとともに使用できるコードを生成します。また、Microsoft Windows 上で同等のインタフェースを作成するコードを生成することもできます。どちらのプラットフォームでもアプリ

ケーションの大部分が同じになるように、コードを構成することができます。この技術は、Sun WorkShop Visual の C++ コード生成機能を活用したものです。詳細は、411 ページの第 11 章「Microsoft Windows 用のデザイン」、および 449 ページの第 12 章「Microsoft Windows と Motif の アプリケーション作成」で説明します。

Sun WorkShop Visual では、別のクロスプラットフォーム方式を使用して、デザインから Java を生成することもできます。詳細は、363 ページの第 10 章「Java 用のデザイン」 を参照してください。

オペレーティング環境

Sun WorkShop Visual 5.0 を使用するには、次のいずれかの構成が必要です。

- Solaris 2.5.1 および Solaris 2.5.1 リリースの Motif
- Solaris 2.6 および Solaris 2.6 リリースの Motif
- Solaris 7 および Solaris 7 リリースの Motif

注 - Solaris 2.5.1、Solaris 2.6、Solaris 7 は、開発者システムサポート、全体ディストリビューションのいずれかのソフトウェアグループのものをインストールしてください。

生成されたコードのコンパイルに必要な条件

Sun WorkShop Visual で生成したコードをコンパイルするには、次のものが必要です。

- X11R5 または X11R6 ヘッダーおよびライブラリ
- Solaris 2.5.1、Solaris 2.6、Solaris 7 のいずれかに付属の Solaris Motif または Solaris CDE パッケージ

Microsoft Windowsモードのときに生成した MFC コードをコンパイルするには、次のものが必要です。

- Microsoft Windows 3.1、3.11、95、98、または Microsoft Windows NT 3.5.1、4.0
- Visual C++

- MFC バージョン 3.0、4.0 または 5.0

基本概念および用語

次に、Sun WorkShop Visual の基本概念および本マニュアルで使用する主要な用語を紹介します。

ウィジェット

ウィジェットは、ユーザーインタフェースを作成するために使用する構築ブロックです。表示されたインタフェース上で特定の外観や動作を持つものもあります。Motif の例としては、プッシュボタン、ラベル、テキストフィールドウィジェットなどがあります。他のウィジェットを収納し組織化するための、コンテナウィジェットと呼ばれるウィジェットもあります。コンテナウィジェットは、ダイナミックディスプレイ上では目には見えません。例としては、フォーム、ブリテンボード、メニューバー、ローカラムウィジェットなどがあります。

すべての Motif ウィジェット、および Sun WorkShop Visual の構成で使用するその他のウィジェットは、Sun WorkShop Visual メインウィンドウ上の左側にあるウィジェットパレット内にアイコンで表示されます。これらのアイコンの 1 つをクリックすると、その型のウィジェットがデザインに追加されます。デザイン内の個々のウィジェットは、ウィジェットクラスのインスタンスと呼ばれます。たとえば、プッシュボタンアイコンを 3 回クリックした場合には、デザインにウィジェットクラス「プッシュボタン」の 3 個のインスタンスが追加されたことになります。

デザイン階層

デザイン内のウィジェットは、デザイン階層で組織化されています。デザイン階層は、最上部にルートを持ち、枝を下方に伸ばしていくツリーとして Sun WorkShop Visual 上に表示されます。デザイン階層は、図 1-1 に示されるように、Sun WorkShop Visual メインウィンドウの大きな描画領域に表示されます。この領域は、構成領域と呼ばれます 。

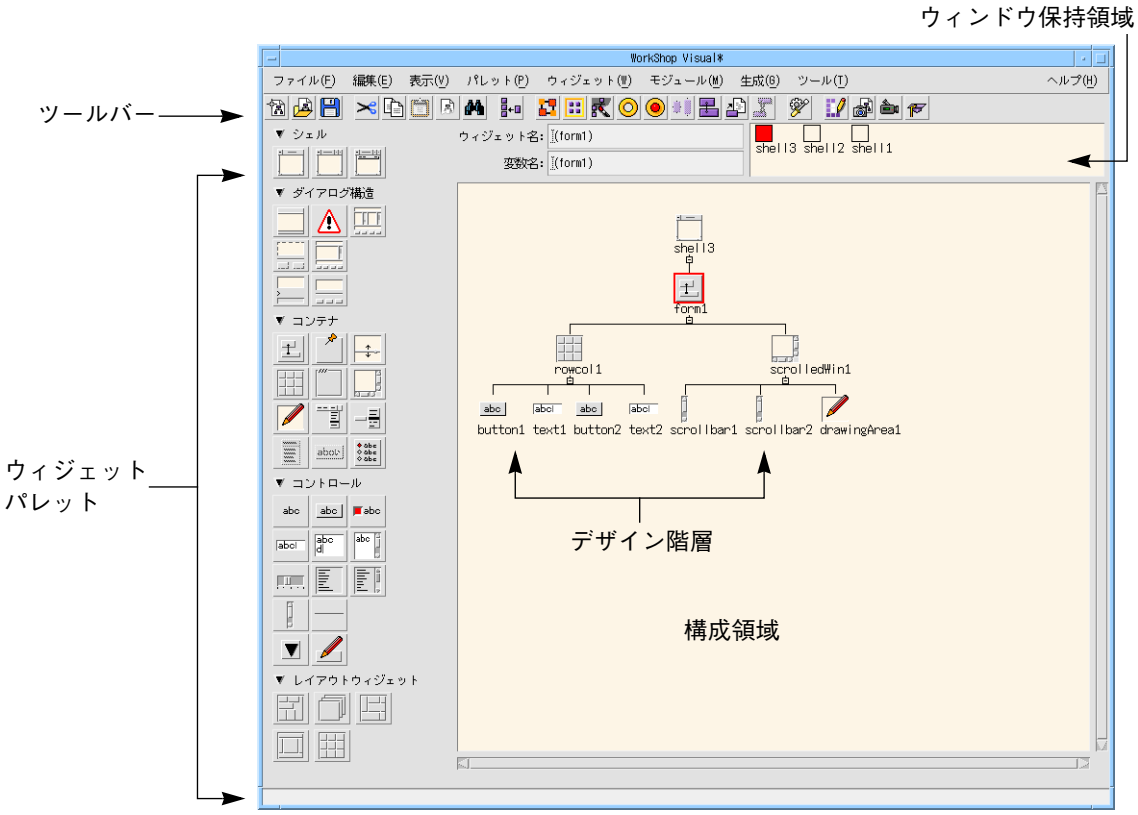


図 1-1　Sun WorkShop Visual メインウィンドウ

階層に追加されたウィジェットは、そのすぐ上にある親ウィジェットの子になります。親ウィジェットは子の外観や動作に影響する場合があるため、この関係は重要です。たとえば、ローカラムウィジェットは、その子となるウィジェットのサイズと位置が自動的に変更されるような厳密な配置を、子に対して強制的に行うことができます。

親ウィジェットは、図 1-2 に示すように、画面上では、子のすぐ上に表示されます。

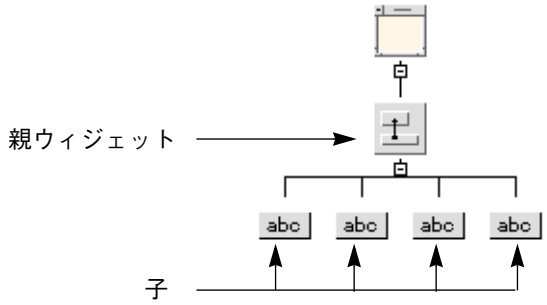


図 1-2　デザイン階層

リソース

リソースは、ウィジェットの外観または動作に影響する、ウィジェットの属性です。リソースの例には、寸法、ラベルまたはテキストフィールドの内容、そして色などが含まれます。Sun WorkShop Visual がウィジェットのインスタンスを作成する場合、各リソースに対して有効なデフォルトが割り当てられます。対話型のリソースパネルを使用すると、ユーザー独自のリソース値を割り当てることができます。

リソースおよびその有効値は Sun WorkShop Visual によってではなく、ウィジェット作成者によって定義されるため、本マニュアルで詳しく説明することはできません。しかし、Sun WorkShop Visual を理解しやすくするため、本書では一般的に使用される Motif リソースを説明しています。ある程度経験を積んだユーザーは、Motif のマニュアルを参照して Motif ウィジェットおよび各ウィジェットに指定可能なリソース設定の詳細情報を得ると良いでしょう。その他のツールキットからウィジェットを使用する場合のガイドラインは、そのウィジェット開発元のマニュアルを参照してください。

あるリソースのグループ (アタッチメント) は、フォームコンテナウィジェット内にあるウィジェットの位置を制御します。Sun WorkShop Visual には、アタッチメントの設定を対話的に行うための配置エディタが用意されています。

ガジェット

いくつかのウィジェットのクラスには、ウィジェットによく似たガジェットと呼ばれるものが存在します。ガジェットはウィジェットに似ていますが、そのリソースの設定には制約があります。ガジェットには、マシンリソース消費が少ないという利点があります。

無効なアクションに対しての保護

Sun WorkShop Visual には、Motif で無効なウィジェット階層またはリソースをユーザーが指定しないように、多くの機能が用意されています。操作しようとした時点では実行不可能なコマンド、適用できないリソース、指定した親ウィジェットの子としては無効なウィジェットのアイコンは、画面上では、図 1-3 に示されるようにグレー表示されます。グレー表示されたコマンド、リソース、アイコンは、選択されても動作しません。

図 1-3　有効および無効なカスケードボタンアイコン

Sun WorkShop Visual は、無効なリソース設定を受け付けません。リソースパネル上のエントリは、入力された値が有効範囲外である場合、あるいはウィジェットの親によって制御または制限されているリソースを変更しようとした場合に拒否されることがあります。この内容については、第 3 章「リソース」で説明します。リソース設定に対しての Motif の規則は複雑であるため、無効な設定によって深刻な結果がもたらされる場合があります。しかし、Sun WorkShop Visual の保護機能により、ユーザーからのリソース設定は確実に有効なものとされます。

オンラインヘルプ

Sun WorkShop Visual では、任意の場所からオンラインヘルプを使用することができます。一般的なヘルプの場合には、メインウィンドウのヘルプメニューをプルダウンし、「ヘルプ」オプションを選択します。ヘルプメッセージは、「Sun WorkShop Visual ヘルプ」または「Netscape」の 2 つのビューアのいずれかで表示することができます。ビューアを変更するには、「ヘルプ」メニューの「ビューア」プルライトメニューから使用したいビューアを選択します。デフォルトは、図 1-4 に示す「Sun WorkShop Visual ヘルプ」です。

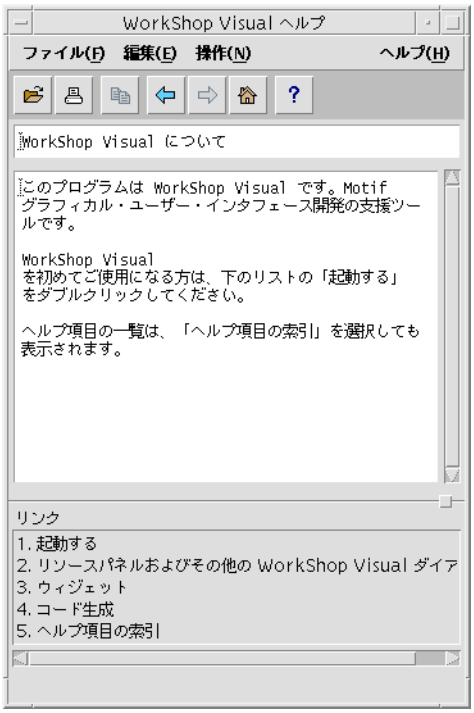


図 1-4　ヘルプビューア

Sun WorkShop Visual ヘルプにはメニューバーとツールバーがあります。通常のファイル操作には「ファイル」メニューを、テキスト編集機能には「編集」メニューを、ファイル間を行き来するには「操作」メニューを使用します。ツールバーボタンを使用すると、メニュー項目にあるほとんどの機能に短縮操作でアクセスすることができます。ツールバー上でマウスのポインタを動かすと、ウィンドウの最下部にあるステータス行に、該当するボタンの機能の説明が表示されます。ヘルプビューアウィンドウの最下部には、関連項目のリストが表示されます。項目の 1 つをダブルクリックすると、その項目のヘルプが表示されます。

ほとんどのダイアログボックスとリソースパネルには「ヘルプ」ボタンがあり、そのボタンをクリックすると、該当するダイアログボックスに関するヘルプを表示することができます。

Sun WorkShop Visual に付属しているヘルプビューアを自作のアプリケーションに組み込んで、ユーザー向けのヘルプとして使用することもできます。組み込み方法については、697 ページの「Sun WorkShop Visual でのヘルプの設定」を参照してください。

Netscape によるヘルプの使い方

Sun WorkShop Visual では、ハイパーテキストヘルプの表示に Netscape ブラウザを使用することもできます。

Netscape を使用するには、ヘルプメニューをプルダウンして「ビューア」プルライトメニューを選択し、次に「Netscape を使用」を選択します。

Netscape では、ファイル間を行き来するための操作キーと操作コマンドを使用することができます。また、テキスト内で強調表示されている語句をクリックして、リンク先にアクセスすることもできます。

ウィジェットパレットのヘルプ

メインウィンドウのヘルプメニューの「ウィジェットパレット」を選択すると、各 Motif ウィジェットのアイコンと名前が表示されたウィジェットパレットがポップアップします。ウィジェットパレット上のアイコンの 1 つをクリックすると、選択されたウィジェットが属するウィジェットクラスについての記述が表示されます。

Sun WorkShop Visual の開発サイクル

Sun WorkShop Visual アプリケーションの作成過程には、以下の 4 段階が含まれます。

1. インタフェースの設計

   この段階には、次の操作が含まれます。

   - ウィジェット階層の構築
   - リソースの設定
   - 配置エディタを使用した配置の調整
   - 個々のウィジェットと関連付けられるコールバックの指定

2. コードの生成

   Sun WorkShop Visual は、インタフェースの表示および操作に必要なすべての C または C++ コードを自動的に生成します。また、Sun WorkShop Visual は、インタフェース・コードをアプリケーション・コードに接続するために必要なすべての #include 文および関数宣言を含むスタブファイルを生成します。

3. コードの作成

   インタフェース・プロトタイプを実際のアプリケーションに接続するため、スタブファイル内にある空の関数括弧の間に必要なコードを供給します。

4. リンク、実行、テスト

   この段階は、ソフトウェアプログラムの開発には必須であるデバッグサイクルの後に行われます。

本マニュアルの構成

本マニュアルは、大きく分けて次の 3 つのセクションから構成されます。

- 学習セクション
- 上級セクション
- リファレンスセクション

学習セクション

Sun WorkShop Visual は対話性の高いツールです。そのため、さまざまな機能の記述を読むよりも、簡単な配置を作成するための手順を実際に実行していくことで、それらの機能を容易に理解することができます。Sun WorkShop Visual を初めて使用するユーザーには、ワークステーションを起動して学習の章を読み、示されている手順を実際に実行して簡単なインタフェースを構築することをお勧めします。学習セクションは 15 ページから始まり、デザイン階層の構築、リソースの設定、配置の調整、コールバックの設定、コードの生成、非常に簡単なコールバック関数の作成といった、開発サイクルの全段階について詳細に説明しています。学習セクションを終了する頃には、ユーザーの手元には動作可能な (初歩的な) インタフェースができ上がっており、それまでにユーザーは Sun WorkShop Visual の主要機能をすべて使用することができます。

X ウィンドウシステムおよび Motif の知識は、Sun WorkShop Visual を学ぶすべての段階で重要ですが、X または Motif の初心者であっても、X および Motif の資料で学習しながら Sun WorkShop Visual の最初の数章を読むことで知識を得ることができます。付録 E「参考資料」では、X および Motif に関する推奨書籍のリストが示されています。コード生成段階では、X、Motif、Sun WorkShop Visual が使用するプログラム言語の 1 つ (C、C++ または UIL) の知識がある程度必要です。

上級セクション

学習セクションは、241 ページの第 7 章「コードの生成」で終わります。その後に続く上級セクションでは、Sun WorkShop Visual を最も効果的に使用するための上級テクニックを説明します。このセクションは、Motif および X に精通していて Sun WorkShop Visual をすでに使用した経験がある、あるいは学習セクションを終了したユーザーを対象にしています。このセクションでは、高度なコード生成機能、メークファイル生成、ウィジェットパレットおよびツールバーの生成、ユーザー定義ウィジェットの統合、高度なレイアウトおよび国際化について説明します。ここには、MFC (Microsoft Foundation Class) や Java を使用した構造化コード生成およびクロスプラットフォーム開発機能を説明するための短い学習用の例も含まれています。いくつかの章では、Sun WorkShop Visual 再現機能、Sun WorkShop Visual 捕獲機能、および AppGuru などのデザインツールについて説明しています。また、Sun WorkShop Visual 再現機能の拡張についても説明しています。さらに、thin クライアント用およびインターネット用の Sun WorkShop Visual スマートコード機能についても説明しています。

リファレンスセクション

リファレンスセクションは、Sun WorkShop Visual の全レベルのユーザーを対象としており、以下の項目を含んでいます。

- すべての Sun WorkShop Visual コマンドの要約
- Motif ウィジェットおよび使用可能なリソース、そしてそれらが Microsoft Windows に割り当てられる方法の要約
- 障害の対処方法のヒント
- Sun WorkShop Visual の動作および外観を変更するために設定することが可能なリソースの記述
- Sun WorkShop Visual 再現スクリプトで使用するキーワード
- Motif と Windows との間でコードを共有可能にする Motif ライブラリについての説明
- 用語集
- 参考資料

本マニュアルで使用する規約

1. キーボード・キーの名前およびマウスボタンは山括弧で囲みます。

   例:<Tab>

   同時に 2 個のキーを押す必要がある場合は、次の形式で示します。[1]

   <最初のキー - 2 番目のキー>

   例:<Ctrl-C>、<Meta-H>、<Shift-Button1>、<Shift-button1> など

2. キーボードで入力するテキストは、次の形式で示します。

   `Type this exactly`

   引用符 (“ ”) は使用しません。引用符が存在する場合には、それらもテキストと一緒に入力します。

3. メニューコマンドにはキーボード・アクセラレータ、つまり、マウスを使用せずにコマンドを実行するために使用するキーが表示されているものもあります。手順の指示では、キーボード・アクセラレータは、メニューコマンドの名前の後に括弧で囲んで示してあります。

- 「ウィジェット」メニューをプルダウンし、「リセット」(<Ctrl-T>) を選択します。

  この例では、メニューから「リセット」を選択するか、あるいは <Ctrl-T> を押すように指示しています。両方の実行を意味したものではありません。

4. 関数名は `XtAppMainLoop()` のように、名前の後にある空の括弧によって識別されます。山括弧 < > は、名前の変数部分を示します。
   デフォルトのウィジェット名は、<ウィジェットクラス><*n*> の形式で示します。ここで <ウィジェットクラス> には、ボタンやラベルのような Motif のクラスが、<*n*> には整数が入ることを意味します。

5. 「クリック」は、特に指定のない限り、常にマウスボタン 1 を使用して行うことを意味します。マウスを構成し直した場合を除き、ボタン 1 は通常は左側のボタンを、ボタン 2 は中央のボタンを、そしてボタン 3 は右側のボタンを示します。

   「2 回クリックする」と「ダブルクリックする」は異なる意味を持ちます。「ダブルクリック」では、マウスボタンを素早く 2 回続けて押す必要があります。「2 回クリック」を行う場合には、どのような速さでクリックを行なっても構いません。

1. Sun 製品のキーボード上では、Meta は、◇の刻印のあるキーのことです。ただし、OpenWindows 以外のウィンドウシステムをご使用の場合、Alt キーが Meta の働きをすることもあります。Ctrl は、Control キーを表わします。また、Button/button はマウスボタンを意味します。

6. 特別な意味を持たない場合と区別する場合は、Motif ウィジェットのクラス名は、「ラベル」、「プッシュボタン」のように「」で囲みます。

   例: このデザインのメインウィンドウには、「メインウィンドウ」ウィジェットではなく、「フォーム」ウィジェットを使用します。

7. 本マニュアルで示されている書籍は、1011 ページの付録 E「参考資料」に一覧されています。

第2章

ウィジェット階層の構築

学習セクションの紹介

本マニュアルの学習セクションは、以下の章から構成されています。

- 第 2 章「ウィジェット階層の構築」
- 第 3 章「リソース」
- 第 4 章「配置エディタ」
- 第 5 章「その他のエディタ」
- 第 6 章「インタフェースの起動 : ユーザーコードの追加」
- 第 7 章「コードの生成」

学習セクションを読み進めながら、ワークステーション上で操作手順を段階的に実行していくと、簡単なインタフェースを構築することができるようになっています。また、インタフェース構築の過程では、Sun WorkShop Visual の主要な機能をすべて紹介します。

手順をすべて実行すると、図 2-1 に示すような学習インタフェースができあがります。

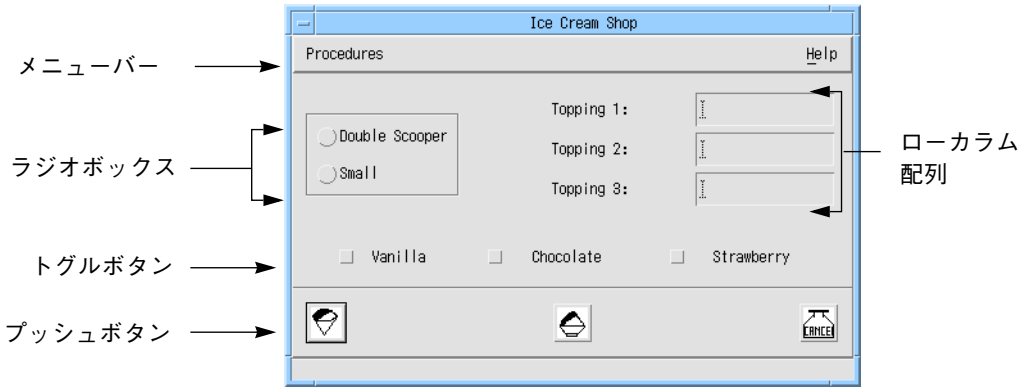


図 2-1　学習インタフェース

クロスプラットフォームに関連する情報を含む一般的な技術を説明する個所を除き、この学習例の大部分は Motif 専用の形式での開発を説明しています。構造化コード生成とクロスプラットフォーム開発の学習例については、それぞれ 321 ページの第 9 章「C++ コードの学習」と 449 ページの第 12 章「Microsoft Windows と Motif の アプリケーション作成」を参照してください。

デザイン階層

インタフェース構築の第 1 段階は、構築に使用するウィジェットを決定し、Sun WorkShop Visual インタフェース上で適切なデザイン階層を開発することです。本章ではこの操作手順を説明していきます。その過程では、以下に示す動作方法を学習します。

- プログラムを起動し、新しいデザインの作成を開始する
- 図 2-1 に示されている一般的な 5 種類のウィジェットを組み合わせて、デザイン階層を構築する
  - メニューバー
  - ラジオボックス
  - ローカラム配列
  - トグルボタン・グループ
  - プッシュボタン・グループ
- ウィジェットへ名前を割り当てる

- Sun WorkShop Visual ファイルを保存し、取り出す
- カット、ペースト、コピー、ドラッグの各機能を使用して、デザイン階層構造を編集する

起動と停止

学習を開始する前に、システムに Sun WorkShop Visual を正しくインストールします。Sun WorkShop Visual がインストールされていない場合、あるいは下記のコマンドを入力しても Sun WorkShop Visual のメインウィンドウが表示されない場合には、インストール手順を参照するか、システム管理者にお問い合わせください。

以下のコマンドのどちらかを使用して、X から Sun WorkShop Visual を起動します。どちらのコマンドを使用した場合にも、Sun WorkShop Visual メインウィンドウが表示されます。

- **`visu`**

  VGA またはその他の小画面ディスプレイの場合は、次のように入力します。

- **`small_visu`**

注 - `small_visu` を使用して Sun WorkShop Visual を起動した場合は、本マニュアルで示されているアイコンとは多少異なる、小さめのアイコンが表示されます。

コマンド行オプション

Sun WorkShop Visual には、多数のコマンド行オプションがあります。これらのオプションは、787 ページの第 24 章「コマンド行の操作」に記述されています。また、`-x` オプションを使用して Sun WorkShop Visual を起動すると、コマンド行オプションの一覧を見ることができます。

以前に保存したデザインでの作業を再開する場合は、`visu` <ファイル名> のようにコマンド行でファイル名を指定することができます。現在のファイル名は、上部のウィンドウ枠に表示されます。変更が保存されていない場合は、ファイル名の後にアスタリスク (*) が表示されます。

クロスプラットフォーム開発機能を使用するために Sun WorkShop Visual を Microsoft Windows モードで起動する場合は、コマンド行オプション `-windows` を使用します。

Sun WorkShop Visual の「ファイル」メニューには、作業の保存、プログラムの終了、後でデザインに戻って作業を行うためのコマンドが用意されています。学習中のどの時点でもこれらのコマンドを使用することができるように、まず、これらのコマンドを紹介します。

保存、別名保存

「ファイル」メニューから「別名保存」(*<Ctrl-A>*) を選択すると、デザインを保存することができます。「別名保存」は、本章で後ほど説明するファイルブラウザを表示します。ファイルブラウザを使用すると、デザインの名前を指定することができます。

すでにファイル名を指定してある場合は、「保存」(*<Ctrl-S>*) を選択することができます。この場合は、ファイル名の指定を要求されないため、「別名保存」よりも短い時間で保存を行うことができます。規約により、Sun WorkShop Visual デザインファイルの名前には、接尾辞として `.xd` が付けられます。

Sun WorkShop Visual の上部のウィンドウ枠には、現在のファイル名が表示されています。変更が保存されていない場合は、ファイル名の後にアスタリスク (*) が表示されます。

「別名保存」ダイアログで、Java 用ビジュアルアプリケーションエディタの Visaj™ に取り込むのに適したファイルを選択するオプションを指定できます。詳細は、402 ページの「Sun WorkShop Visual のデザインを Visaj に移行」 を参照してください。

開く、新規、終了

保存されているデザインファイルを Sun WorkShop Visual に読み込むためには、「開く」(*<Ctrl-O>*) コマンドを使用します。「開く」は、本章で後ほど説明するファイルブラウザを表示します。ファイルブラウザを使用すると、既存のデザインファイルを選択することができます。「新規」(*<Ctrl-N>*) は、構成領域の内容を消去し、新たにデザインを開始します。「終了」(*<Ctrl-E>*) はプログラムを終了します。
これらの 3 つのコマンドは、すべてデザインに行なった変更を破棄します。デザインへの変更が保存されていない場合、Sun WorkShop Visual は、これらのコマンドを実行する前に変更を保存するかどうかをユーザーに尋ねます。

メニューの操作

Sun WorkShop Visual のメニューからは、次の 3 通りの方法でコマンドを選択することができます。

- マウスでクリックする
- キーボードアクセラレータを使用する
- キーボードニーモニックを使用する

アクセラレータ

キーボードアクセラレータは、メニューコマンドを実行するキー入力です。
たとえば、*<Ctrl-S>* を押すと、「保存」コマンドを実行することができます。

アクセラレータは、Sun WorkShop Visual のメインウィンドウのいずれかの領域に入力フォーカスがある場合に動作します。学習セクションでコマンドの実行を指示する場合には、アクセラレータは括弧内に示されます。また、画面上のメニュー内ではコマンド名の右側に表示されます。

ニーモニック

メニュー名およびオプションで下線の付いた文字がニーモニックです。ニーモニックは、マウスを使用せずにメニュー間を行き来するための手段です。ニーモニック文字を使用してメニューをプルダウンするためには、Meta キーを押しながらニーモニック文字を押します。たとえば、*<Meta-F>* を押すと、「ファイル」メニューをプルダウンすることができます。メニューをプルダウンした後は、*<Meta>* を使用せずにニーモニックを押して、任意の項目を選択します。たとえば、ニーモニックを使用して「保存」を呼び出す場合は、*<Meta-F><S>* の順でキーを押します。

第 3 章「リソース」では、Sun WorkShop Visual で作成するメニューバー上へのアクセラレータおよびニーモニックの設定方法を説明します。

ツールバー

Sun WorkShop Visual インタフェースには、ツールバーがあります。ツールバーには、ユーザーが独自に構成した、メニュー項目に対応するボタンを持たせることができます。図 2-2 にデフォルトのツールバー配置を示します。

図 2-2　デフォルトのツールバー配置

ツールバーのボタンを選択すると、対応するメニューボタンとまったく同じ動作が実行されます。

Microsoft Windows モードに特有の要素

Sun WorkShop Visual が Microsoft Windows モードである場合は、デフォルトのツールバー構成には、「様式」オプションメニューと、「Microsoft Windows 準拠」トグルボタンという 2 個の要素が余分に含まれています。これらは次のように表示されます。

図 2-3　Windows 特有のツールバー項目

「様式」オプションメニューは、Motif や Motif XP など、生成する対象コードを示すため、ツールバーのコード生成ボタンを使用する場合に便利です。

「Microsoft Windows 準拠」トグルは、現在のデザインが Microsoft Windows 準拠であるかどうかを示します。つまり、Sun WorkShop Visual が、デザインに対して有効

な Microsoft Windows コードを生成することができるかどうかを示します。また、トグルボタンを使用して、Microsoft Windows に準拠しているかどうかを検査するプロセスを呼び出すこともできます。これらの機能の詳細は、418 ページの「Microsoft Windows 準拠」を参照してください。

ステータス行

ステータス行は、Sun WorkShop Visual ウィンドウの最下部に表示されます。マウスボタンがメニュー項目、ツールバーのボタン、ウィジェットパレットのボタンのうちいずれかの上に移動すると、各メニュー項目またはボタンの機能を説明する文 (プロンプトヘルプ) が表示されます。これは、ピックスマップ・エディタおよび配置エディタ内にあるツールバーなど、すべてのツールバーについても同様です。使用可能なボタンに対してのみ、情報が表示されます。

デザインの開始

ダイアログのデザインには、まずシェルウィジェットを使用します。図 2-4 に示すシェルアイコンは、ウィジェットパレットの左上隅にあります。シェルウィジェットは 3 種類あります。リソースの 1 つを変更することによって、いずれかのシェルウィジェットにすることができます。

1. アプリケーションシェル
   アプリケーションのメインウィンドウです。アプリケーションを実行したときに最初に表示されます。
2. 最上位シェル
   アプリケーションシェルがアイコン化されている場合でも表示されるウィンドウです。このウィンドウだけをアイコン化することもできます。
3. ダイアログシェル
   アプリケーションシェルと別々にアイコン化することはできません。一般的に、アプリケーションのサブダイアログとして使用します。

シェルの種類についての詳細は、82 ページの「シェルの型」 を参照してください。

新しいデザインを作成する際、最初はシェル以外のアイコンを使用できません。

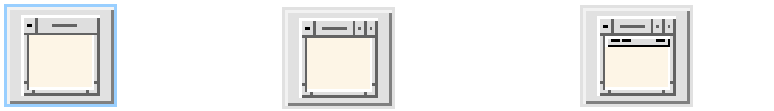


図 2-4　シェルアイコン

- アプリケーションシェルアイコンをクリックします。

シェルアイコンのコピーが構成領域に表示されます。

注 - 使用方法を習得するために、最初はこの方法でアプリケーションのデザインを開始してください。その他の手順として、Sun WorkShop Visual ユーティリティの AppGuru を使用することもできます。詳細は、481 ページの第 13 章「デザインツール」を参照してください。

ウィジェット名

ウィジェットのインスタンスが階層に追加される場合、そのインスタンスには Sun WorkShop Visual によって 2 個の名前 (ウィジェット名と変数名) が割り当てられます。変数名は、コード内でウィジェットを参照する際の名前です。ウィジェット名は、ツールキットがリソースを割り当てるために使用する名前です。デフォルトでは、これらの 2 個の名前は同じで、識別しやすいように <ウィジェット>*n* の形式になっています。<ウィジェット> には、button、form、shell のようなウィジェットのクラス名が、*n* には整数が割り当てられます。この整数は、同じウィジェットクラスのインスタンスが追加されるたびに、button1、button2、または、 form1、form2 のように、1 ずつ増加します。ただし、Sun WorkShop Visual の機能のいくつかは明示的な変数名を要求するため、重要なウィジェットを追加する場合は、明示的な変数名を割り当てるようにしてください。

ウィジェットの命名

シェルウィジェットに名前を付けるには、次のようにします。

1. 画面の上部にある「変数名」のボックスをダブルクリックします。

   ボックスをダブルクリックすると、その中のすべてのテキストが強調表示されます。新しいテキストを入力して Return キーを押すと、強調表示されているテキストと置き換えられます。

2. 次のように入力します。

   ```
   myFirstShell
   ```

   ユーザーが別のウィジェットを作成したり選択するときに、ウィジェットに自動的に名前が割り当てられます。また、Return キーを押して、変数名ボックスに名前を付けることもできます。

注 - 変数名は、Sun WorkShop Visual がコードを生成する際に、ウィジェット構造体でのウィジェットのインスタンスの参照に使用されるため、固有のものでなければなりません。Sun WorkShop Visual は、デザインの他の場所で使用されている変数名の入力を受け付けません。コンパイル時に問題が発生することを避けるため、ウィジェット変数名にはアプリケーション関数または変数の名前、Motif や X の定義あるいは関数名、C または C++ 予約語を使用しないでください。

変数名を変更すると、「ウィジェット名」ボックスにウィジェット名を明示的に指定した場合を除いて、ウィジェット名にも自動的に同じ名前が割り当てられます。ウィジェット名は固有の名前である必要はありません。同じウィジェット名を持つウィジェットは、リソース設定を共有するように構成することができます。共通のウィジェット名によってウィジェットをグループ化すると、1 回の操作でリソースのリセットを行うことができるため便利です。

ウィジェットをグループ化するには、次のようにします。

1. グループ化するウィジェットをすべて選択します。
2. 「ウィジェット名」フィールドに名前を入力します。

   複数のウィジェットを選択する方法については、27 ページの「複数選択」を参照してください。共有リソースの詳細は、277 ページの「共有リソース値」を参照してください。

階層への子の追加

シェルウィジェットは、どのような種類の子も持つことができます。ただし、その数は 1 つだけです。したがって、その他のウィジェットを含むことができるウィジェットを選択する必要があります。一般に、メインウィンドウ、ブリテンボード、フォーム、ダイアログテンプレートが子として使用されます。学習用インタフェースのデザインには、ダイアログテンプレートが適しています。

- ダイアログテンプレートアイコンをクリックします。

図 2-5　ダイアログテンプレートウィジェット・アイコン

アイコンの種類がわからない場合は、Sun WorkShop Visual の「パレット」メニューで名前とアイコンの両方をオンにするか、ヘルプメニュー (*<Meta-H><P>*) からウィジェットパレット・ダイアログを表示します。また、ウィジェットパレット上のアイコンにマウスポインタをあわせると、そのウィジェットの名前が Sun WorkShop Visual メインウィンドウ下部のステータス行に表示されます。

830 ページの「パレットメニュー」および 831 ページの「ウィジェットメニュー」を参照してください。

ダイアログテンプレート

ダイアログテンプレートは、メニューバー、任意の数のボタン、作業領域という 3 種類の子を任意で持つことができます。ダイアログテンプレートは、子メニューバーをウィンドウの上部に配置し、任意の型のボタンであるその他の子すべてをウィンドウの下部に、等間隔で 1 行に配置します。このボタンの行をボタンボックスと呼びます。

ダイアログテンプレートは、メニューバーとボタンボックスの間に作業領域を配置します。ボタンボックスとの間は、セパレータ (水平線) を使用して区切られます。セパレータはダイアログテンプレートの一部として作成され、ウィジェット階層に自動的に表示されます。

図 2-6　階層内のダイアログテンプレート

図 2-1 に示される学習用インタフェースの場合、メニューバー、カスケードボタンおよびボタンボックスを形成するプッシュボタンにはラベルが付けられています。作業領域にはラジオボックスという縦に配列されたトグルボタンが含まれています。

- **ダイアログテンプレートに変数名 `myDiag` を割り当てます。**

  デザインにウィジェットを追加していく場合は、ウィジェットにも変数名を割り当てていくと良いでしょう。明示的な名前を使用すると、ウィジェットの識別が容易になります。また、特定の操作には変数名が必要です。ただし、なんらかの方法でウィジェットを参照しない限り、Sun WorkShop Visual は名前を厳密に要求することはありません。したがって、ここでは学習セクションで使用する名前に対する手順のみを紹介します。

ダイナミックディスプレイの配置

ダイアログテンプレートウィジェットを追加すると、構成領域に、配置が小さな矩形として表示されます。これがダイナミックディスプレイ・ウィンドウです。
Sun WorkShop Visual はこの中にインタフェースの実装例を構築していきます。

ダイナミックディスプレイには、ウィジェットインスタンスの集合が表示されます。

Sun WorkShop Visual はウィジェットの絵の描画は行いませんが、インタフェースが実行時に使用するものと同じ Motif 関数呼び出しを使用して、実際にウィジェットを作成します。この時点では、ダイナミックディスプレイ・ウィンドウにはシェルとダイアログテンプレートが含まれているだけであり、認識できるようなその他の機能はほとんどありません。

ウィジェットを追加して移動を行なっていくに従い、ウィジェットは、完成したインタフェースで表示される場合と同様に、このウィンドウ内に表示されます。階層の表示の妨げにならない場所へダイナミックディスプレイ・ウィンドウを移動するためには、通常のウィンドウマネージャ機能を使用してください。

ウィジェットを階層に追加する場合に、意図したとおりにウィジェットが表示されないことがあります。この場合は、後で配置エディタを使用して正しく表示させることができます。

ダイナミックディスプレイに表示される配置は、デザインを実装した場合に構築されるインタフェースのプロトタイプです。したがって、ボタンのクリック、メニューのプルダウン、テキストフィールドへのテキストの入力などを行うことができます。

現在選択されているウィジェット

Sun WorkShop Visual では、ウィジェットを 1 つだけ撰択、複数を撰択、またはまったく撰択しない状態にすることが可能です。1 つだけ撰択されている場合は、そのウィジェットを現在選択されているウィジェットと呼びます。リソースの設定、カット & ペースト、あるいは子の生成など、どのような操作を行う場合でも、その前にウィジェットを選択しなければいけません。選択されたウィジェットは、構成領域とダイナミックディスプレイにおいて強調表示されます。

選択されたウィジェットの子になることができないウィジェットは、ウィジェットパレット上でグレー表示され、選択することはできません。

通常、ウィジェットが階層に追加されると、そのウィジェットは選択された状態になっています。このため、 ダイアログテンプレートを追加すると、そのダイアログテンプレートが自動的に選択されます。この状態でウィジェットを追加すると、追加されたウィジェットはダイアログテンプレートの子になります。しかし、ウィジェットが子を持つことができない場合には、そのウィジェットは選択できません。別のウィジェットを選択する場合は、構成領域内でそのウィジェットのアイコンをクリックします。

複数選択

ウィジェットを複数選択するには、以下の作業を行います。

- ウィジェットを囲むように矩形をドラッグする
- Shift キーを押したままマウスでウィジェットを選択する
  これで、現在選択しているウィジェットに加えて別のウィジェットを選択することができます。

別のダイアログ内のウィジェットを選択に加えるには、ウィンドウ保持領域で Shift キーを押したままシェルを選択します。このようにすると、現在の選択内容を保持したまま、別のダイアログ内のウィジェットを撰択することができます。

選択するウィジェットが複数の場合は、以下の作業を行うことができます。

- リソース (コアリソースとウィジェットリソースの両方) を設定する
  設定できる項目は、選択したウィジェットすべてに共通するリソースだけです。詳細は、66 ページの「複数選択とリソース」を参照してください。
- 選択したウィジェットを消去する
  これらのウィジェットはデザインから削除されます。

ウィジェットを選択しない場合

構成領域の空白部分をクリックするか、Shift キーを押したまま選択した各ウィジェットをクリックすると、どのウィジェットも選択していない状態にできます。この場合、デザイン全体に影響するような機能のみ実行することができます。

ボタンの追加

配置の下部にあるボタンは、図 2-7 に示すように、 ダイアログテンプレートの子として直接追加することができます。

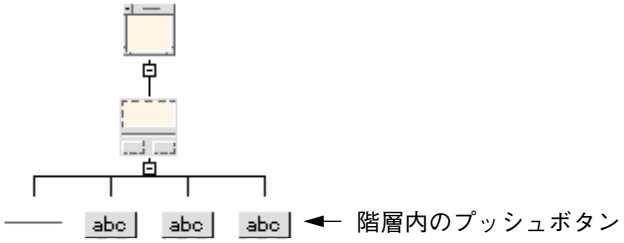


図 2-7　ボタンの階層

- **構成領域においてダイアログテンプレート・ウィジェットを選択し、ウィジェットパレット上のプッシュボタンアイコンを 3 回クリックします。**

  クリックを行うごとに、プッシュボタンがダイアログテンプレートの子として追加されます。プッシュボタンは、ダイナミックディスプレイにも「*button<n>*」というデフォルトラベルで表示されます。59 ページの第 3 章「リソース」では、これらのラベルに適切なテキスト文字列を割り当てることになります。

  ダイナミックディスプレイは、図 2-8 のようになっています。

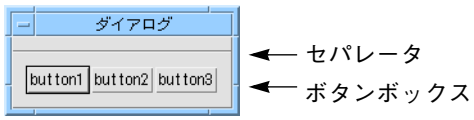


図 2-8　ボタンのダイナミックディスプレイ

ウィンドウのタイトル「ダイアログ」は、Sun WorkShop Visual によって設定されるデフォルトのラベルです。ラベルを変更する方法については、84 ページの「メインシェルのリソース設定」を参照してください。

メニューバーの構築

画面上部にあるメニューバーは、多くのコンピュータ・インタフェースに共通する機能です。Motif では、メニューバー・ウィジェットを提供しています。このウィジェットは、メニューを形成するための一連の他のウィジェットが追加されるまでは目に見えませ

ん。Sun WorkShop Visual は、関連するウィジェットを除くすべてのウィジェットをグレー表示することにより、メニューバー構築の過程が正しく行われるようにします。図 2-9 に追加する必要のある階層を示します。

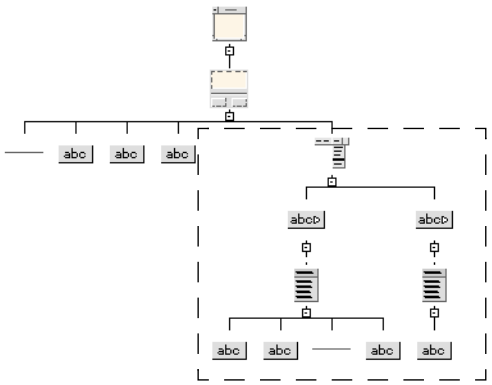


図 2-9　メニューバーとその子の階層

メニューバーの作成

メニューバーを構築するには次のようにします。

1. ダイアログテンプレートウィジェットを選択し、ウィジェットパレット上のメニューバーアイコンをクリックします。

   ダイアログテンプレートは、ダイナミックディスプレイ内で自動的に作業領域の上にメニューバーを配置します。

2. 変数名 **main_menu** をメニューバーに割り当てます。

3. カスケードボタンアイコンを 2 回クリックします。

   カスケードボタンを追加すると、図 2-10 に示すように、ダイナミックディスプレイにはデフォルトラベル「cascade」を持つカスケードボタンが表示されます。

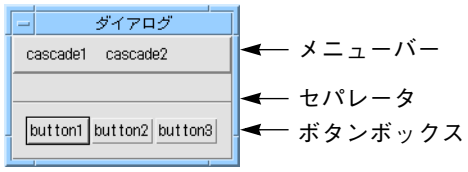


図 2-10 カスケードボタンのダイナミックディスプレイ

これらのボタンは、マウスでクリックして、動作を確認することができます。ただし、この時点ではボタンはメニューを含んでいないため、何も行われません。

4. 構成領域で最初に追加したカスケードボタンを選択し、そのボタンに変数名 **procedure_cascade** を割り当てます。

   このように変数名をロングネームで割り当てた場合、2 番目のカスケードボタンはダイナミックディスプレイのメニューバーには表示されません。両方のカスケードボタンが表示されるように、適切な名前の割り当てを学習セクションで実行します。カスケードボタンを両方とも現時点で表示するには、ダイナミックディスプレイの大きさを変更します。

メニューの追加

メニューを追加するには、次の動作を行います。

1. 階層で左のカスケードボタンを選択し、ウィジェットパレット上のメニューアイコンをクリックします。
2. プッシュボタンアイコンを 2 回クリックします。
3. セパレータアイコンをクリックし、その後プッシュボタンアイコンを再度クリックします。
4. 最後に追加したプッシュボタンをクリックし、変数名 **exit_button** を割り当てます。

   ダイナミックディスプレイ内の左のカスケードボタンには、この時点で、動作可能なメニューが含まれます。メニューは、カスケードボタン上にカーソルを置いてマウスボタン 1 を押し続けると、図 2-11 のように表示されます。

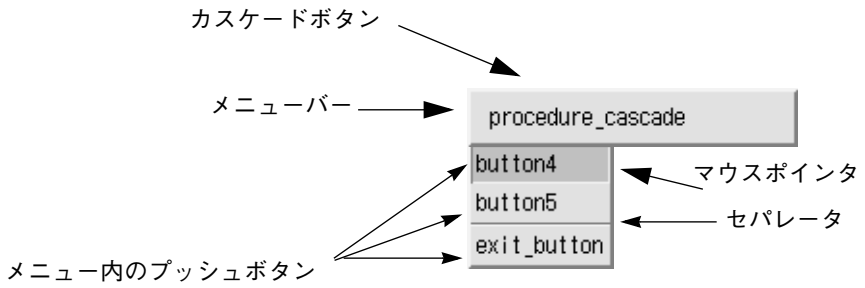


図 2-11 ダイナミックディスプレイ内のメニュー

メニュー内のプッシュボタンには、デフォルトラベルが付いていますが、これらは後で変更することができます。メニューには、プッシュボタン、トグルボタン、カスケードボタン (サブメニュー作成に使用) という 3 種類の子を持たせることができます。これらの項目は、マウスによって選択することができます。また、これらの子には、ラベルやセパレータを持たせることが可能です。ここで使用するメニューは、セパレータを表示してボタンを 2 つの領域に分割しています。

デザイン階層のメニューバー部分を完成させるには次のようにします。

5. 構成領域で 2 番目のカスケードボタンを選択し、変数名 **help_cascade** を割り当てます。
6. メニューアイコンをクリックします。
7. プッシュボタンアイコンをクリックします。
8. 構成領域でプッシュボタンをクリックし、変数名 **help_button** を割り当てます。

この時点で、2 番目のカスケードボタンには 1 個のオプションを持つ動作可能なメニューができました。このメニューは、ダイナミックディスプレイ内でマウスを使用してプルダウンすることができます。

作業領域の追加

現段階で、インタフェースには上部にメニューバーが、下部にいくつかのボタンがありますが、作業領域がないため、その追加を行う必要があります。ダイアログテンプレートは、子としての作業領域を 1 つしか持つことができません。ただし、子には複

数の子を持つコンテナウィジェットを指定することができます。ここでは、アイスクリームの味とトッピングを選べるようにするため、作業領域に複数のウィジェットを取り入れます。したがって、作業領域にはフォームを使用することにします。

1. 階層内でダイアログテンプレートを選択します。
2. フォームアイコンをクリックします。

   フォームは、子が与えられるまでインタフェース上には表示されません。インタフェースのオプションは、以下の 3 グループに分けて配置します。

   - 「Double Scooper」および「Small」トグルを持つラジオボックス
   - トッピングオプションのためのローカラム配列
   - アイスクリームの味を示す 3 個のトグルボタン

ラジオボックスの構築

ラジオボックスは、フォームと同様に目に見えないウィジェットであり、子の動作を制御するためだけに存在します。ラジオボックスは、ラジオボタンとして構成するトグルボタンのグループを収納します。1 回にユーザーが選択できるラジオボタンは 1 個だけです。図 2-12 に追加する必要のある階層を示します。

ラジオボックスを構築するには、次のようにします。

1. 階層においてフォームを選択します。
2. フレームウィジェットアイコンをクリックします。

   図 2-1 で「Double Scooper」および「Small」ラジオボタンを囲んでいる黒い線は、ラジオボックスそのものではなく、ラジオボックスを含んでいるフレームウィジェットです。フレームは、ラジオボックス構成要素の論理的グループを表示するために使用されます。

3. ラジオボックスアイコンをクリックします。
4. トグルボタンアイコンを 2 回クリックします。

   図 2-12 に、結果として生じるラジオボックスの階層を示します。

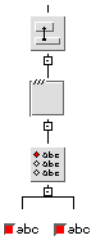


図 2-12　フレームが付いたラジオボックスの階層

ダイナミックディスプレイは、図 2-13 のようになります。

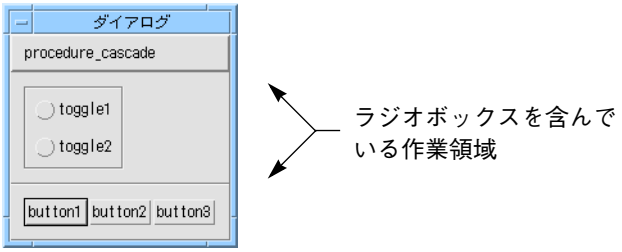


図 2-13　現段階でのダイナミックディスプレイ

ローカラム配列の構築

「ローカラム」コンテナウィジェットは、図 2-1 でトッピングを指定するラベルおよびテキストフィールドの配列に使用されます。

1. 階層でフォームを選択します。
2. ローカラムアイコンをクリックします。
3. 以下の順序でアイコンをクリックします: ラベル、テキストフィールド、ラベル、テキストフィールド、ラベル、テキストフィールド

   ローカラムは、行または列を構成する際に、必ず順番通りに子を取り入れるため、ラベルおよびテキストフィールドウィジェットは、この順序通りに追加されなければなりません。この場合は、行を作成し、その後で各行にラベルとテキストフィールドを持たせます。

図 2-14 に階層のローカラム部分を示します。

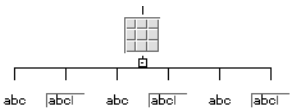


図 2-14 部分的な階層: ローカラムウィジェットとその子

図 2-15 に結果として表示されるダイナミックディスプレイを示します。

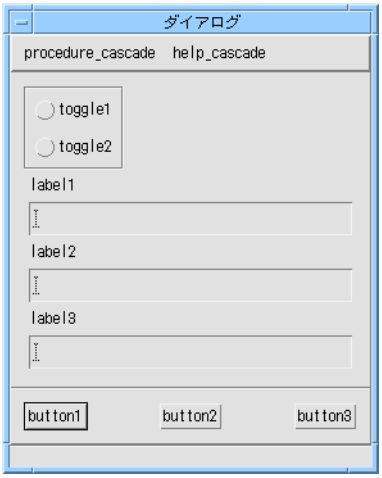


図 2-15 現段階でのダイナミックディスプレイ

デフォルトにより、ローカラムウィジェットは単一の垂直列に配置されるため、図 2-1 に示される配置とは異なっています。学習セクションの後半では、リソースを変更して意図したとおりの効果を実現します。

トグルボタンの追加

配置に必要な最後のオプショングループは、作業領域の下部に示すアイスクリームの味を表わすトグルボタンの集合です。

1. 階層でフォームを選択します。
2. トグルボタンアイコンを 3 回クリックします。

これで学習用インタフェースのデザイン階層は完成しました。完成した階層は、図 2-16 のようになります。

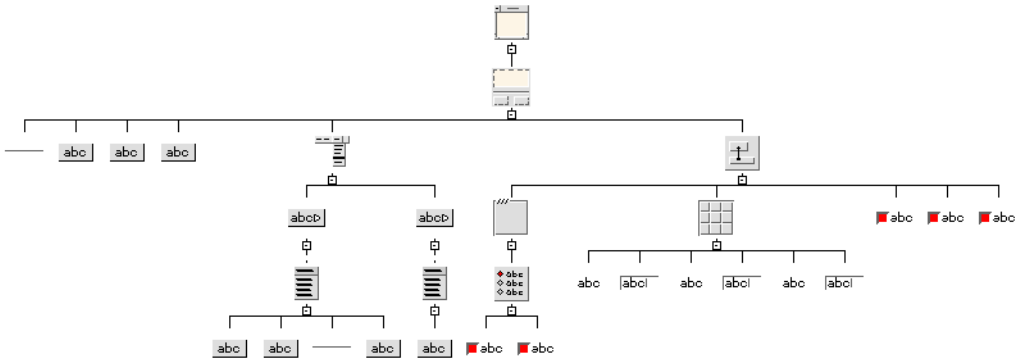


図 2-16 完成した階層

図 2-17 に示されるインタフェースは、まだ完成したものではありませんが、適切な親子関係で必要とされるすべてのウィジェットが含まれています。後の章で、リソースパネルおよび配置エディタを使用して、インタフェースの外観を改良します。

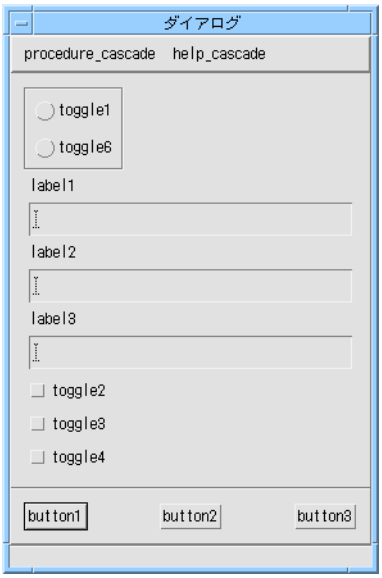


図 2-17 現段階でのダイナミックディスプレイ

追加したトグルボタンとラジオボックス内のボタンの外観には違いがあります。トグルボタンがラジオボックス内にある場合は、ラジオボタンになります。ラジオボタンは丸いインジケータにより区別されます。ラジオボックスの子ではないトグルボタンは、個々にオンとオフに切り換えることができ、正方形のインジケータを持ちます。

デザインへの最後のウィジェットの追加が終わりました。ここで「別名保存」コマンドを使用して作業を保存します。

3. 「ファイル」メニューから「別名保存」を選択します。
4. 「選択」テキストフィールド内のテキストの右側をクリックします。
5. デザインを保存するファイル名を入力します。

   規約により、Sun WorkShop Visual デザインファイルには、接尾辞 `.xd` が付けられます。

6. 「了解」をクリックします。

   すでにデザインを保存してある場合は、「保存」コマンドを使用することができます。

アプリケーションへのウィンドウの追加

以上で、インタフェースのメインウィンドウの設定が終了しました。ただし、大部分のインタフェースには複数のウィンドウがあります。本章では、インタフェースに 2 つめのウィンドウを追加する方法について説明します。2 つめのウィンドウは、図 2-18 に示すような単純なヘルプ画面です。

図 2-18 ヘルプ画面

このヘルプ画面は、ユーザーがインタフェースのヘルプメニューの「About This Layout」オプションを撰択すると表示され、ユーザーが「OK」ボタンをクリックすると消去されます。この動作は、Sun WorkShop Visual の著作権画面の動作と似ています。開始する前に、Sun WorkShop Visual のヘルプメニューをプルダウンし、「WorkShop Visual について」を選択して動作を確認してください。著作権画面は、「WorkShop Visual について」をクリックすると表示され、「了解」ボタンをクリックすると消去されます。

上記の動作を実装するには、以下のようにします。

1. インタフェース用に追加のウィンドウを作成します。
2. 2 つめのウィンドウ内で単純なヘルプ画面をデザインします。
3. ヘルプメニューで「ヘルプ」コマンドが実行されたときにヘルプ画面が表示されるように「表示」リンクを作成します。これは、212 ページの「リンク」で説明します。
4. ユーザーが「OK」ボタンをクリックするとヘルプ画面が消去されるよう、「非表示」リンクを作成します。これも、212 ページの「リンク」で説明します。

2 つめのウィンドウの作成

デザイン階層内で選択しているウィジェットには関係なく、いつでもウィンドウを新規に作成できます。

ダイアログをインタフェースに追加するには、以下のようにします。

1. ウィジェットパレットのダイアログシェルアイコンをクリックします。

   シェルの種類の違いについては、82 ページの「シェルの型」 を参照してください。

   Sun WorkShop Visual は構成領域を消去し、新規ウィンドウ用の階層を表示します。この時点では、新規ウィンドウはシェルだけで構成されています。最初のウィンドウのダイナミックディスプレイは表示されたままなので、2 つめのウィンドウを構築する際に、両方のダイナミックディスプレイを同時に見ることができます。
2. ダイアログテンプレートアイコンをクリックします。
3. ラベルアイコンをクリックします。
4. プッシュボタンアイコンをクリックします。

図 2-19 に、サブウィンドウの階層およびデフォルトのダイナミックディスプレイを示します。この画面は非常に単純なので、作業領域には子を持つコンテナウィジェットの代わりにラベルを使用することができます。ダイアログテンプレートは、ボタンボックス内のプッシュボタンをその上にある作業領域とともに中央に配置します。メニューバーはありません。

図 2-19　2 つめのウィンドウの階層およびデフォルトのダイナミックディスプレイ

ラベルおよびプッシュボタン上にテキストを設定します。

5. デザイン階層内のラベルをダブルクリックして、ラベルのリソースパネルを表示します。
6. 「表示」ページで、「ラベル」ボックスの中をダブルクリックし、次のように入力します。

   ```
   This dialog can be used
   to provide help for your
   application
   ```

   Return キーを使用して、新規行を複数行ラベルに入力します。最終行の最後に新規行を追加しないでください。
7. 「適用」をクリックします。
8. デザイン階層内のプッシュボタンをダブルクリックします。
9. 「ラベル」ボックスの中をダブルクリックし、次のように入力します :OK
10. 「適用」をクリックします。
11. 「閉じる」をクリックします。

ウィンドウ間の移動

デザインにシェルを追加する際は、シェルの型に関係なく、 Sun WorkShop Visual のメインウィンドウの右上のウィンドウ保持領域に、対応するアイコンが表示されます (図 2-22 を参照)。あるウィンドウの階層から別のウィンドウの階層に移動するには、ウィンドウ保持領域内で、移動先のウィンドウに対応するシェルアイコンをクリックします。シェルアイコンの外観は通常よく似ています。また、必ずしも作成順にウィンドウ保持領域に表示されるわけではないため、他のシェルアイコンと区別がしにくい場合があります。そこで、ウィンドウ保持領域内のアイコンを区別することができるよう、すべてのシェルアイコンに明示的な変数名を割り当て、「表示」メニューの「ダイアログ変数名を表示」オプションをオンにすることをお勧めします。

ウィンドウ保持領域でのシェルの順序には意味があります。デザインをファイルに保存する際は、シェルはウィンドウ保持領域に表示された順序 (左から右) で保存されます。ファイルを読み込む際は、Sun WorkShop Visual はその順序を保持し、最初 (左端) のシェルの階層およびダイナミックディスプレイを最初に表示します。これが最初に表示したいシェルではない場合は、シェルアイコンをクリックし、マウスボタン 1 を押したままそのアイコンを新しい位置までドラッグすると、ウィンドウ保持領域内のシェルの順序を変更することができます。

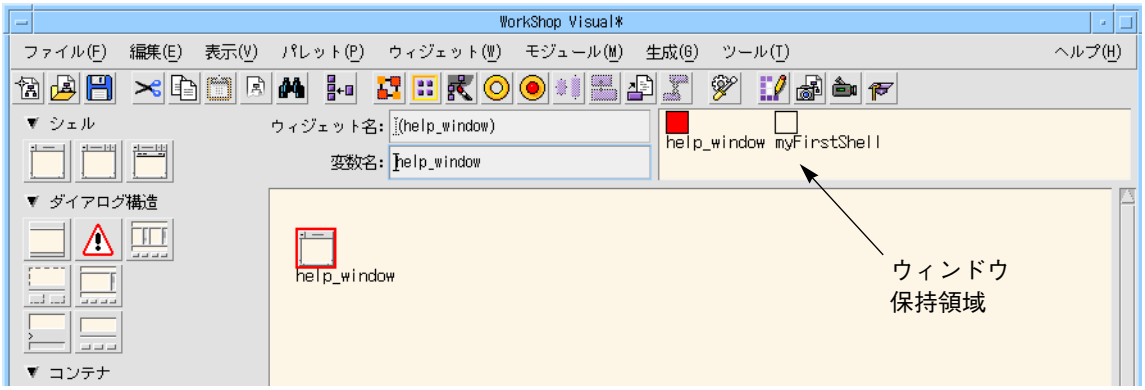


図 2-20 Sun WorkShop Visual 画面の上部

デザイン内の 2 個目のシェルに名前を割り当てます。

1. 階層内でダイアログシェルが選択されていない場合は、シェルを選択します。
2. 「変数名」フィールドの中をダブルクリックします。

3. 次のように入力します。

   help_window

   デフォルトでは、シェルの名前はウィンドウ保持領域に表示されます。シェルの型を確認する場合は、次のようにしてデフォルト表示をオフにできます。

4. Return キーを押すか、階層内の任意の別のウィジェットを選択します。

   デフォルトでは、ウィンドウ保持領域にシェルの名前が表示されますが、名前を表示しないように設定することもできます。この場合にシェルの型を確認するには、次のようにします。

5. 「表示」メニューをプルダウンし、「ダイアログ変数名を表示」のトグルをオフにします。

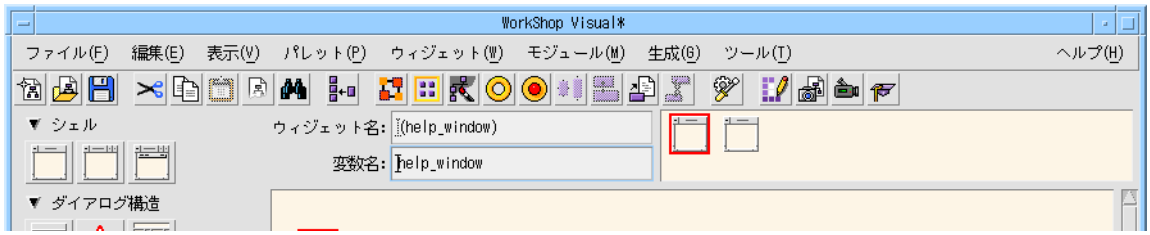


図 2-21　ウィンドウ保持領域でのダイアログ名の非表示

- これで、本章の段階的な学習は終了です。

  この時点で、次の章に直接進んで学習を続けるか、以降のページで解説されているさまざまな編集機能を読んで、実際に使用してみることもできます。

階層の編集

Sun WorkShop Visual は、階層を編集するためのドラッグと、カット & ペースト機能を備えています。これらの機能を使用すると、指定したリソース値を失うことなくデザインを変更することができます。

すべての編集機能は、選択されたウィジェットの子に対して等しく作用します。このため、 フォームまたはローカラムなどのコンテナウィジェット、およびその下のウィジェットすべてを 1 つの単位として移動することにより、コンテナウィジェット内部のウィジェットの相対位置を維持することができます。

階層でのウィジェットのドラッグ

ウィジェットおよびその子を新しい位置にドラッグするためには、ウィジェット上でマウスボタン 1 を押したまま新しい位置へドラッグします。ドラッグしているウィジェットが親として有効なウィジェットの下に正しく位置付けられると、そのウィジェットを新しい親と接続する垂直線が現れます。この線が表示されたら、マウスボタンを放します。線が現れる前にマウスボタンを放すと、ドラッグされたウィジェットは以前の位置に戻ってしまいます。

ウィジェットをドラッグする場合の規則

ウィジェットの場合は、同じ親の下での別の位置、または別の親にドラッグすることができます。ただし、Sun WorkShop Visual では、Motif で無効となっている位置へのウィジェットのドラッグはできません。

メインウィンドウの一部を構成するスクロールバーのように、コンポジットウィジェットの一部であるウィジェットの場合は、その親をドラッグします。自身をドラッグすることはできません。

ウィジェットの子は、そのウィジェットと一緒にドラッグされるため、ウィジェットをその子の下の位置にドラッグすることはできません。この操作を行う場合は、以下に説明するコピー機能を使用します。

ウィジェットのドラッグを開始した後で操作を止めるには、構成領域の空の場所にドラッグを行なって、ドラッグを取り消すことができます。

シェルウィジェットは、ウィジェットのどのクラスの子としても無効であるため、ドラッグすることはできません。

ウィジェットのコピー

元のウィジェットをそのままにしておき、そのウィジェットおよびその子を新しい場所にコピーする場合は、マウスボタン 2 を使用してウィジェットをドラッグします。コピーしたウィジェットには、デフォルトの変数名が割り当てられます。

編集コマンド: カット、ペースト、コピー、消去

「編集」メニューには、階層の変更に使用することができる「カット」、「ペースト」、「コピー」、「消去」コマンドがあります。ウィジェットおよびその子を Sun WorkShop Visual クリップボードにコピーするには、ウィジェットを選択して「コピー」(<*Copy*>) コマンド を使用します。

「カット」(<*Cut*>) は、選択されたウィジェットとその子を削除して、クリップボードにコピーします。「消去」もまた選択されたウィジェットとその子を削除しますが、クリップボードには影響しません。消去された項目は、階層にペーストすることはできません。

「ペースト」(<*Paste*>) は、現在選択されているウィジェットのすぐ下に、クリップボードの内容を挿入します。クリップボードが空である場合、あるいはクリップボードにあるウィジェットが現在選択されているウィジェットの子として有効ではない場合は、「ペースト」は使用できなくなります。ペーストされたウィジェットは、常に選択されているウィジェットの最後の子になります。これを異なる場所に配置するには、選択されているウィジェットをマウスを使用してドラッグします。

ファイルにコピー、ファイルからペースト

Sun WorkShop Visual クリップボードへのコピーに加え、ウィジェットおよびその子はクリップボードファイルにコピーすることができます。また、既存のクリップボードファイルからウィジェットにペーストすることも可能です。この機能を使用すると、標準メニューバーのようなデザインの一部を集めたライブラリを作成することができます。規約により、 Sun WorkShop Visual クリップボードファイル名には、接尾辞 `.cxd` が付けられます。

その他のウィジェットの選択方法

強調表示されていないウィジェットの選択は、マウスを使用するか、階層内を上下左右に進む矢印キーを使用して行うことができます。矢印キーを使用した選択は、構成領域に入力フォーカスがある場合にのみ有効です。矢印キーが使用できない場合は、<*Tab*> を使用して、 Sun WorkShop Visual 画面のさまざまな領域でフォーカスを移動して、目的のウィジェットを選択してください。

検索

「編集」メニューにある検索機能を使用して、プレリュード、コールバック、メソッド、トランスレーション、ウィジェットまたは変数名 (あるいはその両方)、および文字列リソースにある文字列を検索することができます。図 2-22 のような検索ダイアログが表示されます。

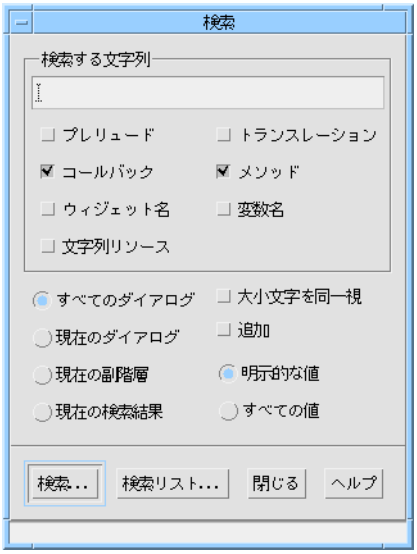


図 2-22 検索ダイアログ

検索ダイアログには、検索対象文字列を入力するテキストボックス、検索する場所を指定するトグル、検索するウィジェットを定義するトグル、検索に影響を与えるオプションの 4 つの領域があります。

検索する文字列

このテキストボックスには、任意の文字列または文字列の一部分を入力することができます。このテキストボックスを空白にすると、すべての文字列が検索されます。検索機能は「検索する文字列」領域で選択されたオプションに従って、ウィジェットの文字列を調べます。

文字列種類の指定領域

この領域には、ウィジェット中で検索する文字列の種類を指定するための、トグルが含まれています。トグルには、プレリュード、コールバック、メソッド、トランスレーション、ウィジェット名、変数名、文字列リソースがあります。これらは、一度にいくつでも選択することができます。1 つも選択しないと、一致する文字列はなくなります。

検索対象の指定

検索機能を使用するには、以下のようにします。

- すべてのダイアログ
  現在のデザインにあるすべてのダイアログ内を検索します。
- 現在のダイアログ
  現在のダイアログ内だけを検索します。
- 現在の副階層
  選択されているウィジェットと、その下の階層内を検索します。
- 現在の検索結果
  前回の検索により、すでに検索リストダイアログに存在するウィジェット内だけを検索します。

検索オプション

一致するものを検索する際に、Sun WorkShop Visual に大文字と小文字の区別を無視させるかどうかを選択できます。また、以前の検索で見つかった既存のウィジェットのリストに追加するかどうかも選択することができます。文字列リソース、ウィジェット名、変数名のうちいずれかを検索している場合は、明示的に設定した値だけを検索するのか、デフォルトを含めたすべての値を検索するのかを選択することができます。

検索リスト

「検索」ボタンを押すと、検索基準に該当するウィジェットのリストが「検索リスト」ダイアログという別のダイアログに表示されます。「検索リスト」ボタンを押すと、検索されたウィジェットのリストを表示します。この動作で検索が繰り返されることはありません。「検索リスト」ダイアログを閉じてしまった後で同じリストを再度表示する場合に便利です。

「検索リスト」ダイアログ

「検索リスト」ダイアログは、1 つまたは複数の検索基準に一致するウィジェットのリストを表示します。このリストからウィジェットを選択し、以下のオプションを使用することができます。

- 移動
  デザイン階層内で対応するウィジェットが選択されます。ウィジェットがフォールドされている階層に含まれている場合には、その階層がアンフォールドされます。ウィジェットが現在のダイアログ内に存在しない場合には、関連するダイアログがまず選択されます。文字列がウィジェットや変数名以外の場所で見つかった場合には、その文字列を含んでいるダイアログまたはリソースパネルが開かれます (例:「コールバック」ダイアログなど)。検出された対象をダブルクリックすると、「移動」を押した場合と同様の効果を得ることができます。

- 次
  リスト内の次のウィジェットが選択され、選択されたウィジェット上で「移動」と同じ動作が行われます。

- 消去
  別の検索を実行できるように、リストを消去します。

ウィジェットを削除すると、リセットあるいはカット & ペーストのような一時的削除と同様に、リストからそのウィジェットが取り除かれます。

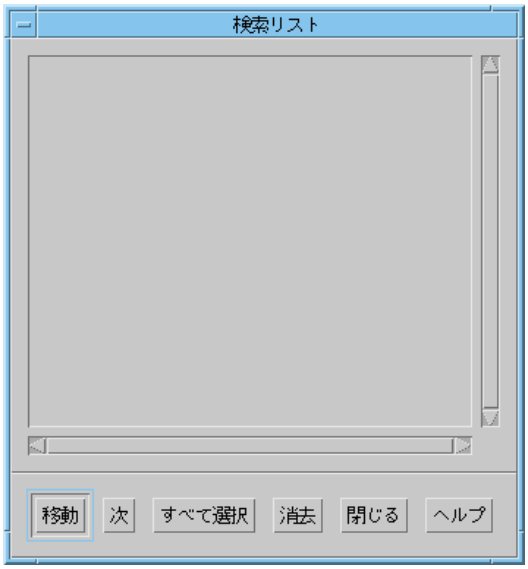


図 2-23 「検索リスト」ダイアログ

スピード検索機能

デザインが複雑になってくると、特定のウィジェットの検索が難しくなりますが、Sun WorkShop Visual では、ダイナミックディスプレイ (デザイン中のインタフェースの模擬実装。最終的にできるアプリケーションの外観と動作の確認に使用します) から直接ウィジェットに移動することができます。ポインタで、ダイナミックディスプレイに表示されたウィジェットを指定して *<Ctrl-G>* を押してください。すると、メインウィンドウのウィジェットが即座に強調表示されます。このとき、必要に応じてウィジェット階層を展開することもできます。

簡単な例を図 2-24 に示します。まず、シェルウィジェットのリソースパネルから「設定」ページを開いて、「キーボードフォーカス」を「ポインタ移動のみ」に設定して (47 ページの「キーボードフォーカスとスピード検索機能」を参照)ください。
このように設定してから、ダイナミックディスプレイでラベルにポインタを合わせて *<Ctrl-G>* を押すと、階層内の対応するラベルが強調表示されます。

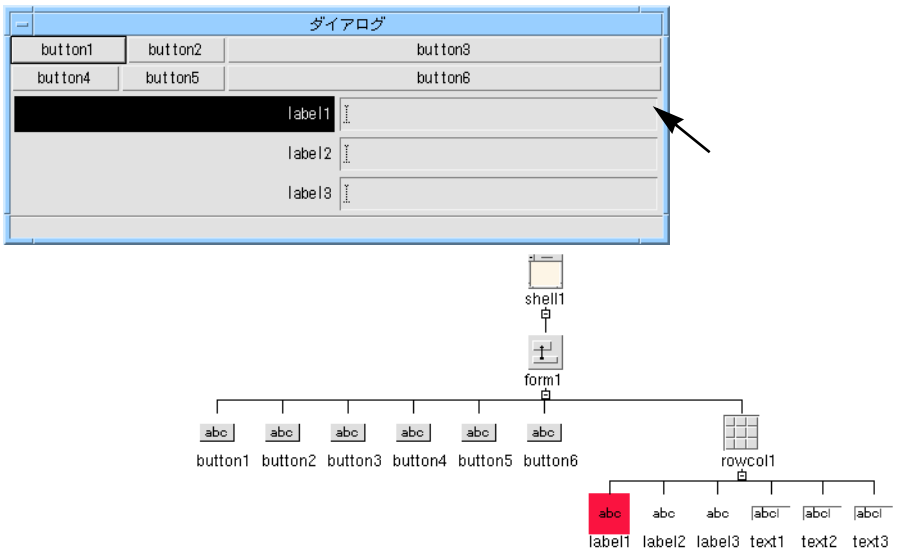


図 2-24　スピード検索機能

キー操作 *<Ctrl-G>* はトランスレーションです。したがって、次のリソースで変更することができます。

```
visu.fastFindTranslation: Ctrl<Key>G
```

生成されたコードでスピード検索を実行してもトランスレーションに影響はありません。

キーボードフォーカスとスピード検索機能

スピード検索機能では、ダイナミックディスプレイでフォーカスしたウィジェットを検索することができます。ラベルなど、通常はフォーカスされないウィジェットをスピード検索する場合は、シェルウィジェットのリソースパネルで「設定」ページを開いて、シェルのキーボードフォーカスを「ポインタ移動のみ」に設定してください。

注 - コードの生成時には、シェルのキーボードフォーカスの設定を元に戻してください。設定を元に戻さずにコードを生成すると、シェルごとにキーボードフォーカスの方式が異なるなど、統一性を欠いたアプリケーションが作成される可能性があります。

スピード検索機能の構成

デフォルトのトランスレーションは *<Ctrl-G>* です。*<Ctrl-G>* は、Motif ウィジェットセットのトランスレーションと衝突することはありません。スピード検索機能は、トランスレーションをサポートする他社製のウィジェットでも機能します。ただし、ウィジェットによっては、このデフォルトのトランスレーションが適当でない場合もあります。トランスレーションの変更は、以下のウィジェットを対象に適用されます。

1. Sun WorkShop Visual のすべてのウィジェット
   次のリソースは、すべてのウィジェットに影響を及ぼします。

   ```
   visu.fastFindTranslation: Ctrl<Key>G
   ```

2. 指定したクラスのすべてのウィジェット
   次の例では、最初の例のクラス「XmText」と 次の例のクラス「xrtTable」のすべてのウィジェットに影響があります。

   ```
   visu.XmText.fastFindTranslation:   Ctrl<Key>F
   visu.xrtTable.fastFindTranslation: Meta<Key>M
   ```

3. 特定のウィジェットのインスタンス
   次の例では、「my_text_widget」というウィジェットだけに影響があります。

   ```
   visu.my_text_widget.fastFindTranslation: Ctrl<Key>K
   ```

推奨されているデフォルトのトランスレーションが特定のウィジェットの動作と衝突する場合は、このウィジェットの型のトランスレーション操作を変更しなければならないことがあります。

スピード検索機能のトランスレーションは、ウィジェットクラスごとに必要な入力操作を指定して、そのクラスに合わせて構成することができます。次に例を示します。

```
visu.XmText.fastFindTranslation:   Ctrl<Key>F
visu.xrtTable.fastFindTranslation: Meta<Key>M
```

スピード検索は、ウィジェットのインスタンスに対しても構成することができます。

```
visu.my_text_widget.fastFindTranslation: Ctrl<Key>K
```

注 - スピード検索機能はオフにすることもできます。このためには、スピード検索機能のトランスレーションを無効にしたり、置換するようなウィジェットのトランスレーションを指定してください。

スピード検索機能を無効にする

予約値 <None> を使用して、指定したクラスまたはインスタンスのウィジェットをスピード検索するトランスレーションのアプリケーションを無効にすることができます。Sun WorkShop Visual のスピード検索メカニズムが正しく動作せず、指定したウィジェットやウィジェットクラスを検索できない場合、visu_config のウィジェットの値を変更する代わりに、スピード検索機能を無効にしなければならないことがあります。以下に 2 つの例を示します。1 番目の例はウィジェットのクラスを、2 番目の例は特定のウィジェットインスタンスを示しています。

```
visu.xintGraphObject.fastFindTranslation: <None>

visu.my_text_widget.fastFindTranslation:  <None>
```

次のように簡単な入力を行うと、どのウィジェットを対象としたスピード検索も無効になります。

```
visu.fastFindTranslation: <None>
```

他社製のウィジェットを使用する場合は、visu_config でウィジェットページを開き、「ウィジェット検索を無効化」トグルを選択すると、スピード検索機能を無効にすることができます。

ガジェットとスピード検索

スピード検索機能でトランスレーションを使用すると、ガジェット (トランスレーションをサポートしておらず、独自のウィンドウも持っていない) を検索することができません。このような場合、Sun WorkShop Visual では、階層内においてトランスレーションをサポートしている、ガジェットの最も近い祖先が検出されます。

表示オプション

Sun WorkShop Visual には、デザイン階層の表示方法を変更する方法が数多く用意されています。そのうちの多くは「表示」メニューから撰択することができます。「表示」メニュー内のオプションは、Sun WorkShop Visual 上での表示方法を変更するだけで、デザインには影響しません。

変数名を表示

このオプション (*<Ctrl-W>*) は、図 2-25 に示すように、構成領域にあるアイコンの下にそれぞれのウィジェット変数名を表示します。表示される名前はウィジェットのクラス名ではなく、ウィジェットに割り当てられる固有の変数名です。

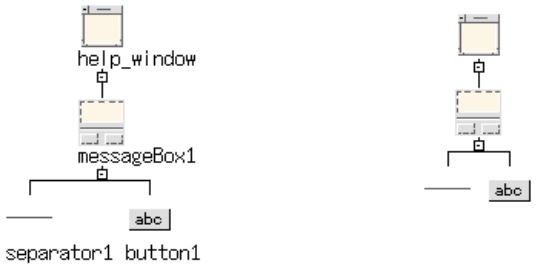


図 2-25 ウィジェット変数名

ダイアログ変数名を表示

デザイン内の各シェルウィジェットは、Sun WorkShop Visual 画面の右上隅の矩形領域にアイコンで表示されます。この矩形領域は、ウィンドウ保持領域と呼ばれます。「表示」メニューの「ダイアログ変数名を表示」(*<Ctrl-D>*) は、図 2-26 に示すように、ウィンドウ保持領域にあるアイコンの下に各シェルウィジェット変数名を表示します。アイコンは、名前を収めるために縮小されます。この機能は、複数のウィンドウを使用して配置を行う場合に便利です。

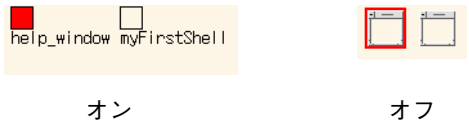


図 2-26 ウィンドウ保持領域内のダイアログ変数名

ツリーの左寄せ

「表示」メニューの「ツリーの左寄せ」オプション (*<Ctrl-L>*) は、構成領域にある階層の外観を、中央から左右に枝を伸ばすツリーから、図 2-27 に示すような右方向にだけ枝を伸ばす左寄せされたツリーに変更します。この機能は、大きなデザイン中で、親ウィジェットを素早く探し出す場合に便利です。

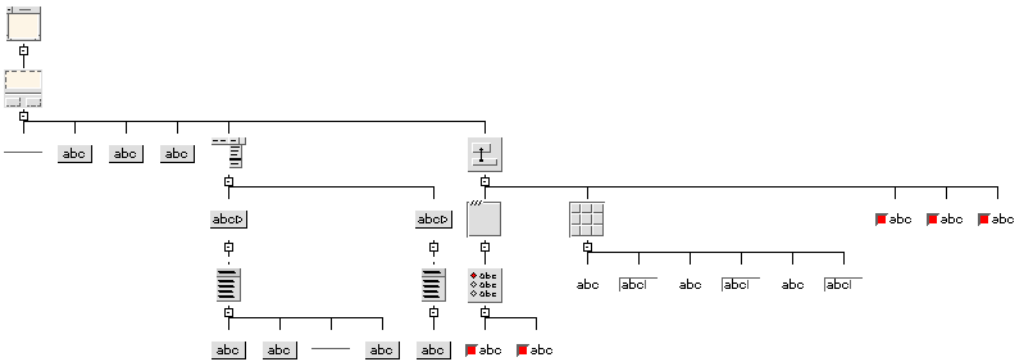


図 2-27　ツリーの左寄せ

ウィジェットの縮小

このオプションは、階層が大きな場合に、ウィンドウのサイズを変更せずに構造のより多くの部分を表示する際に便利です。ウィジェットは、図 2-28 に示すように、同じ形の小さな正方形に縮小されるため、構成領域にはそれだけ多くの範囲が収まります。しかし、ウィジェットクラス、およびフォールドされているウィジェットとフォールドされていないウィジェットとの区別はできなくなります。その他の表示オプションと同様、実際のデザインには影響しません。

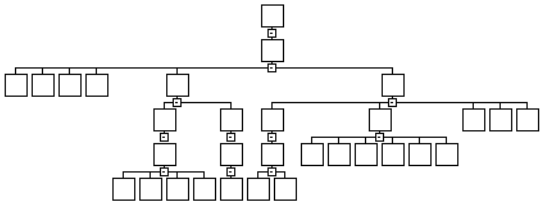


図 2-28　ウィジェットの縮小

ウィジェット注釈

Sun WorkShop Visual には、特定の属性を持つウィジェットを区別するためにデザイン階層に注釈を付ける機能があります。「表示」メニューには、図 2-27 に示すような「ウィジェット注釈ティアオフ」というプルライト・ティアオフメニューがあります。図 2-29 に注釈の付いた階層を示します。

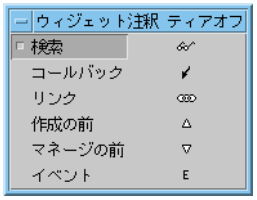


図 2-29 注釈メニュー

このメニューには、6 種類の注釈があり、それぞれがトグルボタンになっています。デザイン階層内でこれらの属性を持つウィジェットを確認するには、その属性トグルを選択します。その属性に対応した記号が、以下の基準に従って階層内で関連のあるウィジェットの横に表示されます。

- コールバック、作成の前プレリュード、マネージの前プレリュード
  ウィジェットにこれらの属性が与えられていれば、対応する記号が表示されます。コールバックを選択すると、メソッド宣言を持つウィジェットや、下位クラスにメソッドが追加されているクラスであるウィジェットに注釈が付けられます。
- リンク
  ウィジェットがリンク元であれば、対応する記号が表示されます。
- 検索
  ウィジェットが以前の検索で見つかっていれば、対応する記号が表示されます。
  詳細は 43 ページの「検索」を参照してください。

図 2-30 に、一連の注釈をすべて使用した例題の階層の一部を示します。

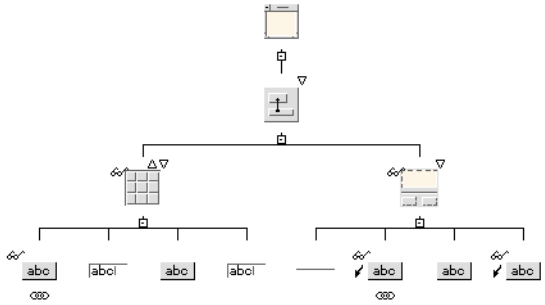


図 2-30 注釈付きの階層

注釈記号の構成変更

注釈記号は、6 個の記号が明確に表示されるように、階層内のウィジェットアイコン周辺に表示されます。アイコンに相対して記号が表示される場所、記号が使用するスペース、ピックスマップの名前はすべて Sun WorkShop Visual リソースファイルに指定されています。この指定は変更することができます。Sun WorkShop Visual のアプリケーションリソースの詳細は、993 ページの付録 D「アプリケーションのデフォルト」を参照してください。例として検索記号のリソースを以下に示します。

```
Visu*annotate_search.annotatePosition:NorthWest

Visu*annotate_search.annotateWidth:10

Visu*annotate_search.annotateHeight:3

Visu*annotateSearchPixmap:an_search.xpm
```

上記の最初の行は、地理的な位置 NorthWest (北西) を指定します。
これは、ウィジェットアイコンを基準とした位置であり、羅針盤上の 8 個の方位の中から指定することができます。

構造の色

「表示」メニューの「構造の色」オプションは、構造化コード生成機能を使用するデザインを構築する際に便利です。このオプションは、関数またはデータ構造、C++ クラス、あるいは子のみを生成するウィジェットの色を指定します。

「構造の色」は、プルライトサブメニューです。サブメニューから「色を表示」を選択して、適切な色で構造の表示を行います。サブメニューは、上辺の破線をクリックすると、カラーコードを参照するためにメニューを切り離すことができます。

構造以外として指定されているウィジェットは、通常の背景色で表示されます。

ウィジェットのフォールド/アンフォールド

「ウィジェット」メニューではフォールド/アンフォールドを使用できます。この機能は、構成領域のウィジェットの外観に影響します。階層が大きくなると、多数の子を持つウィジェットをフォールドして、その子が構成領域内で大量のスペースを占めないようにする必要があります。たとえば、学習用の配置で、子が大量のスペースを占めているメニューバーウィジェットをフォールドします。ウィジェットがフォール

ドされると、そのウィジェットの子は階層には表示されなくなります。ウィジェットのフォールドはディスプレイを見やすくするものであり、デザインからウィジェットを削除する機能ではありません。

ウィジェットのフォールドには、2 つの方法があります。1 つは「ウィジェット」メニューを使用した方法で、もう 1 つは階層内の特別なアイコンを選択する方法です。

「ウィジェット」メニューの使用

フォールドするウィジェットを選択します。「ウィジェット」メニューをプルダウンして「フォールド/アンフォールド」(*<Ctrl-F>*) を選択します。フォールドされているウィジェットを選択して同じコマンドを使用すると、そのウィジェットのフォールドが解除 (アンフォールド) されます。

階層フォールドアイコンの使用

階層内のフォールドアイコンを選択します。このアイコンは、子を持つ各アイコンの下に表示され、外観はマイナス記号 (-) が入った小さなボックスです。このアイコンをクリックすると、その下の階層がフォールドされます。この方法と「ウィジェット」メニューのどちらを使用するかにかかわらず、階層がフォールドされている場合はフォールドアイコンの表示はプラス記号 (+) に変わります。このアイコンを選択すると、その下の階層のフォールドが解除されます。

図 2-31 に、フォールドされたウィジェットを示します。

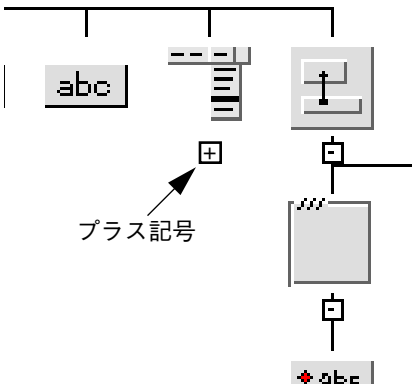


図 2-31 現段階の学習例の階層でメニューバーウィジェットをフォールドした例

階層の印刷

「印刷」ダイアログを使用すると、階層を作成している途中で階層を紙に印刷することができます。図 2-32 に「印刷」ダイアログを示します。

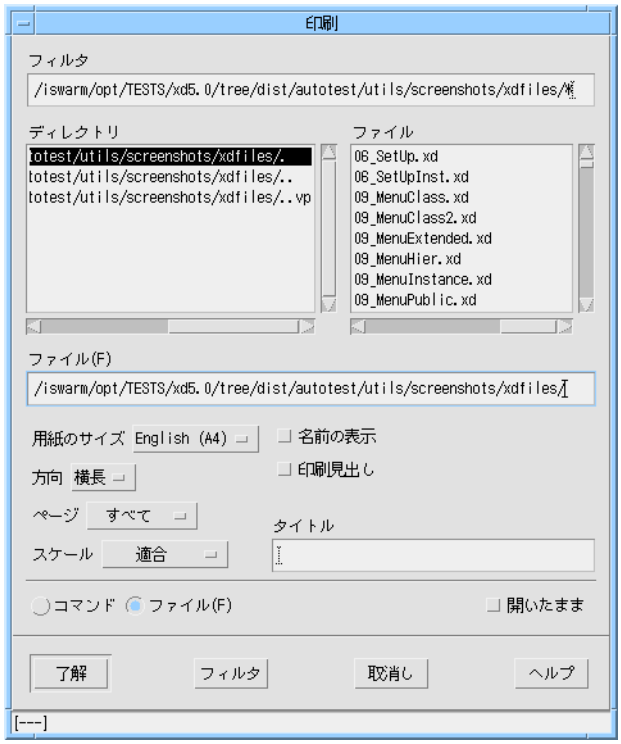


図 2-32 「印刷」ダイアログ

ファイルに印刷する場合は、「ファイル」トグルをクリックし、「ファイル」テキストボックスにファイル名を入力します。プリンタに送る場合は、「コマンド」トグルをクリックして、ファイルへの印刷と同じテキストボックス (「ファイル」は「コマンド」に切り替わる) に `lp` などのコマンドを入力します。出力は PostScript であるため、PostScript プリンタまたはビューアが必要です。

「印刷」ダイアログにあるオプションメニューを使用すると、用紙のサイズ、配置方向、印刷するページ、スケールを指定することができます。「スケール」オプションで縮小スケールを選択すると、実際のサイズの 3 分の 2 の大きさで印刷されます。

「適合」オプションが選択されていない場合は、必要なだけのページを使用して階層全体の印刷が行われるため注意が必要です。「ページ」オプションメニューを使用すると、デザインに複数のウィンドウが含まれている際にはデザインのすべての階層を、あるいは構成領域に現在表示されている階層のみを印刷することができます。

「名前の表示」トグルを選択すると、変数名を印刷することができます。また、「印刷見出し」トグルを選択すると、階層の周囲にフレームが配置され、「タイトル」テキストフィールドで指定したタイトルを印刷することができます。

ファイルブラウザの使用

図 2-33 に示されているファイルブラウザを使用すると、開く、または保存するファイルの名前を指定することができます。ファイルブラウザは、「ファイル」メニューから「開く」または「別名保存」などの、ファイル名の指定を要求するコマンドが選択される場合に表示されます。「選択」フィールドへパス名を入力するか、マウスを使用して「ファイル」リストから既存ファイル名を選択することができます。

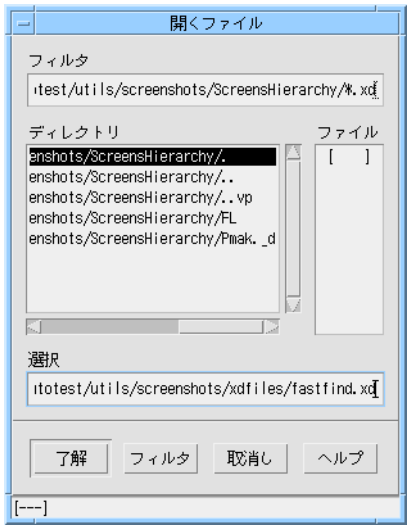


図 2-33 ファイルブラウザ

「フィルタ」テキストフィールドは、現作業ディレクトリおよびファイル名パターンを表示します。このパターンに合致したディレクトリまたはファイル名が「ファイル」リストに表示されます。現作業ディレクトリおよびファイル名パターンを変更する場合には、「フィルタ」テキストフィールドのテキストを編集し、画面の下部にある「フィルタ」ボタンを押してください。

「ディレクトリ」ボックスには、現作業ディレクトリのサブディレクトリが表示されます。ディレクトリ構造内を参照するには、リスト内のディレクトリをクリックして「フィルタ」ボタンを押すか、そのディレクトリをダブルクリックしてください。

ファイル名パターンは、「ファイル」のリストを制御します。現作業ディレクトリ内の、パターンに一致するすべてのファイル名が「ファイル」ボックスに表示されます。パターンを変更するには、テキストを編集して画面下部にある「フィルタ」ボタンを押します。パターンがアスタリスク (*) の場合は、現作業ディレクトリにあるすべてのファイルが表示されます。パターンが `*.xd` の場合は、接尾辞 `.xd` を持つファイルだけが表示されます。ファイルを選択するには、ファイル名をクリックして画面下部にある「了解」ボタンを押すか、ファイル名をダブルクリックしてください。ファイルを選択すると、Sun WorkShop Visual は「開く」、「読む」、「別名保存」などのユーザーが要求した操作を実行します。

ファイルまたは生成されたコードを保存する場合は、既存のファイル名を選択するか、「選択」フィールドに新しいファイル名を指定して「了解」ボタンを押してください。

注 - フィルタ処理が行われた後で現在のディレクトリにファイルが追加されている場合、それらのファイルは、フィルタ処理が再度行われるまで「ファイル」リストに表示されません。これは、ダイアログを閉じた後でファイルが追加された場合でも同様です。

第3章

リソース

はじめに

Motif では、ウィジェットの外観や動作はそのリソースによって制御されます。リソースには、色、フォント、イメージ、テキスト、タイトル、ウィンドウやウィジェットの位置およびサイズ、コールバック、インタフェースの動作に影響するその他すべてのカスタマイズ可能な引数が含まれます。リソースは直接的な影響とともに、間接的な影響を持つ場合があります。たとえば、ボタンのラベルを変更すると、新しいラベルのためのスペースを空けるようにボタンのサイズが一緒に変更されます。

ウィジェットをデザイン階層に加える場合、Sun WorkShop Visual はそのウィジェットのすべてのリソースにデフォルト値を使用します。しかし、多くの場合、ウィジェットを有効にするためにはユーザーがいくつかのリソースを明示的に設定する必要があります。これらのリソース設定を容易にするため、リソースパネルと呼ばれるダイアログ内には、いくつかのリソースがまとめられています。

リソースパネルの種類には、「ウィジェット」と「コア」の 2 つがあります。ウィジェットリソースパネルには、選択したウィジェットのクラスに関連したリソースが含まれています。ウィジェットのすべてのクラスに共通するリソースは基底クラス (コア、プリミティブ、マネージャ) に適用されるので、コアリソースパネルに表示されます。すべてのウィジェットクラスはコアクラスから派生しており、その大部分はプリミティブクラスまたはマネージャクラスからも派生しています。コアリソースパネルについては、88 ページの「コアリソースパネル」を参照してください。

デザインのソースコードを生成する際は、X リソースファイルも生成するよう選択することができます。これは独立したファイルで、一般ユーザーが変更可能なリソース設定を含んでいます。リソースファイルとその生成については、251 ページの「X リソースファイルの設定」を参照してください。リソースファイル内に生成するリソースの制御以外に、Sun WorkShop Visual は緩い結合と密な結合による制御方法も提供しています。この方法を使用すると、デザイン内のウィジェットにリソースを割り当てる方法を一括して管理できます。詳細は、97 ページの「リソースの結合」を参照してください。

本章では、リソースパネルを使用して以下の作業を行います。

- 階層内のウィジェットのリソースの表示
- ラベルおよびさまざまな型のボタンウィジェットのテキストの編集
- キーボードアクセラレータおよびニーモニックの指定
- メニューバーに対するヘルプウィジェットの指定
- ローカラムウィジェットにおける行および列の配置設定
- ダイアログフレーム上部のメインタイトルを含む、シェルウィジェットのリソースの編集

リソースパネルに加え、Sun WorkShop Visual はフォーム内でのウィジェットの配置、フォントおよびピックスマップの設定、*XmString* (Motif コンパウンド文字列) の編集および色の選択を行うための特殊なエディタを用意しています。配置エディタについては 109 ページの第 4 章「配置エディタ」を、フォント、カラー、ピックスマップ、コンパウンド文字列のエディタについては第 5 章「その他のエディタ」を参照してください。

ラベルリソースパネル

学習用インタフェースを作成するには、すべてのラベルとボタンのテキストをデフォルト以外に設定する必要があります。まず、ローカラムウィジェットにある 3 個のラベルを、 以下に示すように変更します。

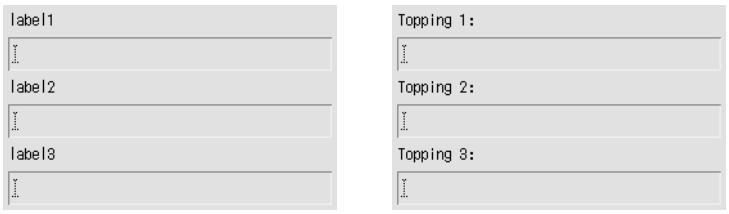


図 3-1　テキスト設定前と設定後のラベル

最初に、1 番目のラベルのリソースパネルを表示します。

1. ローカラムウィジェットの下にある最初のラベルをダブルクリックします。

   ラベルをダブルクリックすると、図 3-2 のようなリソースパネルが表示されます。

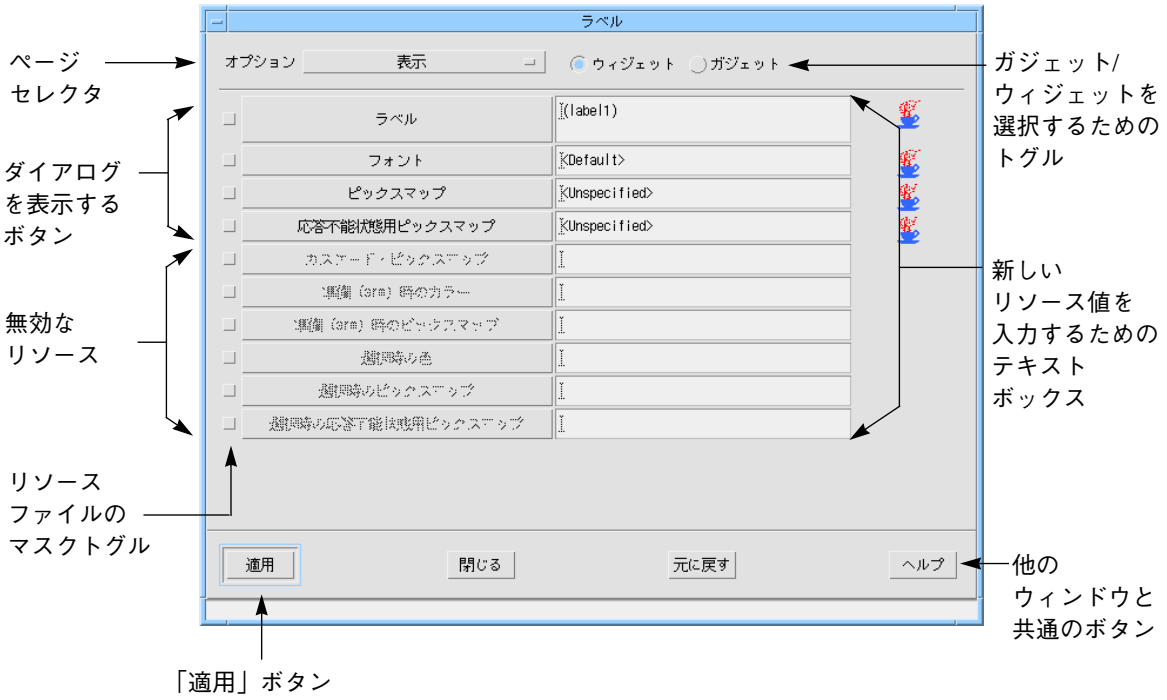


図 3-2　ラベルリソースパネル

リソースパネルは通常、複数のページで構成されています。パネルの左上隅にある「オプション」メニューで、リソースパネルのページを選択することができます。

2. 「表示」ページを撰択します。

   これで、ラベルのテキストを編集することができます。

3. 「ラベル」の右側にあるテキストボックスをダブルクリックします。

   これらのボックス内でのテキスト編集は、ウィジェット名を割り当てる場合と同様の効果があります。ダブルクリックすると、ボックス内のテキストが強調表示されます。

4. 次のように入力します。

   `Topping 1:`

注 - ここでは <Return> を押さないでください。ラベルは複数行を含むことができるため、ここで <Return> を押すとラベルに改行文字が挿入されることになります。誤って <Return> を押してしまった場合は、<Back Space> を使用して改行文字を削除することができます。

---

5. リソースパネル下部の「適用」ボタンをクリックします。

   「適用」コマンドは、新しいリソース値を設定します。「適用」をクリックすると、ダイナミックディスプレイに新しいラベルテキストが表示されます。

注 - デフォルトのロケールが「C」以外で、かつラベルに国際的テキストを使用している場合は、フォントリストリソースを使用言語用に設定しない限り、そのテキストは正しく表示されません。詳細は、716 ページの「アプリケーションのフォントリソースの設定」を参照してください。正しいリソース設定がわからない場合は、705 ページの第 22 章「国際化」を参照してください。

リソースパネルの領域

ウィジェットのクラスが異なる場合は、リソースパネルも異なりますが、すべてのリソースパネルは同じ基本構造になっています。

注釈

リソースを変更すると、図 3-3 の「ラベル」リソースの横に示されるように、リソースパネルの右端に傍線が表示されます。

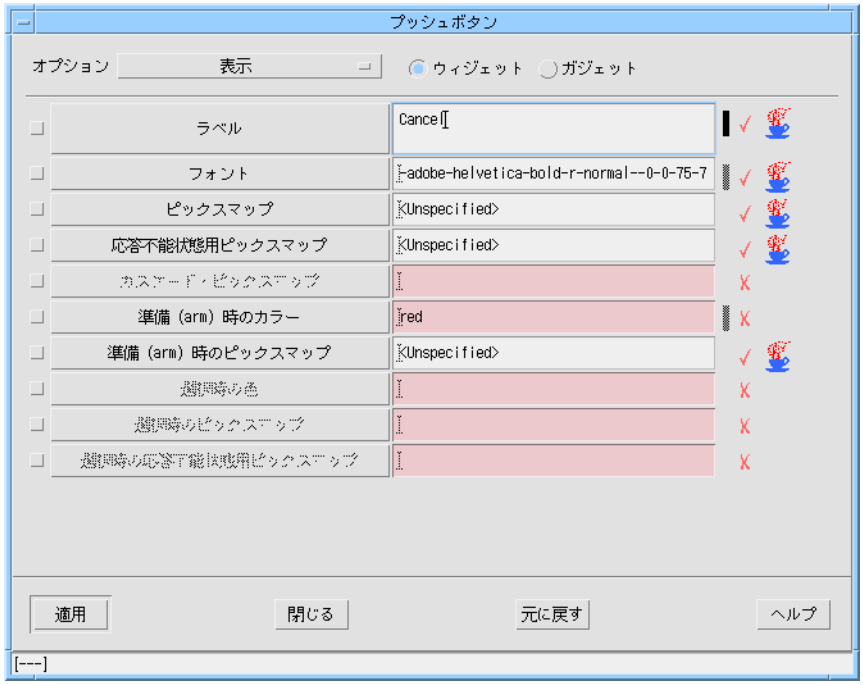


図 3-3　注釈付きのリソース

「適用」を押すと、変更傍線はグレー表示に変わります。

Java コードに加えられることになるリソースには、図 3-3 に示したように、リソースパネルの横にコーヒーカップの絵が表示されます。Sun WorkShop Visual を使用した Java コードの生成の詳細については、363 ページの第 10 章「Java 用のデザイン」を参照してください。

注 - アイコンに十分な色を割り当てることができない場合、Sun WorkShop Visual は文字「J」を使用して Java リソースを表します。

図 3-3 に示されるチェックマークと×印は、Microsoft Windows モードで動作中の場合に限り表示されます。チェックマークはリソースが Microsoft Windows でも適用可能であることを示し、×印は適用できないことを示します。詳細は、417 ページの「準拠か非準拠かの表示」を参照してください。

リソース名

リソース名はリソースパネルの左側に表示されます。リソース名は単純なラベルである場合と、リソース名自体がボタンになっている場合があります。リソース名のボタンを押すとそのリソースに関連するダイアログが表示されます。図 3-2 のリソースパネルでは、すべてのリソース名がボタンになっています。選択されたウィジェットに適用されないリソースは、グレー表示されます。

リソース値

現在のリソース値は右側のボックスに表示されます。これらのボックスは単一行のテキストフィールド、または複数行のテキストボックス、またはメニューオプションになっています。リソースの設定は、これらのボックスにキーボードから入力するか、オプションメニューから撰択します。デフォルト値は括弧内に表示されます。

マスクトグル

各リソース名の左側に、ラベルの付いていないトグルがあります。これは、コード生成段階で X リソースファイル内外のリソースをマスクするために使用するマスクトグルです。この項目については、260 ページの「リソースのマスク」で説明します。リソース値を設定するためにこれらのトグルを設定する必要はありません。

ページセレクタとトグルスイッチ

通常、リソースパネルの主要部分は複数のページで構成されています。パネル上部にあるオプションメニューは、ページセレクタです。ページセレクタを使用して、ページ間を移動することができます。ラベルのリソースパネルには、そのウィジェットをガジェットとして指定するために使用するトグルスイッチも含まれています。ウィジェットクラスに対応するガジェットがある場合は、ウィジェット/ガジェット・トグルが他のウィジェットクラスのリソースパネルに表示されます。ウィジェットクラスによっては、そのクラスに適したその他のトグルスイッチを持つ場合もあります。

共通のコマンド

適用

パネル下部の 4 個のボタンは、共通のコマンドです。「適用」は、新しいリソース設定を有効にします。「適用」をクリックせずに別のウィジェットを選択、またはリソースパネルを閉じると、編集した設定が失われます。

元に戻す、閉じる、ヘルプ

「元に戻す」は、編集したすべての設定を、最後に適用された設定に戻します。「閉じる」は、リソースパネルの表示を終了します。「ヘルプ」は、適切なヘルプ画面を表示します。

図 3-3 で「適用」ボタンが強調表示されていることに注意してください。リソースパネルが入力フォーカスを持っている場合に <Return> を押すと、通常は強調表示されているコマンドが実行されます。しかし、複数行のテキストボックス内で入力を行なっている場合に <Return> を押すと、この <Return> は改行文字として処理されるため、強調表示されているコマンドは実行されません。

次に、他の 2 個のラベルウィジェットに対してテキストを設定します。ラベルのリソースパネルを閉じる必要はありません。ただし、リソースパネルが構成領域を覆っている場合には移動させてください。

1. 2 番目のラベルウィジェットを選択します。

   パネル上の「ラベル」リソースは、新しく選択されたウィジェットの現在の設定を反映させるために変更されます。

2. リソースパネルの「ラベル」ボックスをダブルクリックして、デフォルトラベルを強調表示します。

   テキストボックスで編集を行う場合は、マウスでドラッグしてテキストを強調表示する、<Delete> を使用して強調表示されたテキストを削除する、または現在のカーソル位置に入力を行う、などの方法でも実行できます。

3. 次のように入力します。

   ```
   Topping 2:
   ```

4. 「適用」をクリックします。

5. 3 番目のラベルに対して、手順 1 から 手順 4 までを繰り返します。手順 3 では次のように入力します。

   `Topping 3:`

6. 「閉じる」をクリックします。

   リソースパネルが消去されます。

複数選択とリソース

ウィジェットを複数選択している場合でも、コアリソースパネルとウィジェットリソースパネルの両方でリソースを設定することができます。この場合は、選択したウィジェットすべてに関連するリソースに限って設定することができます。
Sun WorkShop Visual は、選択したウィジェットすべてに共通するウィジェットクラスを見つけることによって、関連するリソースを特定します。たとえばボタンとメニューバーのように、ウィジェット同士が大きく異なる場合は、共通に持っているクラスは、下位レベルのコアウィジェットクラスだけです。このような場合は、変更できるのはコアリソースだけになります。

Microsoft Windows モードでのリソースパネル

Microsoft Windows モードの場合、Sun WorkShop Visual は、作成しているデザインが確実に Microsoft Windows 準拠となるようにします。Motif リソースは、 Microsoft Windows に直接変換を行いません。Microsoft Windows 上で類似したリソース (必ずしもリソースである必要はありません) に変換されるリソースもありますが、変換が不可能なリソースもあります。

Microsoft Windows に変換されないリソースは、次の 2 通りの方法で特定されます。

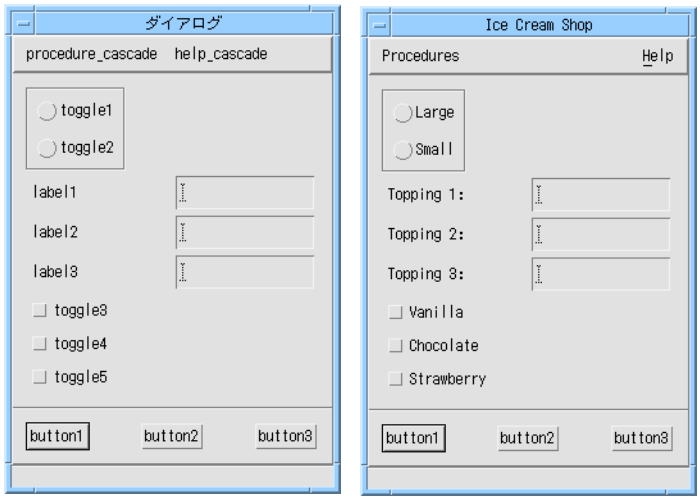
- 注釈アイコン (チェックマークまたは×印)
- 色

Microsoft Windows に変換されないリソースは、リソースパネル内のフィールドがピンク色で表示されます。これらのフィールドに値を入力した場合、Motif に対してはリソースが生成されますが、Microsoft Windows に対しては何も生成されません。これらの Microsoft Windows では適用されないリソースのフィールドを示す色は、

Sun WorkShop Visual アプリケーションリソースファイルにある項目を変更して行うことができます。427 ページの「Microsoft Windows 上のオブジェクトではない宛て先ウィジェット」を参照してください。

ボタンウィジェットリソース

同じ手順を使用して、トグルボタン、カスケードボタン、プッシュボタンのラベルを設定します。図 3-4 に示すように、トグルボタンのラベルを設定します。



デフォルトラベル　　設定後

図 3-4　ラベル設定前と設定後のトグルボタン

1. 3 つのトグルボタンのうちの最初のトグルボタンを ダブルクリックして、リソースパネルを表示します。

   選択されているウィジェットのリソースパネルは、以下の 2 通りの方法でも表示することができます。

   - 「ウィジェット」メニューから「リソース」コマンドを選択する。
   - ツールバーのリソースボタンを押すか、ウィジェットに入力フォーカスがあるときに <Return> を押す。

2. 「ラベル」ボックスをダブルクリックして次のように入力します。

   `Vanilla`

3. 「適用」をクリックします。
4. 2 番目のトグルボタンを選択します。
5. 「ラベル」ボックスをダブルクリックして次のように入力します。
   Chocolate
6. 「適用」をクリックします。
7. 3 番目のトグルボタンを選択します。
8. 「ラベル」ボックスをダブルクリックして次のように入力します。
   Strawberry
9. 「適用」をクリックします。

共有リソースパネル

トグルボタンとラベルは、共通のリソースパネルを使用しています。これは、Motif ではトグルボタンウィジェットクラスはラベルウィジェットクラスから派生しているためです。言い換えると、トグルボタンはラベルの特殊な種類、つまりサブクラスです。反対に、ラベルはトグルボタンのスーパークラスです。

ウィジェットクラスの継承

トグルボタンクラスは、ほとんどのリソースを含む多くの特性をラベルクラスから継承しています。継承により、すべての Motif ウィジェットクラスはクラス階層と呼ばれる階層に組織化されます。クラス階層は、使用可能なウィジェットクラスの抽象的な階層であるため、画面上で構築しているデザイン階層とは区別して考えます。

いくつかの型のボタン、つまりトグルボタン、プッシュボタン、カスケードボタン、描画ボタンは、ラベルクラスから派生しています。これらのクラスのすべてのウィジェットのテキストは、閉じて再度開くという動作を繰り返すことなく、同一のラベルリソースパネルを使用して設定することができます。

図 3-5 に示すように、ラジオボタン (ラジオボックスにあるトグルボタン) のラベルを設定します。

デフォルトラベル　　設定後

図 3-5　ラベル設定前と設定後のラジオボタン

1. ラジオボックスにある最初のトグルボタンをダブルクリックして、リソースパネルを表示します。
2. 「ラベル」ボックスをダブルクリックして次のように入力します。
   Large
3. 「適用」をクリックします。
4. 2 番目のラジオボタンを選択します。
5. 「ラベル」ボックスをダブルクリックして次のように入力します。
   Small
6. 「適用」をクリックします。

デフォルトよりも長いラベルをラジオボタンに割り当てると、新しい幅に対応するようにフレーム・ウィジェットのサイズが変更されます。これは、長いテキスト文字列を収めるためにトグルボタンのサイズが変更されると、それに対応してその親であるラジオボックス、さらにその親であるフレームのサイズも変更される、という連鎖反応を示します。Motif は、この動作を自動的に行います。この動作は学習用の配置ではあまり目立ちませんが、図 3-6 のような結果になります。

デフォルトラベル　　長いテキスト文字列

図 3-6　長いラベルを持つラジオボタン

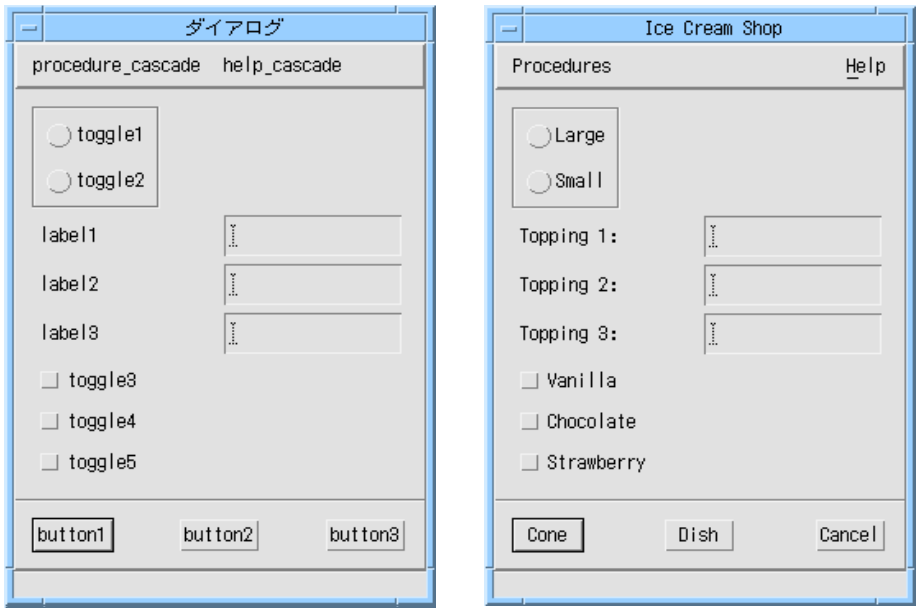
同じラベルリソースパネルを使用して、図 3-7 のようにカスケードボタンのラベルリソースを設定します。

デフォルトラベル　　設定後

図 3-7　カスケードボタンの設定 (メニューバー)

7. 最初のカスケードボタンを選択し、次のラベルを割り当てます。
   Procedures
8. 2 番目のカスケードボタンを選択し、次のラベルを割り当てます。
   Help

   最後に、ボタンボックス内の 3 個のプッシュボタンのラベルを、図 3-8 のように設定します。プッシュボタンは、ラベルを収めるために自動的にサイズ変更を行います。



デフォルトラベル　　設定後

図 3-8　リソース設定前と設定後のボタンボックス

9. 最初のプッシュボタンを選択し、次のラベルを割り当てます。
   Cone
10. 2 番目のプッシュボタンを選択し、次のラベルを割り当てます。
    Dish

11. 3 番目のプッシュボタンを選択し、次のラベルを割り当てます。

```
Cancel
```

ヒント

リソースを設定した後は、必ず「適用」をクリックしてください。「適用」をクリックしないと、新しい設定は有効になりません。新しい設定を適用する前に別のウィジェットを選択すると、図 3-9 に示すような警告が表示されます。

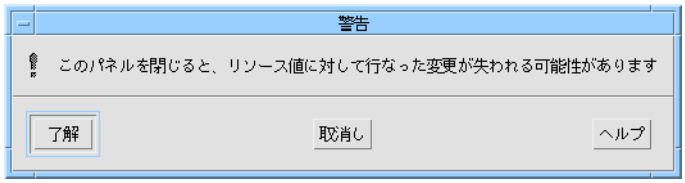


図 3-9　新しいリソースが適用されていないことを示す警告

リソースパネルに戻るためには、このディスプレイの「取消し」をクリックします。

メニュー項目のリソース

次に、メニューバーのメニューにあるボタンに対するリソースを設定します。現時点では、カスケードボタンのラベルが設定されているだけなので、メニューは図 3-10 のように表示されます。

図 3-10　カスケードボタンラベルの設定されたメニュー

1. 最初のメニューウィジェットの子である 1 番目のプッシュボタンを選択します。
2. 「ラベル」リソースを次のように変更します。

```
Wash Dishes...
```

省略符号 (...) は、メニュー項目がコマンドを実行する前に別のダイアログボックスが表示されることを示します。

3. 2 番目のプッシュボタンを選択し、次のラベルを割り当てます。
   `Count Money`
4. 3 番目のプッシュボタンを選択し、次のラベルを割り当てます。
   `Exit`
5. 2 番目のメニューウィジェットの子であるプッシュボタンを選択します。
6. 次のラベルを割り当てます。
   `About This Layout...`

   現段階でプルダウンメニューは、図 3-11 のようになります。

図 3-11 プッシュボタンラベルの設定されたプルダウンメニュー

次の節では、以下の作業を行います。

- 「Exit」ボタンへのアクセラレータ *<Ctrl-E>* の指定
- ユーザーにアクセラレータを示すため、「Exit」ボタンへの「Ctrl-E」ラベルの配置
- 「Help」に「H」を、「About This Layout」に「A」をそれぞれニーモニックとして指定

キーボードのページ

アクセラレータおよびニーモニックは、ラベルリソースパネルの別のページにあるキーボード・リソースで設定します。以下の手順に従って、「Help」カスケードボタンにニーモニック「H」を設定します。

1. 「Help」カスケードボタンを選択します。

2. リソースパネルのページセレクタから「キーボード」を選択します。

   図 3-12 に示されるような「キーボード」ページが表示されます。グレー表示されていないリソースが、カスケードボタンクラスに適用されます。

図 3-12 カスケードボタンのキーボードリソース

ニーモニック

Sun WorkShop Visual では、Sun WorkShop Visual インタフェースのニーモニックと同様に作動するキーボードニーモニックを設定することができます。ニーモニックには、ラベルで使用されていない文字も含め、任意の文字を指定することができます。ラベルに使用される文字の 1 つ、特に最初の文字を使用すると、一般ユーザーには便利なものになります。ニーモニックは、1 つのメニューバーあるいはメニューの中で同じ文字を使用することはできません。

1. 「ニーモニック」ボックスをダブルクリックします。
2. 次のように入力します。

```
H
```

3. 「適用」をクリックします。

4. 「Help」メニューの子であるプッシュボタンを選択します。

5. 「ニーモニック」ボックスをダブルクリックします。

6. 次のように入力します。

   A

7. 「適用」をクリックします。

   「H」と「A」の文字が下線付きで表示されるようになります。これは、Motif でニーモニックを示す手法です。この段階で、マウスを使用する、あるいは *<Meta-H><A>* を押すという 2 通りの方法で「About This Layout」コマンドを呼び出せるようになりました。

   ダイナミックディスプレイでニーモニックを表示する場合は、ウィジェットをリセットする必要があります。これを行うには、カスケードボタンが選択されていることを確認して、ツールバーのリセットボタンを押すか、ウィジェットメニューから「リセット」を選択します。

アクセラレータ

次に、「Exit」ボタンにキーボード・アクセラレータを追加します。Sun WorkShop Visual の場合と同様、アクセラレータは、メニューが表示されている場合でも表示されていない場合でも、メニューコマンドを即座に実行します。

1. 「Exit」ボタンを選択します (最初のメニューの下の 3 番目のプッシュボタン)。

2. 「アクセラレータ」ボックスをダブルクリックします。

3. 次の文字列を入力します: **Ctrl<Key>E**

4. 「適用」をクリックします。

   これらの手順により、アクセラレータが有効になります。インタフェースの実行時には、<Ctrl-E> は「Exit」ボタンと同じ効果を持ちます。アクセラレータの場合は正確な構文が重要です。構文エラーがある場合に適用を行おうとすると、図 3-13 に示されるエラーメッセージが表示されます。

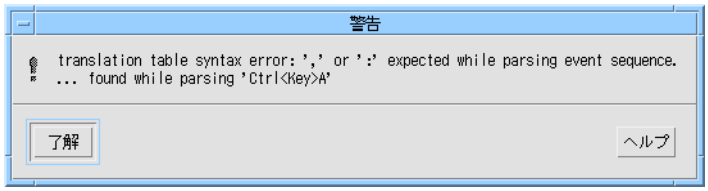


図 3-13 アクセラレータ構文エラーメッセージ

アクセラレータ構文は、トランスレーション・テーブルに使用される構文と同じです。この項目については、221 ページの「トランスレーションとアクション」で説明します。

アクセラレータテキスト

関連するリソースは、アクセラレータテキストです。このリソースは、 メニューオプションの右側に追加のテキストを表示して、アクセラレータが何であるかを示します。アクセラレータテキストは、ただ表示を行うだけであるため、特定の構文を使用する必要はありません。「Control+E」、「Ctrl-E」および「^E」といった形式が一般に使用されます。

1. 「アクセラレータテキスト」の右側のテキストボックスの中をダブルクリックします。
2. 次のテキスト文字列を入力します。

   Control+E
3. 「適用」をクリックします。

左のメニューをプルダウンすると、図 3-14 に示されるように、すべてのボタンには新しいラベルが、「Exit」ボタンには指定されたアクセラレータが一緒に表示されます。

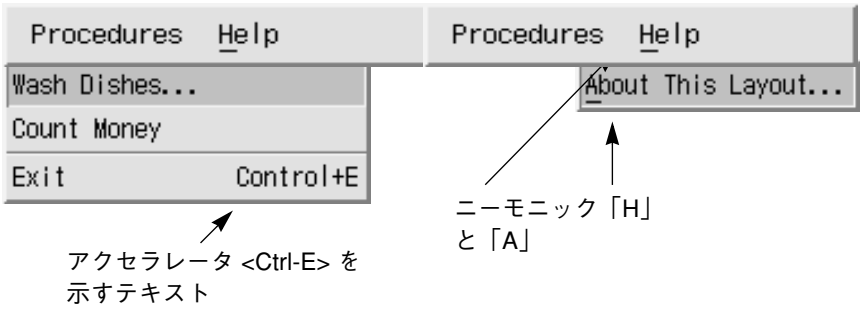


図 3-14 リソースを設定したプルダウンメニュー

4. 「閉じる」をクリックします。

ヘルプ専用ウィジェット

『Motif スタイル・ガイド』では、メニューバーの子であるカスケードボタンの 1 つをヘルプウィジェットとして指定することを推奨しています。ヘルプウィジェットは、常にメニューバーの右端に表示します。

1. メニューバーをダブルクリックして、リソースパネルを表示します。

   メニューバーはローカラムの一種なので、ローカラムのリソースが適用されます。グレー表示されていないリソースが、メニューバーに適用されます。

2. 「表示」が選択されていない場合は、「表示」を選択します。

   ヘルプウィジェットを指定するには、次のようにします。

3. 「ヘルプ・ウィジェット」フィールドをダブルクリックして、次のように入力します。

   help_cascade

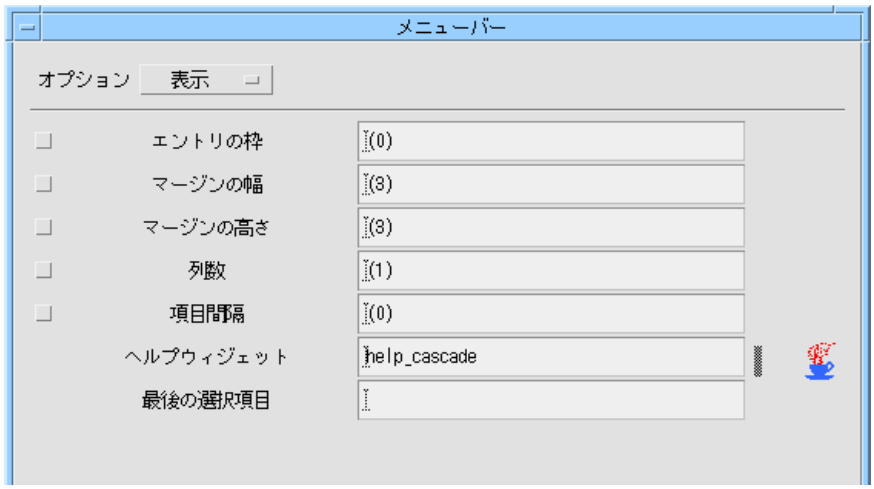


図 3-15 メニューバーの表示リソース

4. 「適用」をクリックします。

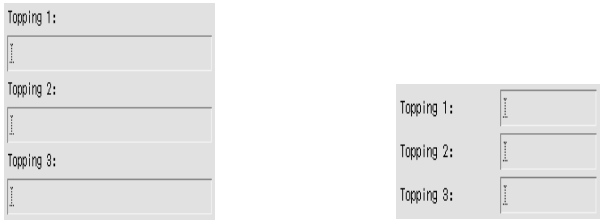
メニューバーにあるカスケードボタンの 1 つをヘルプウィジェットにします。カスケードボタンの変数名は正確に入力しなければなりません。正確に入力を行わないと、 Sun WorkShop Visual はその入力を受け付けません。

ダイナミックディスプレイは、この変更を自動的には表示しません。
ヘルプウィジェットを指定した効果を確認するには、ダイナミックディスプレイのサイズを変更してください。

メニューバーのパネルは開いたままにしておいてください。

ローカラムリソース

ローカラムウィジェットのデフォルト設定は、垂直方向に 1 列の配置です。そのリソースを設定することにより、デフォルトの状態から図 3-16 に示すような 3 個の水平な行の配置に変更することができます。



デフォルトリソースを　　設定後

図 3-16　リソース設定前と設定後のローカラム配列

図 3-16 に示すような配列にするには、テキストフィールドのサイズを設定する必要があります。

1. 階層の 3 個のテキストフィールド・ウィジェットを選択します。

   これを行うには、ウィジェットを囲むように矩形をドラッグするか、ウィジェットを 1 つ選択してから、Shift キーを押したまま残りの 2 つを選択します。

   テキストフィールドウィジェットを選択すると、画面に表示されているローカラムリソースパネルは無効になります。「適用」ボタンはグレー表示され、テキストフィールドへ入力しようとすると、ビープ音が鳴ります。このパネルはローカラムウィジェットと同じ基底クラスを持つウィジェットが選択された場合に、再び有効になります。

2. テキストフィールドウィジェットのリソースパネルを表示し、「表示」ページを選択します。

   テキストフィールドウィジェットの 1 つをダブルクリックするか、「ウィジェット」メニューから「リソース」を選択するか、ツールバー上の「リソース」アイコンを押して、リソースパネルを呼び出します。

   テキストフィールドウィジェットは、テキストウィジェットの変形であり、多くのリソースがテキストウィジェットと共通しています。ただし、テキストウィジェットが複数行のテキストを含むことができるのに対し、テキストフィールドウィジェットは単一行だけを含むという点が異なっています。これらのクラスは対応するガジェットを持たないため、このリソースパネルにはウィジェット/ガジェットのトグルがありません。その代わりに、ウィジェットをテキストフィールドかテキストに変更するためのトグルがあります。このトグルを使用すると、階層またはリソース設定に影響することなく、あるテキストウィジェットの変形から別のものへ変更することができます。

図 3-17 テキストフィールドリソースパネルの「表示」ページ

テキストフィールド・ボックスの幅を狭くする必要があります。テキストフィールド・ボックスのサイズは、テキストフィールドの「桁数」リソースおよび親であるローカラムが強制する規則の 2 種類の要因によって決定されます。まず、「桁数」リソースを小さな値に設定します。

3. 「桁数」ボックスをダブルクリックします。
4. 次のように入力します。

   8

5. 「適用」をクリックします。

   テキストフィールドの幅が狭くなります。

テキストフィールドまたはテキストウィジェットの「桁数」リソースは、ボックスのサイズに影響するだけで、ユーザーが入力可能な文字数には影響しません。入力を特定の文字数に制限する場合は、「最大長」がデフォルトで非常に大きな数に設定されているのでこれを変更します。

ローカラムリソースパネル

3 個の行を水平に配置するためには、ローカラムのリソースを設定する必要があります。

1. 構成領域内で、ローカラムウィジェットを選択します。

   メニューバーのリソースパネルをすでに画面上に表示している場合は、そのパネルが再度有効になります。ただし、リソースパネルのタイトルは「ローカラム」に変更されます。画面上にパネルが表示されていない場合は、ローカラムをダブルクリックして表示してください。

2. 「設定」ページを選択します。

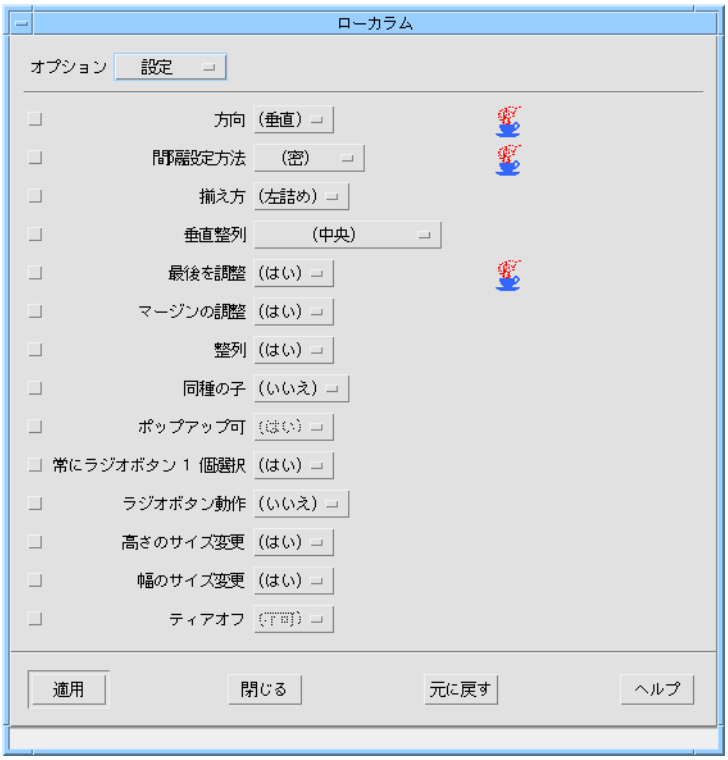


図 3-18 ローカラムリソースパネルの「設定」ページ

図 3-18 に示される「設定」ページは、指定可能な値が限られているリソースを表示します。右側の列にはテキストフィールドの代わりにオプションメニューがあります。

3. 「方向」オプションメニューで、「水平」を選択します。

4. 「間隔設定方法」オプションメニューで、「列」を選択します。

   「水平」方向は、ローカラムを列ではなく行に配置します。 ローカラムに複数の行または列を持たせる場合には、「列」間隔設定方法を選択する必要があります。これらの 2 つのリソースの効果を確認するには次のようにします。

5. 「適用」をクリックします。

   この時点で、ローカラムはすべて同じサイズのセルを持つ水平な行で表示されます。「配置方向」が「水平」である場合、行および列の意味は逆になります。したがって、3 個の行を作成するためには、「列数」リソースを設定する必要があります。

6. 「表示」ページを選択します。

7. 「列数」ボックスをダブルクリックし、次のように入力します。
   3

8. 「適用」をクリックします。

   結果は図 3-19 のようになります。

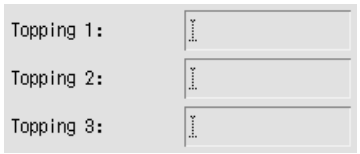


図 3-19 水平方向のダイアログのローカラム部分

シェルのリソース

シェルウィジェットは、第 1 にアプリケーションと X ウィンドウシステムとの間のインタフェースとして存在します。その動作はおもにウィンドウマネージャと、含まれるウィジェットにより制御されます。ただし、シェルウィジェットには、いくつかの特殊なリソースがあります。

- シェルウィジェット myFirstShell をダブルクリックしてリソースパネルを表示します。

このパネルの上部には、シェルウィジェットの型を設定するトグルスイッチがあります。ウィジェットパレットには 3 つのシェルウィジェットがありますが、トグルの設定が異なるだけで、すべて同じウィジェットです。

シェルの型

すべてのウィンドウはシェルウィジェットで始まりますが、ウィンドウおよびシェルウィジェットにはいくつかの種類があります。Sun WorkShop Visual には、以下に示すシェルウィジェットがあります (図 3-20 参照)。

- アプリケーションシェル
  アプリケーションのメインウィンドウ。アプリケーションを実行すると最初に表示されます。
- 最上位シェル
  アプリケーションシェル以外のウィンドウ。 アプリケーションシェルがアイコン化された場合でも表示され、個別にアイコン化することができます。
- ダイアログシェル
  アプリケーションシェルと切り離してアイコン化することはできないウィンドウ。

図 3-20 パレットのシェルウィジェット

注 - 上記の動作は、`mwm` を使用した場合のものです。`twm` の場合は、シェル動作は同じですが外観が異なります。これは、`twm` がダイアログシェルを擬似アイコンに変えてサイズを小さくするためです。擬似アイコンは、ディスプレイを整理して見やすくするためのものです。内部では、本物のアイコンと擬似アイコンは区別されており、画面上での外観は異なっています。

アプリケーションシェルを閉じると、デザイン内のすべてのウィンドウが閉じられます。

Sun WorkShop Visual インタフェースにおけるシェルの型の例

種々の型のシェルの例として、Sun WorkShop Visual インタフェースを見てみましょう。ウィジェットパレットおよび構成領域を持つメインウィンドウがアプリケーションシェルです。ダイアログ、リソースパネル、配置エディタウィンドウはすべてダイアログシェルです。これらのウィンドウは、ウィンドウマネージャを使用して個別にアイコン化することはできません。

これらのウィンドウをディスプレイから消すためには、ウィンドウマネージャを使用するか、「閉じる」ボタンをクリックしてウィンドウを閉じます。「閉じる」ボタンは、「活性化 (activate)」コールバックを使用して内部的にウィンドウを閉じます。メインウィンドウがアイコン化される場合、開いていたすべてのダイアログシェルは消去されます。メインウィンドウが復元されると、 ダイアログシェルも再び表示されます。

「ヘルプ」メニューの「ウィジェットパレット」で表示される「ウィジェットパレットの情報」パネルは、最上位シェルの例です。このパネルは Sun WorkShop Visual の起動時には自動的に表示されませんが、いったん表示すると、個別にアイコン化することができます。「ウィジェットパレットの情報」パネルのポップアップ・サブウィンドウは、その他すべてのダイアログシェルと同様にメインアプリケーションシェルの子であるため、「ウィジェットパレットの情報」パネルを使用して閉じたり、アイコン化することはできません。

ダイナミックディスプレイ内のシェルの型

シェルの型は、ダイナミックディスプレイ内には反映されません。ダイナミックディスプレイに対して作成されるすべてのウィンドウは、実際にはダイアログシェルであるため、個別にアイコン化することはできません。Sun WorkShop Visual の構成を変更して、ダイアログシェルではなく、最上位シェルを使用することができます。この操作については、付録 D「アプリケーションのデフォルト」を参照してください。各ウィンドウに対してユーザーが指定したシェルの型は、実行時に生成コードによって作成されます。

アプリケーションシェルの必要条件

各デザインには、1 つ以上のアプリケーションシェルが必要です。デザイン内にアプリケーションシェルがない場合には、そのアプリケーションではウィンドウは一切表示されません。Sun WorkShop Visual は、アプリケーションシェルが存在しない場合にはコード生成時に警告メッセージを発します。

1 つのデザインには、2 個以上のアプリケーションシェルを持たせることができます。この場合には、Sun WorkShop Visual が生成する `main()` プログラムによってすべてのアプリケーションシェルが作成されますが、その中の 1 つだけが表示されます。

Sun WorkShop Visual は、どのシェルを最初に表示する必要があるかを判断することはできません。デザイン内に 2 個以上のアプリケーションシェルを持たせる場合には、独自の `main()` プログラムを作成するか、または適切なアプリケーションシェルを起動するように生成された `main()` プログラムを編集する必要があります。同様の効果を確実に実現するためには、最初のウィンドウにアプリケーションシェルを 1 つだけ持たせ、その他の一次ウィンドウに最上位シェルを使用します。そしてコールバックまたはリンクを使用して最初のウィンドウを除くすべてのウィンドウを表示するようにします。複数のアプリケーションシェルの使用はお勧めできません。

それぞれのアプリケーションは、最低 1 個 (通常は 1 個のみ) のアプリケーションシェルを必要とします。アプリケーションシェルはメインウィンドウとして、さらにアプリケーションと X ウィンドウシステム間のインタフェースとして機能します。

従属するウィンドウは、ダイアログシェルまたは最上位シェルのどちらかにすることができます。この相違点については、82 ページの「シェルの型」を参照してください。

メインシェルのリソース設定

デザインのメインシェルである *myFirstShell* はアプリケーションシェルになっているので、タイトルを変更するだけです。

1. 表示されていない場合は、myFirstShell のリソースパネルを表示します。
2. 「表示」ページを選択します。
3. 「タイトル」ボックスをダブルクリックし、次のように入力します。
   `Ice Cream Shop`

図 3-21　ダイアログ (シェル) リソースパネルの「表示」ページ

4. 「適用」をクリックします。
5. 「閉じる」をクリックします。

この段階で、配置は図 3-22 のようになります。

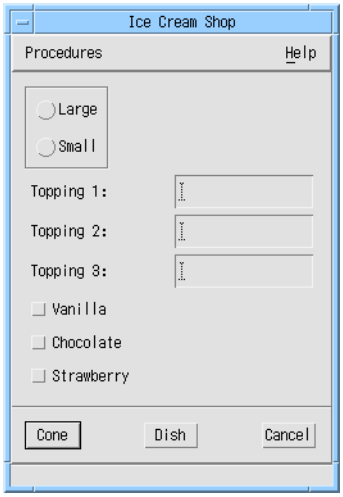


図 3-22 現段階での学習用インタフェース

2 次ウィンドウのリソースの設定

ここまでで作成した学習用アプリケーションには、ヘルプウィンドウ用に別のシェルがあります。このシェルは、ダイアログシェルです。このウィンドウのタイトルを変更する必要があります。

リソースパネルを表示するには、次のようにします。

1. ヘルプウィンドウのシェルをダブルクリックします。
2. 「表示」ページを選択します。
3. 「タイトル」ボックスをダブルクリックし、次のように入力します。
   `Help`
4. 「適用」をクリックします。
5. 「閉じる」をクリックします。

リソースパネル内での操作

Motif は非常に多くのリソースを持っているため、Sun WorkShop Visual は複数のページで構成されるリソースパネルを使用して画面上にリソースを表示します。リソースパネルの構造が理解できたら、ウィジェットクラスおよびページ別のリソースの一覧が記載されている、851 ページの第 27 章「ウィジェット・リファレンス」を参照することをお勧めします。

ウィジェットクラスのリソースパネルで使用できないリソースは、88 ページの「コアリソースパネル」にあります。

設定

通常は、複数の選択肢があるリソース、つまり設定値が限定されているリソースは、「設定」のページにあり、オプションメニューを使用して設定することができます。その他の、新しい値の入力を必要とするリソースや、別のダイアログを呼び出して新しい値を設定するリソースは、すべて別のページに分かれています。「設定」ページの場合を除いては、リソースはおおまかな項目別に編成されています。

表示、マージン

ウィジェットの外観に影響するリソースは、通常は「表示」ページにあります。テキスト、色、フォント、寸法などがそうです。コアリソースパネルにも、ウィジェットの寸法と位置に影響するリソースがいくつあります。

ラベルおよびラベルから派生するウィジェットは、表示リソースを多く持っているため、1 ページ以内に収めることができません。ラベルリソースパネルは、これらのリソースを色、フォント、テキスト、ピックスマップリソースを含む「表示」ページと、サイズ、マージン幅リソースを含む「マージン」ページに分割しています。

キーボード

「キーボード」ページでは、キーボード・ニーモニックおよびアクセラレータを持つことができるウィジェットにそれらの設定を行うことができます。

コアリソースパネル

Sun WorkShop Visual は、これらの広域なスーパークラスのリソースへのアクセスを可能にするコアリソースパネルという単一のリソースパネルを用意しています。特定のウィジェットに対してコアリソースパネルを表示するには、次のようにします。

1. ウィジェットを選択します。
2. 「ウィジェット」メニューをプルダウンし、「コアリソース」を選択します。

図 3-23 に、コアリソースパネルのページを示します。

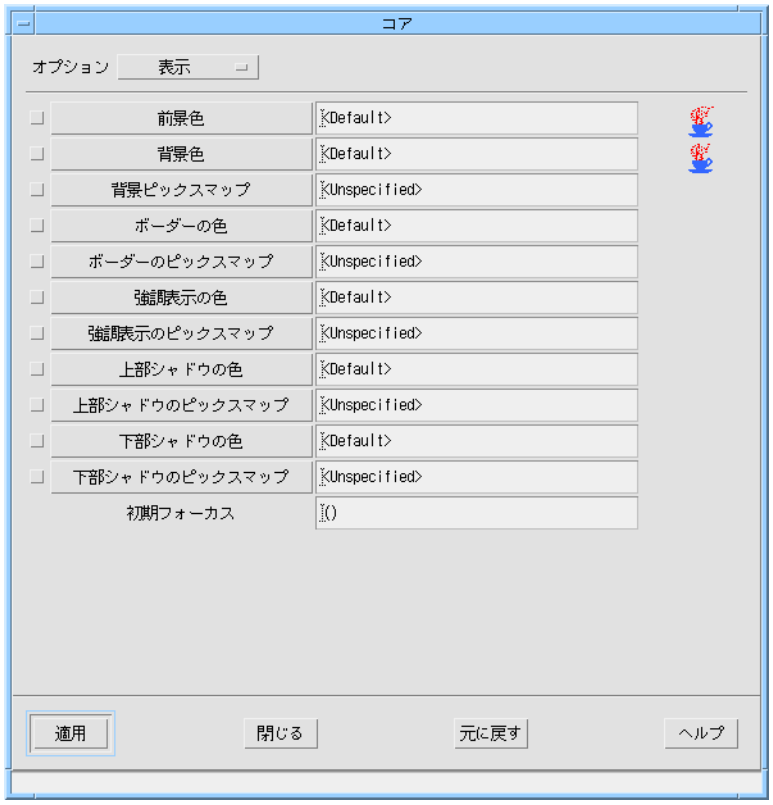


図 3-23 コアリソースパネルの「表示」ページ

コアリソースパネルの「表示」ページ

コアリソースパネルの「表示」ページには、基本の色とピックスマップリソースがあります。たとえば、ウィジェットの前景、背景、強調表示、シャドウの色を設定することができます。これは、第 5 章「その他のエディタ」で実際に行います。

コアリソースパネルの「寸法」ページ

「寸法」ページには、ウィジェットの画面上の位置やサイズに影響するリソースがあります。クラス固有の寸法リソースは、コアパネルの設定を書き換える場合もあります。 テキストフィールドの「シャドウの厚さ」またはプッシュボタンの「強調表示の厚さ」を設定して、その効果を試してみると良いでしょう。

コアリソースパネルの「設定」ページ

「設定」ページでは、ほとんどのウィジェットクラスに適用する設定を行うことができます。これらの設定には複数の選択肢が設けられています。また、「マネージ時にマップ」設定を変更することもできます。この設定は「マネージトグル」で説明するように、マネージトグルと連動しています。

コアリソースパネルの「コード生成」ページ

「コード生成」ページでは、コード生成に関して大幅に制御を行うことができます。たとえば、特定のウィジェットを静的、局所的、大域的に指定することができます。ウィジェットの呼び出しを変更する理由については、208 ページの「コールバックのウィジェットへのアクセス」を参照してください。C++ を使用している場合には、ウィジェットを限定公開、非公開、公開として指定することもできます。C++ での呼び出しについては、324 ページの「ウィジェットメンバーのアクセス制御」で説明しています。さらに、「コード生成」ページを使用すると、独自の派生 C++ クラスを生成することができます。詳細は、295 ページの「C++ クラス」で説明します。

マネージトグル

マネージトグルは、「コード生成」ページにも表示されます。デフォルトでは、すべてのウィジェットがマネージとして生成されます。ただし、ダイアログシェルの子ではない選択ボックス内の「適用」ボタンはマネージされません。マネージされている状態から変更するには、コアリソースパネルの「コード生成」ページにある「マネー

ジ」トグルを使用します。
これによって通常は、生成されたコードからウィジェットをマネージするコードが省略されます。ただし、コンポジットウィジェットの構成要素であるウィジェットまたはガジェットの場合は、トグルがオフのときは生成されたコードはウィジェットを明示的にアンマネージ (マネージ解除) します。これは、ツールキットがこれらのウィジェットを常にマネージとして作成するためです。選択ボックスの「適用」ボタンの場合は、トグルがオンのときはボタンを明示的にマネージするコードが生成されます。ウィジェットの操作方法については、210 ページの「ウィジェットの操作」を参照してください。

「リソースの結合にインクルード」トグル

「リソースの結合にインクルード」というラベルが付けられたトグルは、選択されたウィジェットのリソースの生成を参照します。詳細は、104 ページの「密な結合」で説明します。

コアリソースパネルの「ドロップサイト」ページ

「ドロップサイト」ページでドロップサイトを指定することができます。
ドロップサイトとは、Motif 1.2 で導入されたドラッグ & ドロップ機能を使用して、別のウィジェットからある種のデータを受け取るために用意されたウィジェットです。ドラッグ & ドロップ機能を使用すると、マウスでデータを選択してドラッグすることによって、ウィジェット間でデータを受け渡すことができます。

ドラッグの初期化は、通常はコールバックまたはアクション関数内から行われるダイナミック関数によって行われます。Sun WorkShop Visual は、ドロップの受け取りに対しては単純なサポートを行なっていますが、ドラッグのソースに対してはサポートしていません。

ウィジェットをドロップサイトとして設定する方法については、218 ページの「ドラッグ & ドロップ」を参照してください。

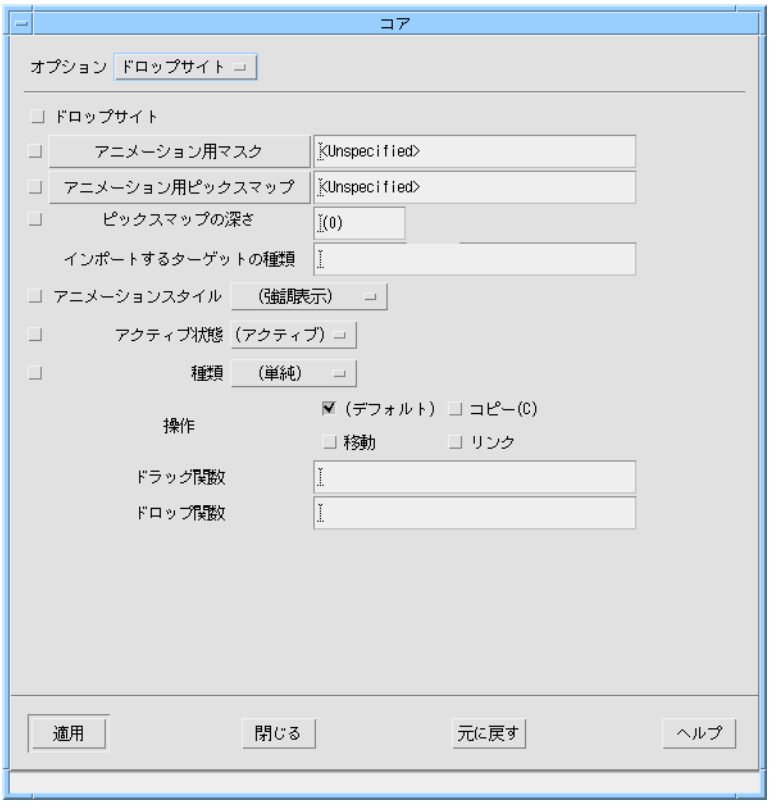


図 3-24 「ドロップサイト」ページ

コンストレイント・パネル

Motif には、コンストレイント・ウィジェットと呼ばれる 2 種類のウィジェットクラス (区画ウィンドウおよびフォーム) があります。これらのクラスのウィジェットは、コンストレイント・リソースと呼ばれる一連の特殊なリソースを持っています。これらのリソースは、子のサイズや位置を制御します。コンストレイント・ウィジェットは、その子に対してそれぞれ別々のコンストレイント・セットを持っています。Sun WorkShop Visual では、それらのリソースを子のリソースであるかのように設定することができます。

コンストレイント・ウィジェットの子に対するコンストレイント・リソースを表示するには、次のようにします。

1. コンストレイントウィジェットの子 (例：フォームの子) を選択します。
2. ウィジェットメニューをプルダウンし、「コンストレイント」を選択します。

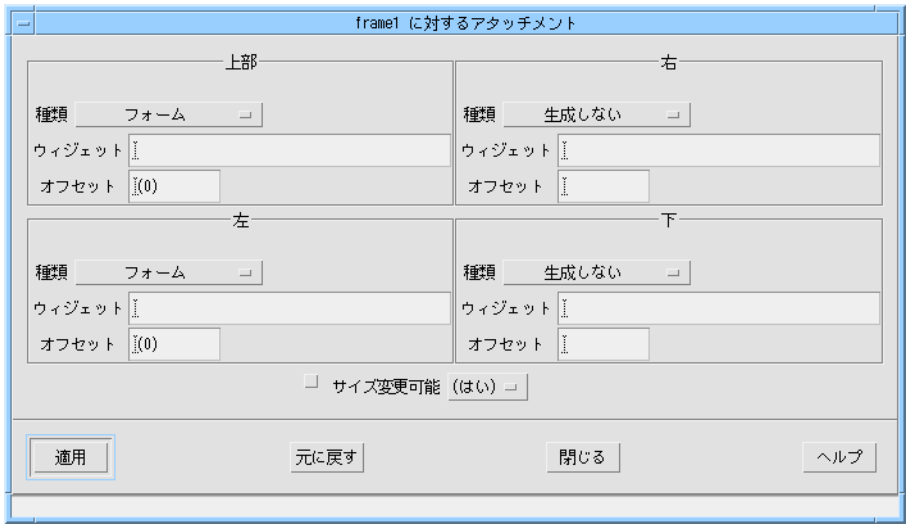


図 3-25 コンストレイント・パネル

図 3-25 は、学習用配置のフォームの子であるフレームに対するデフォルトのコンストレイント・リソースパネルです。このパネルでは、フレームの上部がフォームの上部に、左端がフォームの左端に、0 ピクセルのオフセットで接続 (アタッチ) されていることを示しています。このため、フレームはフォームの左上に配置されています。フレームの下部および右端は制約されていません。

フォームがその子に強制するコンストレイント・リソースは、複雑に相互作用するため、第 4 章「配置エディタ」で説明する配置エディタを使用して設定を行うと良いでしょう。コンストレイント・パネルは、おもに配置エディタの補助として、すでに設定されているコンストレイントを表示するのに便利です。上級ユーザーは、パネルそのものに値を設定することもできます。このパネルへの値の設定は、その他のリソースパネルの場合と同様に、新しい値を入力して「適用」をクリックすることにより実行できます。

区画ウィンドウ・ウィジェットの子に対するコンストレイント・リソースパネルは、異なる値の集合を表示します。このパネルについては、 875 ページの「区画ウィンドウ (PanedWindow)」で説明します。

デフォルトリソース設定

Sun WorkShop Visual はデフォルトのリソース設定を括弧内に表示します。ユーザーがインタフェースを構築する場合の値と同じであっても、デフォルト設定は、明示的に設定する値とは異なります。デフォルト設定は、生成されたコードまたは X リソースファイルに追加されないという点が異なっています。インタフェースでデフォルト設定を使用し、そのインタフェース設計時に使用したものを異なるマシン上で実行する場合には、実行しているマシンに対しての Motif デフォルトが使用されます。

プッシュボタンのラベル、またはローカラムの列数などの多くのリソースでは、移植性の問題が生じることはまずありません。寸法や色などのその他のリソースは、マシンによって異なります。インタフェースに移植性を持たせるためには、そのようなリソースにデフォルト値を使用するか、あるいはコード生成時にそれらのリソースを X リソースファイルに入れて、各マシンに対して編集できるようにする必要があります。これについては、251 ページの「X リソースファイルの設定」で説明します。

図 3-26 に、これまでの節で明示的に設定されたリソースを持つリソースパネルを示します。明示的な設定は、変更傍線でマークされています。

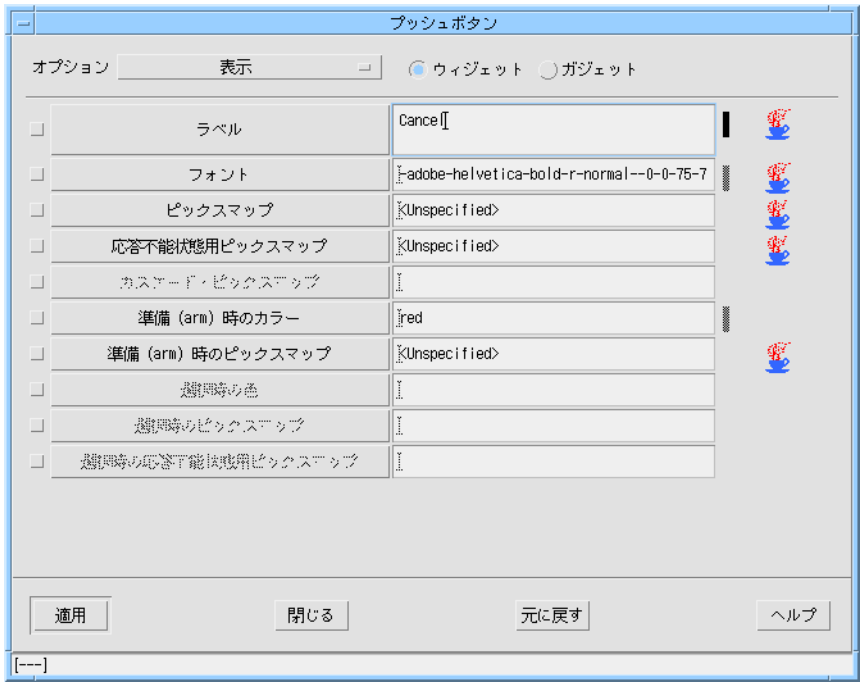


図 3-26 明示的に設定されたリソースを持つリソースパネル

明示的に設定したリソースをデフォルト設定に戻すには、次のようにします。

1. リソースパネル上のリソーステキストフィールドにあるすべてのテキストを削除します。
2. 「適用」をクリックします。

「設定」のページで、あるリソースに対してデフォルト値を選択するには、次のようにします。

3. オプションメニューから、括弧内に表示されているオプションを選択します。
4. 「適用」をクリックします。

注 - この場合、現段階のダイナミックディスプレイで使用されている値、つまり最後にリソースに対して割り当てられた値がデフォルトの値になります。アプリケーションを Sun WorkShop Visual 内からではなく単独で実行する場合は、システムのデフォルト値が使用されます。Sun WorkShop Visual 内でシステムのデフォルトを確認するには、対象となるウィジェットをリセットする必要があります。

リセットコマンド

なんらかのリソースを設定する場合には、Sun WorkShop Visual はダイナミックディスプレイで選択されたウィジェットにその値を適用しようとします。ダイナミックディスプレイに表示されるものは、ウィジェットのインスタンスの集まりであることに注意してください。Sun WorkShop Visual はただウィジェットの絵を描画するわけではなく、実際に Motif 関数呼び出しを行なってウィジェットの作成を行います。Sun WorkShop Visual がリソースを設定する場合は、適切な Motif 関数呼び出しを作成してそのウィジェットに対するリソースの値を設定します。

通常、値を設定した結果は、最初にその値を使用してウィジェットを作成した場合と同じです。しかし、これが必ずしも当てはまるわけではありません。Sun WorkShop Visual のウィジェットメニューには、選択されたウィジェットとその子を破壊して、最後に適用されたリソース設定を使用してウィジェットおよびその子を作成し直す「リセット」(<Ctrl-T>) というコマンドがあります。配置が意図した通りに表示されない場合は、「リセット」コマンドを使用してください。以下の手順では、「リセット」が必要とされる例を示します。

1. 階層で「ヘルプ」カスケードボタンウィジェットをダブルクリックして、リソースパネルを表示します。
2. 「キーボード」ページを選択します。
3. 「ニーモニック」テキストフィールドからすべてのテキストを削除して、以前に設定したニーモニックを削除します。
4. 「適用」をクリックします。

この時点で、「ニーモニック」テキストフィールドは <デフォルト> 設定に戻ります。しかし、ダイナミックディスプレイの「ヘルプ」ボタンには、もう存在していないニーモニックを表わす下線の付いた「H」が表示されたままです。表示を更新するためには次のようにします。

5. ウィジェットメニューから「リセット」を選択します。

   リセットは、選択されたウィジェットとその子にのみ影響します。階層の下の方のウィジェットをリセットする場合、ダイナミックディスプレイ上の位置が不正確なまま残されることがあります。たくさんのリソースを設定する場合には、シェルをリセットして、リソースの設定による結果が確実に表示されるようにすることをお勧めします。

   「リセット」コマンドは、フォーム・ウィジェットおよびそのアタッチメントリソースを使用している場合に特に役に立ちます。これについては、第 4 章「配置エディタ」で説明します。

6. ニーモニックを元の状態に戻します。

拒否されるリソース設定

Motif およびその他のウィジェットツールキットには、リソースの有効な設定を制御する規則があります。また、Sun WorkShop Visual はシミュレーションではなくウィジェットの実際のインスタンスを使用して動作するため、ツールキットの持つ規則に合わない新しいリソース設定は拒否されてしまいます。規則には、特定のウィジェットに対しての有効な値、 ローカラムなどの親ウィジェットの必要条件、そしてデザインを構築するために使用しているマシンの必要条件が含まれています。

たとえば、大画面ワークステーションで使用するためのインタフェースを設計している場合は、寸法のリソースを大きなピクセル数に設定する必要があります。小画面マシンで設計を行なっている場合は、インタフェースが将来大画面のマシンで実行することになっていても、必要な値を設定することができません (この場合には、X リソースファイルを使用することによって必要な幅を設定することが可能です)。

Motif が新しいリソース設定を拒否する場合、以前の設定には戻さずに、デフォルトと階層内の他のリソース設定にもとづいて新しい値を計算します。この新しい値は、リソースパネルおよびダイナミックディスプレイに反映されます。

リソースの結合

本章の前半で各ウィジェットに対してリソースを設定する方法を説明しました。デザインで設定したリソースすべてを含むリソースファイルを Sun WorkShop Visual がどのように生成するかについては、251 ページの「X リソースファイルの設定」で説明します。

リソースとウィジェットの結び付きは通常、結合と呼ばれます。各リソースには、明示的に設定され、ソースファイル内に生成するようユーザーが要求したリソース設定が 1 行生成されます。リソースファイルの形式については、274 ページの「リソースファイルの構文」を参照してください。どのリソースをリソースファイル内に生成するかを指定する方法については、254 ページの「コード生成オプション」を参照してください。

Sun WorkShop Visual では、リソースは次の 2 つの方法で設定します。

1. リソースパネルで各ウィジェットに対するリソースを設定

2. 緩い結合を使用

どちらの方法を選択するかは、リソースを使用したいウィジェットの数によって決まります。大規模なデザインでは、特定クラスのすべてのウィジェットを同じ外観にしたい場合があります (すべてのボタンを緑色にするなど)。緩い結合を使用して、このような設定を行うことができます。各リソースに適用されるリソースは、リソースパネルを使用して設定する必要があります。

密な結合は、アプリケーションのリソースファイルで矛盾が発生することを防ぐために使用します。これは、104 ページの「密な結合」で詳細に説明します。

緩い結合

緩い結合を設定すると、明示的に設定しているリソースがない場合に使用される、デフォルトリソースを指定することができます。緩い結合ダイアログは柔軟性に富んでいます。「緩い結合」ダイアログでは、ウィジェット名、クラス名、さらには記述と一致するあらゆるウィジェットが適用されることを示すワイルドカードを使用して、

ウィジェットを参照することができます。また、結合を行う「汎用性」を指定することもできます。この汎用性については、以下の例で詳細に説明します。緩い結合は、次のような場合に便利です。

- デザイン内のすべてのボタンに同じ背景色を持たせたい場合
- あるローカラム内のすべてのボタンを同じフォントで表示したい場合
- アプリケーション内のすべての「ヘルプ」ボタンまたは「了解」ボタンに同じラベルを共有させたい場合

上記を実現するには、緩い結合を設定します。次に簡単な例を紹介します。

緩い結合の例

1. 図 3-27 に示すように、各種シェル、フォーム、もう 1 つのフォーム、3 個のプッシュボタンを含むウィジェット階層を作成します。

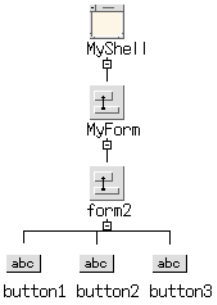


図 3-27 緩い結合の階層

2. 図 3-27 に示すように、シェルに「MyShell」、フォームの 1 つに「MyForm」と名前を付けます。

   リソースでは、ウィジェット名が使用されます。ただし、変数名を設定すると、ウィジェット名も同じ名前に設定されます。

3. シェル、「MyForm」と名前を付けたフォーム、最初のボタンを選択します。

   ウィジェットを 1 つ選択して、Shift キーを押したまま残りの 2 つを選択します。

4. 「ウィジェット」メニューから「緩い結合」を選択します。

   「緩い結合」ダイアログが表示されます。

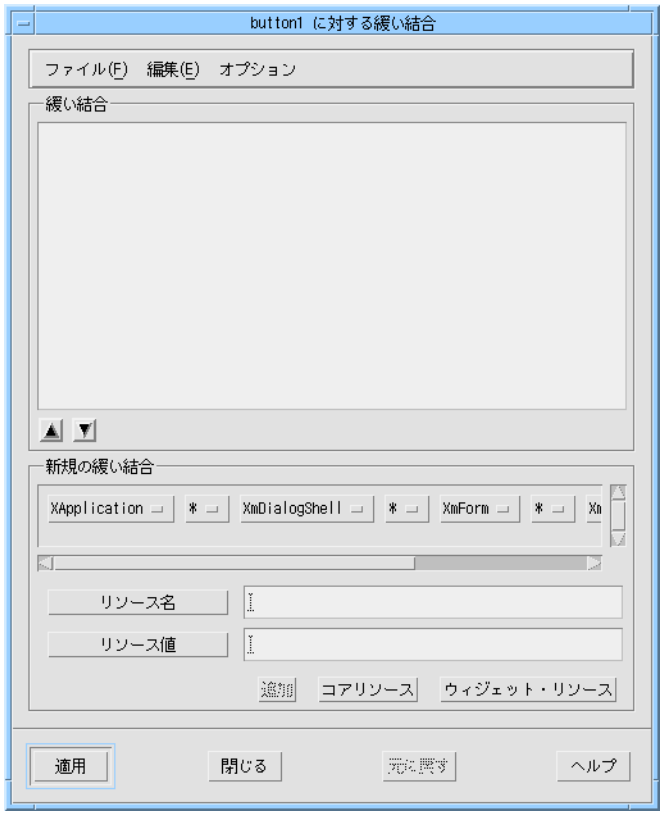


図 3-28　「緩い結合」ダイアログ

「緩い結合」ダイアログ

図 3-28 に示す「緩い結合」ダイアログには、次のような領域があります。

メニューバー

このメニューバーには、「ファイル」、「編集」、「オプション」の 3 つのメニューがあります。「ファイル」メニューには、外部リソースファイルからの結合の読み込み、マージ、継承を行うためのコマンドがあります。これらの操作については、102 ページの「外部リソースファイルからのリソース」で詳細に説明します。「編集」メニューを使用すると、結合をカット、コピー、ペーストに加えて、削除することもできます。リスト内には生成された順序で結合が表示されるため、カット機能およびペースト機能は重要です。また、ファイルは X ツールキットによって順に読み取られ

るため、リソースファイル内のリソースの順序は重要です。互いに矛盾するリソースがある場合は、後で生成されたリソースが以前に生成されたリソースを上書きします。「オプション」メニューには、「継承された結合を使用」というコマンドがあります。このコマンドで、「緩い結合」リストに表示される継承された結合を使用するかどうかを選択できます。

既存の結合

ウィンドウの最上部には、現在定義されている緩い結合のリストがあります。リストの下に上下の矢印ボタンがあります。これらのボタンを使用してリスト内の結合の順序を変更することができます。このリストに結合が表示されている順序は、リソースファイル内に生成された順序です。

新規の緩い結合

既存の結合リストの下に、オプションメニューを含むスクロールウィンドウがあります。このオプションメニューでは、ウィジェット階層内で選択したウィジェットの結合が選択されています。

リソース名とリソース値

結合の表示の下に、テキストボックスが 2 つあります。1 つはリソースの名前の入力に、もう 1 つはその値の入力に使用します。

リソースパネル

ダイアログの最下部に、コアリソースパネルおよびウィジェットリソースパネルを表示するボタンがあります。リソースパネルは、ウィジェット階層内で選択した最下層のウィジェット (リーフウィジェットとも呼ばれます) のリソースを表示します。また、現在定義されている結合のリストに結合を追加する「追加」ボタンもあります。

結合の作成

前述の例を使用して、説明を続けます。

1. 「緩い結合」ダイアログ内のオプションメニューを、ドットがそれぞれアスタリスクで置換されるように変更します。

   これは、シェルおよび 1 つのフォームを名前で参照し、ボタンをその Motif クラス名で参照する結合を示しています。つまり、その結合が命名したフォームの下にあるボタンをすべて参照することを意味します。「新規の緩い結合」領域のオプションで、各ウィジェットの間にアスタリスク (*) を選択しています。これは、命名した各ウィジェットの間に別のウィジェットが存在し得ることを意味しています。この例では、他のフォームも含めることができます。

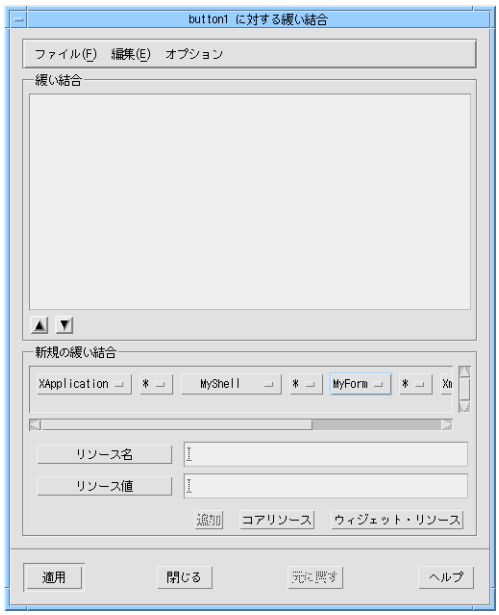


図 3-29 緩い結合のウィジェットオプション

2. 「ウィジェット・リソース」を押します。

   ボタンウィジェットのリソースパネルが表示されます。

3. ラベルリソースを「Bound」に設定し、「適用」を押して、リソースパネルを閉じます。

   「リソース名」テキストボックスには、X ウィンドウが認識するラベルリソースの名前である「labelString」という文字列が入っています。「リソース値」テキストボックスには、リソースパネルに入力した値である「Bound」という文字列が入っています。次の行が緩い結合リストに追加されます。

```
XApplication*MyShell*MyForm.XmPushButton.labelString: Bound
```

4. 「適用」を押します。

次回リソースファイルを生成する際には、この行が生成されたリソースファイルに含まれます。この行の意味は、次のとおりです。

「XApplication」クラスのアプリケーション内にある「MyShell」というシェルの子孫 (間に存在するウィジェットの数に関係なく) である「MyForm」というフォームの (間に存在するウィジェットの数に関係なく)子孫であるプッシュボタンウィジェットは、すべて、そのラベルが「Bound」に設定されます。

外部リソースファイルからのリソース

「緩い結合」ダイアログの「ファイル」メニューには、以下の 3 つの項目があります。

- 結合を読み込む
- 結合をマージ
- 結合を継承

「結合を読み込む」または「結合をマージ」を選択すると、リソースファイル名の入力を要求するファイル選択ボックスが表示されます。「ファイル」メニューから「結合を継承」を選択すると、「継承された緩い結合」ダイアログが表示されます。

「継承された緩い結合」ダイアログ

「継承された緩い結合」ダイアログには、テキストボックスとボタンの 2 つの領域があります。これらの領域を使用して、最後に撰択したファイルから結合および結合のリストを継承することができます。この結合リストは、読み取り専用のリストです。「リソースファイル」というラベルの付いたボタンを押すとファイル選択ボックスが表示され、リソースファイルを指定することができます。

このダイアログに表示されている結合を継承するには、「オプション」メニューの「継承された結合を使用」トグルをオンにします。

次回リソースファイルを生成する際に「オプション」メニューのトグルボタンが設定されていると、命名したリソースファイルへの参照が生成されます。これによって、実際に「継承された緩い結合」ダイアログに一覧されているすべてのリソースが継承されます。リソースを継承したくない場合は、「継承された結合を使用」トグルをオフにします。トグルボタンをオフにすると、リソース結合は継承されません。

結合の調整

98 ページの「緩い結合の例」で紹介している例は、単純なものです。「緩い結合」ダイアログでリソース結合を変更する方法は他にもあります。現在定義されている結合内の各要素は、図 3-30 に示すように対応するオプションメニューを使用して変更することができます。

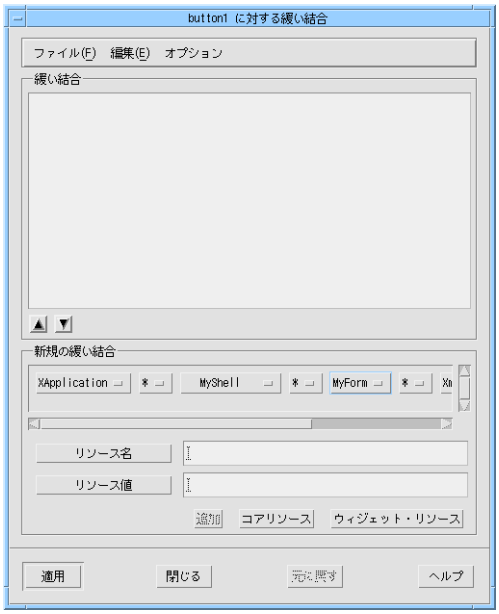


図 3-30 対応するボタンを持つ現在の結合

これらのオプションメニューを使用して、結合を調整することができます。調整方法には、次の 3 種類があります。

- ワイルドカードの変更
- ウィジェットへの参照の変更
- リソースの変更

これらについて、以下で説明します。

ワイルドカード

各ウィジェット名の間に、ワイルドカード文字があります。これは、ピリオド (.) またはアスタリスク (*) です。ピリオドは、右側のウィジェットが左側のウィジェットの直系の子孫であるという意味です。アスタリスクは、左側と右側のウィジェットの間に命名していない別のウィジェットがいくつかあり得るという意味です。

ウィジェット・リファレンス

ウィジェットを参照する方法は 3 つあります。Motif クラス名 (ボタンの場合は XmPushButton、フォームの場合は XmForm など)、指定したウィジェット名、疑問符文字 (?) のうちいずれかを選択することができます。疑問符は、記述中のこの位置に必ずなんらかのウィジェットを指定しなければならないことを意味するワイルドカードです。

リソース

現在定義されている結合の最後の項目として、「リソース名」および「リソース値」があります。この結合に対してリソースを複数設定している場合は、設定されているリソースは、対応するオプションメニューにリストされます。緩い結合を設定するリソースは、いくつでも構いません。

注 - ウィジェットの名前が変更された場合は、そのウィジェットをリセットしない限り、緩い結合は名前の変更されたウィジェットには適用されません。

密な結合

複数のウィジェットが同じアプリケーションクラス内に同じウィジェット名を持っている場合は、Sun WorkShop Visual が生成するデフォルトのリソース結合が曖昧なものになる可能性があります。密な結合とは、曖昧になる可能性を減らすために、リソース結合に特別なウィジェットを命名する方法です。リソースファイル内に存在するリソース結合の構造は、274 ページの「リソースファイルの構文」で説明します。図 3-31 に示す階層に対して Sun WorkShop Visual が生成したデフォルトのリソース結合は次のとおりです。

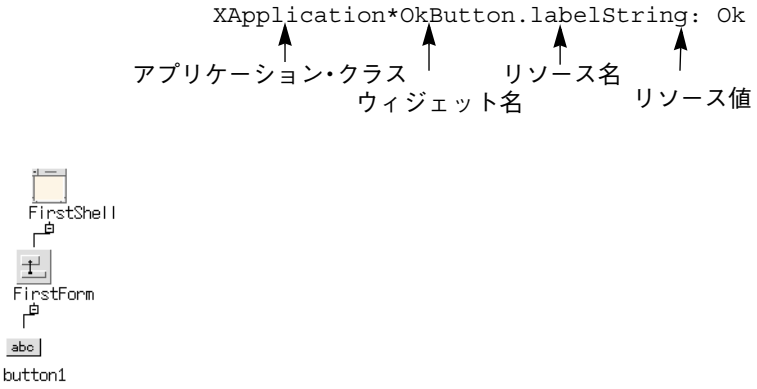


図 3-31 最初の階層

この例では、ウィジェット名が「OkButton」という、1 つのウィジェット (「リーフ」ウィジェット) だけを明示的に取り上げています。

注 - このウィジェットの変数名は設定されていません。リソースファイルでは、ウィジェット名が重要です。

アプリケーションで、同じウィジェット名を持つリーフウィジェットを複数持っている場合があります。ただし、個々のウィジェットに異なるリソース設定が必要な場合は、上述のデフォルトのリソース構文は使用しないでください。この構文では、同じ名前を持つすべてのウィジェットのリソースに対して同じ設定が適用されるため、一部のウィジェットのリソース設定が失われます。リソース結合で明示的に指定するウィジェットの数が多いほど、リソースファイルでウィジェット名に関して矛盾が発生する可能性は低くなります。

密な結合の例

図 3-31 の例を使用して、名前は同じでラベルが異なるリーフウィジェット (この場合は PushButton) を複数持つことができることを説明します。

1. 図 3-30 に示すような階層を作成します。名前は、ウィジェット名として表示されたものを使用してください。

2. パレットから別のシェル (任意の型) を選択し、図 3-32 に示すような 2 番目の階層を作成します。ボタンには「OkButton」というウィジェット名を指定し、最初の階層内のボタンと同じウィジェット名になるようにします。

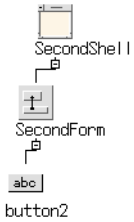


図 3-32 2 番目の階層

3. 最初の階層のボタンのラベルを次のように指定します。

   Ok

4. 2 番目の階層のボタンにラベルを次のように指定します。

   Apply

5. 最初の階層で「FirstShell」を選択し、そのコアリソースパネルを表示します。

6. 「コード生成」ページを選択します。

7. 「リソースの結合にインクルード」というラベルが付いたトグルを設定します。

   これは、ウィジェットのリソース結合を「密」にするトグルです。

8. 「適用」を押します。

   次回リソースファイルを生成する際に、密なリソース結合を持つウィジェットが次のように表示されます。

   ```
   XApplication*FirstShell*OkButton.labelString: Ok
   ```

9. 2 番目の階層に対しても同じ作業を行います。ウィジェット「SecondShell」をリソース結合に含めます。

   2 番目の階層に対する結合は、次のようになります。

   ```
   XApplication*SecondShell*OkButton.labelString: Apply
   ```

   これで、同じウィジェット名を持つ 2 つのボタンのリソースが簡単に区別できます。リソース結合にはウィジェットをいくつでも追加することができます (この例では、フォームも追加できます)。これで、結合はより密になり、曖昧さが少なくなります。

密なリソース結合の推奨

デザイン中のすべてのシェルに対して「リソースの結合にインクルード」トグルを設定することをお勧めします。こうすることで、結合がより密になり、曖昧な部分が少なくなります。

リソース生成の比較

図 3-27 に示す緩い結合の例の最初のボタンのラベルリソースを明示的に設定しても、緩い結合を設定していないとリソースファイル内に次の行が生成されます。

```
XApplication*button1.labelString:Bound
```

104 ページの「密な結合」で説明している密な結合を使用して、シェル (MyShell) に密な結合を設定しても、緩い結合を指定していないとリソースファイルには次の行が生成されます。

```
XApplication*MyShell*button1.labelString:Bound
```

98 ページの「緩い結合の例」で説明したように、すべてのボタンを包含するために緩い結合を設定すると、リソースファイル内に次の行が生成されます。

```
XApplication*MyShell*MyForm*XmPushButton.labelString:Bound
```

その他のリソースに関する参考資料

ウィジェットリソースの詳細は、851 ページの第 27 章「ウィジェット・リファレンス」を参照してください。第 27 章「ウィジェット・リファレンス」では、各 Motif ウィジェットクラスに対して最もよく使用されるリソースについて記述しています。リソースを設定する前に、この概要情報をご覧になることをお勧めします。

Motif ウィジェットリソースの詳細については、Motif プログラミング・マニュアルなどの参考文献が多く発売されていますので参照してください。その他の参考文献については、付録 E「参考資料」をご覧ください。

他のウィジェットを用いて Sun WorkShop Visual を使用している場合には、ウィジェットの開発元で提供している資料も参照してください。

第4章

配置エディタ

はじめに

インタフェース設計の次の段階では、ウィジェットを物理的に配置します。ダイアログテンプレートによって、メニューバーはウィンドウの上部に、ボタンはウィンドウの下部に自動的に配置されます。しかし、フォーム内でのウィジェットの配置はユーザーの好みで変更できます。

図 4-1 に、第 3 章までの学習用インタフェースの外観と、本章での手順に従って修正を行なった後の外観を示します。

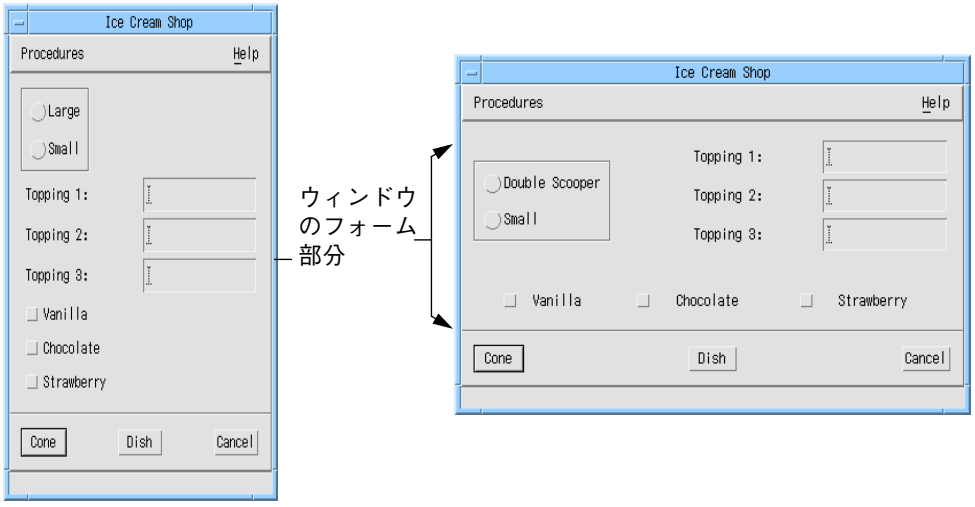


図 4-1　学習用インタフェースのデフォルトと修正後の配置

フォーム内でウィジェットを配列することを配置といいます。配置を変更するには、Sun WorkShop Visual の対話型配置エディタを使用して以下のステップを実行します。

1. ウィジェットを好みの配置に移動します。
2. 固定オフセットでフレームの上部および左側をフォームの端に接続します。
3. 固定オフセットでローカラムをフレームに接続します。
4. 配置の一番下にある 3 個のトグルボタンの上部を揃えます。
5. 3 個のトグルボタンの間隔を均等にするために位置アタッチメントを設定します。

概念

Motif ウィジェットのクラスには、その子にジオメトリ (物理的な位置関係) の規則を強制できるものがあります。ラジオボックス、ローカラムおよびメニューバーの子が特別の方法で配置されていることはすでに説明しました。メニューバーの場合には、カスケードボタンは単一行に配置され、ローカラムの場合には、すべての子はグリッド状に配置されます。

フォーム、ブリテンボードおよび描画領域のウィジェットでは、柔軟に子の配置を行うことができます。これら 3 つのウィジェットを配置ウィジェットといいます。

Sun WorkShop Visual には、これらのウィジェットの子を配置するための対話型配置エディタがあります。

アタッチメント

アタッチメントは、あるウィジェットを配置ウィジェットまたは配置ウィジェットの別の子に対して、特定の相対的な位置に強制するコンストレイント (制約) です。
3 種類のすべての配置ウィジェットでは、子の左上隅および右下隅を配置ウィジェットに相対する任意の *x*、*y* 位置に接続することができます。ウィジェットの左上隅を制約すると、その位置が固定されます。また、右下隅を制約すると、そのサイズを固定することになります。これらがブリテンボードおよび描画領域ウィジェットにより適用されるアタッチメントです。

フォームアタッチメント

フォームウィジェットを使用すると、指定した間隔 (ピクセル単位) で子ウィジェットの端をフォームの端に接続することができます。これにより、ブリテンボードや描画領域で提供するアタッチメントと同様に、特定の *x*、*y* 位置での配置が行われます。フォームの場合、この型のアタッチメントはフォームアタッチメントと呼び、フォームで使用できる他の型のアタッチメントと区別しています。

位置アタッチメント

フォームでは、フォームの幅あるいは高さの合計に対してのパーセンテージで指定される位置アタッチメントも使用できます。位置アタッチメントを使用すると、ウィンドウが大きくなるにつれてウィジェットの間隔が広くなるため、余分なスペースが生じた場合にはインタフェースでそのスペースを活用することができます。

ウィジェットアタッチメント

ウィジェットアタッチメントを使用すると、指定した絶対間隔 (ピクセル単位) でフォームの 2 個の子を互いに接続することができます。アタッチメントは、ウィジェットの端と端、あるいは上部と下部で行うことができます。ウィジェットのグループを揃え、配置するための機能も備えています。

これらのフォームの追加機能を使用すると、メインウィンドウまたは個々のウィジェットがサイズ変更された場合でも、体裁が悪くならないようにすることができます。たとえば、ウィジェットをフォームの両端に接続すると、 メインウィンドウが拡大された場合には一緒に広がるようにすることができます。また、2 個のウィジェットの端と端を接続すると、片方のウィジェットのサイズが変更された場合でも相互の間隔が保たれます。

配置エディタの表示

ウィジェットを選択して「ウィジェット」メニューで「配置」を選択するか、図 4-2 に示すツールバーの「配置」ボタンをクリックすると、階層内にある配置ウィジェットの配置エディタを表示することができます。

図 4-2　ツールバー上の「配置」ボタン

フォーム内の配置を編集するためには、次のようにします。

1. フォームを選択します。
2. ツールバー上の「配置」ボタンをクリックします。

図 4-3 のようなダイアログが表示されます。

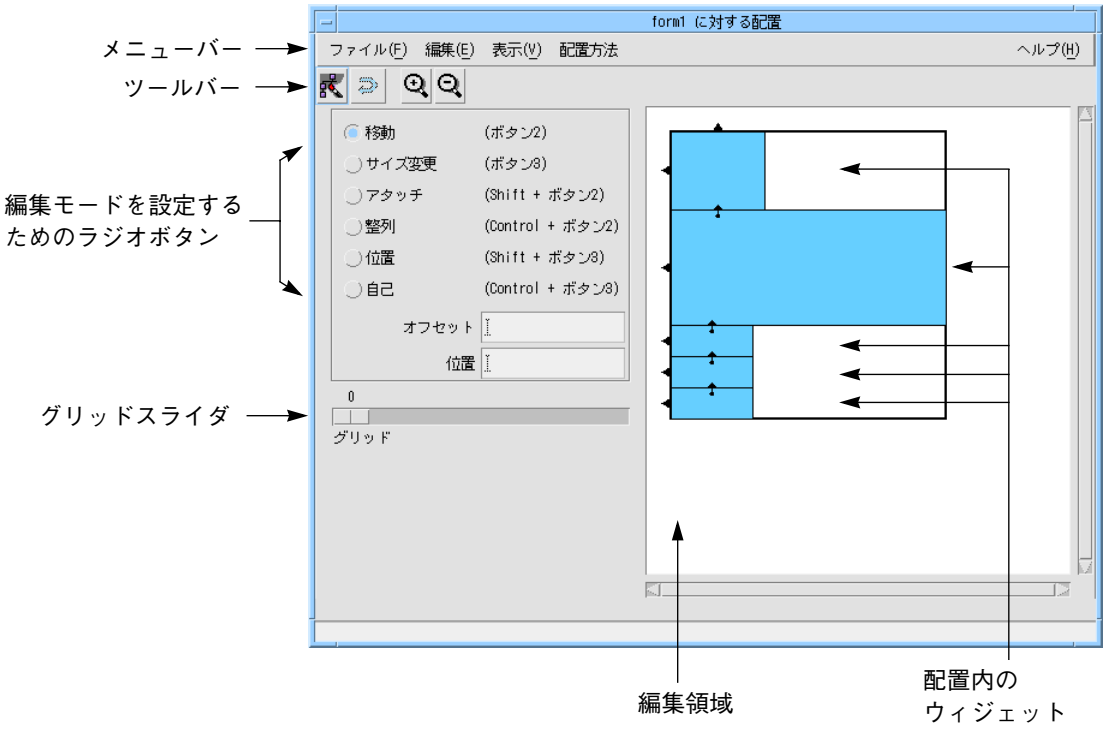


図 4-3　「配置エディタ」ダイアログ

編集領域

編集領域は、配置の略図を表示します。編集領域内のウィジェットは、フォームを表わす大きなボックス内に、ボックスとして枠組みが表示されます。図 4-3 におけるボックスは上から下の順に、フレーム、ローカラムそして 3 個のトグルボタンを表わします。

ウィジェット同士が重なり合った場合でも境界線は表示されるため、ウィジェットの位置を確認することができます。重なり合ったウィジェットのうちの 1 つを選択する場合は、最もサイズの小さな包含ウィジェットが選択されます。これによって、小さなウィジェットが大きなウィジェットに隠れて選択できないという問題を回避することができます。

注 - 配置エディタは、フォームの子を表示しますが、それよりも下の階層は表示しません。たとえば、フレームの中にあるラジオボックスおよびそのトグルボタンや、 ローカラム内のラベルおよびテキストフィールドは表示されません。
また、フォームの外側にあるメニューバーやプッシュボタンも表示されません。
配置エディタには、メニューバー、ツールバー、コマンドボタンがあります。
そして、いくつかの配置エディタコマンドにはキーボードアクセラレータがあります。これらを使用する場合、Sun WorkShop Visual メインウィンドウにおける別機能のアクセラレータと同じキャラクタ使用する時には、配置エディタ画面に入力フォーカスがあることを確認して下さい。

編集モード

エディタ画面の左側にあるラジオボタンを使用すると、複数の編集モードの中から 1 つを選択することができます。編集モードを選択すると、マウスボタン 1 にその機能が割り当てられます。

ラジオボタンの隣に示されているマウスボタンとキー操作の組み合わせを使用すると、現在の選択とは関係なく任意の編集モードを使用することができます。
たとえば、ウィンドウの移動には常にマウスボタン 2 を、そしてアタッチメントの設定には Shift を押しながらマウスボタン 2 を使用することができます。

以下に示す 6 つの編集モードがあります。

- 移動
  120 ページの「大まかな配置: 移動モード」を参照してください。
- サイズ変更
  146 ページの「サイズ変更モード」を参照してください。
- アタッチ
  127 ページの「ウィジェット間のアタッチメント」を参照してください。
- 整列
  132 ページの「ウィジェットの整列: 整列モード」を参照してください。135 ページの「ウィジェットの整列: グループの整列」で説明されているツールバー上の「整列」ボタンの使い方とは少し異なります。
- 位置
  142 ページの「均等配置: 位置モード」を参照してください。
- 自己
  144 ページの「自己モード」を参照してください。

セレクション: 単独、一次、二次

ウィジェットを選択するには、編集領域内の該当するウィジェット上をクリックします。さらに別のウィジェットも選択するには、Shift キーを押しながら目的のウィジェットを選択します。一次セレクションは常に太線の境界線で表示され、二次セレクションは点線の境界線で表示されます。最後に選択されたウィジェットが常に一次セレクションです。整列は一次セレクションを基準にして行われるため、どのウィジェットが一次セレクションなのかを知っておく必要があります。

配置エディタのツールバー

メニューバーの下には、以下に示すボタンを含むツールバーがあります。これらのボタンは、メニューからも使用できます。

115 ページの「リセット」を参照してください。

115 ページの「元に戻す」 を参照してください。

117 ページの「ズームイン、ズームアウト」を参照してください。

117 ページの「ズームイン、ズームアウト」を参照してください。

配置エディタの「ファイル」メニュー

「ファイル」メニューには「閉じる」という項目しかありません。この項目を選択すると、画面から配置エディタウィンドウが消えます。

配置エディタの「編集」メニュー

「編集」メニューには、以下に示す 3 つの機能があります。

元に戻す

「元に戻す」ボタンは、配置エディタで行われた最後の操作を元に戻します。「元に戻す」を繰り返しクリックすると、複数の操作をさかのぼって元に戻すことができます。他の配置エディタコマンドの場合と同様に、元に戻した結果を目で確認するには「リセット」ボタンを押さなければならないことがあります。

リセット

「リセット」(<Ctrl>-T) は、選択しているウィジェットの現在のインスタンスとその子を破壊し、ダイナミックディスプレイを再度作成します。

配置ウィジェットのリセットを行うまで、新しいコンストレイントがダイナミックディスプレイウィンドウに正確に反映されない場合があります。このため、配置エディタを使用している場合、特に変更を行なっても意図した結果が出ない場合には、頻繁にリセットを行うようにしてください。

フォーム内のコンテナウィジェットがリセット後に適切な表示を行わない場合には、上の階層のウィジェットを選択して再度リセットを行います。

配置エディタの「表示」メニュー

配置エディタの「表示」メニューには、以下に示す 4 種類の便利な表示コマンドがあります。

端の強調表示

このオプションは、ポインタがウィジェット内にある場合に、ポインタに最も近い端を強調表示します。ウィジェットに設定したアタッチメントは、強調表示されている端に適用されます。デフォルトは「端の強調表示」が適用されています。

- **配置エディタ上のフォームウィジェットの周囲で、ポインタを移動させてください。ポインタがウィジェットを表わしているボックス内にある場合に、ポインタに最も近い端が強調表示されることが確認できます。**

注釈

このオプションは、編集領域にある各ボックス内に識別のための文字列を表示します。「表示」メニューをプルダウンし、「注釈」を選択すると、「ウィジェット名」と「クラス名」という選択肢を持つプルライトメニューが表示されます。
「ウィジェット名」(<Ctrl-W>) は、各ウィジェットの変数名を表示します。
「クラス名」(<Ctrl-N>) は、各ウィジェットのクラス (例:「フレーム」または「ローカラム」など) を表示します。1 つのオプションを選択すると、もう 1 つのオプションは使用できなくなります。オプションが設定されている場合に同じオプションを再度選択すると、表示からすべての注釈が取り除かれます。

注釈オプションを図 4-4 に示します。

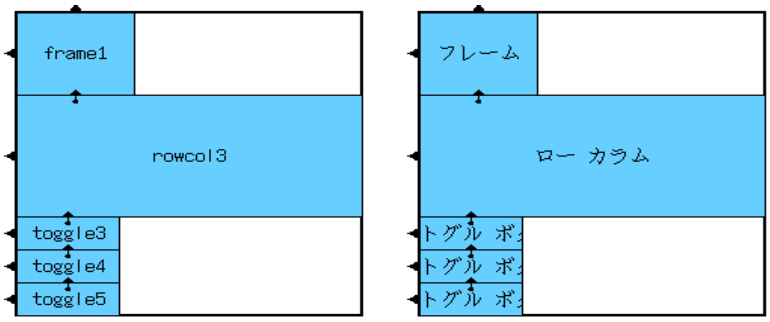


図 4-4　注釈オプション

- 「表示」メニューをプルダウンし、「注釈」サブメニューから「クラス名」を選択すると、表示内容の変化を確認することができます。

ズームイン、ズームアウト

「ズームイン」を使用すると表示スケールが大きくなり、「ズームアウト」を使用するとスケールが小さくなります。

配置エディタの「配置方法」メニュー

「配置方法」メニューには、パレット上の編集モード (113 ページの「編集モード」を参照してください) の他にもいくつかのコマンドがあります。このメニューからは、「整列」と「均等配置」の 2 つの機能を使用することができます。

整列

「整列」を押すと、使用できる整列機能がプルライトメニューに表示されます。これらの機能は、ウィジェットを 2 つ以上選択した場合に限り使用できます。
このメニューにある整列機能の詳細は、135 ページの「ウィジェットの整列: グループの整列」を参照してください。

均等配置

「均等配置」を選択すると配置方法を指定するためのプルライトメニューが表示され、そのメニューから、さらに 2 種類のプルライトメニューが表示されます。均等配置機能は、ウィジェットを 3 つ以上選択した場合に限り使用できます。このメニューにある均等配置機能の詳細は、138 ページの「均等配置」を参照してください。

グリッド

配置上にグリッドを表示するには、配置エディタウィンドウの左下にあるグリッドスライダを使用します。グリッドの間隔は、2 から 50 ピクセルの間で選択することができます。ピクセル数は、1 : 1 のスケールとみなされるため、ズームコマンドを使用する場合にはグリッドは配置にあわせて伸縮します。

「移動」および「サイズ変更」モードでは、グリッドが表示されている場合は、移動およびサイズ変更がグリッド単位で行われます。ウィジェットを移動する場合、その左上隅が最も近くにあるグリッドの交差点に配置されます。また、同様に、ウィジェットのサイズ変更を行う場合には、右下隅がグリッドの交差点に配置されます。

スライダをゼロに設定すると、グリッドは表示されなくなります。

- グリッドスライダを 10 ピクセルに設定します。

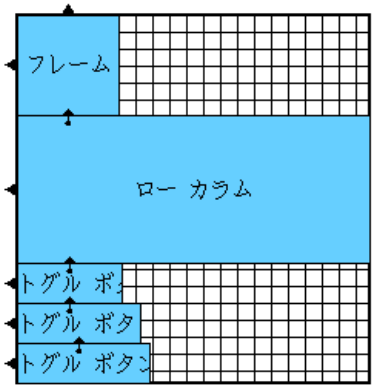


図 4-5　10 ピクセルのグリッドを使用する学習用配置

サイズ変更の方針

ほとんどのフォームリソースは、デフォルト値のままにしておくことをお勧めします。ただし、「サイズ変更の方針」リソースは、「拡大のみ」に設定した方がよい場合もあります。

1. フォームウィジェットをダブルクリックするか、ツールバーの「リソース」ボタンを押して、フォームのリソースパネルを表示します。
2. リソースパネルの「設定」ページを選択します。
3. 「サイズ変更の方針」を「拡大のみ」に設定します。
4. 「適用」をクリックして、その後「閉じる」をクリックします。

このリソースを設定しないと、ウィジェットを移動する際にフォームは即座に最小サイズに縮小してしまいます。「拡大のみ」オプションを使用すると、配置に余分のスペースが生じることもありますが、フォームのリセット時にはそのスペースはなくなります。

デフォルト配置

配置エディタ内のアタッチメントは、図 4-6 に示される記号を使用して表示されます。

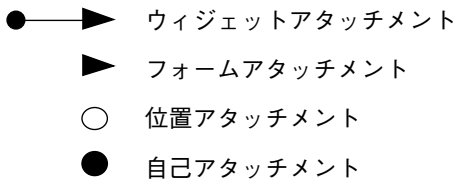


図 4-6　配置エディタで使用される記号

使用している配置には、すでにフォームアタッチメントとウィジェットアタッチメントの 2 種類のアタッチメントが存在しています。フォームアタッチメントは、ウィジェットの片方の上端に、矢印で示されます。デフォルト配置の一番上にあるウィジェットであるフレームには、フォームの上部および左側に、このようなアタッチメ

ントが 2 個あります。他の 4 個のウィジェットは、それぞれフォームの左側に接続されます。すべてのウィジェットは、階層に追加された順序で上から下に重ねられます。

下の 4 個のウィジェットはそれぞれその上のウィジェットに接続されています。ウィジェットアタッチメントは、塗りつぶされた小さな矢先で一方に塗りつぶした円のある矢印で表示されます。

デフォルトのメニューバーアタッチメント

上述のデフォルトアタッチメント以外では、メニューバーにのみデフォルトのアタッチメントがあります。メニューバーは、ウィンドウの最上部に配置して適当な大きさにサイズ変更する必要があります。そのため、メニューバーの右端からフォームの右辺までデフォルトアタッチメントが設定されています。

大まかな配置: 移動モード

「移動」コマンドを使用すると、新しい位置にウィジェットをドラッグすることができます。このコマンドは、ウィジェットの左上隅を固定する 2 個のフォームアタッチメントを設定します。配置を開始する最適な方法は、このモードを使用して、最終配置で予定しているおおよその位置にすべてのウィジェットを移動することです。その後で、他のモードを使用して精密にウィジェットの位置やサイズ変更を指定することができます。

移動中のアタッチメントの削除

ウィジェットを移動すると、そのウィジェットに関連するアタッチメントがすべて削除されます。ウィジェットのアタッチメントは保持されます。削除されないようにするには、Control キーを押しながらウィジェットを移動します。整列や均等配置などの機能はアタッチメントにもとづいて実行されるため、デフォルトでは Sun WorkShop Visual がアタッチメントを削除します。新しく作成したアタッチメントと既存のアタッチメントとの間で矛盾が発生し、フォームからエラー報告をすることが頻繁にあります。いったんこのような状況になると、回復は不可能ではありませんが、非常に困難です。130 ページの「循環アタッチメント」を参照してください。このような状況を避けるには、配置機能を実行する前には必ず Sun WorkShop Visual がアタッチメントを削除するように設定することをお勧めします。

移動

「移動」モードを起動するには次のようにします。

1. 「移動」トグルをクリックします。

   「移動」モードでは、マウスボタン 1 を使用してウィジェットを配置内でドラッグすることができます。また、現在のモードとは関係なく、マウスボタン 2 を使用してウィジェットを移動させることもできます。

2. ローカラムに対応するボックス内にポインタを置きます。

3. マウスボタン 1 を押したまま、フォームの右端から外に出るまでローカラムを右方向にドラッグします。

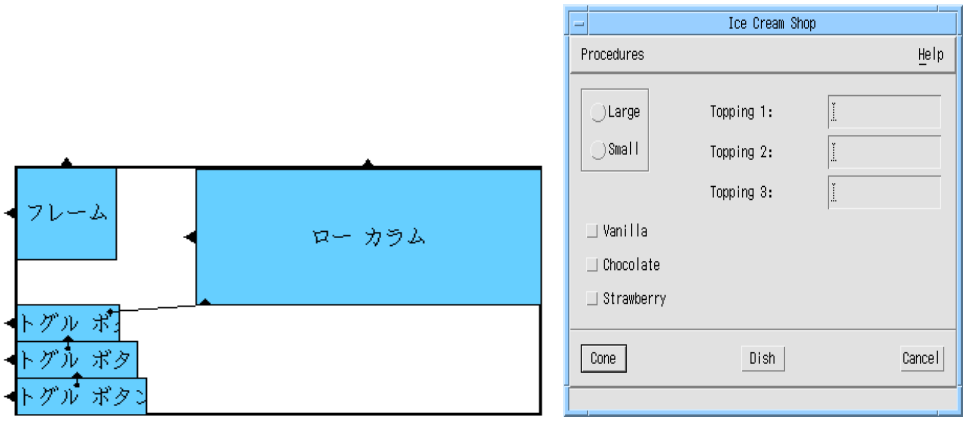
   この場合、ローカラムの右マージンがフォームの端を超えるまでドラッグすることになりますが、問題はありません。マウスボタンを離すと、フォームは自動的にすべての子が収まるようにサイズ変更が行われます。

4. フレームの上部に揃うまでローカラムを上方にドラッグします。

5. フォームをリセットします。

   正しく表示されない場合は、シェルをリセットしてからフォームを選択し、配置エディタを再表示します。

   図 4-7 に現段階での配置の外観および結果のダイナミックディスプレイを示します。ダイアログテンプレートはフォームの新しい幅に適合するために、自動的にメニューバーのサイズおよびプッシュボタンの間隔を調整します。



配置エディタ上の配置　　ダイナミックディスプレイ

図 4-7　配置エディタに表示される大まかな配置

移動モードの作用

ローカラムおよびフレームの上部および左端には、フォームの上部および左端にアタッチメントで接続されていることを示す矢印があります。「移動」は、フォームに 2 個のアタッチメントを設定してウィジェットの左上隅を新しい位置に固定します。これらのアタッチメントは、間隔が 0 (ウィジェットはフォームの端に接触する) でも、または 0 以外の間隔 (ウィジェットとフォームとの間に空間ができる) でも作成することができます。この間隔はオフセットと呼ばれます。この場合、ローカラムの左端はゼロ以外のオフセットでフォームに接続されています。オフセットについては、この章で後述します。「移動」モードの場合、オフセットはポインタの位置にもとづいて計算されます。

アタッチメントの置換

ウィジェットの片方の端には、1 個のアタッチメントだけを作成することができます。「移動」は端にあるアタッチメントを壊し、移動されるウィジェットの上部および左端に新しい 2 個のアタッチメントを設定します。デフォルト配置の場合、ローカラムの上部はフレームの下部に設定されていました。しかし「移動」によってローカラムの上部に新しいアタッチメントが設定されたため、このアタッチメントはもう存在しません。「移動」は、ウィジェットから作成されたアタッチメントを削除しますが、他のウィジェットから作成されているアタッチメントは維持します。

注 - Control キーを押しながら移動を行うと、アタッチメントを維持したままウィジェットを移動することができます。

他のアタッチメントに影響するアタッチメント

ローカラムウィジェットを上に移動する場合、3 個のトグルボタンが一緒に移動します。これは、ウィジェット間のアタッチメントにより起こるものです。最初のトグルボタンの上部は ローカラムの下部に接続されているため、ローカラムが上方に移動される場合にはトグルボタンの上部も一緒に移動することになります。その他の 2 個のトグルボタンも、2 番目のボタンが最初のボタンと 3 番目のボタンに接続されているため、連鎖反応として移動します。

ローカラムを移動しても、トグルボタンからローカラムへのアタッチメントは、取り除かれません。これは、このアタッチメントがローカラムにではなく、トグルボタンに属しているためです。この区別については、本章で後述します。

オフセット

現段階では、ローカラムおよびフレームの上部が揃っています。しかし、これらがフォームに接続されている方法には大きな違いがあります。フレームの上部は、デフォルト配置から残っているデフォルトアタッチメントであり、ローカラムは、本章の 121 ページの「移動」で設定されたゼロアタッチメントで接続されています。

デフォルトと明示的オフセット

デフォルトオフセットは、フォームの「水平間隔」と「垂直間隔」リソースにより制御されています。明示的オフセットは、配置エディタ内での移動または整列などのアクションを介して、デフォルトオフセットを上書きします。デフォルトでは両方のリソースが 0 であるため、フレームとフォームの上部との間は 0 ピクセルです。間隔を設定し直した結果を確認するには次のようにします。

1. フォームを選択して、ツールバーの「リソース」をクリックします。
2. フォームリソースパネル上で「表示」ページを選択します。

3. 「横の間隔」ボックスをダブルクリックし、次のように入力します。

   20

4. 「縦の間隔」ボックスをダブルクリックし、次のように入力します。

   20

5. 「適用」をクリックして、その後「閉じる」をクリックします。

6. 配置エディタで「リセット」をクリックします。

   図 4-8 に結果を示します。フレームおよび 3 個のトグルボタンを含む、デフォルトオフセットを使用するすべてのアタッチメントは、この時点で 20 ピクセル間隔を使用しています。ローカラムは、そのアタッチメントが「移動」を使用して設定されたものであり、間隔リソースを参照しない明示的オフセットを使用しているため移動しません。結果としては、フレームとローカラムは揃わなくなります。

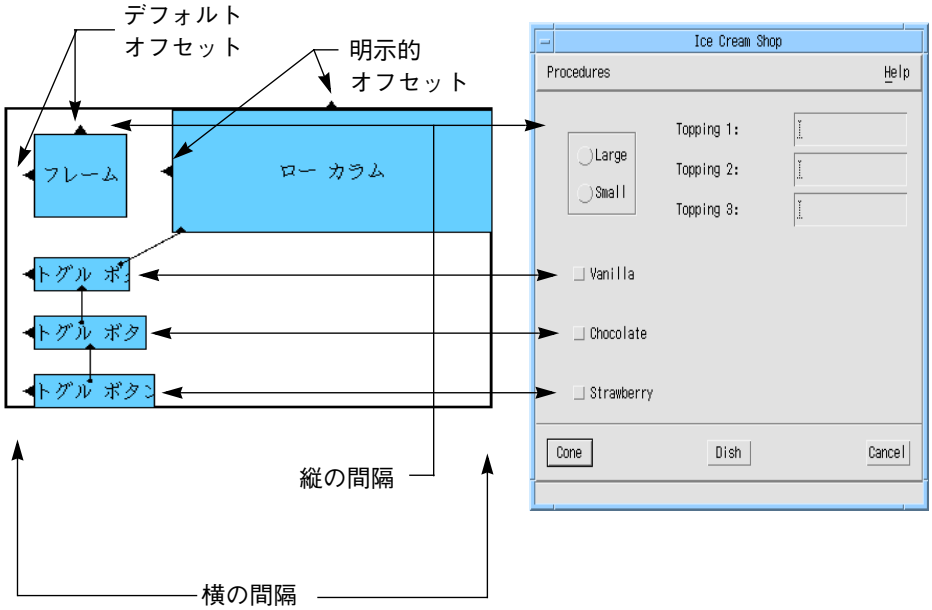


図 4-8　垂直および水平間隔のリセットによる効果

トグルボタンの間の余分なスペースにより、フォームは全体的に大きくなるように強制されます。ダイアログテンプレートもまた、フォームを収めるためにサイズ変更を行います。

デフォルトオフセットの利点は、実行の配置の間隔をユーザーが制御できるということです。しかし、デフォルトオフセットのおもな欠点は、不要であっても配置内のすべての間隔が等しくなるということです。また、デフォルトオフセットは明示的オフセットが同じ値のままでも変化することがあり得るため、デフォルトと明示的オフセットを混同しないように注意してください。たとえば、間隔を変更したことによりフレームとローカラムが揃わなくなった場合は、一方が明示的オフセットを、もう一方がデフォルトオフセットをフォームの上部から使用していることが原因です。配置エディタ画面は、明示的およびデフォルトオフセットを区別しません。

フォームへのアタッチメント

ウィジェットの端のすぐ内側からフォームの端のすぐ外側にドラッグを行うと、ウィジェットをフォームに接続することができます。これは「アタッチ」モードでボタン 1 を使用して、あるいはすべてのモードの場合に <Shift-button 2> を使用して実行できます。アタッチメントは、「オフセット」フィールドのオフセット値を使用して設定されます。「オフセット」フィールドが空の場合には、デフォルトオフセットが使用されます。

1. 「アタッチ」トグルをクリックします。

   「アタッチ」モードの場合、あるいは <Shift-button 2> を押している場合、ポインタは十字型になります。

   ここで、デフォルトアタッチメントのいくつかを 0 ピクセルの明示的オフセットを使用するアタッチメントと置き換えます。

2. 「オフセット」ボックスをクリックし、次のように入力します。

   ```
   0
   ```

   フレームの左端をフォームの左端に接続します。

3. 十字型ポインタをフレーム左端のすぐ内側に配置し、左端を強調表示します。
4. マウスボタン 1 を押したまま、フォーム左端のすぐ外側にドラッグします。
5. マウスボタンを離します。

新しいアタッチメントは、古いアタッチメントと同様に、塗りつぶされた三角形としてフレームの端に表示されます。明示的な 0 オフセットはフレームをフォームの端に移動するため、効果を確認することができます。この移動が発生しない場合には、アタッチメントの設定を再度行なってみてください。

「Vanilla」トグルボタンにも同様の操作を行います。

6. 十字型ポインタを一番上のトグルボタンの左端のすぐ内側に配置し、その端を強調表示します。
7. マウスボタン 1 を使用してフォーム左端のすぐ外側にドラッグします。
8. マウスボタンを離します。

   トグルボタンはフォームの端に移動します。

   フレームの上部は、現在の位置でも体裁よく見えます。フォームの上部から 20 ピクセルというオフセットを使用すると、ローカラムに対してフレームが中央に寄せられます。しかし、デフォルトではなく明示的オフセットにすると、フォーム間隔リソースを変更した場合でも変化せずにそのまま維持されます。明示的オフセットに変更するためには、アタッチメントを置き換える必要があります。

9. 「オフセット」フィールドをダブルクリックして、次のように入力します。
   20
10. 十字型ポインタをフレーム上部のすぐ内側に置き、その端を強調表示します。
11. マウスボタン 1 を使用して、フォーム上部のすぐ外側の位置にドラッグします。
12. マウスボタンを離します。

    図 4-9 に結果を示します。ダイナミックディスプレイで同じ表示が行われない場合には、次のようにします。

13. 「リセット」をクリックします。

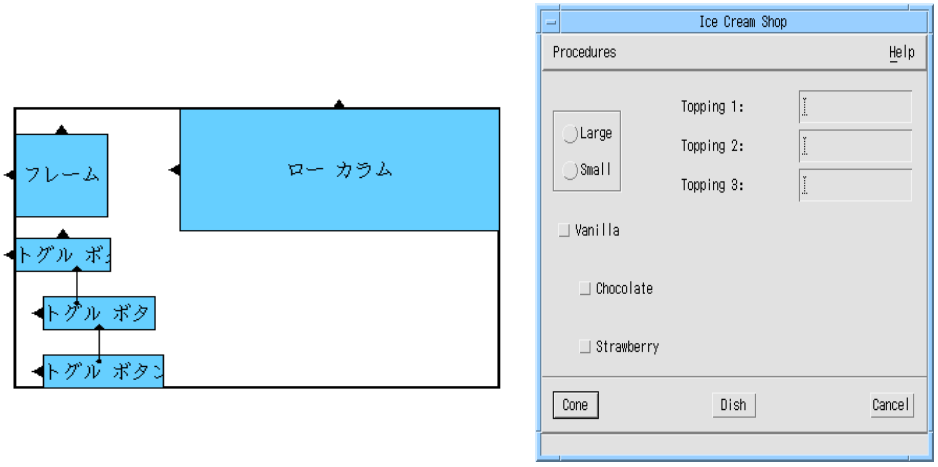


図 4-9　20 ピクセルフォーム間隔のフォーム配置

ウィジェット間のアタッチメント

フォーム内の 2 個のウィジェット間のアタッチメントの設定は、ウィジェットをフォームの端に接続する場合と似ています。1 組のウィジェットを接続するためには、単純に、片方のウィジェットのすぐ内側から別のウィジェットのすぐ内側に十字型ポインタをドラッグします。

ウィジェット端部間のアタッチメント

2 個のウィジェットを接触させて、あるいはオフセットを使用して端と端で接続するためには、一方の右端をもう一方の左端に接続するか、あるいは一方の上部をもう一方の下部に接続します。

前のセクションから引き続き、「アタッチ」モードを使用します。ローカラムの左側をフレームの右側に接続します。

1. 「オフセット」ボックスをダブルクリックして、次のように入力します。

   50

2. ポインタをローカラムの左端のすぐ内側に配置して、左端を強調表示します。

3. マウスボタン 1 を押したまま、フレームの右端が強調表示されるまでフレームのすぐ内側の位置までドラッグします。

4. ボタン 1 を離します。

   新しいアタッチメントは、フレームの右端を指している矢先でローカラムの左端の中央に塗りつぶした円のある矢印で表示されます。ローカラムは、フレームの右端から 50 ピクセルの間隔を置いて、 ローカラム自身の位置を変更します。

   これにより現段階で配置が大幅に変更されることはありませんが、フレーム内の文字列が変更される可能性がある場合は、この種類のアタッチメントが大変役に立ちます。ローカラムは、この時点でフォームの端にではなく、フレームの右端に相対して配置されています。したがって、フレームが拡大された場合でも、ローカラムは 50 ピクセルの間隔を保持します。

5. 構成領域にあるラジオボタン内の最初のラジオボタンをダブルクリックします。

6. リソースパネルの「表示」ページに移動して、ボタンのラベルを次のように変更します。

   `Double Scooper`

   結果を図 4-10 に示します。

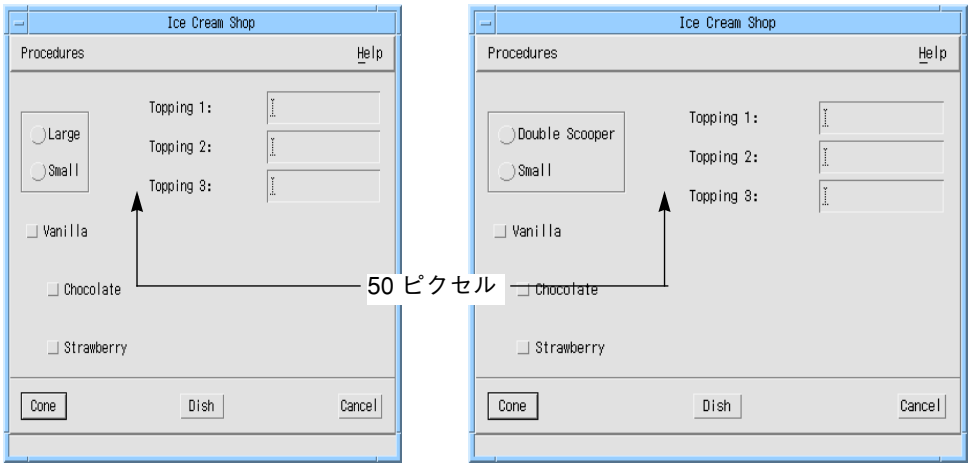


図 4-10 フレームのサイズ変更動作

アタッチメントの方向

アタッチメントは対称ではありません。ウィジェット A からウィジェット B へポインタをドラッグしてアタッチメントを作成する場合、そのアタッチメントはウィジェット A に起因している、と言います。これを示すために、Sun WorkShop Visual はウィジェット A 内にウィジェット B への矢印を描画します。先程作成したアタッチメントは、ローカラムに起因しています。

アタッチメントは、それが起因するウィジェットにのみ適用します。たとえば、先程作成したアタッチメントはローカラムを、フレームに相対する特定の位置に制約します。フレームが移動またはサイズ変更された場合には、ローカラムもそれに従って移動します。ローカラムが移動またはサイズ変更された場合には、フレームへの影響はありません。

1 つの座標にのみ影響するアタッチメント

一番上のトグルボタンには、ローカラム下部へのアタッチメントがあります。このアタッチメントは、デフォルト配置から残されたものであり、デフォルトのオフセットを使用しています。したがって、これはフォームの間隔リソース (20 ピクセルに設定されている) によって制御されます。

ローカラムは、一番上のトグルボタンから離して移動させることができました。これは以下の 2 つの規則によるものです。

1. ウィジェットの上部または下部のアタッチメントは、その *y* 座標にのみ影響する。
2. ウィジェットの左または右端のアタッチメントは、その *x* 座標にのみ影響する。

一番上のトグルボタンは、ローカラムの下部から 20 ピクセル (現在の垂直間隔) 離れた場所にあります。トグルボタンの左または右側とローカラムとの間にはアタッチメントがないため、ローカラムの水平方向の位置はトグルボタンに影響を及ぼしません。

このアタッチメントの間隔を 10 ピクセルに変更します。

1. 「オフセット」フィールドをダブルクリックし、次のように入力します。
   ```
   10
   ```
2. 十字型ポインタを一番上のトグルボタン上部のすぐ内側に配置して、その端を強調表示します。

3. マウスボタン 1 を押したまま、ローカラムの下部のすぐ内側にドラッグし、その端を強調表示します。

4. マウスボタンを離します。

   新しいアタッチメントは、古いアタッチメントと置き換えられ、一番上のトグルボタンの位置が変わります。

循環アタッチメント

フォームウィジェットに対しての Motif の規則では、ウィジェット A をウィジェット B に接続し、さらにウィジェット B をウィジェット A に接続することを禁止しています。これは循環アタッチメントと呼ばれます。 ウィジェット A から B、B から C、そして C から A という大きなアタッチメントのループもまた循環アタッチメントと見なされ、エラーの原因となります。配置にこのようなループが含まれている場合、図 4-11 に示されるような警告メッセージが表示されます。この場合は、ループを切断するために複数のアタッチメントを壊さなければなりません。

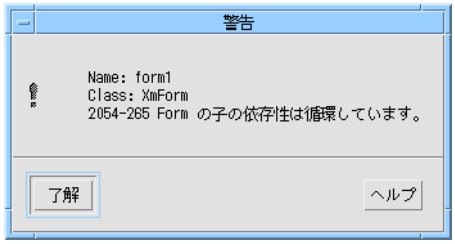


図 4-11 循環の警告メッセージ

循環は、ウィジェットのアタッチメントの場合にのみ問題となります。フォームへのアタッチメントおよび位置アタッチメント (142 ページの「均等配置: 位置モード」を参照) は、親フォームからではなく子ウィジェットにのみ起因するため、循環アタッチメントは生じません。

循環を避ける方法

循環を避ける良い方法は、ウィジェット間のすべてのアタッチメントを 2 方向 (通常は上および左) のみにすることです。インタフェースを配置する場合、左上隅から開始して下方に進み、そして右に進みます。2 個のウィジェットを接続する場合には、

常に下、または右にあるウィジェットからアタッチメントを作成するようにします。このようにすると、すべてのアタッチメント矢印は同じ方向を指すため、循環アタッチメントが誤って作成されることはありません。

アタッチメントの削除

配置エディタにおいてアタッチメントを削除するには、以下の方法のどれかを使用します。

1. ウィジェットの端に任意の型の新しいアタッチメントを設定します。これにより古いアタッチメントが削除されて置き換えられます。
2. 「移動」を使用してウィジェットの位置を変更します。これによりそのウィジェットに起因するすべてのアタッチメントが削除されます。
3. 「アタッチ」モード (<Shift-Button 2>) を使用し、アタッチメントの起因するウィジェットの、端のすぐ内側をクリックします。これにより、新しいアタッチメントを設定することなく設定の削除が行われます。「端を強調表示」モードを使用すると、十字型ポインタを簡単に正しい位置に配置することができます。
4. 「元に戻す」をクリックし、最後に追加されたアタッチメントを削除します。

Motif では、各ウィジェットは少なくとも 2 つの端が接続されている必要があります。すべてのアタッチメントを削除した場合には、リセットを行うと、フォームとウィジェットの間に最後の配置にもとづいた単純な移動型アタッチメントが作成されます。

矛盾するアタッチメント

循環していない場合でも、互いに矛盾するアタッチメントを指定してしまう可能性もあります。たとえば、正のオフセットを使用して、ウィジェットの左端をフォームの右端に接続してしまう場合があります。矛盾したアタッチメントを持つフォームをリセットする場合、Motif はそれらのすべてを満たす配置を計算しようとします。計算のループを何度繰り返しても満足できる配置が見つからない場合には、そのループは中断され、図 4-12 のような警告メッセージが表示されます。このような状況では、問題を生じさせているアタッチメントを取り除くまでは、一部のウィジェットが非常に小さく表示されたり、あるいはフォーム自体が非常に広くまたは長くサイズ変更されている場合があります。

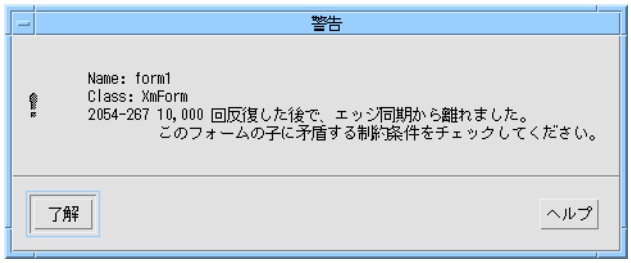


図 4-12 脱出の警告メッセージ

循環アタッチメントの場合と同様、矛盾するアタッチメントは、取り除くまでは作業を進めることができません。

アタッチメントの数の制限

ウィジェットは 4 個の端それぞれに起因するアタッチメントを 1 個ずつ持つことができます。アタッチメントはフォームアタッチメント、ウィジェットアタッチメントまたは位置アタッチメント (142 ページの「均等配置: 位置モード」を参照) から任意の型にすることができます。ウィジェットの端に起因する新しいアタッチメントを指定すると、その場所にすでに存在しているアタッチメントは新しいアタッチメントと置き換えられます。

ウィジェットへのアタッチメントは、それが他のウィジェットに起因している限り、その数に制限がありません。

ウィジェットの整列: 整列モード

2 個のウィジェットの整列は、オフセットゼロで両方の上部と上部を接続することにより実行することができます。配置エディタの「整列」モードを選択すると、簡単に整列を行うことができます。「整列」モードは、明示的なゼロオフセットを使用するだけの簡単なアタッチメントです。2 個のウィジェットを下部と下部、左端と左端、または右端と右端で揃えることもできます。この機能を使用する場合には、循環を作り出さないように注意してください。

最初の 2 個のトグルボタンの上部を揃えるには次のようにします。

1. 「移動」(マウスボタン 2) を使用して、2 個のトグルボタンをほぼ水平な位置に揃えます。

   一番上のトグルボタンは移動させてはなりません。

   「整列」の結果と、その後で「均等配置」を実行した結果を確認することができるよう、図 4-13 に示すように、ウィジェットの上部とそれらの端の間隔をわざと少しだけずらしておきます。

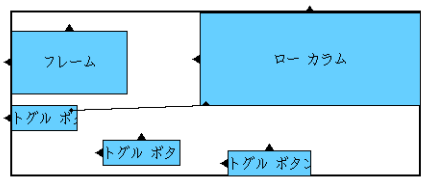


図 4-13 整列前のトグルボタン

2. 「整列」トグルをクリックします。
3. 中央のトグルボタンの上部のすぐ内側に十字型ポインタを配置します。
4. マウスボタン 1 を押したまま、最初 (左) のトグルボタンの上部のすぐ内側の位置にドラッグします。
5. マウスボタンを離します。

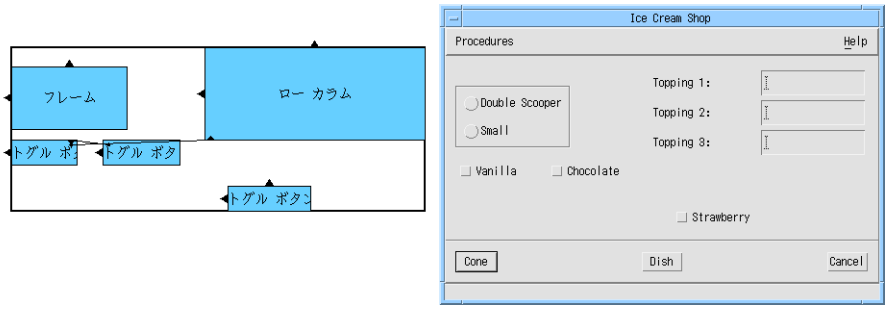


図 4-14 最初の 2 個のトグルボタンの整列の後

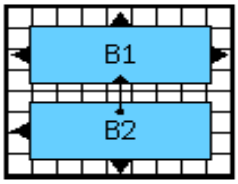
整列の作用

整列がどのように作用するかを理解するには、まず、アタッチメントが一方向のウィジェットの位置にのみ影響することを覚えておきます。たとえば、ウィジェット 1 の上部をウィジェット 2 の下部に接続する場合は、以下のようになります。

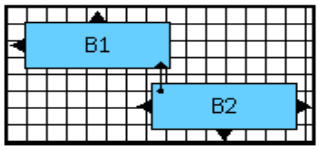
```
top_1 = bottom_2 + offset
```

ここで top_1 はウィジェット 1 の上部の y 座標を表わし、bottom_2 はウィジェット 2 の下部の y 座標を表わします。これは、2 個のウィジェットが水平方向で重なるかどうかには関係なく該当します。

例 A
B1 の下部に B2 の上部をオフセット 10 で接続する。
効果: B2 の上端は B1 の下端よりも 10 ピクセル下になる。



例 B
例 A と同じウィジェットアタッチメント。ウィジェットは x 方向には重ならない。



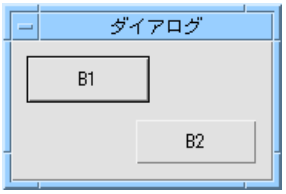


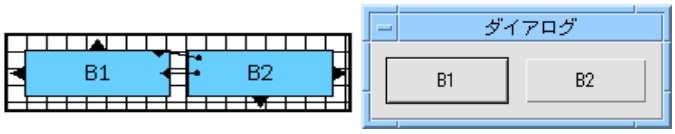
図 4-15 2 個のウィジェットの上部から下部へのアタッチメント

同様に、2 個のウィジェットの上部と上部を接続する場合は、以下のようになります。

```
top_1 = top_2 + offset
```

オフセットが 0 である場合には、2 個のウィジェットの上部の y 座標は同じになります。つまり、これらのウィジェットの位置が揃っていることになります。

例 A
B1 の上部に B2 の上部をオフセット 0 で接続する。効果: ウィジェットの上部が揃う。



例 B
B1 の上部に B2 の上部をオフセット 10 で接続する。効果: B2 の上端は B1 の上部よりも 10 ピクセル下になる。

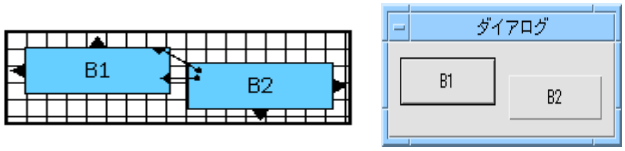


図 4-16 2 個のウィジェットの上部と上部のアタッチメント

図 4-16 の例 A では、「整列」モードで使用するアタッチメントの型を示しています。例 B では、0 以外のオフセットを使用する類似したアタッチメントを示しています。その効果は整列に似ていますが、段差が生じます。

ウィジェットの整列: グループの整列

「整列」モードを使用する、あるいはオフセット 0 で「アタッチ」モードを使用して 2 個のウィジェットの左、右、上または下の端を接続すると、それらのウィジェットを対で整列させることができます。「配置方法」メニューの整列機能を使用すると、ウィジェットのグループの位置を素早く揃えて並べることができます。この機能で設定されるアタッチメントは、「アタッチ」または「整列」モードでウィジェットの整列に使用するものと同じ種類です。

これらのボタンは、複数のウィジェットが選択されている場合にのみ使用することができます。ウィジェットの複数選択および一次セレクションの詳細は、114 ページの「セレクション: 単独、一次、二次」を参照してください。.

整列機能

「配置方法」メニューの「整列」オプションには、次の 6 種類のオプションが選択できるプルライトメニューがあります。

- 左端
- 左端と右端

- 右端
- 上端
- 上端と下端
- 下端

整列の使用方法

グループ整列の効果を確認するには次のようにします。

1. 「移動」モードまたはマウスボタン 2 を使用して、2 番目のトグルボタンの位置をずらします。
2. 「移動」モードまたはマウスボタン 2 を使用して、右側のトグルボタンの右端がフレームの右端に揃うように、トグルボタンを移動します。

   次に、3 個のトグルボタンの上部をグループとして揃えます。
3. 「移動」モードにしてから、右のトグルボタンをクリックします。

   「移動」モードでウィジェットをクリックすると、そのウィジェットの枠が強調表示され、選択されていることを示します。最初のウィジェットをクリックした後、<Shift-Button 1> を使用してウィジェットをグループに追加する必要があります。図 4-17 のように、一次セレクションのウィジェットの枠が強調表示され、その他のウィジェットの枠は点線で表示されます。

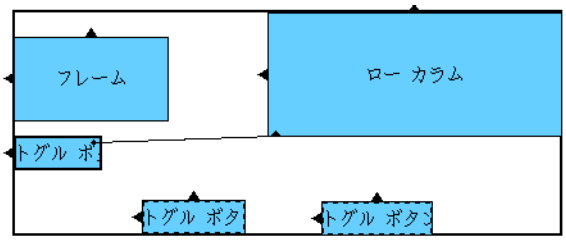


図 4-17 一次セレクションと二次セレクション

   ここで「整列」を選択すると、一次セレクションを基準に整列が実行されます。
4. 中央のトグルボタンを <Shift-Button 1> でクリックします。
5. 左のトグルボタンを <Shift-Button 1> でクリックします。

トグルボタンの上部の位置を揃えるには、次のようにします。

6. 「配置方法」メニューの「整列」プルライトメニューから「上端」を選択します。

図 4-18 に示すアタッチメントが設定されます。

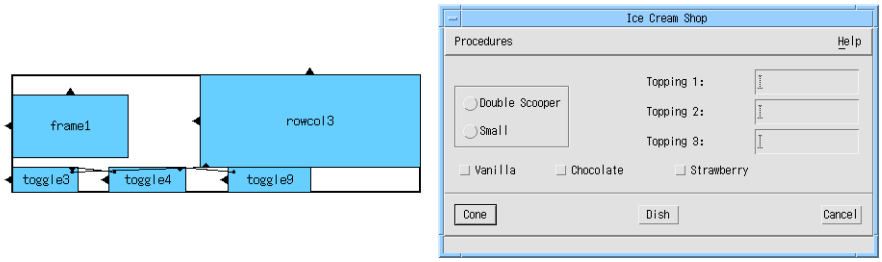


図 4-18　3 個のトグルボタンの整列後の配置

グループ整列が設定するアタッチメントの方向

ウィジェットのグループの位置を揃える場合、選択された最後のウィジェットは影響を受けずに、そのウィジェットに合わせて他のウィジェットの位置が揃えられます。最後に選択するウィジェットを除いて、ウィジェットの選択順序は意味を持ちません。Sun WorkShop Visual は、ウィジェットの位置の順序でアタッチメントを設定します。最後に選択されるウィジェット以外の、グループ内の各ウィジェットは、その隣のウィジェットに接続され、すべてのアタッチメントは最後に選択されたウィジェットの方向を向いています。例で使用する配置においては、右のウィジェットは中央のウィジェットに、中央のウィジェットは左のウィジェットに接続されています。

図 4-19 は、この一般的な規則を示したものです。ウィジェットのグループが最初の図で示されている順序で選択された場合は、結果のアタッチメントは 2 番目の図に示されている順序でウィジェットを接続します。

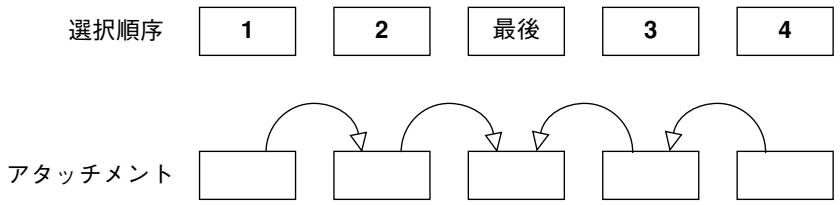


図 4-19 グループ整列により設定されるアタッチメントの方向

この規則は、左または右端に位置を揃えてあるウィジェットの列に対しても同様に作用します。

注 - 他のアタッチメントを設定する場合には、ウィジェットのグループの整列が循環しないように注意してください。一番上または左端のウィジェットを揃える場所に配置し、その後他のウィジェットを右から左へ、あるいは下から上へと位置を揃えていく方法が最も効果的です。

均等配置

「配置方法」メニューの「均等配置」プルライトメニューにある均等配置機能を使用すると、指定されたスペースにウィジェットのグループを等間隔で分散させることができます。この機能は、複数のウィジェットが選択されている場合にのみ使用することができます。ウィジェットの複数選択の詳細は、114 ページの「セレクション: 単独、一次、二次」を参照してください。

均等配置機能

「配置方法」メニューの「均等配置」を選択すると、プルライトメニューが表示され、そのメニューからさらにプルライトメニューが 2 つ表示されます。

水平

「水平」を選択すると、以下に示す項目を持つプルライトメニューが表示されます。

- 中央
  最も離れている 2 つのウィジェットの間で、ウィジェットの中心が水平方向に等間隔になるように配置されます。

- 端
  最も離れている 2 つのウィジェットの間で、ウィジェットの端と端の間が水平方向に等間隔になるように配置されます。

- 定数
  ウィジェットとウィジェットの水平方向の間隔が、「オフセット」フィールドで指定されている距離になるように配置します。「オフセット」フィールドが空である場合は、デフォルトのフォーム間隔が使用されます。このモードの場合、ウィジェットのグループが占めるスペースの合計が変化する場合があります。

均等配置するウィジェット同士のサイズが大きく違っていると、「中央」均等配置と「端」均等配置の違いが最も良く識別できます。

垂直

「垂直」を選択すると、以下に示す項目を持つプルライトメニューが表示されます。

- 中央
  上端と下端にある 2 つのウィジェットの間で、ウィジェットの中心が垂直方向に等間隔になるように配置されます。

- 端
  上端と下端にある 2 個のウィジェットの間で、ウィジェットの端と端の間が垂直方向に等間隔になるように配置されます。

- 定数
  ウィジェットとウィジェットの垂直方向の間隔が、「オフセット」フィールドで指定されている距離になるように配置します。「オフセット」フィールドが空である場合は、デフォルトのフォーム間隔が使用されます。このモードの場合、ウィジェットのグループが占めるスペースの合計が変化する場合があります。

水平均等配置の場合と同じように、均等配置するウィジェット同士のサイズが大きく違っていると、「中央」均等配置と「端」均等配置の違いが最も良く識別できます。

均等配置の使用方法

この機能を使用して、フォームの一番下の部分に 3 個のトグルボタンを均等に配置することができます。

1. 右側のトグルボタンをクリックします。

   ウィジェットが前セクションで選択されたままの状態である場合、新しいウィジェットをクリックして新たにセレクショングループを作成すると、選択が解除されます。

2. 中央のトグルボタンを <Shift-Button 1> でクリックします。

3. 左側のトグルボタンを <Shift-Button 1> でクリックします。

   これで、3 個のボタンがすべて強調表示されました。最後に選択されたトグルボタンが一次セレクション (ウィジェットの枠が強調表示されているもの) になります。114 ページの「セレクション: 単独、一次、二次」を参照してください。ただし、「均等配置」機能の場合は選択された順序は関係がありません。

   ボタンの端と端の間を等間隔にして、水平方向に配置します。

4. 「配置方法」メニューのプルライトメニューから「均等配置」を選択します。

5. 次に表示されるプルライトメニューから「水平方向」を選択します。

6. 「端」を選択します。

   この例ではウィジェットのサイズが同じなので、「中心」を選択しても同様の結果になります。

   図 4-20 に結果を示します。

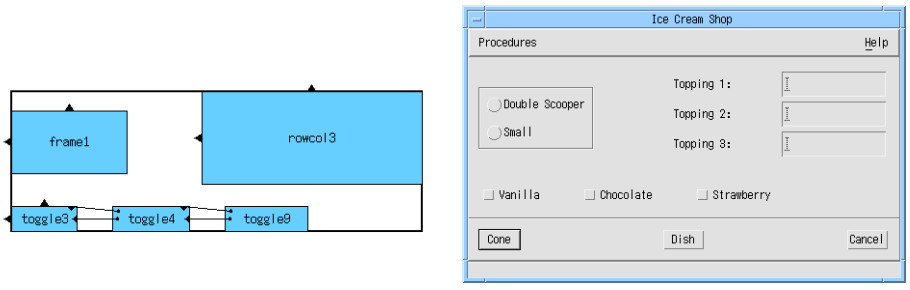


図 4-20 ウィジェットの端の均等配置

均等配置で設定されるアタッチメントの方向

「均等配置」では、グループ整列 (「配置方法」メニューから「整列」を選択する) とは異なる順序でアタッチメントが設定されます。「均等配置」の場合、アタッチメントは図 4-21 に示されるように、常に下から上、または右から左という特別な順序で作成されます。それぞれのウィジェットは、隣のウィジェットに接続されます。「均等配置」の場合は「整列」とは異なり、ウィジェットを選択した順序が何らかの影響を及ぼすことはありません。

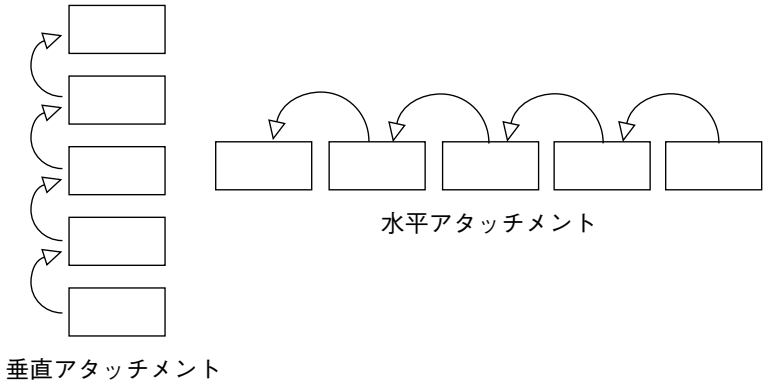


図 4-21　「均等配置」が設定したアタッチメントの方向

配置エディタの画面では、トグルボタンの端から端を接続している矢印はすべて右から左の方向を向いています。「表示」メニューをプルダウンして一時的にクラス名注釈を画面から取り除くと、矢印がもっとはっきりと見えます。

循環を防ぐためのヒント

「均等配置」は前述の通りにアタッチメントを設定するだけなので、あらかじめ計画をたてておかないと容易に循環アタッチメントが生じてしまいます。たとえば、すでに実行してきたように、右から左へウィジェットを選択し、それらを「整列」し、さらに「均等配置」することができます。しかし、それらを左から右に選択し、「整列」を行い、そして同じウィジェットのグループに「均等配置」を行うことはできません。これを実行すると、循環アタッチメントが生じます。

循環を避けるためには、その他のすべてのアタッチメントを右から左または下から上に作成してください。

また、「整列」および「均等配置」によって設定されたアタッチメントと既存のアタッチメントの間に、矛盾が発生することがあります。詳細は、120 ページの「移動中のアタッチメントの削除」を参照してください。

均等配置: 位置モード

位置アタッチメントを使用すると、ウィジェットの端をフォームの合計幅または高さのパーセンテージで表わされる位置に接続することができます。この機能により、ウィンドウがサイズ変更された場合に、インタフェース内のウィジェットが余分なスペースを利用することができます。位置は常にフォームの左上隅から測ります。ウィジェットの上または下端に *n* % の位置アタッチメントがある場合には、その端はフォームの上部から *n* % 下方の位置に配置されます。ウィジェットの左または右端に *n* % の位置アタッチメントがある場合には、その端はフォームの左端から *n* % 右に配置されます。

位置は「位置」フィールドにパーセント値として指定されます。ユーザーが値を入力しない場合には、値はゼロと見なされます。位置アタッチメントは、ウィジェットのアタッチメントされた端に、塗りつぶされていない円で表示されます。

まず、現在のウィンドウ動作を実証します。

1. **現在の幅よりも広くなるようにウィンドウのサイズを変更します。**

   図 4-22 に示されるように、ローカラムはフレームとの間に同じ間隔を保ちます。

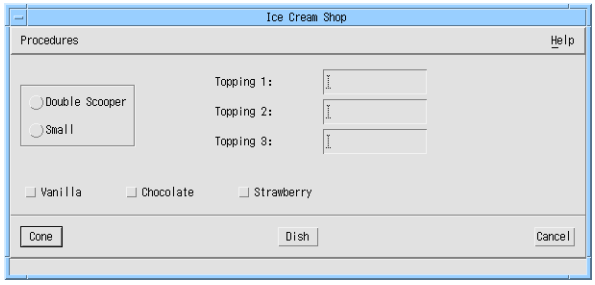


図 4-22 固定オフセットアタッチメントを持つローカラムの動作 (ウィンドウをサイズ変更した場合)

固定オフセットを使用したアタッチメントを設定すると、このように動作します。生じた余分なウィンドウ幅には、ウィジェットは配置されません。このサイズ変更動作は、多くのインタフェースに共通しています。ただし、位置アタッチメントを使用して、利用可能なウィンドウスペースを活用するようにローカラムを移動させることができます。

これを行うには、次のようにして、ローカラムの左端の 45 % の位置アタッチメントを指定します。

2. 「位置」トグルをクリックします。
3. 「位置」テキストボックスをダブルクリックして、次のように入力します。

   45

4. ローカラムの左端のすぐ内側にポインタを配置してその端を強調表示し、クリックします。

   位置アタッチメントがローカラムの端に白ぬきの円として表示されます。このアタッチメントは既存のアタッチメントとそれを表わす矢印を上書きします。この型のアタッチメントは、ウィンドウサイズとは関係なく、ローカラムの左端をフォームから 45 % 離れた場所に配置します。この効果を確認するには次のようにします。

5. ウィンドウの幅を狭くするようにサイズ変更を行い、次に幅を広げます。

   図 4-23 に示すように、ローカラムは余分なスペースを埋めるために移動しています。このサイズ変更動作が、位置コンストレイントの大きな利点です。

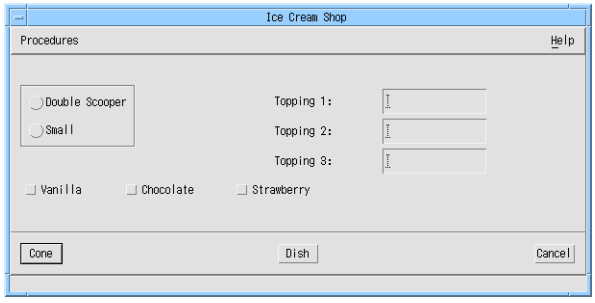


図 4-23 位置アタッチメントを持つローカラムの動作 (ウィンドウをサイズ変更した場合)

この型のアタッチメントの短所は、位置アタッチメントはフォームのサイズによってのみ計算されるものであり、その他のウィジェットのサイズに合わせた調整は行わないということです。これは、図 4-24 に示すように、他のウィジェットがサイズ変更した場合には問題です。フレーム内のラベルの 1 つが長くなった場合、フレームはローカラムに近づいてしまう、あるいは重なってしまう可能性があります。

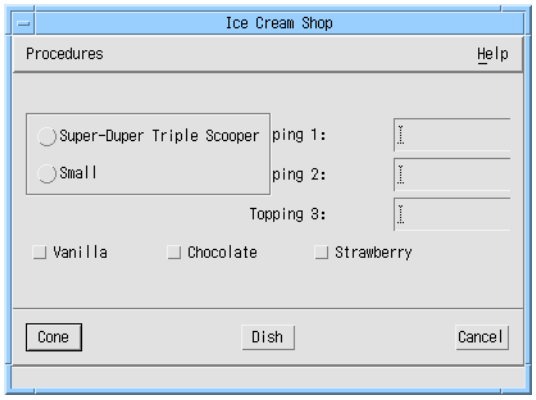


図 4-24 位置アタッチメントを持つローカラムの動作 (別のウィジェットがサイズ変更した場合)

この動作と、50 ピクセルのオフセットを持つウィジェットアタッチメントのある配置が示されている図 4-10 を比較してください。アタッチメントの型は、配置内のウィジェットの型と実行時に行われる可能性のある変更にもとづいてユーザーが選択します。配置の方法については、655 ページの第 20 章「複雑な配置」に詳しく説明されています。

自己モード

「自己」モードは、位置アタッチメントを設定するためのもう 1 つの方法です。パーセント値を入力する代わりに、ウィジェットの端の 1 つをクリックすると、ウィジェットの現在位置とフォームの現在のサイズにもとづいてパーセンテージが計算されます。そのため、「自己」を使用する場合には、「位置」フィールドにパーセンテージを指定する必要がありません。また、フィールドに指定されている値は無視されます。しかし、「移動」あるいは他のコマンドを使用して、ウィジェットを希望する場所に移動する必要があります。

「自己」は、「均等配置」と組み合わせて使用する場合に特に効果的です。固定した間隔を使用してボタンをフォームの下部にすでに設定してあります。各ボタンの両端に「自己」アタッチメントを設定すると、使用できる可能性のある余分なウィンドウスペースにボタンを配置させる一方で、均等なスペースを維持することができます。

デフォルトでは、「自己」アタッチメントはグリッドの間隔で設定されます。したがって、「均等配置」によって設定される詳細な間隔を活用するためには、グリッドを無効にする必要があります。

1. グリッドスライダを 0 に設定します。
2. 「自己」トグルをクリックします。

   「自己」モードの場合、位置はフォームの全体サイズに相対して計算されます。ウィンドウのサイズを変更したため、以下の操作を行います。

3. フォームをリセットします。

   フォームをリセットすると、その子に現在設定されているアタッチメントにもとづいて、最適なサイズが計算されます。

4. 右側のトグルボタンの右端をクリックします。

   「自己」アタッチメントがトグルボタンの端に塗りつぶされた円として表示されます。

5. 右側のトグルボタンの左端をクリックします。
6. 中央のトグルボタンの右端をクリックし、その後左端をクリックします。
7. 左側のトグルボタンの右端をクリックし、その後左端をクリックします。
8. フォームをリセットします。

   「自己」アタッチメントが塗りつぶされた円として表示されます。Motif では、「自己」アタッチメントは位置アタッチメントと同じであるため、Sun WorkShop Visual ではユーザーがフォームをリセットした後にその両者の区別を付けることができません。フォームのリセットを行なった場合には、「自己」アタッチメントは塗りつぶされていない円で表示され、位置アタッチメントとまったく同様に動作します。フォームリセット後のトグルボタンの配置は図 4-25 のようになります。

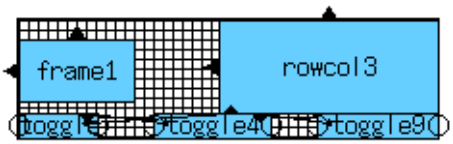


図 4-25 フォームリセット後の「自己」アタッチメント

9. デザインを保存します。

これで、本章の学習セクションは完了です。この章の残りの部分では、その他の配置機能を説明します。

サイズ変更モード

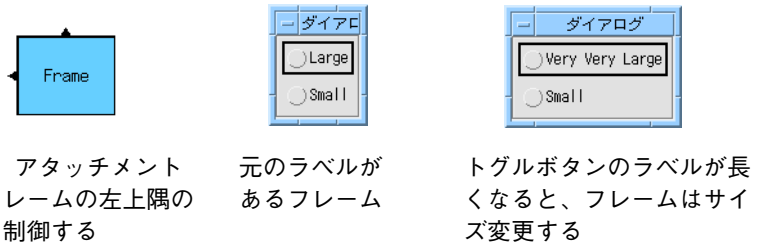
「サイズ変更」は、「移動」と同様に機能しますが、ウィジェットの下部および右側にアタッチメントを設定します。「サイズ変更」を使用するためには、以下の操作を行います。

1. 「サイズ変更」トグルをクリックします。
2. マウスボタン 1 を使用して、ウィジェットの右下隅を希望の場所にドラッグします。

「サイズ変更」は、ウィジェットのサイズを固定する場合に、ブリテンボードおよび描画領域で使用すると役に立ちます。フォームの場合には、「サイズ変更」はウィジェットの右下隅を、配置ウィジェットの左上隅に相対する特定の *x*、*y* 位置に接続します。ウィジェットの左上隅のアタッチメントと組み合わされる場合には、ウィジェットのサイズが固定されます。ブリテンボードまたは描画領域の場合は、「サイズ変更」は単純にウィジェットの幅および高さのリソースを設定します。

ウィジェットに独自の最適なサイズを計算させる場合には、ほとんどのウィジェットがそれぞれ最適な動作を行うため、このオプションは、通常はフォーム配置には使用されません。

図 4-26 に、ウィジェットが独自にサイズ変更を行う代表的な例を示します。



「移動」アタッチメントは、フレームの左上隅の位置を制御する

元のラベルがあるフレーム

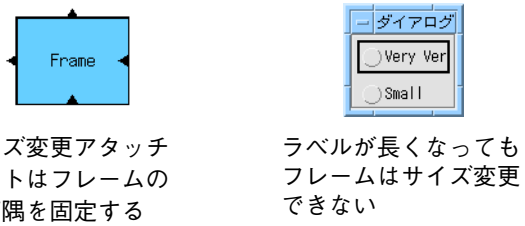
トグルボタンのラベルが長くなると、フレームはサイズ変更する

図 4-26　フレームウィジェットのサイズ変更動作

図 4-26 のフレームは、「移動」コマンドのみによって制約されます。ユーザーがトグルボタンの 1 つに対してラベルテキストを変更する場合、フレームの下部および右側は制約されていないため、自由にフレーム自体でサイズ変更することができます。

しかし、フレームが「サイズ変更」によって設定されているアタッチメントを持つ場合には、

図 4-27 に示すように、サイズ変更を行うことはできません。



サイズ変更アタッチメントはフレームの右下隅を固定する

ラベルが長くなってもフレームはサイズ変更できない

図 4-27　移動およびサイズ変更アタッチメントを一緒に使用した場合の結果

図 4-27 に示されている「移動」および「サイズ変更」の組み合わせはフレームの 4 個の端すべてを固定するため、大きくなったラベルを収めるために拡大することはできません。Motif では、ラベル文字列の一部だけを表示することにより、この状況に対処しています。

「サイズ変更」は、ブリテンボードおよび描画領域ウィジェットが位置アタッチメントまたはウィジェットアタッチメントを提供しないため、主にこれらのウィジェットで使用します。「サイズ変更」は、ウィジェットに特定のサイズを維持するように強制して、ウィジェット同士が重なりあわないようにします。「サイズ変更」の短所は、自動サイズ変更の利点が使用できなくなってしまうことです。サイズを変更する

可能性のあるウィジェットに最適な動作をさせるためには、フォームを使用して、ウィジェットを互いに接続します。すると、ウィジェットの 1 つがサイズ変更した場合に、他のウィジェットがその変更に対応して移動します。

コンストレイントパネルの使用

前章で説明したコンストレイントパネルを使用して、フォームの子にあるアタッチメントの表示および調整を行うことができます。コンストレイントパネルは、アタッチメントの表示および微調整に対してのみ、その使用をお勧めします。コンストレイントパネルは、ウィジェットから起因するアタッチメントのみを表示することに注意してください。大幅な変更を行う場合には、配置エディタを使用します。

1. 配置エディタ画面の「ファイル」メニューから「閉じる」を選択します。
2. 構成領域のローカラムを選択します。
3. ウィジェットメニューをプルダウンし、「コンストレイント」を選択します。

このコマンドにより、図 4-28 に示されるように、ローカラムに設定されているアタッチメントが表示されます。

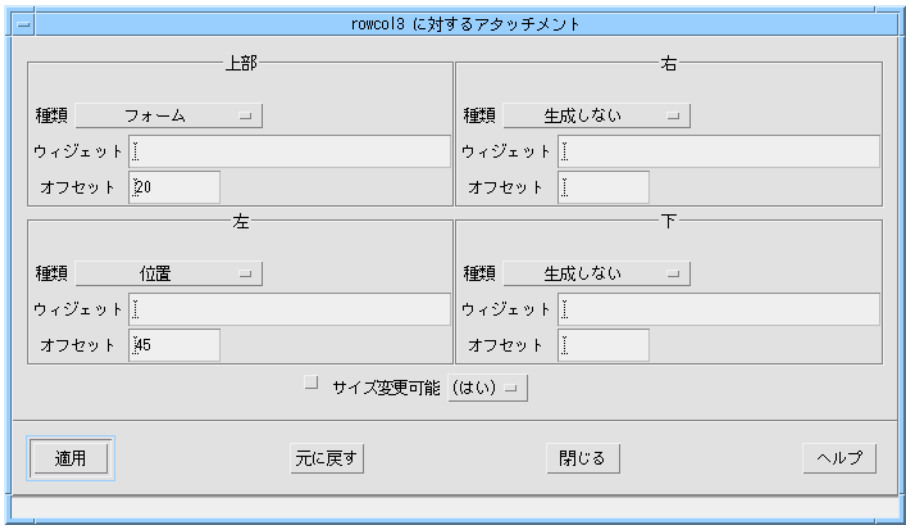


図 4-28 ローカラムのコンストレイントパネル

ローカラムの上部には、オフセット 0 のフォームへのアタッチメントがあります。フォームの左端には、45 % の位置アタッチメントがあります。ローカラムの下部および右側にはアタッチメントはありません。

コンストレイントパネルは、以下の作業に使用することができます。

- フォームの子にあるアタッチメントの表示
- ウィジェットの各端がアタッチメント先となっているウィジェット名の表示
- デフォルトオフセット (括弧内に表示) と明示的オフセットとの区別
- 「種類」オプションメニューからの選択による「フォーム」、「位置」、あるいは「ウィジェット」といったアタッチメントの型の変更
- 「種類」メニューから「なし」を選択することによる、アタッチメントの削除 (ウィジェットが *x* および *y* に最低 1 個のアタッチメントを持っている場合)
- 「オフセット」フィールドへの新しい値の入力による、オフセットまたは位置の調整
- フォームの子をサイズ変更した場合に、フォームもサイズ変更するかどうかの指定この指定は「サイズ変更可能」オプションで行います。「なし」を選択すると、親ウィジェットであるフォームのサイズは、子ウィジェットがサイズ変更された場合も、変更されません。

4. コンストレイントパネルでの変更を有効にするためには、「適用」をクリックします。

その他の配置ウィジェット

ブリテンボードウィジェット、描画領域ウィジェット、ローカラムウィジェットも、配置エディタ上で使用することができます。

ブリテンボードと描画領域

配置エディタは、フォームの場合と同様にブリテンボードおよび描画領域にも使用することができます。配置エディタを他の配置ウィジェットで使用する場合には、いくつかの相違点があります。以下の記述は、ブリテンボードおよび描画領域にのみ該当します。

すでに説明したように、ウィジェット、自己および位置アタッチメントは使用することができません。配置エディタはアタッチメントを示す矢印または三角形の表示を行わず、配置されたウィジェットの位置およびサイズのみを表示します。

配置エディタを最初に表示する場合、いくつかのウィジェットが最初から直接重ねられて配置されていることがあります。この場合には、「移動」を繰り返して使用し、それらのウィジェットを重ならない位置にドラッグします。

メインウィンドウで使用できる編集モードは、「移動」および「サイズ変更」のみです。また、「グループ整列」と「均等配置」は使用できますが、「整列」モードは使用することができません。「グループ整列」および「均等配置」は、ウィジェットを相互に接続せず、単に配置をしなおします。

ウィジェットは相互に接続することができないため、循環アタッチメントが生じる恐れはありません。

内部的には、配置エディタはフォームの場合のように、ブリテンボードおよび描画領域のリソースの設定は行いません。その代わり、子ウィジェットのコアサイズと位置リソースを設定することにより配置を決定します。コンストレイントパネルは、これらの配置ウィジェットには使用できません。サイズおよび位置リソースを表示するためには、子ウィジェットに対してのコアリソースパネルを表示します。

ローカラム

ローカラムはマネージャウィジェットなので、配置エディタを呼び出すことができます。ただし、使用できる機能は「サイズ変更」だけです。それ以外の機能をローカラムに適用することはできません。

配置エディタの制限事項

次の条件に該当するウィジェットは、配置エディタ上で移動、またはコンストレイントの設定を行うことはできません。

- マネージされていない場合
- 定義のインスタンスの一部で、かつユーザーに定義のインスタンスへのアクセス権がない場合 (詳細は、316 ページの「インスタンスの変更と拡張」を参照してください)

ウィジェットをマネージするには、コアリソースパネルの「コード生成」ページにある「マネージ」トグルをオンにします。

第5章

その他のエディタ

はじめに

Sun WorkShop Visual には、頻繁に行う作業が簡単に実行できるように特殊なエディタが用意されています。これらのエディタを使用して、以下の作業を実行することができます。

- 色を設定する
- テキスト文字列のフォントを設定する
- ラベルに、テキストではなくビットマップまたはピックスマップを使用する
- ピックスマップを作成する
- コンパウンド文字列を作成および編集する
- コールバックとプレリュードを編集する

本章では、これらのエディタをすべて使用して学習用インタフェースをカスタマイズします。エディタはウィジェットリソースパネルから呼び出され、編集結果はリソースパネルのそれぞれのフィールドに追加されます。また、「ツール」メニューからカラー選択パネル、ピックスマップ・エディタ、フォント選択パネルを表示することもできます。これらのエディタは、選択した任意のウィジェットに対して個別に使用することができます。

色の設定

前景色および背景色はすべてのウィジェットの基本要素であるため、これらのリソースはコアリソースパネルに配置されています。階層内のウィジェットに対してこれらの色を設定するには、次の手順に従って操作を行います。

1. 階層内で「Strawberry」トグルボタンアイコンを選択します。
2. ツールバー上の「コアリソース」ボタンをクリック、あるいはウィジェットメニューをプルダウンして、「コアリソース」をクリックします。

   前章では、リソースパネルの右側のテキストボックスに直接リソースの設定値を入力しました。しかし、本章で説明するエディタを使用するには、コアリソースパネルの左側にあるボタンをクリックします。
3. 「表示」ページで「前景色」プッシュボタンをクリックします。

   図 5-1 に示されるようなカラー選択パネルが表示されます。

図 5-1　カラー選択パネル

X カラーリストからの選択

「X カラー」には、すでに名前の付いた色が多数表示されています。初めてインタフェースの色を選択する場合には、これらの標準の色を使用することをお勧めします。

1. スクロールリストに表示されている X カラーを下方にスクロールします。
2. 色を選択します。

   色を選択するときは、現在カラー選択エディタ上で選ばれている色がダイアログの右上部に表示されます。左側にはウィジェットの現在の色が表示されるので、2 つの色を比較することができます。

maroon (くり色) のような暗い赤みがかった色を選択します。これらの色は、リストの中程にまとめられています。選択された色はカラー選択パネルの上部に表示されます。

3. カラー選択の「適用」をクリックします。

   これにより、選択した色がコアリソースパネル上の「前景色」リソースに適用されます。これをウィジェットに適用するには、以下の操作を行う必要があります。

4. コアリソースパネルの「適用」をクリックします。

   ダイナミックディスプレイでは「Strawberry」トグルボタンの色が変化します。前景色の代わりに背景色が変わって見える場合がありますが、心配する必要はありません。これは、階層内でウィジェットが選択されているために、前景色と背景色を入れ替えることによってその選択をダイナミックディスプレイで反映させているためです。

   本当の色を確認するためには以下の操作を行います。

5. 階層内でシェルを選択します。

   これで、ダイナミックディスプレイには「Strawberry」トグルボタンの正しい前景色ならびに背景色が表示されます。

色の構成要素の使用

X カラーリストからの選択は、色を指定する方法の 1 つにすぎません。別の方法として、構成要素を使用して色を作成する、というものがあります。次は、この方法を使用して「Strawberry」トグルボタンの背景色を設定します。

1. 階層内で「Strawberry」トグルボタンを選択します。
2. コアリソースパネルで、「背景色」をクリックします。
3. スライダを使用して、色を変更します。

   カラー選択の上部にあるスライダを使用すると、色の三原色である赤、緑、青の要素、色相、彩度および明度を個々に制御することができます。変更は、カラー選択パネルの上部に即座に反映されます。「カラー名」テキストボックスには、選択した値を連結した値が表示されます。

これは背景色であるため、淡い (彩度の低い) 色を使用すると良いでしょう。明るい色を使用することにより、暗めの色を使用しているラベルとのコントラストがはっきりします。

4. 希望通りの色になったら、155 ページの「X カラーリストからの選択」の手順 3と手順 4 を繰り返してその色を適用します。

次のセクションへ進む前に、カラー選択パネルで選択した色を変更しないでください。

カラーオブジェクト

視覚的効果の高いインタフェースを作成するためには、使用する色に一貫性を持たせることが大切です。Sun WorkShop Visual には、ユーザーの指定する名前に色を設定する、カラーオブジェクト機能があります。この機能を使用して、学習用インタフェースの中央部にあるすべてのウィジェットに同一の背景色を適用します。

1. 階層内で「Strawberry」トグルボタンを選択します。
2. コアリソースパネルで、「背景色」をクリックします。
3. カラー選択パネル内でカラーオブジェクトのリストの下にある「名前」テキストフィールドの中をクリックします。
4. 次のように入力します。
   background
5. 「バインド」をクリックします。

選択した色が「background」という名前のカラーオブジェクトに設定されます。「選択」ボックスには、background が山括弧で囲まれて表示されます。

6. カラー選択パネルおよびコアリソースパネルで「適用」をクリックします。

背景色リソースに対する設定をカラーオブジェクト名として入力すると、次の操作によって、その他のウィジェットにこの背景色を適用することができます。

7. 階層内で「Vanilla」トグルボタンを選択します。
8. コアリソースパネルで、「背景色」テキストフィールドをダブルクリックします。
9. 次のように入力します。
   <background>

角括弧を使用しなければいけないことに注意してください。オブジェクト名と文字列値は、角括弧により区別されます。たとえば、カラーオブジェクト名 *<red>* は、色としての *red* とは区別されます。

10. 「適用」をクリックします。

「Vanilla」トグルボタンの色が変更されます。次に、この背景色を「Chocolate」トグルボタンおよび同じ背景色を使用するその他のウィジェットに適用します。

カラーオブジェクトのバインド変更

カラーオブジェクトを使用する場合、そのカラーオブジェクトを参照する、すべてのウィジェットの色を容易に変更することができます。このため、さまざまな色を簡単に試すことができます。

1. カラー選択パネルで、「カラーオブジェクト」リストの「background」をクリックします。

カラー選択パネルの右上部に背景色が表示され、オブジェクトのリストの下にあるテキストフィールドに「background」が表示されます。

2. カラー選択パネルの上部にあるスライダを使用して色を変更します。
3. 「バインド」をクリックします。

カラーオブジェクトを参照するすべてのリソースが新しい色に変更され、その変更はただちにダイナミックディスプレイに反映されます。

新しい色の作成、作成した色のカラーオブジェクトへの設定、カラーオブジェクトのカラーリソースへの割り当てを実行してみてください。また、X カラーリストの色をカラーオブジェクトに割り当てることも可能です。これらの手順を繰り返すことにより、ユーザー独自のカラーパレットを構築することができます。カラーオブジェクトに割り当てる色は変更する場合もあるため、カラーオブジェクト名は色の名前ではなく、「background」のように機能の名前にします。

カラーオブジェクトはデザインファイルに保存されます。つまり、デザインファイルが異なっている場合には、違う色を保存するために同じ名前を使用できるということを意味します。たとえば、あるデザインではカラーオブジェクト「background」が黄色であり、別のデザインでは青であるかもしれません。しかし、同一のデザイン内では、「background」のようなオブジェクト名は常に同じ色を参照します。

「グローバル」にすると、複数のファイルでカラーオブジェクトを共有できます。コードを生成するときに適切なトグルを設定して、カラーオブジェクトのファイル間での共有を制御します。詳細は、256 ページの「大域オブジェクト関数」を参照してください。

フォントの設定

フォント選択パネルを使用すると、ウィジェットで表示するテキストのフォントスタイルおよびサイズを選択することができます。使用可能なフォントスタイルはシステムによって決定されるため、新しいフォントを作成することはできません。

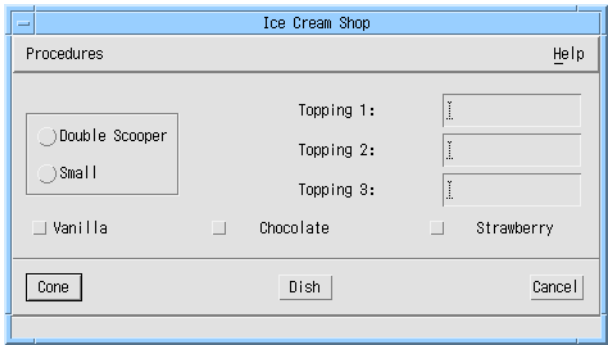
これまで、学習用インタフェース内のすべてのテキストにはデフォルトフォントを使用してきました。本セクションでは、フォント選択パネルを使用して以下の作業を行います。

- フォントを選択する
- 単一ウィジェットのフォントを設定する
- フォントオブジェクトを特定のフォントに設定する
- フォントオブジェクトを複数のウィジェットに適用する

本セクションでは、テキスト文字列での単一のフォントの使用について説明します。単一のラベル文字列に複数のフォントを使用することができる複合フォントオブジェクトについては、186 ページの「コンパウンド文字列」で説明しています。

フォントの選択

学習用インタフェースの視覚的効果を高めるために、図 5-2 に示すように、いくつかのフォントをデフォルトから斜体に変更します。



デフォルトのフォント

Oblique (斜体) フォント

図 5-2　フォント設定前および設定後のトグルボタン

まず、フォント選択パネルを表示します。

1. 「Double Scooper」ラジオボタンをダブルクリックして、リソースパネルを表示します。
2. リソースパネルの「表示」ページで、「フォント」をクリックします。

   図 5-3 に示されるフォント選択パネルが表示されます。

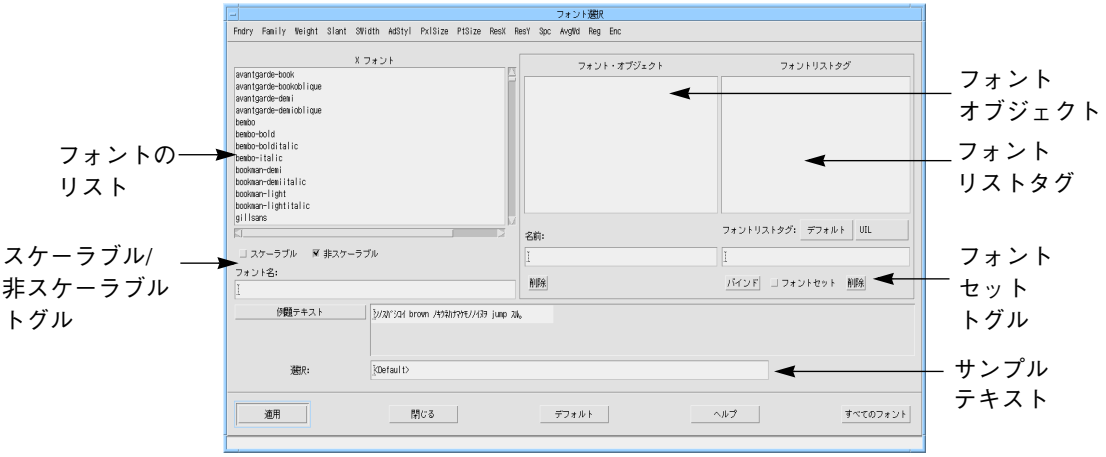


図 5-3　フォント選択パネル

フォント選択パネルの領域

フォント選択パネルは、システムで使用可能なすべてのフォントのリストを表示します。マシンによってインストールされているフォントが異なるため、表示されるリストが図とは異なる場合があります。

リスト内には非常に多くのフォントが存在します。これらのフォントは、メニューバーを使用してフォントファミリ、ウェート (線の太さ)、およびポイントサイズなどの基準ごとにフィルタすることができます。

フォントリストの下にあるトグルボタンを使用すると、スケーラブルフォント、非スケーラブルフォント、あるいはその両方を選択することができます。

リストからフォントを選択すると、「フォント名」および「選択」フィールドに名前が表示されます。

フォント選択パネルは、フォントオブジェクトおよびそれに関連付けられているフォントリストタグの一覧も表示します。単純なフォントオブジェクトは、フォント名の別名としても使用することができます。186 ページの「コンパウンド文字列」では、複合フォントオブジェクトの使用について説明します。

フォントリストのフィルタ

これから「Double Scooper」ラベルのフォントを 14 ポイントの太字 (bold) の斜体 (oblique)　Helvetica 書体に変更します (ワークステーションにこのフォントがインストールされていない場合には、別のフォントを選択してください)。リスト内には多数のフォントがあるため、条件を設定してリストをフィルタすると良いでしょう。「フォント名」および「選択」フィールドには、行われた変更が反映されます。

1. メニューバーから Family メニューをプルダウンして「Helvetica」を選択します。

   これにより、リストから Helvetica 以外のすべてのフォントが取り除かれます。

2. Weight メニューをプルダウンして「bold」を選択します。

3. Slant メニューをプルダウンして、「oblique」を表わす「o」を選択します。

4. PtSize (ポイントサイズ) メニューをプルダウンして「140」を選択します。

   ポイントサイズはポイントの 10 倍で指定されるため、これで 14 ポイントフォントを選択することになります。

   これらの操作を行うと、フォントリストの項目数は約 4 個に減少します。代表的なフォント名を図 5-4 に示します。1 つのフォントを指定するためには数多くのフィールドが必要とされますが、それらのフィールドのほとんどは高度な印刷にのみ使用されるものです。最も頻繁に使用する可能性があるフィールドは、Family、Weight、Slant および PtSize です。また、75 dpi (インチごとのドット数) または 100 dpi などの表示解像度の指定が必要な場合もあります。これらのフィールドを図 5-4 に示します。

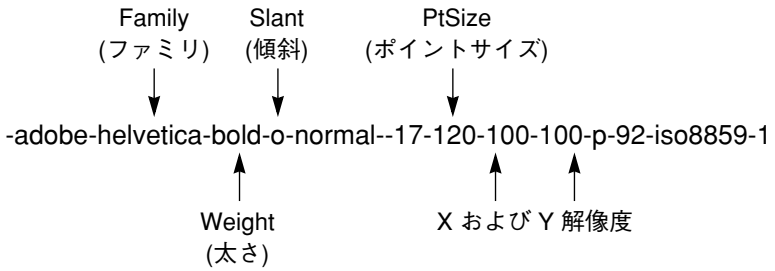


図 5-4　代表的なフォント名

図 5-4 の例では、水平および垂直の両方に 100 dpi を指定しています。これはほとんどのワークステーションディスプレイに適しています。

5. ResX メニューをプルダウンして、解像度 100 または 75 dpi を選択します。

   使用するディスプレイに適した解像度を選択してください。適した解像度がわからない場合は 100 dpi を使用してください。

フォントの適用

これまで行なった操作で、インタフェースのフォントを指定することができました。次は、このフォントを「Double Scooper」ラジオボタンに適用します。

1. フォント選択パネルの「適用」をクリックします。

   リソースパネルの「フォント」リソースに選択したフォントが適用されます。

2. リソースパネルの「適用」をクリックします。

   「Double Scooper」ラジオボタンには、大きめで太字の斜体フォントのラベルが付きます。

3. 「Small」ラジオボタンを選択します。

4. フォント選択パネルの PtSize メニューをプルダウンして、「100」を選択します。

   フィルタリングが変更された場合には、「サンプルテキスト」ボタンを押して、テキスト例を表示しなおすことができます。フォントによっては、読み込みに大変長い時間を要するものがあるため、Sun WorkShop Visual はテキスト例の再表示を自動的には行いません。

5. フォント選択パネルおよびリソースパネルで「適用」をクリックします。

   「Small」ラジオボタンには、「Double Scooper」ラベルよりも小さめの太字傾斜フォントのラベルが付けられます。先に進む前に、以下の操作でフォント選択パネルをリセットしてすべてのフォントを表示します。

6. フォント選択パネルの下部にある「すべてのフォント」をクリックします。

   フォントフィルタのすべての要素は「*」にリセットされ、使用可能なすべてのフォントが表示されます。

スケーラブルフォントラジオボタンで使用するフォントは非スケーラブルフォントです。非スケーラブルフォントとは、固定されたポイントサイズでのみ使用できるフォントのことです。X は、任意のサイズで表示できるスケーラブルフォントもサポートしています。スケーラブルフォントを選択するには、以下の操作を行います。

1. フォント選択パネルで「非スケーラブル」トグルをクリアします。

   リストからすべての非スケーラブルフォントが取り除かれ、リストが空になります。

2. 「スケーラブル」トグルを設定します。

   リストにスケーラブルフォントが追加されます。これらのフォントは、ピクセルサイズまたはポイントサイズを調整することによってサイズを指定することができます。ピクセルサイズは、フォント記述子の最初の数字フィールドであり、ポイントサイズは、2 番目の数字フィールドです。これらのフィールドは両方とも、すべてのスケーラブルフォントに対して最初 (デフォルトサイズ) は 0 に設定されています。

3. 「Vanilla」トグルボタンを選択します。

4. リスト内でフォント名の 1 つをクリックします。

   選択したフォントは、「フォント名」、「選択」、「サンプルテキスト」フィールドに表示されます。

5. 「フォント名」フィールド (「選択」フィールドではない) をクリックします。

6. 2 番目の数字フィールド (ポイントサイズフィールド) を編集して 240 (24 ポイント) にします。

7. Return キー を押すか、「サンプルテキスト」ボタンをクリックして、そのテキストサイズの例を表示します。

   スケーラブルフォントを使用する場合、ピクセルサイズまたはポイントサイズのどちらか (両方ではありません) に対して 0 以外の値を指定します。0 が指定されているフィールドは、明示的に指定された値に合わせて調整されます。両方のフィールドに 0 以外の値を指定した場合、それらの値が一致しないとエラーメッセージが表示されます。

   ポイントサイズ値 (ポイントの 10 倍で指定) は、通常ピクセルサイズ値よりも大きな値です。ポイントサイズ 240 は、30 から 35 の間のピクセルサイズに相当します。正確な比率は、画面の解像度と特定のフォントにより異なります。

Sun WorkShop Visual にはアウトラインスケールとビットスケールの 2 種類のスケーラブルフォントがあります。サンプルテキストがギザギザになっている場合、そのフォントはビットスケールです。アウトラインスケールのフォントのみをリストするためには、Fndry メニューをプルダウンして「bitstream」を選択します。

動作を確認し、フォント選択パネルを非スケーラブルフォントに設定しなおします。

8. 「非スケーラブル」トグルをクリックします。

単純フォントオブジェクト

これまで、2 個のラジオボタンに対して、個々にフォントの設定を行いました。この方法で複数のウィジェットに同じフォントを設定する場合、後で変更を行う際にはそれぞれのウィジェットに対して個別に変更作業を行う必要があります。また、Sun WorkShop Visual がアプリケーションに対して生成したコードは、各ラベルに対して同じフォントをロードするために個別にシステムの呼び出しを実行することになり、効率が悪くなります。

したがって、複数のウィジェットが同一のフォントを使用する場合には、単純フォントオブジェクトを作成します。すると、プログラミングと保守の両方を容易に行うことができます。フォントオブジェクトは、フォントのリストの別名です。単純フォントオブジェクトは、1 要素からなるリストの別名です。

1. メニューを使用して 12 ポイントの bold oblique helvetica フォントを選択します。

   X フォントのリストに複数のフォントが表示される場合には、その中の 1 つを選択します。

2. フォントオブジェクトのリストの下にある「名前」フィールドに次のように入力します。

   `option_labels`

   フォントオブジェクトの指定は変更される場合があるため、その名前には、サイズやスタイルではなく機能名を付けるようにします。

3. 「バインド」をクリックします。

   「option_labels」という名前のフォントオブジェクトが作成されます。この中のリストには、12 ポイントの helvetica フォント 1 つだけしか含まれていません。これには、フォントリストタグ <default> が関連付けられています。フォントリストタグの使用については、186 ページの「コンパウンド文字列」で説明します。

「選択」フィールドは自動的に更新を行なって名前「<option_labels>」を表示します。山括弧 (<>) は、それがフォント名ではなくフォントオブジェクトであることを示します。このフォントは、「適用」をクリックした場合にリソースに適用されます。この段階ではまだ適用しないでください。後ですべてのトグルボタンに対して、一括してフォントオブジェクトを適用します。

4. 3 個のトグルボタン「Vanilla」「Chocolate」「Strawberry」および 3 個のラベル「Topping1」「Topping2」「Topping3」をすべて選択します。

   これを行うには、ウィジェットを囲むように矩形をドラッグするか、ウィジェットを 1 つ選択してから、Shift キーを押したまま残りの 2 つを選択します。

5. フォント選択パネルで「適用」をクリックします。

   フォントオブジェクトがトグルボタンに適用されます。

6. リソースパネルで「適用」をクリックします。

   フォントオブジェクトが、現在選択されているウィジェットすべてに適用されます。

   現段階で、インタフェースは図 5-5 のようになります。

図 5-5　フォントを適用したインタフェース

フォントオブジェクトの変更

トグルボタンおよびラベルのテキストは、フォントオブジェクトが設定されたフォントで表示されています。このフォントスタイルは、フォントオブジェクトを別のフォントに設定するだけで変更できます。

1. Slant メニューをプルダウンして「regular」を表わす「r」を選択します。

2. Family メニューをプルダウンして「Times」を選択します。

   フォントリストは 12 ポイントの太字 Times フォントを表示します。リストが空の場合は、異なるフォントファミリを選択します。

3. 「バインド」をクリックします。

   フォントオブジェクトが、Times フォントに対応して変更され、すべてのラベルおよびトグルボタンはダイナミックディスプレイ上で即座に変更されます。

   フォントオブジェクトはデザインファイルと一緒に保存されます。これは、異なるデザインファイルでは異なるフォントを持つフォントオブジェクトに対しても、同じ名前が使用できることを意味します。たとえば、あるデザインではフォントオブジェクト *option_labels* が Helvetica であり、別のデザインでは同じ名前のフォントオブジェクトが Times である場合もあります。

   フォントオブジェクトを「グローバル」にすると、複数のファイルで共有できます。コードを生成するときに適切なトグルを設定して、フォントオブジェクトのファイル間での共有を制御します。詳細は、256 ページの「大域オブジェクト関数」 を参照してください。

ピックスマップの選択

ラベルおよびボタンには、テキスト文字列の代わりにピックスマップを使用することができます。Sun WorkShop Visual では、ピックスマップの作成および適用を行うための 2 種類のエディタを備えています。まず最初に、ピックスマップ選択パネルを使用して、ピックスマップのウィジェットへの適用の基礎を学習します。その後、ピックスマップエディタを使用してカスタムピックスマップをいくつか作成します。

Sun WorkShop Visual では、標準 X ビットマップエディタを使用して作成されたビットマップを使用することができます。また、パブリックドメインの *Xpm* 形式のピックスマップを使用することが可能で、その形式のピックスマップを構築するためのエディタも含まれています。*Xpm* ライブラリは、Sun WorkShop Visual に含まれています。

ビットマップとピックスマップ

Xpm 形式のピックスマップは、その他のどのユーティリティで作成されたものでも使用することができます。ビットマップとはモノクロで Xbm 形式のもの、ピックスマップとはカラーで *Xpm* 形式のものを指します。ただし、Sun WorkShop Visual ピックスマップエディタを使用することで X ビットマップを編集することもできます。この編集を行う場合、Sun WorkShop Visual はビットマップを 2 色のピックスマップに変換し、*Xpm* 形式で描画します。このピックスマップを 2 色のままにしておくこともできますが、他の色を追加することもできます。

ビットマップの選択

まず最初に、トグルボタンのラベルを X ビットマップのうちの 1 つに置き換えます。

1. 階層内で「Cone」プッシュボタンをダブルクリックします。
2. リソースパネルの「表示」ページで、「ピックスマップ」をクリックします。

Sun WorkShop Visual は、ピックスマップ選択パネルを表示します。図 5-6 にピックスマップ選択パネルの例を示します。

図 5-6　ピックスマップ選択パネルの例

3. X ビットマップから任意のビットマップを選択します。

   選択されたビットマップは、ピックスマップウィンドウの上部に表示されます。

4. ピックスマップウィンドウで、「適用」をクリックします。

   選択した内容がリソースパネルの「ピックスマップ」リソースに適用されます。

5. リソースパネルにおいて、「適用」をクリックします。

   ダイナミックディスプレイにはテキストラベルだけが表示されますが、この段階では、トグルボタンはテキストラベルとピックスマップラベルの両方を持っています。ピックスマップラベルを表示するためには、表示するラベルの種類を制御するリソースを変更する必要があります。

6. リソースパネルの「設定」ページで、「種類」の設定を「ピックスマップ」に変更します。

7. リソースパネルで「適用」をクリックします。

トグルボタンはテキストラベルの代わりにピックスマップを表示します。この場合、ピックスマップは役に立つ情報を含んでいないため、「種類」リソースを「文字列」に戻します。

システムに別の X ビットマップファイルがある場合には、それらのファイルを使用することもできます。「開く」をクリックすると、ファイル選択パネルが表示されます。そのファイル選択パネルを使用すると、ビットマップファイルの検索および X ビットマップのリストへの追加を行うことができます。

また、X ビットマップのリストの下にあるテキストボックスにビットマップの名前を入力して「追加」をクリックすると、その名前をリストに追加できます。ビットマップがまだ存在していない場合でも、その名前をリストに追加してリソースに適用することができます。開発時あるいは実行時など、後からビットマップを供給することができます。

ピックスマップの選択

ピックスマップ選択パネルの右側のリストは、現在定義されているピックスマップオブジェクトのリストです。これらは、オブジェクトとしてバインドされたピックスマップです。リスト内の名前は、オブジェクトの名前です。新規のピックスマップオブジェクトに命名し、次にピックスマップを作成するか既存のピックスマップまたはビットマップを読み取ることができます。オブジェクトを新規に作成するには、リストの下にあるテキストボックスに名前を入力します。ピックスマップを作成したオブジェクトに対応させるには、「編集」ボタンを押します。ピックスマップの読み取り方法または作成方法については、170 ページの「ピックスマップの編集」で説明します。

ピックスマップの編集

Sun WorkShop Visual では、ピックスマップ作成のためのエディタを提供しています。ピックスマップエディタを使用すると、独自のピックスマップを作成する以外にも、以下の作業を行うことができます。

- ピックスマップをピックスマップ・オブジェクトに設定する
- ピックスマップまたはピックスマップのファイル名をピックスマップエディタの構成領域にドラッグする

- Xpm 形式のピックスマップファイルを読み込む
- X ビットマップファイルを読み込み、それを編集用にピックスマップに変換する
- Xpm 形式でピックスマップをファイルに書き込む

ピックスマップを Xpm 1 形式に書き込むか Xpm 3 形式に書き込むかを選択できます。Xpm 3 形式が最新なので、この形式を使用してください。Xpm 1 形式 は、古いピックスマップ操作アプリケーションとの互換性を保証するために提供されています。

Sun WorkShop Visual 上のウィジェットで使用されるすべてのピックスマップは、ピックスマップ・オブジェクトに設定されていなければなりません。まず、図 5-7 に示すように、「Cone」プッシュボタンに対してのピックスマップを作成します。

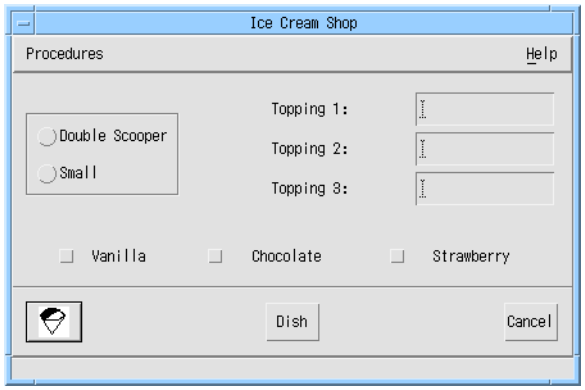


図 5-7　ピックスマップボタンのある学習用インタフェース

167 ページの「ピックスマップの選択」から作業を続けている場合には、すでにピックスマップ選択パネルが表示されているため、手順 1 と手順 2 は省くことができます。

1. 階層内で「Cone」プッシュボタンをダブルクリックします。
2. リソースパネルの「表示」ページで、「ピックスマップ」をクリックします。
3. ピックスマップ選択パネルの「ピックスマップ・オブジェクト」部にあるテキストボックスに、次のように入力します。

   cone
4. 「編集」ボタンをクリックします。

図 5-8 に示すようなピックスマップ・エディタが表示されます。

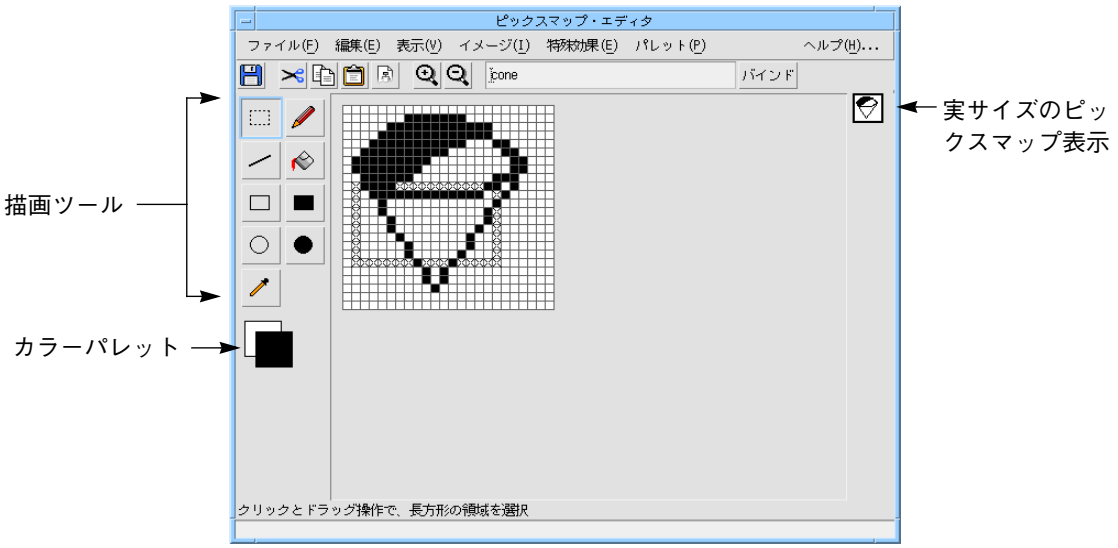


図 5-8　ピックスマップエディタ

「ピックスマップ・エディタ」ウィンドウの最上部にはメニューバーがあり、その下にはツールバーがあります。ウィンドウの左側にはツールパレットがあり、現在設定されている前景色と背景色がツールパレットの下に表示されています。中央には描画領域があり、その右側には作成したピックスマップが実サイズで表示されます。

ピックスマップ・エディタのツールバー

図 5-9 に示すツールバーには、次のような操作を行うためのボタンがあります。

- 開く
  173 ページの「ピックスマップ・エディタのファイルメニュー」にあるようにピックスマップファイルを開きます。
- カット
  ピックスマップの選択領域をカットします。174 ページの「ピックスマップ・エディタの編集メニュー」を参照してください。

- コピー
  ピックスマップの選択領域をコピーします。174 ページの「ピックスマップ・エディタの編集メニュー」を参照してください。
- ペースト
  クリップボードの内容をピックスマップにペーストします。174 ページの「ピックスマップ・エディタの編集メニュー」を参照してください。
- 消去
  ピックスマップの選択領域を消去します。174 ページの「ピックスマップ・エディタの編集メニュー」を参照してください。
- ズームイン/ズームアウト
  ズームインはイメージ全体を拡大し、ズームアウトはイメージを縮小して表示します。これらは表示オプションであり、ピックスマップには影響しません。

図 5-9　ピックスマップ・エディタのツールバー

ピックスマップ・エディタのファイルメニュー

ファイルメニューには、「新規」、「ファイルを開く」、「XPM ファイルを保存」、「閉じる」の 4 つのオプションがあります。

新規

新規は、これまでのイメージをすべて削除して、空白の編集領域を新規に作成します。変更後にイメージをバインドしていない場合は、警告メッセージが表示されます。イメージをバインドしたい場合は、185 ページの「ピックスマップ・オブジェクト」で説明するように「取消し」を選択してからイメージをバインドします。イメージをバインドしたくない場合は、「了解」を押して処理を続行します。

ファイルを開く

システム上に X ピックスマップファイルがある場合は、それらのファイルを自分のインタフェースに使用することができます。「ファイル」メニューから「ファイルを開く」を選択すると、ファイル選択パネルが表示され、ピックスマップ・エディタにピックスマップファイルを読み込むことができます。Sun WorkShop Visual は、ピックスマップを XPM 3 形式で読み取ります。また、X ビットマップファイルを読み取り、それをピックスマップ・エディタで編集するためのピックスマップへの変換も行います。

XPM ファイルを保存

ピックスマップ・エディタの「ファイル」メニューから「XPM ファイルを保存」を選択すると、作成したピックスマップをファイルに書き込むことができます。

Sun WorkShop Visual は、XPM 1 形式および XPM 3 形式で書き込みを行います。通常は、ピックスマップは XPM 3 形式で保存します。XPM 1 は、他社製のピックスマップ処理ユーティリティの旧バージョンとの互換性を持たせるために用意されています。

閉じる

エディタウィンドウを閉じます。最後にバインドしてから変更を加えた場合は、警告メッセージが表示されます。イメージをオブジェクトにバインドする方法については、185 ページの「ピックスマップ・オブジェクト」を参照してください。

作業の保存

XPM ファイルへのバインドまたは書き込みを行うごとに、作成中のピックスマップの現在の状態を保存してください。作業中には、頻繁に保存を行うことをお勧めします。

ピックスマップ・エディタの編集メニュー

「編集」メニューには、「元に戻す」、「カット」、「コピー」、「ペースト」、「消去」、「切り取り」、「すべて選択」の 7 つの項目があります。

「編集」メニューのほとんどのオプションは、ピックスマップの選択された部分に作用します。選択を行うには、図 5-10 に示すツールパレット上の選択ツールを使用します。

図 5-10 選択ツール

1. 選択ツールをクリックします。
2. 描画領域をクリックしてドラッグします。

矩形フレーム内の×印でマークされたピクセルおよびフレーム内すべてのピクセルが選択されます。イメージの一部を選択した場合は、その領域の寸法と位置が、ウィンドウ下部のステータス行に表示されます。「編集」メニューの「すべて選択」オプションを選択すると、イメージ全体を選択することができます。

元に戻す

編集領域での直前のアクションを元に戻します。

カット

イメージの選択部分を削除してクリップボードに移動します。

コピー

イメージの選択部分をクリップボードにコピーします。

ペースト

クリップボードの内容を選択領域にペーストします。選択領域がクリップボード上の領域よりも小さい場合は、クリップボードのイメージの左上を選択領域の左上に合わせ、クリップボードの領域を選択領域に収まる分だけ描画します。

消去

イメージの選択部分をクリップボードにコピーしないで削除します。

切り取り

選択領域を切り取り、その領域だけを表示します。

すべて選択

イメージ全体を選択します。

ピックスマップ・エディタの表示メニュー

「表示」メニューには、「ドラッグカラー」というラベルが付いたプルライトメニュー項目が 1 つあります。この項目は、ドラッグする際の選択矩形の色に関係します。この項目にマウスポインタをあわせて右側に移動させると、「反転」ラベルおよび「Xor」ラベルが付いた 2 個のラジオボタンが表示されます。これらは、選択矩形の表示方法を説明しています。

ピックスマップ・エディタのイメージメニュー

「イメージ」メニューには「サイズ変更」という項目が 1 つだけあります。「サイズ変更」を選択すると、図 5-11 に示すダイアログが表示されます。

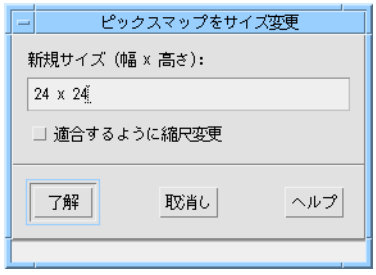


図 5-11 サイズ変更ダイアログ

このダイアログを使用して、ピックスマップエディタ上に表示されるピックスマップの大きさを変更することができます。

1. イメージに大きさ「幅 x 高さ」という形式で新規のサイズを入力します。

2. 「適合するように縮尺変更」トグルをオンにします。

ピックスマップエディタ上に表示されるイメージの大きさが、指定した寸法に適合するように変更されます。

ピックスマップ・エディタの特殊効果メニュー

「特殊効果」メニューには、「水平に反転」、「垂直に反転」、「180 度回転」、「グレー表示」の 4 つの項目があります。

反転

「水平に反転」では水平方向、「垂直に反転」では垂直方向の軸を中心に、選択領域の鏡像を作成します。

回転

「180 度回転」は、選択領域を 180 度回転します。すなわち、対角線を中心に選択領域の鏡像を作成します。

グレー表示

この項目は、選択領域の各ピクセルの「排他的論理和 (xor)」をとり、グレー表示の選択領域を作成します。これは、無効状態を示すピックスマップを作成する場合に便利です。

ピックスマップ・エディタのパレットメニュー

「パレット」メニューを使用すると、カラーパレットを編集したり、パレットを新規に読み取ることができます。「パレットを編集」および「パレットを読み取り」という 2 つの項目があります。

パレットを編集

180 ページの「カラーパレット」で説明するパレットエディタが呼び出されます。

パレットを読み取り

保存したピックスマップファイルのパレットを読み取ることができます。この項目を選択すると、ファイル選択ダイアログが表示され、*Xpm* 形式ファイルの名前の入力を求められます。ファイルが読み取られると、既存のパレットが保存したファイルの色のパレットに置き換わります。次に、Sun WorkShop Visual は現在のイメージがあればそのイメージが新規のパレットの色を使用するよう変更します。Sun WorkShop Visual は、可能な限り最も近い色合わせを使用します。

Sun WorkShop Visual にはいくつかのパレットがキューブファイル (cubennn 形式のファイル) として用意されています。また、Sun WorkShop Visual がアイコン用に使用するパレットも用意されています。これらのファイルは `$VISUROOT/lib/palettes` にあります。$VISUROOT は、Sun WorkShop Visual のインストールディレクトリです。

ツールパレット

イメージを描画したり色付けする手助けとして、図 5-12 に示すようなツール用のパレットが、ピックスマップ・エディタのウィンドウの左側にあります。

図 5-12 ツールパレット

描画ツールはすべて、現在の前景色を使用します。ただし、前景色ではなく現在の背景色を使用することもできます。180 ページの「カラーパレット」を参照してください。また、イメージ内の特定の色のインスタンスをすべて変更することもできます。変更の方法については、180 ページの「カラーパレット」を参照してください。

ウィンドウ上で操作を行うには、以下の描画ツールをいずれか 1 つ選択する必要があります。ツールは常に 1 つだけ選択されている状態になっています。

これは選択ツールです。イメージの矩形領域を選択することができます。多くのメニュー機能は、選択領域に対して処理を行います。

これは矩形塗りつぶしツールです。このツールを選択すると、現在の前景色で塗りつぶされた矩形を描画することができます。

これは塗りつぶしツールです。塗りつぶされた領域上でマウスボタンをクリックすると、現在の前景色でその領域が塗りつぶされます。

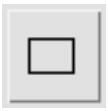
円の輪郭を前景色で描画することができます。円は中心から描画されます。

個々のピクセルを前景色で描画することができます。

このツールを選択すると、前景色で塗りつぶされた円を中心から描画することができます。

このツールを選択すると、前景色で直線を描画することができます。

これはドロッパーツールです。このツールを使用して、イメージから色を選択します。このツールを選択してイメージ内の色をクリックすると、その色が前景色として設定されます。これを簡単に設定する方法については、178 ページの「ツールパレット」を参照してください。



このツールを選択すると、矩形の輪郭を前景色で描画することができます。

ドロッパーツールの短縮操作

任意のツールを選択した状態で、Control キーを押したまま色をクリックすると、ドロッパーツールとまったく同じように、前景色が設定されます。Control キーを押したままマウスボタン 2 で色をクリックすると、背景色が設定されます。

編集領域へのドラッグ

Motif のドラッグ & ドロップ機能を使用して、ピックスマップおよびピックスマップファイル名を編集領域内にドラッグすることができます。これは通常、ソース上でマウスボタン 2 を押し、そのソースをピックスマップ・エディタの編集領域までドラッグしてからマウスボタンを離すことを意味します。この操作ができない場合は、システム管理者にシステムの構成を問い合わせてください。

ドラッグしたピックスマップは、自分で作成したピックスマップの場合と同様に、編集領域に表示されます。また、そのピックスマップで使用されている色をすべて含むカラーパレットも追加されます。

カラーパレット

ピックスマップ・エディタでは、カラーパレットは重要です。カラーピックスマップを作成するには、使用したい色を指定する必要があります。カラーパレットでピックスマップ用に定義された色は、そのピックスマップとともに保存されます。イメージの作成中は、パレット内のどの色が現在の背景色および前景色なのかを知っている必要があります。

背景色および前景色

ツールパレットの下にある 2 つの矩形の色が、現在の前景色および背景色を示します。前景色は、描画に使用される色です。背景色は、エディタが残したスペースを塗りつぶすために使用されます。「カット」および「消去」は、両方とも背景色で塗りつぶされたスペースを残します。これらの操作については、174 ページの「ピックスマップ・エディタの編集メニュー」を参照してください。「グレー表示」は背景色を使用してグレー表示効果を生成します。177 ページの「ピックスマップ・エディタの特殊効果メニュー」を参照してください。「サイズ変更」によりサイズを拡大した場合は、広がったスペースを背景色で塗りつぶします。

背景色と前景色の交換

ピックスマップ・エディタで描画する場合は、現在の前景色が使用されます。ただし、マウスボタン 2 を押したまま描画すると、現在の背景色が使用されます。

カラーパレットの表示

ポインタを背景色か前景色の四角の上に置いてマウスボタンを押すと、カラーパレットが表示されます。

カラーパレットを編集して、イメージで使用したい色を含めることができます。この方法については、180 ページの「カラーパレット」で説明します。

背景色および前景色の変更

現在の背景色および前景色を変更するには、カラーパレットを表示し、新しい色の上でマウスボタンを離します。

イメージ内の色の変更

カラーイメージがあり、特定の色のインスタンスをすべて別の色に変更したい場合は、以下のようにします。

1. 変更したい色が現在の前景色であることを確認します。
2. Control キーを押して、別の前景色を選択します。

   これで、イメージ内の前の色が表示されていた場所には、すべて新しい色が表示されます。

カラーパレットの編集

「パレット」メニューから「パレットを編集」を選択すると、図 5-13 のようなパレットエディタが表示されます。

図 5-13 パレットエディタ

色の付いた矩形はそれぞれがボタンになっています。色の付いた矩形を選択するには、その矩形の上でマウスボタンを押します。選択した色の名前がパレットエディタウィンドウの最下部に表示されます。

色の削除

「削除」ボタンは、現在選択されている色をパレットから削除します。その色がピックスマップ内で使用されていた場合は、その色の代わりに背景色が使用されます。

色の追加

「新規」ボタンを押すと、パレットの最後尾に色が追加されます。新しい色は、パレットの最後尾の矩形に表示されます。

色の編集

色を変更するには、その色をダブルクリックするか、またはその色を選択して「編集」を押します。これで、カラーエディタが呼び出されます。カラーエディタについては、154 ページの「色の設定」を参照してください。

透明色

選択した色に対して「透明」というラベルが付いたトグルを設定すると、その色はピックスマップでは「透明」になります。これは、最終アプリケーションでピックスマップが表示されるときに、ピックスマップの透明な領域にはその下にある色が表示されることを意味します。ボタンの背景色などをピックスマップを通して表示する場合に、このボタンを使用することができます。

ただし、透明色をサポートしているのは XPM です。Motif のプッシュボタンウィジェットおよびラベルウィジェットは、透明色をサポートしていません。XPM ライブラリを使用して色「none」をピックスマップを表示しているウィジェットの背景色に変換することができます。XPM ライブラリの使用方法については、以下のファイルを参照してください。

```
$VISUROOT/contrib/xpm/doc/xpm.ps
```

$VISUROOT は Sun WorkShop Visual のインストールディレクトリです。

透明色は、他社のウィジェットの方がサポートが多いことがあります。

カラーパレットエディタでは、透明色に対して「none」という名前が表示されます。

カラーパレットの保存

カラーパレットは、対応するピックスマップと一緒に保存されます。つまり、カラーパレット用に特別にピックスマップを作成し、別の時にパレットにロードすることができます。Sun WorkShop Visualは、そのピクチャーで使用されている色の数に保存される色数を合わせます。カラーパレットを保存するためにピックスマップを作成する場合、そのパレットの色それぞれに 1 ピクセル対応させてください。

カラーパレットの読み込み

保存したピックスマップからカラーパレットを読み込む方法については、177 ページの「ピックスマップ・エディタのパレットメニュー」を参照してください。

注 - 多数の色を使用する場合は、787 ページの「対話的使用のためのオプション」で説明するように、コマンド行オプション -L を指定して Sun WorkShop Visual を実行してください。こうすると、Sun WorkShop Visual はユーザー専用のカラーマップを使用します。このオプションを指定しないと、多数の色を使用することができなくなります。

ピックスマップ・エディタの使用

まず、作成するピックスマップのサイズを設定します。

1. 「イメージ」メニューから「サイズ変更」を選択します。
2. サイズ変更ダイアログのテキストボックスに次のように入力します。

   ```
   40 x 40
   ```

3. 「了解」を押します。

   描画領域の実サイズのピックスマップ表示およびグリッドがサイズ変更されます。次に、描画を行います。

4. 前景色として黒を選択します。

   前景色を変更する方法については、180 ページの「カラーパレット」を参照してください。

5. 矩形塗りつぶしツールをクリックします。

   矩形塗りつぶしツールがどれか分からない場合は、178 ページの「ツールパレット」を参照してください。

6. 描画領域をクリックしてドラッグを行い、黒く塗りつぶされた矩形を作成します。

   描画領域の実サイズのピックスマップ表示が更新されて、描画したピックスマップが表示されます。

7. 描画ツールに慣れるまで、実際に使用して練習します。

   描画作業を実行する前に、編集メニューのオプションを学習し、カラーパレットに何色か追加しておくと後で役に立ちます。

8. カラーパレットに何色か追加します。

   カラーパレットの色が増えました。描画ツールを使用して、さらにいろいろと試してみてください。準備ができたら、ツールと色を使用してアイスクリームのコーンを表示するピックスマップを作成します。

9. アイスクリームのコーンを描画します。

ピックスマップ・オブジェクト

完成したピックスマップを表示するには、まずそのピックスマップをピックスマップ・オブジェクトに設定する必要があります。

1. ピックスマップエディタで、「バインド」フィールドに「cone」と入力します。
2. 「バインド」をクリックするか、「バインド」フィールドで Return キー を押します。

   ピックスマップ選択パネルでは、ピックスマップ・オブジェクト名がピックスマップ・オブジェクトのリストおよび選択ボックス内に表示されます。

3. ピックスマップ選択パネルで「適用」をクリックします。
4. リソースパネルで「適用」をクリックします。
5. 「設定」のページで、「種類」設定を「ピックスマップ」に変更します。
6. リソースパネルで「適用」をクリックします。

   ピックスマップ・オブジェクトは、カラーオブジェクトに似た動作を行います。また、複数の場所に同じピックスマップを使用することができます。ピックスマップに対して変更を行なった場合は、バインドを行うと即座にダイナミックディスプレイに反映されます。

   ピックスマップ・オブジェクトは、デザインファイルと一緒に保存されます。したがって、異なるデザインファイルでは、ピックスマップ・オブジェクトが異なっている場合であっても同じ名前を使用できることを意味します。たとえば、あるデザインではピックスマップ「*cancel*」が取り消しスタンプの描画であり、別のデザインでは国際禁止記号である場合があります。

   試しに「Dish」および「Cancel」ボタンに対してピックスマップ・オブジェクトを作成して追加しても良いでしょう。

   参考: 図 5-14 で示されている「Cancel」スタンプは、鉛筆ツールではなく、線ツールで作成されました。

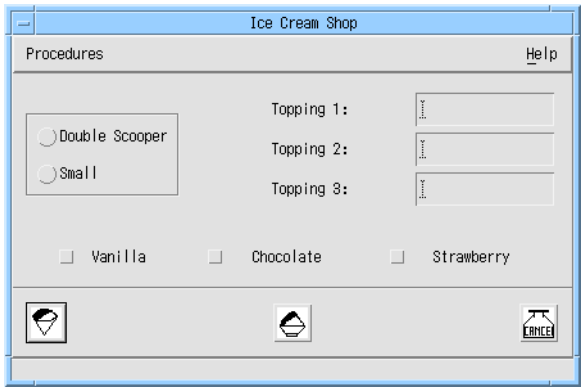


図 5-14 3 個のピックスマップボタンがある学習用インタフェース

7. デザインを保存します。

ピックスマップオブジェクトを「グローバル」にすると、複数のファイルで共有できます。コードを生成するときに適切なトグルを設定して、ピックスマップオブジェクトのファイル間での共有を制御します。詳細は、256 ページの「大域オブジェクト関数」 を参照してください。

コンパウンド文字列

学習例にあるラベルは、すべて単純なテキスト文字列を使用しています。本章では、内部構造を使用して複雑な文字列の作成を可能にする、Sun WorkShop Visual コンパウンド文字列エディタについて説明します。複合フォントオブジェクトとともに、これらの文字列を使用すると、複数のフォントを使用するラベル、あるいは全体または一部が右から左に書かれているラベルを表示することができます。

一般的に、文字列とは、順序付けられた EUC 文字のリストですが、文字列値を持つほとんどの Motif リソースは、異なる種類の文字列表現であるコンパウンド文字列を使用します。

Motif コンパウンド文字列は、テキストおよびテキストフィールドにある通常の文字列を除く、すべての文字列リソース値に対して使用されます。この場合、Motif コンパウンド文字列と X のコンパウンド・テキストフォーマットを混同しないように気を付けてください。Motif コンパウンド文字列は、その操作を行う Motif ツールキット関数の命名規則により、多くの場合は *XmString* と呼ばれます。

コンパウンド文字列は、コードを変更することなく、テキストを複数の言語およびフォントで表示できるようにエンコード (コード化) する手段です。ここでは、複数のフォントで表示される文字列の作成方法を説明します。複数の言語の使用については、705 ページの第 22 章「国際化」および Motif のマニュアルを参照してください。Motif のマニュアルの参考資料のリストについては、付録 E「参考資料」を参照してください。

概念上は、コンパウンド文字列は以下に示す 4 種類の構成要素から成ります。

- テキスト文字列 (バイト列)
- フォントリストタグ
  フォントリストタグは、以前は「文字セット」と呼ばれていたもので、この名称は現在も多くの Motif マニュアルで使用されています。
- 増加方向インジケータ: 右から左、あるいは左から右
- 改行区切り文字

これらの構成要素の型はどのような順序でも構いませんが、一般に、各テキスト文字列構成要素の前には、フォントリストタグ構成要素を置きます。

コンパウンド文字列をウィジェットのラベルに使用するには、ウィジェットのフォントリソースに対してフォントリストを指定します。フォントリストは、(フォント, タグ) の対の集合です。フォントリストタグは、各テキスト文字列構成要素に対して使用する、フォントリスト内のフォントを示します。

コンパウンド文字列エディタの機能に慣れるため、Sun WorkShop Visual をワークステーションで実行している時に、以下のような例を実行してください。

- **ファイルメニューから「新規」を選択します。**

この演習では、ラベルウィジェット上で、図 5-15 に示すように The Guardian という新聞名を再現してみます。

図 5-15 最終的なテキスト文字列の外観

複合フォントオブジェクトの作成

以上で、フォントを選択し、ウィジェットに適用する方法を学習しました。また、1 つのフォントに対応する簡単なフォントオブジェクトも作成しました。複合フォントオブジェクトを使用すると、さらに複合視覚的効果が得られます。複合フォントオブジェクトは、フォントのリストに対応しています。文字列の別の部分をリストの異なるフォントを使用して表示することができます。本節の段階的な例では、複合フォントオブジェクトを使用します。

ラベルを使用するウィジェットを作成し、そのウィジェットのリストに複数のフォントが含まれるフォントオブジェクトを指定します。

1. フォームの子としてラベルを持つウィジェット階層を作成します。
2. ラベルウィジェットをダブルクリックして、リソースパネルを表示します。
3. 「フォント」リソースボタンをクリックして、フォント選択パネルを表示します。
4. メニューを使用して、24 ポイントの medium italic Times フォントを選択します。
5. 「フォント・オブジェクト」フィールドの「名前」フィールドに次のように入力します。

   masthead
6. 「フォント・オブジェクト」フィールド右の「フォントリストタグ」フィールドに、次のように入力します。

   italic
7. 「バインド」をクリックします。

リストに 1 つのフォントを持つ「*masthead*」というフォントオブジェクトが作成されます。このフォントには、「*italic*」とタグが付けられます。

注 - タグ名は任意に付けることができます。ただし、「フォントリストタグ」フィールドの上にある「デフォルト」ボタンおよび「UIL」メニューで、事前定義されているいくつかのタグを選択することができます。「デフォルト」オプションを選択すると、タグ *XmFONTLIST_DEFAULT_TAG* が作成されます。UIL メニューには、UIL (Motif のユーザーインタフェース言語) で使用できるタグが提供されています。UIL をご使用の場合にリストにはないタグを使用すると、UIL コンパイラの多くが重大な内部エラーを引き起こすので、注意してください。UIL ユーザーでなければ、UIL オプションは無視してください。

次に、フォントオブジェクトに別のフォントを追加します。2 番目のフォントには、異なるタグを持たせます。

8. 24 ポイントの bold 標準 Helvetica フォントを選択します。
9. 「フォントリストタグ」フィールドに、次のように入力します。

   bold

   masthead フォントオブジェクト名は変更してはなりません。
10. 「バインド」をクリックします。

    これにより、フォントオブジェクトリストに bold フォントが追加され、そのフォントに bold というタグが付けられました。これらの 2 個のフォントは、「フォントリストタグ」リスト内でそれぞれのタグを選択すると、「サンプルテキスト」フィールドに表示されます。

    この時点で、2 個の異なるフォントを使用した文字列の表示に使用できる、フォントオブジェクトが作成されました。このフォントオブジェクトをラベルウィジェットに適用します。
11. フォント選択パネルの「適用」をクリックします。
12. ラベルリソースパネル内の「適用」をクリックします。

    ラベル内のテキストは、斜体の Times フォントで表示されます。これは、フォントがリストの最初にあるためです。両方のフォントを使用してテキスト文字列を表示するためには、コンパウンド文字列を作成する必要があります。

コンパウンド文字列の作成

リソースパネルの「ラベル」テキストフィールドにテキストを入力する場合、テキスト文字列構成要素と、改行に対応する区切り文字構成要素だけを含むコンパウンド文字列が作成されます。フォントリストタグを含むコンパウンド文字列を作成するためには、コンパウンド文字列エディタを使用する必要があります。

1. リソースパネル内の「ラベル」ボタンをクリックします。

   図 5-16 に示すようなコンパウンド文字列エディタが表示されます。

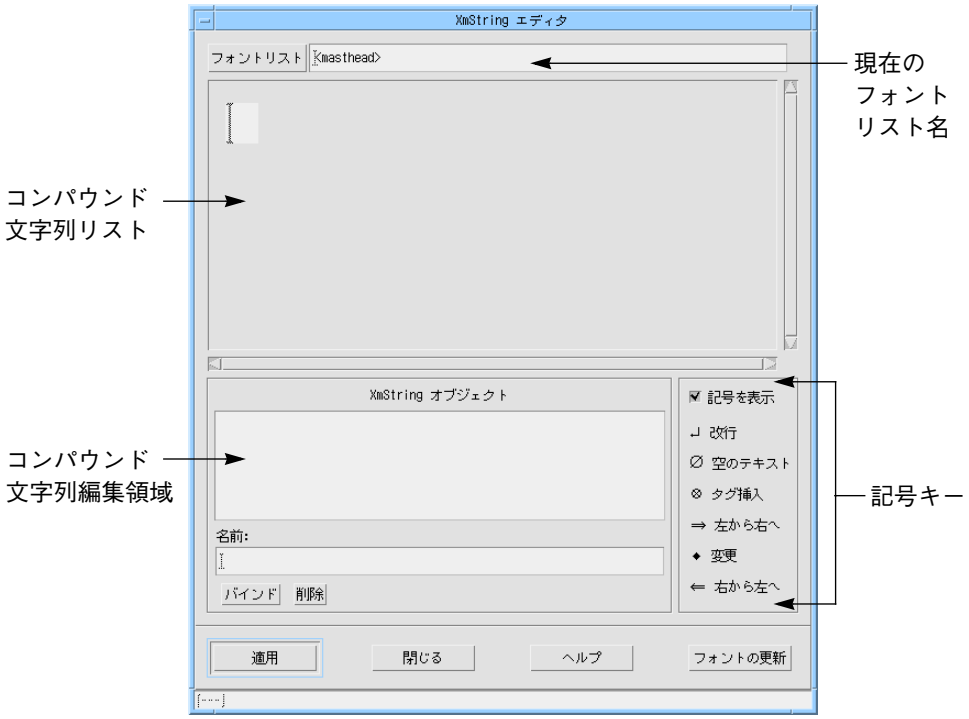


図 5-16 コンパウンド文字列エディタ

コンパウンド文字列エディタは、現在のフォントリスト名、編集領域、既存のコンパウンド文字列オブジェクトのリストを含んでいます。フォントリスト名は、現在ウィジェットで使用しているフォントオブジェクト名に対応させるため、「masthead」となっています。

コンパウンド文字列にテキストを入力すると、入力したテキストは編集領域に表示されます。テキストの表示には、画面上部で指定されているフォントリストが使用されます。空のテキスト構成要素および増加方向インジケータなどその他の構成要素は、記号で表示されます。「記号を表示」トグルを使用して記号の表示をオフにすると、ウィジェット内でテキストがどのように見えるかを確認することができます。

「記号を表示」トグルがオンになっていない場合は、次のようにします。

2. 「記号を表示」トグルをクリックします。

コンパウンド文字列を作成するには、次のようにします。

3. 編集領域内で I 型カーソルをクリックします。

カーソルが点滅してエディタがテキスト入力に対して準備ができていることを示します。

4. ワード間にスペースを置かずに、以下の文字列を入力します。

```
TheGuardian
```

テキストは、図 5-17 に示すように、フォントリストの最初のフォントである斜体の Times フォントを使用して表示されます。

図 5-17 コンパウンド文字列の初期テキスト

テキスト表示を部分ごとに異なるフォントにするためには、コンパウンド文字列にフォントリストタグを挿入する必要があります。

5. ポインタを The と Guardian の間に配置し、マウスボタン 3 を押します。

ポップアップメニューが表示されます。このメニューから項目を選択すると、文字列に対応する型の構成要素が挿入されます。「削除」を選択すると、現在の構成要素が削除されます。Guardian という単語を bold の Helvetica フォントで表示するためには、ポインタ位置に適切なフォントタグ (bold) を挿入する必要があります。

6. 「フォントリストタグ」メニューをプルライトして、bold を選択します。

フォントリストタグを示す記号が挿入され、図 5-18 に示すように表示が変更されます。

図 5-18 フォントリストタグの付いたコンパウンド文字列

タグを誤った位置に挿入した場合は、マウスボタン 1 を使用してタグを移動することができます。

現時点では、The という単語は斜体フォントで正確に表示されていまが、これは Motif がデフォルトでフォントリスト内の最初のフォントを使用するためです。コンパウンド文字列を完全なものにするためには、文字列の先頭に斜体のタグを挿入します。

7. ポインタを The の前に配置し、マウスボタン 3 を押します。

8. 「フォントリストタグ」メニューをプルライトして italic を選択します。

   図 5-19 に示すように 2 番目のフォントリストタグが挿入されます。

   図 5-19　フォントリストタグの付いたコンパウンド文字列

   これでコンパウンド文字列が完成しました。ラベル上にどのように表示されるかを確認します。

9. 「記号を表示」トグルをクリックして記号の表示をオフにします。

   図 5-20 に表示すように、フォントリストタグを使用せずに文字列が再描画されます。

   図 5-20　記号表示のないコンパウンド文字列

コンパウンド文字列オブジェクトの作成

コンパウンド文字列オブジェクトを作成し、それを文字列に結び付けます。

1. XmString オブジェクトのリストの下にある「名前」テキストフィールドに、次のように入力します。

   guardian

2. 「バインド」をクリックします。

   オブジェクトの名前が *XmString* オブジェクトリストに表示されます。最後に、コンパウンド文字列オブジェクトをラベルに適用します。

3. コンパウンド文字列エディタで「適用」をクリックします。

   コンパウンド文字列オブジェクトがラベルリソースパネルに適用されます。

4. ラベルリソースパネルで「適用」をクリックします。

   ラベル内のテキストは、選択されたフォントでコンパウンド文字列を表示します。

増加方向インジケータ

これまで、この演習ではテキストとフォントリストタグという 2 つの構成要素について実例を示してきました。他の 2 つの構成要素は区切り文字として使用される改行と増加方向インジケータです。

改行は、テキスト内の行の分割を行います。テキストの入力中に *Return* キーを押す、あるいはメニューを使用して改行を挿入することができます。改行の挿入、移動および削除を行なってみる前に、作業がよくわかるように「記号を表示」トグルがオンであることを確認します。

増加方向インジケータを使用すると、左または右からの方向でテキストを作成することができます。デフォルトの増加方向は左から右となっていますが、文字列の任意の部分を右から左に表記するように指定することができます。

1. 「記号を表示」トグルがオフの場合は、「記号を表示」トグルをクリックして、記号を表示します。
2. Guar と dian の間にポインタを配置し、マウスボタン 3 を押します。
3. メニューをプルダウンして「右から左」を選択します。

コンパウンド文字列は、図 5-21 のように変更されます。

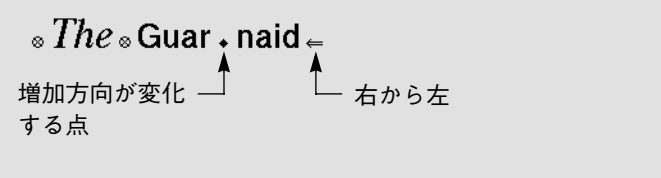


図 5-21 増加方向を変更したコンパウンド文字列

小さなひし形は、「左から右」への増加方向を「右から左」に変更することを表わしています。矢印は、増加方向インジケータであり、作用するテキストの開始部分に表示されます。これは「右から左」へのセグメントであるため、テキストの開始部分は左ではなく右側になります。「左から右」へ変更した場合は、新しいテキストの開始部分 (左端) に右側を向いている矢印で示されます。

「右から左」方向のセグメントにおいて改行やフォントリストタグを挿入すると、通常の読み取り方向が左から右である場合には、予想とは反対の効果を生じるように見える場合があります。

記号は、それに続くテキストに影響を与えます。つまり、「右から左」へのセグメントでは、左側にあるテキストに影響を与えますので注意してください。

最後に、コンパウンド文字列オブジェクト「guardian」を新しい文字列に結び付けます。

4. 「バインド」をクリックします。

図 5-22 に示すように、ラベルテキストが変更されます。

図 5-22 最終的なテキスト文字列の外観

文字列オブジェクトを「グローバル」にすると、複数のファイルで共有できます。コードを生成するときに適切なトグルを設定して、文字列オブジェクトのファイル間での共有を制御します。詳細は、256 ページの「大域オブジェクト関数」を参照してください。

フォントの変更の更新

新しいフォントオブジェクトを作成する場合、あるいはコンパウンド文字列エディタを使用している時に別のフォントオブジェクトを使用する場合、コンパウンド文字列エディタの「フォントの更新」ボタンを押すことにより、ウィジェットのリソースダイアログにある「フォント」フィールドを更新することができます。これにより、ウィジェットは確実に、エディタが使用しているフォントオブジェクトと同じものを使用します。フォントオブジェクトの内容を変更した場合は、この操作は必要ではありません。別のフォントオブジェクトを使用する場合にのみ、更新の操作が必要です。

コールバックおよびプレリュードの編集

Sun WorkShop Visual は、コールバックおよびコードプレリュードを編集する機能を提供しています。コールバックを指定してスタブファイルの生成を一度行うと、この機能を使用することができます。また、コードプレリュード編集の場合は、一度コードプレリュードを定義してメインのソースファイルを生成すると、この機能を使用することができます。

- コールバックの追加については、264 ページの「コールバック関数の追加」を参照してください。
- スタブファイルの生成については、248 ページの「スタブファイルの設定」を参照してください。
- コードプレリュードについては、280 ページの「生成されたファイルのカスタマイズ : プレリュード」を参照してください。
- メインのソースファイルの生成については、245 ページの「基本ソースファイルの設定」を参照してください。

まず、コールバックまたはプレリュードが宣言されているウィジェットを選択します。コールバックの場合は、編集機能は「コールバック」ダイアログから使用することができます。コードプレリュードの場合は、「ウィジェット」メニューから使用することができます。「コールバック」ダイアログについては、199 ページの「コールバックのデザイン」を参照してください。

コールバックまたはプレリュードを編集する場合は、Xterm ウィンドウ内で動作しているテキストエディタが使用できます。使用するエディタは、付録D「アプリケーションのデフォルト」で説明されている *editor* リソースのデフォルトによって決まります。

スタブファイルまたはメインのソースファイルをまだ作成していない場合は、作成するよう要求するメッセージが表示されます。

コールバック編集の詳細は、264 ページの「コールバック関数の追加」を参照してください。コードプレリュード編集の詳細は、282 ページの「コードプレリュード」で記述します。

第6章

インタフェースの起動 : ユーザーコードの追加

はじめに

Sun WorkShop Visual は、できる限りコードを作成することなくアプリケーションの開発ができるように設計されています。ただし、アプリケーション機能を実行し、それをユーザーインタフェースにリンクするためにはユーザー自身でコードを作成する必要があります。また、ユーザーインタフェースの動作を制御するためにもコードの作成が必要になります。Sun WorkShop Visual では、それらのコード生成作業が容易に行えるように、以下のような機能が用意されています。

- コールバック
  個々のウィジェットのコールバック関数またはコールバック・メソッドに名前を付けたり、スケルトン関数を生成することができます。
- リンク
  Sun WorkShop Visual は、典型的ないくつかのコールバックイベントを認識できます。それらのイベントをユーザーインタフェースを使用して追加し、その効果をすぐに確認することができます。
- ドラッグ & ドロップ
  ウィジェットが、Motif ドラッグ&ドロップ機能を経由してデータを受け取ることができるように、設定することができます。
- トランスレーションとアクション
  イベントに対する反応を、ウィジェットごとに変更することができます。

- Xt 手続き
  Xt 作業、入力、タイムアウト、アクション、言語の各手続きおよびイベントハンドラをデザインに簡単に追加できます。

本章では、前章までに構築した学習の例を使用して、デザインにコールバックとリンクを追加する方法について記述します。また、前述の各機能についても、それぞれ詳細に説明します。

コールバック

コールバックのリストとは、ユーザーアプリケーションでユーザーアクションにより起動されるように指定されている、1 つ以上のコールバック関数のリストです。
ユーザーのアクションには、マウスボタンの押下、キーボード選択およびポインタの移動があります。コールバックを設定すると、ウィジェット内で特定のユーザーアクションが行われるたびにインタフェースに指示を出して、コールバックリストにある関数を呼び出すことができます。図 6-1 にコールバックダイアログを示します。

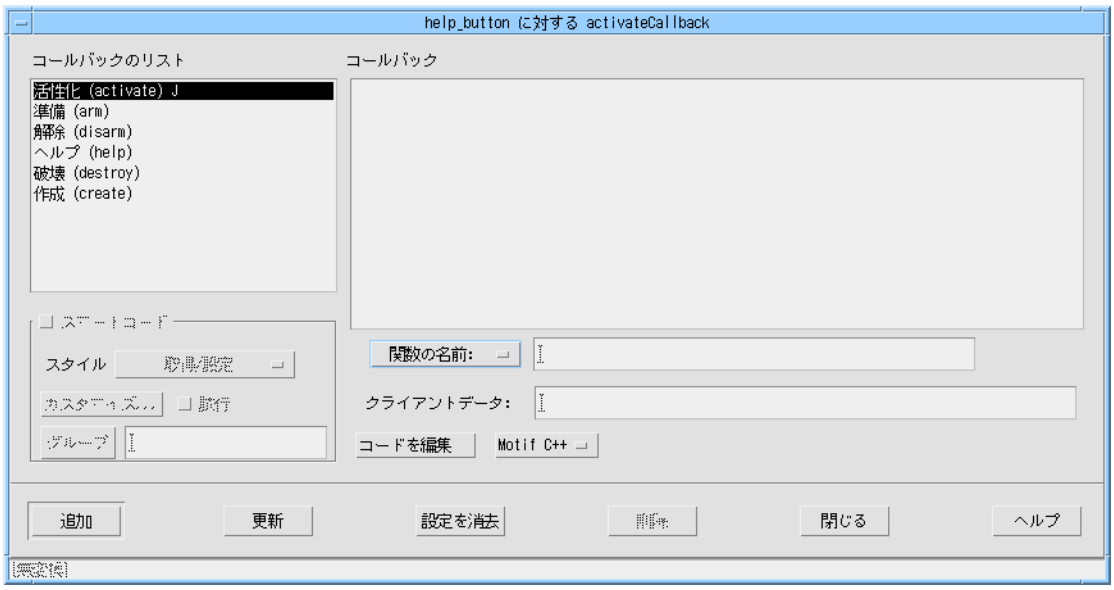


図 6-1　コールバックダイアログ

コールバックの右側に「M」が表示されている場合はメソッドが、「C」が表示されている場合はコールバックが宣言されていることを示します。

Java コードに適用されるコールバックには、図 6-1 に示したように、後に文字「J」が表示されます。コールバック関数は Java コードには変換されないため、Java アプリケーションにコールバック関数を含めたい場合は、コールバック・メソッドを使用してください。Sun WorkShop Visualを使用した Java コードの生成の詳細については、363 ページの第 10 章「Java 用のデザイン」を参照してください。

アスタリスク (*) は、そのコールバックが Microsoft Windows ではサポートされていないことを示します。

コールバックダイアログの左下の領域でスマートコードのコールバックを設定できます。これらのコールバックは、指定したウィジェットを別のコードの層で「包み込む」ことで、 ツールキットから独立させます。スマートコードは、thick クライアント、thin クライアントやサーバーの各アプリケーションをデザインから作成する場合に便利です。スマートコードについての詳細は、559 ページの第 16 章「取得と設定用のスマートコード」を参照してください。本章の学習セクションではスマートコードは使用しませんが、他の章の学習セクションには多くの使用例が用意されています。

コールバックのデザイン

次の手順では、これまでの章で作成した学習デザインを使用してコールバックの最も簡単な例を指定します。「Exit」ボタン (*exit_button*) をクリックすると、プログラムを終了する *quit()* という関数を 1 つだけ持つコールバックリストが活性化されます。

「コールバック」ダイアログを使用すると、関数のリストとユーザーアクションを関連付けることができます。*quit()* がまだ作成されていない場合であっても、*quit()* と *exit_button* をこの段階で関連付けることができます。

quit() およびその他のコールバック関数は、C または C++ で作成され、Sun WorkShop Visual で生成されたコードとリンクされます。この学習セクションの264 ページの「コールバック関数の追加」では、*quit()* 関数を作成します。コールバック作成に関しては、206 ページの「コールバック関数」で詳しく説明します。

1. 学習用デザインの保存ファイルを開きます。
2. 「Exit」ボタン (exit_button) をクリックします。
3. ツールバー上の「コールバック」ボタンを押すか、「ウィジェット」メニューから「コールバック」を選択します。

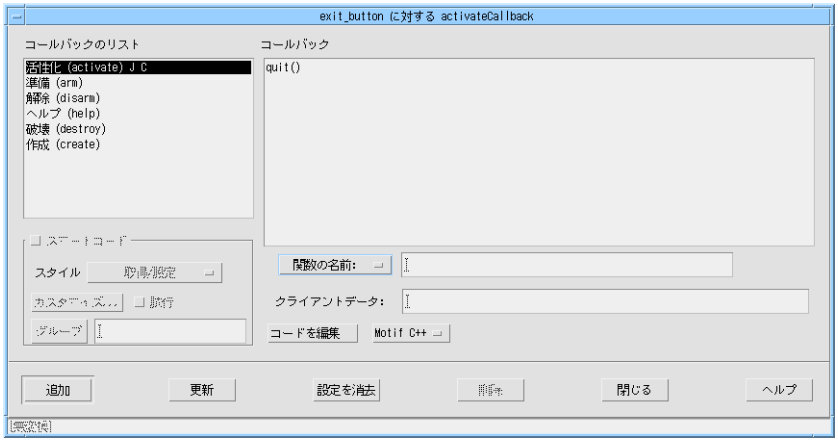
次に、*quit()* と「活性化 (activate)」を関連付けます。活性化とは、ウィジェット内にポインタを置いた状態で、ユーザーがマウスボタンを押して放すことを意味します。*<Return>* あるいは Motif ユーザーズガイドに記述されているその他のキーを使用しても活性化することができます。

「コールバック」ダイアログに表示されているコールバックを選択すると、そのコールバックに関連付けられているコールバック関数のリストが表示されます。活性化コールバックを追加するには、次のようにします。

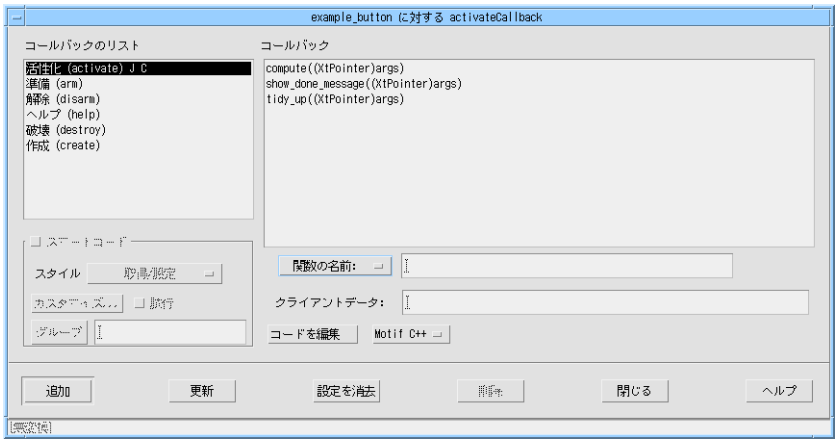
4. コールバックリストの中の「活性化 (activate)」をクリックします。

   コールバックルーチンのリストには、ウィジェットに局所的なコールバックと継承されたコールバック (以降、継承コールバックと呼ぶ) の両方を表示します。継承コールバックの詳細は、201 ページの「継承されたコールバック」を参照してください。

   継承されていないコールバックのみ、変更することができます。図 6-2 には、代表的な 2 つの例を示します。例 A は、学習用インタフェースの「Exit」ボタンのコールバックを、例 B は少し複雑な例を示しています。



例 A



例 B

図 6-2　コールバック・テキストボックスの構文例

継承されたコールバック

定義のインスタンスであるウィジェットの場合は、定義中の対応するウィジェットからコールバックを継承することができます。

図 6-3 に示すように、継承されたコールバックは角括弧 [ ] で囲まれて表示されます。

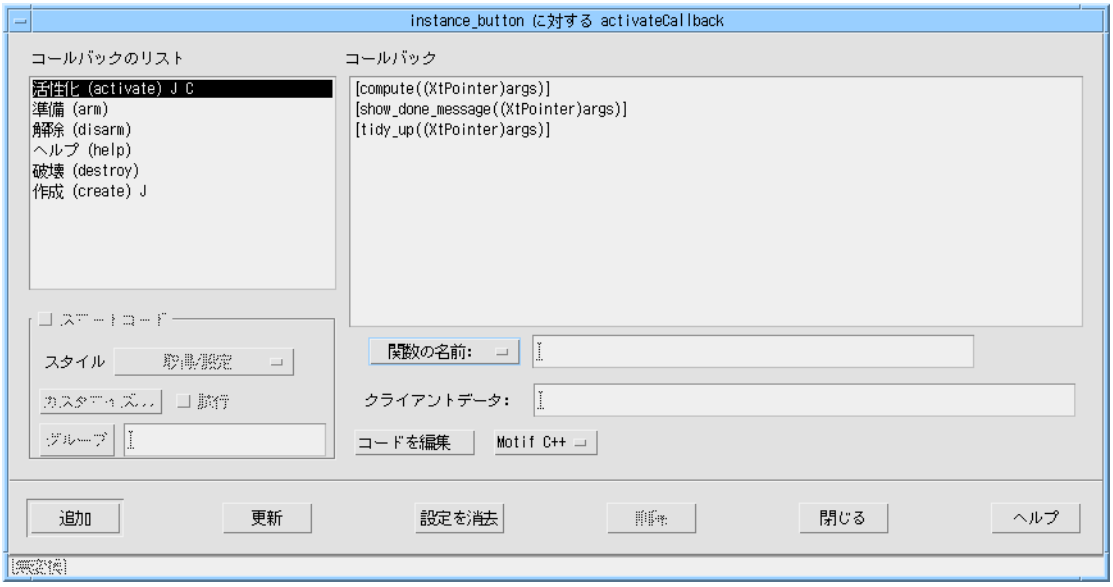


図 6-3　継承されたコールバック

コールバック構文

一般に、コールバックリスト内の各関数呼び出しの構文は C 構文と同じです。ただし、関数を示す ( ) は自動的に追加されるため、指定する必要はありません。

注 - 関数またはメソッドの名前に関数を示す括弧 ( ) や引数を加えた場合は、名前の一部として扱われます。追加した括弧や引数は、自動的には個別の構文として認識されません。

コールバック名を指定する際には、使用する言語も選択する必要があります。

1. まだ選択していない場合は、「コールバック」領域の下にあるオプションメニューで「関数の名前」を選択します。

   (学習例でも使用した) C を使用する場合は「関数の名前」を、C++ を使用する場合は「メソッドの名前」を選択してください。

2. 「関数の名前」テキストボックスをクリックします。

3. 次のように入力します。

   quit

4. 「追加」をクリックします。

   Return キーを押しても同様の結果になります。

5. 「閉じる」を押して、ダイアログを終了します。

6. デザインを保存します。

実行順序

コールバックリストは C コードのように見えますが、論理的な流れはありません。つまり、 *if...else* や *while* のような C の論理演算子は使用できず、また、コールバックが特定の順序で実行されるとは限りません。リスト内のすべての関数は、指定されたイベントが発生した場合に、それらが入力された順序とは関係なく実行されます。実行の順序が重要である場合には、必要な順序でサブルーチン呼び出しを含んでいる単一のコールバック関数を作成することができます。

クライアントデータ

「クライアントデータ」テキストボックスに指定したデータは、コールバックへ渡されます。コールバックで何らかのデータを必要とする場合は、大域変数ではなく、この機能を使用することをお勧めします。「クライアントデータ」テキストボックスに、引数として表示させたい文字列を入力してください。通常 C および C++ 構文でキャストされる型を追加することもできます。ただし、関数の引数を示す括弧 ( ) は自動的に追加されるため、入力する必要はありません。コールバック関数へ渡される引数の詳細およびその例については、206 ページの「コールバック関数の引数」 を参照してください。

注 - クライアントデータは、コールバック関数には追加できますが、コールバック・メソッドには追加できません。詳細は、300 ページの「コールバック・メソッド」 を参照してください。

メソッド

選択したウィジェットが C++ クラスに包含されていて、プルダウンメニューから「メソッドの名前」が選択されて名前テキストフィールドの左側に表示されている場合に、「メソッド」ボタンが名前テキストフィールドの右側に表示されます。「関数の名前」が選択されている場合はボタンは表示されません。「メソッド」ボタンを押すと、選択したウィジェットの包含クラスですでに定義されているコールバック・メソッドのリストが表示されます。

リストからコールバック・メソッドを選択して「了解」ボタンを押すと、選択したコールバック・メソッドが「メソッドの名前」テキストフィールドに表示されます。ウィジェットを包含する方法の詳細は、295 ページの「C++ クラス」を参照してください。

コードを編集

「コードを編集」ボタンを押すと、Sun WorkShop Visual を実行したままで、スタブファイル (生成されたファイルで、指定したコールバックを含む) を編集することができます。詳細は、264 ページの「コールバック関数の追加」を参照してください。使用するエディタの指定方法については、806 ページの「コールバックおよびプレリュード編集の設定」を参照してください。

様式オプションメニュー

「コードを編集」ボタンの横には、生成するコードの様式を選択するためのオプションメニューがあります。選択できるコード様式は以下のとおりです。

- Motif C
- Motif C++
- Motif XP
- Microsoft Windows

注 - 最後の 2 つのオプションは、Microsoft Windows モードの場合に限り表示されます。これらのオプションについては、417 ページの「様式メニュー」を参照してください。

様式オプションメニューは、「コードを編集」機能と連動しています。スタブファイルを編集する際は、使用する言語を指定する必要があります。ダイアログを呼び出すと、Sun WorkShop Visual が使用言語を認識し、対応するメニューを設定しようとします。しかし、1 つのデザインで 2 つの言語を使用して作業している場合など、不可能な場合があります。このメニューで、正しいオプションを選択しているかどうかを常に確認してください。

更新

「更新」ボタンは、現在のコールバックの設定を変更します。

スマートコードでないコールバックをスマートコードのコールバックに変えたい場合、そのコールバックが 1 箇所でしか使われていなければ、「更新」ボタンを使用する必要はありません。しかし、そのコールバックが 2 カ所以上で使われている場合は、「更新」ボタンで変更を別の場所で使用されているコールバックにも反映する必要があります。これは、1 つのコールバックの中で「スマートコードであるもの」と「スマートコードでないもの」の 2 種類が存在することはできないためです。スマートコードではないコールバックが選択された場合は、「スマートコード」トグルは使用できない状態になります。

既存のコールバックをスマートコードのコールバックに変更した場合で、すでにそのコールバックに対してスタブファイルを生成しているときは、スタブファイル中のスマートコードでないコールバックのスタブの名前を変更するか削除して、Sun WorkShop Visual が新しいスマートコードを生成できるようにしてください。

設定を消去

「設定を消去」ボタンは、スマートコードのコールバックが 1 つ以上あり、さらに新しいコールバックルーチンを追加したい場合に便利です。このボタンを押すと、すべてのコールバックやデフォルト以外の設定が選択されていない状態になります。そのため、これまで選択されていたコールバックのスマートコードの設定を誤って変更してしまうことがありません。

削除

「削除」ボタンは、現在のコールバックリスト全体を削除します。この操作を元に戻すことはできないため、「削除」ボタンは十分に注意して使用してください。

学習デザインの例については、212 ページの「リンク」以降で引き続き説明します。次の節では、コールバックおよびその使用方法についてさらに詳細に記述します。

コールバック関数

271 ページの「作成関数」では、Sun WorkShop Visual がアプリケーションのダイアログに必要なウィジェットを作成し、その初期リソース値を設定する方法について記述しています。しかし、アプリケーションを動作させるのはコールバック関数とトランスレーションです。トランスレーションの詳細は、221 ページの「トランスレーションとアクション」 を参照してください。

ほとんどのコールバック関数は、類似した構造を持っています。代表的なコールバック関数は、以下に示す動作のいくつか、またはすべてを実行します。

- テキストウィジェット内のテキストやトグルボタンの状態などの情報を、ウィジェットから取り出す
- この情報を、アプリケーション関数への呼び出しの引数として使用する
- これらの関数の結果を使用して、ウィジェットの属性を変更する
  これらの属性には、値 (テキストウィジェットの入力テキストなど) だけではなく、応答性 (ユーザー入力に対して)、可視性、さらにそのウィジェットが存在するかどうかということも含まれます。

コールバック関数の引数

コールバック関数は、次の 3 種類の引数を受け取ります。

- コールバックが呼び出したウィジェット
- 呼び出しデータ
- クライアントデータ

呼び出しデータは、ウィジェットの開発者によって定義されたデータ構造体へのポインタです。呼び出しデータ構造体については、Motif またはそのウィジェット・ツールキットの開発元の資料を参照してください。Motif の参考資料の詳細は、1012 ページの「X および Motif に関する資料」を参照してください。

クライアントデータは、任意の変数または構造のアドレスを渡すために使用できるポインタです。コールバックを登録する場合、そのコールバック関数に渡されるクライアントデータ引数に対しての値を指定することができます。

Sun WorkShop Visual では、クライアントデータの名前は、コールバック関数の単一のオプション引数としてコールバックダイアログで指定されます。指定方法については、203 ページの「クライアントデータ」を参照してください。これは、適切なプレリュードで定義し、初期化した構造体へのポインタとすることができます。たとえば、典型的なプレリュードは以下のようになります。

```
/* メインダイアログシェルに対するマネージの前プレリュード */
/* rungrep コールバックに対するクライアントデータを定義して初期化する */
static rcdata_t rcdata = {
    &hitstring,
    &errorshell,
    &errorform,
    &errortext,
    &mainshell
};
/* シェルに対するマネージの前プレリュード終わり */
```

コールバックは以下のように指定されます。

```
rungrep((XtPointer)&rcdata)
```

コールバックダイアログで上述の内容を指定するには、次のようにします。

1. 「関数の名前」テキストボックスに関数名 `rungrep` を入力します。
2. キャストを含めた引数を「クライアントデータ」テキストフィールドに次のように入力します。

   `(XtPointer)&rcdata`

   構造体 *rcdata_t* の宣言は、通常はコールバック関数モジュールおよび生成されたコードに (モジュール・プレリュードとして ***#include*** を追加することで) 含まれているヘッダーファイルにあります。コールバック関数はクライアントデータを (*rcdata_t **) にキャストした後、データをアクセスすることができます。

構造体 *rcdata* は、変数そのものの値ではなく、ウィジェット変数に対してのポインタを含むように定義されていることに注意してください。これにより、*rcdata* はウィジェットが作成される前に初期化することができます。ウィジェット変数の値がコピーされる構造体を定義することもできますが、その場合には、すべてのウィジェットが作成されるまでこの構造体は初期化されないため、注意が必要です。

C++ でのコールバック

クラスメンバー関数をコールバック関数としてウィジェットに追加することは理想的ですが、残念ながらこれは不可能です。コールバック関数は C ライブラリによって呼び出され、クラスメンバー関数が必要とする呼び出し文脈 (*this* ポインタ) を提供することができないためです。Sun WorkShop Visual は、コールバックから自動的に呼び出すための方法「コールバック・メソッド」を提供しています。コールバック・メソッドの詳細は、300 ページの「コールバック・メソッド」を参照してください。

コールバックのウィジェットへのアクセス

すべてのコールバック関数には、その関数の属するウィジェットのアドレスが引数として渡されます。この引数はウィジェット型の変数です。Sun WorkShop Visual が生成するコードでは、そのウィジェットの変数名がポインタとして使用されます。次のような方法を使用すれば、コールバック関数がその関数の属するウィジェットではなく、ユーザーが作成したアプリケーションの中のウィジェットにアクセスするように設定できます。

クライアントデータ構造

コールバック関数の属するウィジェット以外のウィジェットの名前を、クライアントデータ構造の一部として渡します。クライアントデータ構造については 206 ページの「コールバック関数の引数」を参照してください。

大域的ウィジェット変数

最も簡単な方法は、ウィジェット変数を大域的として定義することです。大域的変数として定義すると、その変数をコールバック関数モジュール内で外部宣言して、コールバック関数から呼び出すことができます。Sun WorkShop Visual が生成した「外部宣言」ファイルをスタブファイルにインクルードすると、この操作が行われます。

Sun WorkShop Visual は、デフォルトで、名前の付いたウィジェットを大域的として宣言します。この動作は、コアリソースパネルでウィジェットの記憶クラスを設定することにより、変更することができます。

大域的変数の利点は、その単純さです。しかし、多くの大域的変数を持つことはプログラムの構造にとって良いことではありません。また、大域的変数に意味のある名前を持たせて重複を避けるために、命名規則に気を配らなければなりません。

生成されたコードの組み込み

#include を使用して基本モジュールをコールバック関数のモジュールに組み込んで、大域的変数の必要性を減らすことができます。基本モジュールは、X および Motif ヘッダーファイルを組み込まずに生成することをお勧めします。

上述のようにしても、Sun WorkShop Visual は名前を付けたウィジェットを大域的として宣言します。ウィジェットの記憶クラスを静的に変更し、コールバック関数のモジュールに対して局所的にすることができます。

このテクニックは、1 つのコールバック関数が、単一のデザイン内にあるすべての、または、ほとんどのウィジェットを呼び出す必要がある場合に効果的です。さらに複雑な状況においては、アクセス用関数をコールバック関数のモジュールに追加することができます。コールバック関数はアクセス用関数を介して、他のコールバックモジュールに局所的であるウィジェットを操作することができます。

ウィジェットへは、X ツールキット簡易関数 `XtNameToWidget()` を使用してアクセスすることもできます。ウィジェット名をこの関数に渡すと、ウィジェットのポインタが返されます。詳細は X ツールキットのマニュアルを参照してください。

ウィジェットの操作

208 ページの「コールバックのウィジェットへのアクセス」に記述しているように、デザインの中のウィジェットは、さまざまな方法で操作することができます。本節では、操作方法の概要を説明します。詳細な説明は省略しますが、適切な関数とその関連資料を示します。

ツールキット簡易関数

Motif ツールキットは、一部のウィジェットの属性を獲得し、設定するための簡易関数を数多く提供しています。これらの関数はすべて、 *XmTextSetString()*、*XmTextGetString()*、*XmToggleButtonGetState()* のように、影響を与えるウィジェットクラスにちなんで名前が付けられています。これらの簡易関数は、『Motif プログラマーズ・リファレンス』で説明されています。

まず、簡易関数に着目してください。これらは最も簡単に使用することができ、しかも効率的です。

簡易関数はウィジェットの引数をとります。これは適切なクラスのウィジェットへのポインタです。ウィジェットのクラスが不適切な場合は、一般に簡易関数はコアダンプします。また、簡易関数にはウィジェットおよびガジェットの両バージョンがあるため、不適切なバージョンが使用された場合にはコアダンプが起こります。

リソースの設定と獲得

簡易関数が存在しない場合は、*XtGetValues()* または *XtSetValues()* を使用して直接ウィジェットのリソースを獲得または設定しなければなりません。これはウィジェットプログラミングの基礎であるため、X または Motif に関する書籍で詳しく解説されています。

すべてのリソースがウィジェット作成後に設定できるとは限りません。各ウィジェットクラスのアクセス制御については『*Motif* プログラマーズ・ガイド』を参照してください。

ウィジェットの有効化と 無効化

ウィジェットを使用できなくする (つまり、ユーザー入力に対して応答不可能にする)、あるいは再度使用できるようにするためには、*XtSetSensitive()* を使用します。リソース *XmNsensitive* は直接設定しないでください。

ウィジェットが応答不可能になると、その子孫もすべて応答不可能になります。応答不可能なウィジェットは通常、グレー表示されます。

テキストまたはテキストフィールドウィジェットが応答不可能になると、ユーザーはキー入力でテキストをパンおよびスクロールすることができなくなるため、表示範囲が限られます。リソース *XmNeditable* は *False* に設定することをお勧めします。

ウィジェットの表示と非表示

ウィジェットを表示または非表示にする方法には、マネージ (管理) とマップの 2 通りがあります。

ウィジェットがマネージされていない場合は、ウィジェットの親はウィジェットのためのスペースを確保しないため、そのウィジェットは画面上に表示されません。ウィジェットをマネージから外すには *XtUnmanageChild()* または *XtUnmanageChildren()* を使用し、また、マネージを行うには *XtManageChild()* または *XtManageChildren()* を使用します。Sun WorkShop Visual は、ウィジェットが作成された後にそれらをマネージするためのコードを生成しますが、コアリソースパネルの「マネージ」トグルによってマネージするかどうかを変更することができます。

ウィジェットがマネージされていてもマップされていない場合は、親はウィジェットのためのスペースを確保していますが、画面には表示されず、空のスペースがあるだけです。ウィジェットは通常、マネージされると自動的にマップされます。これは、「コアリソース」パネルの「設定」ページにあるリソース `XmNmappedWhenManaged` で制御することができます。

マップおよびアンマップ (マップ取り消し) は、配置を変更せずにダイアログ内のウィジェットの表示を変更するためによく使用されます。ウィジェットのマネージおよびアンマネージ (マネージ取り消し) を行うと、配置が変更されます。

ダイアログシェルの子をマネージまたはアンマネージすることにより、ダイアログ全体を表示したり非表示にしたりすることができます。ダイアログが最上位シェルを使用している場合は、シェルに *XtPopup()* および *XtPopdown()* を使用します。

注 - Sun WorkShop Visual の組み込みリンク機能を使用してウィジェットを自動的に、かつ動的に表示/非表示にする方法については、212 ページの「リンク」を参照してください。

ウィジェットの作成と破壊

Sun WorkShop Visual は、ダイアログに対してウィジェットを作成するためのコードを生成します。デフォルトの *main()* プログラムは、起動時にすべての作成関数を呼び出します。ウィジェット作成は比較的時間がかかるため、作業全体に支障をきたす場合があります。そこで多くの場合、ダイアログが最初にポップアップされるまでは、その作成を延期する慣例となっています。ダイアログがすでに作成されたかどうかを判断するためには、ポップアップを実行するコールバック関数内で静的なブール型のフラグを使用することができます。

完全なダイアログを作成するためのコードの作成に加え、ダイアログ部分に対しての作成関数を生成することもできます。これについては、307 ページの「子のみを生成するウィジェット」で説明します。たとえば、これらの関数をコールバック関数から呼び出して、再使用可能な構成要素のインスタンスをもう 1 つ作成することもできます。

ウィジェット (およびそのすべての子) を破壊するためには、*XtDestroyWidget()* を使用します。しかしウィジェットを破壊して再度作成すると効率が悪いため、ウィジェットをアンマネージし、必要であれば再度マネージすることをお勧めします。

リンク

Sun WorkShop Visual には、リンクと呼ばれる事前定義されているコールバック関数があります。以下の 6 種類のリンクが使用できます。

- 表示
  ウィジェットおよびその子を画面上に表示します。
- 非表示
  ウィジェットを表示されないようにします。ウィジェットは破壊されるのではなく、単に隠されるだけです。

- マネージ
  既存のウィジェットをマネージします。
- アンマネージ
  ウィジェットをアンマネージします。
- 有効化
  ボタンまたはコマンドを使用可能にします。
- 無効化
  ボタンまたはコマンドを使用不可能 (グレー表示) にします。

リンクとコールバックの違い

プッシュボタン、矢印ボタンおよびカスケードボタンだけがリンク元になることができます。すべてのリンクは「活性化」イベントにより有効になります。リンクはデザイン内のどのようなウィジェットでも表示、非表示、マネージ、アンマネージ、有効化、無効化することができます。1 個のボタンに複数のリンクを持たせることができます。

リンクとは、Sun WorkShop Visual が設定するコールバックです。しかし、コールバックと異なり、ダイナミックディスプレイで動作するため、ウィンドウ動作のプロトタイプの作成に使用することができます。コードを生成する場合には、ダイナミックディスプレイでの動作とまったく同様に動作するリンクを取り込む、あるいは単純なリンクの代わりに複雑なコールバックを代用することができます。

リンク追加の制限事項

リンクを追加するには、以下のいずれかの条件を満たしている必要があります。いずれの条件にも合致しない場合は、「追加」ボタンは無効になり、リンクの追加を行うことができません。

- リンクの宛先となるウィジェット、つまり表示、非表示、マネージ、アンマネージ、有効化、無効化されるウィジェットが、明示的な変数名を持っていること。リンク元となるウィジェットには明示的な変数名は必要ありません。
- リンクの宛先となるウィジェットがシェルの場合は、その直接の子もまた明示的な変数名を持っていること。

- リンクの宛先および元となるウィジェットは、静的または局所的な変数として指定されていないこと。静的および局所的なウィジェット変数の詳細は、308 ページの「宣言範囲の変更」を参照してください、
- リンクの宛先となるウィジェットは「子のみ生成」として宣言されていないこと。この条件の意味がわからない場合は307 ページの「子のみを生成するウィジェット」 を参照してください。
- リンクの元となるウィジェットが定義のインスタンスの一部である場合は、インスタンスのルートウィジェットも明示的な名前を持っていること。インスタンスの詳細は、316 ページの「インスタンス」を参照してください。
- リンクの元となるウィジェットが C++ クラスの定義のインスタンスの一部である場合は、ウィジェットが「公開」として宣言されていること。C++ クラスおよびメンバーアクセスの詳細は、295 ページの「C++ クラス」を参照してください。
- Java コードを生成する場合は、ソースウィジェットは抽象的であってはならず、クラス内に含まれていなければなりません。「抽象的」という言葉は、ファイル選択ボックスのスクロールテキストなどの、複合ウィジェットを選択した場合に階層に現れる付随ウィジェットを表します。Sun WorkShop Visual を使用した Java コードの生成の詳細については、第 10 章「Java 用のデザイン」を参照してください。

注 - リンクの宛先ウィジェットの変数名が変更された場合は、そのウィジェットに設定されたリンクはすべて無効になります。

学習の例でのリンク設定

これから、作成したヘルプ画面を表示して、適切なタイミングで非表示にするという、一般的なリンクの構成を設定していきます。このような設定を行うには、以下の操作を行います。

- ヘルプメニューの「About This Layout」ボタンに「表示」リンクを設定する
- ヘルプ画面の「OK」プッシュボタンに「非表示」リンクを設定する

構成領域では現在、「OK」プッシュボタンが表示されています。次に、このプッシュボタンに「非表示」リンクを設定します。

1. プッシュボタンを選択します。

2. 「変数名」フィールドをダブルクリックします。

3. **ok_button** と入力し、<Return> を押して新しい名前を登録します。

4. ウィジェットメニューをプルダウンして、「リンク編集」を選択します。

   図 6-4 に示されるパネルが表示されます。

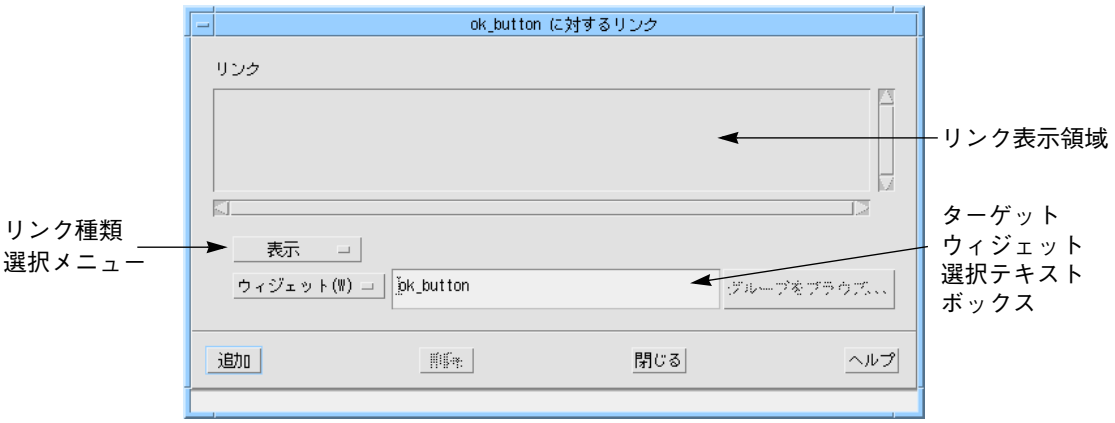


図 6-4　デフォルトのリンクウィンドウ

「OK」ボタンが選択された場合にヘルプ画面全体が消去されるように、「非表示」リンクのターゲットを *help_window* ウィジェットに設定します。

ターゲットウィジェットを選択するには、以下の操作を行います。

5. デザイン階層でシェルを選択します。

   シェルの名前である *help_window* がリンクパネルの「ウィジェット」フィールドに表示されます。しかし、「追加」コマンドはまだ使用できません。これは、シェルの直接の子であるダイアログテンプレートに名前が付いていないためです。すでに説明したように、シェルへのリンクを設定する前に、シェルの子に名前を付ける必要があります。

   ダイアログテンプレートに名前を付けている間、リンクパネルは開いたままにできます。

6. ダイアログテンプレートを選択します。

7. 「変数名」フィールドをダブルクリックし、次のように入力します。

   dialog_2

8. シェルを選択します。

   これでシェルは有効なターゲットウィジェットとなり、「追加」が使用できるようになります。

   次に、リンクの種類を選択します。

9. リンク種類選択メニューから「非表示」を選択します。

10. 「追加」をクリックします。

    図 6-5 に示されるように、リンク表示領域には新しいリンクが表示されます。

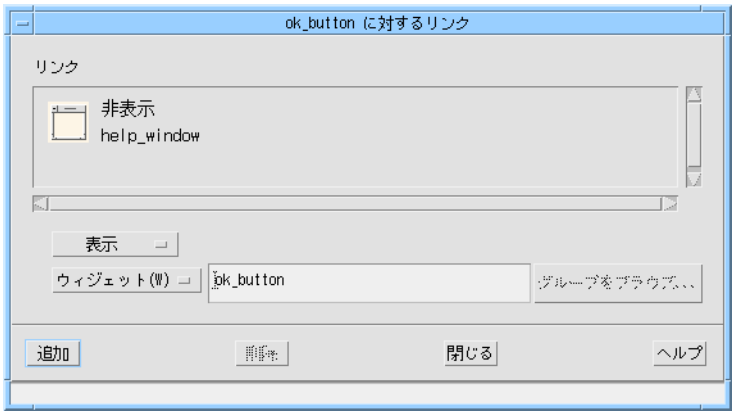


図 6-5　新しい「非表示」リンクがあるリンクウィンドウ

11. 「閉じる」をクリックします。

12. 新しいリンクを実行するには、ダイナミックディスプレイで「OK」ボタンをクリックします。

    ヘルプ画面が消去されます。構成領域でシェルを選択しなおすと、ヘルプ画面を復元することができます。

    メインウィンドウにおいてボタンが押された場合にヘルプ画面を表示する「表示」リンクも設定することができます。この設定を行うには、以下の操作を行います。

13. ウィンドウ保持領域の *myFirstShell* アイコンをクリックします。

    構成領域にメインウィンドウの階層が表示されます。

    次に、ヘルプメニューにあるプッシュボタンに新しいリンクを設定します。

14. 2 番目のメニューの子であるプッシュボタン、*help_button* を選択します。

    リンクウィンドウは、リソースパネルや配置エディタとは異なり、現在選択されているプッシュボタンに対して自動的にリンクの追加を開始しません。現在選択されているボタンに対してリンク編集するためには、以下の作業を行う必要があります。

15. ウィジェットメニューをプルダウンし、「リンク編集」を選択します。

    リンクウィンドウにはプッシュボタンの名前と、現在のリンク (現時点では存在しません) が表示されます。ヘルプ画面のシェルであるターゲットウィジェットを選択します。

16. ウィンドウ保持領域で *help_window* アイコンをクリックします。

17. リンク種類選択メニューで「表示」を選択します。

18. 「追加」をクリックします。

    リンク表示領域に新しいリンクが表示されます。

    以下の操作を行なって、これらの 2 個のリンクを実行します。

19. ウィンドウ保持領域で *myFirstShell* アイコンをクリックします。

20. ダイナミックディスプレイの「Help」メニューをプルダウンし、「About This Layout」を選択します。

    このプッシュボタンにある「表示」リンクにより、ヘルプ画面が表示されます。

21. ダイナミックディスプレイにあるヘルプ画面の「OK」ボタンをクリックします。

    プッシュボタンの「非表示」リンクにより、ヘルプ画面が消去されます。手順 20 と手順 21 は、何度でも繰り返すことができます。

22. デザインを保存します。

リンクの削除

リンクを削除するためには、次のようにします。

- リンク表示領域で、そのリンクのアイコンを選択し、「削除」をクリックします。

学習デザインの例については、第 7 章「コードの生成」以降で引き続き説明します。ここでは、Sun WorkShop Visual を使用してユーザーが作成したアプリケーションに機能を追加する方法について記述します。

ドラッグ & ドロップ

Motif 1.2 では、X 選択メカニズムを使用してアプリケーションと通信することができる、高度なドラッグ & ドロップ機能を提供しています。Sun WorkShop Visual のサポートは、ユーザーにアプリケーションでドロップサイトを指定させる単純なものです。ドラッグの開始はコールバックまたはアクション関数内から行われる動的関数であるため、Sun WorkShop Visual は明示的なサポートは行いません。

ドロップサイトは特定の型のデータを転送メカニズムから受け取ることができるウィジェットです。 Sun WorkShop Visual は、コアリソースパネルのドロップサイトページを使用してドラッグ & ドロップのサポートを行います。

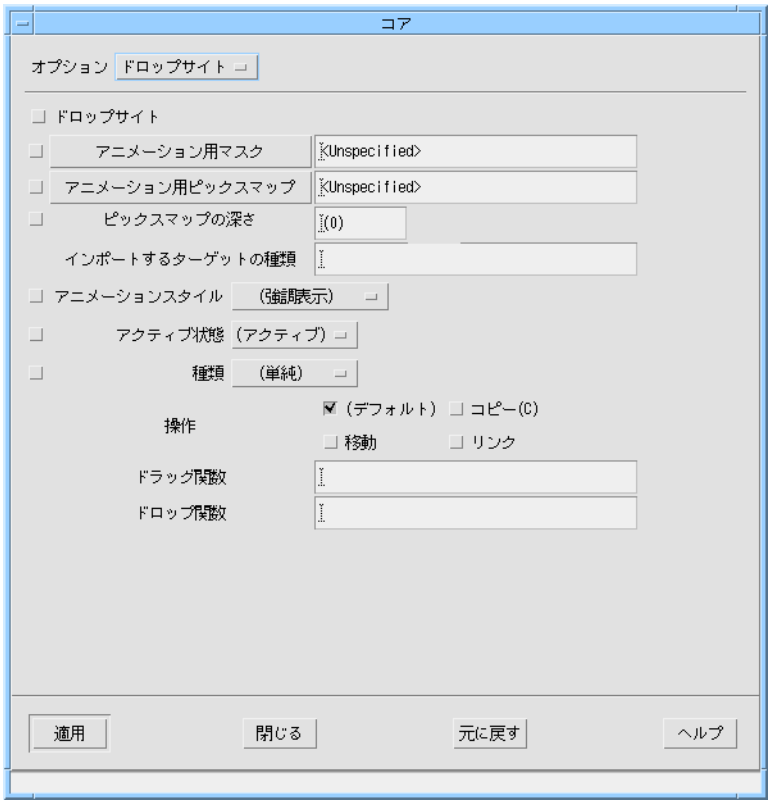


図 6-6　ドロップサイトのページ

ウィジェットをドロップサイトとして指定するためには、「ドロップサイト」トグルをオンにし、インポートするターゲットの種類およびドロップ関数を指定します。「インポートするターゲットの種類」フィールドは、ドロップ関数によって処理することができる型を指定するために、アトムに変換される文字列のリストです。リストは、コンマまたはスペースで区切られた文字列として指定されます。

デフォルトでは、Motif はラベル (およびラベルの派生) ウィジェット上でマウスボタン 2 が押された場合に、ドラッグ操作を開始して、*labelString* または *labelPixmap* を転送します。Sun WorkShop Visual は、Motif のデフォルト動作を利用するために、「インポートするターゲットの種類」に指定されている型をインポートするドロップサイトのウィジェットに、ドロップ関数を追加します。以下の学習例を使用して、ドロップサイトの動作を確認してください。

1. アプリケーションシェルに 2 個のプッシュボタンを含むローカラムを追加して、ダイアログを作成します。
2. それぞれのウィジェットに次のように名前をつけます。
   shell、*rowcolumn*、*MyButton1*、*MyButton2*
3. *MyButton1* の「ドロップサイト」ページをポップアップします。
4. 「ドロップサイト」トグルをオンに設定し、アニメーションスタイルを「シャドウ・イン」に設定します。
5. 「インポートするターゲットの種類」フィールドに、次のように入力します。
   `COMPOUND_TEXT`
6. 「ドロップ関数」フィールドに次のように入力します。
   `drop_button1`

   「ドロップ関数」フィールドおよび「ドラッグ関数」フィールドには、ドロップおよび動的ドラッグ動作を処理するための関数名を指定します。
7. 変更を適用して、ダイアログを閉じます。
8. メインウィンドウ上のいずれかのテキスト (たとえば変数名フィールドのテキスト *MyButton2*) を、ダイナミックディスプレイウィンドウの *MyButton1* の上までドラッグします。

   「MyButton1 」ボタンがシャドウイン表示され (ボタン枠の内側に影が表示され)、ドラッグターゲットとして有効なドロップサイトであることを示します。
9. マウスボタンを離して、ウィジェットにテキストをドロップします。

   Sun WorkShop Visual の提供するドロップ関数によって、ドロップされたテキストが単純にコピーされます。
10. *MyButton2* を選択して、手順 4 を繰り返します。
11. 「インポートするターゲットの種類」フィールドに次のように入力します。
    `PIXMAP`
12. 「ドロップ関数」フィールドに次のように入力します。
    `drop_button2`
13. ツールバー上のピックスマップをダイナミックディスプレイウィンドウの MyButton2 の上までドラッグします。

ドロップサイトの使用例、およびドラッグの開始については、Motif のマニュアルを参照してください。

通常はドロップサイトではないウィジェットに対しては、*XmDropSiteRegister()* への呼び出しを持つ、C および C++ に対してのコードが生成されます。テキストウィジェットはデフォルトで *COMPOUND_TEXT* をインポートすることができるドロップサイトです。「ドロップサイト」トグルをオフに設定すると、このウィジェットを使用不可能にすることができ、また、適切なリソースを変更するだけで修正することができます。

転送を処理するためには、ドロップ関数を指定する必要があります。ドロップ関数は、生成されたコード中では外部関数として宣言されているだけです。

トランスレーションとアクション

ウィジェットには動作があります。たとえば、ユーザーがプッシュボタン上でマウスボタン 1 を押すと、そのプッシュボタンは強調表示されます。ユーザーがマウスボタンを離すと、プッシュボタンの外観は通常に戻り、活性化コールバックリストの関数が呼び出されます。

この動作は、プッシュボタンウィジェットに固定されているのではありません。イベントへの応答として行われるアクションにイベントを割り当てる、ウィジェットのトランスレーションテーブルによって決定されます。ウィジェットが作成されると、そのトランスレーションテーブルは、一連のエントリがデフォルトになるように初期化されます。たとえば、プッシュボタンウィジェットのデフォルト・トランスレーションテーブルには、次の内容が含まれます。

```
<Btn1Down>:Arm()

<Btn1Up>:Activate() Disarm()
```

コロンの前に存在するのはイベント指定であり、右側はウィジェットが応答として実行するアクションの名前です。第 2 のテーブルであるアクションテーブルは、アクションを実行する関数のアドレスにそのアクション名を割り当てるために使用されます。たとえば、プッシュボタンのデフォルト・アクションテーブルには以下の内容が含まれます。

```
"Arm",Arm

"Activate",Activate
```

```
"Disarm",Disarm
```

各行の最初の項目はアクションの名前で、次の項目は関数の名前です。規約および常識の観点から、アクションと関数の名前は同じであるか、あるいは少なくとも明確に関連している必要があります。

Sun WorkShop Visual 内のウィジェットのトランスレーションテーブルは変更することができますが、ウィジェットのアクションテーブルは変更できません。ただし、ウィジェットごとのアクションテーブルの後に検索される、アプリケーションに対して大域的なアクションテーブルに、新しいアクションを定義することは可能です。これには、「トランスレーションテーブルの修正」に示すようなコードが必要です。

注 - Microsoft Windows 上ではトランスレーション機能はサポートされていません。したがって、Microsoft Windows モードでは「トランスレーション」ダイアログの「適用」ボタンはピンク色で表示されます。

トランスレーションテーブルの修正

Sun WorkShop Visual 内のウィジェットのトランスレーションテーブルは、簡単に修正することができます。修正の手順を理解するには、次に示す簡単な演習を行うことをお勧めします。

1. プッシュボタンを含む単純なウィジェット階層を作成します。
2. ウィジェット階層でプッシュボタンアイコンを選択します。
3. 「ウィジェット」メニューから、「トランスレーション」を選択します。

   図 6-7 に示すトランスレーションダイアログが表示されます。

図 6-7　トランスレーションダイアログ

4. 「追加」の下のフィールドをクリックし、次のように入力します。

```
Ctrl<Key>q: ArmAndActivate()
```

5. 「適用」をクリックします。

これにより、プッシュボタンウィジェットの新しいトランスレーションが追加されました。この時点でその効果を試すことができます。

注 - アクションが見つからないことを示すエラーダイアログが表示されますが、これは参考用で、実際には変更は行われています。Sun WorkShop Visual でアクションテーブルを作成する方法についての詳細は、229 ページの「追加のアクション」 を参照してください。

---

6. ダイナミックディスプレイ・ウィンドウのプッシュボタン上にマウスポインタを置きます。

7. **<Ctrl-Q>** を押します。

新しいトランスレーションを実行すると、マウスボタン 1 をクリックすることと同じ動作が行われます。ウィンドウが入力フォーカスを持っていない場合はトランスレーションは動作しません。また、入力フォーカス動作は、ウィンドウマネージャの構成によって異なります。

このボタンが他のイベントのトリガーとなるように、指定したトランスレーションを変更することもできます。ウィジェットのリセットが行われるまでは、新しいトランスレーションだけでなく、以前に指定したトランスレーションも有効であることに注意してください。

追加、上書き、置換

トランスレーションダイアログには、「上書き」と「追加」のラベルが付いた領域があります。新しいトランスレーションは、どちらかのまたは両方の領域に入力することができます。これらの領域は、既存のトランスレーションと同じイベントを使用してトランスレーションを指定する場合に限り、使い分ける必要があります。「上書き」領域に新しいトランスレーションを入力すると、新しいトランスレーションは既存のものと置き換えられます。「追加」領域を使用すると、既存のトランスレーションが優先されます。

トランスレーションが上書きされない限り、トランスレーションの追加を行なっても、ウィジェットに対する既存のデフォルト・トランスレーションに影響はありません。Motif ウィジェットは、それらが意図されている動作を生成するデフォルト・トランスレーションを多数持っているため、特に注意してください。

「置換」トグルを設定すると、既存のトランスレーションはすべて取り除かれ、新しく入力されるトランスレーションに置き換えられます。このため、「置換」を使用する場合には十分な注意が必要です。すべてのデフォルト・トランスレーションを取り除く「置換」と、1 つずつ置き換えを行う「上書き」を混同しないように注意してください。

トランスレーションテーブルの構文

トランスレーションテーブルの構文は複雑です。次の節以降で、Sun WorkShop Visual で使用する構文について記述します。詳細は、X ツールキットのマニュアルを参照してください。

トランスレーションテーブルの各エントリは、以下の形式になっています。

[修飾キーリスト]<イベント>[,<イベント>...][(回数)][詳細]:

```
[アクション([引数群])...]
```

角括弧 [ ] は、項目が任意であることを示しています。また、省略符号 (...) は、項目が繰り返される可能性があることを示しています。

修飾キーリスト

修飾キーリストは、修飾キー (Control、Shift など) およびマウスボタン (X ではマウスには 5 個のボタンが使用できることになっています) が押されているか、などの状態を表わします。最も頻繁に使用される修飾キーは Ctrl、Shift、Alt です。これらは c、s、a と省略形にすることができます。

修飾キーリストが省略されている場合、その修飾キーの状態は重要ではありません。

`<Key>Q` は *<Q>*、*<Ctrl-Q>*、*<Alt-Meta-Q>* などに対応します。

特定の修飾キーがリストに示されていない場合、その状態は重要ではありません。

`Ctrl<Key>Q` は、*<Ctrl-Q>*、*<Ctrl-Meta-Q>*、*<Ctrl-Alt-Meta-Q>* などに対応します。

修飾キーリストには、複数の修飾キーを指定することができます。

`Ctrl Meta <Key>Q` は、*<Ctrl-Meta-Q>* には対応しますが、*<Ctrl-Q>* または *<Meta-Q>* には対応しません。

修飾キーが押されてはいけないことを指定する場合は、修飾キーの前にチルド (~) を付けます。

`Ctrl ~Meta<Key>Q` は *<Ctrl-Q>* に対応しますが、*<Ctrl-Meta-Q>* には対応しません。

押される修飾キーが指定と正確に一致している必要があることを指定する場合は、感嘆符 (!) で修飾キーリストを開始します。

`!Ctrl<Key>Q` は、*<Ctrl-Q>* に対応しますが、*<Ctrl-Meta-Q>* またはマウスボタンと同時に押す <Ctrl-Q> には対応しません。

修飾キー「None」は、いかなる修飾キーも押してはいけないことを意味します。

`None<Key>Q` は、<Q> に対応しますが、*<Ctrl-Q>* や *<Alt-Meta-Q>* などには対応しません。

通常、トランスレーションでは大文字と小文字を区別しません。`<Key>Q` は、*<Q>* と *<q>* の両方に対応します。修飾キーリストの前にコロン (:) を置くことにより、トランスレーションに大文字と小文字を区別するように指定することができます。

`:<Key>Q` は、*<Q>* には対応しますが、*<q>* には対応しません。

イベントと回数

イベント には、X イベントの名前、あるいは多数の別名の 1 つを指定します。最もよく使用されるイベントには、*Key* (キーを押すこと)、*BtnDown* と *BtnUp* (マウスボタンの場合)、そしてボタン N*Down*とボタン N*Up* (N は 1 から 5 までの数) などがあります。すべてのイベントと別名のリストについては、Xt の資料を参照してください。

`<Key>a` は *<a>* に対応します。

`<Btn1Up>` は、マウスボタン 1 を離すことに対応します。

トランスレーションには、連続する一連のイベントを指定することができます。この場合、コンマで区切ります。

`<Key>Q,<Key>A` は、*<Q>* の次に間をおかず *<A>* が続くことに対応します。

`<Btn1Down>,<Btn1Up>` は、マウスボタン 1 のクリックに対応します。

ボタンの押下および解除イベントと一緒に回数を指定して、複数のクリックを検出することができます。回数は、1 から 9 までの数字で、プラス記号 (+) がその後に続く場合があります。

`<Btn1Down>(2)` は、マウスボタン 1 を 2 回押すことに対応します。

`<Btn1Up>(3+)` は、マウスボタン 1 を 3 回以上離すことに対応します。

回数指定をする場合は、ボタンイベントはすべて短い期間 (通常は 200 ミリ秒以内) に発生しなければなりません。短い期間内に発生しないイベントは、対象外となります。

詳細

イベント指定の最後のフィールドは、「詳細」と呼ばれます。「詳細」は通常、キーイベントと一緒にのみ使用されます。この場合、押されるキーの指定は「詳細」が行います。

詳細フィールドで指定される値は、ヘッダー *<X11/keysymdef.h>* にあるように、接頭辞 *XK_* が取り除かれているキーシム (*keysym*) です。ほとんどのキーの場合、これはキー上の文字と同じです。

<Key>a は、*<a>* に対応します。

アルファベット以外のキーの場合は、「キーシム」の名前を確認してください。「+」に対しての「キーシム」は、*XK_plus* です。したがって、以下のようになります。

<Key>plus は、*<+>* に対応します。

ここで行われる照会では、大文字と小文字が区別されないため、プラスキー上にある他の記号 (多くのキーボードの場合は <、=、>) にも対応します。

Motif では、入力される特定のキーイベントを Motif 仮想キーシムに変換するため、さらに複雑になります。トランスレーションテーブルには、X のキーシムの代わりに、これらの仮想キーシムを使用することをお勧めします。

<Key>Delete ではなく、<Key>osfDelete を使用します。

仮想キーシムのリストを以下に示します。これらの解釈については、『*Motif* プログラマーズ・リファレンス』の *VirtualBindings(3X)* を参照してください。

表 6-1　OSF 仮想キーシム

osfActivate	*osfAddMode*	*osfBackSpace*	*osfBeginLine*
osfCancel	*osfClear*	*osfCopy*	*osfCut*
osfDelete	*osfDown*	*osfEndLine*	*osfHelp*
osfInsert	*osfLeft*	*osfMenu*	*osfMenuBar*
osfPageDown	*osfPageLeft*	*osfPageRight*	*osfPageUp*
osfPaste	*osfPrimaryPaste*	*osfQuickPaste*	*osfRight*
osfSelect	*osfUndo*	*osfUp*	

「詳細」フィールドをマウスボタン・イベントに使用して、特定のマウスボタンを指定することもできます。しかし、イベントフィールドでのマウスボタンの指定の方が簡単であるため、この方法はあまり一般的ではありません。

<BtnDown>Button1 は、<Btn1Down> と同じです。

アクション

トランスレーションテーブルの右側にあるアクションは、複雑なものではありません。通常、各アクションは後に括弧が続いている名前にすぎません。括弧内には任意の数の文字列引数を指定することができますが、ほとんどのアクション関数は、引数の指定を予定していません。引数は、引用符で囲んではなりません。スクロールバーウィジェットに対する代表的な追加トランスレーションを次に示します。

```
<Key>d:IncrementDownOrRight(0)

<Key>u:IncrementUpOrLeft(0)
```

複数のアクションを指定することもできますが、まったく指定しなくてもかまいません。既存トランスレーションを、イベント指定が同じでアクションの異なるトランスレーションを使用して上書きすることにより、ウィジェットのデフォルト動作の一部を使用禁止にすることができます。

多くの場合、使用されるアクションは、ツールキットによって事前定義されています。229 ページの「追加のアクション」では、独自のアクションの追加方法を説明します。

トランスレーションテーブルの順序

イベントが受け取られると、トランスレーションテーブルは上から下に検索が行われます。検索は、イベントに一致するイベント指定を持った最初のエントリで終了します。つまり、最も詳細なイベント指定を持つエントリが最初にくるように、トランスレーションテーブルを編成するべきです。たとえば、トランスレーションテーブルに次のようなエントリがあるとします。

```
<Key>q: action1()

Ctrl<Key>q: action2()
```

ユーザーが *<Q>* または *<Ctrl-Q>* を入力した場合、検索は最初のエントリで終了し、どちらの場合においても *action1()* が呼び出されます。*<Ctrl-Q>* で action2 を呼び出すようにするには、エントリの順番を逆にする必要があります。

トランスレーションテーブルの順序の詳細については、X ツールキットの資料を参照してください。

使用可能なアクション

トランスレーションテーブルを変更することにより、通常はアクションを引き起こさないイベント列に対しても、ウィジェットにアクションを実行させることができます。ユーザーは、独自のアクション関数を書くことができますが、トランスレーションの最大の利点は、ユーザーがウィジェットの組み込みアクションの 1 つを使用できるということです。

Motif ツールキットの組み込みアクションは、『*Motif* プログラマーズ・リファレンス』に記述されています。各ウィジェットの記述には、デフォルトのトランスレーションと呼び出すアクションの両方が含まれています。プリミティブ・ウィジェットの中には、多くのアクションを提供するものもあります。

これらのアクションの 1 つを使ったトランスレーションを使用する場合は、

Sun WorkShop Visual で直接そのテストを行うことができます。逆に、プッシュボタンの *Activate()* アクションのように、ウィジェットのコールバックリスト中の関数を呼び出す組み込みアクションも存在します。この場合、適切なイベント列においてそのアクションを呼び出すようにトランスレーションの指定を行い、通常のコールバック関数中にコードを置いた方が簡単です。

追加のアクション

要求を満たすような組み込みアクションが見つからない場合は、アクションを実行する独自のアクション関数を書くことができます。図 6-8 にトランスレーションの例を示します。この例では、*<Ctrl-e>* が押されており、アクション「doActionE」が起動されています。

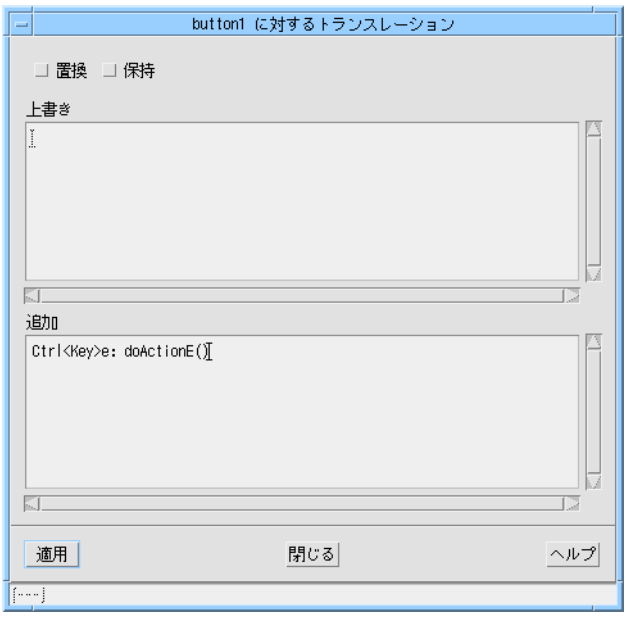


図 6-8　トランスレーションの例

独自のアクション関数を定義する場合、定義したアクション関数をアプリケーションの「アクションテーブル」に追加する必要があります。図 6-9 に示すように「アクションテーブル」ダイアログでアクション関数を定義すると、Sun WorkShop Visual が自動的にこれを行います。「アクションテーブル」ダイアログは、「モジュール」メニューから「アクション手続き」を選択すると表示されます。

図 6-9　「アクションテーブル」ダイアログ

図 6-8 の例では、「doActionE」がアプリケーションで定義が必要なアクションです。図 6-9 はこのアクションの定義を示しています。「アクション名」にはトランスレーションダイアログで使用した名前を指定します。「手続き名」には、呼び出される関数の名前を指定します。わかりやすさのために、通常これらの名前には同じ名前を使用します。

アクション関数にはクライアントデータは関連付けられていませんが、トランスレーションダイアログのアクションで定義した引数が渡されます。トランスレーションダイアログでは、任意の数の文字列引数を括弧で囲んで指定できます。

アクション関数は、デザイン内にいくつでも定義できます。Sun WorkShop Visual は、定義したアクション関数のスタブ、および関連付けられたアクションテーブルをメインコードファイルに生成します。

Xt 手続き

Sun WorkShop Visual では、以下の Xt 手続きを追加できます。

1. 追加のイベント手続き
   この手続きには、入力ソースとタイムアウトがあり、イベントのソースとして処理されます。同様に、Xt 作業手続きも追加のイベント手続きに含まれており、処理するイベントがない場合に呼び出されます。

2. 言語手続き
   アプリケーションの言語対応をカスタマイズする方法として、X アプリケーションの開始時に、これらの関数の 1 つが呼び出されます。言語手続きはいくつでも指定できます。

3. イベントハンドラ
   これらの関数は、事前定義したアクションが 1 つでも起動されると呼び出されます (例 : マウスボタン 1 が押された場合)。これらの手続きはトランスレーションテーブルを介しません。

上記の最初の 2 種類の Xt 手続きはアプリケーション全体に対して指定します。3 番目の手続きはウィジェットごとに指定します。各手続きについての詳細は、以下で説明します。

X の代替イベントソース

Xt (X ツールキット) アプリケーションは、通常、X サーバーからのイベントを待ちます。キーボードの押下やマウスクリックなどのユーザーアクションは、X サーバーを介して Xt アプリケーションに送られます。ユーザーアクションが発生しなければ、アプリケーションはアクションが発生するのを待つだけです。つまり、Xt アプリケーションは膨大な時間をアクション待ちに費やす可能性があるということです。そのため、Xt では、他に処理するイベントがない場合に、呼び出す手続きを登録できます。さらに、X サーバーに起因していない特定のイベントに応答する手続きを登録することもできます。追加することのできる「追加の」手続きは次のとおりです。

1. Xt 作業手続き

2. 入力手続き

3. タイムアウト手続き

Sun WorkShop Visual 内に追加したウィジェットコールバック関数と同様に、追加した Xt 手続きがスタブファイルへのスタブとして生成されます。ウィジェットコールバックの編集と同じ方法で、Sun WorkShop Visual 内からスタブを編集できます。手続きを追加するためのコードは、Sun WorkShop Visual によってメインモジュールに生成されます。

注 - メインモジュールを生成しなければ、Xt 関数はアプリケーションに追加されません。

これらの手続きを追加するには、「モジュール」メニューから対応する項目を選択してください。各関数について以下で詳細に説明します。

Xt 作業手続き

作業手続きは、Xt が他に処理するイベントがない場合に呼び出される関数です。そのため、作業手続きは、Xサーバーからのイベントに干渉せずにバックグラウンドでバッチ処理を行う場合に使用できます。作業手続きの一般的な使い方の 1 つにプログラムの初期設定があります。アプリケーションの起動時には、多くのウィジェットを作成しなければならない場合が頻繁にあります。ウィジェットの作成には膨大な時間がかかるため、作業手続きを使用することで、アプリケーションはユーザーアクションに対して即座に応答することができます。

作業手続きは True または False の値を返します。True は、関数が実行後に作業待ち行列から削除されることを示します。False は、待ち行列にユーザーイベントがなければ、次回に作業手続きが再度呼び出されることを示します。作業手続きは、いつでも必要な数だけ登録することができます。

作業手続きを追加するには、「モジュール」メニューから「作業手続き」を選択してください。これにより、図 6-10 に示すダイアログが表示されます。作業手続きはいくつでも追加できますが、作業手続きの実行中は他のイベントをアプリケーションが処理できないことを留意しておいてください。そのため、作業手続きはすぐに返される必要があります。作業手続きの優先順位は、一般的に、登録された順になります。

図 6-10 「作業手続き」ダイアログ

入力手続き

ファイルやパイプをイベントのソースとして設定する場合に入力手続きを使用します。入力手続きは、ファイルが読み取り (または書き込み) 準備できたときに呼び出されます。入力手続きを追加するには、「モジュール」メニューから「入力手続き」を選択してください。図 6-11 に示すダイアログが表示されます。

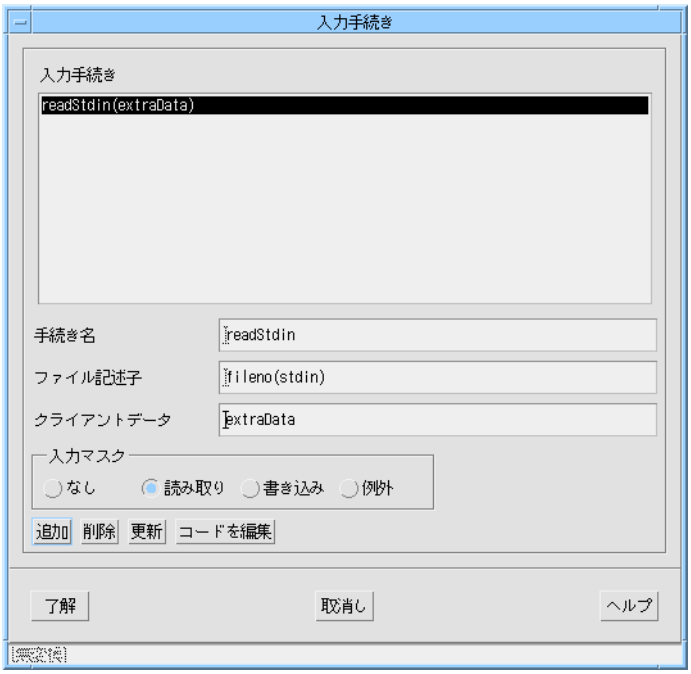


図 6-11 「入力手続き」ダイアログ

入力手続きを定義するには、ファイルやパイプのファイル記述子を指定する必要があります。「入力マスク」には、ファイルに必要なアクセスの種類を指定してください。ファイルに指定したアクセスが設定されると、入力手続きが呼び出されます。入力手続きはいくつでも定義できます。定義された順に追加されていきます。

タイムアウト手続き

タイムアウト手続きは　指定した時間の経過後に関数を実行する場合に使用します。タイムアウトが呼び出されるのは一度だけです。したがって、タイムアウト手続きを定期的に呼び出す場合は、終了する前に、そのタイムアウト手続きを別のタイムアウト手続きとして追加する必要があります。これを行うには、次の関数呼び出しを実行します。

```
XtAppAddTimeOut(appContext, timeoutPeriod, procedure, clientData);
```

タイムアウト手続きを追加するには、「モジュール」メニューから「タイムアウト手続き」を選択してください。図 6-12 に示すダイアログが表示されます。

図 6-12 「タイムアウト手続き」ダイアログ

「タイムアウトまでの時間」はミリ秒で指定してください。

UNIX で頻繁に使用される方法としてシグナルを使った時間割り込みプログラミングがありますが、X 環境ではタイムアウト手続きの方が適しています。タイムアウト手続きはいくつでも定義できます。定義した手続きはすべて、Sun WorkShop Visual によってメインコードファイルに生成されます。

言語手続き

アプリケーションは特定のロケールの環境で動作します。キーボード入力の受け取り方、文字および日付の形式や時間文字列の表示の仕方はロケールによって決まります。そのため開発者は、世界各国で使用できるようにアプリケーションをカスタマイズできます。

Sun WorkShop Visual は、X ツールキットのルーチン `XtSetLanguageProc` への呼び出しをメインコードファイルに生成します。このルーチンへの引数の 1 つが、ロケールを設定する手続きの名前です。Xt にはデフォルトの言語手続きが用意されていますが、独自の定義付けを行なって、ロケールの設定方法を追加したり、特定のロ

ケールだけをサポートするように指定することもできます。言語手続きを定義するには、「モジュール」メニューから「言語手続き」を選択してください。図 6-13 に示す「言語手続き」ダイアログが表示されます。

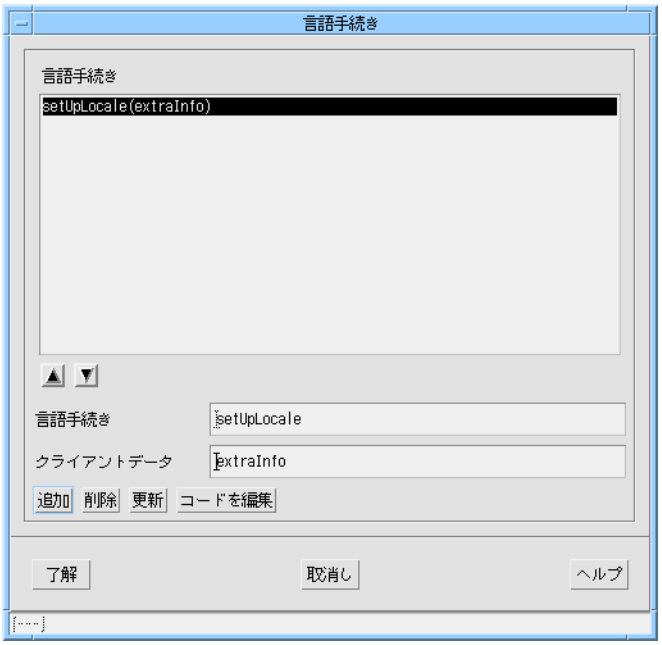


図 6-13 「言語手続き」ダイアログ

言語手続きはいくつでも指定できますが、有効になるのは 1 つだけです (`XtSetLanguageProc` の引数として渡すことのできる言語手続きは 1 つだけであるため)。Sun WorkShop Visual は、「言語手続き」ダイアログの一番上に表示されている手続きを引数として渡します。ダイアログ中の手続きの順序は、言語手続きのリストの下にある矢印ボタンで変更できます。その場合、該当する手続きを選択して、その手続きがリストの一番上に表示されるまで上向き矢印ボタンを押し続けてください。スタブは、すべての言語手続きに対してスタブファイルに生成されます。

イベントハンドラ

イベントハンドラは、ウィジェットのトランスレーションテーブルを介さない、低レベルの入力処理を実行する場合に役立ちます。イベントハンドラは、特に、多数のイベントがある場合に適しています。トランスレーションとそれに関連付したアクションでイベントハンドラと同様の機能を果たすこともできますが、イベントハンドラを使用する方がイベントの処理が簡単になります。

他の Xt 関数とは異なり、イベントハンドラはアプリケーション全体ではなく個々のウィジェットで定義されます。

イベントハンドラを追加するには、イベントが必要なウィジェットを選択した状態で「ウィジェット」メニューから「イベントハンドラ」を選択します。図 6-14 に示すダイアログが表示されます。

図 6-14　イベントハンドラのダイアログ

このダイアログには、関数を入力するためのテキストフィールドとイベントマスクがあります。テキストフィールドの横にはボタンがあり、このボタンを押すと使用可能な手続きやイベントマスクを示すサブダイアログが表示されます。新しい関数を定義できますが、イベントマスクはサブダイアログに表示されている有効なイベントマスクでなければなりません。Javaコードに適用できるイベントマスクには、図 6-15 に示す Java のコーヒーカップのマークが注釈として付けられます。

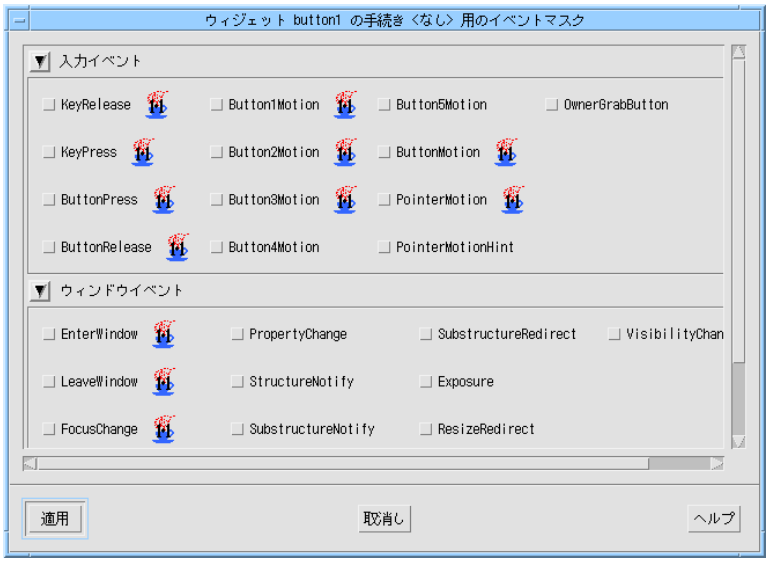


図 6-15　イベントマスク

「マスク不可」トグルがオンになっている場合は、イベントハンドラはマスクできないイベント (ClientMessage、GraphicsExpose、MappingNotify、NoExpose、SelectionClear、SelectionNotify、SelectionRequest) も受け取ります。これらのイベントは特に便利なものではないため、通常このトグルはオフに設定します。

「生データ」トグルは、raw イベントハンドラの追加を Sun WorkShop Visual に通知する場合に使用します。このイベントハンドラは、登録されたイベントに対しては即座に応答しません。ただし、 別のイベントハンドラなどによって、イベントがどこかで選択されるとハンドラが起動されます。

raw イベントハンドラの使い方の 1 つに別のイベントハンドラの「シャドウ」があります。raw イベントハンドラともう 1 つのイベントハンドラの両方が同じイベントマスク (ただし、一方だけが raw イベントマスク) で追加されると、適切なイベントが発生したときに両方のハンドラが呼び出されます。この場合、raw イベントハンドラがもう 1 つのハンドラが受け取ったイベントを記録します。

コールバック関数と同様に、同じイベントハンドラを複数 (任意) のウィジェットに対して指定できます。内容の設定は「クライアントデータ」テキストフィールドで行います。

一般的に、ウィジェットに組み込み処理に対するサポートがほとんどない場合、または大量の入力や定型的でない入力に対する処理が必要な場合には、イベントハンドラが使用されます。たとえば、描画領域ウィジェットにはイベントハンドラを使用した方がよいでしょう。

イベントハンドラは、通常、Sun WorkShop Visual によって一般的なウィジェット構成の一部としてコードファイルに生成され、スタブはスタブファイルに生成されます。

イベントハンドラには「ウィジェット注釈」があります。「表示」メニューのプルライトメニューからイベントハンドラ注釈を示す「イベント」を選択すると、イベントハンドラが定義されているウィジェットが一目でわかります。

Xt 手続きの編集

コールバックと同様に、Xt 手続きに対して生成したスタブは、好みのエディタを使用して Sun WorkShop Visual 内から編集できます。

第7章

コードの生成

はじめに

これまで、Sun WorkShop Visual の対話型機能を使用してユーザーインタフェースの動作プロトタイプを作成してきました。本章では、デザインを独立したプログラムに変換するために必要なコード生成機能について説明します。
コード生成は、C、C++、Java、または UIL で動作します。

以下の作業を進めながら説明していきます。

- デザインに対しての基本モジュールを生成する (基本モジュールにはインタフェースのプロトタイプ作成に必要なすべてのコードが含まれています)
- コールバック関数の作成に役立つスタブファイルを生成する
- プロトタイプをコンパイルしてリンクを行い、実行する
- コード中に固定されていないリソース設定を含む X リソースファイルを生成する
- *quit()* コールバックを作成する

また、さまざまなファイルに生成されたコードの分析、および種々のファイルの上手な構成の仕方についても説明します。

前提知識

生成されたコードファイルを理解し、コールバック関数のコードを作成するためには、C、C++、Java、または UIL についての知識が必要です。
また、X Window System についての知識もある程度必要となります。

生成メニュー

「生成」メニューは、ソースコード、X リソースファイルおよびメークファイルをデザインから生成するために使用します。「生成」メニューには、「C」、「C++」、「UIL」、「X リソース」、「メークファイル」、「Java」、「生成」の 7 個のオプションがあります。最初の 6 個のオプションでは、「コード生成」ダイアログで設定したファイルを生成することができます。

「生成」オプションを選択すると「コード生成」ダイアログが表示されます。

注 - Microsoft Windows モードの場合、生成メニューには Microsoft Windows リソースファイルを生成するための選択肢が含まれています。これについては、462 ページの「アプリケーションの構築」で説明します。

本章の例では、C を使用します。C++ の生成は C の場合とまったく同じであるため、学習例に C++ を使用することも可能です。UIL の手順は、 1 つの手順を除いて C の手順とほぼ同じです。異なる手順については、244 ページの「言語の設定」で説明します。

Java の生成の詳細については、363 ページの第 10 章「Java 用のデザイン」 を参照してください。

「コード生成」ダイアログ

「コード生成」ダイアログを表示するには、「生成」メニューをプルダウンして「生成」を選択します。

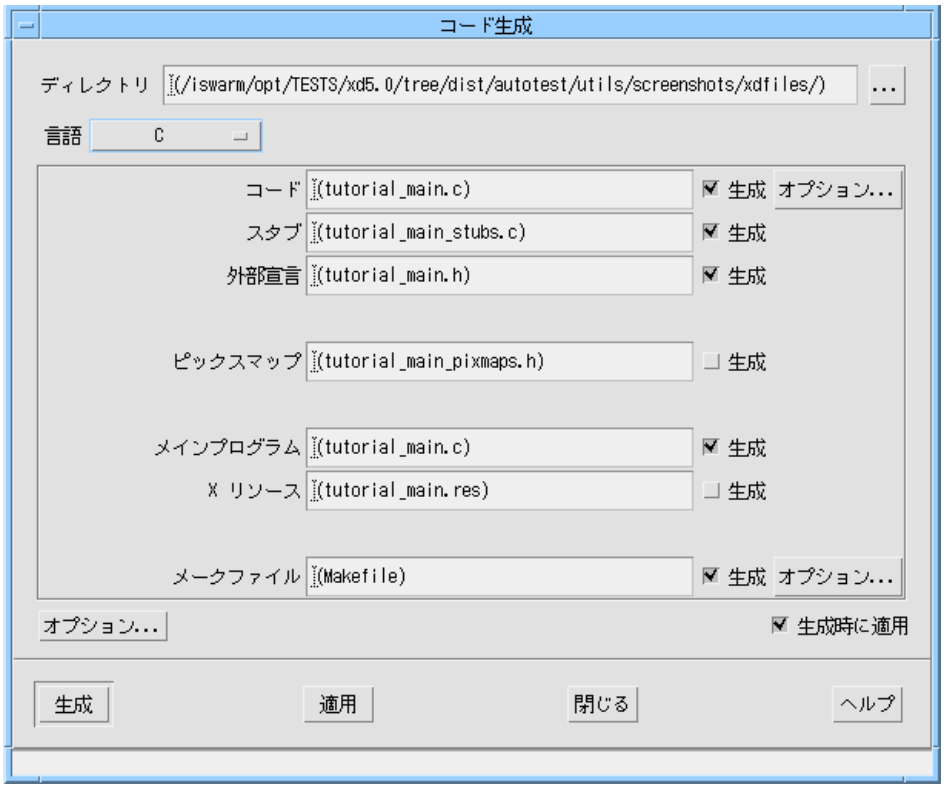


図 7-1　「コード生成」ダイアログ

このダイアログには、デザインから生成できるすべてのファイルの概要が表示されます。ダイアログには、デフォルトのファイル名を含むデフォルト設定がいくつかあります。デフォルトのファイル名は、生成されるファイルの種類および言語に適したものになっています。デフォルト設定は、アプリケーションリソースファイルで変更することができます。詳細は、997 ページの「フィルタ」を参照してください。

ダイアログの設定

生成するファイルを個々に設定する前に、すべてのファイルに影響する 2 つのオプションを設定する必要があります。

- 使用言語
- ファイルを生成するベースディレクトリ

言語の設定

言語の場合は、ダイアログの最上部にある「言語」オプションメニューを使用します。言語の選択肢としては、C、C++、UIL、Java があります。Microsoft Windows モードでは、さらに「C++ (Motif XP)」および「C++ (Microsoft Windows MFC)」という選択肢もあります。Microsoft Windows モード、Motif XP、MFC の詳細については、411 ページの第 11 章「Microsoft Windows 用のデザイン」を参照してください。この学習では C 言語を使用するため、次の作業を行なってください。

- 「言語」オプションメニューで「C」が選択されていることを確認します。

注 - 言語として「Java」を選択している場合は、「生成」ダイアログの使い方がまったく異なります。そのため、Java コードを生成するための「生成」ダイアログについては、第 10 章「Java 用のデザイン」の 389 ページの「生成ダイアログ」で説明します。

UIL 使用時の注意事項

UIL で作業を行う場合、そのコード生成関数は基本的には C の場合と同じです。しかし、UIL は C ほど強力な言語ではないため、Sun WorkShop Visual の機能の中には UIL では実装できないものがあります。デザインを完全に機能させるためには、UIL ファイルに加えて補足の C ファイルを生成する必要があります。

「言語」オプションメニューから「UIL」を選択する場合は、UIL ファイルの名前を「UIL」フィールドに、補足の C ファイルの名前を「コード」フィールドに入力します。また、コンパイルする UIL ファイルの名前も指定する必要があります。指定するには、コードの「オプション」ボタンを押し、「UID ファイル」フィールドに名前を入力します。

「インクルード」(245 ページの「基本ソースファイルの設定」を参照)、「メインプログラム」、「リンク」(254 ページの「コード生成オプション」を参照) は、 C ファイルには生成できますが、UIL には生成することができません。

UIL は Motif 固有の言語であるため、Motif ツールキット以外のウィジェットでは動作しません。デザインに他のツールキットからのウィジェットが含まれている場合には、C または C++ を使用する必要があります。

ベースディレクトリの設定

ベースディレクトリを設定するには、「ディレクトリ」というラベルが付いたテキストボックスに直接入力するか、「ブラウズ」ボタンを押します。「ブラウズ」ボタンを押すと、ファイル選択ボックスが表示され、ディレクトリの検索や選択を行うことができます。生成されるファイルのファイル名は、ベースディレクトリからの相対パスで表示されます。デフォルトでは、ベースディレクトリは保存したデザインを最後に開いたディレクトリか、デザインを新規に作成している場合は Sun WorkShop Visual を呼び出したディレクトリです。デフォルトは括弧で囲まれて表示されますが、.xd ファイルには保存されません。明示的に名前を付けたディレクトリが保存されます。

ファイル名テキストボックスには、絶対パス名を入力することができます。しかし、ファイルごとにこのパスを入力しなければならないため、この方法はお勧めできません。最初にディレクトリを設定し、すべてのファイルをそのディレクトリからの相対と見なした方が、簡単で入力ミスも少なくなります。

基本ソースファイルの設定

「コード」というラベルが付いたテキストボックスに、基本ソースファイルのファイル名を入力します。このファイルの説明については、269 ページの「基本モジュールの分析」を参照してください。

1. 「コード」テキストボックスに次のように入力します。

   `icecream.c`

   規約により、基本ソースファイルおよびスタブファイルを含むすべての C ファイルには、接尾辞 .c が付けられます。

2. テキストボックスの横にある「生成」トグルが設定されていることを確認します。

   「生成」トグルを設定しているファイルだけが生成されます。

3. 「コード」ファイル名の横にある「オプション」というラベルが付いたボタンを押します。

   基本ソースファイルに関連するオプションが表示されます。

図 7-2　基本ソースファイルの「C コードのオプション」ダイアログ

基本ソースファイルの「C コードのオプション」ダイアログには、以下のオプションがあります。

- ANSI C
  Sun WorkShop Visual に ANSI C を生成させたい場合は、このオプションを選択します。

- Motif ヘッダーファイルをインクルード
  このオプションは、デフォルトで選択されています。このオプションの選択を解除する場合は、必ず Motif ヘッダーをインクルードする場所を別に指定する必要があります。指定しないと、コンパイルエラーが起こります。学習のための例を実行する場合は、このオプションを選択してください。このオプションは、Microsoft Windows モードで MFC 様式のコードを生成する場合は表示されません。

- MFC ヘッダーファイルをインクルード
  このオプションは、Microsoft Windows モードで MFC 様式のコードを生成する場合にのみ表示されます。このオプションはデフォルトで選択されています。選択されている場合は、MFC 様式のコードファイルにヘッダーファイルがインクルードされます。

- ヘッダーファイルをインクルード
  基本ソースファイルにヘッダーファイルをインクルードするための行を追加する場合は、このオプションを選択してヘッダーファイルの名前を入力します (「ヘッダーファイル」と「外部宣言ファイル」は、Sun WorkShop Visual と『Sun WorkShop Visual ユーザーズガイド』では同一のものを指します)。外部宣言ファイル (249 ページの「外部宣言ファイルの設定」を参照) を生成する場合は、通常、基本ソースファイルにその外部宣言ファイルがインクルードされるようにそのファ

イルの名前を入力します。学習のための例ではヘッダーファイルをインクルードしないため、このオプションは選択しないでください。自身で作成したソースコードにヘッダーファイルをインクルードする際の補足情報については、248 ページの「ヘッダーファイルをインクルードする際の注意事項」を参照してください。

注 - コードファイルとメインプログラムファイルを別々に生成する場合は、両方のファイルで同一の変数名が使用されるように、生成された外部宣言ファイルをインクルードしなければなりません。

- **ピックスマップ・ファイルをインクルード**
  これは「ヘッダーファイルをインクルード」オプションと類似しています。独立したピックスマップファイルを生成して、生成したピックスマップファイルを基本ソースファイルにインクルードしたい場合は、このトグルをオンにしてテキストボックスにピックスマップファイルの名前を入力します。これにより、明示的なピックスマップ定義の代わりに *#include* 指令が基本ソースファイルに生成されます。ピックスマップファイルの生成については、250 ページの「ピックスマップファイルの設定」を参照してください。学習用の例を実行する場合は、このオプションは選択しないでください。

- **UID ファイル**
  UIL を生成した場合にのみ有効になります。このテキストボックスには UID ファイルの名前を指定します。詳細は、244 ページの「UIL 使用時の注意事項」を参照してください。

4. 「了解」ボタンを押して「C コードのオプション」ダイアログを閉じます。
5. 「コード生成」ダイアログの「適用」ボタンを押します。

   これによって、コードを生成しなくても設定内容が保存されます。

#include 生成制御

デザインの一部として、生成したコードにインクルードファイルとして追加されるようにファイルの名前を指定できます。デフォルトでは、Sun WorkShop Visual はインクルード文を次のような山括弧 < > で囲みます。

```
#include <externs.h>
```

この動作はすべての言語型や構成に適しているわけではありません。次のリソースを使用して、「#include」文の生成方法を制御できます。

```
visu.defaultIncludeInQuotes: true

visu.defaultIncludeObjectFileInQuotes: false
```

1 番目の文は、外部宣言ファイルを引用符で囲むか囲まないかを制御します。2 番目の文は、Sun WorkShop Visual が自動生成するヘッダーファイル (`xdclass.h` ファイル) の生成を制御します。

この機能はデフォルトの動作に優先します。基本ソースファイルの「コードのオプション」ダイアログでヘッダーファイルの名前を引用符 " " または 山括弧 < > で明示的に囲んで入力すると、デフォルト値は無効になります。

ヘッダーファイルをインクルードする際の注意事項

外部のヘッダーファイル (Sun WorkShop Visual を使用しないで作成したもの) と Sun WorkShop Visual を使用して作成した外部宣言ファイルの両方をコードファイルにインクルードしたい場合は、外部のヘッダーファイルが Sun WorkShop Visual の生成した外部宣言ファイルをインクルードしていることを確認して、「ヘッダーファイルをインクルード」テキストボックスに外部のヘッダーファイルの名前を入力してください。

「コードのオプション」ダイアログの「ヘッダーファイルをインクルード」トグルをオンに設定して外部宣言ファイルの名前をテキストボックスに入力すると、そのファイルがコードファイル、(生成する場合は) スタブファイル、および (コードファイルとは別に生成される場合は) メインプログラムファイルにインクルードされます。

コードファイルとメインプログラムファイルを別々に生成する場合は、外部宣言ファイルで定義された変数名が両方のファイルで使用されるため、必ず外部宣言ファイルをインクルードしてください。

スタブファイルの設定

デザインのすべてのコールバックおよびコールバック・メソッドに対して、コールバックスタブ、すなわち指定したコールバック名またはメソッド名を持つ「空」ルーチンが生成されます。これらのルーチンは、スタブファイルという独立したソースファイルに生成されます。デザインにコールバックまたはメソッドがある場合は、コールバックは基本ソースコードから参照されるため、スタブファイルを生成することによってアプリケーションをコンパイルすることができます。ただし、必要な関数は自身でコールバックルーチンに追加しなければなりません。コールバックの追加については、264 ページの「コールバック関数の追加」を参照してください。

Sun WorkShop Visual はコールバックスタブとともに特別なコメントを生成します。これらのコメントについては、268 ページの「スタブファイルのコメント」を参照してください。

現在、学習用のデザインにはコールバック関数 `quit()` 1 つだけが含まれています。次に空の `quit()` 関数でスタブファイルを生成します。この空の関数により、アプリケーションをプロトタイプとしてコンパイル、リンク、実行することができます。264 ページの「コールバック関数の追加」で、`quit()` に本当の機能を追加してアプリケーションを完成させます。

1. 「スタブ」テキストボックスに次のように入力します。
   `stubs.c`
2. 「スタブ」テキストボックスの横にある「生成」トグルをオンにします。
   「生成」トグルがオンに設定されているファイルだけが生成されます。

   スタブファイルには個別のオプションはありません。

外部宣言ファイルの設定

Sun WorkShop Visual は、スコープが大域的なすべてのウィジェットに対しての外部宣言、デザインのための C++ クラス定義、C 構造体の定義を含むヘッダーファイルを生成することができます。大域的ウィジェットには、ユーザーが明示的に命名したウィジェットおよびコアリソースパネルにおいて大域的として指定したウィジェットが含まれます。

外部宣言ファイルを設定するには、「外部宣言」テキストボックスにファイルの名前を入力し、「生成」トグルをオンにします。規約により、外部宣言ファイルには接尾辞 `.h` を付けます。

- 学習のための例には外部宣言ファイルは必要ないので、「外部宣言」というラベルが付いたテキストボックスの横にある「生成」トグルが設定されていないことを確認してください。

  生成された外部宣言ファイルを基本モジュールに組み込む方法については、245 ページの「基本ソースファイルの設定」の「コードのオプション」ダイアログの記述を参照してください。Sun WorkShop Visual は明示的な型定義の代わりに *#include* 指令を基本ソースファイルに生成します。ユーザーがこの動作を行う場合、大域的ウィジェットは基本ソースファイルに割り当てられたままです。

外部宣言ファイルは、大域的ウィジェットの呼び出し、または型定義を参照するスタブファイルやその他のコードファイルへの組み込みを行う場合にも役に立ちます。詳細は 209 ページの「大域的ウィジェット変数」 を参照してください。

外部宣言ファイルには、個別のオプションはありません。

ピックスマップファイルの設定

ピックスマップファイルは、外部宣言ファイルと類似しています。デザイン内にあるすべてのピックスマップの静的宣言を持つヘッダーファイルを生成します。ピックスマップファイルを生成することで、扱いにくいピックスマップ構造の定義を、基本ソースファイルとは別のファイルに入れておくことができます。

ピックスマップファイルを生成するためには、「ピックスマップ」テキストボックスにファイルの名前を入力し、対応する「生成」トグルをオンにします。規約により、ピックスマップファイルには接尾辞 .h を付けます。

- 学習用の例ではピックスマップファイルは必要ありません。「ピックスマップ」というラベルが付いたテキストボックスの横にある「生成」トグルが設定されていないことを確認してください。

メインプログラムファイルの設定

メインプログラムファイルとは、*main()* 関数を含むファイルです。独立したファイルを生成することによって、この関数を残りのソースコードと切り離すことができます。これは、ユーザーインタフェースを開始する前に、独自の初期化を実行したり、アプリケーションの別の場所を呼び出すように関数を編集する必要がある場合に便利です。このファイルを編集したときは、生成は必ず 1 回だけにしてください。以降の生成が変更内容を上書きするためです。*main()* 関数の詳細は、274 ページの「main() 関数プログラム」を参照してください。

main() 関数を基本ソースファイルに生成したい場合は、「メインプログラム」というラベルが付いたテキストボックスの名前が「コード」テキストボックスの名前と同じで、かつ対応する「生成」トグルが設定されていることを確認してください。

学習用の例では、以下の作業を行います。

1. ベースディレクトリが設定されているかどうかを調べます。

   245 ページの「ベースディレクトリの設定」を参照してください。

2. 「メインプログラム」というラベルが付いたテキストボックスに **icecream.c** と入力します。
3. 「メインプログラム」テキストボックスの横にある「生成」トグルが設定されていることを確認します。

   メインプログラムには、個別のオプションはありません。

X リソースファイルの設定

第 3 章「リソース」 で記述したように、リソース設定は有効でなければなりません。そうでない場合は、インタフェース実行時にリソースが適用されません。リソース値は、基本ソースファイルまたは X リソースファイルのどちらかで有効にすることができます。ソースファイルへリソースを生成することを、リソースをハードワイヤするといいます。

1. ベースディレクトリが設定されいることを確認してください。

   245 ページの「ベースディレクトリの設定」を参照してください。
2. 「X リソース」とラベルの付いたテキストボックスに次のように入力します。
   icecream.res
3. テキストボックスの横にある「生成」トグルを設定します。

   オプションメニューを使用して、生成するリソースおよび生成先のファイルを設定することができます。詳細は 254 ページの「コード生成オプション」 を参照してください。

X のリソースファイル認識

X は `icecream.res` をアプリケーションのリソースファイルとして自動的に認識するわけではありません。このファイルを検索する場所を X が認識できるように、指定されたアプリケーションリソース・ディレクトリ (POSIX 準拠システム上では `/usr/lib/X11/app-defaults`) にリソースファイルをコピーすることをお勧めします。そのディレクトリ内のファイル名はアプリケーションクラス名で、接尾辞を持たない「*XDTutorial*」となります (アプリケーションのクラス名の設定方法については 254 ページの「コード生成オプション」を参照してください)。この方法により、アプリケーション固有のリソースファイルと、使用する他のファイルとの混同を避けることができます。

アプリケーションリソースディレクトリへのアクセス権が許可されていない場合は、以下の操作を行います。

4. 環境変数 XENVIRONMENT をリソースファイルのファイル名に設定します。

   実際の構文は、使用するシェルによって異なります。C シェルの場合は、次のように入力します。

   ```
   setenv XENVIRONMENT icecream.res
   ```

   Bourne シェルの場合は、次のように入力します。

   ```
   XENVIRONMENT=icecream.res export XENVIRONMENT
   ```

   別の方法で X に X リソースファイルを認識させることもできます。詳細は、X ウィンドウシステムの資料を参照してください。参考文献の名前については、付録 E「参考資料」を参照してください。

メークファイルの設定

Sun WorkShop Visual は、正しい依存性を持ち、デザインに必要なすべてのファイルに対するコンパイル命令を作成するメークファイルを生成することができます。学習用の例では、メークファイルは、`icecream.c` と `stubs.c` の両ファイルをコンパイルし、その結果として生じるオブジェクトファイルを必要なライブラリにリンクします。

1. 基本ソースファイルとスタブファイルがこれまで生成されていたディレクトリまたは今後生成されるディレクトリに、ベースディレクトリが設定されていることを確認します。

2. 「メークファイル」というラベルが付いたテキストボックスに「**Makefile**」と入力し、「生成」トグルが設定されていることを確認します。

3. メークファイルテキストボックスの横にある「オプション」というラベルが付いたボタンを押します。

   図 7-3 に示すような「メークファイルオプション」ダイアログが表示されます。

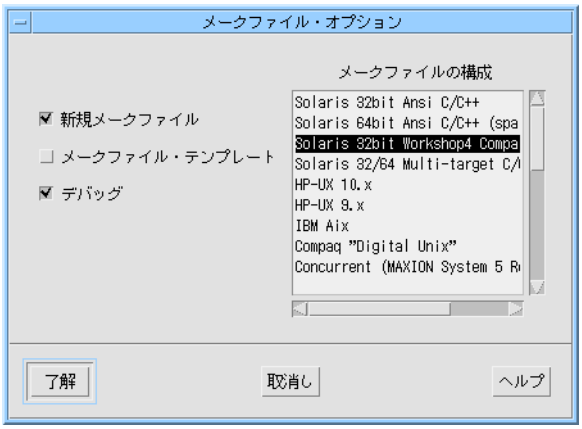


図 7-3　「メークファイルオプション」ダイアログ

Sun WorkShop Visual は、メークファイルを生成するために内部メークファイルテンプレートを使用します。内部メークファイルテンプレートには、各種プラットフォームおよび環境に関する情報が含まれています。 Sun WorkShop Visual は、テンプレート機能を使用して、複数のデザインから生成されたソースを 1 つのアプリケーションに構築するメークファイルを作成することもできます。

「メークファイルオプション」ダイアログの「新規メークファイル」と「メークファイルテンプレート」トグルで、生成するメークファイルの種類を指定します。生成できるメークファイルは、1 つのデザインからソースを構築する簡易メークファイルと、ソースをさらに追加できるテンプレート付きメークファイルです。「デバッグ」トグルは、コンパイル時に -g フラグを使用するかどうかを制御します。

このダイアログの右側には、Sun WorkShop Visual が自動的にメークファイルを生成できるプラットフォームとコンパイラを示したリストがあります。このリストには、64 ビットのアプリケーションをコンパイルするためのオプションもあります。このダイアログの詳細については、639 ページの第 19 章「メークファイル生成」を参照してください。

この節では、学習用デザインからソースを構築することのできる簡易メークファイルの生成方法について説明します。テンプレートの使い方やメークファイルのカスタマイズなど、メークファイル生成の詳細については、639 ページの第 19 章「メークファイル生成」 を参照してください。メークファイルテンプレートのアプリケーションリソースを変更する方法については、付録 D「アプリケーションのデフォルト」の 1001 ページの「生成」を参照してください。

1. 学習用アプリケーションを実行する場合は、「新規メークファイル」トグルをオンにして、「メークファイルテンプレート」トグルはオフにしてください。

   学習用のアプリケーションにはこれ以上ソースファイルは追加しないので、テンプレートコメントは必要ありません。

2. 必要に応じて「デバッグ」トグルをオフにできます。

   この学習用アプリケーションではデバッグは行いません。

3. ダイアログの右側のリストで選択されている項目が、ターゲットプラットフォームおよびコンパイラと一致していることを確認します。

   デフォルトで選択されている項目をそのまま使用できますが、一致しているかどうか分からない場合はシステム管理者に確認してください。このリストの詳細については、641 ページの「メークファイルの種類のリスト」を参照してください。

4. 「了解」を押して変更内容を保存し、「メークファイルオプション」ダイアログを閉じます。

   「生成」ダイアログに戻ります。

コード生成オプション

ファイルを生成する前に、コード生成オプションを変更する必要があるかどうかを必ず確認してください。「コード生成」ダイアログの最下部にある「オプション」ボタンを押すと、図 7-4 に示すダイアログが表示されます。

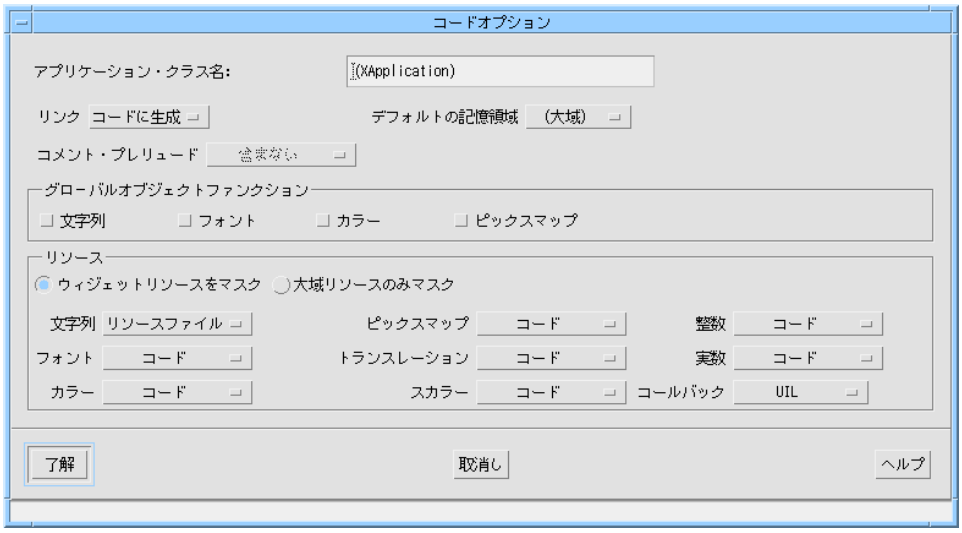


図 7-4　「コードオプション」ダイアログ

このダイアログを使用して、リンク用コードの生成場所、リンク用コードを生成するかどうかの制御、アプリケーションクラス名の指定、各種リソースの生成場所の制御を行うことができます。以下で、それぞれのオプションを説明します。

アプリケーション・クラス名

アプリケーション・クラス名は、X リソースファイルを生成する際にリソース設定を識別するために使用されます。リソース値がシステム全体の X リソースと混同されないようにするため、デフォルトの「XApplication」以外の名前を割り当てます。

リンクのコード生成

「リンク」というラベルが付いたオプションメニューを使用すると、リンクを基本ソースファイルに生成する (コード生成)、スタブファイルに生成する、まったく生成しないのうちのいずれかを選択することができます。または、リンクコード宣言だけを生成するよう選択することもできます。

リンク関数の一部として生成されたコードは、汎用関数セット (各種リンク用) およびこれらの関数の宣言で構成されています。関数は常に同じなので、アプリケーション全体で 1 セットだけ関数を生成すれば十分です。複数のデザインからコードを生成

し、各デザインにリンクが含まれている場合は、関数の生成は 1 度だけで十分ですが、宣言はデザインごとに生成する必要があります。宣言は常に基本ソースファイルに生成されます。

大域オブジェクト関数

Sun WorkShop Visual では、カラー、フォント、文字列、ピックスマップの各オブジェクトの大域アクセス関数を生成できます。そのため、アプリケーションの複数のファイルでこれらのオブジェクトを共有することが可能です。「コードオプション」ダイアログのトグルを使用して、カラー、フォント、文字列、ピックスマップの各オブジェクトを大域的にすることができます。これらのトグルを選択すると、Sun WorkShop Visual は、オブジェクトを返すオブジェクトごとに関数を 1 つずつ作成します。Sun WorkShop Visual は、次のアルゴリズムを使用してオブジェクトの名前で関数の名前を構築します。

```
xdGet + <object_name> + <object_type> + Object
```

<object_name> にはオブジェクトにバインドした名前を指定します。*<object_type>* にはオブジェクトの型に合わせて「`Color`」、「`Font`」、「`String`」、「`Pixmap`」のいずれかを指定します。たとえば、「`Foreground`」というカラーオブジェクトは、デフォルトでは次のアクセス名で生成されます。

```
xdGetForegroundColorObject
```

`xdGet` タグはリソースを介して上書きできます。

```
visu.globalObjectFunctionStart: xdGet
```

たとえば、次の場合を想定します。

```
visu.globalObjectFunctionStart: my
```

さらに、1 つのカラーオブジェクトが「`Background`」の場合、カラーの大域オブジェクトトグルがオンになっていると、次のコードが生成されます。

```
Pixel myBackgroundColorObject() {
                    ...
}
```

言語が MFC の場合は次のコードが生成されます。

```
CBrush myBackgroundColorObject() {
                    ...
}
```

デフォルト記憶領域の制御

Sun WorkShop Visual は、デフォルトで、作成される関数に対する局所変数として各ウィジェットを生成します。ウィジェットに名前を指定しても、Sun WorkShop Visual は変数を大域的にします。名前付きウィジェットの場合、このデフォルト動作は、「コードオプション」ダイアログで「デフォルトの記憶領域」オプションメニューを変更して制御できます。

ウィジェットがクラスに属している場合でも (クラスを作成しているウィジェットの子の場合など)、「コードオプション」ダイアログで「デフォルトの記憶領域」オプションを選択していても選択していなくても、ウィジェットは属しているクラスの変数を保持します。

名前付きウィジェットのデフォルトの記憶領域は「大域」です。次のリソースを設定して、このデフォルトを変更できます。

```
visu.defaultStorageType:static
```

このリソースで使用できる値は次のとおりです。

- 「(大域)」(Sun WorkShop Visual で通常使用するデフォルト。この場合は「大域」)
- 「大域」
- 「常に表示」

コメントプレリュード

このオプションメニューは、生成されたコードの中でプレリュードを追加できる領域の周囲に記述される、特別なコメントを参照します。プレリュードの追加については、280 ページの「生成されたファイルのカスタマイズ : プレリュード」を参照してください。コメントの追加を「含む」(プレリュードが指定されているかどうかに関係なく)、「定義時にのみ含む」(すなわちプレリュードが指定された場合のみ)、「含まない」のうちのいずれで行うかを選択できます。デフォルトは「含まない」です。

以下の例は、シェルのマネージ前プレリュードのコメントを示しています。コメント内にプレリュードがないので、「含む」が選択されたオプションであると見なします。

```
/* visu: prelude for shell1: pre-manage >>> */


/* <<< pre-manage ends. */
XtSetArg(al[ac], XmNallowShellResize, TRUE); ac++;
```

```
shell1 = XmCreateDialogShell ( parent, "shell1", al, ac );
                ac = 0;
                /* visu: prelude for form1: pre-create >>> */

                /* <<< pre-create ends. */
                XtSetArg(al[ac], XmNautoUnmanage, FALSE); ac++;
                form1 = XmCreateForm ( shell1, "form1", al, ac );
```

リソースの操作

「オプション」ダイアログの中央のパネルには各リソース型が列挙されています。

Motif の定義しているリソース型を以下に示します。

- 文字列 (NULL 終了文字列、および XmString を含む)
- フォント
- カラー
- ピックスマップ
- トランスレーション
  「ウィジェット」メニューにあるコマンドで定義されます。221 ページの「トランスレーションとアクション」を参照してください。
- スカラー (ブール型を含む複数選択肢を持つリソース)
  ほとんどのリソースパネルでは、これらのリソースは「設定」のページにあります。
- 整数 (スカラーではない整数リソース設定)
- 実数 (浮動小数点数リソース値)

各リソースが生成されるファイルを指定することができます。あるリソースがソースコードに生成された場合は、ソースコードにハードコードされるため、リソースファイルからは変更できません。通常は、ソースコードに生成されなかったリソースはすべて X リソースファイルに生成されます。X リソースファイルは、アプリケーションのユーザーが編集することのできるファイルです。Xリソースファイルについての詳細は、251 ページの「X リソースファイルの設定」を参照してください。

「コールバック」とラベルのついたオプションメニューは、言語型に UIL を選択している場合だけに表示されます。このオプションで、コールバックを UIL コードと C コードのどちらに登録するか選択することができます。デフォルトでは UIL に登録されます。ただしクライアントデータを使用している場合は、構造型が UIL では定義されていないため、コールバックを C コードに登録する必要があります。C および UIL ファイルについては、244 ページの「言語の設定」を参照してください。

学習の例を実行する場合は、「コードオプション」ダイアログで以下の変更を行います。

1. 「リンク」オプションメニューから「コードに生成」を選択します。
2. 「アプリケーション・クラス名」テキストボックスに、次のように入力します。
   `XDTutorial`
3. リソースの種類のオプションメニューが、図 7-5 に示すとおりに設定されていることを確認します。

   「コールバック」オプションメニューが表示されることに注意してください。このメニューは、言語型に UIL を選択している場合だけに表示されます。
4. 「ウィジェットリソースをマスク」ラジオボタンがオンになっていることを確認します。

   このラジオボタンの意味については、260 ページの「リソースのマスク」で説明します。
5. 「了解」をクリックします。

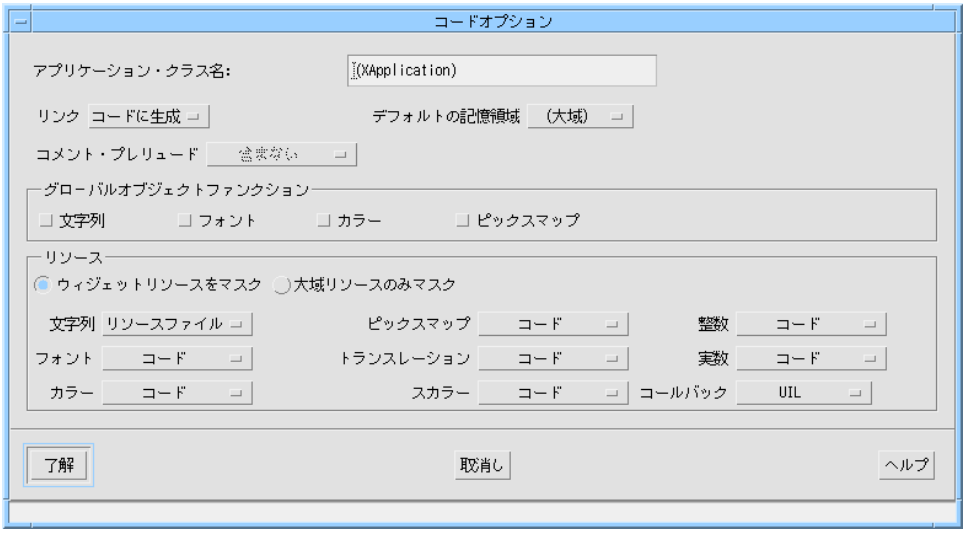


図 7-5　コード生成のリソース設定 (言語型 UIL)

リソースのマスク

リソースパネルには、図 7-6 に示すように各リソースの横にラベルの付いていないトグルがあります。

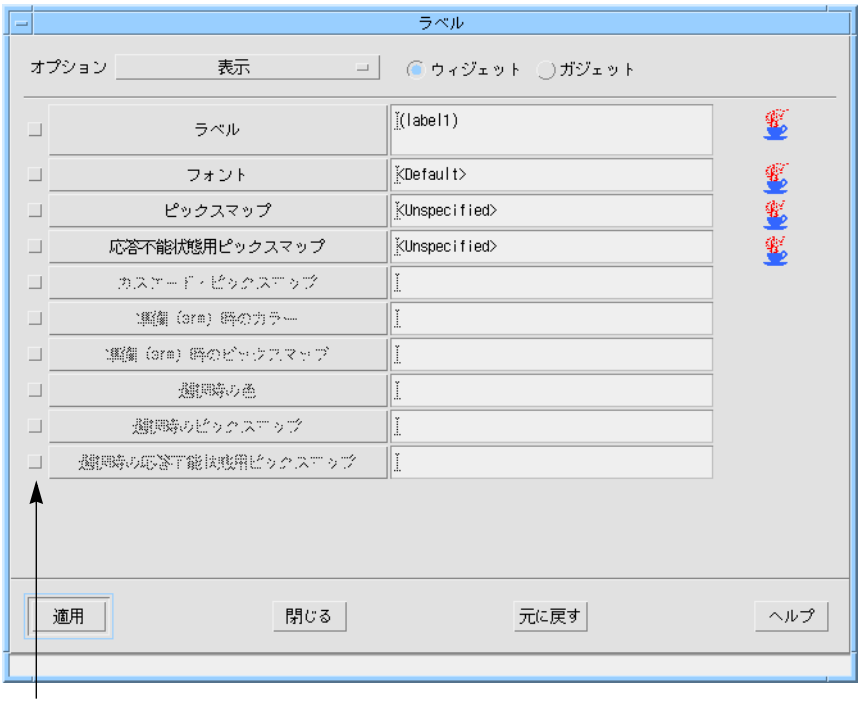


図 7-6　リソースパネルのマスクトグル

これらのトグルは、「コードオプション」ダイアログの「ウィジェットリソースをマスク」ラジオボタンおよび「大域リソースのみマスク」ラジオボタンとともに機能します。これらのトグルをすべて使用すると、個別にリソースの生成を制御することができます。

ウィジェットリソースのマスク

「ウィジェットリソースをマスク」を設定している場合は、以下の説明を参照してください。

個々のリソースがリソースパネルトグルを設定していない場合は、「コード生成」ダイアログ内の該当する種類のオプションメニューに従ってリソースが生成されます。たとえば、「ラベル」のリソースパネルのトグルがオフに設定されている場合は、「文字列」というラベルが付いたオプションメニューで指定されたファイルに対して、ラベル文字列が生成されます。

個々のリソースがリソースパネルトグルを設定している場合は、「コード生成」ダイアログ内のオプションメニューによって該当する種類に指定されたファイルとは「反対の」ファイルに対して、リソースが生成されます。たとえば、リソースパネルトグルをオンにした状態で「整数」オプションメニューが「コード」に設定されている場合はリソースファイルに対して、また、オプションメニューが「リソースファイル」に設定されている場合はコードファイルに対して整数リソースが生成されます。

言い換えると、生成ダイアログのオプションメニューが規則を確立し、リソースパネルにあるトグルがこの規則の例外を識別します。

大域リソースのみマスク

「大域リソースのみマスク」を設定している場合は、以下の説明を参照してください。

生成ダイアログのオプションメニューは、この段階で大域オブジェクト (フォント、カラー、ピックスマップオブジェクト) にのみ適用します。これらのオブジェクト (のみ) は、すべてのリソースについて 261 ページの「ウィジェットリソースのマスク」で記述したとおりに制御されます。

その他のすべてのリソースは、個々のリソースパネルトグルによってのみ制御されます。それらのリソースは、リソースパネルトグルがオフの場合はコードファイルに、リソースパネルトグルがオンの場合はリソースファイルに生成されます。

マスク効果の例

多くの場合、設計者は編集が容易に行えるように、ほとんどの文字列を X リソースファイルに生成しようとします。これにより、X リソースファイルを編集するだけで、他の言語のアプリケーションを生成することができます。このためには、文字列を X リソースファイルに生成すると効果的です。しかし、会社の住所など、ユーザーが変更できないようにしたい文字列もいくつか存在します。これらの文字列リソースは、個々のマスクトグルを設定することにより、ハードワイヤすることができます。

同様に、企業のイメージカラーとして決まっている色を除く色をユーザーが編集できるようにしたい、という場合もあるでしょう。これを行うには、会社のイメージカラーを制御する個々のリソースについてマスクトグルを設定します。コードの生成時には、リソースファイルにグループ化して色を生成します。Sun WorkShop Visual はタグの付けられたリソースをソースコードにハードワイヤします。

デフォルト設定

リソースパネル上の括弧内に示されるデフォルトリソース値は、どちらのファイルにも生成されることはありません。この場合、Motif は実行時にリソース値を計算します。結果は、プログラムを実行するプラットフォームによっては、インタフェース構築時に確認したデフォルト値と異なる場合があります。デフォルト値を使用すると、多くの場合はアプリケーションに移植性を持たせる際に役立ちます。

コード生成ダイアログの終了

生成したいファイルおよびそのオプションの設定を「コード生成」ダイアログで終了したら、「適用」ボタンを押して設定を保存するか、「生成」ボタンを押してすぐにファイルを生成します。「適用」ボタンを押した場合は、生成する準備が完了したときに「コード生成」ダイアログを呼び出して「生成」ボタンを押します。または、「コード生成」ダイアログを表示せずにツールバーの関連するボタンを選択して、生成することもできます。この場合、最後に適用した設定が使用されます。「生成時に適用」トグルを設定して「生成」ボタンを押すと、「適用」ボタンを押して「生成」ボタンを押した場合と同様の処理が行われます。

学習例でのコード生成

生成したいファイルおよびそれに関連するオプションを生成ダイアログで設定したら、コードを生成し、プロトタイプのアプリケーションを実行します。

1. プロトタイプですべてのファイルを生成するには、生成ダイアログを表示して「生成」ボタンを押します。
2. 「コード生成」ダイアログ上で、現作業ディレクトリが、「コード生成」ダイアログで指定した (ファイルが生成された) ディレクトリであることを確認します。
3. `$VISUROOT` に Sun WorkShop Visual のインストールディレクトリのパス名を設定します。
4. プロトタイプを構築するには、次のように入力します。

   ```
   make
   ```

5. X が X リソースファイルを見つけられることを確認します。

   251 ページの「X リソースファイルの設定」を参照してください。

6. プロトタイプを実行するには、次のように入力します。

```
icecream
```

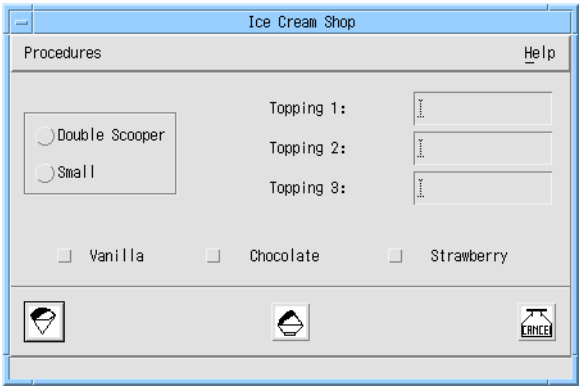


図 7-7　実行中のインタフェースのプロトタイプ

ダイナミックディスプレイの場合と同じように、プロトタイプにあるすべてのウィジェットが機能します。ボタンのクリック、メニューのプルダウンなどを行うことができます。さらに、プロトタイプにリンクを組み込むこともできます。画面右側にあるメニューをプルダウンしてその中の 1 つの項目をクリックすることにより、ヘルプ画面を表示することができます。ヘルプ画面内のボタンをクリックすると、ヘルプ画面が閉じます。

プロトタイプでも、*exit_button* がクリックされると *quit()* が呼び出されますが、*quit()* を機能させるためのコードがまだないため、動作は何も実行されません。このコードの追加については、以下のセクションで説明します。プロトタイプを確認し終わったら、プロトタイプを終了します。

7. ウィンドウマネージャを使用してメインウィンドウを閉じます。

コールバック関数の追加

quit() に関数を追加すると、本章での学習は完了です。Sun WorkShop Visual では、スタブファイル内のコールバックを直接編集する手段が提供されています。

Sun WorkShop Visual 内でのコールバックコードの編集

「コールバック」ダイアログには「コードを編集」というラベルが付いたボタンがあります。このボタンの横にあるオプションメニューで、Sun WorkShop Visual にスタブファイルの使用言語を指示することができます。これにより、C と C++ の両言語を生成している場合に曖昧さが生じることを防ぐことができます。通常は、Sun WorkShop Visual が正しい値を選択するため、このオプションメニューを変更する必要はありません。「コールバック」ダイアログについては、199 ページの「コールバックのデザイン」を参照してください。

「コードを編集」ボタンを押すと、Sun WorkShop Visual は Sun WorkShop 編集サーバーを使用します。別の編集ウィンドウにスタブファイルが開かれて、選択したコールバックの関数角括弧の中の挿入ポイントに入力できます。他のコールバックを選択すると、常に正確な場所に入力できるように、挿入ポイントがファイル内を移動します。「編集サーバー」ウィンドウにまだ保存していない変更内容がある場合にスタブファイルの生成を試みると、まず変更内容を保存するように促されます。

スタブファイルの編集の前に、Sun WorkShop Visual は最後のスタブファイル生成以降にデザインに変更があったかどうかをチェックします。変更が見つかった場合は、コードを再生成するかを編集の前に尋ねます。再生成しない場合は、ファイルを編集することはできません。

スタブファイルを再生成しようとするときに編集ウィンドウに保存していない変更がある場合は、まずそのファイルを保存するかどうかを尋ねるメッセージが表示されます。最初にファイルを保存すると、追加したコールバックコードはすべて保持されます。詳細は、267 ページの「追加のスタブファイル生成」を参照してください。変更を保存しないで再生成するように選択すると、最後の保存以降の変更はすべて失われます。

コールバックの編集

Sun WorkShop Visual を使用せずにコールバックを編集する場合は、次のようにします。

- テキストエディタを使用して **`stubs.c`** を開き、手順 5 へ進みます。

Sun WorkShop Visual を使用してコールバックを編集する場合は、次のようにします。

1. ウィジェット *exit_button* を選択します。

2. ツールバー上の「コールバック」ボタンをクリックするか、「ウィジェット」メニューから「コールバック」を選択します。
3. コールバックのリストから quit() を選択します。
4. 「コードを編集」というラベルのついたボタンを押します。

   ファイル `stubs.c` が開きます。

スタブファイルの編集

`quit()` に対してのコードは次のようになります。

```
void
quit (Widget w, XtPointer client_data, XtPointer xt_call_data )
{
                XmPushButtonCallbackStruct *call_data =
(XmPushButtonCallbackStruct *) xt_call_data;
}
```

5. quit() の中括弧 {} の間のテキストを次の内容で置き換えます。

   `exit (0);`
6. ファイルを保存します。
7. 実行可能モジュールを作り直し、プログラムを実行します。

   詳細は 263 ページの「学習例でのコード生成」を参照してください。

   「Exit」ボタンが機能するようになります。
8. プログラムを終了するには、インタフェースの Procedures (procedure_cascade) メニューをプルダウンして、「Exit」ボタンをクリックします。

   コールバックの詳細は、コールバックに引き渡される引数、およびデザイン内のウィジェットの呼び出し方法や操作方法も含め、206 ページの「コールバック関数」で説明します。

   これで、学習の例は終了しました。本章の残りでは、前節で取り上げた問題をさらに詳細に説明します。

追加のスタブファイル生成

後に続けて同じスタブファイルを生成する場合、Sun WorkShop Visual はファイル内の特殊なコメントから、どのコールバックおよびメソッドがすでに生成されているかを判断します。このようにして、ユーザー独自のコードをスタブに追加すると、スタブは上書きされません。

Sun WorkShop Visual は、新しいコールバックまたはメソッドをスタブファイルの末尾に追加します。新しいスタブファイルが生成された場合には、古いバージョンは、指定された名前に接尾辞 .bak の付いたファイルにコピーされます。

注 - Sun WorkShop Visual 内でウィジェットからコールバックを削除した場合は、スタブファイルの中にコールバックのスタブが残されます。ルーチンを削除したい場合は、そのスタブファイルを開いて、コールバックルーチンの上にあるコメントを削除する必要があります。

以下に、quit という名前のコールバックを 1 つ含むスタブファイルの例を示します。

```
/*
** Generated by Sun WorkShop Visual
*/
```

プレリュードの開始

```
/*
** Sun WorkShop Visual generated prelude.
** Do not edit lines before "End of Sun WorkShop Visual generated
prelude"
** Lines beginning ** Sun WorkShop Visual Stub indicate a stub
** which will not be output on re-generation
*/
/*
**LIBS: -lXm -lXt -lX11
*/
```

プレリュードの終了

```
/* End of Sun WorkShop Visual generated prelude */
```

スタブを示す特殊コメント

```
/*
** Sun WorkShop Visual Stub about_sh_c::DoSetText
*/
void
quit (Widget w, XtPointer client_data, XtPointer xt_call_data )
{
                    XmPushButtonCallbackStruct *call_data =
                        (XmPushButtonCallbackStruct *)
xt_call_data;
}
```

スタブファイルのコメント

ファイルの先頭に、Sun WorkShop Visual が読み出し、最終的には破棄するプレリュードがあります。このプレリュードは常に新しく再生成されます。
Sun WorkShop Visual は、すべてのスタブの前にコールバックまたはメソッドの名前を示すコメントを生成します。このように、Sun WorkShop Visual は既存のスタブファイルを読み出して、新たに生成する必要のあるスタブを判別します。

注 - スタブを再生成する場合を除いて、これらのコメントを変更しないでください。

コールバックスタブの再生成

Sun WorkShop Visual にスタブを再生成させるには、スタブの前にあるコメントとスタブ自体を取り除いてください。どちらか一方だけを取り除いた場合には、以下のような結果になります。

- スタブを取り除き、コメントを残している場合には、スタブは生成されない
- コメントを取り除き、スタブを残している場合には、同じスタブが重複して生成される

スタブを再生成すると、ルーチンの内容が失われることに注意してください。

特殊コメントがコールバックまたはコールバック・メソッドに一致しなくなった場合でも、Sun WorkShop Visual は古いスタブを取り除きません。

ファイル全体の再生成

何らかのコールバックの削除、あるいは名前の変更などにより、古いものが生成され続けてしまい、ファイルが不要コードで一杯になってしまうことがあります。このような場合には、Sun WorkShop Visual にファイル全体を新しく再生成させる必要があります。これを行うには、単純にスタブファイルおよび接尾辞 `.bak` の付いたファイルを取り除くか、その名前を変更します。Sun WorkShop Visual は、スタブファイルに指定した名前を見つけることができない場合は、新しいファイルを生成します。

基本モジュールの分析

基本コードモジュールには、上から下への順序で以下の部分が含まれています。

- ヘッダー (必要に応じて)
- 大域変数宣言
- リンク関数の宣言または関数そのもの (必要に応じて)
- フォントやピックスマップに必要な構造体およびフォントとピックスマップ・オブジェクトを設定するためのコード
- デザイン階層のためのウィジェット作成コード
- *main()* 関数 (必要に応じて)

この節では、ファイル内のコードを分析します。

- **テキストエディタを使用して `icecream.c` を開き、それを読みながら分析します。**

  ファイルのオプション部分は制御パネルにあるトグルを設定することにより組み込んだり、外したりすることができます。これらのトグル、およびオプション部分の組み込みの長所と短所については、254 ページの「コード生成オプション」で説明します。

ヘッダー部分

基本モジュールには、ヘッダー項目が以下に示す順序で含まれています。

- モジュール見出し (存在する場合)
- Sun WorkShop Visual の見出し
- *#include* 文 (必要に応じて)
- モジュール・プレリュード (存在する場合)

ファイルには、モジュール見出しやモジュール・プレリュードはまだ必要ありません。これらの場所に挿入するコードは、 Sun WorkShop Visual の中から指定することができます。コードの指定の手順については、280 ページの「生成されたファイルのカスタマイズ : プレリュード」で説明します。

いくつかの標準 Sun WorkShop Visual コメントの後、モジュール内のコード に必要な *#include* 文のリストがあります。*#include* 文はオプションであり、「コード生成」ダイアログのトグルにより制御されます。245 ページの「基本ソースファイルの設定」を参照してください。

また、Sun WorkShop Visual が使用する基底クラスを定義するために、Sun WorkShop Visual 独自のヘッダーファイルをインクルードする必要があります。インクルードするファイルの名前を変更したい場合や、基底クラスのヘッダーファイルをインクルードしない場合に変更する必要のあるアプリケーションリソースの詳細は、1001 ページの「生成」を参照してください。

リンク関数またはリンク宣言

モジュールにはリンク関数のためのコードが含まれます。以下のコード部分では、代表的なリンク関数を示します。

```
void XDunmanage_link ( Widget w, XtPointer client_data, XtPointer
call_data )
```

このコードの生成はオプションであり、「コード生成」ダイアログによって制御されます。254 ページの「コード生成オプション」を参照してください。

変数宣言

この部分では、デザイン内で大域的に定義されたすべてのウィジェットが宣言されます。以下に代表的な行を示します。

```
Widget exit_button = (Widget) NULL;
```

```
Widget help_cascade = (Widget) NULL;
```

ここでは大域的なウィジェットだけが宣言されます。デフォルトでは、ウィジェットの有効範囲は局所的となっています。局所的ウィジェットは、その親であるシェルを作成する関数において定義されるため、アプリケーションのその他の場所で参照することはできません。ウィジェットを大域的にするためには、次のどちらかの操作を行います。

- コアリソースパネル上でウィジェットを大域的として指定する
- ウィジェットに明示的な名前を指定する

アプリケーションシェルおよび最上位シェルの変数名は、Sun WorkShop Visual では常に大域的であるため、それらを局所的にすることはできません。別の種類のシェルの詳細は、82 ページの「シェルの型」を参照してください。

変数名

変数名は固有のものでなければなりません。変数名が既存のウィジェット名と重複するウィジェットをデザインに「読む」または「ペースト」を行うと、Sun WorkShop Visual は自動的に重複を取り除き、ウィジェットの型に数値を添えた形式 、たとえば *shell4*、*form5* などの形式をとる新しい局所的な名前を割り当てます。

規約によりウィジェットの変数名は小文字で始めます。これにより、Motif の宣言との衝突を避けることができます。

作成関数

デフォルトにより、Sun WorkShop Visual はデザインのシェルウィジェットの作成関数を生成します。作成関数は、生成されるコードの中心部となります。各作成関数では以下の動作が行われます。

- シェルウィジェットそのものを作成する
- シェルおよびその子の子を作成して管理する

- シェルのすべての子に対してハードワイヤされたすべてのリソースを設定する
- コールバックを追加し、(オプションで) コールバックを持つシェルの子にリンクを追加する

作成関数はシェルを表示しません。通常は、*main()* 関数またはコールバック関数にある関数呼び出しによってウィンドウが表示されます。

デフォルトでは、作成関数はシェルの変数名にもとづいて、*create_*<シェル名> の形式をとります。デザインには、*help_window* という名のダイアログシェルと、*myFirstShell* という名のアプリケーションシェルがあります。したがって、作成関数も *create_myFirstShell* と *create_help_window* の 2 つが存在します。作成関数の名前は、コードプレリュード (本章で後述) において変更することができます。

create_help_window は、次の形式をとります。

```
void create_help_window (Widget parent)
{
                    . . .
}
```

関数の本文には、ダイアログシェルそのものを作成する関数呼び出しが含まれています。

```
help_window = XmCreateDialogShell ( parent, "help_window", al, ac );
```

ダイアログシェルはアプリケーションシェルとは異なり、親である別のシェルに依存しています。様々なシェルの型と個々の動作の詳細は、82 ページの「シェルの型」を参照してください。

create_help_window はまた、シェルのすべての子を作成します。ユーザーが明示的変数名を指定した子のダイアログテンプレートが作成され、大域的変数に割り当てられます。

```
dialog_2 = XmCreateMessageBox ( help_window, "dialog_2", al, ac );
```

ユーザーにより明示的に命名されていないラベルは、以下に示すように局所的変数に割り当てられます (コード内の変数名は異なる場合があります)。

```
label1 = XmCreateLabel ( dialog_2, "widget14",al,ac);
```

アプリケーションシェルの作成関数である *create_myFirstShell* は、異なるタイプのシェルであるため引数が異なります。

```
void create_shell_1 (Display *display, char *app_name, int app_argc,
char **app_argv)
{
. . .
}
```

これらの引数についての説明は、284 ページの「作成の前プレリュード」を参照してください。

この関数は create_help_window に似ていますが、メインウィンドウの方がヘルプウィンドウよりも多くの子ウィジェットを持っているため、メインウィンドウの関数の方がより長くなっています。

コールバック関数

基本モジュールにはコールバック関数そのものは含まれていません。そのかわり、基本モジュールは各ウィジェットのコールバック・リストに、指定されたコールバックを追加します。*create_myFirstShell* には、次に示す行が含まれます。これらの行は *exit_button* を作成して *quit* コールバックを追加するためのものです (これらの行は物理的に離れていても構いません)。

```
exit_button = XmCreatePushButton ( rowcoll, "exit_button", al, ac );
. . .
XtAddCallback (exit_button, XmNactivateCallback, quit,NULL);
```

quit() の外部宣言は、ソースファイルの早い時点で生成されます。

ウィジェット *help_button* にある「表示」リンクは、Sun WorkShop Visual 関数 *XDmanage_link* に活性化コールバックを挿入します。*help_button* を作成してリンクを追加するコードは、*exit_button* を作成したそのコールバックを追加するコードに似ています。

```
help_button = XmCreatePushButton ( rowcol2, "help_button", al, ac );
. . .
XtAddCallback (help_button,XmNactivateCallback, XDmanage_link,
(XtPointer) &xd_links[0] );
```

main() 関数プログラム

生成ダイアログにおいて「メインプログラム」トグルが設定されている場合には、別のファイル、または基本モジュールの終わりに最小の *main()* 関数が生成されます。*main()* 手続きを別ファイルに生成する方法については 250 ページの「メインプログラムファイルの設定」を参照してください。

Sun WorkShop Visual の main() 関数では、以下の動作が行われます。

- X サーバーへの接続を開く
- X ツールキットを初期化する
- 最初のアプリケーションシェルのための作成関数を呼び出す
- デザイン内のその他のシェルのための作成関数を呼び出す
- *XtRealizeWidget()* を呼び出して最初のアプリケーションシェルを表示する
- *XtAppMainLoop()* (戻ることがない) を呼び出す
- *exit()* を呼び出す (このコード行は、形式を整えるためだけのもので実行はされない)

いままで確認してきたように、この *main()* 関数によって、インタフェースの実行とその動作の確認をすることができます。多くのアプリケーションの場合、ほとんどの関数はコールバックで処理されるため、*main()* に追加する必要があるコードはほとんどありません。しかし、アプリケーションの他の部分を初期化する必要がある場合には、別の *main()* 手続きのソースファイルをコード生成ダイアログで一度だけ生成し、そのファイルを編集する必要があります。終了用のコードは、アプリケーションを終了させるために呼び出されるコールバック関数に入れるべきです。

- ファイル **`icecream.c`** を閉じます。

リソースファイルの構文

生成されたリソースファイルのデフォルトの構文は、以下のようになります。

<アプリケーション名>*<ウィジェット名>.<リソース>: <値>

識別しやすくするために、ウィジェットの変数名 (ウィジェット名ではない) をリソースリストの前にコメントで示します。しかし、ウィジェットのグループが 1 つのウィジェット名を共有する場合には、グループからの変数名が 1 つだけ表示されます。また、リソースがまったくファイルに生成されていないウィジェットに対してもコメントが生成されます。

構文の例

- X リソースファイル icecream.res をテキストエディタで開き、内容を確認します。

以下に示すファイルの部分には、文字列リソースだけが含まれています。

```
! button1
XDTutorial*button1.labelString: Cone

! button2
XDTutorial*button2.labelString: Dish

! button3
XDTutorial*button3.labelString: Cancel

! procedure_cascade
XDTutorial*procedure_cascade.labelString: Procedures

! button4
XDTutorial*button4.labelString: Wash Dishes...

! button5
XDTutorial*button5.labelString: Count Money

! exit_button
XDTutorial*exit_button.labelString: Exit
XDTutorial*exit_button.accelerator: Ctrl<Key>E
XDTutorial*exit_button.acceleratorText: Control+E
```

```
! help_cascade
XDTutorial*help_cascade.labelString: Help
XDTutorial*help_cascade.mnemonic: H

! help_button
XDTutorial*help_button.labelString: About This Layout
XDTutorial*help_button.mnemonic: A
```

エンドユーザーは、X リソースファイル内で値を編集することによって、これらの文字列を変更することができます。たとえば、2 行目を次のように変更することができます。

```
XDTutorial*procedure_cascade.labelString: Closing Up
```

また、ユーザーのホームディレクトリにある .Xdefaults ファイルを使用して、このファイルの値を書き換えることもできます。

リソース結合へのインクルード

デザインのウィジェットに対して「リソースの結合にインクルード」トグル (コアリソースパネルの「コード生成」ページ) を設定している場合は、リソースファイルの様子が多少異なります。Sun WorkShop Visual がデフォルトでリソースファイル用に生成する構文は非常に汎用的で、アプリケーション全体で指定したウィジェット名を持つすべてのウィジェットに適用します。同じ名前を持つウィジェットが複数ある場合がよくあります。密な結合を使用すると、ウィジェットリソースに関してより幅広く制御することができます。

注 - ここでは、生成されたリソースファイルの構文だけを説明します。密な結合の詳細は、104 ページの「密な結合」を参照してください。

---

104 ページの「密な結合」の例を使用すると、次のような行が生成されます。

```
XDTutorial*FirstForm*OkButton.labelString: Ok
```

緩い結合

緩い結合を設定している場合は、生成されたリソースファイルの最上部に表示されます。緩い結合は個々のウィジェットを参照しないため、その構文は多少異なります。

注 - ここでは、リソースファイルの構文だけを説明します。緩い結合の詳細は、97 ページの「緩い結合」を参照してください。

---

97 ページの「緩い結合」の例を使用すると、リソースファイルに次のような行が生成されます。

```
XDTutorial*XmDialogShell*MyForm.MyButton.labelString: Bound
```

共有リソース値

ウィジェットを完全に識別するためには、X はウィジェット自身の名前に加え、階層内のすべての祖先を示すリストを必要とします。生成された X リソースファイルでは、Sun WorkShop Visual は特定の祖先のリストの代わりに、ワイルドカード (*) を使用します。このため、各ウィジェットはアプリケーション名とウィジェット名だけで区別され、ウィジェット名を共有するウィジェットは X リソースファイルに生成されたリソースも共有します。

次に示す行は、Sun WorkShop Visual 自身の X リソースファイルから取り出したものです。

```
/* ダイアログのボタン */

visu*apply_button.labelString: 適用

visu*cancel_button.labelString: 閉じる
```

Sun WorkShop Visual には、*apply_button* というウィジェット名を持つ複数のボタンがあります。これらのボタンはすべてラベル文字列「適用」を共有しています。同様に、 *cancel_button* という名前のすべてのボタンはラベル文字列「閉じる」を共有しています。これらの文字列は、X リソースファイルにある 1 行を編集することにより、すべてのボタンについて一度に変更することができます。

また、ソースファイルに生成したリソースは、ウィジェットが名前を共有している場合でも、各ウィジェットごとに常に設定されます。

ファイルの整理

Sun WorkShop Visual では、ユーザーの好みに合わせて柔軟にファイルを並べておくことができます。そのため、必要なコードがすべて重複することなく含まれ、アプリケーションへのリンクが正しく行われるよう、ユーザー自身が配慮する必要があります。

また、コードを生成した後でも、 Sun WorkShop Visual で作成したインタフェースを変更できるようなファイルの構成にしておく必要があります。どのように変更してもコードおよび (おそらく) リソースファイルを再生成する必要があります。それまでに行なったプログラム作業を上書きしないように、ファイルとディレクトリを設定してください。

これらの点に考慮しながら、本節ではコードファイルを組織化する方法を説明します。

別ディレクトリの使用

それぞれの Sun WorkShop Visual アプリケーションに対して個別のディレクトリを使用するようにしてください。デザインを開始する前にディレクトリを作成します。デザインファイルの保存およびすべてのコードファイルとリソースファイルの生成は、そのディレクトリで行います。

生成されたファイルを変更しない

エラーを防ぐために、ユーザー独自のプログラムはすべて、Sun WorkShop Visual で生成された基本モジュールと X リソースファイルの外で行います。コードプレリュード、モジュール・プレリュード (280 ページの「生成されたファイルのカスタマイズ : プレリュード」参照) および生成ダイアログにあるさまざまなスイッチを使用すると、Sun WorkShop Visual 内から基本モジュールを制御することができます。同様にリソースプレリュードを使用すると、X リソースファイルを調整することができます。これらのファイルを Sun WorkShop Visual の外で編集しない場合は、デザインを変更する際に、今まで行なった作業を犠牲にすることなくファイルの再生成を行うことができます。

main() を別ファイルに生成

生成された *main()* 手続きは、ほとんどの場合編集しなければなりません。そこで、*main()* を別のファイルに生成することをお勧めします。*main()* のファイル (メインプログラムファイルと呼びます) は自由に編集することができます。いったんメインプログラムファイルに変更を加えたら、そのファイルは再生成しないように注意してください。*main()* プログラムを別のファイルにする方法については、250 ページの「メインプログラムファイルの設定」を参照してください。

スタブファイル

生成された他のファイルとは異なり、スタブファイルは編集されることを前提としています。再度生成された場合でも、スタブファイルに加えた変更は保持されます。コールバックの追加、編集、および内容理解については、第 6 章「インタフェースの起動 : ユーザーコードの追加」を参照してください。スタブファイルの編集については 264 ページの「コールバック関数の追加」を参照してください。

リンクを置く場所

アプリケーションがリンクを使用する場合には、リンク関数とリンク関数宣言をコードファイルかスタブファイルに生成する必要があります。

アプリケーションが複数のデザインファイルから生成されたコードを使用する場合は、すべての基本モジュールにリンク関数宣言を生成し、1 つのファイルにのみリンク関数を生成します。

インクルードファイルを置く場所

構築手順で基本モジュールとアプリケーションコードのコンパイルを別々に行う場合には、基本ソースコードモジュール用の「ヘッダーファイルをインクルード」トグルをオンにする必要があります。これまでの学習では、この手順を使用しました。詳細は 245 ページの「基本ソースファイルの設定」を参照してください。

#include 指令を作成してアプリケーションのコードファイルに生成されたコードを組み込み、すべてのコードを一緒にコンパイルするという方法もあります。この場合は、基本モジュールに対してのみ「ヘッダーファイルをインクルード」をオンにして、スタブファイルを生成する際にオフにします。

生成されたファイルのカスタマイズ : プレリュード

Sun WorkShop Visual を使用すると、Sun WorkShop Visual の生成した基本ソースファイルまたは X リソースファイルを編集してユーザー自身のコード (ここではプレリュードと呼ぶ) を追加することができます。Sun WorkShop Visual にプレリュードを入力すると、Sun WorkShop Visual がそのプレリュードを認識し、コードの中の適切な場所に挿入します。プレリュードには、コードを挿入する場所によって区別される、いくつかの種類があります。

モジュール・プレリュード

「モジュール」メニューにある「モジュール・プレリュード」コマンドを使用すると、図 7-8 に示すダイアログを使用して、ユーザー自身が作成したコードに見出しプレリュード、モジュール・プレリュード、およびリソース・プレリュードを入力することができます。

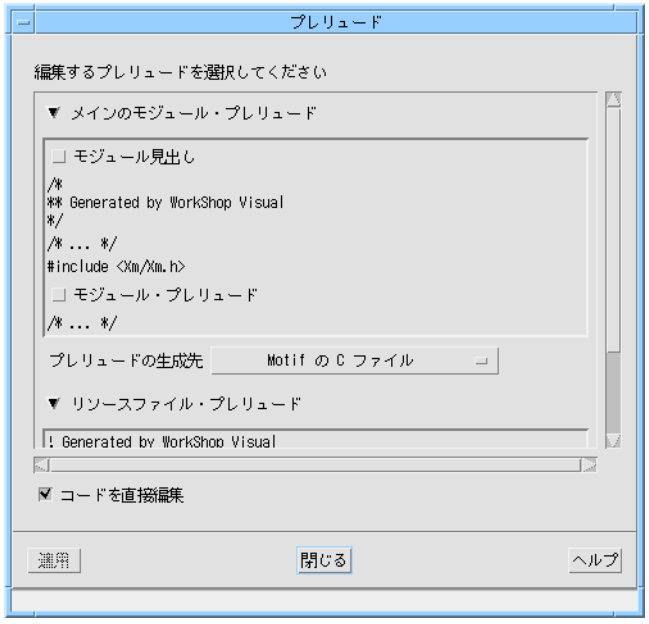


図 7-8　「モジュール・プレリュード」ダイアログ

学習例のテキストに表示されるトグルの 1 つを選択すると、その種類のプレリュードを編集することができます。コードプレリュードは次の 2 つの方法で追加することができます。

- 生成したコードの編集
- ダイアログへのコードの入力

「コードを直接編集」トグルを使用して、上記の方法のうち希望する方法を指定します。「コードを直接編集」トグルをオンにした場合は、コードを追加できるように、生成されたコードのファイルが開かれます。詳細については、287 ページの「編集機能の使用」を参照してください。

「コードを直接編集」トグルをオフにしている場合は、ダイアログの右側に大きなテキストウィジェットが表示されます。ここに、コードを入力します。テキストエディタを使用して別のコードを入力するときと同じように、正しくコードを入力してください。すなわち、行末マーカー、括弧規約などを含め、対象となる言語の規則および規約に準拠する必要があります。プレリュードの最終行の後は、必ず Return キーを押してください。

見出しプレリュード

見出しプレリュードは、メインプログラムファイル、コードファイル、外部宣言ファイル、スタブファイルの先頭に挿入されます。通常は、モジュール見出しにはプログラム名、SCCS ID またはバージョン番号などの情報のコメントが含まれます。

モジュール・プレリュード

モジュール・プレリュードは、生成されたコードファイルの先頭で Sun WorkShop Visual の生成した *#include* 指令 (存在する場合) の直後に挿入されます。モジュール・プレリュードは、コードに必要な *#define* または *#include* 指令あるいは外部宣言を設定することができます。モジュール・プレリュードは、基本モジュールにのみ生成され、スタブファイルには生成されません。

リソース・プレリュード

リソース・プレリュードは X リソースファイルの先頭に挿入され、以下の構文でアプリケーションのリソース設定、つまり、個々のウィジェットではなくアプリケーション全体に対する設定を指定します。

<アプリケーション名>*<リソース>: <値>

たとえば、次のように指定します。

```
visu*symbolFont: -*-symbol-medium-r-normal--14*
```

このようなリソース設定は、アプリケーション内のすべてのウィジェットに適用されますが、個々のウィジェットまたは共通の名前を持つウィジェットのグループでの詳細なリソース設定により書き換えられます。

リソースプレリュードにコメント行や SCCS または RCS キーワードを指定することもできます。

モジュール全体および個々のウィジェットに対する緩い結合の設定方法については、97 ページの「緩い結合」を参照してください。

コードプレリュード

モジュール・プレリュードがモジュール全体に適用されるのに対し、コードプレリュードは個々のウィジェットに割り当てられます。つまり、コードプレリュードが挿入された後に、ウィジェットが作成されてマネージされます。コードプレリュードを追加するには次のようにします。

1. コードに追加するウィジェットを選択します。
2. 「ウィジェット」メニューから「コードプレリュード」を選択します。

   図 7-9 に示す「コードプレリュード」ダイアログが表示されます。

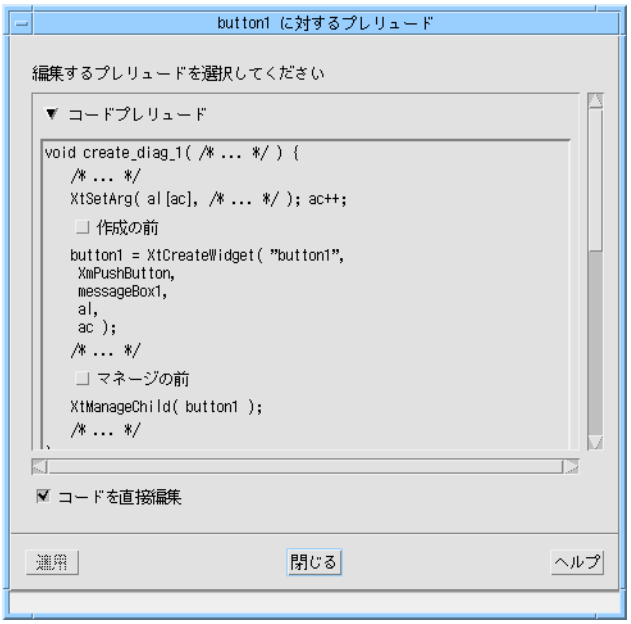


図 7-9　コードプレリュードダイアログ

「コードプレリュード」ダイアログ

「コードプレリュード」ダイアログには、選択したウィジェットに対して生成されたコードを表わすテキストが表示されます。このコードは見本用で、プレリュードが追加される位置を確認することができます。これは、実際の生成されたコードではありません。

このダイアログには、C コード (「コードプレリュード」ラベル) 用と、C++ (「メソッドプレリュード」ラベル) 用の 2 つの領域があります。テキストの両方の領域内にあるトグルを使用して、各種プレリュードを編集するかどうかを選択することができます。C で使用できるコードプレリュードには、「作成の前」と「マネージの前」の 2 種類があります。詳細は、284 ページの「作成の前プレリュード」および 286 ページの「「マネージの前」プレリュード」を参照してください。

また、C++ で使用されるメソッドプレリュードには「公開」、「限定公開」、「非公開」の 3 種類があります。コールバック・メソッドのアクセス制御の詳細は、302 ページの「メソッドのアクセス制御」を参照してください。ただし、プレリュードではメ

ソッドとデータメンバーを両方とも追加することができます。学習の例の中で、コードプレリュード・ダイアログを使用してデータメンバーを追加する方法を、338 ページの「クラスメンバーをプレリュードとして追加」で説明します。

「コードプレリュード」または「メソッド・プレリュード」をクリックすると、対応するテキスト領域がフォールドされ、別の種類のプレリュードだけが表示されます。

コードプレリュードの追加

見本のテキストに表示されるトグルの 1 つを選択すると、その種類のプレリュードを編集することができます。コードプレリュードは、次の 2 つの方法で追加することができます。

- 生成されたコードの編集
- ダイアログへのコードの入力

「コードを直接編集」トグルを使用して、上記の方法のうち希望する方法を指定します。「コードを直接編集」トグルをオンにしている場合は、コードを追加できるように生成コードが開かれます。詳細については、287 ページの「編集機能の使用」を参照してください。

「コードを直接編集」トグルをオフにしている場合は、ダイアログの右側に大きなテキストウィジェットが表示されます。ここに、コードを入力します。テキストエディタを使用して別のコードを入力するときと同じように、正しくコードを入力してください。行末マーカー、括弧規約などを含め、対象とする言語の規則および規約に準拠する必要があります。

作成の前プレリュード

「作成の前」プレリュードは、選択されたウィジェットが作成される前に、コードに挿入されます。

選択されたウィジェットがシェルウィジェットではない場合は、「作成の前」プレリュードはウィジェットの親であるシェルの作成関数に挿入され、制限されることなく任意のコードを提供することができます。「作成の前」プレリュードは一般的に、ウィジェット作成時にのみ設定することができるリソースを設定する場合に使用されます。シェル用の「作成の前」プレリュードは異なっており、285 ページの「シェルの作成の前プレリュード」で説明します。以下に示す生成されたコードの断片は、「作成の前」プレリュードが追加される場所を示します。

```
                    ...
/* visu: prelude for rowcol1: pre-create >>> */
                    ここに「作成の前」プレリュードを入力
/* <<< pre-create ends. */
rowcol1 = XmCreateRowColumn ( shell11, "rowcol1", al, ac );
                    ...
```

「コードを直接編集」を選択して、かつ生成コードを直接編集している場合は、コードを必ず特別なコメントに囲まれた領域に入力してください。こうしておくと、コードの再生成時にこのコードが保持されます。特別なコメントは変更および削除しないでください。

シェルの作成の前プレリュード

シェル用のコードプレリュードは、他のウィジェットのコードプレリュードとは多少異なります。「作成の前」プレリュードは、シェルの作成関数の関数ヘッダーを置き換えるために使用されます。そして必要に応じて、追加引数を定義することができます。

デフォルトの関数ヘッダーでは、生成された関数の本体は引数として渡される変数の 1 つまたは複数を参照します。これらの変数はスコープ内に存在している必要がありますが、引数として引き渡すか、または大域変数として宣言するかを選択することができます。以下の変数もスコープ内に存在している必要があります。

アプリケーションシェルに必要な変数

```
Display *display;
char *app_name;
int app_argc;
char **app_argv;
```

ダイアログシェルまたは最上位シェルに必要な変数

```
Widget parent;
UIL に対する C に必要な変数
MrmHierarchy hierarchy_id;
MrmCode *class;
```

シェルウィジェットに生成前プレリュードを指定しないと、作成関数名はデフォルトで *create_*<ウィジェット変数名> に設定され、引数は必須引数だけになります。

注 - シェルに作成の前プレリュードを指定すると、生成されたデフォルトの main( ) 関数プログラムの作成関数の呼び出しが不正になる場合があります。

注意 - シェルの「作成の前」プレリュードは Motif XP および MFC コードでは直接編集することができません。コードプレリュードにはプラットフォーム独自のコードが含まれているため、通常クロスプラットフォームのデザインにはコードプレリュードを使用することはお勧めしません。

「マネージの前」プレリュード

「マネージの前」プレリュードは、「作成の前」プレリュードの少し後方の、ウィジェットコールバックが追加される直前に挿入されます。このプレリュードは、コールバックに対してクライアントデータを設定する場合などに使用されます。その他には、テキストウィジェットの値の設定、スクロールリストウィジェットへの項目の挿入、あるいはその他の動的初期化などに使用されます。以下の例は、マネージの前プレリュードが挿入される個所を示す、生成されたコードの断片です。

```
                    ...
/* visu: prelude for shell1: pre-manage >>> */
                ここに「マネージの前」コードを入力
/* <<< pre-manage ends. */
XtSetArg(al[ac], XmNallowShellResize, TRUE); ac++;
XtSetArg(al[ac], XmNargc, app_argc); ac++;
XtSetArg(al[ac], XmNargv, app_argv); ac++;
                    ...
```

「コードを直接編集」を選択して、かつ生成されたコードを直接編集している場合は、コードを必ず特別なコメントに囲まれた領域に入力してください。こうしておくと、コードの再生成時にこのコードが保持されます。特別なコメントは変更および削除しないでください。

シェルの「マネージの前」プレリュード

シェルの「マネージの前」プレリュードは、関数中の局所的宣言の直後に挿入されます。それ以外の点では、他のウィジェットに対する場合と同様です。

編集機能の使用

生成されたコードファイルを使用するとき、Sun WorkShop Visual は Sun WorkShop 編集サーバーを使用します。選択した種類のプレリュードのファイルの適切な場所で、別の編集ウィンドウにファイルが開きます。他の種類のプレリュードを選択すると、ユーザーが常に正確な場所に入力できるように、挿入ポイントがファイル内を移動します。「編集サーバー」ウィンドウに保存していない変更内容がある場合にコードファイルの再生成を試みると、まず変更内容を保存するように促されます。

「プレリュード選択」ダイアログ

プレリュードを追加した後でコードファイルを再生成すると、図 7-10 に示す「プレリュード選択」ダイアログが表示されます。「プレリュード選択」ダイアログに直接コードを追加した場合は、「プレリュード選択」ダイアログは表示されません。

図 7-10 「プレリュード選択」ダイアログ

このダイアログで、ファイルが最後に生成されてから変更を加えたプレリュードを個別に選択したり、それらのプレリュードを受け入れるか拒否するかを指定することができます。プレリュードを選択し、矢印キーを押してそれらのプレリュードをあるリストから別のリストに移動します。「了解」ボタンを押すと、受け入れるように設定したファイルだけが再度生成されます。あるいは、新たに追加されたファイルをすべて受け入れるかすべて拒否するかを選択することもできます。「コードプレリュード選択」ダイアログで「了解」ボタンを押すと、それまでに拒否したプレリュードは削除され、復帰することはできません。

以前に行なったコード生成の前に追加されたプレリュードは、保持されています。「プレリュード選択」ダイアログで制御できるプレリュードは、コードファイルが最後に生成された後で追加されたものだけです。

注 - 「コードを直接編集」トグルをオフにして、最後にコードファイルを生成してからプレリュードを追加している場合は、「プレリュード選択」ダイアログが表示されます。これは、新規プレリュードは必ず「プレリュード」ダイアログに表示される必要があるためです。

第8章

構造化コード生成および再使用可能な定義

はじめに

この章では、Sun WorkShop Visual の構造化コード生成機能を使用して、再使用可能なウィジェット階層を作成する方法を説明します。この機能は、再使用可能なウィジェット階層を作成する上で、非常に便利です。これらの再使用可能階層は定義と呼ばれ、ウィジェットパレット上に存在します。また、他のウィジェットと同様、階層に追加することもできます。定義の詳細についても、本章で記述します。

構造化コード生成

Sun WorkShop Visual は、ユーザーの生成コードを構造化し、柔軟性を高め、簡単に再使用ができるようにする制御機能を備えています。この項を読む前に、269 ページの「基本モジュールの分析」 で説明されているデフォルト生成コードの構造を再度読み返し、理解しておくことをお勧めします。特にデフォルトのコードでは、デザイン中の各シェルごとに別々の作成関数があることに注意してください。ウィジェットに名前が付けられていない場合、そのウィジェットは局所的に宣言されます。また、名前が付けられている、あるいはウィジェットがアプリケーションシェルである場合は大域的に宣言されます。

構造化コード制御を使用すると、以下のことが実行できます。

- 階層内の任意のウィジェットに、その子孫を含むウィジェットを返す、独自の作成関数を持たせる

- 任意のウィジェットに、ウィジェットおよび名前の付けられている子孫を含む構造体を返す、独自の作成関数を持たせる
- 任意のウィジェットを、子孫ウィジェットをメンバーに持つ C++ クラスとして定義する
- 子ウィジェットの集合を保存するだけの容器的役割を持つウィジェットを作る
- ウィジェットを明示的に大域的、局所的、または静的として指定する

コードを構造化するための Sun WorkShop Visual の制御機能は、コアリソースパネルの「コード生成」のページにあります。

関数構造体

構造化コード生成の最も簡単な例は、ウィジェットを関数構造体として指定するものです。これにより、Sun WorkShop Visual はウィジェットとその子孫を作成する独立した関数を生成します。この関数は、格納ウィジェットの作成関数によって呼び出されます。

ウィジェットを関数構造体として指定するには、コアリソースパネルの「コード生成」ページを選択し、「構造」オプションメニューから「関数」を選択します。

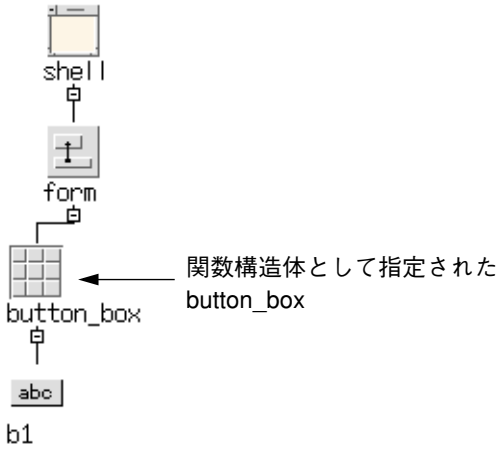


図 8-1　構造の例

図 8-1 に示されている階層は、わかりやすくするために多少単純化されていますが、次のコードを生成します。[1]

```
Widget shell = (widget) NULL;
Widget form = (Widget) NULL;
Widget button_box = (Widget) NULL;
Widget b1 = (Widget) NULL;
/* これは button_box の作成関数です */
Widget create_button_box (Widget parent)
{
                Widget children[1];      /* マネージする子 */
                Arg al [64];               /* 引数リスト */
                register int ac = 0;       /* 引数の個数 */
                Widget button_box = (Widget) NULL;

                button_box = XmCreateRowColumn ( parent,
                    "button_box", al, ac );
                b1 =  XmCreatePushButton ( button_box, "b1", al,
ac);
                children[ac++] = b1;
                XtManageChildren(children, ac);
/* ボタンボックスが作成され、マネージされずに返されます */
                return button_box;
}
/* シェルの作成関数がボタンボックス作成関数を呼び出します */
void create_shell (Widget parent)
{
                Widget children[1];      /* マネージする子 */
                Arg al[64];                /* 引数リスト */
                register int ac = 0;      /* 引数の個数 */
```

1. コメント行は生成されません。

```
                XtSetArg (al[ac], XmNallowShellResize, TRUE); ac++;
                shell = XmCreateDialogShell ( parent, "shell", al,
ac );
                ac = 0;
                XtSetArg (al[ac], XmNautoUnmanage, FALSE); ac++;
                form = XmCreateForm ( shell, "form", al, ac );
                ac = 0;
                button_box = create_button_box ( form );
/* ボタンボックスのコンストレイント・リソースが親の作成関数内に設定されます */
                XtSetArg(al[ac], XmNtopAttachment, XmATTACH_FORM);
                ac++;
                XtSetArg(al[ac], XmNleftAttachment, XmATTACH_FORM);
                ac++;
                XtSetValues ( button_box,al, ac );
/* ボタンボックスは、この時点でマネージされます */
                children[ac++] = button_box;
                XtManageChildren(children, ac);
}
```

ここで、モジュールは、階層全体を作成するための *create_Shell()*、およびボタンボックスを作成するための *create_button_box()* という 2 個の関数を持っていることになります。

データ構造体

次のタイプのコード構造は、データ構造体です。これは、Sun WorkShop Visual がウィジェットとその子孫に対して独立した作成関数を生成する点で、関数構造体に似ています。ウィジェットがデータ構造体として指定された場合、Sun WorkShop Visual はウィジェットおよびその子孫を含む構造体に対して *typedef* を生成します。

ウィジェットの作成によって、その型の構造体が作成、設定され、その構造体へのポインタが返されます。削除関数 (*delete_*<ウィジェット名>) も生成されるため、割り当てられたメモリーを解放することができます。

ウィジェットをデータ構造体として指定するには、コアリソースパネルから「コード生成」ページを選択し、「構造」オプションメニューから「データ構造体」を選択します。

前に示した階層使用 (但し、*button_box* をデータ構造として指定) すると、以下のコードが生成されます。

```
/* 最初に、データ構造体に対しての型宣言が生成されます */[1]
typedef struct button_box_s {
                Widget button_box;
                Widget b1;
} button_box_t, *button_box_p;
Widget shell = (Widget) NULL;
Widget form = (Widget) NULL;
button_box_p button_box = (button_box_p) NULL;
/* 作成関数によって button_box 構造体へのポインタが返されます */
button_box_p create_button_box (Widget parent)
{
                Widget children[1];      /* マネージする子 */
                button_box_p button_box = (button_box_p)NULL;
/* 構造体にスペースが割り当てられ、フィールドが埋められます */
                button_box = (button_box_p) XtMalloc ( sizeof (
                    button_box_t ) );
                button_box->button_box = XmCreateRowColumn (
parent,
                    "button_box", al, ac );
                button_box->b1 = XmCreatePushButton
                    ( button_box->button_box, "b1", al, ac );
                children[ac++] = button_box->b1;
```

1. コメント行は生成されません。

```
                XtManageChildren(children, ac);
                return button_box;
}
/* 割り当てられたメモリーを解放するための削除関数が提供されます */
void delete_button_box (button_box_p button_box)
{
                if ( ! button_box )
                    return;
                XtFree ( ( char * )button_box );
}
/* 再度シェル作成関数がボタンボックス作成関数を呼び出します */
void create_shell (Widget parent)
{
                Widget children[1]; /* マネージする子 */
                Arg al[64]; /* 引数リスト */
                register int ac = 0; /* 引数の個数 */
                shell = XmCreateDialogShell ( parent, "shell", al,
ac );
                form = XmCreateForm ( shell, "form", al, ac );
                button_box = create_button_box ( form );
                XtSetArg(al[ac], XmNtopAttachment, XmATTACH_FORM);
                ac++;
                XtSetArg(al[ac], XmNleftAttachment, XmATTACH_FORM);
                ac++;
/* button_box ウィジェットは、構造体内部で参照されます */
                XtSetValues ( button_box->button_box,al, ac );
                ac = 0;
                children[ac++] = button_box->button_box;
                XtManageChildren(children, ac);
                ac = 0;
```

}

C++ クラス

C++ クラスの使用方法は、データ構造の場合と非常によく似ています。
Sun WorkShop Visual は階層内の個々のウィジェットを *C++* のクラスを使用して隠ぺいすることはしませんが、そのかわり階層の一部分を独自のクラスとして指定します。C++ のクラスとして指定されたウィジェットには対応したクラスが定義されます。そのウィジェットの名前の付いた子孫ウィジェットは、クラスのメンバーとなり、ウィジェット作成およびウィジェット破壊のメソッドが供給されます。また、クラスにそれ自体がクラス (のポインタ) であるメンバーが含まれる場合には、これらのメンバーを作成および破壊するためのコンストラクタ、およびデストラクタ・メソッドが生成されます。ウィジェットはクラスのインスタンス作成時に作成されるのではなく、ウィジェット作成関数が明示的に呼び出された場合に作成されることに注意してください。したがって、クラスのインスタンスを破壊することによって、ウィジェットが破壊されることはありません。

ウィジェットを C++ クラスとして指定するには、コアリソースパネルの「コード生成」ページを選択し、「構造」オプションメニューから「C++/Java クラス」を選択します。ウィジェットを C++ クラスとして指定し、C を生成する場合には、ウィジェットはデータ構造体として扱われるので、注意してください。

この節では C++ クラスについて説明します。Sun WorkShop Visual の Java クラスについての詳細は、363 ページの第 10 章「Java 用のデザイン」を参照してください。

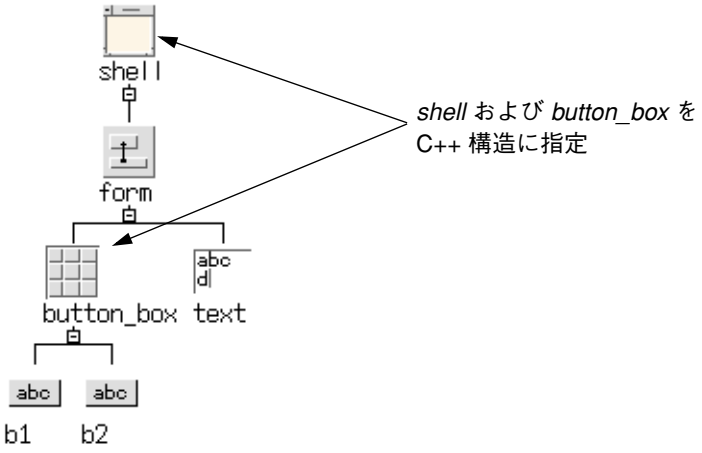


図 8-2　C++ クラス構造の例

この例で生成されるコードを以下に示します。わかりやすくするために、簡略化してあります。[1]

button_box および *shell* に対してクラスが宣言されます。

```
class button_box_c: public xd_XmRowColumn_c {

public:

                  virtual void create (Widget parent, char
*widget_name =

                      NULL);

protected:

                  Widget button_box;

                  Widget b1;

                  Widget b2;

};


typedef button_box_c *button_box_p;
```

shell クラスはコンストラクタとデスクトラクタ関数を持ちます。これは、クラス (またはデータ構造体) へのポインタであるメンバーを持っているためです。

```
class shell_c: public xd_XmDialog_c {

public:
```

1. コメント行は生成されません。

```
                virtual void create (Widget parent, char
*widget_name = NULL);
                shell_c();
                virtual ~shell_c();
protected:
                Widget shell;
                Widget form;
                Widget text;
                button_box_p button_box;
};

typedef shell_c *shell_p;

shell_p shell = (shell_p) NULL;
```

この時点で、作成関数はクラスのメソッドとなります。このメソッドは、本リリースで提供される Sun WorkShop Visual の基底クラスにおいて公開宣言されます。

```
void button_box_c::create (Widget parent, char *widget_name)
{
                Widget children[2];      /* マネージする子 */
                Arg al[64];               /* 引数リスト */
                register int ac = 0;      /* 引数の個数 */

                if ( !widget_name )
                    widget_name = "button_box";

                button_box = XmCreateRowColumn ( parent,
widget_name,
                    al, ac );
```

_xd_rootwidget は部分階層のルートにあるウィジェットを保存するクラスの限定公開メンバーです。これにより、基底クラスはウィジェットを操作することができます。

```
                _xd_rootwidget = button_box;
```

```
                b1 = XmCreatePushButton ( button_box, "b1", al, ac
);
                b2 = XmCreatePushButton ( button_box, "b2", al, ac
);
                children[ac++] = b1;
                children[ac++] = b2;
                XtManageChildren(children, ac);
                ac = 0;
}
```

シェルの作成メソッドは、ボタンボックスの作成メソッドを呼び出します。

```
void shell_c::create (Widget parent, char *widget_name)
{
                Widget children[2];      /* マネージする子 */
                Arg al[64];               /* 引数リスト */
                register int ac = 0;      /* 引数の個数 */

                if ( !widget_name )
                    widget_name = "shell";
                XtSetArg(al[ac], XmNallowShellResize, TRUE); ac++;
                shell = XmCreateDialogShell ( parent, widget_name,
al,
                    ac );
                ac = 0;
                _xd_rootwidget = shell;
                XtSetArg(al[ac], XmNautoUnmanage, FALSE); ac++;
                form = XmCreateForm ( shell, "form", al, ac );
                ac = 0;
                text = XmCreateText ( form, "text", al, ac );
```

ボタンボックスクラスは、コンストラクタ・メソッドでインスタンス生成されるため、この時点ではウィジェットのみが作成される必要があります。

```
                button_box->create ( form, "button_box" );
```

```
                XtSetArg(al[ac], XmNtopAttachment, XmATTACH_FORM);
                ac++;
                XtSetArg(al[ac], XmNtopWidget,
                    button_box->xd_rootwidget( )); ac++;
                XtSetArg(al[ac], XmNleftAttachment, XmATTACH_FORM);
                ac++;
                XtSetValues ( text,al, ac );
                ac = 0;

                XtSetArg(al[ac], XmNtopAttachment,
XmATTACH_WIDGET);
                ac++;
                XtSetArg(al[ac], XmNleftAttachment, XmATTACH_FORM);
                ac++;
                XtSetValues ( button_box->xd_rootwidget(),al, ac );
                ac = 0;
                children[ac++] = text;
                children[ac++] = button_box->xd_rootwidget();
                XtManageChildren(children, ac);
                ac = 0;
}
shell_c::shell_c()
{
```

子クラスのインスタンスを生成します。

```
                button_box = new button_box_c;
}
shell_c::~shell_c()
{
```

子クラスを解放します。

```
                delete button_box;
}
```

ウィジェットが C++ クラスに指定され、C コードが生成される場合、ウィジェットはデータ構造体と同様に扱われます。

デフォルトでは、生成クラスは Sun WorkShop Visual の基底クラスの 1 つから派生しますが、「C++ アクセス」オプションメニューの下のフィールドに基底クラスを指定して、変更することもできます。本リリースで供給される Sun WorkShop Visual 基底クラスは、生成コードを正確に実行するための最小限のサポートを提供しています。これらのクラスを変更および拡張すると、ご使用の GUI 開発手法に適した再使用可能なメソッドを作成することができます。

子孫ウィジェットは名前が付けられている場合、あるいはそれ自体がデータ構造であったり C++ クラスである場合は、クラスの限定公開メンバーとして現れます。したがって、C++ クラスウィジェットそのもの、およびクラスメンバーとして呼び出す必要のある子孫に名前を付けることは重要です。デフォルトのアクセス制御の変更は、「C++ アクセス」オプションメニューから必要なレベル (公開、限定公開、または非公開) を選択することにより実行できます。

C++ クラスウィジェットに対して、名前の付けられていないウィジェットを使用した場合でも、即座にエラーが生じることはありません。しかし、Sun WorkShop Visual が名前のつけられていないウィジェットに対して自動的に割り当てられる数は、階層の編集時に変更される可能性があるため、名前の付けられていないウィジェットを使用することはお勧めできません。

コールバック・メソッド

コールバック関数を呼び出す X ツールキット関数は、以下の形式のコールバック関数をとります。

```
void my_callback (Widget, XtPointer, XtPointer)
```

通常のメンバー関数はコールバック関数には適していません。これは C++ コンパイラが、オブジェクトに対してインスタンスデータを検索する最初の引数 (*this* ポインタ) を X メンバー関数に余分に渡してしまうためです。コールバック関数として通常のメンバー関数を使用すると、メンバー関数はウィジェットポインタをインスタンスデータポインタとして解釈するため、意図したとおりには動作しません。

Sun WorkShop Visual は、この問題を回避するために一般によく使用される技術を使用しています。静的メンバー関数 (*this* ポインタを予定していない) が宣言され、コールバック関数として使用されます。

```
static void my_callback (Widget, Xtpointer call_data, XtPointer
client_data)
```

インスタンスへポインタを引き渡すためには、クライアントデータ (client_data) 引数が使用されます。静的メンバー関数は、単にインスタンスポインタを使用して通常の非静的メンバー関数を呼び出し、そのウィジェットと引数 (call_data) を渡します。非静的メンバー関数は、以下の形式をとります。

```
virtual void my_callback (Widget, XtPointer call_data)
```

Sun WorkShop Visual は、両方の関数宣言、静的コールバック関数に対する全コード、およびユーザーの書いた標準メンバー関数に対するスタブを生成します。この関数は仮想として宣言されているため、動作を変更するために派生クラスにおいて書き換えることができます。この手法については、ダグラス A ヤング著、磯谷正孝訳の『オブジェクト指向プログラミングと C++: OSF/Motif 版』 (Douglas Young 『*Object-Oriented Programming with C++ and OSF/Motif*』) に詳しく記述されています。

コールバック・メソッドの編集

コールバック・メソッドを追加する場合は、まだそのメソッドが宣言されていなければその宣言も追加されます。「コールバック」ダイアログ内の「メソッド」ボタンを押すと、現在選択しているウィジェットを包含するクラスで宣言されているメソッドが表示されます。

Sun WorkShop Visual はデフォルトで、メソッドを純粋仮想ではなく、公開アクセスを持つものとして宣言します。これらの属性を変更したい場合は、「メソッド宣言」ダイアログを使用して変更してください。詳細は、302 ページの「メソッド宣言」を参照してください。

メソッド宣言

コールバックをメソッドとして追加する場合は、効率的に処理するために、Sun WorkShop Visual はメソッドの宣言をそのウィジェットの包含するクラスに追加します。包含するクラスであるウィジェットを選択し、「ウィジェット」メニューから「メソッド宣言」を選択することにより、メソッド宣言の表示、追加および削除を行うことができます。メソッド宣言ダイアログを図 8-3 に示します。

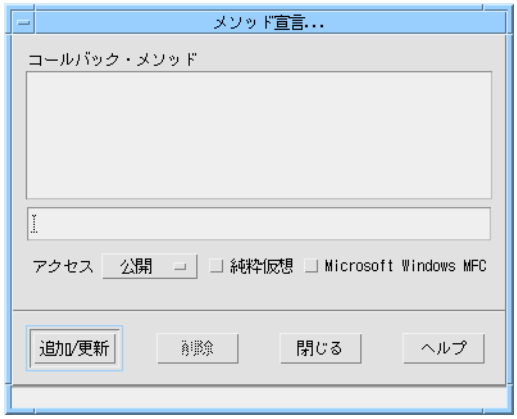


図 8-3　メソッド宣言ダイアログ

53 ページの「構造の色」で説明するように「表示」メニューの「構造の色」を使用し、メソッドを追加したウィジェットに最も近い上位クラスを選択します。このウィジェットが C++ クラスとして定義されている場合は、これは当然同じウィジェットとなります。

メソッドのアクセス制御

デフォルトでは、Sun WorkShop Visual の追加するメソッドのアクセスは「公開」です。「メソッド宣言」ダイアログにある「アクセス」オプションメニューを使用すると、個々のコールバック・メソッドに対してのアクセスを、「公開」、「非公開」、「限定公開」の中からを選択できます。

純粋仮想メソッド

「純粋仮想」トグルを設定して、非静的メンバー関数を純粋仮想関数として宣言することができます。たとえば、このトグルをメニューバークラスのコールバック・メソッド *OnNew()* に設定した場合、Sun WorkShop Visual はメソッドを以下のように宣言します。

```
class menubar_c:  public xd_XmMenubar_c {
...
public:
...
                  virtual void OnNew( Widget, Xtpointer) = 0;
};
```

この関数は純粋仮想関数であるため、*menubar_c* は抽象クラスとなり、*menubar_c::OnNew()* の実装を行う必要はありません。*menubar_c* のインスタンスを作成することはできませんが、他のクラスの基底クラスとして使用することが可能となります。

デフォルトでは、Sun WorkShop Visual の追加するメソッドは純粋仮想ではありません。

コールバック・メソッドの削除

コールバック・メソッドをウィジェットから削除するということは、単にそのメソッド (ウィジェットへのコール) の使用を止めるということを意味します。
Sun WorkShop Visual でコールバック・メソッドを追加すると、メソッドの宣言が自動的に追加されます。メソッドの宣言も一緒に削除したい場合は、包含するクラスであるウィジェットのメソッド宣言リストから宣言を削除する必要があります。詳細な手順および Sun WorkShop Visual を使用して追加された宣言の詳細は、302 ページの「メソッド宣言」を参照してください。

構造体の変更とメソッドの無効化

上記のように、追加されたコールバック・メソッドは包含するクラスで宣言されます。このウィジェット (包含するクラス) の構造体を変更した結果ウィジェットがクラスでなくなると、メソッドは無効になります。このような場合、Sun WorkShop Visual は に示すような「無効化されたメソッド」ダイアログを表示します。

このダイアログはモード付きなので、デザインで作業を続けるにはダイアログを閉じなければなりません。このダイアログが表示されるのは、メソッド宣言が無効になるようにウィジェットの構造体を変更したときだけです。

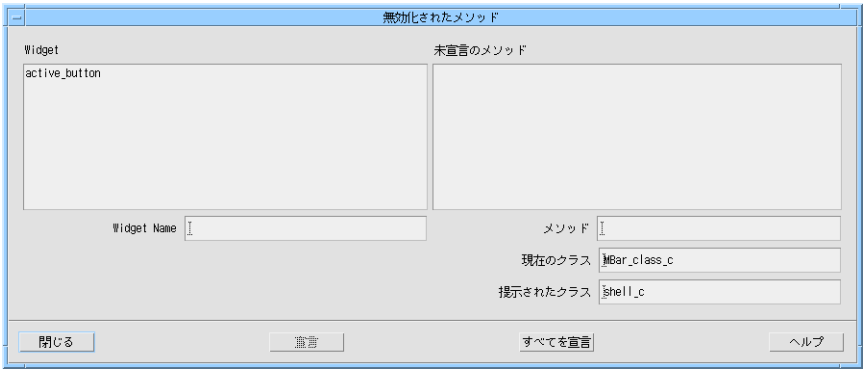


図 8-4　「無効化されたメソッド」ダイアログ

左側のウィジェットリストには、構造体を変更することでメソッドが無効化されるすべてのウィジェットが表示されます。ウィジェットを選択すると、無効化されたメソッドが右側に表示されます。選択した各メソッドについて、現在宣言されているクラスと、メソッドを宣言できる新しいクラスがこのダイアログに表示されます。「提示されたクラス」は常に最上位に最も近い上位クラスです。適当なクラスが他に存在しない場合は、このダイアログはメソッドが関数になることを警告します。

「宣言」を押すと、選択したメソッドの宣言が「提示されたクラス」に変わります。「すべてを宣言」を押すと、無効化された各メソッドがそれぞれの「提示されたクラス」に変わります。

メソッドプレリュード

「コードプレリュード」ダイアログを使用して、C++ クラスにデータや関数メンバーを追加することができます。「公開メソッド」、「限定公開メソッド」または「非公開メソッド」を選択し、テキスト領域 (直接編集している場合はコード) に宣言を入力します。基本モジュールおよび外部宣言ファイルの両方のクラス宣言に C++ コードプレリュードが生成されます。

派生クラスの作成

クラスに関数を追加する場合、Sun WorkShop Visual によって生成されたクラスから派生した新しいクラスを書くことをお勧めします。派生クラスと生成された基底クラス間の論理的な違いを使用して、メンバーを追加して仮想関数を実装することができます。

デフォルトでは、Sun WorkShop Visual はルートウィジェットの変数名をもとに C++ クラス名を決定します。したがって、ウィジェット *menubar* のクラスは *menubar_c* となります。

```
class menubar_c: public xd_XmMenuBar_c {
...
};
```

Sun WorkShop Visual はクラスのインスタンスを作成するコードを生成する場合に、同じ名前を使用します。

```
menubar = new menubar_c;
```

Sun WorkShop Visual がクラスをある名前で生成し、別の名前でインスタンスを作成するようにデフォルト動作を変更することができます。

```
menubar = new mymenubar_c;
```

デフォルト動作の変更を行うには、コアリソースパネルの「コード生成」ページにある「インスタンス名」を使用します。

基底クラスの変更

デフォルトでは、Sun WorkShop Visual はルートウィジェットの型に適した基底クラスから生成クラスを派生させます。たとえば、ウィジェット階層のルートにメニューバーを持つクラスは、*xd_XmMenuBar_c* から派生しています。基底クラスの名前は、コアリソースパネルで変更することができます。

Sun WorkShop Visual 製品には、基底クラスの実装例が含まれています。これらをそのまま、あるいは修正を加えて使用して、特定のアプリケーションの分野に適した機能を追加することができます。

実装例の基底クラスを構築するためのメークファイルも含まれています。Sun WorkShop Visual は基底クラスについて以下の 2 点を仮定します。

1. Widget 型のデータメンバー *_xd_rootwidget* が存在する

2. 付属関数 `xd_rootwidget( )` があり、値 *_xd_rootwidget* を取り出して返す

これらの仮定は、クラス *xd_base_c* 中で使用されています。このクラスには他のいくつかの基本的な制約も存在します。

```
class xd_base_c
{
public:
                xd_base_c() {_xd_rootwidget=NULL;}
                Widget xd_rootwidget() const {return
_xd_rootwidget;}
protected:
                Widget _xd_rootwidget;
private:
                void operator=(xd_base_c&); // 代入なし
                xd_base_c(xd_base_c&);      // デフォルトのコピーなし
};
```

Sun WorkShop Visual は、使用される基底クラスについてその他の制約は行いません。つまり言い換えると、基底クラスのセットは、それらが *xd-base_c* (または別の基底クラスで Sun WorkShop Visual の仮定を満たすもの) から派生したものであれば使用可能です。

基底クラスのコンストラクタに対して、クラス名と一緒に実際の引数を渡すことも可能です。引数が供給される場合 (基底クラス文字列が '()' を含んでいる場合)、そのクラスはコンストラクタを持つように強制され、引数文字列は基底クラスに渡されます。たとえば、ウィジェット menubar に対して「基底クラス」文字列を mymenubar_c ("Hello World") に設定すると、Sun WorkShop Visual は以下のように生成を行います。

```
class menubar_c : public mymenubar_c {
public:
                menubar_c();
```

```
                    ...
};
...
menubar_c::menubar_c () : mymenubar ( "Hello World" )
{
}
...
menubar = new menubar_c;
```

子のみを生成するウィジェット

子のみ生成構造体オプションを選択すると、あるウィジェット (子のみを生成するウィジェット) を別の構造体を保持するための構造体として指定することができます。 子のみを生成するウィジェットは、階層内のそれらの子孫に対してのコンテキストを提供しますが、子のみを生成するウィジェット自身に対してのコードは生成しません。以下に示す例を考察してみましょう。

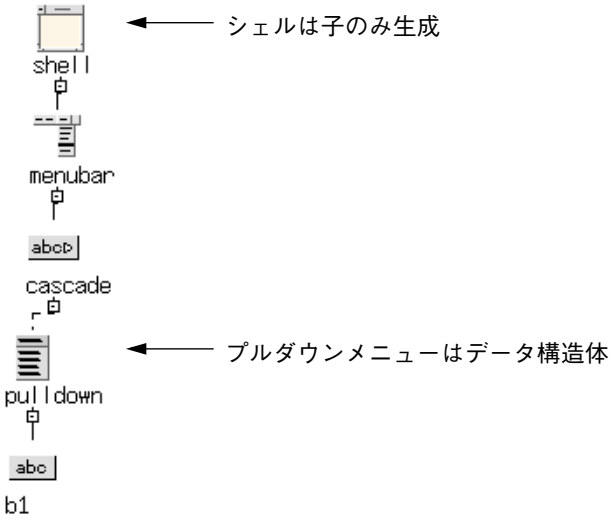


図 8-5　子のみ生成構造体の使用例

図 8-5 に示すデザインからコードを生成する場合、Sun WorkShop Visual はメニュー構造に対してのみコードを作成します。この機能を使用すると、アプリケーションプログラムで制御可能なデザインの一部分だけを生成することができます。

注 - ウィジェットを「子のみ生成」として指定した場合は、構造化されているか、名前の付けられている子ウィジェットに対してのみコードが生成されます。したがって、「子のみ生成」ウィジェットの下の階層のウィジェットが、すべて構造化されていないか名前が付けられていない場合は、コード内にはウィジェット宣言のみが生成されます。

Microsoft Windows モードでの子のみ生成構造体

Microsoft Windows モードでは、シェルの子は C++ クラスにはできません。Microsoft Windows モードでシェルの子だけを C++ で生成するには、シェルの下に「ダミーの」コンテナウィジェット (ローカラムやフォーム) を追加して、そのコンテナウィジェット以下を C++ クラスにします。この方法は、Microsoft Windows モードで定義を作りたい場合にも役立ちます。これは、ウィジェットの集まりを定義するにはルートウィジェットが構造化されていなければならないからです。

構造化コード生成と UIL

何らかの構造を含むデザインに対して UIL を生成する場合の手順は、基本的には C および C++ の場合と同じです。独立した階層が UIL ファイルに生成され、それぞれの作成関数がコードファイルに生成されます。作成関数は UIL 階層から適切なウィジェットを取り出し、適切にデータ構造フィールドへの記入を行います。

宣言範囲の変更

ウィジェットは通常、何らかの方法で構造化されていない場合、あるいは名前が付けられていない場合には、閉じた作成関数内で局所的に宣言されます。ウィジェットが構造化されているか、名前が付けられている場合には、それらはそれが含まれる構造体 (存在する場合) 内で、あるいは大域的変数として宣言されます。このデフォルト動作は、コアリソースパネルにあるウィジェットの記憶クラスを設定することにより、

変更できます。記憶クラスを「局所」に設定すると、本来は大域的として宣言される、または構造体内で宣言されるはずのウィジェットが作成関数に対して局所的に宣言されます。記憶クラスを「大域」に設定すると、名前の付いていないウィジェットまたは構造の中の名前の付いた要素が大域的に宣言されます。「大域」設定は、ウィジェット型リソースおよび 309 ページの「到達不能ウィジェット」で説明されているリンクに対して特に有効です。「静的」オプションは「大域」に似ていますが、宣言はモジュールに対して静的です。

名前の付いていないウィジェットを強制的にデータ構造にすることはできません。名前のないデータ構造ウィジェットの子は、データ構造の作成関数に対して局所的に作成され、管理されます。

到達不能ウィジェット

ブリテンボード用の *XmNdefaultButton* のようなウィジェット型のリソースと一緒に、あるいはリンクと一緒に構造化コード生成を使用する場合、範囲外のウィジェットを参照するデザインを指定してしまうことがあります。これらは、到達不能ウィジェットとみなされます。Sun WorkShop Visual は、このような事例の検出を行い、コード生成時に警告を発します。また、「子のみ生成」構造体を使用した場合や階層を実行時に動的に作成した場合、到達不能ウィジェットが予想外の障害を発生させる可能性もあります。

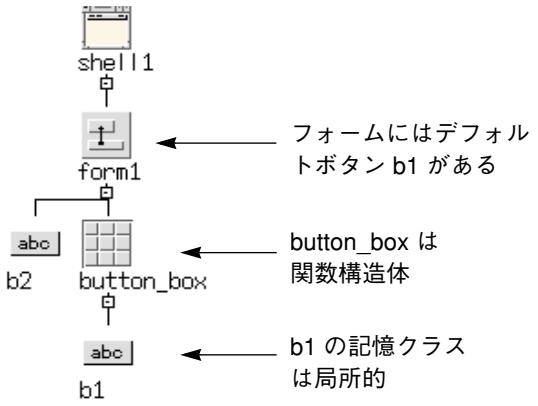


図 8-6　到達不能ウィジェットのある階層

図 8-6 に到達不能ウィジェットを示します。*b1* はフォームの作成関数からアクセス可能でデフォルトボタン引数として使用可能でなければなりません。しかし、*b1* は *button_box* 関数に対して局所的であるため、フォームの作成関数のスコープには入っていません。Sun WorkShop Visual は、この状況を検出すると、以下の警告をコード生成時に表示します。

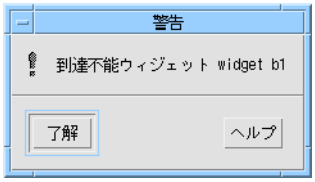


図 8-7　到達不能ウィジェットエラー

コードは生成されますが、予想通りにはコンパイル、あるいは実行しない場合があります。この場合の最も簡単な解決方法は、記憶クラスオプションを使用して適切なウィジェットを強制的に大域的と指定することです。

定義

ウィジェットの階層が構造 (C++ クラスまたは C 構造体) としてカプセル化された場合、それを定義に変化させることにより、他のデザインで再使用することができます。定義とは Sun WorkShop Visual のウィジェットパレットに追加して、再使用できるウィジェット階層です。パレットから定義を選択することにより、デザインに定義のインスタンスが作成されます。このインスタンスをさらに修飾し、再び定義にすることも可能です。

前提条件

ウィジェット階層は、以下の場合に定義となることができます。

1. ルートウィジェットがデフォルトではない変数名を持っている
2. ルートウィジェットが C++ クラスまたは構造体として指定されている
3. ルートウィジェットが別の定義の一部ではない
4. ウィジェット階層が定義を含んでいない

5. ウィジェット階層が大域的または静的ウィジェットを含んでいない

定義の指定

定義を指定する場合、対象となるウィジェットを含むデザインファイルが保存され、かつ、対象となるウィジェットに定義として印が付けられている必要があります。ウィジェットに定義としての印をつけるためには、「ウィジェット」メニューにある「定義」を使用します。定義を作成すると、その中のウィジェットは凍結され、ウィジェットのリソースパネルは使用不可能となるため、ウィジェットの追加やウィジェット名の変更を行うことはできません。一時的に定義の指定を解除して、定義を構成するウィジェットを編集することはできます。ただし、定義を使用するデザインとの矛盾を避けるため、この操作は慎重に行なってください。詳細は、315 ページの「定義の変更」を参照してください。

他のデザインにおいて定義を使用可能にするためには、Sun WorkShop Visual はその定義に対しての外部参照を必要とします。これは、「定義を編集」ダイアログで編集できる「定義」ファイルにより提供されます。

定義の短縮操作

パレットメニューにある「定義する」ボタンを使用すると、簡単に新しい定義を追加することができます。このボタンは、現在選択されているウィジェットを定義として指定し、そのデザインを保存してパレットに定義を追加します。定義に対してのヘッダーファイル名は、コード生成ダイアログにある型宣言ファイル名から取り込まれます。アイコンは使用されません。

定義ファイル

Sun WorkShop Visual は、定義ファイルを読み出して、その内容をパレットに表示します。定義ファイル名は、*definisionFileName* リソースの設定により指定されます。デフォルト値は `$HOME/.xddefinitionsrc` です。

複数のプロジェクトで作業を進める必要があり、それぞれのプロジェクトが異なる定義のセットを使用する場合、リソースを設定して定義ファイルを変更することができます。以下に例を示します。

```
visu.definitionsFileName:/home/project6/xddefs
```

このリソースの値は環境変数を含むことができます。以下に例を示します。

```
visu.definitionsFileName:$PROJECT_ROOT/xddefs
```

新しい設定に変更する場合は、Sun WorkShop Visual を終了して再起動します。

定義ファイルの編集

定義ファイルの変更は、「パレット」メニューにある「定義を編集」ボタンを使用して行います。このボタンは、図 8-8 に示すダイアログを表示します。

図 8-8　パレットへの定義の追加

このダイアログを使用して、新しい定義の追加、削除あるいは既存の定義を編集することができます。定義を追加する場合には、以下を入力します。

- 定義
  定義名
- ウィジェット名
  定義のルートウィジェットの変数名
- 保存ファイル
  保存されているデザインファイルの名前 (.xd)

また、以下の指定を行うことができます。

- アイコンリソース
  定義に対してピックスマップ・ファイルを割り当てるために使用されるリソース名。詳細は、807 ページの「アイコンファイルの指定」を参照してください。
- アイコンファイル
  アイコンリソースを使用しても見つからない場合にウィジェットパレットとして使用される、ビットマップまたは xpm ピックスマップを含んでいるファイル。
- インクルードファイル
  対応する構造体またはクラスを宣言するヘッダーファイルの名前。定義のインスタンスが使用されると、このファイルが生成コードに自動的にインクルードされます。そのため、コンパイラがファイルを見つけることができるかどうかを確認する必要があります。定義から生成された外部宣言ファイルと同じ名前を指定してください。
- リソースファイル
  定義の値を含んでいるリソースファイルの名前。定義のインスタンスが使用されると、このファイルは生成リソースファイルにインクルードされます。定義に対して外部ファイルが生成された場合に、指定された名前に対応します。
- ファミリ
  この定義が属するファミリ、すなわちグループ。これは、ウィジェットパレット上に定義を表示する場合に限って関係があります。定義はまとめられてファミリとなります。ファミリは常に 1 つ表示されています。表示されるファミリを変更するには、ウィジェットパレットの定義の上にあるオプションメニューからファミリを選択します。デフォルトでは、定義は「Default」のファミリに割り当てられています。ファミリには、任意の名前を指定することができます。また、同じファミリにまとめる定義の数には、制限はありません。

- ヘルプ
  ユーザーにヘルプを提供するために使用されるドキュメントとマーカーの組み合わせ。319 ページの「定義のオンラインヘルプ」を参照してください。

- MFC オフセット
  このフィールドは、Sun WorkShop Visual が Microsoft Windows モードである場合にのみ表示されます。Microsoft Windows アプリケーションでは、個々のコントロールを識別するために固有の番号が指定されます。Sun WorkShop Visual は、コントロールごとに固有の番号を生成するため、通常は問題ありません。ただし、すでに多数のコントロールを持っているインスタンスにウィジェットを追加する場合には、番号が重なってしまう場合もあります。MFC オフセットは、インスタンスに追加されるコントロールの ID に加算されます。MFC オフセットの数を増やすことで、コントロールの ID と定義内にあるコントロールとの衝突を避けることができます。

作成時に設定されない属性は、後で設定することができます。たとえば、定義のテストおよびデバッグは、定義のアイコンをデザインする前に実行できます。

「仮設定」を使用して、現在選択されているウィジェットのダイアログ上にあるフィールドのいくつかを自動的に設定することができます。

ベースディレクトリ

定義が相対ファイル名 (/ で始まらない名前) で指定されている場合、Sun WorkShop Visual はファイル名の前にベースディレクトリを追加します。ベースディレクトリが指定されていない場合は、定義デザインファイルを含んでいるディレクトリが使用されます。

ベースディレクトリを指定するには、「定義を編集」ダイアログを表示して「ベースディレクトリ」をクリックし、新しいディレクトリを選択して「了解」をクリックします。すると、新しいベースディレクトリが定義ファイルに保存され、Sun WorkShop Visual の現在のセッションに即座に適用されます。現在のデザインが既存の定義のインスタンスを含んでいる場合、ベースディレクトリを変更することはできません。

定義の変更

定義内にあるウィジェットは凍結されています。したがって、ウィジェットの追加または削除、名前の変更、配置エディタでのコンストレイントの設定、リソースのリセットは、行うことができません。定義を変更するためには、一時的に定義を解除する必要があります。定義を変更する必要がある場合には、以下に示す手順に従って操作を実行してください。

1. 定義を含んでいる保存ファイルを開きます。
2. 定義のルートウィジェットを選択します。
3. ウィジェットメニューをプルダウンし、「定義」トグルをオフにします。

   トグルをオフにすると、定義にあるウィジェットの凍結が解除されるため、必要に応じた変更、修正を行うことができます。
4. 編集が終了したら、ルートウィジェットを選択し、「定義」トグルを再度オンにします。
5. コードファイルと外部宣言ファイルを再生成します。
6. デザインを保存します。

定義の変更による影響

定義を変更すると、その定義を使用するすべてのデザインファイルに影響があります。定義を使用するデザインを開くたびに、Sun WorkShop Visual は定義を含むファイルを開き、これら両方のファイルからの情報を組み合わせます。定義が変更されると、Sun WorkShop Visual は、変更前の定義を使用していたデザインが新しい定義を受け入れるように試みます。

受け入れられない変更がある場合には、Sun WorkShop Visual はエラーメッセージを表示し、デザインの不適合部分を一時的に Sun WorkShop Visual のクリップボード・ファイルに保存します。以下の方法のいずれかで、不適合を解決します。

- クリップボード・ファイルをユーザーのデザインにペーストし、その内容と新しい定義を手動で解決する
- クリップボード・ファイルの内容をすべて破棄する

- 変更を保存することなく Sun WorkShop Visual を終了する、あるいはデザインと互換性がある定義に戻す

不適合の確率を最小限にするには、次のことを守ってください。

- 定義内のウィジェット名を変更しない
- 定義内のウィジェットの置き換えは、同名のサブクラスウィジェットのみで行う たとえば、ラベル「foo」とプッシュボタン「foo」を置き換えても、通常は不適合は発生しません。

インスタンス

定義のインスタンスは、パレットの適切なボタンをクリックするだけで作成できます。インスタンスはカラー背景で表示されます。

定義とインスタンスは、それぞれに個別のデザインとして作成されなければなりません。Sun WorkShop Visual 内では定義とインスタンスは同一のデザインの中に存在していますが、それぞれに対するコードが個別のコードでなければ、コンパイルは行われません。

定義ファミリ

定義はそのファミリごとに、ウィジェットパレット上でまとめられます。ウィジェットパレットで定義の上にあるオプションメニューを使用すると、現在表示されているファミリを変更することができます。定義のファミリを指定する方法の詳細は、312 ページの「定義ファイルの編集」を参照してください。

インスタンスの変更と拡張

定義のインスタンス作成は、構造 (C 構造体または C++ クラス) のインスタンス作成に対応しています。変更が生成されるコードに反映される場合に限り、インスタンスを作成した後で変更を行うことができます。たとえば、ウィジェットがアクセス可能である (例: C++ の場合、ウィジェットに名前があり、適切なアクセスモードを持っている) 場合にのみ、ウィジェットにリソースを設定したり、あるいはウィジェットに子を追加することが可能です。ウィジェットの除去、あるいはその名前の変更を行う

ことはできません。ただし、ルートウィジェットは例外です。インスタンスのルートウィジェットは常に (メンバー関数 *xd_rootwidget()* を介して) アクセス可能であるため、常に変更することができます。

注 - アクセス権のないウィジェットは、配置エディタ上で移動およびコンストレイントの設定を行うことはできません。

派生構造の作成

定義から派生した新しい構造を作成すると役に立つ場合が多くあります。派生構造を作成するには、コアリソースダイアログのコード生成ページにある「構造」オプションを設定します。派生構造は、定義と同じ値にのみ設定することができます。つまり、C 構造体から C++ クラスを派生させることはできません。

定義のコールバック・メソッドの書き換え

インスタンス内の、定義から継承されたメソッドは書き換えることができます。その場合は、インスタンスは定義とは異なる動作を行います。

インスタンスを含むコードのコンパイル

定義のインスタンスを含むデザインから生成されたコードをコンパイルするには、定義コードともリンクする必要があります。これを行うには次の 2 つの方法があります。

1. 定義コードを含むライブラリとリンクする
2. 定義コードとインスタンスコードをコンパイルする

各方法については以下で説明します。

ライブラリの使用

定義コードを含むライブラリをインスタンスコードとリンクさせるには、まず定義のコードをライブラリにコンパイルします。通常、UNIX または C や C++ では、次の方法でコンパイルします。

```
make <definitioncode>.o
```

```
ar r <definitionlib>.a <definitioncode>.o
```

次に、以下が可能になるように、インスタンスを含むコードのメークファイルを編集する必要があります。

1. コンパイラが定義を含むヘッダーファイルを検出できる
   Sun WorkShop Visual は、このヘッダーファイルをインスタンスに対して生成されたコードに自動的にインクルードします。
2. リンカーが定義コードを含むライブラリを検出できる
   ライブラリのフルパス名を「EXTRALIBS」に追加します。

定義とインスタンスのコンパイル

定義のインスタンスをコンパイルするもう 1 つの方法として、定義、定義のインスタンス、対応するメークファイルを同じディレクトリに生成する方法があります。Sun WorkShop Visual では、定義とインスタンスの両方が同じアプリケーションにコンパイルされるようにメークファイルを構成することができます。以下に、その手順を示します。

1. インスタンスを含むデザインを開きます。
2. 「メインプログラム」を生成することを確認します。
3. 「メークファイル・オプション」ダイアログの「新規メークファイル」と「メークファイル・テンプレート」トグルをオンにします。
4. 必要なすべてのファイルを生成します。
5. 定義デザインを開きます。
6. 「メインプログラム」生成トグルをオフにします。
7. 「メークファイル・オプション」ダイアログの「新規メークファイル」トグルをオフにして、「メークファイル・テンプレート」トグルはオンのままにしておきます。
8. 「コードオプション」ダイアログで、「リンク」を「生成しない」(リンク関数は生成済みなので、2 回目の生成ではエラーになる) に設定します。
9. コード、外部宣言ファイル、メークファイル (必要であればリソースも) を生成します。
10. コマンドプロンプトに「**make**」と入力します。

以上で、インスタンスを含むアプリケーションが 1 つ完成します。

定義およびリソースファイル

定義の構成要素であるウィジェットに対するリソース値は、プログラム中に書き込むか、またはリソースファイル内に指定することができます。

インスタンスと定義リソースファイル

定義に対してリソースファイルを指定すると、Sun WorkShop Visual は定義のインスタンスを持つデザインのリソースファイル内にそのファイルをインクルードします。リソースファイルを読み出す Xlib メカニズムが、この指令を解釈して使用し、定義に対してのリソースファイルを検索します。

定義のオンラインヘルプ

定義についての情報を記録して、他の開発者にその内容を伝えるために、定義に対してのオンラインヘルプをつけることができます。<Tab> と矢印キーを使用して、定義のアイコンやボタンに移動し、<osfHelp> (通常は <F1>) を押すと、オンラインヘルプが Sun WorkShop Visual インターフェース内で呼び出されます。

ヘルプファイルは Sun WorkShop Visual ヘルプディレクトリのサブディレクトリに保存されています。

ヘルプディレクトリは helpDir リソースにより決定します。デフォルトでは、`$VISUROOT/lib/locale/${`*LANG*`}/help` となっています。

`VISUROOT` は、Sun WorkShop Visual のインストールディレクトリです。*LANG* は使用するロケールで、デフォルトでは「`C`」です。

テキストのヘルプドキュメント

テキストのヘルプドキュメントは HTML 形式です。ファイルの名前は、ドキュメント名とマーカー名を連結して形成されています。この連結処理は visu.userHelpCatString リソースで行われます。デフォルトではこのリソースは「.」

に設定されています。ファイルには「.html」接尾辞が付けられます。Sun WorkShop Visual は Sun WorkShop Visual.helpDir リソースで指定した検索パスに従いファイルを捜します。

第9章

C++ コードの学習

はじめに

この章では、Sun WorkShop Visual の C++ コード生成機能を使用してアプリケーションに構造を追加する方法、および C++ クラスに対応する再使用可能なウィジェット階層の作成方法を説明します。これらの再使用可能階層は定義と呼ばれ、ウィジェットパレット上に表示されます。また、他のウィジェットと同様 3 階層に追加することができます。この章では基本的には C++ について説明していますが、コールバック・メソッドの節を除き、ほとんどの項目は C での構造コード生成にも適用することができます。相違点がある場合は、注釈で説明しています。

本章は、以下のような形式で説明を進めていきます。

- ウィジェット階層に対応する C++ クラスを作成する
- コールバックを処理するクラス・メソッドを使用する
- 派生クラスおよびプレリュードを使用して生成されたクラスにメンバーを追加する
- Sun WorkShop Visual クラスが派生している基底クラスを修正または置き換える
- クラスを再使用可能な定義に変換し、定義をウィジェットパレットに配置する
- 定義を変更する
- 定義のインスタンスを作成し修正する
- 派生クラスを使用して定義のインスタンスを拡張する
- コールバック・メソッドを書き換える

■ 定義に対してのリソースファイルを生成し使用する

本章の内容をよりよく理解するには、Sun WorkShop Visual を起動し、説明されている操作手順を実際に実行してみることをお勧めします。

構造化コード生成の詳細については、第 8 章「構造化コード生成および再使用可能な定義」を参照してください。

C++ クラスの作成

Sun WorkShop Visual の C++ クラスは、子を持つどのようなウィジェットにも対応します。ウィジェットを C++ クラスとして指定すると、Sun WorkShop Visual はそのウィジェット、および名前が付けられている子孫ウィジェットをデータメンバーとして使用してクラスを生成します。このクラスは、データメンバーおよびメンバー関数を追加して拡張することができ、その結果、階層全体に関連する属性を一箇所に集めることができます。

C++ クラスの指定

以下の操作手順を使用して、メニューバーウィジェットを含むウィジェット階層を作成し、その後メニューバーを C++ クラスとして指定します。この例は、Microsoft Windows のコード生成とは互換性がないことに注意してください。

1. 同一の親ディレクトリを持つ新しいディレクトリ *libmenu* と *cmd* を作成します。
2. ディレクトリ *libmenu* へ移動して、Sun WorkShop Visual を起動します。
3. 図 9-1 に示すようなウィジェット階層を構築します。

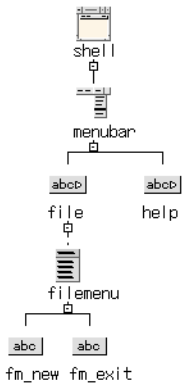


図 9-1　メニューバーウィジェット階層

明示的に名前が付けられたウィジェットだけがクラスのメンバーとして作成されます。名前が付けられていないウィジェットは、それらを作成する関数に対して局所的になります。ウィジェットの命名により、そのクラスのメンバー関数から直接アクセスすることができるようになります。それらのウィジェットはデフォルトで限定公開メンバーであるため、任意の派生クラスのメンバー関数からもアクセスすることができます、

4. 図 9-1 に示すようにウィジェットの変数名を設定します。
5. カスケードボタンおよびプッシュボタンに対しての「ラベル」リソースをそれぞれ次の文字列に設定します。
   File、Help、New、Exit
6. シェルのリソースパネルを使用してシェルウィジェットをアプリケーションシェルとして指定し、シェルウィジェットのタイトルを「Demo」に指定します。

   これで階層の例が完成しました。では次に、メニューバーを C++ クラスとして指定します。

7. ウィジェット階層内でメニューバーウィジェットを選択します。
8. コアリソースパネルの「コード生成」のページを表示します。
9. 図 9-2 に示すように、「構造」オプションメニューから「C++/Java クラス」を選択します。

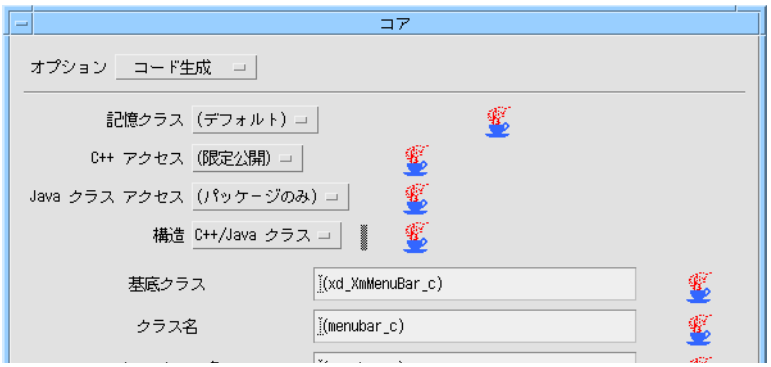


図 9-2　ウィジェットのクラスとしての指定

10. 「適用」をクリックします。

Sun WorkShop Visual は、デフォルトでウィジェット名をデフォルトのクラス名およびインスタンス名として使用するため、*menubar* と名前の付けられているウィジェットからは *menubar_c* という名前のクラスが作成されます。*menubar_c* の基底クラスはウィジェットクラスに依存します。この場合は、*xd_XmMenuBar_c* となります。後で説明しますが、これらの名前は変更することができます。

ウィジェットメンバーのアクセス制御

生成された *menubar_c* クラスは、クラスウィジェット (*menubar*) および名前が付いているすべての子孫ウィジェットをメンバーとして含みます。デフォルトでは、これらのウィジェットは限定公開メンバーとなっていますが、コアリソースパネルを使用して任意のウィジェットのアクセス制御を変更することができます。以下の操作手順に従って、ヘルプメニューをクラスの公開メンバーにします。

1. ウィジェット階層内で、「*help*」と名前が付けられているカスケードボタンを選択します。
2. コアリソースパネルの「コード生成」ページを表示します。

図 9-3 に「コード生成」のページを示します。

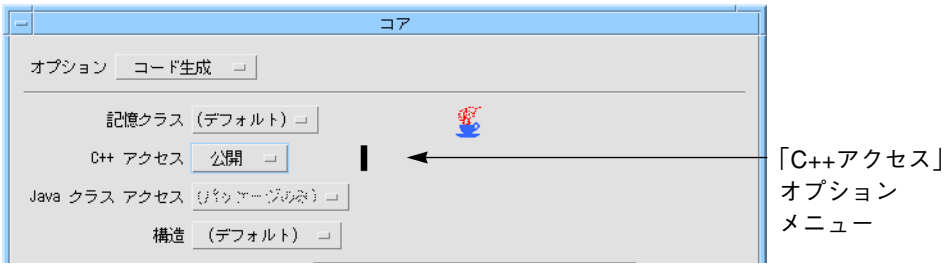


図 9-3　メンバーアクセス制御

3. 「C++ アクセス」オプションメニューから「公開」を選択し、「適用」をクリックします。

C++ クラス・コード生成

クラスに対して生成されたコードは、生成された外部宣言ファイルにあるクラス宣言と、C++ の基本コードファイルにある実装で構成されます。これらのファイルを生成するには、以下のように操作を行います。

1. 「コード生成」ダイアログを表示して、「言語」メニューで「C++」が選択されていることを確認します。
2. 「コード」テキストフィールドに *menubar.cpp* と入力して、「生成」トグルをオンにします。
3. 「外部宣言」テキストフィールドに *menubar.h* と入力して、「生成」トグルをオンにします。
4. 「メインプログラム」テキストフィールドに *menubar.cpp* と入力して、「生成」トグルをオンにします。
5. 「メークファイル」テキストフィールドに *Makefile* と入力して、「生成」トグルをオンにします。
6. 「メークファイル」テキストフィールドの横にある「オプション」ボタンを押します。

   「メークファイル・オプション」ダイアログが表示されます。
7. 「メークファイル・オプション」ダイアログで「新規メークファイル」と「メークファイル・テンプレート」を両方オンにして、「了解」ボタンを押します。

8. 「コード」テキストフィールドの横にある「オプション」ボタンを押します。

   図 9-4 に示す「C++コードのオプション」ダイアログが表示されます。

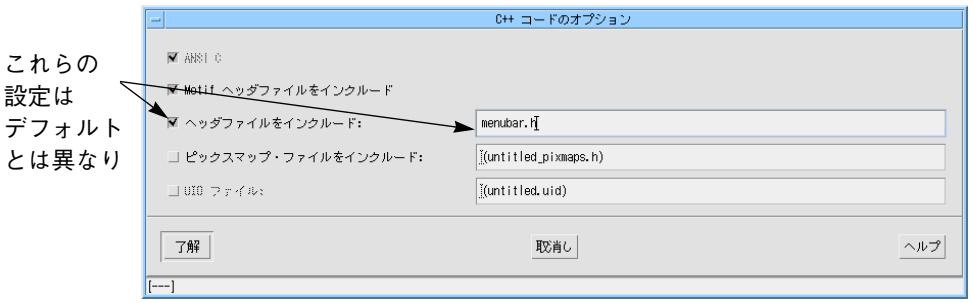


   図 9-4　「C ++コードのオプション」ダイアログ

9. 図 9-4 で示すように「ヘッダーファイルをインクルード」テキストフィールドに *menubar.h* と入力して、「了解」ボタンを押します。

10. 「コード生成」ダイアログの下の方にある「オプション」ボタンを押して、オプションを図 9-5 で示すように設定して、「了解」ボタンを押します。「文字列」リソースがコードに生成されるように設定されていることを確認してください。

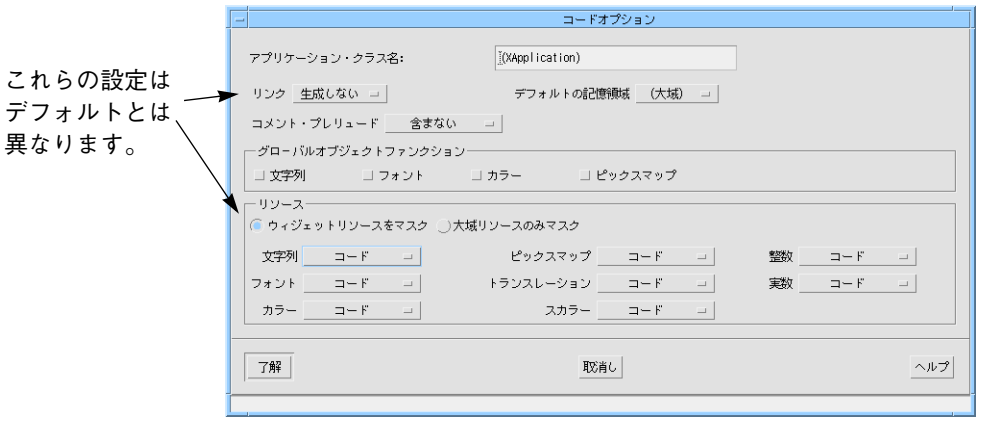


   図 9-5　menubar_c クラスの「コードオプション」ダイアログ

11. 「コード生成」ダイアログの「生成」ボタンを押します。

C++ 外部宣言ファイルである *menubar.h* には、クラスに対しての宣言が含まれています。

```
...
class menubar_c: public xd_XmMenuBar_c {
public:
                  virtual void create (Widget parent, char
*widget_name =
                      NULL);
                  Widget help;
protected:
                  Widget menubar;
                  Widget file;
                  Widget filemenu;
                  Widget fm_new;
                  Widget fm_exit;
};


typedef menubar_c *menubar_p;
...
```

このメニューバーに対しての新しいクラスは、既存のクラスである *xd_XmMenuBar_c* を基底クラスとしています。メニューバーおよび名前の付いているその子孫ウィジェットは、すでに公開として指定した *help* ウィジェットを除いては限定公開メンバーです。

C++ 基本コードファイルである *menubar.cpp* は、新しいクラスに対しての作成関数を含んでいます。この関数は、メニューバーウィジェットおよびその子孫を作成します。これらは、*menubar_c* のコンストラクタでは作成されません。このため、クラスのインスタンスを作成した後にウィジェットを作成することが可能になります。

```
...
#include <menubar.h>
...
void menubar_c::create (Widget parent, char *widget_name)
```

```
{
                Widget children[2];/* マネージする子 */
                Arg al[64];/* 引数リスト*/
                register int ac = 0;/* 引数の個数 */
                XmString xmstrings[16];/* XmStrings の一時記憶領域 */
                if ( !widget_name )
                    widget_name = "menubar";
                menubar = XmCreateMenuBar ( parent, widget_name, al,
                    ac );
                _xd_rootwidget = menubar;
                xmstrings[0] = XmStringCreateLtoR("File",
                    (XmStringCharSet)XmFONTLIST_DEFAULT_TAG);
                XtSetArg(al[ac], XmNlabelString, xmstrings[0]);
ac++;
                file = XmCreateCascadeButton ( menubar, "file", al,
                    ac );
                ac = 0;
...
                children[ac++] = file;
                children[ac++] = help;
                XtManageChildren(children, ac);
                ac = 0;
}
```

menubar.cpp ファイルには、階層全体に対しての作成関数も含まれています。この関数は、クラス外の任意のウィジェットを作成します。この例ではシェルだけです。そして、*menubar_c* クラスのインスタンスを作成し、クラスのウィジェットメンバーを作成するための *menubar_c::create()* を呼び出します。

```
void create_shell (Display *display, char *app_name, int app_argc,
char **app_argv)
{
                Widget children[1]; /* マネージする子 */
```

```
                    Arg al[64]; /* 引数リスト */
                    register int ac = 0; /* 引数の個数 */
                    XtSetArg(al[ac], XmNallowShellResize, TRUE); ac++;
                    XtSetArg(al[ac], XmNtitle, "Demo"); ac++;
                    XtSetArg(al[ac], XmNargc, app_argc); ac++;
                    XtSetArg(al[ac], XmNargv, app_argv); ac++;
                    shell = XtAppCreateShell ( app_name,
"XApplication",
                        applicationShellWidgetClass, display, al, ac );
                    ac = 0;
                    menubar = new menubar_c;
                    menubar->create ( shell, "menubar" );
                    XtManageChild ( menubar->xd_rootwidget());
}
```

生成された C++ コードのコンパイル

コードの生成時に「メインプログラム」トグルを設定したため、*menubar.cpp* にはメインプログラムも含まれます。したがって、アプリケーションは現状のまま構築することができます。

Sun WorkShop Visual が生成した C++ コードは、そのまま簡単に構築することができます。ただし、生成されたクラスが派生する xd_XmMenuBar_c などの基底クラスが使用できることが必要です。`$VISUROOT/src/xdclass/lib` ディレクトリは、デフォルト基底クラスに対してのソースを含んでいます。`$VISUROOT/src/xdclass/h` は、ヘッダーファイルを含んでいます。

libxdclass.a ライブラリが構築されていない場合は、以下の手順に従って、供給されるメークファイルを使用するライブラリを構築します。

1. Sun WorkShop Visual インストールディレクトリ中の `src/xdclass/lib` ディレクトリに移動します。
2. `$VISUROOT` が、Sun WorkShop Visual のインストールのルートを指すように設定します。

3. 次のように入力します。

   make

   make が終了すると、libxdclass.a ライブラリが使用可能となります。

   これで、生成されたメークファイルを使用してプログラムを構築する準備ができました。

注 - 生成されたメークファイルには $VISUROOT への参照が含まれているため、プログラムを構築する前に必ず環境変数 $VISUROOT を設定してください。

---

4. 次のように入力して、メニューバーのプログラムを構築します。

   make

5. 次のように入力して、アプリケーションを実行します。

   menubar

   アプリケーションは、クラスがまったく存在しない場合と同様の外観および動作を持ちます。

コールバック・メソッド

これまで、例としてウィジェット階層をクラスとして指定する方法と、生成されるコードの形式を見てきました。現段階では、ウィジェット階層をクラスとしては活用していません。

注 - この節は、C++ プログラミングに特に関連している説明です。C の枠組みの中では、従来のコールバックの機構を使用することができるため、C++ プログラミングを行わないユーザーは、344 ページの「定義の作成」に進んでください。

コールバックおよびメンバー関数

Sun WorkShop Visual では、クラスメンバー関数をコールバック関数として指定することができる、単純なメカニズムを提供しています。これらのメカニズムをコールバック・メソッドと呼びます。使用する技術については、300 ページの「コールバック・メソッド」で詳しく説明しています。

コールバック・メソッドの指定

コールバック・ダイアログを使用すると、イベントに対する応答で呼び出されるメンバー関数を指定することができます。特定のウィジェットに対してコールバック・メソッドを指定すると、呼び出されるメソッドは、そのウィジェットの最も近くにあるクラス指定されたウィジェット (たいていはウィジェットそのもの) に属します。たとえば、メニューバー例にあるメニューボタン上のコールバック・メソッドは、*menubar_c* クラスのメンバー関数を呼び出します。

以下の手順を使用して、*fm_new* ボタンのクラスメソッドを宣言します。

1. ウィジェット階層で *fm_new* ボタンを選択します。
2. 「ウィジェット」メニューから「コールバック」を選択するか、ツールバーの「コールバック」ボタンを押して、コールバックダイアログを表示します。
3. 「コールバックのリスト」から「活性化 (activate)」を選択します。
4. 「メソッドの名前」フィールドに次のように入力します。

   `OnNew`
5. 「追加」をクリックします。

   これにより、図 9-6 に示すように、局所的メソッドのリストに *OnNew* が追加されます。

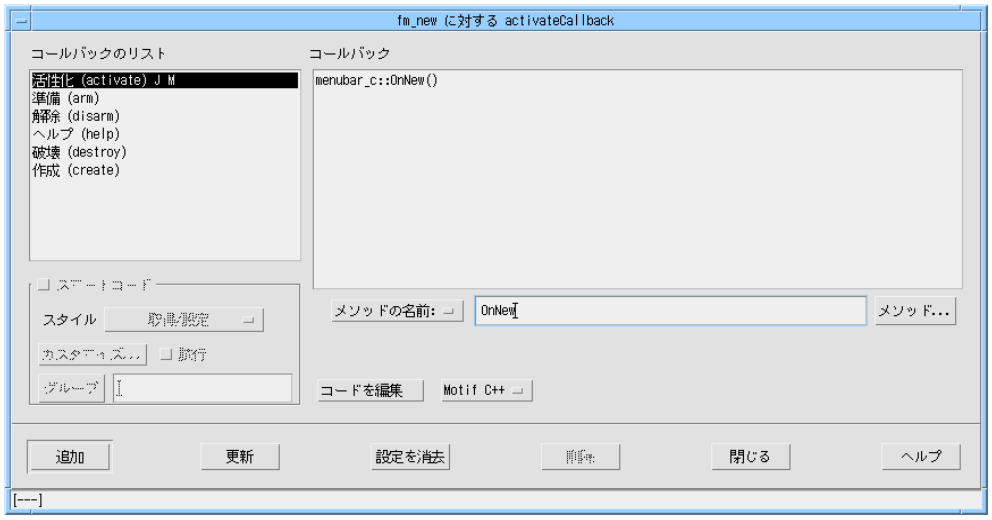


図 9-6　コールバック・メソッドの指定

これは、メンバー関数 *menubar_c::OnNew()* をボタン *fm_new* の活性化コールバックを扱うメソッドとして指定します。この作業を行う際、ユーザーがまだ宣言を行なっていない場合には、Sun WorkShop Visual は親ウィジェットである *menubar* 上でもメソッドの宣言を行います。

同様に、 *fm_exit* ボタンにコールバック・メソッドを入力します。

6. ウィジェット階層で *fm_exit* ボタンを選択します。
7. 「コールバック」ダイアログで「活性化 (activate)」を選択します。
8. 「メソッドの名前」フィールドに次のように入力します。

   `OnExit`

9. 「追加」をクリックします。
10. 「コールバック」ダイアログを閉じます。

クラスには *menubar_c::OnNew()* と *menubar_c::OnExit()* という 2 つのコールバック・メソッドが存在することになります。

コールバック・メソッドに対してのコード生成

クラス内からコールバック・メソッドを使用すると、Sun WorkShop Visual は 2 つの別のメンバー関数に対し、ひとつに対しては完全な実装を、もうひとつに対してはスタブという宣言を生成します。

1. 「コード生成」ダイアログを表示して、「言語」オプションが「C++」に設定されていることを確認します。
2. 「スタブ」とラベルの付いたテキストボックスに *menubarS.cpp* と入力して、「生成」トグルをオンにします。
3. 「メークファイル・オプション」ダイアログを表示します。
4. 「新規メークファイル」トグルをオフに、「メークファイル・テンプレート」トグルをオンにして、「了解」ボタンを押します。
5. 「コード生成」ダイアログでメークファイルの「生成」トグルをオンにします。
6. 「生成」ボタンを押します。

menubar.h 内のクラス宣言を見てください。各コールバック・メソッドに対して 2 つの新しいメンバー関数が追加されています。

```
class menubar_c: public xd_XmMenuBar_c {
public:
...
                    static void OnExit( Widget, XtPointer, XtPointer );
                    virtual void OnExit( Widget, XtPointer );
                    static void OnNew( Widget, XtPointer, XtPointer );
                    virtual void OnNew( Widget, XtPointer );
};
```

これらの関数の静的バージョンのみが、Xt コールバックが予定している引数リストを持ちます。したがって、Xt がコールバック・メソッド *menubar_c::OnNew()* を呼び出す場合には、C++ コンパイラは引数リストにもとづいて静的バージョンを選択します。

menubar の作成関数には以下に示す行が含まれていることに注意してください。

```
XtAddCallback (fm_new, XmNactivateCallback, OnNew, (XtPointer)
```

```
                    this);
```

静的関数に対してのコードは、*menubar.cpp* にも生成されます。この関数は、クライアントデータとして渡されるインスタンス・ポインタを使用して、単純に非静的な仮想メンバー *OnNew(Widget, XtPointer)* を呼び出します。

```
void menubar_c::OnNew( Widget widget, XtPointer client_data,
XtPointer call_data )
{
                menubar_p instance = (menubar_p) client_data;
                instance->OnNew ( widget, call_data );
}
```

ユーザーは、非静的仮想メンバー関数 *OnNew(Widget, XtPointer)* に対するコードを書く必要があります。この関数に対してのスタブ (原型) は、*menubarS.cpp* に生成されます。

```
void
menubar_c::OnNew (Widget w, XtPointer xt_call_data )
{
                XmAnyCallbackStruct *call_data =
(XmAnyCallbackStruct *)
                    xt_call_data;
}
```

Sun WorkShop Visual は、このパターンにもとづいて、階層内で使用されるすべてのコールバック・メソッドに対してコードを生成します。この例では、類似したコードが OnExit() に対して生成されます。

OnNew() および *OnExit()* は、*fm_new* および *fm_exit* プッシュボタンから呼び出されますが、関数は *menubar_c* クラスのメソッドです。これは、クラス内でウィジェットの動作を定義するすべてのコールバック関数は 1 カ所に保存されていることを意味します。また、すべてのコールバック関数はクラスのインスタンスデータへのアクセス権を持っており、それを使用して情報を共有することができます。

コールバック・メソッドの実装

アプリケーション動作は、コールバック・メソッドを実装することにより追加されます。Sun WorkShop Visual の編集機能を使用して、コールバック・メソッドを編集することもできます。以下に示す手順で、*OnExit()* メソッドを実装します。

1. *fm_exit* ボタンを選択します。
2. 「コールバック」ダイアログを表示して、*OnExit()* コールバックを選択します。
3. 「コードを編集」ボタンを押します。

   <現作業ディレクトリ>/libmenu にある menubarS.cpp という名前のファイルが開きます。このファイルに記述された OnExit メソッドにコードを追加することができます。コールバックの編集についての詳細は、265 ページの「Sun WorkShop Visual 内でのコールバックコードの編集」を参照してください。

4. 以下のように、*menubar_c::OnExit()* の実装を行います。

```
void
menubar_c::OnExit (Widget w, XtPointer xt_call_data )
{
                  XmAnyCallbackStruct *call_data =
(XmAnyCallbackStruct*)
                      xt_call_data;
                  exit(0);
}
```

5. 「コールバック」ダイアログを閉じます。
6. 次のように入力してメニューバーのプログラムを構築します。

   make

7. 次のように入力してアプリケーションを実行します。

   menubar

8. ファイルメニューから「閉じる」を選択します。

   プログラムが終了したことを確認してください。

メソッドの属性の編集

コールバック・メソッドには、アクセスレベルおよび純粋仮想か否かという 2 つの属性があります。アクセスレベルは、そのメソッドが派生クラスおよび外部コードからアクセスできるかどうかを決定します。また、メソッドは、基底クラスでの実装を持たないことを示す純粋仮想として指定することができます。詳細は、第 8 章「構造化コード生成および再使用可能な定義」の 295 ページの「C++ クラス」を参照してください。

属性はコールバックが最初に指定される時には、デフォルトに設定されています。属性の初期設定は、クラスの祖先を持つ、またはクラスである任意のウィジェットで行うことができます。しかし、その変更はクラスのルートウィジェット上で行われます。たとえば、クラス *menubar_c* の *OnNew()* コールバック・メソッドは、*menubar* ウィジェットそのものを使用してのみ編集することができます。

1. ウィジェット階層でメニューバーウィジェットを選択します。
2. 「ウィジェット」メニューで「メソッド宣言」を選択し、「メソッド宣言」ダイアログを表示します。

   クラスに対して宣言されたコールバック・メソッドのリスト (特定のイベントに対して呼び出されるメソッドのリストではない) が表示されます。このパネルを使用すると、コールバック・メソッドの編集、およびそのクラスのイベントによってではなく派生クラスにより呼び出される可能性のあるメソッドを宣言することができます。
3. メソッド *OnNew()* を選択し、「純粋仮想」トグルを設定します。
4. 「追加/更新」ボタンをクリックして変更を適用します。図 9-7 に示すような結果になります。

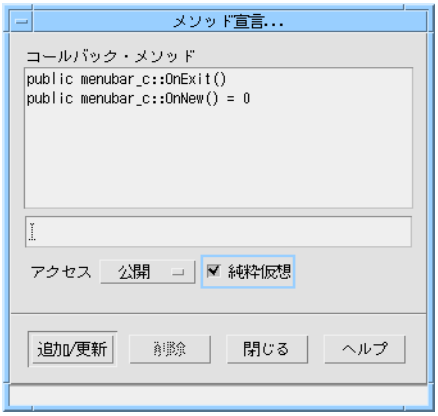


図 9-7　コールバック・メソッドの編集

5. コードを再度生成します。

   純粋仮想メンバー関数に対する実装は、無効ではありませんが行う必要はありません。

6. スタブファイル *menubarS.cpp* を *temp.cpp* にコピーします。

7. スタブファイル *menubarS.cpp* を編集して *OnNew()* に対してのスタブ (例: 中括弧で囲まれたコード) を削除します。

8. 前回と同じ手順でメニューバープログラムを構築します。

   C++ コンパイラは、次のようなエラーメッセージを発します。

   `"menubar.cpp" 100 行目: 抽象クラス menubar_c に対して変数を作成することはできません。`

   menubar_c クラスが純粋仮想関数を含んでいるためインスタンス化することはできません。したがって、エラーが発生することになります。この時点では、このクラスは基底クラスとしてのみ使用することができます。本章の後半では、このクラスを派生クラスの基礎として使用します。

9. *temp.cpp* を *menubarS.cpp* にコピーして、*temp.cpp* を削除します。

クラスメンバーの追加

この節および次の「派生クラスの作成」では、Sun WorkShop Visual の生成されたクラスにメンバーを追加する技術を説明します。ここで説明する技術は以下の 2 つです。

- Sun WorkShop Visual のプレリュード・メカニズムの使用
- 派生クラスの生成

注 - これらは、C++ プログラミングについてのみ有効であることに注意してください。

クラスメンバーをプレリュードとして追加

少数のメンバーを追加する最も簡単な方法としては、プレリュード機能を使用します。この機能を使用すると、Sun WorkShop Visual にコードの断片を入力することができ、また、それらを生成されたコードに渡すこともできます。

注 - Sun WorkShop Visual の編集機能を使用して、生成されたコードに直接プレリュードを入力することもできます。詳細は 280 ページの「生成されたファイルのカスタマイズ : プレリュード」を参照してください。

次の例では、コードを生成する前にコードプレリュードダイアログを使用してプレリュードを追加します。

1. ウィジェット階層でメニューバーウィジェットを選択します。
2. ウィジェットメニューをプルダウンして、「コードプレリュード」を選択します。

   図 9-8 に示すダイアログが表示されます。

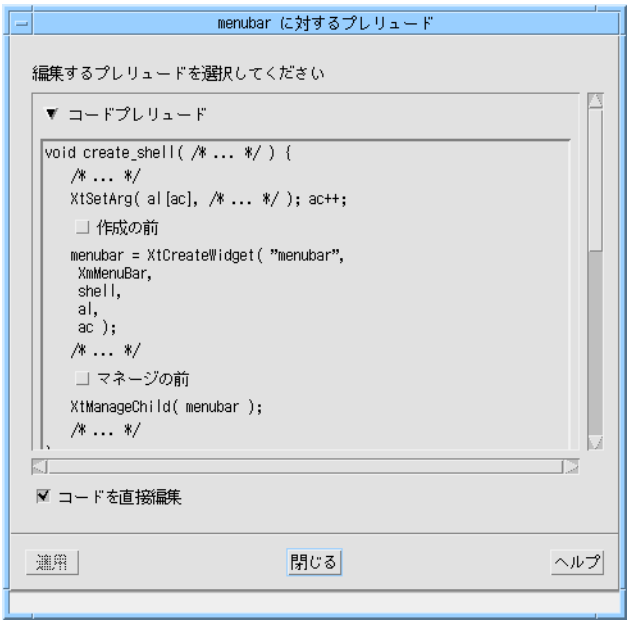


図 9-8　クラスへの限定公開メンバーの追加

3. コードプレリュードダイアログの「コードを直接編集」トグルをオフにします。

   プレリュードダイアログの右側に編集領域が表示されます。この編集領域にプレリュードを追加します。

4. 「メソッド・プレリュード」とラベルのついたテキスト内にある「限定公開メソッド」トグルをオンにします (このトグルを表示するには、テキストをスクロールしなければならない場合もあります)。

5. 右側のテキスト領域で Tab キーを押して次のように入力し、Return キーを押します。

   ```
   int modified;
   ```

6. 「適用」をクリックし、次に「閉じる」をクリックします。

   この操作の結果を確認するためには、以下のようにします。

7. コードを再度生成します。

8. コードファイル *menubar.h* 内で、クラス *menubar_c* にメンバーが追加されていることを確認します。

派生クラスの作成

コードプレリュード・ダイアログを使用すると、少数のコードを簡単に挿入することができます。実質的な機能を追加するためには、多くの場合、生成されたクラスから派生する新しいクラスを作成する方が良いでしょう。2 つのクラス間の論理的な違いを使用して、メンバーの追加および仮想関数の実装を行うことができます。

注 - ここでは、C++ プログラミングについてのみ言及していることに注意してください。

デフォルトでは、ルートウィジェットの変数名をもとに C++ クラスの名前が決まります。したがって、ウィジェット *menubar* のクラスは *menubar_c* になります。

```
class menubar_c: public xd_XmMenuBar_c {
...
};
```

クラスのインスタンスを作成するためにコードを生成する場合は、同じ名前が使用されます。

```
menubar = new menubar_c;
```

生成されたクラスをある名前で宣言し、別の名前でインスタンスを作成するように、その動作を変更することができます。たとえば次のようにします。

```
menubar = new mymenubar_c;
```

この変更を行うためには、コアリソースパネルにあるコード生成ページの「インスタンス名」を使用します。

1. ウィジェット階層でメニューバーウィジェットを選択します。
2. コアリソースパネルの「コード生成」ページを表示します。
3. 「インスタンス名」を *mymenubar_c* に設定します (図 9-9 参照)。

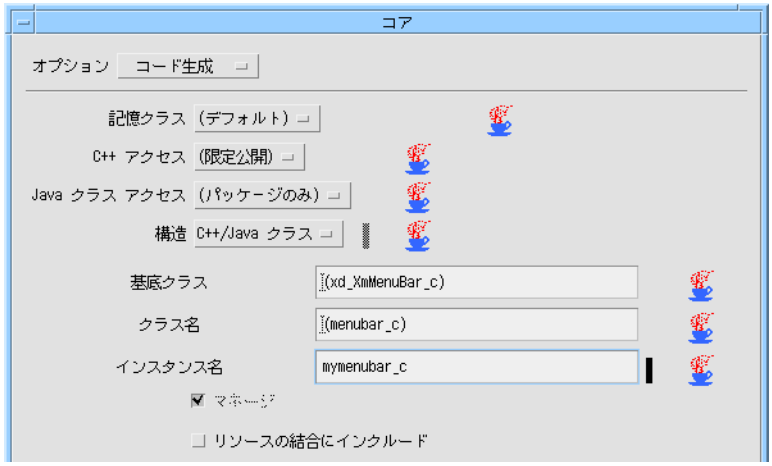


図 9-9　インスタンスの名前の変更

4. 「適用」をクリックして、「閉じる」をクリックします。

5. コードを再度生成します。

   元のクラスである *menubar_c* は、以前とまったく同様に宣言されています。しかし、Sun WorkShop Visual がクラスのインスタンスを作成するためにコードを生成する場合には、「インスタンス名」での名前が使用されます。

```
menubar = new mymenubar_c;
```

派生クラスのコード作成

Sun WorkShop Visual は、*mymenubar_c* クラスに対してコードを生成しません。ユーザーは、クラスおよび含まれるメソッドを実装するためのコードを宣言する、ヘッダーファイルを提供する必要があります。新しいクラスは、*menubar_c* から派生しなければならないことを除いては、何ら制約を受けていません。この例の場合は、以下に示すコード例を使用します。

1. 以下のコードを使用して、新しいファイルである *mymenubar.h* に、派生クラス *mymenubar_c* に対してのクラス宣言を作成します。

```
#ifndef _mymenubar_h
#define _mymenubar_h
#include <menubar.h>
class mymenubar_c: public menubar_c {
```

```
public:
// コンストラクタ
                    mymenubar_c();
                    // 継承される純粋仮想に対して実装を提供する
                    void OnNew(Widget, XtPointer);
};
#endif
```

新しいクラスは *menubar_c* から派生するため、Sun WorkShop Visual でそのクラスに対して宣言したすべてのウィジェットメンバーおよびメンバー関数が継承されます。新しいメンバーは任意の数だけ追加することができます。ここでは、コンストラクタ関数と *OnNew()* 仮想コールバック・メソッドの実装を追加します。

2. 以下のコードを使用して、新しいファイル *mymenubar.cpp* に、派生クラス *mymenubar_c* に対しての実装を作成します。

```
#include <mymenubar.h>
mymenubar_c::mymenubar_c()
{
                    modified = TRUE;
}
void
mymenubar_c::OnNew(Widget, XtPointer)
{
                     // 修正されたフラグをリセットする
                    if (modified)
                        modified = FALSE;
}
```

これで、クラスに対してのすべてのコードが完了します。生成された C++ コードモジュール *mymenubar.cpp* は、派生クラス *mymenubar_c* に対してのヘッダーファイルを組み込む必要があります。これは、C++ の「コード生成」ダイアログにある「ヘッダーファイルをインクルード」を使用して実行することができます。

3. 「コード生成」ダイアログを表示します。

4. 「コード」ファイル名フィールドの横にある「オプション」ボタンを押します

   「コードオプション」ダイアログが表示されます。

5. 「ヘッダーファイルをインクルード」フィールドに *mymenubar.h* と入力して「ヘッダーファイルをインクルード」トグルをオンにし、「了解」ボタンをクリックします。

   図 9-10 に、新しいファイル名を表示した「コードオプション」ダイアログを示します。

図 9-10 宣言ヘッダーの変更

6. コードを生成します。
7. メークファイルの中の行「XD_ALL_C_SOURCES=...」の前に以下の行を追加します。

```
XD_CC_SOURCES=mymenubar.cpp
XD_CC_OBJECTS=mymenubar.o
```

8. メークファイルの末尾に以下の行を追加します。

```
mymenubar.o:mymenubar.cpp
                $(CCC) $(CCFLAGS) $(CPPFLAGS) -c mymenubar.cpp
```

注 - コンパイラの命令行のインデントは意図的なものです。

---

9. メークファイルを保存します。
10. 前回と同じ手順でメニューバープログラムを構築します。

この時点で、「ファイル」メニューの「新規」ボタンを押すと、アプリケーションはクラス *mymenubar_c* を使用し、その*OnNew()* メソッドを呼び出します。この動作が正しく行われることを確認するには、*mymenubar_c::OnNew()* を拡張してメッセージを出力します。

コンストラクタに対しての実際の引数を、クラス名と一緒に供給することができます。たとえば、「インスタンス名」文字列を「mymenubar_c ("Hello World")」に設定すると、Sun WorkShop Visual は次のようなコードを生成します。

```
menubar = new mymenubar_c ( "Hello World" );
```

定義の作成

ウィジェットの階層がクラスとしていったんカプセル化されると、それを定義に変換することにより他のデザインで再度使用することができます。
定義は、Sun WorkShop Visual ウィジェットパレットに追加することができる、再使用可能なウィジェットのグループです。パレットから定義を選択すると、その構造のインスタンスがデザイン内に作成されます。Sun WorkShop Visual がインスタンスを含んでいるコードを生成する場合、ウィジェットを作成するための定義の作成関数が呼び出されます。

前提条件

ウィジェットの階層は、以下の条件を満たす場合に定義にすることができます。

- ルートウィジェットが変数名を持っている
- ルートウィジェットが C++ クラスまたは C データ構造として指定されている
- ルートウィジェットが別の定義の一部ではない
- ウィジェット階層が定義を含んでいない
- ウィジェット階層が大域的または静的ウィジェットを一切含んでいない

定義の指定

メニューバークラスを定義として指定するためには、以下のように操作します。

1. 階層内でメニューバーウィジェットを選択します。
2. ウィジェットメニューをプルダウンして、「定義」トグルを設定します。

   メニューバーウィジェットおよびその子孫が色付きの区画で囲まれ、定義を構成することを示します。
3. デザインを *menubar.xd* として保存します。

注 - 定義を含んでいるデザインは、パレットに追加する前に保存する必要があります。Sun WorkShop Visual は、定義が使用されるごとに保存されたデザインファイルを使用します。単一のデザインファイル内には複数の定義を持たせることができますが、各ファイルに定義をひとつだけ持たせておくと、追跡が簡単になります。

定義を作成すると、ウィジェットが定義内に凍結されます。凍結されたウィジェットのリソースパネルは使用できなくなるため、ウィジェットの追加またはウィジェット名の変更を行うことはできません。定義を構成するウィジェットの変更は、一時的に定義を取り除くことによってのみ行うことができます。この作業は、定義を使用するデザインとの矛盾を発生させるおそれがあるので、十分な注意が必要です。詳細は、第 8 章「構造化コード生成および再使用可能な定義」、315 ページの「定義の変更」を参照してください。

パレットへの定義の追加

この節では、新しい定義をウィジェットパレットに追加する方法を説明します。

1. デザイン階層でメニューバーウィジェットを選択します。
2. 「パレット」メニューから「定義を編集」を選択します。

   図 9-11 に示すダイアログが表示されます。

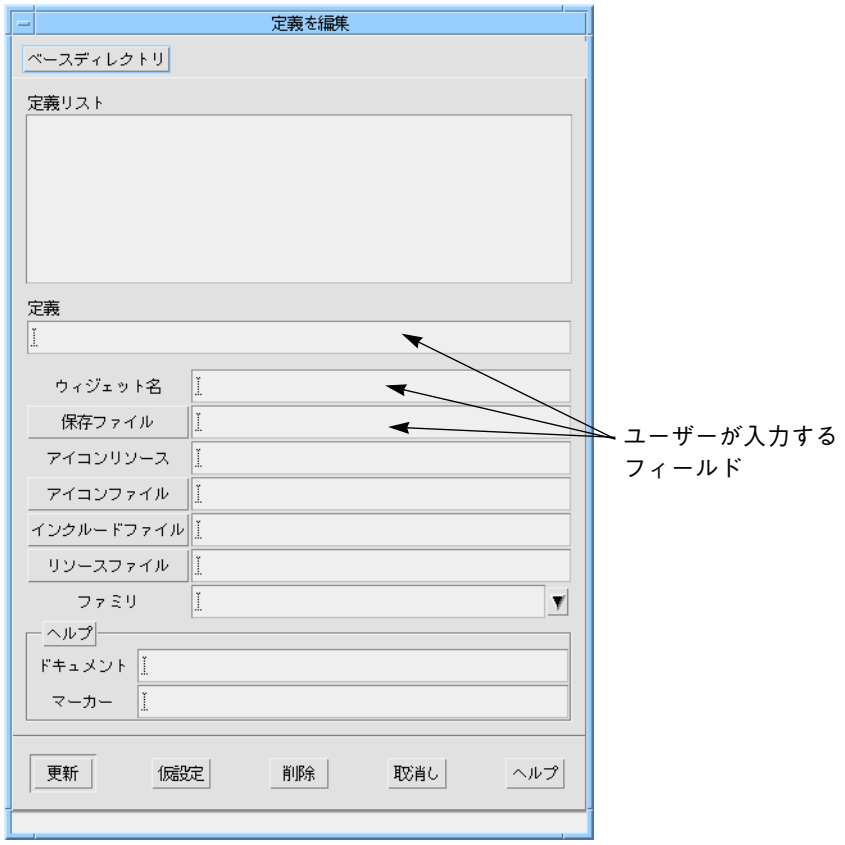


図 9-11 パレットへの定義の追加

このダイアログを使用して、すでに定義マークが付けられているウィジェットを新しい定義として追加、定義を削除、あるいは既存の定義を編集をすることができます。定義を追加するには、以下の情報を入力する必要があります。

- 定義: 定義名
- ウィジェット名: 定義のルートウィジェットの名前
- 保存ファイル: 保存されたデザインファイルの名前 (.xd)

その他の属性については、第 8 章「構造化コード生成および再使用可能な定義」の 319 ページの「定義のオンラインヘルプ」を参照してください。

作成時に設定されない属性は、後で設定することができます。たとえば、定義のテストおよびデバッグは、そのアイコンをデザインする前に実行できます。

3. 「仮設定」ボタンを押します。

   「定義」、「ウィジェット名」、「保存ファイル」、および「インクルードファイル」フィールドにそれぞれの内容が設定されます。

   メニューバーに指定した「インスタンス名」の名前は、定義が使用されるごとに適用されます。

4. 「アイコンファイル」フィールドに次のように入力します。

   ```
   menubar.xpm
   ```

   アイコンは使用しなくても構いません。アイコンを指定しない場合は、ウィジェットパレットに追加される定義プッシュボタンのラベルとして、ルートウィジェットの名前が使用されます。指定したアイコンファイルが存在しない場合は、定義のルートウィジェットの (Sun WorkShop Visual で使用している) アイコンファイルが、パレット上に表示されます。

   Sun WorkShop Visual のピックスマップ・エディタを使用して、定義にするデザインのアイコンを作成することもできます。その場合はウィジェット階層での選択を示すために使用される領域の色には「none」を使用します。ピックスマップエディタの詳細は 184 ページの「ピックスマップ・エディタの使用」を、新しいウィジェットパレットの作成方法については 807 ページの「パレットアイコン」を参照してください。

   Sun WorkShop Visual リソースファイルを使用して、アイコンを指定することもできます。アイコンの指定を行うには、Sun WorkShop Visual リソースの名前を「アイコンリソース」フィールドに指定し、そのリソースを Sun WorkShop Visual リソースファイルにあるファイル名に設定します。

   「定義を編集」ダイアログにあるその他のフィールドについては、本章で後述します。

5. 「更新」をクリックします。

   指定したアイコンが Sun WorkShop Visual ウィジェットパレットに表示されます。子としてメニューバーを持つことができるウィジェットが選択されると、そのアイコンは有効になります。

定義に対してのコードの生成

定義に対してのコードは、公開ヘッダーファイル (外部宣言ファイル) 内の宣言と、実装を含んでいるコードモジュールのふたつの部分で構成されます。インスタンスを含んでいるアプリケーションをコンパイルするために、コードモジュールが公開である必要はありません。ライブラリにおいて、コンパイルされた形式で使用可能にすることができます。

定義に対してのコードだけ、つまりシェルまたは他のウィジェット、およびメインプログラムを除いたコードだけを生成するには、以下の手順を使用します。

シェルウィジェットに対してコードが生成されないようにするためには、次のようにします。

1. 階層でシェルウィジェットを選択します。
2. コアリソースパネルを表示して、「コード生成」ページを表示します。
3. 「構造」オプションメニューを「子のみ生成」に設定して、「適用」をクリックし、「閉じる」をクリックします。

   この操作の後、Sun WorkShop Visual は C++ クラス、関数またはデータ構造体として指定されていないシェルおよびシェルの子を無視します。コードはメニューバーおよびその子孫にのみ生成されます。

   また、メインプログラムの生成を以下のようにして抑制します。

4. 「コード生成」ダイアログを開きます。
5. 「メインプログラム」に対する「生成」トグルをオフにします。
6. 「コード」、「スタブ」、および「外部宣言」に対する「生成」トグルがオンになっていて、外部宣言ファイル名として *menubar.h* が指定されていることを確認します。
7. 「メークファイル」に対する「生成」トグルをオフにします。
8. 「生成」ボタン押します。
9. デザインファイルを保存します。

これにより、定義および対応するクラスのコード作成に必要な処理が完了し、アプリケーションで使用できるようになります。この実装を再使用するには、ライブラリとして使うことが一般的です。

10. 次のように入力して、ライブラリを生成します。

```
make menubar.o
make menubarS.o
make mymenubar.o
ar r libmenu.a *.o
```

インスタンスの作成

定義はパレット上のウィジェットと同様の方法で使用することができます。パレット上のボタンをクリックすると、定義のインスタンスが作成されます。Sun WorkShop Visual は定義の階層をツリーにコピーします。ここで階層の修正および拡張を行うことができます。生成されたコードには、Sun WorkShop Visual がインスタンスを作成するための定義の作成関数の呼び出しを組み込みます。

menubar 定義を使用して新しいデザインを構築するには、以下のようにします。

1. 「ファイル」メニューから「新規」を選択します。
2. シェル、メインウィンドウ、新しい menubar 定義のウィジェットパレットアイコンをクリックします。

   新しい定義がウィジェットパレットに追加されて、347 ページの手順 4 で指定したアイコンが作成されます。

   これにより、図 9-12 に示すウィジェット階層が作成されます。

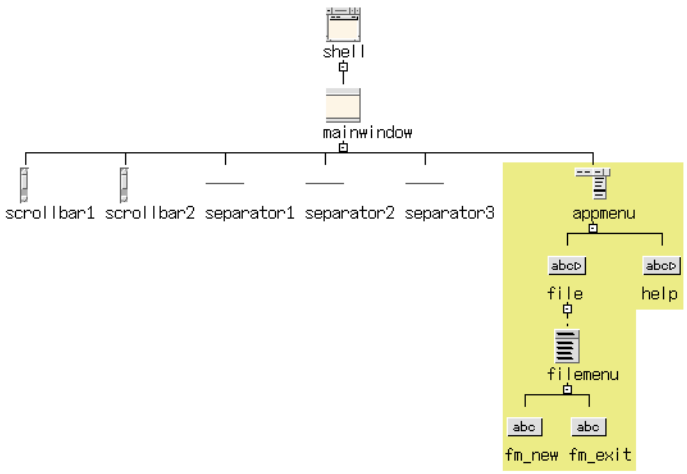


図 9-12 定義のインスタンスを含んでいる階層

インスタンスの構成要素は、単一の項目を形成することを示す色付きのボックスで囲まれています。ルートウィジェットを除くすべてのウィジェットには、元の定義で持っていた名前と同じ名前が指定されます。ルートウィジェットには、<ウィジェットクラス><*n*> 形式のデフォルト名が割り当てられます。コードの信頼性を高めるためにも、明示的な名前を割り当てます。

3. インスタンスのルートウィジェット、 シェルおよびメインウィンドウ に、図 9-12 のように名前を付けます。
4. シェルリソースパネルを使用して、シェルウィジェットをアプリケーションシェルに指定します。

インスタンスの修正および拡張

インスタンスを作成した後、生成されたコード内で変更を実行できる場合に限り、インスタンスを変更することができます。たとえば、ウィジェットが公開アクセスを持っている場合はそのウィジェットへのリソースの設定、あるいは子の追加を行うことができます。ウィジェットの削除、名前の変更はできません。しかし、ルートウィジェットは例外となります。インスタンスのルートウィジェットはメンバー関数 *xd_rootwidget()* を使用して呼び出すことができるため、常に修正することができます。

この例では、定義のすべての構成要素は *help* ボタンを除いて限定公開となっています。これは、*help* ボタンのラベルのみが変更可能であることを意味します。同様に、*help* ボタンの下に別のウィジェットを追加することができますが、*filemenu* メニューには追加することはできません。

1. ウィジェット階層内で *help* ウィジェットを選択します。
2. ウィジェットパレットでメニューアイコンをクリックし、プッシュボタンアイコンをクリックします。

   help カスケードボタンの下に選択項目を 1 つだけ持つメニューが追加されます。
3. ウィジェット名を図 9-13 に示すように設定します。

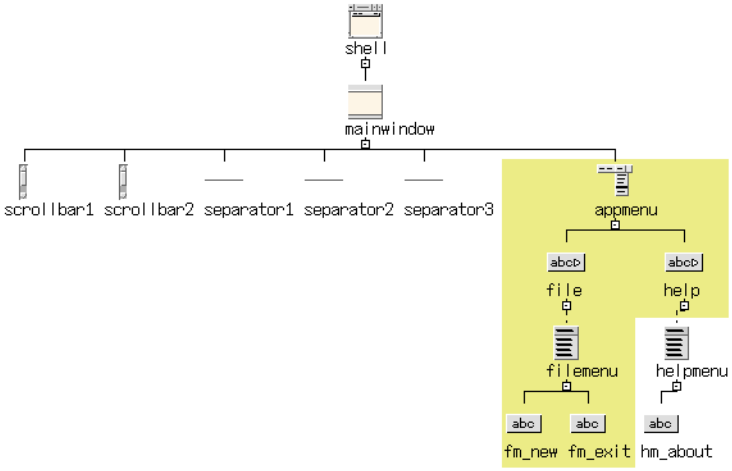


図 9-13 定義のインスタンスの拡張

4. *hm_about* ボタンのラベルを「About...」に設定します。

現在、ユーザーは定義内の他のウィジェットを修正することはできません。たとえば、*filemenu* にボタンを追加することはできません。しかし、定義からのサブクラスを作成すると、公開ウィジェットに加えて限定公開ウィジェットも修正することができます。その方法は、次の「派生クラスの作成」で説明します。

派生クラスの作成

定義に対応しているクラスのほとんどのメンバーは限定公開であるため、定義のインスタンスでそれらを呼び出すことはできません。これを解決するためには、次の 2 通りの方法があります。

- 元の定義を修正して、メンバーを公開にする
- インスタンスをクラスとして指定する (インスタンスクラスは定義クラスから派生しているため、限定公開メンバーへのアクセス権を持つ)

2 番目の方法は、カプセル化された構造をよりよく維持することができます。また、コールバック・メソッドを活用することもできます。

1. ウィジェット階層内でメニューバーウィジェットである *appmenu* を選択します。
2. コアリソースパネルの「コード生成」ページを表示します。
3. 「構造」オプションメニューから「C++ クラス」を選択して「適用」をクリックし、リソースパネルを閉じます。

   menubar ウィジェットは図 9-14 に示されるように、クラスとして指定されます。このクラスは定義に対応するクラスから派生されます。

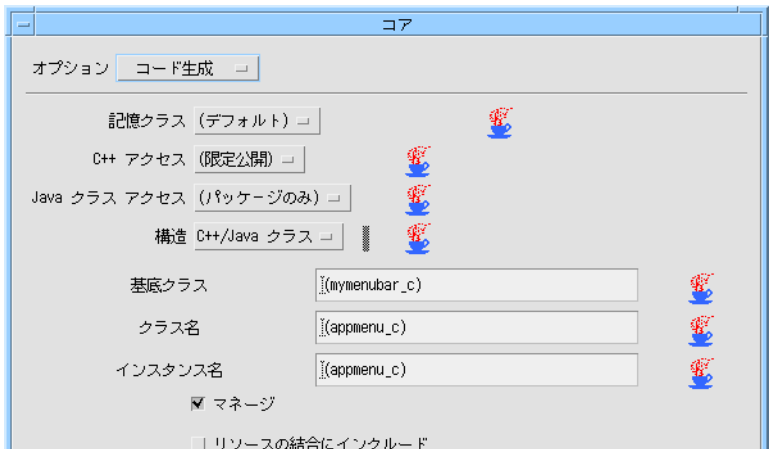


図 9-14 定義からの派生クラスの作成

クラスのメンバー関数は、 *mymenubar_c* の限定公開メンバーを呼び出すことができるため階層内の任意の場所にウィジェットを追加することができます。

4. ウィジェット階層内で *filemenu* ウィジェットを選択します。
5. メニューに 2 個のプッシュボタンを追加し、「Open...」および「Save...」というラベルを付けます。
6. 新しいボタンの変数名を *fm_open* および *fm_save* に設定します。
7. マウスボタン 1 を使用して、新しいボタンを図 9-15 に示す位置までドラッグします。

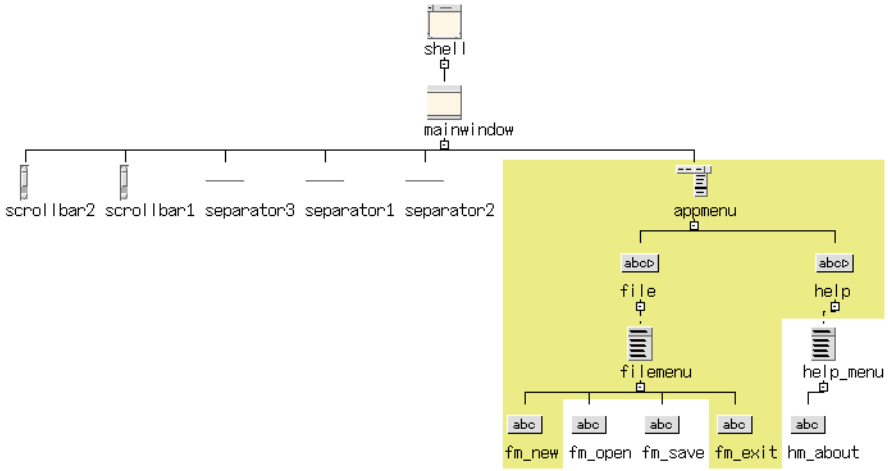


図 9-15 派生クラスの拡張

定義の順序は変更できません。新しいウィジェットを定義に追加することはできますが、すでに定義内の一部であるウィジェットを移動することはできません。

8. 階層内で *fm_open* を 選択します。
9. 「ウィジェット」メニューから「コールバック」を選択するか、ツールバーの「コールバック」ボタンを選択して、コールバックダイアログを表示します。
10. コールバックリストから「活性化 (activate)」を選択します。
11. 「メソッドの名前」フィールドに次のように指定します。

    `OnOpen`
12. 「追加」をクリックします。

13. 上述の手順を繰り返し、 *fm_save* に対しての「活性化 (activate)」コールバック・メソッドを *OnSave* に設定します。

14. 「コールバック」ダイアログを閉じます。

この方法は C 構造体に対しても有効です。Sun WorkShop Visual は定義の構造体の拡張である新しい構造体を生成します。

コールバック・メソッドの書き換え

メニューバーに対応するクラス *appmenu_c* は、継承される *OnNew()*、*OnExit()*、および *appmenu_c* 自体により定義される *OnOpen()*、*OnSave()* という 4 種類のコールバック・メソッドを持っています。しかし、継承されるメソッドは書き換えることができるため、派生クラスに基底クラスとは異なる動作を持たせることができます。

注 - この節の内容は C++ コードにのみ当てはまります。

---

1. ウィジェット階層でウィジェット *appmenu* を選択します。

2. 「ウィジェット」メニューから「メソッド宣言」を選択します。

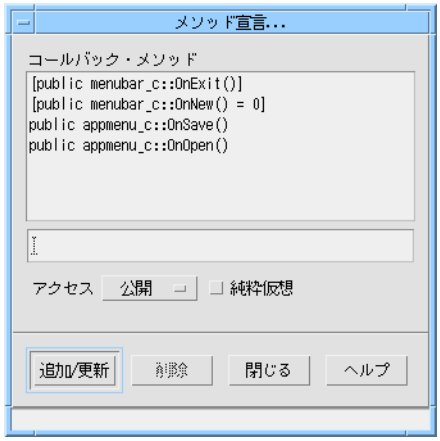


図 9-16 「メソッド宣言」ダイアログ

OnExit() と *OnNew()* が *menubar_c* から継承されるのに対し、*OnSave()* と *OnOpen()* は *appmenu_c* に局所的です。継承されたメソッドは角括弧 [ ] で囲まれて表示されます。

3. テキストフィールドに、次のように入力します。
   `OnExit`

4. 「純粋仮想」トグルがオフになっていることを確認して、「追加/更新」を押します。
   OnExit() が局所的メソッドのリストに追加されます。この時点で書き換えを行うことができます。

書き換えられたメソッドの実装

書き換えられたメソッドを実装するには、Sun WorkShop Visual が生成したスタブを完成させます。

1. 「コード生成」ダイアログを表示して、「言語」オプションが「C++」に設定されていることを確認します。

2. 「ディレクトリ」フィールドに、**`/u/mgs/TUTORIAL/cmd`** などのようにディレクトリ **`cmd`** へのフルパス名を指定します。
   これは、コードの生成先となるディレクトリを指定するための手順です。

3. 「コード」テキストフィールドに **`app.cpp`** と入力して、横にある「生成」トグルをオンにします。

4. 「スタブ」テキストフィールドに **`appS.cpp`** と入力して、横にある「生成」トグルをオンにします。

5. 「外部宣言」テキストフィールドに **`app.h`** と入力して、横にある「生成」トグルをオンにします。

6. 「メインプログラム」テキストフィールドに **`app.cpp`** と入力して、横にある「生成」トグルをオンにします。

7. 「メークファイル」テキストフィールドに **`Makefile`** と入力して、横にある「生成」トグルをオンにします。

8. 「メークファイル・オプション」ダイアログで「新規メークファイル」トグルと「メークファイル・テンプレート」トグルを両方オンにして、「了解」ボタンを押します。

9. 「コード」テキストボックスの横にある「オプション」ボタンを押します。

10. 「ヘッダーファイルをインクルード」テキストフィールドに **app.h** と入力し、トグルをオンにして「了解」ボタンを押します。

11. 「コード生成」ダイアログの最下部にある「オプション」ボタンを押して、「コードオプション」ダイアログのオプションを 図 9-17 のとおりに設定します。

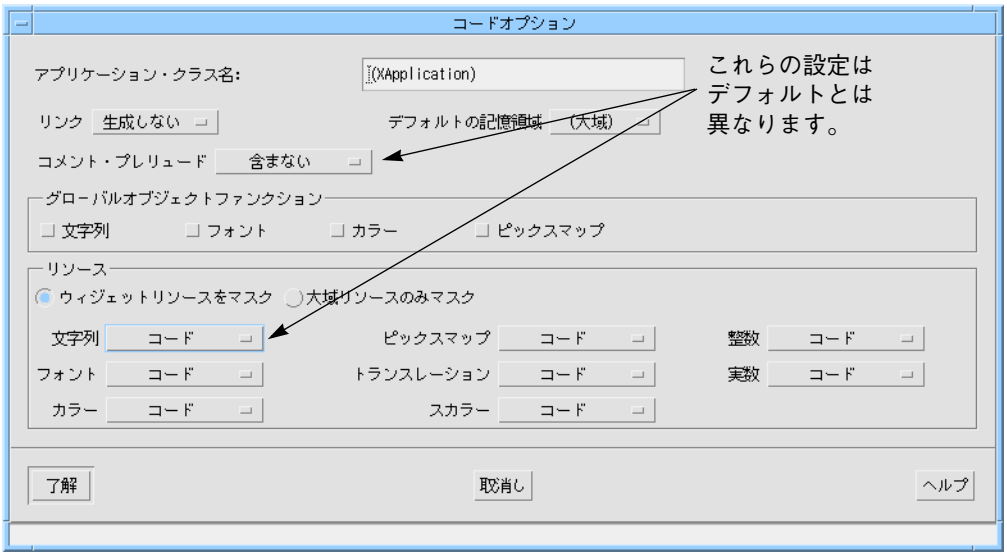


図 9-17 アプリケーションのコード生成

12. 「了解」ボタンを押します。

13. 「生成」ボタンを押し、「コード生成」ダイアログを閉じます。

14. スタブファイル *appS.cpp* を以下のように完成させます。

```
...
#include <iostream.h>
void
appmenu_c::OnExit (Widget w, XtPointer xt_call_data)
{
                    ...
                    if (modified)
```

```
                    XBell(XtDisplay(w), 100);
                else
                    exit(0);
}
void
appmenu_c::OnSave (Widget, XtPointer xt_call_data)
{
                ...
                cout << "Saving..." << endl;
                modified = FALSE;
}
void
appmenu_c::OnOpen (Widget, XtPointer xt_call_data)
{
                ...
                cout << "Opening..." << endl;
                modified = TRUE;
}
```

この *OnExit()* の実装は、定義における実装を書き換えます。関数 *XBell()* は X サーバー上でベルを鳴らします。*OnSave()* と *OnOpen()* は、適切なメッセージを出力し、*modified* フラグを更新するだけのダミーの関数です。

アプリケーションのメニューバーの機能は、現段階では以下のようになっています。

- クラスのコンストラクタにおいて、*modified* フラグは *TRUE* に初期設定されます。「Open...」により *TRUE* に、また「New」および「Save...」により *FALSE* に設定されます。
- 修正されたフラグが設定されていない場合は、「Exit」はアプリケーションを終了します。フラグが設定されている場合はベルを鳴らします。
- 「Open...」および「Save...」は、標準出力で情報メッセージを作成します。

15. メークファイルの中の「`MOTIFLIBS=`」で始まる行と「`CFLAGS=`」で始まる行を以下のように変更します。

```
MOTIFLIB=-lmenu -lXpm -lXm -lXt -lX11
CFLAGS=-I. ${XINCLUDES} -I${XPMDIR} -I../libmenu \
-L../libmenu
```

16. メークファイルを保存します。
17. 次にように入力して、プログラムを構築します。

    make
18. アプリケーションを実行し、メニューが意図したとおりに動作することを確認します。
19. アプリケーションをディレクトリ cmd で app.xd として保存します。

定義とリソースファイル

定義の構成要素であるウィジェットのリソース値は、明示的に設定することも、リソースファイルに指定することもできます。定義のリソースファイルを指定するとそのファイルは、その定義のインスタンスを含んでいるすべてのデザインのリソースファイルにインクルードされます。

ここまでの例では、すべてのリソースは明示的に設定されていました。以下の手順を使用して、リソースファイル中の文字列リソースを使用するようにメニューバー定義を再生成してみましょう。

1. libmenu ディレクトリのデザインファイル menubar.xd を開きます。
2. 「コード生成」ダイアログを表示して、「言語」メニューが「C++」に、「ディレクトリ」が libmenu ディレクトリに設定されていることを確認します。
3. 「コード生成」ダイアログから「コードオプション」ダイアログを表示して、「文字列」オプションメニューを「リソースファイル」に設定し、「了解」ボタンを押します。

   これにより、ボタン上のラベルなどの明示的に記述された文字列リソースが取り除かれます。

   ここで、文字列リソースを含んでいるリソースファイルを生成します。

4. 「コード生成」ダイアログで、「*X* リソース」テキストボックスに menubar.res と入力して、横にある「生成」トグルをオンにします。
5. 「コード」テキストボックスの横にある「生成」トグルがオンに、「メインプログラム」および「メークファイル」の横にある「生成」トグルがオフになっていることを確認します。
6. 「生成」ボタンを押します。
7. デザインを保存します。
8. libmenu ディレクトリで、次にように入力します。

   make clean

   古いオブジェクトファイルが削除されます。
9. 349 ページの手順 10 と同じ手順でライブラリを再度構築します。

定義の編集

前節では、文字列リソースがリソースファイルに保存されるように、定義の生成されたコードを変更しました。次に、*menubar* 定義を編集してリソースファイルを指定します。

1. パレットメニューから「定義を編集」を選択します。
2. 「定義を編集」ダイアログで、定義リストから *menubar* を選択します。
3. 「リソースファイル」フィールドに、次の行を入力します。

   ../libmenu/menubar.res
4. 「更新」をクリックします。

   定義への変更がユーザーの定義ファイル ($HOME/.xddefinitionsrc) に保存されます。

注 - 「定義を編集」ダイアログを使用すると、随時リソースファイルの指定を行うことができます。この手順を実行するために、元の *menubar.xd* デザインを読み込む必要はありません。

インスタンスと定義リソースファイル

定義に対してリソースファイルを指定すると、Sun WorkShop Visual は定義のインスタンスを含むすべてのデザインにそのファイルをインクルードします。リソースファイルを読み取る Xlib メカニズムは、指示文を解釈し、定義に対するリソースファイルの検索に使用します。以下の手順を使用して、この動作を *app.xd* アプリケーションで試してみます。

1. cmd ディレクトリのデザインファイル app.xd を開きます。
2. 「コード生成」ダイアログを表示して、「言語」メニューが「C++」に設定されていることを確認します。
3. 「*X* リソース」テキストボックスに **app.res** と入力して、横にある「生成」トグルをオンにします。
4. 「コード」および「メインプログラム」テキストボックスの横にある「生成」トグルを、両方オンにします。
5. 「スタブ」、「外部宣言」、および「メークファイル」の横にある「生成」トグルがすべてオフになっていることを確認します。
6. 「コード生成」ダイアログから「コードオプション」ダイアログを表示して、「文字列」オプションメニューを「リソースファイル」に設定し、「了解」ボタンを押します。
7. 「生成」ボタンを押します。

   アプリケーションに対する X リソースファイルには、以下の指令が含まれています。

   ```
   ! Generated by Sun WorkShop Visual
   #include "../libmenu/menubar.res"
   ```

   Xlib は、この *#include* 指令を、アプリケーションリソースファイルを含んでいるディレクトリへの相対的なパス名の指定として解釈します。詳細は、Xlib の資料を参照してください。

8. 次のように入力して、プログラムを構築します。

   ```
   make
   ```

9. 環境変数 XENVIRONMENT をリソースファイル名に設定します。

   設定するための構文は、使用するシェルの種類によって異なります。以下の例を参考にしてください。C シェルの場合は、次のように入力します。

```
setenv XENVIRONMENT app.res
```

Bourne シェルの場合は、次のように入力します。

```
XENVIRONMENT=app.res; export XENVIRONMENT
```

注 - この他にも、X に X リソースファイルを認識させる方法があります。詳細は、X ウィンドウシステムに関する資料を参照してください。参考資料については、付録 E「参考資料」をご覧ください。

---

10. アプリケーションを実行し、メニューが意図したとおりに動作することを確認します。
11. デザインを保存して、Sun WorkShop Visual を終了します。

第10章

Java 用のデザイン

はじめに

本章では、Sun WorkShop Visual を使用して、デザインから Java コードを生成する方法について説明します。本章は、以下のように構成されています。

1. 必要条件 (365 ページ)
   この節では、生成された Java コードを使用するために必要な条件について説明します。365 ページを参照してください。

2. Java を対象とする Sun WorkShop Visual の使い方 (366 ページ)
   この節では、Sun WorkShop Visual を使用して Java 準拠のデザインを作成する方法について説明します。

3. Java のバージョン (370 ページ)
   この節では、Java 1.0 または 1.1 用のコードを生成する方法、および各リソースがどのバージョンの Java に適用できるかを知る方法について説明します。

4. ウィジェット (374 ページ)
   この節では、Java で Motif デザインを生成する方法について説明します。

5. Java クラス用の新しいウィジェット (375 ページ)
   この節では、Motif には対応するものがない、Java レイアウトコンポーネント用の特別なレイアウトウィジェットについて説明します。

6. イベントモデル (382 ページ)
   この節では、Sun WorkShop Visual で、リスナーオブジェクトを備えた Java 1.1 イベントモデルを使用する方法について説明します。初めてのユーザーのために、簡単なチュートリアルを用意しています。

7. 生成ダイアログ (389 ページ)
   Java コードを生成できるよう、「コード生成」ダイアログが変更されました。

8. 生成されたコード (393 ページ)
   この節では、Java 用に生成されるコードについて説明します。

9. Sun WorkShop Visual のデザインを Visaj に移行 (402 ページ)
   この節では、Java 用のビジュアルアプリケーションビルダーである Visaj に取り込める形式でデザインを保存し、Java デザインを有効に活用する方法について説明します。

10. Motif ウィジェットから Java クラスへの対応付け - MWT ライブラリ (405 ページ)
    この節では、Motif ウィジェットを Java クラスに対応付けるクラスのライブラリについて説明します。

Java とは何か

Java は、とりわけ C や C++ に似た要素を持つプログラム言語であり、特にインターネット環境に対応するライブラリが用意されています。また、Java には高度な移植性、オブジェクト指向性、およびインタプリタ機能があります。Java はスレッド化されているだけでなく、自動記憶管理機能を備え、例外を使用します。これらの用語をご存知ない場合には、まず 1014 ページの「Java に関する資料」に記載する書籍をご参照ください。

生成される Java コード

Sun WorkShop Visual で生成したコードは、アプレットや通常のアプリケーションの形式にすることができます。アプレットとは、Web ページに埋め込まれる小さなアプリケーションであり、そのページがブラウズされたときに実行されます。

生成される Java コードには何の制限もありません。つまり、Java に対応しているプラットフォームであれば、どのようなプラットフォーム上でも、何度でも使用できます。Sun WorkShop Visual では、目的のプラットフォームとは無関係に Motif ベースのプラットフォーム上で設計できるため、生成されるコードには Motif ウィジェットを Java クラスに対応付けたクラスのライブラリがインポートされます。このライブラリは *MWT* と呼ばれ、Sun WorkShop Visual リリースに付属しています。Java では、クラスをパッケージにまとめることができます。MWT は `uk.co.ist.mwt` というパッケージに含まれています。

Motif ウィジェットから Java クラスへの対応付けの詳細については、405 ページの「Motif ウィジェットから Java クラスへの対応付け - MWT ライブラリ」を参照してください。

必要条件

生成された Java コードをコンパイルし、実行するには、Java コンパイラとインタプリタが必要です。アプレット用のコードを生成する場合は、アプレットを表示するためのアプレットビューアや HTML ブラウザも必要になります。

Sun WorkShop Visual から Java コードを生成するには、Sun WorkShop Visual で使用していたもの以外に、特別な環境変数を設定する必要はありません。しかし、生成されたコードをコンパイルし、実行するには、CLASSPATH 環境変数の設定が必要です。この変数は、PATH 環境変数と同様に、ディレクトリを列挙したものです。このディレクトリには、`$VISUROOT/lib/java_classes` が含まれていなければなりません (VISUROOT は Sun WorkShop Visual のインストールディレクトリです)。また、現作業ディレクトリ (.) など、アプリケーションで使用するクラス定義を生成したディレクトリも含める必要があります。さらに、Java コンパイラとインタプリタが置かれているディレクトリが PATH に含まれていることも確認してください。

Java を対象とする Sun WorkShop Visual の使い方

Sun WorkShop Visual を使用して Java コードを生成する方法には、Sun WorkShop Visual を使用して C++ や Microsoft Windows のコードを生成する方法と幾分似たところがあります。その理由は、Java が C++ のようなクラスベースの言語であるためです。クラスのインスタンス化、派生クラス、およびメソッドの処理方法は同じです。

注 - したがって、C++ を対象とする Sun WorkShop Visual の使い方を十分に理解していない場合には、必ず第 8 章を参照してください。

Java 準拠デザインの作成

「モジュール」メニューには、「Java 準拠」というトグルがあります。このトグルをオンに設定すると、Java コードの生成に適したデザインにするよう Sun WorkShop Visual に指示することになります。そのデザインを Java コードで再現できない場合には、Java 準拠不良ダイアログが表示されます。

Java 準拠不良ダイアログ

Java 準拠不良ダイアログを使用すると、Java コードで再現できないデザイン部分が表示されます。このダイアログを図 10-1 に示します。

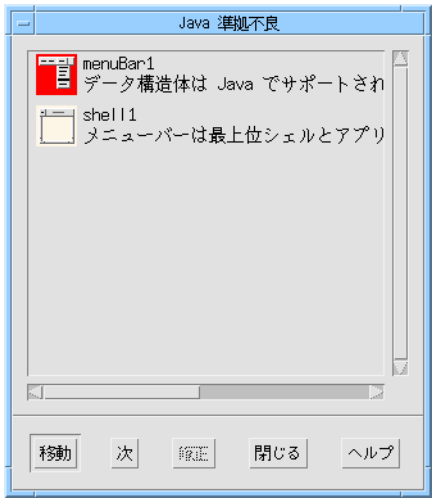


図 10-1 Java 準拠不良ダイアログ

ダイアログの下に並んでいるボタンの機能は次のとおりです。

- **移動**
  Java 準拠不良ダイアログで選択状態になっているウィジェットが、デザイン内で選択状態になります。ウィジェットが、現在表示されていないシェル部分にある場合、または階層内の折り畳まれたセクションにある場合には、ウィジェットが見えるように表示が変化します。
- **次**
  Java 準拠不良ダイアログでの選択部分が、ウィジェットリスト内の次の項目に移動します。
- **修正**
  ダイアログ内で選択されているウィジェットの準拠エラーの原因を修正できます。このボタンは、準拠エラーをこのように自動的に解決できる場合にのみ有効になります。
- **閉じる**
  ダイアログが閉じます。
- **ヘルプ**
  ダイアログに関するヘルプが表示されます。

Java コードの生成に関するデザイン上の制限

Java コードに変換できないユーザーインタフェース機能を次に示します。

1. シェルでない親を持つファイル選択ボックス。

2. 作業領域のあるファイル選択ボックス。

3. クラスであるファイル選択ボックス。

ファイル選択ボックスには親としてシェルが必要です。Java にはモード付きファイル選択ボックスしかありません。

4. メニューバーとシェル間のクラスであるウィジェット。

この制限があるのは、シェルがメニューバーを見つけられるようにするためです。

5. シェル内の複数のメニューバー。

Java では、シェルにはメニューバーが 1 つしかありません。

6. メニューのないカスケードボタン。

Java にはカスケードボタンに相当するものがないため (メニューはメニューバーに直接追加されます)、カスケードボタンを単独で使用しても意味がありません。

7. ポップアップメニューのあるデザイン。

Java はポップアップメニューに対応していません。

8. メニューバーの複数行文字列。

メニューバーでは複数行の文字列を使用できません。他の場所では使用できます。

9. クラスでないリンク元。

Java では、リンク元はクラスでなければなりません。

10. クラス内にないコールバック元。

これは現行バージョンの AWT イベントモデルの制限であり、このモデルの開発に伴って、今後のバージョンでは改善される可能性があります。

11. メニュー内のピックスマップ。

12. メニュー内のラベル。

13. アプレットのメインウィンドウにあるメニュー。

Java では、11〜13 の 3 項目は使用できません。

アプレットのデザインに関する制約

アプレットには、次に示すように、いくつかのデザイン上の制約があります。

1. アプレットには、親としてのアプリケーションシェルが必要です。
2. アプレット自身はクラスであってはなりませんが、その直接の子はクラスであることが必要です。
3. 1 のシェル以外は最上位シェルとしてください。
4. すべての最上位シェルをクラスにしてください。

クラスにするかどうかの判断

Java コンパイラ使用時には、1 つのクラスメソッドに使用できる変数が 64 個以下に制限されます。すぐにはピンとこないかもしれませんが、この制限はデザインに大きな影響を与えます。64 個というと多いように感じますが、実際にはすぐに使い切ってしまいます。この制限に到達した場合は、64 個までの変数を使用できるメソッドが増えるように、ウィジェットをクラスとして追加指定しなければなりません。デザインがこの制限を超えている場合は、Java コードの生成時に通知されます。

ウィジェットを簡単にクラスにするには、マウスの右ボタンを押してください。「クラス作成」など、便利なコマンドのあるメニューが表示されます。

コールバック宣言用の包含クラス

Java デザインのウィジェットには、他の種類のデザイン中のウィジェットと同様に、コールバックを与えることができます。コールバックが (382 ページの「イベントモデル」に記述するように) リスナーオブジェクトを定義する場合を除いて、コールバックはメソッドにしてください。メソッドは、ウィジェットを含む「包含クラス」で宣言されます。コールバックを与えられたウィジェット自身がクラスである場合には、コールバック・メソッドはウィジェットのクラス内で宣言されます。クラスでない場合には、Sun WorkShop Visual は階層を検索して、クラスである最も近い祖先を探して、そのウィジェットのクラスでメソッドを宣言します。

Java のバージョン

デフォルトでは、Sun WorkShop Visual は Java バージョン 1.1 をサポートします。Java 1.0 のコードを生成するには、生成オプションダイアログで適切なオプションを選択してください。これについては、370 ページの「生成されるコードの Java バージョン」を参照してください。

リソースとコールバックには、それらをサポートする Java のバージョンを示すマークが付けられます。詳細については 371 ページの「リソースとコールバックのバージョン記号」を参照してください。

Java 1.1 では、イベントモデル (ウィジェット間でのメッセージの受渡し方法) が変更されました。Sun WorkShop Visual は、リスナーオブジェクトを使用することで、このモデルを完全にサポートしています。詳細については 382 ページの「イベントモデル」を参照してください。

使用している Java のバージョンを調べるには、コマンド行で `-version` オプションスイッチを指定して Java を実行してください。こうすると、バージョンが標準出力に出力され、Java はすぐに終了します。

生成されるコードの Java バージョン

図 10-2 に示すように、Java オプションダイアログには新しいオプションメニューがあります。このメニューを使用すれば、生成されるコードの Java バージョンを選択できます。

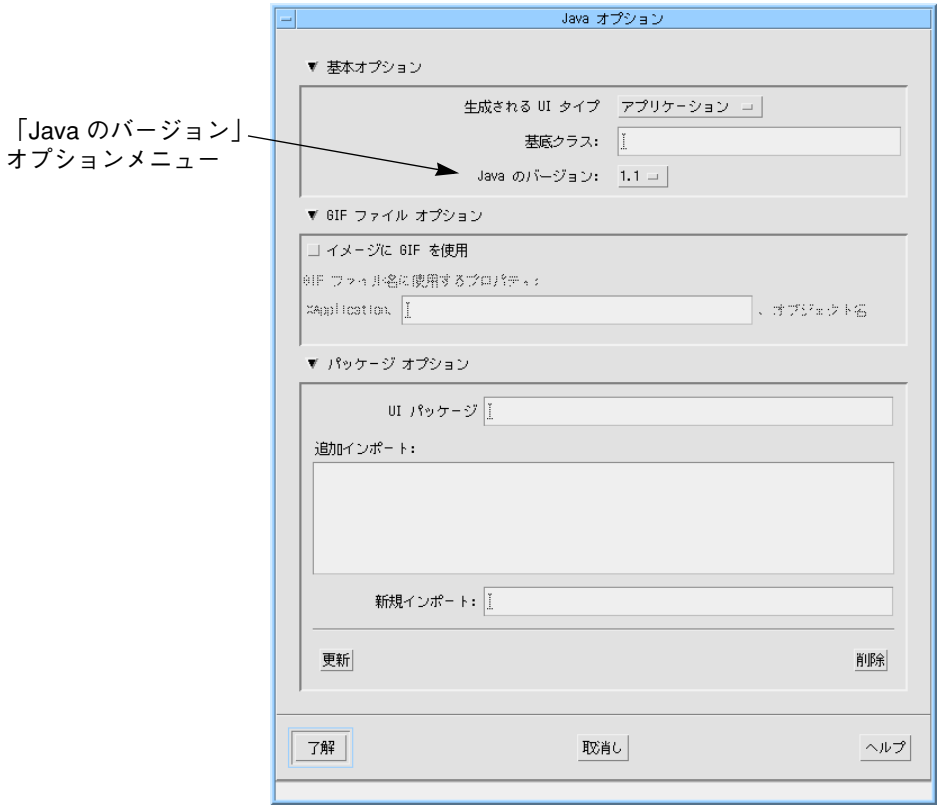


図 10-2 Java オプションダイアログ

Java オプションダイアログを表示するには、「コード生成」ダイアログの Java ページにある「Java オプション ...」というボタンを押してください。

コマンド行からのコード生成には、どちらのバージョンであれ、このダイアログで最後に選択されたバージョンが使用されます。選択が行われなかった場合には、デフォルトの「Java 1.1」が使用されます。

リソースとコールバックのバージョン記号

リソースパネルには、Java サポートを示すための 3 つの注釈記号があります。図 10-3 に示すように、「1.1」というテキストが表示されたコーヒーカップは、リソースが Java 1.1 でのみサポートされることを示します。

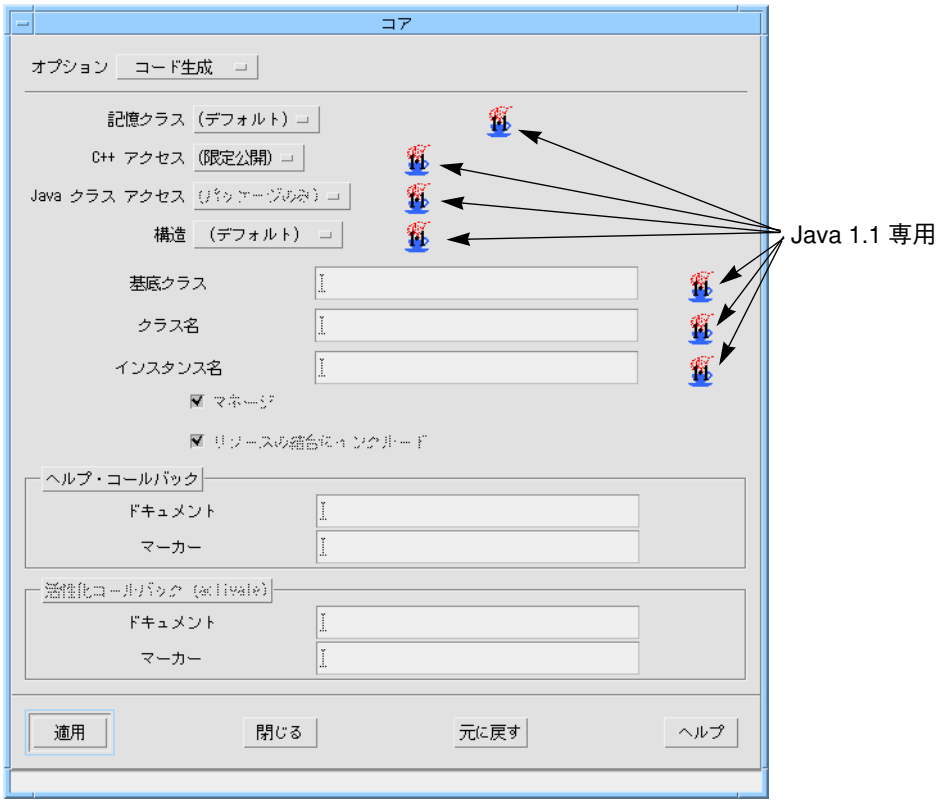


図 10-3　ポップアップメニューの Java 1.1 コアリソース

図 10-4 に示す「1.0」というテキストが表示されたコーヒーカップは、リソースが Java 1.0 でのみサポートされることを示します。テキストのないコーヒーカップは、リソースが両方のバージョンの JDK でサポートされることを示します。コーヒーカップがない場合は、リソースが Java でサポートされないことを示します。

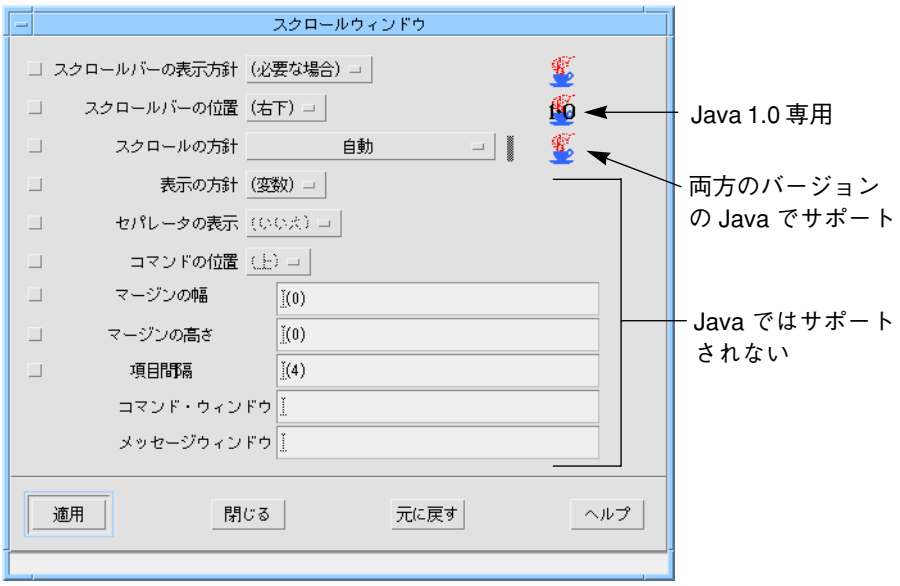


図 10-4 リソースパネルの注釈

コールバックダイアログ内のコールバックにも注釈記号が 3 つあります。コールバックが Java 1.0 でのみサポートされる場合は、コールバック名の後に文字列「J1.0」が表示されます。文字列「J1.1」は、コールバックが Java 1.1 でのみサポートされることを示します。「J」だけの場合には、コールバックが両方のバージョンでサポートされることを示します。リソースの場合と同様、注釈がない場合には、コールバックが Java では使用できないことを意味します。これらの注釈の例を図 10-5 に示します。

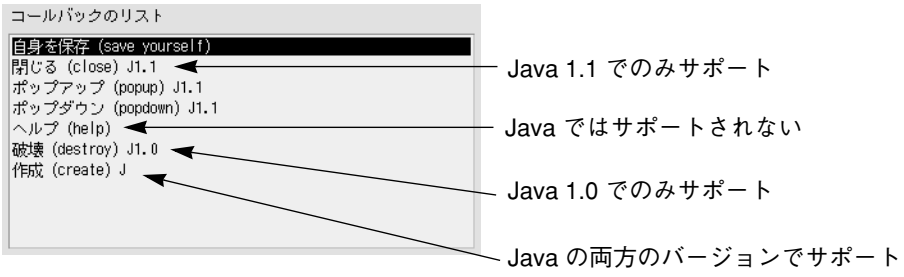


図 10-5 コールバックダイアログの注釈

ウィジェット

Java コンポーネントのセットと、Motif ウィジェットのセットは同じではありません。つまり、Motif ベースのプラットフォーム上で構築されたデザインから Java コードを生成する際の問題を解決するには、次のようにして互いの過不足を補う必要があります。

1. Sun WorkShop Visual で使用可能な Motif ウィジェットを、 Java クラスに対応付ける。

2. Motif ウィジェットには対応付けできない Java クラスを表現するためのウィジェットを追加する。

上記の 1 を行うために、Sun WorkShop Visual には MWT が用意されています。これは、Motif ウィジェットをまねるクラスのライブラリです。このライブラリは *MWT* と呼ばれ、`$VISUROOT/lib/java_classes` にあります。このライブラリ内のクラスは、`uk.co.ist.mwt` というパッケージにまとめられています。
Sun WorkShop Visual から生成されたすべてのコードは、このパッケージをインポートします。

内部的には、Sun WorkShop Visual がどのウィジェットがどの Java クラスに対応するかを決定します。この対応付けの詳細と、Java コードに関連するリソースについては、405 ページの「Motif ウィジェットから Java クラスへの対応付け - MWT ライブラリ」を参照してください。

上記の 2 を行うために、Motif には対応するクラスのない、以下の Java レイアウトクラスがウィジェットとして用意されています。

- カード
- フロー
- ボーダー
- グリッド
- グリッドバッグ

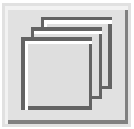
以下、これらについて説明します。

Java クラス用の新しいウィジェット

新しいウィジェットは 5 つあり、それぞれが個別の Java レイアウトマネージャを模倣しています。ウィジェットは Sun WorkShop Visual の一部であるため、パレットに表示されます。Sun WorkShop Visual のリリースには、これらのウィジェットのソースコードも付属しています。ソースコードは `$VISUROOT/src/java_widgets` にあります (`VISUROOT` は Sun WorkShop Visual のインストールディレクトリです)。このディレクトリには、これらのウィジェットの詳細が記述された `README` というファイルも入っています。

次からのサブセクションでは、新しい Java ウィジェットとそのリソースの使い方を 1 つずつ説明します。

カードウィジェット



カードウィジェットは、Java の CardLayout クラスの Motif 版です。すべての子ウィジェットを同じ大きさで配置し、最上位の子だけが見えるようにします。子のそれぞれに「ページ番号」を付けることができ、指定したページ番号の子を表示するようカードウィジェットに指示できます。

カードウィジェットの代表的な用途は、オプションメニューまたは「タブ」ボタンで制御する複数のページを持つダイアログです。

カードウィジェットに関連するリソースには、次の 3 つがあります。

- 水平間隔 (horizontalSpacing)
- 垂直間隔 (verticalSpacing)
- 現行ページ (currentPage)

間隔リソースは XtRDimension 型であり、カードの周囲の間隔を表します。

現行ページリソースでは、カードのどの子を「最上位」として表示するかを指定します。このリソースは XtRInt 型です。

カードウィジェットの子にはそれぞれ、コンストレイントリソースである「ページ番号 (pageNumber)」があります。そのため、子ウィジェットにページ番号を割り当てることができます。このリソースは XtRShort 型です。

特定の子を表示するには、カードウィジェットの現行ページリソースを、該当する子に対して指定しておいた「ページ番号」に設定します。

なお、ユーザーのコード内でこれを実行できるように、カードウィジェットのソースには、XdCardShowPage (Widget card, int page) という便利な関数が定義されています。

フローウィジェット

フローウィジェットは、Java の FlowLayout クラスの Motif 版です。子ウィジェットを行内で左から右に配置します。1 つの行に入らなくなると、次の行に移ります。つまり、ワープロで段落に単語を並べていくような形で配置が行われます。行については、左詰め、右詰め、中央揃えが可能です。

フローウィジェットに関連するリソースには、次の 3 つがあります。

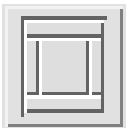
- 水平間隔 (horizontalSpacing)
- 垂直間隔 (verticalSpacing)
- 水平配置 (horizontalAlignment)

水平間隔リソースと垂直間隔リソースは、XtRDimension 型です。これらのリソースは、子ウィジェットの行間隔と列間隔を表します。

水平配置リソースは XdRAlignment 型であり、左詰め、右詰め、中央揃えのいずれかに設定できます。

これらのリソースの効果は、段落内に文字を配置することと同様です。

ボーダーウィジェット



ボーダーウィジェットは、Java の BorderLayout クラスの Motif 版です。最大 5 つの子を、図 10-6 に示すように、東、西、南、北、中央という 5 つの位置に割り当てることができます。ボーダーウィジェットは 6 つ以上の子を持つことができますが、余った子は単にデフォルト位置に追加されるだけで配置されません。

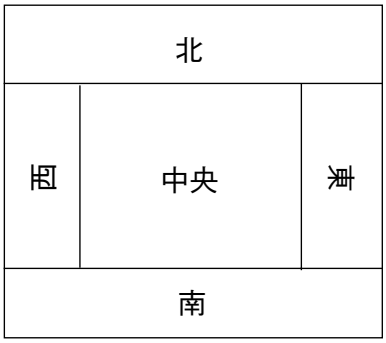


図 10-6 「ボーダーレイアウト」の概略図

北と南の子ウィジェットは、ボーダーウィジェットの幅全体に拡大されます。西と東の子ウィジェットは、北と南の子ウィジェット間の縦方向の間隔を満たすように収められます。中央の子ウィジェットは、残りの空間を満たします。このようにボーダーウィジェットは、MainWindow ウィジェットに取って代わる優れた手段です。MainWindow に関連して発生する多くの問題と不都合が解決します。なお、上記の 5 つの位置の中に空白のものがあってもかまいません。

ボーダーウィジェットに関連するリソースには、次の 2 つがあります。

- 水平間隔 (horizontalSpacing)
- 垂直間隔 (verticalSpacing)

いずれも XtRDimension 型であり、子ウィジェット間の間隔を表します。

ボーダーウィジェットの子にはそれぞれ、「ボーダー配置」というコンストレイントリソースがあります。これは XdRBorderAlignment 型であり、東、西、南、 北、中央のいずれかに設定できます。

グリッドウィジェット

グリッドウィジェットは、Java の GridLayout クラスの Motif 版です。すべての子を同じ大きさで、グリッドパターンに沿って配置します。グリッドの列数はリソースで指定しますが、行数は計算されます。グリッドウィジェットの大きさを変更すると、それに応じて子の大きさが変更されます。

グリッドウィジェットに関連するリソースには、次の 3 つがあります。

- 水平間隔 (horizontalSpacing)

- 垂直間隔 (verticalSpacing)
- 列数 (numColumns)

水平間隔リソースと垂直間隔リソースは XtRDimension 型であり、子の行間隔と列間隔を表します。

列数リソースは XmRShort 型であり、グリッドに表示する列数を表します。

グリッドバッグウィジェット

グリッドバッグウィジェットは、Java の GridBagLayout クラスの Motif 版です。グリッドバッグウィジェットは複雑ですが、Motif と Java のすべてのレイアウトウィジェットにおいて、最も高い柔軟性を持っています。グリッドバッグウィジェットの発想は、グリッドの行と列によって形成されたセルにウィジェットを配置するというものです。ただし、グリッドウィジェットの場合とは異なり、グリッドバッグウィジェットでは、行と列の高さと幅を変えられるため、さまざまなタイプのウィジェットを収めることができます。行と列の大きさはそれぞれ、その行で最大の高さを持つ子ウィジェット、またはその列で最大の幅を持つ子ウィジェットによって決まります。子ウィジェットは、複数の行または列に拡大することができます。また、セル内での位置を指定して、そこに配置することもできます。さらに、グリッドバッグの大きさを変更したときに、子ウィジェットの大きさをどのように変更するかも制御できます。詳細については、380 ページの「グリッドバッグの大きさの変更」を参照してください。

グリッドバッグに子を追加すると、既存の子ウィジェットの右側に配置されます。この位置を変更するには、子ウィジェットごとにコンストレイントダイアログを使用する必要があります。

グリッドバッグのコンストレイントダイアログ

グリッドバッグの子のコンストレイントダイアログを図 10-7 に示します。

図 10-7　グリッドバッグのコンストレイントダイアログ

コンストレイントダイアログの最上部には 2 つのオプションメニューが表示されます。セル配置リソースでは、配置対象の複数のセル内でウィジェットを配置したい位置を、「北」、「南」、「東」、「西」、「北東」、「北西」、「南東」、「南西」、「中央」のいずれかで指定します。セル充填リソースでは、配置対象の複数のセルを対象に、その複数のセル全体の大きさにウィジェットを拡大するかどうかを指定します。「水平」を選択すると、複数の列全体に拡大されます。「垂直」を選択すると、複数の行全体に拡大されます。「両方」を選択すると、複数の行と列全体に拡大されます。「なし」を選択すると、ウィジェットは元の大きさのままで、セル配置リソースで指定した位置に配置されます。

行と列のリソースを使用すると、選択したウィジェットをグリッド内のどこに表示するかを指定できます。一方、これと似た名前の行数と列数のリソースでは、そのウィジェットが占める行と列の数を指定します。この場合、ウィジェットは指定の行数と列数全体には拡大されません。その動作は、後述するセル充填リソースで制御されます。「行数」と「列数」のテキストボックスに入力できる値には、特別な値が 2 つあります。

1. 「列数」または「行数」のテキストボックスに 0 の値を入力すると、ウィジェットは現在の位置から端にある行または列までの大きさに拡大されます。

2. 「列」または「行」のテキストボックスに -1 の値を入力すると、新しいウィジェットは、既存のウィジェットの垂直方向 (列の場合) または水平方向 (行の場合) に隣接して配置されます。

行の重みリソースと列の重みリソースは、リサイズ方針を表します。これについては、380 ページの「グリッドバッグの大きさの変更」を参照してください。

「X 方向を埋める」リソースおよび「Y 方向を埋める」リソースでは、コンポーネントの各端部にそれぞれ水平および垂直に追加する内部パディングの量を指定します。

外部パディングリソース (右、左、下、上) では、選択したウィジェットの各端部に現れるマージンを指定します。

グリッドバッグの大きさの変更

グリッドバッグの子のコンストレイントダイアログには、行の重みリソースと列の重みリソースが表示されます。これらのリソースは、グリッドバッグの大きさを変更した場合に行と列の大きさの変更に影響を与えます。

グリッドバッグの子ウィジェットごとに行の重みと列の重みを指定することもできますが、グリッドバッグは、各行と各列における最大値を検索して、その数値を行や列の全体の計算に使用します。そのため、ウィジェットごとに重みを設定する必要はありません。各行と列で 1 つのウィジェットに設定すれば済みます。

行と列の重みを設定するとどのような効果があるのかをわかりやすく説明するために、以下に例を示します。

行の重みの例

まず、行の重みの例を取り上げます。行の重みを図 10-8 に示すように設定したとします。先頭の行の重みは 3 で最大、真ん中の行は 2、下の行は 1 です。これらの最大値の設定だけが意味をなし、それぞれの行の他の重みは無視されます。

グリッドバッグはすべての行の重みの合計を計算します。この例では 6 になります。この状態で、グリッドバッグを下方向に拡張したとします。増えたスペースは、次のように各行に配分されます。

- 重みが 3 の行には、増加スペースの半分が配分されます。3 は 6 の半分であるためです。
- 重みが 2 の行には、増加スペースの 3 分の 1 が配分されます。2 は 6 の 3 分の 1 であるためです。
- 重みが 1 の行には、増加スペースの残り、つまり 6 分の 1 が配分されます。1 は 6 の 6 分の 1 であるためです。

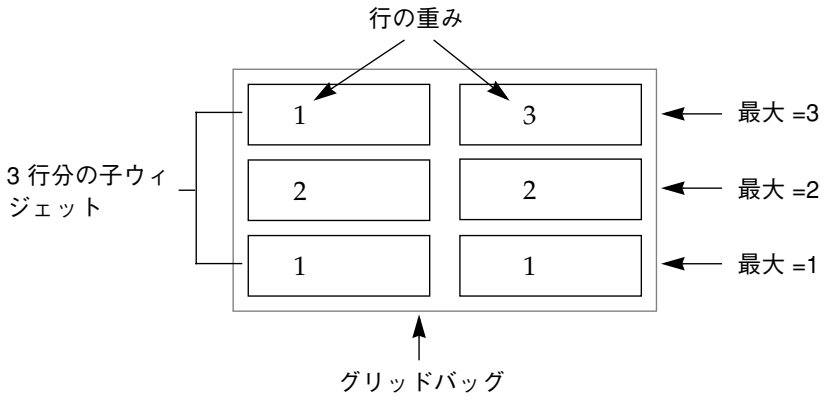


図 10-8 行の重み設定の例

列の重みの例

列に関するスペース配分の計算方法は、上述の行の場合と同じです。図 10-9 に示すような列の重み設定のウィジェットがあるとします。この図は、行の重みを列の重みに置き換えただけで、図 10-8 と同じ図です。列の重みの最大値は、左列が 2、右列が 3 です。グリッドバッグが注目するのは各列で最大の列の重み値だけであり、他の列の重み設定値は無視されます。最大の列の重み値の合計は 5 になります。増加スペースが生まれた場合には、これらの数値に基づいて配分計算が行われます。

この図のグリッドバッグを右方向に拡張したとします。増えたスペースは、次のように各列に配分されます。

- 列の重みが 2 の左列には、増加スペースの 5 分の 2 が配分されます。2 は 5 の 5 分の 2 であるためです。
- 列の重みが 3 の右列には、増加スペースの残り、つまり 5 分の 3 が配分されます。3 は 5 の 5 分の 3 であるためです。

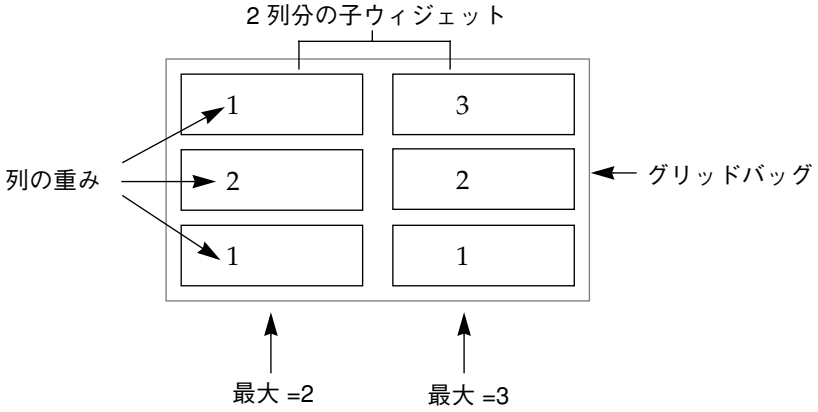


図 10-9 列の重み設定の例

グリッドバッグのリソース

グリッドバッグウィジェットにはリソースが 2 つあります。

- 水平間隔 (horizontalSpacing)
- 垂直間隔 (verticalSpacing)

いずれも XmRInt 型であり、行間隔と列間隔を表します。

イベントモデル

バージョン 1.0 と 1.1 の Java では、イベントモデル (オブジェクト間でメッセージを受け渡す方法) が変更されています。生成するバージョンとして「Java 1.1」を選択した場合には、新しいイベント処理の仕組みが使用されます。

どちらの仕組みを使用していても、(コールバック関数ではなく) コールバックメソッドを登録した場合には、イベントが生成されたときにそのメソッドが呼び出されます。この 2 つのバージョンの主な違いは、コールバック「関数」の処理方法です。

Java 1.0 のコードを生成した場合には、すべてのコールバック関数は無視され、生成されません。Java ではクラスを取り扱うため、メソッドの概念だけがサポートされています。

しかし、Java 1.1 のコードを生成するときには、コールバック関数は、Sun WorkShop Visual によって生成できる外部リスナーオブジェクトへの呼び出しとして扱われます。つまり、C++ から Java に移行するときにコード内で関数が使用されている場合には、Java 1.1 のコードを生成することで等価な動作を維持することができます。

注 - Java ファイル選択ボックスのボタンを押しても、イベントは生成されません。このため、生成された Java コードでは、ファイル選択ボックスからのリンクは機能しません。

Sun WorkShop Visual でのリスナーオブジェクト

Java 1.1 のコードを生成するとき、Sun WorkShop Visual は (メソッドではなく) 関数として定義されたコールバックをリスナーオブジェクトとして解釈します。リスナーオブジェクトは、ユーザーインタフェースのフロントエンドで発生するアクションと残りのアプリケーションとの間のインタフェースを実現するクラスのインスタンスです。これらは「補助」クラスや「アダプタ」クラスとも呼ばれています。

コールバック関数からリスナーオブジェクトを生成する方法は、次のリソースによって制御されます。

visu*javaWrapFunctions:true

デフォルトは「true」です。true の場合には、次の項で説明するように、Sun WorkShop Visual はラッパーオブジェクトを生成します。385 ページの「リスナーオブジェクトを自分で制御」で説明するように、リスナーオブジェクトのコードを自分で処理したい場合には、このリソースを「false」に設定してください。

コールバックオブジェクトの使用

Java 1.1 が生成されるとき、コールバック関数はリスナーオブジェクトへの参照として扱われます。コールバックオブジェクトは、個別のファイルに生成されます。このファイルの名前は関数名に「callback」を付加したものになります。たとえば、「myFunction」というコールバック関数がある場合には、`myFunctionCallback.java` という別個のファイルに以下のクラスが生成されます。

```
public class myFunctionCallback
{
```

```
        static myFunctionCallback me = null;

        public void doit( AWTEvent e, Object context)
        {
                // ここにコードを記述
        }

        static myFunctionCallback getInstance()
        {
                return new myFunctionCallback();
        }

        static myFunctionCallback get()
        {
                if (me == null)
                    me = getInstance();
                return me;
        }
}
```

このファイルは、まだ存在しない場合にだけ生成されます。このファイルは上書きされないため、追加したコードが失われる恐れはありません。

各コールバックのオブジェクトのインスタンスは 1 つだけです。複数の異なるウィジェットに同一のコールバック関数を追加した場合には、これらのウィジェットは、1 つのインスタンスを返す `get()` メソッドを呼び出すことによって、1 つのコールバックオブジェクトを共有することになります。このオブジェクトは、`get()` が最初に呼び出されたときに生成されます。

コールバックオブジェクトには、標準のコールバックスタブに似た `doit()` メソッドがあります。ユーザーのコードは、この `doit()` に追加します。`doit()` メソッドは、定義された関数を持つオブジェクトの包含するクラス内のリスナーオブジェクトから呼び出されます。コンテキストオブジェクトとイベント自身は、`doit()` メソッドに渡されます。次に例を示します。

```
{ // Java の大域的なコールバックオブジェクト
                  final myFunctionCallback myFunctionHandler =
                      myFunctionCallback.get();
                  if ( myFunctionHandler != null )
                      button1.addActionListener(
                          new ActionListener()  {
                              public void actionPerformed(ActionEvent
event)
                              {
                                  myFunctionHandler.doit(event,
                                      XApplication.this);
                              }
                          }
                      );
}
```

これはデフォルトの動作です。javaWrapFunctions リソースを「false」に変更すると、次の節で説明する動作が行われます。

リスナーオブジェクトを自分で制御

javaWrapFunctions リソースを「false」に設定した場合には、上述のコールバックオブジェクトのコードは生成されません。ボタンの活性化コールバックの場合は、包含するクラスには次のコードが生成されます。

```
if (myFunction != null)
                  button1.addActionListener( myFunction );
```

ここで、「button1」は、定義されたコールバック関数を持つウィジェットの名前であり、「myFunction」はコールバック関数の名前です。

生成されるコードは意図的に不完全なものとなっており、そのままではコンパイルできません。以下に示す 2 種類のコードを追加する必要があります。

1. リスナーオブジェクト「myFunction」の宣言
2. 「myFunction」がインスタンスとなるクラスの定義

最初のコードは比較的簡単です。次のようになります。

```
EventListener myFunction = new EventListener();
```

このコードでは、定義しようとするクラスである「EventListener」のインスタンスとして、リスナーオブジェクト「myFunction」を宣言します。

2 番目のコードはもう少し複雑です。

```
class EventListener implements TextListener, ActionListener {
public void actionPerformed(ActionEvent e) {
        Object source = e.getSource();
        if (source == button1)
            // ここに動作を行うためのコードを追加
                }
}
```

これは新しい EventListener クラスの定義です。button1 でイベントが発生すると、メソッド「actionPerformed」が呼び出されます。このようなクラスを「公開」アクセス機能を持つ専用のファイルに定義すれば、どのコンポーネントからも使用できるようになります。次に、コンポーネント(この場合は「button1」)にアクセスできることを確認する必要があります。

注 - コールバックダイアログで Java 準拠とマークされたコールバックだけが生成されます。コールバックリストに Java マークのないコールバックは、Java コードの生成時に無視されます。

リスナーオブジェクトとしての X イベント

ウィジェットを選択した状態でウィジェットメニューから「イベントハンドラ ...」を選択すると、237 ページの「イベントハンドラ」で説明するイベントハンドラダイアログが表示されます。Java 準拠のイベントマスクを指定した場合には、ここで追加される手続きは、Java コードが生成されるときにコールバック関数と同じに扱われます。Java 準拠でないイベントマスクは、Java コードが生成されるときに無視されます。

Sun WorkShop Visual では、コールバック関数の場合と同様に、手続きに対して与えられた名前を持つリスナーオブジェクトがあるものと想定します。Sun WorkShop Visual は、それぞれの X イベントマスクに対応して、Java イベントタイプごとにリスナーオブジェクトを登録します。たとえば、図 10-10 では、以下のイベントマスクを持つイベントハンドラを示します。

```
PointerMotionMask | KeyReleaseMask | EnterWindowMask
```

このイベントマスクは、「MyButton」というウィジェット上の「exampleHandler」という手続きに対するものです。

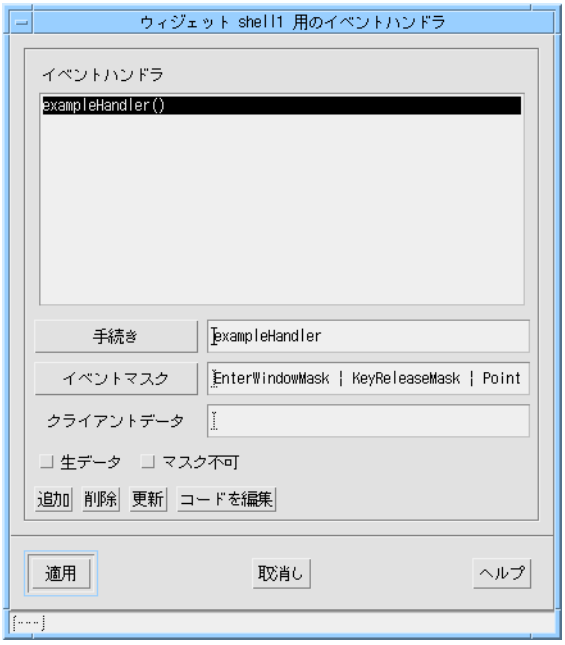


図 10-10 イベントハンドラの例

このようなイベントハンドラに対して、Sun WorkShop Visual は以下の Java コードを生成します。

```
if (MyHandler != null)
     MyButton.addMouseMotionListener(myHandler);
if (MyHandler != null)
     MyButton.addKeyListener(myHandler);
if (MyHandler != null)
```

```
MyButton.addMouseListener(myHandler);
```

次に、生成されたコードファイルに行を追加することによって、myHandler は他の場所で定義されたクラスのインスタンスであることを宣言する必要があります。次に例を示します。

```
MyHandler myHandler = new MyHandler();
```

このコードをコンパイルするには、クラス MyHandler を定義する必要があります。このクラスは次のシグニチャーを持つことになります (必要ならば「公開」にすることもできます)。

```
class MyHandler implements MouseMotionListener, KeyListener,
                                                MouseListener {
```

MyHandler クラスには、3 つのリスナークラスのすべてのメソッドの定義が含まれます。

コールバック・メソッドのシグニチャーに関するバージョンの非互換性

Java 1.0 のコールバック・メソッドのシグニチャーを次に示します。

```
void foo(Event x)
```

Java 1.1 の場合には、次のシグニチャーになります。

```
void foo(AWTEvent x)
```

その結果、既存の Java 1.0 ファイルの先頭に新しい Java 1.1 ファイルを生成した場合には、すべてのコールバック・メソッドに対して新しいスタブが生成されます。古いスタブはもちろん残されますが、もはや呼び出されるメソッドではありません。デザインを Java 1.0 から Java 1.1 に移行するときには、古いメソッドのコードを新しいメソッドに移動して、イベントオブジェクトを使用するすべてのコードを更新してください。

生成ダイアログ

「言語」オプションメニューから Java を選択した場合、その他の言語の場合とはまったく異なる「コード生成」ダイアログが表示されます。図 10-11 に示したように、生成可能なファイルのリストが表示されます。

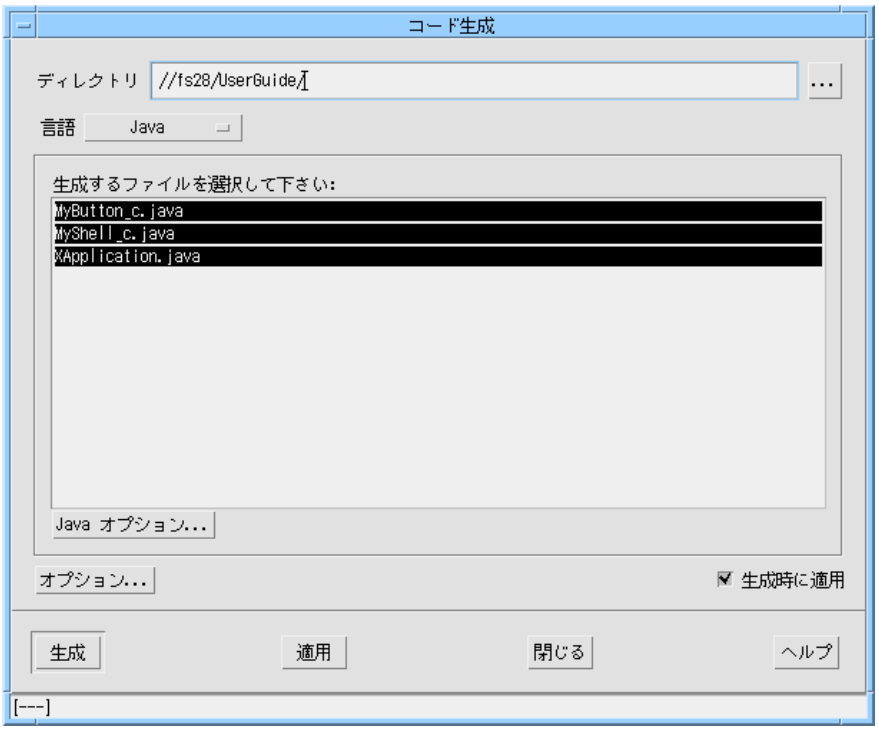


図 10-11 Java 用の「コード生成」ダイアログ

「ディレクトリ」というテキストボックスに、ファイルが生成される場所が示されています。この生成先を変更するには、ディレクトリ名をテキストボックスに直接入力するか、「...」ボタンを押してファイル選択ダイアログを使用します。

生成されるのは、選択状態 (強調表示) のファイルのみです。ファイルを選択または選択解除したいときは、該当ファイルをクリックします。複数のファイルを一度に選択するときは Shift キーを併用し、ファイルを個別に選択リストに追加するときは Control キーを併用します。このダイアログに示されるファイルの詳細については、

393 ページの「Java のファイル」を参照してください。生成されたコードをコンパイルし、実行する方法については、400 ページの「生成されたファイルの使用」を参照してください。

「Java オプション」というボタンを選択すると、図 10-12 のような、Java コードの生成に関するオプションが並んだダイアログが表示されます。「オプション」ボタンを押したときに表示されるコードオプションダイアログの詳細については、254 ページの「コード生成オプション」を参照してください。

図 10-12 Java 生成オプションダイアログ

Java 生成オプションダイアログ

Java 生成オプションダイアログを使用すると、Java コードの生成に関する次の 5 点を制御できます。

1. アプリケーションとアプレットのどちらを作成するか
2. アプリケーションの基本クラスの名前
3. 生成する Java コードのバージョン
4. `gif` ファイルを使用するデザインにするかどうか
5. クラスをパッケージにまとめる場合は、作成するパッケージの名前

アプリケーションかアプレットかの選択

デフォルトでは、デザインからアプリケーションが生成されます。アプレットを生成したい場合は、「オプション」メニューから該当するオプションを選択してください。「アプレット」を選択した場合、生成されるメインコードファイルには、アプリケーションを HTML ブラウザ内で実行できるようにするためのコードも生成されます。

基本クラスの変更

メインコードファイルには、デザインにリンクしている 1 つのクラスが入っています。このクラスの名前は、コードオプションダイアログでアプリケーションクラスとして指定した名前 (またはデフォルトの XApplication) です。これを別のクラスから拡張するには、「基本クラス」というテキストボックスに名前を入力してください。アプリケーションを生成することにした場合に、基本クラスを指定しなかったときは、アプリケーションクラスは拡張されません。アプレットを生成することにした場合に、基本クラスを指定していなかったときは、デフォルトでは「Applet」という Java クラスが使用されます。

GIF の使用

ダイアログの GIF オプションの領域は、アプリケーションとアプレットのどちらを生成するかによって、形式が異なります。ただし、いずれの場合も、セクションの最上部には「イメージに GIF を使用」というトグルがあります。このトグルを選択しなかった場合は、セクションの残りは無効になります。アプリケーションを生成する場

合にこのトグルを選択すると、ピックスマップオブジェクト用に生成されるコードでは、すべてのピックスマップはそれぞれ別の gif ファイルに入るものと見なされます。一方、「イメージに GIF を使用」トグルを設定しなかった場合は、ピックスマップオブジェクトは、<ApplicationClassName>PixmapObjects.java というファイルに整数の配列として格納されます。「イメージに GIF を使用」トグルを設定した場合は、このファイルには GIF ファイルを読み込むためのコードが入ります。また、この場合は、Java コードファイルのリストにさらに 1 つのファイルが追加されます。追加されるのは、<ApplicationClassName>Properties というプロパティファイルです。このファイルでは、gif が次のように表されています。

```
<ApplicationClassName>.<ApplicationImagesName>.<PixmapObjectName>
```

ここで、「ApplicationImagesName」は、このダイアログの「GIF オプション」領域にあるテキストフィールドに入力した値です。たとえば、デフォルトのアプリケーションクラス名である XApplication を使用し、「TreeIcon」というピックスマップオブジェクトを作成し、テキストフィールドに「MyApplicationImages」という値を入力した場合には、TreeIcon ピックスマップのプロパティ名は次のようになります。

```
XApplication.MyApplicationImages.TreeIcon
```

また、ピックスマップオブジェクトが「LeafIcon」という別のオブジェクトである場合には、プロパティ名は次のようになります。

```
XApplication.MyApplicationImages.LeafIcon
```

アプレットを生成する場合に「イメージに GIF を使用」トグルを設定していると、ダイアログの「GIF オプション」セクションには、GIF イメージがあるディレクトリを (ドキュメントのベースディレクトリに相対的に) 指定するためのテキストボックスが表示されます。

アプリケーションまたはアプレットで gif を使用したい場合は、gif ファイルを作成する必要があります。この作業を支援するため、Sun WorkShop Visual には、すべてのピックスマップオブジェクトを対象にして XPM ファイルを一度に作成する機能があります。この機能を使用するには、「コード生成」ダイアログの「ピックスマップ」ページに進んでください。このページには、デザイン内のピックスマップオブジェクトのリストが表示されています。

「生成」を押すと、オブジェクトごとに <PixmapObjectName>.xpm という名前の XPM ファイルが作成されます。作成の際には、XPM2 形式と XPM3 形式のどちらかを選択できます。XPM ファイルの作成後、変換ユーティリティを使用して、それらのファイルを GIF に変換する必要があります。変換ユーティリティとしては、*pbmplus* や *netpbm* など、いくつかのユーティリティが無料で配布されています。

パッケージの指定

Java 生成オプションダイアログの一番下には、パッケージの名前を指定するためのテキスト領域があります。パッケージとは、使用目的に合わせてクラスをグループ化したものです。クラスをパッケージとして生成することにした場合は、パッケージをインポートするための文がメインコードファイルに追加されます。

注 - パッケージの構成と命名には一定の規則があります。詳細については Java のマニュアルを参照してください。

生成されたコード

Java のコードは、C や C++ のコードとは構造が若干異なっています。Java のコードは必ずクラス内になければなりません。また、単独のクラスはそれぞれ専用のファイルに格納しなければなりません。`main` 関数はアプリケーションクラスのメソッドです。Java インタプリタはこのメソッドを見つけ出して、アプリケーションを開始します。

Java のファイル

「コード生成」ダイアログで言語設定のメニューから「Java」を選択すると、選択可能なファイルのリストが表示されます。このリストは、デザインの構造に応じて次のいずれかになります (山括弧で囲んだ部分は、デザインで使用している名前に応じて変わります)。

1. `<ApplicationClassName>.java`
   アプリケーションを定義しているクラスである、アプリケーション用の主要コードファイルです。このクラスの 1 つのメソッドは `main` 関数です。ファイル名は、アプリケーションクラスの名前になります。デフォルトでは「XApplication」です。この名前は、「コードオプション」ダイアログで変更できます。

2. `<ApplicationClassName>Links.java`
   ウィジェット間でダイナミックリンクを実現するためのコードが入ったファイルです。リンクを設定しない場合は、このファイルは不要です。

3. <ApplicationClassName>PixmapObjects.java
   ピックスマップオブジェクトを指定している場合は、それに対応するファイルが表示されます。

4. <ApplicationClassName>FontObjects.java
   フォントオブジェクトを指定している場合は、それに対応するファイルが表示されます。

5. <ApplicationClassName>ColourObjects.java
   カラーオブジェクトを指定している場合は、それに対応するファイルが表示されます。

6. <ApplicationClassName>StringObjects.java
   文字列オブジェクトを指定している場合は、それに対応するファイルが表示されます。

7. <WidgetClassName>_c.java
   (コアリソースパネルで) Java クラスと指定したウィジェットには、クラス定義が入った専用のコードファイルが必要です。そのため、デザイン内のクラスと同数のコードファイルが表示されるはずです。何らかの理由でデザイン内の特定のウィジェットを無視する必要がある場合を除いて、これらのファイルは生成しなければなりません。

コマンド行でのコード生成

コマンド行から Java コードを生成する場合に備えて、Sun WorkShop Visual には -J というフラグが追加されています。

コード例

図 10-13 に示す階層とデザインに対して、Sun WorkShop Visual が生成したコードを次に示します。このコードを自分で生成するには、以下の手順に従ってください。

1. シェルウィジェットを選択します。
2. シェルに「MyShell」という名前を付けます。
3. 「MyShell」をアプリケーションシェルにします。
   これは、シェルのリソースパネル内で行います。
4. シェルにダイアログテンプレートを追加します。

5. ダイアログテンプレートに「MyMessageBox」という名前を付けます。
6. 3 つのプッシュボタンウィジェットと 1 つのフォームウィジェットを選択します。
7. フォームの子としてスクロールテキストを選択します。
8. スクロールテキストに「MyScrolledText」という名前を付けます。
9. スクロールテキストを選択し、マウスの右ボタンを押します。表示されたメニューから「クラス作成」を選択します。

   これは、選択したウィジェットの構造を簡単に変更する方法です。このようにする代わりに、コアリソースパネルを表示して「コード生成」ページに進み、「構造」オプションメニューを「C++/Java クラス」に変更してもかまいません。

   これで、図 10-13 に示す階層ができたはずです。

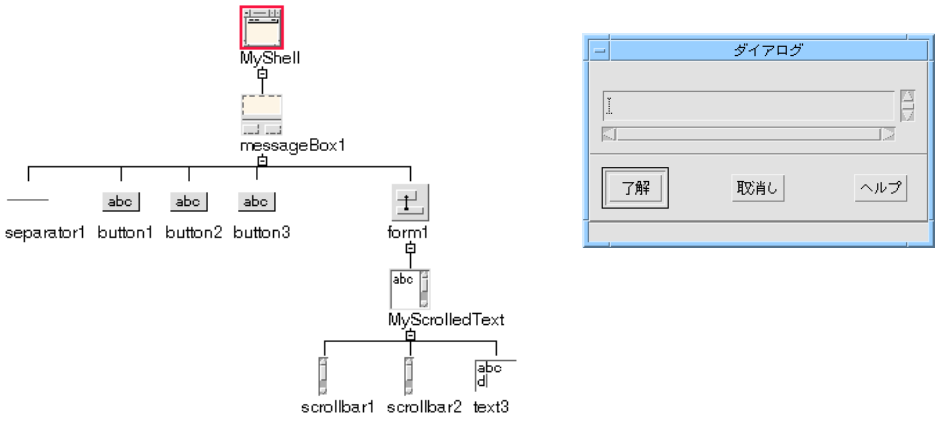


図 10-13 コード例の階層とデザイン

この簡単なデザインには、生成されるコードを示すという目的しかないため、リソースは設定していません。「コード生成」ダイアログで「Java」を選択すると、図 10-14 に示すファイルが表示されます。

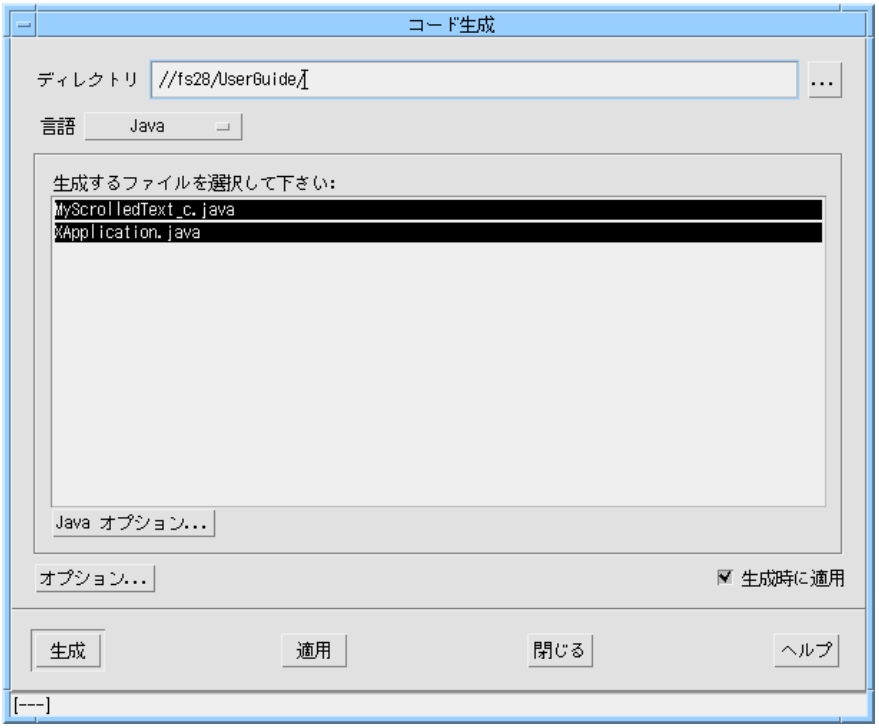


図 10-14 コード例の生成ダイアログ

アプリケーションクラス名を指定していないため、ファイル名にはデフォルト (XApplication) が使用されます。アプリケーションの開始点となる `main` メソッドを含む、`XApplication.java` ファイルが必要になります。また、スクロールテキストウィジェットのクラス定義である `MyScrolledText_c.java` ファイルも必要です。そのため、次のような作業を行います。

10. XApplication.java と MyScrolledText_c.java の両方が選択状態になっていることを確認します。
11. ダイアログの上部にある「ディレクトリ」を確認します。このディレクトリがファイルの生成先となるので、必要に応じて変更します。
12. 「生成」を押します。

ウィジェットをクラスにする時期については、369 ページの「クラスにするかどうかの判断」を参照してください。

`XApplication.java` ファイルのコードは、次のとおりです。

```
/*
** Generated by Sun WorkShop Visual/Java(tm) Edition
*/
import java.awt.*;
import uk.co.ist.mwt.*;
/*
** Sun WorkShop Visual/Java(tm) Edition - main program
*/
/**
 * A class representing the user interface specified by the entire
 * design.
*/
public class XApplication  {
                protected MyScrolledText_c MyScrolledText;
                protected Panel MyMessageBox;
                DlogTemplateLayout MyMessageBoxLayout = new
                    DlogTemplateLayout();
                protected Frame MyShell;
                public XApplication()  {
                    Button button3;
                    Button button4;
                    Button button5;
                    Panel form2;
                    FormLayoutManager form2Layout = new
                        FormLayoutManager();
                    MyShell = new Frame();
                    MyMessageBox = new Panel();
                    MyShell.add( "Center", MyMessageBox );
                    MyMessageBox.setLayout( MyMessageBoxLayout );
                    button3 = new Button( "button3" );
```

```
                        MyMessageBox.add( button3 );
                        button4 = new Button( "button4" );
                        MyMessageBox.add( button4 );
                        button5 = new Button( "button5" );
                        MyMessageBox.add( button5 );
                        form2 = new Panel();
                        MyMessageBox.add( form2 );
                        form2.setLayout( form2Layout );
                        MyScrolledText = new MyScrolledText_c();
                        form2.add( MyScrolledText );
                        FormLayoutConstraints MyScrolledTextConstraints
=
                                new FormLayoutConstraints();
                        form2Layout.constrain( MyScrolledText,
                                MyScrolledTextConstraints );
                        MyShell.pack();
                        MyShell.layout();
                        MyMessageBox.layout();
                        form2.layout();
                        MyScrolledText.layout();
                        MyShell.show();

        }

        public static void main( String args[] )  {
                        new XApplication();
        }

}  /* ...class XApplication (main application class) */
/*
```

```
** (end of Sun WorkShop Visual/Java(tm) Edition generated main
program)
*/
/*
** (end of Sun WorkShop Visual/Java(tm) Edition generated code)
*/
```

Now, this is MyScrolledText_c.java:

```
/*
** Generated by Sun WorkShop Visual/Java(tm) Edition
*/
import java.awt.*;
import uk.co.ist.mwt.*;
/**
 * A class represented by the widget MyScrolledText_c in the design
file
 */
// Private: classHeader Sun WorkShop Visual-generated code - do not edit >>>
class MyScrolledText_c extends TextArea  {
// Private: classHeader <<< Sun WorkShop Visual-generated code ends.

// Private: instanceVars Sun WorkShop Visual-generated code-do not edit >>>
// Private: instanceVars <<< Sun WorkShop Visual-generated code ends.
/**
* The constructor method for MyScrolledText_c
*/
// Private: constructor Sun WorkShop Visual-generated code - do not edit >>>
                public MyScrolledText_c()  {

                }
// Private: constructor <<< Sun WorkShop Visual-generated code ends.
// Private: endClass Sun WorkShop Visual-generated code - do not edit >>>
```

```
}
// Private: endClass <<< Sun WorkShop Visual-generated code ends.
/*
** (end of Sun WorkShop Visual/Java(tm) Edition generated code)
*/
```

これらのファイルは、必要に応じて修正して自由に使用できます。ただし、MWT を組み込む必要があります。次の行を加えると、Sun WorkShop Visual に付属のすべての Java クラスライブラリが Java プログラムにインポートされます。

```
import uk.co.ist.mwt.*;
```

つまり、生成されたファイルを別のプラットフォームに持ち込んで、結果のアプリケーションをそこで実行したい場合には、Java クラスライブラリの MWT もインポートする必要があるということです。

生成されたファイルの使用

生成した Java コードファイルは、Java に対応しているプラットフォームで自由に使用できます。ただし、そのプラットフォームに *MWT* ライブラリがあることが前提です。生成されたファイルから Java アプリケーションを作成するには、以下の手順に従ってください。

1. CLASSPATH 環境変数が、365 ページの「必要条件」に示したとおりに設定してあることを確認します。
2. 生成したクラスが入っているディレクトリが CLASSPATH のリストに含まれていることを確認します。

   このディレクトリは、現作業ディレクトリでもかまいません。その場合は、CLASSPATH リストに . (ピリオド) が含まれていることを確認してください。
3. Java コンパイラが入っているディレクトリが PATH 環境変数に含まれていることを確認します。
4. **javac <ApplicationName>.java** と入力します。

   これで、<ApplicationName> に拡張子「.class」が付いたファイルが生成されます。
5. **java <ApplicationName>** と入力します。

   Java アプリケーションが画面に表示されます。

コールバックスタブ

Java コードファイル内のコードはすべて、クラス内になければなりません。つまり、所属するクラスの一部としてコールバックスタブが生成されています。他の言語にあるような、別ファイルのコールバックファイルはありません。生成コードに追加されたすべてのコードは、それが保護のためのコメントの外側にある限りはそのまま残されるため、これによって不都合が生じることはありません。この件については、401 ページの「編集内容の保持」で説明します。

編集内容の保持

前述のコード例には、次のような部分がありました。

```
/**
* The constructor method for MyScrolledText_c
*/
// Private: constructor Sun WorkShop Visual-generated code - do not edit >>>
                    public MyScrolledText_c()  {


                    }
// Private: constructor <<< Sun WorkShop Visual-generated code ends.
```

この部分では、追加コードを保持する方法を示すために、多数のコメントを使用しています。最初のコメントは、コードをわかりやすくするために加えているだけです。一方、2 番目と 3 番目のコメントには特別な意味があります。
「`// Private: ... - do not edit`」で始まるコメントは、次に続く部分をそのまま残す必要があることを表しています。対応する「`// Private: ... ends`」は、そのままで残す部分の終わりを示しています。

この特殊なコメントの外側にコードやコメントを加えた場合は、ファイルを生成し直すとそのままで残されます。たとえば、コールバック・メソッドにコードを追加した場合に、そのコードが必ず残されるようにできます。

Javadoc

Java コンパイラとインタプリタには、javadoc という自動文書化ツールも含まれています。.java ファイルを発見すると、javadoc は、宣言および「/**」で始まるコメントを解析して、クラス継承のつながり、クラス内のすべての公開フィールド、および特殊コメントに含まれた追加情報を集めた HTML ページを作成します。特殊コメントの仕組みによって、javadoc が作成する文書に必要な情報が追加されます。Sun WorkShop Visual によって生成されるコードには、javadoc の基本的な情報がいくつか含まれています。

Sun WorkShop Visual のデザインを Visaj に移行

注 - 現段階では Visaj は日本語化されていません。そのため、開発できるアプリケーションは英語版のみになります。

もちろん、Sun WorkShop Visual の Java コード生成機能を使用してデザインを Java に変換することもできます。この方法では、Java で使用できる共通のコードベース (Motif では C/C++、Microsoft Windows では MFC) を維持することができます。しかし、Java だけを使用する場合には、IST の Java 用ビジュアルアプリケーションビルダーである Visaj に移行することをお勧めします。製品の詳細については、以下の URL を参照してください。

```
http://www.ist.co.uk
```

デザインを Sun WorkShop Visual から Visaj に移行するには、以下の手順を実行するだけで済みます。

- Sun WorkShop Visual のファイル保存ダイアログを使用して、デザインを特別な形式で保存する
- Visaj の「Import (インポート)」ダイアログを使用して、ファイルを読み込む

この節では、Sun WorkShop Visual と Visaj の両方から移行を行う方法について説明します。この節の最後には、障害対処のヒントを掲載しています。

Sun WorkShop Visual から

Visaj でファイルを保存するには、「ファイル」メニューから「別名保存」を選択し、「保存形式」オプションメニューから「visaj ブリッジの形式」を選択します。

ファイル名を指定してから「了解」ボタンを押します。この名前は、デザインのデフォルトファイル名としては使用されません。

デザインが Java 1.1 に準拠していない場合には、準拠ダイアログが表示されます。これは、非準拠デザインから Java を生成するときに表示されるものと同じダイアログです。エラーを訂正してから、もう一度保存してください。

ファイルが生成されるとき、画面にウィンドウが一瞬表示されることがあります。

Visaj へ

ファイルメニューの「Import (インポート)」プルライトメニューには、「X-Designer bridge file (X-Designer ブリッジファイル)」という項目があります (X-Designer と WorkShop Visual の保存ファイルは完全に互換性があります)。これが選択されると、ファイルダイアログが表示されます。Sun WorkShop Visual から保存されるファイルの名前を入力します。

使用する Visaj のバージョンでは対応できない機能を Sun WorkShop Visual で使用した場合[1]、検出された機能に関するダイアログが表示されます。このダイアログに表示される項目は次のとおりです。

One or more classes were expanded into simple component hierarchies

(意味: 1 つ以上のクラスが単純な構成要素の階層に展開されました)
Sun WorkShop Visual では、どのコンポーネントでもクラスにできます。Visaj では、クラスはさまざまなデザインに含まれる独立した階層として扱われます。デザイン内でこれらを一緒に使用する方法を次に示します。

1. 各クラスのコードを生成します。
2. コードをコンパイルします。
3. コンパイルされたクラスを JavaBeans としてパレットに追加します。

1. これは、バージョン 2.0 の Visaj の場合に該当します。

4. 新たに追加された JavaBeans を大きなデザインで使用します。

この方法で Sun WorkShop Visual から Visaj に変換すれば、Sun WorkShop Visual で自動的に行う場合に比べて、クラスの使用方法を柔軟に制御できます。

インポートの際に、Visaj はクラスを「拡張」します。もちろん、デザインをインポートした後でサブ階層をカットし、新しい Visaj デザインにペーストすることによって、この拡張を手直しすることもできます。

One or more frames were moved into their own classes

(意味: 1 つ以上のフレームが自分自身のクラスに変換されました)
Visaj は、クラスごとに 1 つのフレームをサポートします。デザインに複数のアプリケーションシェルや最上位シェルが含まれている場合には、そのデザインは複数 (シェルごとに 1 つ) の Visaj クラスファイルに変換されます。

これとは対照的に、ダイアログはすべてアプリケーションシェル (アプリケーションシェルがない場合には最上位シェル) が親になるように設定されます。しかし、アプリケーションシェルがなく、最上位シェルもなく、複数のダイアログがある場合には、各ダイアログは専用のクラスに置かれます。

String/font/color objects in your design were expanded into simple property settings

(意味: デザイン中の文字列、フォント、カラーのオブジェクトが単純な属性設定に展開されました)
Visaj には文字列、フォント、カラーの各オブジェクトに対応するオブジェクトがないため、オブジェクトへのすべての参照は、単純なリソース値に展開されます。たとえば、ラベル myLabel が値が「Hello」である文字列オブジェクト、<myObject> を持つ場合には、インポート後、myLabel がテキスト「Hello」を持ち、<myObject> は使用されなくなります。

All XmForms in the design were changed to absolute positioning using a null layout

(意味: すべての XmForm がヌル配置で絶対的に位置決めされました)
Visaj では、レイアウトマネージャ `com.pacist.mwt.FormLayoutManager` は存在しません。その結果、このマネージャによってレイアウトが制御されるコンポーネントは、絶対的に位置決めされます。

注 - いずれかの Java レイアウトマネージャを使用するよう、デザインを変更することをお勧めします。

インポートの後

Visaj にインポートされたデザインは、Sun WorkShop Visual での Motif デザインとまったく同じになります。これは Java コードを直接に生成した場合と同じですが、Visaj では、Java のアプリケーション開発を続行することができます。

Motif ウィジェットから Java クラスへの対応付け - MWT ライブラリ

Sun WorkShop Visual を使用すると、Motif ベースのアプリケーションでアプリケーションユーザーインタフェースをデザインして、Motif 用の C と C++ コードおよび Java コードを生成することができます。このようなことができるのは、Motif のウィジェットを Java クラスに対応付ける、MWT という Java クラスのライブラリが Sun WorkShop Visual に用意されているためです。Motif のウィジェットによって、Java に対応する要素のある場合とない場合があります。明確な対応付けのない場合は、使用可能な Java クラスを使用して MWT ライブラリが Motif ウィジェットのまねをしようとします。以下のサブセクションでは、Sun WorkShop Visual のウィジェットパレットで使用可能な Motif ウィジェットを示し、生成される Java コードで使用される Java クラスや MWT クラスを明らかにします。

シェル

シェルをアプリケーションシェルまたは最上位シェルに設定すると、Java Frame クラスに対応付けられます。ダイアログシェルに設定した場合は、Java Dialog クラスに対応付けられます。ただし、シェルの子がファイル選択ボックスである場合は、シェルとファイル選択ボックスの両方が 1 つの Java FileDialog クラスに対応付けられます。

メッセージボックス

メッセージボックスウィジェットは、XmNdialogType リソースの値に従って 3 種類の Java クラスに対応付けることができます。XmNdialogType リソースは、メッセージボックスリソースパネルの「設定」ページで設定できます。この 3 種類の対応付けは次のとおりです。

- 「ダイアログの種類」リソースを、「エラー」、「情報」、「質問」、「警告」、「作業中」のいずれかに設定している場合
  この場合、ウィジェットは、MWT ライブラリに含まれる IconMessagePanel クラスに対応付けられます。
- 「ダイアログの種類」リソースを「メッセージ」に設定している場合
  この場合、ウィジェットは、MWT ライブラリに含まれる MessagePanel クラスに対応付けられます。
- 「ダイアログの種類」リソースを「テンプレート」に設定している場合
  この場合、ウィジェットは、MWT ライブラリの DialogTemplateLayout クラスと組み合わせた Java Panel クラスに対応付けられます。

メインウィンドウ

メインウィンドウ・ウィジェットは、基本的に、特定のリソース設定を行なった複合ウィジェットです。このウィジェットを模倣するため、Sun WorkShop Visual はメインウィンドウを、設定されているリソース、および子ウィジェットの数に従って、Panel クラスと BorderLayout クラスの各種の組み合わせに対応付けます。

描画領域

描画領域ウィジェットは、MWT ライブラリの DrawingAreaLayout と組み合わせた Java Panel クラスに対応付けられます。

ダイアログテンプレート

ダイアログテンプレートウィジェットは、MWT ライブラリの DialogTemplateLayout クラスと組み合わせた Java Panel クラスに対応付けられます。

メニューバー

Java MenuBar クラスに対応付けられます。

ブリテンボード

ブリテンボードウィジェットは、MWT ライブラリの BulletinBoardLayout クラスと組み合わせた Java Panel クラスに内部的に対応付けられます。

コマンド

MWT ライブラリの CommandPanel クラスに対応付けられます。

メニュー

メニューウィジェットは、Java Menu クラスに対応付けられます。なお、これはプルダウンメニューです。ポップアップメニューは Java クラスには対応付けられません。

フォーム

フォームウィジェットは、MWT ライブラリの FormLayoutManager クラスと組み合わせた Java Panel クラスに対応付けられます。

選択プロンプト

選択プロンプトウィジェットは、MWT ライブラリの SelectionPanel クラスに対応付けられます。このウィジェットを、やはり SelectionPanel クラスに対応付けられる選択ボックスと区別するため、内部クラスメソッドが呼び出されます。

ラジオボックス

このウィジェットは、Java CheckBoxGroup クラスおよびレイアウトマネージャと組み合わせた Java Panel クラスに対応付けられます。

区画ウィンドウ

区画ウィンドウ・ウィジェットは、MWT ライブラリの PanedWindowLayout クラスと組み合わせた Java Panel クラスに対応付けられます。

選択ボックス

MWT ライブラリの SelectionPanel クラスに対応付けられます。

ローカラム

ローカラムウィジェットは、間隔設定方法リソースが「列」と「密」のどちらに設定されているかに応じて、(MWT ライブラリの) ColumnPackedRCLayout クラスか TightPackedRCLayout クラスと組み合わせた Java Panel クラスに対応付けられます。

スクロールウィンドウ

スクロールウィンドウの「スクロールの方針」リソースを「自動」に設定している場合、このウィジェットは MWT ライブラリの ScrolledPanel クラスに対応付けられます。「スクロールの方針」を「自動」に設定していない場合は、MWT ライブラリの ScrollablePanel クラスに対応付けられます。

ファイル選択

ファイル選択ウィジェットは、Java FileDialog クラスに対応付けられます。この Java クラスはダイアログであるため、ファイル選択ウィジェットの親シェルはこの対応付けに含まれます。両方のウィジェットが 1 つの Java クラスになります。

ラベル

ラベルの「種類」リソースを「ピックスマップ」に設定している場合、このウィジェットは MWT ライブラリの ImageArea クラスに対応付けられます。それ以外の場合は、Java Label クラスに対応付けられます。

カスケードボタン

このウィジェットに関しては、対応付けは行われません。このウィジェットのラベルリソースは、その子であるメニューのリソースになります。

テキストフィールド

Java TextField クラスに対応付けられます。

プッシュボタン

ラベルの「種類」リソースを「ピックスマップ」に設定している場合、このウィジェットは、MWT ライブラリの ImageButton クラスに対応付けられます。それ以外の場合は、メニュー内にあるかどうかによって異なります。メニュー内にない場合は、Java Button クラスに対応付けられます。プルダウンメニュー内にある場合は、MenuItem クラスに対応付けられます。オプションメニュー内にある場合は、ラベルリソースは親である Choice クラスのリソースとしてそのまま残されます。

オプションメニュー

オプションメニューのラベルウィジェットにラベルリソースを設定している場合、オプションメニューは、子として 1 つの Label と 1 つの Choice をもつ、FlowLayout クラスと組み合わせた Panel クラスに対応付けられます。ただし、オプションメニューの子である Label にラベルリソースを設定していない場合は、オプションメニューは Choice クラスに対応付けられます。これらのコンポーネントはすべて標準 Java クラスのものです。

テキスト

テキストウィジェットは、Java TextArea クラスに対応付けられます。

トグルボタン

トグルボタンウィジェットがメニュー内にない場合は、Java CheckBox クラスに対応付けられます。メニュー内にある場合は、Java CheckBoxMenuItem クラスに対応付けられます。

セパレータ

セパレータウィジェットがメニュー内にある場合は、ラベルがダッシュ (-) に設定された上で、Java MenuItem クラスに対応付けられます。メニュー内にない場合は、MWT ライブラリの Separator クラスに対応付けられます。

スクロールテキスト

スクロールテキストウィジェットは、Java TextArea クラスに対応付けられます。

描画ボタン

描画ボタンウィジェットは、MWT ライブラリの ImageButton クラスに対応付けられます。

スケール

スケールウィジェットに子がない場合は、MWT ライブラリの Scale クラスに対応付けられます。子がある場合は、MWT ライブラリの ScalePanel クラスに対応付けられます。

リスト

リストウィジェットは、Java List クラスに対応付けられます。

矢印ボタン

矢印ボタンウィジェットは、MWT ライブラリの Java ArrowButton クラスに対応付けられます。

スクロールバー

スクロールバーウィジェットは、Java ScrollBar クラスに対応付けられます。

スクロールリスト

スクロールリストウィジェットは、Java List クラスに対応付けられます。

第11章

Microsoft Windows 用のデザイン

はじめに

本章では、Sun WorkShop Visual を使用してアプリケーションを Microsoft Windows で実行できるようにデザインする方法について説明します。Sun WorkShop Visual は、デザインに対してMFC (Microsoft Foundation Classes) を生成します。生成されたデザインは、Microsoft Windows 環境に直接移植できます。本章の内容は以下のとおりです。

1. アプリケーションの生成
   C++ コード生成の 3 種類の様式およびダイアログテンプレートファイルの生成オプションについて簡単に説明します。

2. Microsoft Windows モードでの起動
   デザインが MFC コード生成に適しているかどうかを確認できるように Sun WorkShop Visual を実行する方法について説明します。

3. Sun WorkShop Visual ウィンドウ
   Windows デザインの作成に Sun WorkShop Visual を実行した場合の外観の違いについて説明します。416 ページを参照してください。

4. Microsoft Windows 準拠
   Sun WorkShop Visual による検査で、デザインが Windows アプリケーションの生成に適しているかどうかを確認できない場合の詳細について説明します。418 ページを参照してください。

5. 準拠不良
   取り込んだデザインが Windows 準拠検査に合格しなかった場合の対応処置について説明します。425 ページを参照してください。

6. リンクの使用
   Microsoft Windows 用のデザインでリンクを使用する方法について説明します。427 ページを参照してください。

7. 特定のウィジェットとリソースの型についての注意事項
   以下を使用する場合に注意すべき内容について説明します。

   a. マネージャウィジェットとその配置 (428 ページ)

   b. フォント (431 ページ)

   c. ピックスマップ、 ビットマップ、アイコン (432 ページ)

   d. 色 (433 ページ)

   e. 描画領域 (438 ページ)

8. サン以外のウィジェットの使用
   Microsoft Windows デザインでサン以外のウィジェット (またはユーザーウィジェット) を使用する方法について説明します。433 ページを参照してください。

9. メソッド宣言
   生成された MFC コード中でのメソッド宣言場所を制御する方法について説明します。437 ページを参照してください。

10. アプリケーションクラス
    MFC コードで CWinApp 生成を変更する方法について説明します。440 ページを参照してください。

11. ファイル名
    Microsoft Windows に移植するファイルを命名する場合の注意事項について説明します。440 ページを参照してください。

12. コード生成時の諸注意
    Microsoft Windows 用に生成するコードについて説明します。441 ページを参照してください。

13. Sun WorkShop Visual の設定
    Microsoft Windows 用のデザインの作成に関連しているリソースについて説明します。446 ページを参照してください。

第 12 章では、Windows 互換のデザインの作成およびそれに対する MFC コードの生成について手順を追って説明します。その手順に従って実際の作業を行うことで、Sun WorkShop Visual を使用して Windows アプリケーションを作成する方法について詳しく学ぶことができます。

前提知識

この章は、すでに Sun WorkShop Visual についてある程度の知識を持ったユーザーを対象にしています。具体的には、第 8 章「構造化コード生成および再使用可能な定義」に記述するような構造化コード生成と、C++ コードの生成の概念を理解していることが必要です。C++ に関する記述は、295 ページの「C++ クラス」を参照してください。

また、第 9 章「C++ コードの学習」 を参照して C++ コードの生成を実習することをお勧めします。

アプリケーションの生成

異なるプラットフォームに移植可能なアプリケーションを開発する際には、プラットフォームに固有の部分をカプセル化して、アプリケーションの本体部分が実装に関する詳細情報から切り離されるようにすることが最も良い方法です。Sun WorkShop Visual は、C++ コード構造機能を使用して、同一の公開インタフェースを持つユーザーインタフェース (ただし実装は 2 種類) に対しての一連のクラスを生成します。

Sun WorkShop Visual では、これらの実装を様式と呼びます。生成可能な C++ コードには、3 種類の様式があります。

- *Motif*
  Sun WorkShop Visual の Windows 対応バージョンを使用していない場合には、C++ コードの生成と同じになります。Sun WorkShop Visual では、ソースとともに非常に単純な一連の基底クラスが提供されています。

- *Microsoft Windows MFC*
  Microsoft Windows プラットフォームで対象とする基底クラスは、Microsoft Foundation Class です。これらは、ユーザーインタフェースの構築に使用できる、やや高レベルなコントロールと関数の集合を提供しています。

- *Motif XP*
  Sun WorkShop Visual の一部として、ソースとともに提供される一連の基底クラスです。Motif 様式基底クラスとは非常によく似ていますが、Microsoft Foundation Class に対応する名前が付けられていること、そして基本機能のほんの一部しか提供していないという点で異なっています。これは、MFC インタフェース全体を提供するものではなく、開発者がユーザーインタフェース内のコードを共有可能にすることだけを目的としています。実際の目的は、アプリケーションの残りの部分のコードを共有することです。

ダイアログテンプレートの使用

MFC を使用して Microsoft Windows 用のコードを生成する場合、以下を選択できます。

1. デザインのダイアログをダイアログテンプレートとして生成する。ダイアログテンプレートとは、ダイアログを記述する Microsoft Windows のリソースファイルです。

2. MFC コードでデザインを生成する。

1 番目の方法を選択した場合、Sun WorkShop Visual は、ダイアログテンプレートからウィジェットを取り出してダイアログを制御する MFC コードを生成します。その結果、完全なアプリケーションが生成されます。

- リソースファイルとしてダイアログを生成するには、「コードオプション」ダイアログの「リソースとして生成」トグルをオンにします。

  「リソースとして生成」トグルをオンにしないと、デザインに対して通常の MFC が生成されます。

  MFC 対応アプリケーションの開発に対しても Sun WorkShop Visual はその C++ モデルの機能をフルに発揮します。つまり、サブクラス作成および継承機能を使用して他の機能を追加することができるのです。たとえば、この機能を使ってユーザー定義ウィジェットをクロスプラットフォーム対応にすることができます。

Microsoft Windows モードでの起動

Microsoft Windows モードで Sun WorkShop Visual を実行するには、次の 3 通りの方法があります。開発目的に合った一番良い方法を選択するようにしてください。その際、同一のSun WorkShop Visual の実行形式を共有するユーザーの中には Microsoft Windows モードを望まないユーザーがいる可能性も考慮してください。

リソースファイル

Sun WorkShop Visual アプリケーション・リソースを含んでいるファイル内のリソースで、以下のように Microsoft Windows モードを指定することができます。

```
Sun WorkShop Visual.windows:true
```

コマンド行オプション

Sun WorkShop Visual を起動する際に、コマンド行のフラグを使用して Microsoft Windows モードを指定することができます。

```
Sun WorkShop Visual -windows
```

別モードの Sun WorkShop Visual

Microsoft Windows モードで Sun WorkShop Visual を起動する 3 番目の方法は、別のアプリケーションのような外観を持たせて常に Microsoft Windows モードにしておくものです。この方法は、前述のアプリケーション・リソースを使用します。

1. $VISUROOT/bin ディレクトリの Sun WorkShop Visual シェルスクリプトへのハードリンクを作成します。
2. 作成したファイルに `visuwin` などの名前を付けます。
3. 手順 2 で指定したものと同一の名前で、$VISUROOT/lib ディレクトリの Sun WorkShop Visual バイナリへのハードリンクを作成します。
4. シンボリックリンクの名前を使用して、ホームディレクトリにある *.Xdefaults* に次のように windows オプションを追加します。

   ```
   visuwin.windows:true
   ```

この方法では、単純に `visuwin` と入力します。

Sun WorkShop Visual が Microsoft Windows モードで起動します。デフォルトでは Microsoft Windows モードとならないため、その他のユーザーが誤って Microsoft Windows モードの Sun WorkShop Visual を起動してしまうことがありません。

注 - この技法は、VGA 画面などの小画面に対してのバージョンを呼び出す際に使用されます。プログラム `small_visu` も、Sun WorkShop Visual リソースファイルから別のリソースセットを取り出す Sun WorkShop Visual バイナリへのハードリンクです。

Sun WorkShop Visual ウィンドウ

Microsoft Windows モードでの Sun WorkShop Visual ウィンドウは、通常のモードとは若干異なっています。主な外観の違いは、上部にあるツールバーに項目が 2 個 (同様にメニューが 2 個) 余分にあること、そしてウィジェットパレットにいくつかのウィジェットが含まれていないことです。以上の違いのうち、後者については、418 ページの「Microsoft Windows 準拠」で説明します。ここでは、前者を説明します。

Microsoft Windows 準拠トグル

Microsoft Windows 準拠トグルは、ツールバーおよびモジュールメニューにそれぞれ 1 つずつ存在します。両方とも同じ機能を持っています。

図 11-1 ツールバー にある準拠ボタン (左 2 つ) とモジュールメニュー (右)

これらのトグルは、現在のデザインが Microsoft Windows 準拠であるかどうかを示します。 Microsoft Windows モード以外で作成したデザインを読み出す、あるいは準拠する階層の領域にカット&ペーストを行う場合には、Microsoft Windows に準拠しない構造を作成してしまう可能性があります。

このような場合には、問題のある場所を示すメッセージが表示され、ツールバーおよびメニューバーにある Microsoft Windows 準拠トグルの設定が解除されます。設定が解除されている場合には、ツールバー・トグルは赤い×印を表示します。デザインに適切な変更を加えた後で、どちらかのトグルを押すと、再度準拠の検査が行われます。この時点で、デザインが Microsoft Windows 準拠である場合には、ツールバーにあるボタンが設定されます (赤い×印は消去されます) 。また、Microsoft Windows 準拠ではない場合には、問題のある場所を示すエラー・メッセージが表示され、トグルは設定解除されたままとなります。

様式メニュー

ツールバーおよび「コード生成」ダイアログ内にある様式メニューは、同一のものです。このメニューは、生成するコードの様式 (Motif、Motif XP または Microsoft Windows) を指定します。このメニューは C++ コード生成にのみ影響します。

図 11-2 ツールバーにある様式メニュー

準拠か非準拠かの表示

Microsoft Windows 様式でのデザイン生成の障害となる Microsoft Windows 非準拠の問題に加えて、デザインには Microsoft Windows 実装では何の効果も持たない多くの属性 (ボタン上のラベルの整列など) が存在します。これは、Microsoft Windows ツールキットがその概念を持たないためです。Sun WorkShop Visual ではアイコンと色の違いを使用して、このような無効な属性を示します。

1. アイコンによる表示
   リソースパネル内では、Microsoft Windows 上で無効なリソースは×印のアイコンで示され、有効なリソースはチェック印のアイコンで示されます。

2. 色による表示
   Motif 様式では有効でも Microsoft Windows 様式では無効なリソース設定は、デフォルトでは、そのテキスト入力フィールドとオプションメニューがピンク色で表示されます。

選択したウィジェットの変数名フィールドがピンク色[1] で表示されている場合は、そのウィジェットに相当する Microsoft Windows オブジェクトは存在しないことを意味しています。たとえば、Microsoft Windows の場合、メニューバーはダイアログの属性の 1 つであり、メニューバーのオブジェクトというものは存在しません。したがって、Sun WorkShop Visual はメニューバーウィジェットに対しては変数名をピンクで表示することになります。

Sun WorkShop Visual は同様の方法で、あるリンクが Microsoft Windows コードでは再現できないことも示します。リンクについての詳細は、427 ページの「リンクの使用」 を参照してください。

Microsoft Windows 準拠

Motif および X ツールキットは Microsoft Windows ツールキットとほとんど類似性がありません。ツールキットの外観は類似していますが、実際の用途は大きく異なっています。このため、デザインに対して Microsoft Windows コードを生成するには、いくつかの制限が発生します。この制限は、Microsoft Windows モードの場合に発生します。*Microsoft Windows* モードでは、制限に準じていないデザインは開発者自身がデザインを修正する必要があります (たとえば、既存のデザインを読めるようにするため)。そこで、Sun WorkShop Visual はそのデザインが *Microsoft Windows* に準拠しているかどうかを調べます。デザインが Microsoft Windows 準拠ではない場合には、Motif 様式の C++ コードだけが生成されます。

この項では、Microsoft Windows 様式のコードを生成するために Sun WorkShop Visual が課す制約について説明します。Microsoft Windows モードの場合、これらの制約に準拠していないデザインは作成はできません。

1. ピンク表示はカラーディスプレイでのみ識別することができます。

構造の制約

Motif と Microsoft Windows では、イベントの処理方法が異なっているため、クラスにできないウィジェットがいくつか存在します。Microsoft Windows の場合では、ある種のウィジェットに関連するイベントは常に、そのウィジェットを包含するクラスに送られます。その他のウィジェットは、Microsoft Windows メッセージを処理するためにクラスにする必要があります。以下に、これらの制約を示します。

- クラスにできないウィジェット
  - メニューバー
  - ポップアップメニュー
  - カスケードボタン
  - オプションメニュー
  - シェルの子であるすべてのウィジェット
- クラスでなければならないウィジェット
  - シェル
  - スクロールウィンドウ (シェルの子ではない場合)
  - フレーム
  - ラジオボックス (フレームの子ではない場合)
  - 描画領域 (メインウィンドウ、スクロールウィンドウまたはシェルの子ではない場合)
  - 区画ウィンドウ
  - 区画ウィンドウの子

Microsoft Windows に準拠していないデザインを読み込む際に最も起こりやすいエラーは、シェルが C++ クラスとして構成されていないということです。このエラーは修正しやすく、準拠不良ダイアログで自動的に修正することができます。詳細は、本章 425 ページの「準拠不良」を参照してください。

C 構造体

Microsoft Windows 準拠デザインでは関数やデータの構造化オプションはサポートされません。

クラスおよびコールバック

デザインが高度に構造的であり、C++ クラスを多用し、コールバック・メソッドが子ウィジェットの中に分散されているという場合には、構造的なエラーはかなり複雑なものになります。この問題の起こる状況とその対応策を、次の例に示します。

例

ウィジェットにコールバック・メソッドがある場合、そのウィジェットを含むクラスでメソッドが宣言されます。次の例では、Sun WorkShop Visual を Microsoft Windows モード以外で使用して、メニューバー (MBar_class) をクラスとして宣言し、ボタンにコールバック・メソッドを与えました。

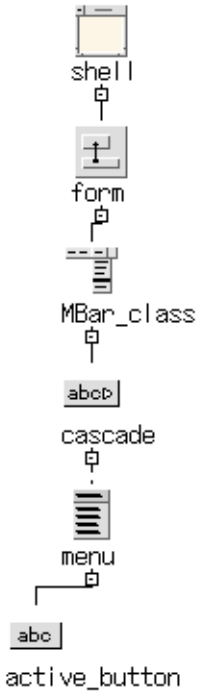


図 11-3 非準拠の階層

このメソッドは MBar_class 中で宣言されます。次にこのデザインを Microsoft Windows モードで読み込むと、次のようなエラーメッセージが表示されます。これはメニューバーをクラスにすることができないためです。

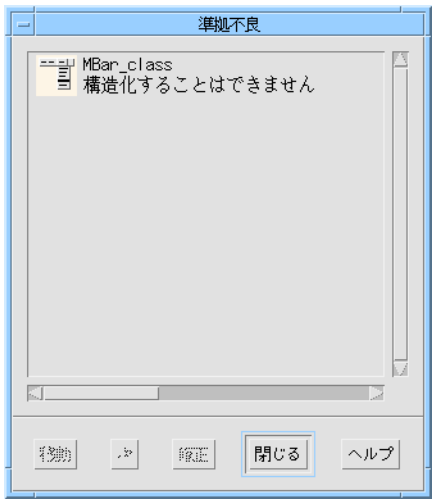


図 11-4 準拠していないことを示すエラーメッセージ

コアリソースダイアログを使用してメニューバーのクラス定義を削除すると、次のようなダイアログが表示されます。

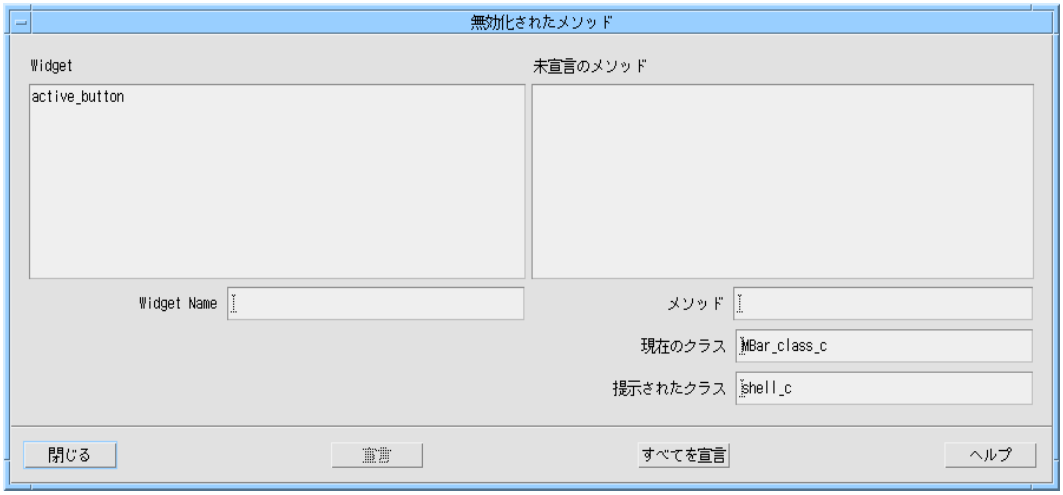


図 11-5 メソッド・コールバック宣言が無効であることを伝えるダイアログ

この場合はメソッドの宣言をメニューバーから削除し、シェルに追加してください。メソッド宣言はフォームに追加することはできません。これは、フォームはメニューバーと同様、Microsoft Windows オブジェクトとして実装されないためです。

Sun WorkShop Visualは、上記のようにウィジェットクラス定義の削除を支援していますが、あるウィジェットをクラスにするかどうかによって、アプリケーションは大きな影響を受けます。構築するアプリケーションの構造を充分に考慮した上で、クラス定義の指定または削除を実行するようにしてください。

メニューバーの制約

Microsoft Windows でのメニューバーは、ダイアログの属性を設定して作成します。そのため、2 つの準拠上の制約が生じます。

- シェル 1 個に付き、メニューバーは 1 個だけサポートされる
  ダイアログ内に 2 個以上のメニューバーを含むデザインは作成できません。
- メニューバーの親は、シェルの子でなければならない
  メニューバーは、シェルの直接の子となることはできません。また、ウィジェット階層内でシェルの孫よりも深い位置に存在することはできません。

ファイル選択ボックス

ファイル選択ボックスは、ダイアログシェルまたは最上位シェルの子でなければなりません。Microsoft Windows の場合は、事前に定義されているファイル選択ダイアログでファイル選択が行われます。このダイアログは、作業領域だけをその単一の子として含むことができ、メニューバーや (作業領域がマネージしていない) 他のボタンをサポートすることはできません。

サポートされていないウィジェット

以下に示すウィジェットは、Microsoft Windows で対応するコントロールを持っていないため、移植性のあるデザインに使用することはできません。

- 選択ボックス
- コマンド
- メッセージボックス
- 選択プロンプト
- 描画ボタン
- 矢印ボタン

最後の 2 つのボタン以外はすべて、Windows コントロールとしてサポートされているダイアログテンプレートを使用して類似した機能を実装できます。

スケール

スケールウィジェットは、子コントロールをサポートすることができない Microsoft Windows スクロールバーコントロールとして実現されます。したがって、子を持つスケールウィジェットは、Microsoft Windows 準拠制約に違反することになります。

フレームとラジオボックス

包含制御へメッセージを引き渡す手順の関係から、フレームとラジオボックス(フレームの子でない場合)は両方ともクラスでなければなりません。クラスとして指定されていないと、格納コントロールへのメッセージの受け渡しが行われず、メッセージがダイアログで処理されません。ただしシェルの子はクラスにはなれないことから、フレームとラジオボックスは、どちらもシェルの直接の子になることはできません。

フレームの 2 番目の子は、ラベルウィジェットでなければなりません。このラベルウィジェットはフレームのタイトルになります。Microsoft Windows におけるフレームコントロール (実際は CButton) は、タイトルの属性のみを持っています。タイトルとして別のコントロールを使用することはできません。 1 番目の子には、どのウィジェットでも使用できます。

メインウィンドウとスクロールウィンドウ

Microsoft Windows は、自動スクロールをサポートしていません。したがって、Microsoft Windows モードの Sun WorkShop Visual では、自動スクロールオプションは使用できません。メインウィンドウウィジェットは、作業領域とメニューバーのみを子として持つことができます。コマンドウィンドウやメッセージウィンドウはサポートされていません。

区画ウィンドウ

Windows 準拠のデザインは、セパレータ、メインウィンドウ、オプションメニュー、またはメニューバーといった子を持つ区画ウィンドウを含むことができません。子は定義またはインスタンスにすることはできません。

定義

XmNlabelType リソースは、定義の構成要素であるウィジェットに対して、別のデザインでインスタンス化される時に明示的に設定することはできません。つまり、定義中でボタンの XmNlabelType リソースが設定されていない場合、その定義が使用される時に XmNlabelType に PIXMAP を設定することはできません。これは、Microsoft Windows がビットマップを使用してボタンを実装する場合には、異なるクラス (Cbuttonではなく、CBitmapButton) を使用するためです。また、制約により、作成後に変数のクラスを変更することはできません。

同じ理由から、プルダウンメニューを持たないカスケードボタンが定義中にある場合、そのインスタンスにはプルダウンを追加することはできません。

また、419 ページの「構造の制約」で説明しているように、シェルの子は C++ クラスにはできないため、定義のルートウィジェットにはできません。シェルの子を定義にする場合は、シェルとその子の間に「ダミーの」コンテナウィジェットを追加してください。このようにすれば、このコンテナウィジェットをルートウィジェットとして、子を定義にすることができます。

さらに、サブクラス化されていないインスタンスに対し、追加されるメソッドのあるウィジェットも定義には使用できません。たとえば、ローカラム定義のインスタンスがあり、そのルートウィジェット (つまりローカラム) は、その構造をクラスと設定していない場合を考えてみてください。Microsoft Windows モードではない場合には、ローカラムのインスタンスにボタンを追加して、その包含スコープ内 (たとえば、シェルクラス) で宣言されているメソッド・コールバックを指定することができます。これは、Microsoft Windows モードでは行うことができません。イベントはそれを包含するコントロール (この場合はローカラムを表わす CWnd) によって処理されなければならないからです。

定義内のウィジェットには名前を指定する必要があります。これは、Microsoft Windows で使用するウィジェットだけでなく、すべての C++ 定義にも必要なことです。

準拠不良

Microsoft Windows モード以外での Sun WorkShop Visual で作成したデザインを読み出すか、カット&ペーストを使用してデザインを作成すると Windows 非準拠になる可能性があります。Windows 非準拠になった場合は、準拠不良ダイアログが表示され、非準拠の原因となっているウィジェットが表示されます。

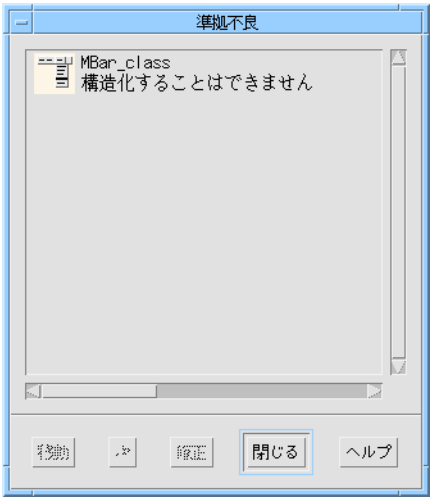


図 11-6 準拠不良ダイアログ

非準拠ウィジェットのリストをスクロールして、ウィジェットを選択することができます。ウィジェットを選択し、次のボタンの 1 つを押します。

- 移動
  デザイン内でウィジェットを強調表示します。ウィジェットが現在のダイアログ以外のダイアログの一部である場合には、関連するダイアログが選択されます。ウィジェットがフォールドされているデザインの一部である場合には、そのデザインがアンフォールドされます。ウィジェット上でダブルクリックを行うと、移動を押した場合と同様の効果を得ることができます。構造の問題の場合には、Sun WorkShop Visual はコアリソースダイアログをコード生成ページで開きます。
- 次
  リスト内の次のウィジェットに移動し、階層内でそのウィジェットを選択します。

- 修正
  可能な場合は問題を解決します。Sun WorkShop Visual は、ウィジェットの構造に関連しているエラーのみを解決することができます。その他のエラーは、自動的に解決することはできません。発生する可能性がある準拠エラーについては、418 ページの「Microsoft Windows 準拠」を参照してください。

例

この例では、どのように準拠不良が検出されるかを示します。図 11-7 に示されている階層では、ローカラムウィジェット *rc_is_class* がクラスに指定されています。

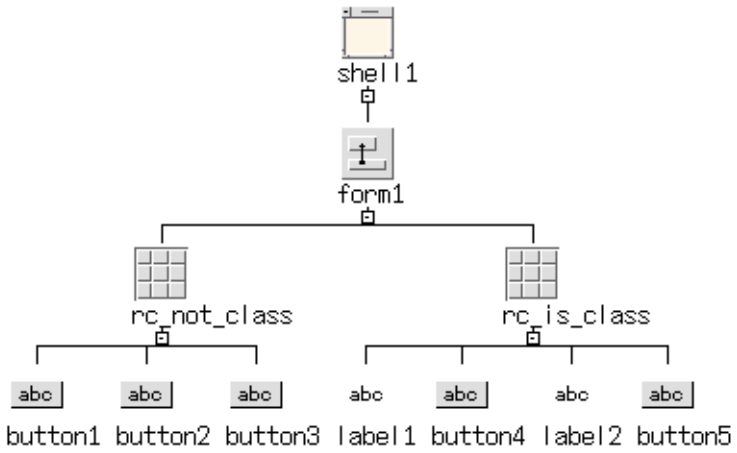


図 11-7 クラスとして定義されているローカラム

rc_is_class をカットし、フォームを消去してペーストを行う (つまり、*rc_is_class* をシェルの子としてペーストする) と、Sun WorkShop Visual はデザインを非準拠として、図 11-8 に示される準拠不良ダイアログを表示します。このダイアログは、シェルの子がクラスになることができないことを示します。

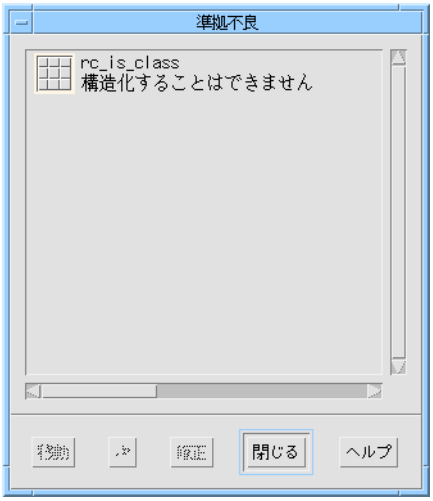


図 11-8 シェルの子としてクラスをペーストした場合のエラー

操作を続けることはできますが、 *rc_is_class* の構造が変更されない限り、デザインは Microsoft Windows 準拠にはなりません。ツールバーにある Microsoft Windows 準拠トグルに赤い×印が表示され、Microsoft Windows 準拠ではないことが表示されます。

リンクの使用

Motif 様式では、リンクは事前定義されているコールバックです。Microsoft Windows では、ボタンのメッセージ・ハンドラによって呼び出される単純な大域関数として実装されています。ただし、Microsoft Windows においてリンクを使用する場合には、いくつかの制約があります。これらの制約は、リンクに対してコードを生成するかどうかにのみ影響するものであり、デザインの準拠には影響しません。

Microsoft Windows 上のオブジェクトではない宛て先ウィジェット

リンクの宛て先として選択されているウィジェットが Microsoft Windows 上でオブジェクトとして割り当てられていない場合には、リンク編集ダイアログ内で追加ボタンがピンク色になります。リンクを追加することはできますが、追加されたリンクは

Motif でのみ有効で、Microsoft Windows コードには生成されません。これを示すために、リンクのリスト内にあるウィジェットはピンク色となります。Microsoft Windows オブジェクトに割り当てられるウィジェッの詳細は、898 ページの「Motif ウィジェットの Microsoft Windows ウィジェットへの変換」を参照してください。

リンクの宛て先としてのメニュー内のボタン

宛て先ウィジェットがメニュー内のプッシュボタンであるリンクを追加する際には、リンクの種類は有効化または無効化に制限されます。Microsoft Windows ではメニューボタンを表示または非表示、マネージまたはアンマネージすることはできません。

リンクの宛て先としてのファイル選択ダイアログ

ファイル選択ボックスは、Microsoft Windows 上では、DoModal メソッドを呼び出すと表示される CFileDialog というモード付きダイアログとして実装されます。このメソッドは、ファイル選択が完了する、あるいは取り消されるまで戻りません。このため、Microsoft Windows の場合には、表示 (show) リンクだけがサポートされています。MFC、Motif XP の様式に対しては、選択が完了、または取消されるまで DoModal メソッドが戻らないようにコードが構成されます。

マネージャウィジェットとその配置

Motif マネージャウィジェットに相当するものは Microsoft Windows にはありません。フォームまたはローカラムなどのウィジェットは、Microsoft Windows では存在しません。クラスではないこれらのウィジェットが 1 つでもデザインに存在すると、生成されるコードでは無視されます。ウィジェットがクラスの場合は、Sun WorkShop Visual は、代わりにキャンバスを生成します。

デザインからダイアログテンプレートを生成する場合、ダイアログテンプレートは「平坦」になるため、ほとんどのマネージャウィジェットは無視されます。ダイアログテンプレートでは、Motif でよく使用される包含階層は使用しません。したがって、テンプレートファイルのデザインは平坦なものになります。ウィジェットはサブコンテナから取り出されて、ダイアログの子になります。これによって、デザイン全体でシステムリソースを使用することができます。

Windows のリソースファイル (またはダイアログテンプレート) にメインアプリケーションコードと一緒にデザインを生成すると、Microsoft Windows IDE (Integrated Development Environment) を使用してデザインの配置を変更できます。ここで行なった変更内容は Sun WorkShop Visual に戻すことはできませんが、配置変更が必要になったときにすぐに変更するには便利な方法です。

ダイアログテンプレートを生成したくない場合は、Sun WorkShop Visual は生成の際の大きさおよび位置の絶対値をコード中に生成します。したがって、たとえば幅 100 ピクセルで高さ 30 ピクセルのプッシュボタンが (10, 200) の位置に配置されている際に、このボタンに対してユーザーが x, y、幅および高さのリソースを明示的に設定しなかった場合でも、それらのリソースは Motif ツールキットによって自動的に計算され、その明示的な値が Microsoft Windows コード内で使用されます。これには、Motif で対応しているダイアログに非常によく似た Microsoft Windows ダイアログが生成されるという利点があります。

フォントとその外観

Sun WorkShop Visual は、Microsoft Windows コントロールに対して絶対サイズを生成するため、ダイアログのサイズは、その中に表示されるテキストにどんなフォントを使用しても適切でなければなりません。そのためには、デザイン過程でも、ダイアログを実際に表示する際に使うフォントと類似したサイズのフォントを使用することをお勧めします。これには 2 つの方法があります。

1. コントロールに対して XmNfontList リソースを設定、あるいはシェルで適切なフォントリソースを設定して、コントロールに特定のサイズのフォントを強制的に使用させる。
   ダイアログは類似した大きさになります。

2. Sun WorkShop Visual が Microsoft Windows で使用されるデフォルトのフォントに近いフォントを使用してデザイン中のウィンドウを表示するようにリソースを設定する。
   結果として Sun WorkShop Visual はそのフォントに適した絶対的なサイズを生成します。Motif コードは、通常の方法でデフォルトフォントを使用します。
   Sun WorkShop Visual に、Microsoft Windows のデザインには特定のフォントを使用させるためには、ユーザーのホームディレクトリ中にある .Xdefaults で、リソースを適切なフォントに設定してください。

   ```
   visu*dialog.labelFontList:\
   -adobe-helvetica-medium-r-normal-*-14-*-*-*-*-77-iso8859-1
   ```

```
visu*dialog.buttonFontList:\
-adobe-helvetica-medium-r-normal-*-14-*-*-*-*-77-iso8859-1
visu*dialog.textFontList:\
-adobe-helvetica-medium-r-normal-*-14-*-*-*-*-77-iso8859-1
```

ご使用の Microsoft Windows システムによって適切なフォントは異なります。適切なフォントの決定にあたっては、サイズと平均的な幅の値が重要になります。

サイズ変更動作

Microsoft Windows モードでは、ユーザーによるダイアログのサイズ変更動作を実現するため、Sun WorkShop Visual は、OnSize メッセージ・ハンドラを生成します。Sun WorkShop Visual は、Motif のジオメトリ管理を正確に再現しようとはせず、特定のマネージャウィジェットの動作に類似したサイズ変更動作を行うハンドラを生成します。以下がこれらのマネージャウィジェットです。

- スクロールウィンドウ
- フォーム
- フレーム
- ダイアログ・テンプレート

これらのマネージャは、サイズ変更動作を生成するためには、クラスである必要はありません。Sun WorkShop Visual は、それを包含するクラスに対し、そのすべての子孫ウィジェットを扱うハンドラを生成します。ユーザーがたとえばサブクラスを介して独自のハンドラを提供する場合には、コアリソース・ダイアログのコード生成ページにある「MFC の OnSize ハンドラ」トグルの設定を解除することにより、サイズ変更ハンドラの生成を抑制することができます。図 11-9 を参照してください。

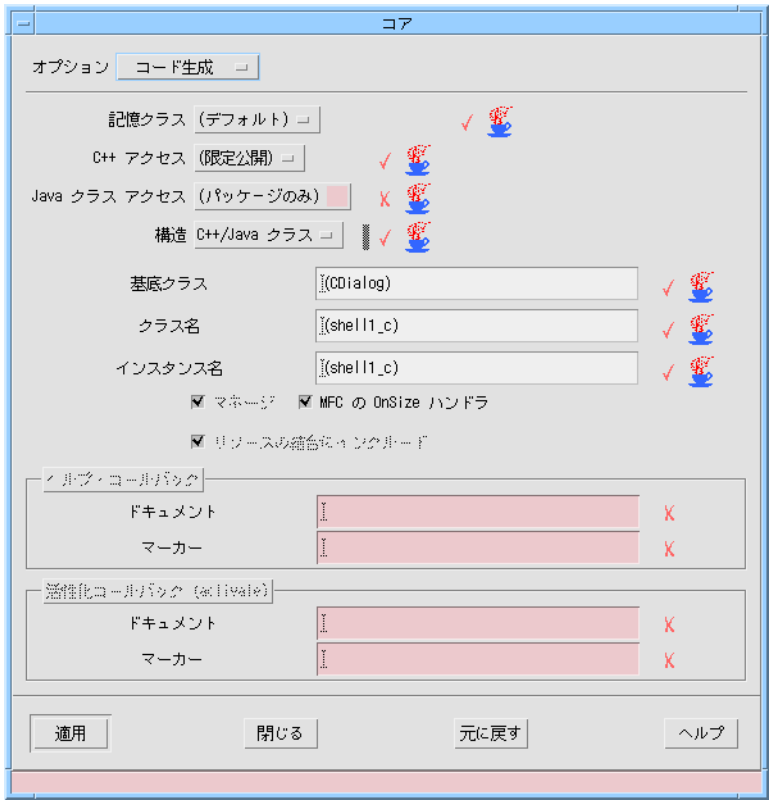


図 11-9 コアリソースパネルの「コード生成」ページ

フォント

Microsoft Windows コードにフォントを生成するためには、フォントオブジェクトを使用する必要があります。これは、フォントの Microsoft Windows 上での持続性を保証するためです。これは、フォントオブジェクトを使用することによってのみ保証できます。

Sun WorkShop Visual では、フォント選択ダイアログ内のフォントリストから項目が選択された場合には、適用ボタンをピンク色にするといった視覚的な警告手段を提供しています。ただし、フォントオブジェクトのリストから項目が選択された場合には、適用ボタンはピンク色ではなくなります。

フォントリストとコンパウンド文字列

Sun WorkShop Visual でオブジェクトに対してフォントリストを指定すると、リスト内の最初のフォントが Microsoft Windows でのオブジェクトに対して使用されます。タグの混在するコンパウンド文字列は、指定されている最初のフォントのみを使用して Microsoft Windows に変換されます。

フォントの命名

Microsoft Windows 内のフォントは、 X ウィンドウズとはかなり異なった方法で名前が付けられています。しかし、Microsoft Windows には洗練された照合アルゴリズムがあるため、Sun WorkShop Visual がかなり粗雑なマッピングを使用してフォント指定を変換しても、また、Microsoft Windows では使用できないフォントを指定した場合であっても、それらは妥当な外観のフォントに置き換えられます。

ピックスマップ、 ビットマップ、アイコン

Sun WorkShop Visual で作成されたピックスマップ・オブジェクトは、Microsoft Windows ビットマップまたはアイコンに変換されます (オブジェクトがボタンであるか、ラベルであるかによります) 。この変換は、Microsoft Windows 用リソースファイルの生成の際、自動的に行われます。X モノクロームビットマップは、Microsoft Windows 変換できません。

「コード生成」ダイアログから「Microsoft Windows リソース」を選択した場合には、Sun WorkShop Visual は、ユーザーが作成した各ピックスマップに対してリソースファイルおよびビットマップまたはアイコンファイルが作成されることを通知します。ビットマップファイルには接尾辞 .bmp が、また、アイコンファイルには接尾辞 .ico が付けられます。Microsoft Windows に対しては、ファイルにピックスマップを保存する必要はありません。Motif モードで使用する場合は、保存してください。Microsoft Windows モードでは、アイコンは常に 32 x 32 ピクセルで表示されます。

ピックスマップを使用するボタン

ピックスマップ型のラベルを持つ Motif ボタンの場合には、Sun WorkShop Visual は Microsoft Windows 対応の CBitmapButton を生成します。Microsoft Windows の CBitmapButton は、Motif のピックスマップを使用するボタンとは異なって、境界線 (ボーダー) を持たないため、実際には、ボタンのようには見えません。このため、ピックスマップデザイン内に境界線を組み込んでおくという方法もあります。

色

Microsoft Windows のオブジェクトに割り当てられるウィジェットの背景および前景色は、自由に設定することができます。これらの色は、RGB (赤、緑、青) の値を使用して Microsoft Windows コードに生成されます。Microsoft Windows では、通常、X/Motif ほど色が豊富ではありません。このため、2 種類のプラットフォーム間では同一の色に見えない場合があります。ただし、デフォルトでは、Microsoft Windows 95 の「見た目と使い心地」の色が使用されます。

カラーオブジェクト

Sun WorkShop Visual では、通常の方法で背景および前景色を指定してください。背景色は、カラーオブジェクトでなければなりません。前景色は、カラーオブジェクトである必要はありません。

サン以外のウィジェットの使用

Sun WorkShop Visual は、デフォルトの Motif セットに加えて、Xt ツールキットのウィジェットをサポートするように拡張できます。ここではデフォルトで用意されている以外のウィジェットを、「ユーザー定義ウィジェット」または「サン以外のウィジェット」と呼びます。この項目については、第 23 章「ユーザー定義ウィジェット」で詳細に説明しています。新たに追加したウィジェットは、Sun WorkShop Visual ウィジェットパレットに表示され、事前に定義された Motif のウィジェットと同じように使用できます。

Microsoft Windows モードでは、現時点ではサン以外のウィジェットに対しての特別なサポートはありませんが、Sun WorkShop Visual の C++ モデルの持つ柔軟性のため、サン以外のウィジェットも使用することができます。

Sun WorkShop Visual の自動処理

Sun WorkShop Visual は、サン以外のウィジェットをあたかもその派生したクラスのインスタンスであるかのように扱います。たとえば、XmPushButton からサン以外のウィジェットが派生している場合、Sun WorkShop Visual は、そのウィジェットをあたかも XmPushButton であるかのようにデザインに追加します。次に、Motif XP での XmPushButton の実装方法にあわせて、CButton クラスにもとづく Motif XP コードを生成してサン以外のウィジェットのインスタンスを生成します。これによって Microsoft Windows コードは、プッシュボタンを使用した場合とまったく同じになります。

上記の例で使用している XmPushButton クラスは Motif XP によってサポートされています。しかし、サン以外のウィジェットが Motif XP によってサポートされていないクラスから派生している場合は、Sun WorkShop Visual はそのウィジェットを扱うことができず、ウィジェットのコードも生成しません。この場合、次の「サン以外のウィジェットに対する Sun WorkShop Visual の構成」で説明する方法を使用してください。

サン以外のウィジェットに対する Sun WorkShop Visual の構成

特定のサン以外のウィジェット (上記を参照) に対するSun WorkShop Visual のデフォルト動作に満足いかない場合は、C++ モデルの柔軟性を使用して Sun WorkShop Visual の動作を拡張できます。

1. サン以外のウィジェットから Windows コントロールへの明確な対応関係を Windows コントロール (組み込み MFC クラスでも別のサン以外のコンポーネントでも可) に指定します。たとえば、サン以外のウィジェットの多くが ComboBox ウィジェットを含んでいますが、これは明らかに組み込み MFC クラスである CComboBox に対応しています。

2. Sun WorkShop Visual にクラスの名前を指定します。コアリソースパネルの「コード生成」ページで、選択したサン以外のウィジェットに対して生成される C++ クラス構造を指定します。Sun WorkShop Visual は Motif 構成要素に対して独自の対

応関係を使用してこの「コード生成」ページに記入しますが、サン以外のウィジェットに対する処理方法を常に用意できているわけではありません。処理方法が不明な場合は、単純に基本のキャンバスを生成します。この場合は、Sun WorkShop Visual が割り当てた対応関係を、ユーザー自身が適切なものに修正する必要があります。Sun WorkShop Visual は、修正されたとおりにコードを生成します。つまり、MFC コードとして生成されるオブジェクトは正しいものになります。

UNIX では、ユーザーは見せかけの (ダミー) クラスを組み込む必要があります。その結果、必要に応じて、純粋な Motif C++ または Motif XP のいずれかにコンパイルできます。追加が必要なコードを以下に示します。

```
/* 純粋な Motif C++ コード */
#ifdef    XD_MOTIF
#define MY_BASE_CLASS xd_XtWidget_c  /* Sun WorkShop Visual の
                                                基底ウィジェットクラスを使用
する */
#endif /* XD_MOTIF */

/* Motif XP C++ コード (厳密な MFC API を使用する UNIX C++) */
#ifdef    XD_MOTIF_MFC
#define MY_BASE_CLASS CWnd /* Motif XP の「MFC」基底コントロール
                                        クラスを使用する */
#endif /* XD_MOTIF_MFC */

/* Windows C++ コード (本来の MFC) */
#ifdef    XD_WINDOWS_MFC
#define MY_BASE_CLASS   CComboBox /* MFC コントロールクラスの実
                                                マッピングを使用する */
#endif /* XD_WINDOWS_MFC */

#if   defined(XD_MOTIF_MFC) || defined(XD_MOTIF)
class MyMappingClass_c : public MY_BASE_CLASS
{
```

```
}
#endif /* XD_MOTIF_MFC || XD_MOTIF */
```

Motif XP の生成では、Motif XP ライブラリを拡張することもできます。詳細は、957 ページの「Motif XP の強化」 を参照してください。

サン以外のウィジェットのリソース

サン以外のウィジェットで、認識できないウィジェットの場合、どの Motif リソースがどの MFC リソースに対応するべきか Sun WorkShop Visual は判断できません。このようなウィジェットの場合、リソースに対するコードを手動で追加する必要があります。以下に、その方法をいくつか示します。

1. すべてのサン以外のウィジェットのリソースに緩い結合を設定します。これにより、サン以外のウィジェットのリソースが X リソースファイルに移動し、Sun WorkShop Visual で保持されるため、X 上で動作するようになります。次に、MFC リソースも X 上で動作できるように MFC リソースファイルを手動で編集します。緩い結合の作成方法と使用方法についての詳細は、97 ページの「緩い結合」を参照してください。
2. リソースとコードオプションを正しく設定して UNIX 上でコードを生成し、サン以外のウィジェットに対してはコード中にリソースを生成しないようにします。そして、1 と同じように MFC リソースファイルを手動で編集します。
3. コードプレリュードを指定して必要なリソースを追加します。コードプレリュードには任意のものを指定できます。たとえば、以下に示す内容をマネージの前プレリュードとして指定することもできます (my_text をクラスと仮定)。

```
#ifdef    XD_MOTIF
XtVaSetValues(my_text->xd_rootwidget(), XmNvalue, "Hello", 0) ;
#endif /* XD_MOTIF */

#if   defined(XD_MOTIF_MFC) || defined(XD_WINDOWS_MFC)
my_text->SetWindowText("Hello World") ;
#endif /* XD_MOTIF_MFC || XD_WINDOWS_MFC */
```

メソッド宣言

「メソッド宣言」ダイアログは、Microsoft Windows モードの場合には「Microsoft Windows MFC」というラベルが付いているトグルボタンが存在します。

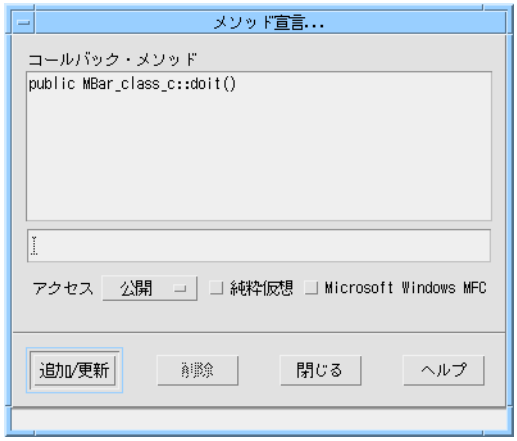


図 11-10 「Microsoft Windows MFC」トグルを持つ「メソッド宣言」ダイアログ

このトグルは、Microsoft Windows コードを生成する際に、包含するクラスのクラス構造でメソッドが宣言されているかどうかを示すために使用されます。Motif のコードを生成する場合、メソッドは包含するクラスで宣言されます。包含するクラスとは、(明示的、あるいは自動的に) その構造がクラスに設定されている、最も近い祖先です。

これは、 Microsoft Windows コードのスタブにメソッドが現れるかどうかを示すトグルではありません。その表示は、「コールバック」ダイアログで行われます。そのダイアログでアスタリスク (*) が表示されているコールバック・メソッドは、Microsoft Windows のスタブファイルに生成されません。通常は Sun WorkShop Visual が提供するデフォルトを使用する場合でも、「Microsoft Windows MFC」トグルを使用すると、メソッド宣言を制御することができます。選択したクラスでメソッドを宣言する、あるいはユーザー独自のクラスでメソッドを宣言するには、メソッド宣言ダイアログを使用します。メソッドは必ずどこかで宣言する必要があります。

「コールバック」ダイアログでメソッドコールバックを追加する際に、そのメソッドが宣言されていない場合、Sun WorkShop Visual は包含するクラス中にこれを宣言します。包含するクラスの中でメソッドを宣言したくない場合は、「メソッド宣言」ダイアログを使用して「Microsoft Windows MFC」トグルをオフに設定してください。

描画領域

描画領域がスクロールウィンドウ、メインウィンドウまたはシェルの子ではない場合は、基本の CWnd クラスとして作成されます。その他の場合は、Microsoft Windows コード生成に対しては無視されます。詳細については、898 ページの「Motif ウィジェットの Microsoft Windows ウィジェットへの変換」を参照してください。

Microsoft Windows に対しての描画コールバックの追加

Motif XP クラスライブラリは、描画サポートを一切含んでいません。描画サポートが必要な場合、プラットフォームに固有のコードを書かなければなりません。ただし、Sun WorkShop Visual では、Motif 様式および Microsoft Windows メッセージ・ハンドラに対してのみデフォルトで宣言されるコールバック・メソッドを追加することができます。Microsoft Windows モードの場合は、Sun WorkShop Visual は複数のトグルボタンを描画領域リソースパネルに追加します。これらは、生成コードに Microsoft Windows メッセージ・ハンドラを追加するために使用できます。

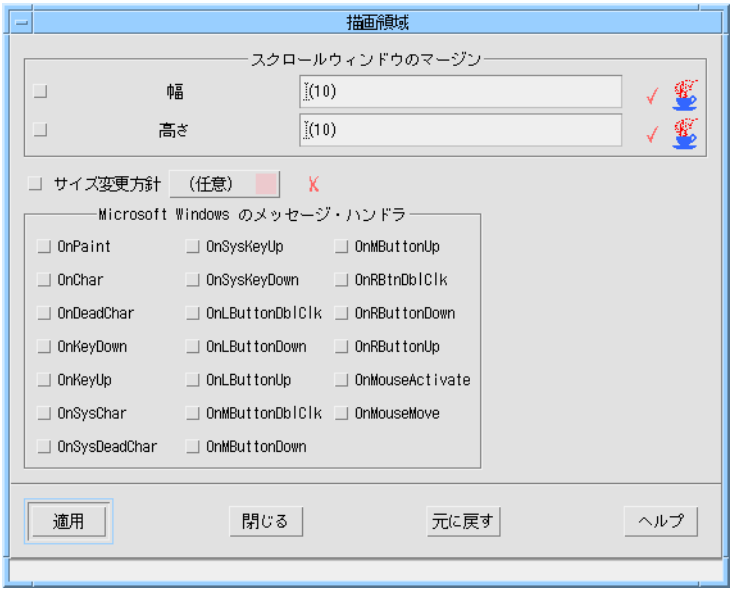


図 11-11 描画領域リソースパネル

描画領域に対しての Microsoft Windows メッセージ・ハンドラ

たとえば、図 11-11 のパネルで OnRButtonDown トグルを選択すると、以下のスタブがユーザーのコールバック・スタブファイルに追加されます。

```
afx_msg void scrolled_win_c::OnRButtonDown( UINT nFlags, CPoint point
)
{
}
```

Microsoft Windows においては、afx_msg は擬似キーワードであることに注意してください。以下に示す行は、C++ 外部宣言ファイルに追加されます。

```
//{{AFX_MSG(scrolled_win_c)
afx_msg void OnRButtonDown( UINT nFlags, CPoint point ):
//}}AFX_MSG
DECLARE_MESSAGE_MAP ()
```

これにより、Microsoft Windows にメッセージ・ハンドラが登録されます。

アプリケーションクラス

Sun WorkShop Visual は、MFC C++ 様式ではアプリケーションを表わすために、CWinApp クラスのインスタンスを生成します。このクラスを構成するには、「モジュール」メニューから表示される「アプリケーション・クラス」のダイアログを使用します。図 11-12 にダイアログを示します。

図 11-12 「アプリケーション・クラス」ダイアログ

注 - 自身で定義した場合も Sun WorkShop Visual のデフォルト設定で定義した場合も、アプリケーションの宣言は、メイン手続き (「コード生成」ダイアログ中の「メインプログラム」) の生成中に生成されるだけで、その他では生成されません。

ファイル名

PC 上のファイル名は、ドットの前に 8 バイト以下、ドットの後に 3 バイト以下の合計 12 バイト (ドットを含む) に制限されています。2 種類のプラットフォーム間でファイルを共有する場合には、この制限は Microsoft Motif コードを生成する際にも適用されます。また、MS-DOS および Microsoft Windows は大文字小文字の区別を行いません。したがって、ファイル名を区別するために大文字と小文字で違いを付けないようにしてください。

ピックスマップ

上記の制限は、ピックスマップ・オブジェクトを命名する際にも適用されます。ピックスマップを作成した後で、Sun WorkShop Visual に Microsoft Windows リソースファイルを生成するように要求すると、Sun WorkShop Visual はピックスマップエディタに指定した名前を使用して、自動的に Microsoft Windows ビットマップおよびアイコンを別々のファイルに生成します。したがって、8 バイトを超える名前を指定した場合には、Microsoft Windows で問題が生じることになります。

C++ コード

異なるコンパイラには、ファイル名の拡張子に関してさまざまな制約があります。接尾辞 `.cxx` は、一般的に許容されているようです。また、`.cpp` は多くのコンパイラで許容されています。さらに、コンパイルシステムによっては、`.C` および `.c++` を受容するものもあります。Visual C++ では、C++コードを含むファイルに対して`.c` が指定された場合には、受け入れられません。

「コード生成」ダイアログのデフォルトのフィルタを変更する方法については、447 ページの「ファイル名フィルタの設定」を参照してください。

メークファイル

Microsoft Windows MFC のコードファイルには、メークファイルは生成されません。`make` は、UNIX のユーティリティです。Microsoft Windows 版のアプリケーションを構築するには、Visual C++ などの Microsoft Windows 開発環境を使用してください。

注 - UNIX プラットフォームで `make` を使用している場合は、名前の重複などの混乱を避けるために、コードの様式ごとに異なるディレクトリにコードファイルを生成することをお勧めします。

コード生成時の諸注意

コードを生成する場合に、いくつかの点について考慮しておく必要があります。詳細は以下の各項で説明しますが、各項の概要は次のとおりです

1. ダイアログテンプレートの生成
   Microsoft Windows リソースファイルとしてダイアログを生成する方法について説明します。これによって、Windows 環境でより簡単に操作できるようになります。

2. 拡張 MFC
   生成した Windows アプリケーションで MFC 4 (およびそれ以上) の 3D の見た目と使い心地を使用できるようにする方法について説明します。

3. プロジェクトファイル
   Sun WorkShop Visual が生成する Visual C++ プロジェクトファイルについて説明します。

4. 保存ファイルとコードファイルの同期
   保存ファイルと生成したコードファイルの同期を保つ方法について説明します。

5. ダイアログの点滅
   MFC コードの生成時にダイアログがリアライズ (実体化) される (すなわち表示される) ことを示す警告について説明します。

6. 日本語フォントの使用
   日本語テキストを含む MFC コードを生成する場合の処理方法について説明します。

ダイアログテンプレートの生成

MFC に対してコードを生成する場合、「コードオプション」ダイアログ (「コード生成」ダイアログの下部にある「オプション」ボタンを押すと表示される) に「リソースとして生成」トグルが表示されます (図 11-13 を参照)。このトグルがオンになっていると、Sun WorkShop Visual はデザインにダイアログテンプレートとしてダイアログを生成します。ダイアログテンプレートは、ダイアログを記述する Microsoft Windows のリソースファイルです。Sun WorkShop Visual は、ダイアログテンプレートからウィジェットを取り出してダイアログを制御する MFC コードをいくつか生成します。その結果、完全なアプリケーションが生成されます。「リソースとして生成」トグルがオフになっていると、デザインに対して通常の MFC が生成されます。デフォルトでは、このトグルはオンになっています。

図 11-13 「コードオプション」ダイアログ

ダイアログテンプレートを使用すると、以下の利点があります。

1. フォント設定などのシステムのデフォルトのリソースを使用できます。通常の MFC では使用できません。
2. Microsoft Windows IDE を使用してダイアログの配置を簡単に変更できます。ただし、変更した配置内容を Sun WorkShop Visual に戻すことはできません。

リソースファイル (またはダイアログテンプレート) が生成されるデザインはダイアログをベースにしています。つまり、主要な「フレーム」ウィンドウはありません。そのため、Motif と Microsoft Windows の表示内容は非常に似たものになります。

拡張 MFC

「コードオプション」ダイアログの「MFC 4.X 拡張」トグルを使用すると、Sun WorkShop Visual は MFC の拡張された作成関数を使用するコードを生成し、それによってアプリケーションの見た目と使い心地が 3D (立体的) になります。

「MFC 4.X 拡張」トグルをオンにすると、ビットマップをラベルに、アイコンをボタンに対応付けることもできます。その結果、立体表示される (3D) ボタンとイメージを作成できます。

このように処理されるイメージのサイズは自然に確定されます。そのため、Motif デザインの表現は明確なものになります。

拡張 MFC コードは、「リソースとして生成」トグルを使用しなくても生成できます。

デフォルトでは、「MFC 4.X 拡張」トグルはオンになっています。

「MFC 4.X 拡張」トグルを使ってで生成したコードは MFC 5 でも動作します。

プロジェクトファイル

Microsoft Windows 用にMFC コードを生成する場合、Microsoft Visual C++ で使用するプロジェクトファイルも「コード生成」ダイアログで指定したベース名を使用して生成されます。これらのファイルは次のとおりです (`<generate_filename>` は、「コード生成」ダイアログで指定したベース名を示します)。

- `<generate_filename>.dsw`
  メインプロジェクトファイルです。このファイルは、次に示すファイルの情報を使用します。

- `<generate_filename>.dsp`
  生成したファイルに関する情報が含まれています。

メインプロジェクトファイル (接尾辞「dsw」が付いている) を Microsoft Visual C++ で開いてください。これらのプロジェクトファイルは、Visual C++ のバージョン 5 以降で使用するのに適しています。

保存ファイルとコードファイルの同期

Sun WorkShop Visual では、定義の保存ファイル中にウィジェット ID 番号を含める必要があります。そうすることで名前が付けられていない構成要素の配置を間接的に変更しても正しいコードを生成することが可能となります。ただし、コードが生成される前に、デザイン内でウィジェット ID に影響するような変更 (リセットなど) が行われた場合には、問題が生じる可能性があります。Sun WorkShop Visual は、このような同期の乱れを検出すると、ファイルを保管するようメッセージを表示します。

ダイアログの点滅

Sun WorkShop Visual が正確に配置情報を生成することができるように、ダイアログはリアライズ (実体化) される必要があります。Sun WorkShop Visual は、Microsoft Windows コードが生成される際に、リアライズ (実体化) されていないダイアログを自動的に表示して、すぐに非表示にします。ディスプレイ上でダイアログが瞬間的に表示されることがあるのは、このためです。

日本語フォントの使用

Sun WorkShop Visual で生成した Microsoft Foundation Class (MFC) コードが日本語テキストを適切に扱うためには、そのコードを Microsoft Windows 上でコンパイルする前に特別な処理をする必要があります。この処理は、Sun WorkShop Visual 付属のフィルタユーティリティ、`visutosj` で行います。

`visutosj` は、MFC コード中で使用されるテキスト文字列を EUC から Shift-JIS に変換し、MFC の CFont 作成 (create) メソッドの中の DEFAULT_CHRSET の値を SHIFTJIS_CHARSET に変更します。

`visutosj` の使用方法は以下のとおりです。

```
visutosj [-xf*] [ファイル]
```

引数の意味は以下のとおりです。

表 11-1 引数の意味

`-x`	`visutosj` ユーティリティの概要を表示。
`-f` <フィルタ>	使用する Shift JIS のフィルタを指定 (デフォルトは `jconv`)。
`-*`	任意のオプションを指定可能。指定したオプションは、フィルタへ引き渡されます。
[ファイル]	任意のファイル名。指定のない場合は標準入力が使用されます。

Sun WorkShop Visual の設定

Microsoft Windows モードでも Sun WorkShop Visual に適用されるアプリケーションリソースは、数多く存在します。これらのリソースの 1 つに、Sun WorkShop Visual をデフォルトで Microsoft Windows モードで実行するかどうかを示す、`-windows` オプションがあります。このリソースについては、415 ページの「Microsoft Windows モードでの起動」を参照してください。

生成された行に Ctrl-M を追加

Sun WorkShop Visual は、Microsoft Windows のコードを生成する際にデフォルトで、各行の改行記号の前にキャリッジリターンとして *Ctrl-M* を追加します。この Ctrl-M は MS-DOS で使用するため、自身で作成したファイルを変換プログラムで処理する際に必要になります。各行の末尾に Ctrl-M を追加したくない場合は、Sun WorkShop Visual のリソースファイルを以下のように設定してください。

```
visu.mfcCarriageReturn:false
```

Microsoft Windows ではないリソースフィールドの色の設定

デフォルトでは、Sun WorkShop Visual はリソースパネルにあるフィールド、またはボタンや他のテキストフィールドが Microsoft Windows に適用できない場合には、その要素をピンク色で表示して、適用不可能であることが示されます。この色は、以下に示す Sun WorkShop Visual アプリケーションリソースファイルの行を変更することで、別の色にできます。

```
visu.mfcTextWarningBackground:#ecc9c9eacdda
```

上記の例は、デフォルトの内容 (ピンク色) を示します。この大きな数値の部分は、よりわかりやすい、色の名前に変更することもできます。

ファイル名フィルタの設定

「コード生成」ダイアログのファイルを入力するテキストフィールドには、デフォルトのファイル名フィルタが使用されています。このファイル名フィルタは、アプリケーションのリソースファイルを変更することで、別のものにできます。以下の行を検索してください。

```
visu.c++Filter:  *.c
```

```
visu.c++StubsFilter:*.c
```

これらは、Motif コード (Motif および Motif XP の両方) 生成のためのデフォルトの内容です。Microsoft Windows コード生成のためには、次の 2つのファイル名フィルタがあります。

```
visu.visualC++Filter:*.cpp
```

```
visu.visualC++Filter:*.cpp
```

2 種類のプラットフォーム間でコードを共有する場合には、2 つの異なる様式についてファイル名フィルタが同じになるようにします。両方のプラットフォームに互換性を持つファイル名についての注意事項については、440 ページの「ファイル名」を参照してください。

第 12 章

Microsoft Windows と Motif のアプリケーション作成

はじめに

本章では、Motif と Microsoft Windows の両方で、図 12-1 に示す簡単なアプリケーションを作成する方法について説明します。さらに、 本章の最後の 475 ページの「ソースの共有化」では、コールバックファイルを UNIX と Microsoft Windows で共有する方法を示します。

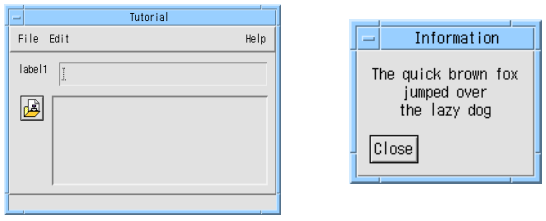


図 12-1 最終のアプリケーション

デザインの開始

この学習では、アプリケーションのメインウィンドウとサブダイアログをまとめる方法について説明します。さらに、ユーザーインタフェースの構築に加えて、リンクやコールバックを使用し、ピックスマップなどのリソースを設定します。また、描画領域にメニューをポップアップする方法も示します。さらに、これらすべてを Microsoft Windows に移植します。

読者は Sun WorkShop Visual の基本的な使用法に精通しているものとします。したがって、この学習用の完成した階層が収められているデザインファイルは、あらかじめ Sun WorkShop Visual に添付されています。手順 2 の説明に従ってこのファイルを開いても、指示に従って階層を自分で構築してもかまいません。

1. Microsoft Windows モードで Sun WorkShop Visual を起動します。

   起動方法については、415 ページの「Microsoft Windows モードでの起動」を参照してください。

2. 自分で階層を構築する時間を節約したい場合には、次のファイルを開いて、454 ページの手順 18 に進んでください。

   ```
   $VISUROOT/src/examples/tutorial/windows.xd
   ```

   ここで、VISUROOT は、インストールした Sun WorkShop Visual のルートディレクトリです。

   注 - 自分で階層を構築する場合には、手順 3～手順 17 を実行してください。

3. アプリケーションシェルから始めます。これにフォームを追加し、フォームの中にメニューバー、ボタン、スクロールウィンドウ、ラベル、およびテキストフィールドを置きます。

   この階層を図 12-2 に示します。

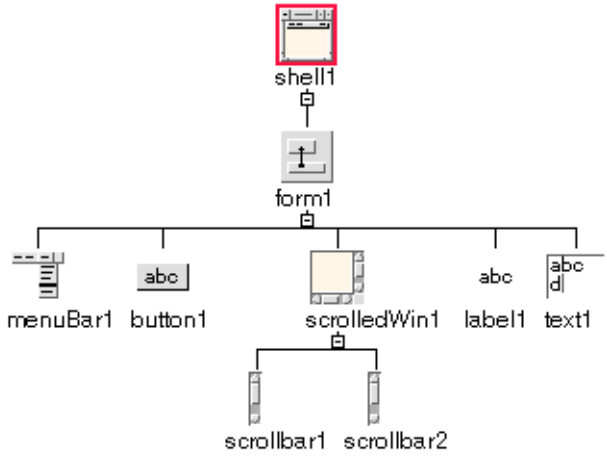


図 12-2 最初の階層

4. シェルに「shell」、メニューバーに「menubar」、スクロールウィンドウに「popup_window」、ボタンに「subd_pixmap」と名前を付けます。

   その他のウィジェットには名前を付ける必要はありません。[ウィジェットクラス][番号] という形式の名前 (たとえば「button1」) が、Sun WorkShop Visual によってデフォルトで割り当てられます。

5. メニューバーに、3 つのカスケードボタンを追加します。それぞれ「file_menu」、「edit_menu」、「help_menu」という名前を付けます。

6. 「file_menu」カスケードボタンにメニューを追加します。このメニューに、2 つのボタン、セパレータ、およびもう 1 つのボタンを追加します。

7. メニューに追加した最後のボタンを「exit_b」という名前を付けます。

8. 「edit_menu」カスケードボタンにメニューを追加します。このメニューに 3 つのボタンを追加します。

9. 「help_menu」カスケードボタンにメニューを追加します。このメニューに 2 つのボタンを追加し、最初のボタンに「subd_b」という名前を付けます。

   メニューバーの完成した階層を図 12-3 に示します。

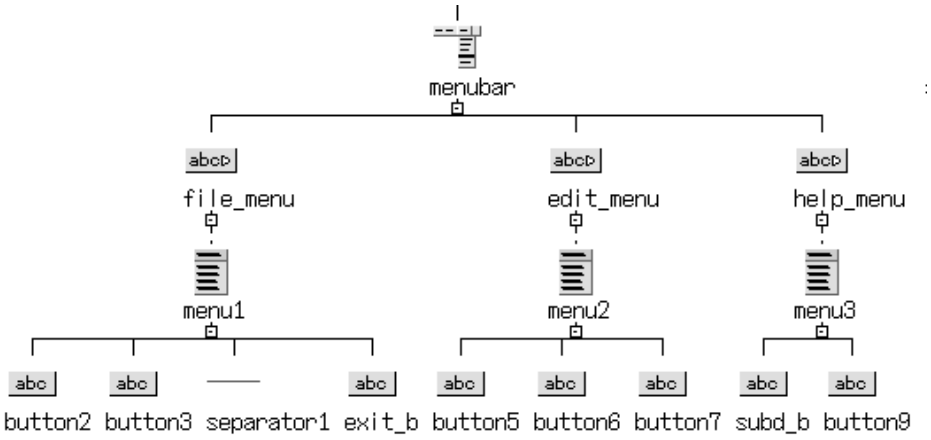


図 12-3 メニューバーの下の階層

10. 「popup_window」スクロールウィンドウに描画領域を追加します。これに「draw_area」という名前を付けます。

11. 描画領域にポップアップメニューを追加し、それに「popup」という名前を付けます。ポップアップメニューに 3 つのボタンを追加します。

    完成したスクロールウィンドウ階層を図 12-4 に示します。

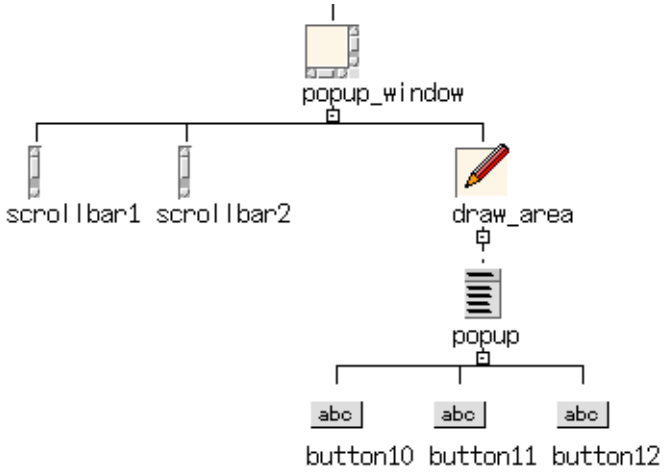


図 12-4 スクロールウィンドウの下の階層

12. ダイナミックディスプレイが図 12-5 のようになるように、フォームの配置エディタを使用して、フォーム内のウィジェットの配置を調整します。

    ウィジェットを適切な場所に移動し、アタッチメントを指定する必要があります。この操作については、109 ページの第 4 章「配置エディタ」を参照してください。

    フォームウィジェットは (まだ C++ クラスにしていないため) Microsoft Windows コードには生成されませんが、アタッチメントと位置については、生成コード内で Sun WorkShop Visual によって計算と処理が行われます。このようにして、サイズ変更動作は変わることなく保持されます。詳細については、428 ページの「マネージャウィジェットとその配置」を参照してください。

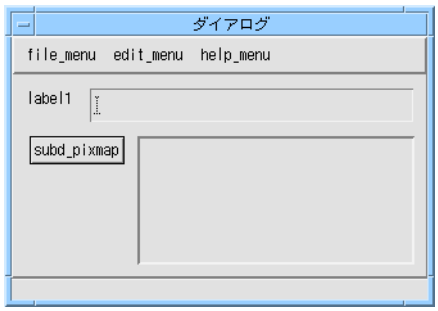


図 12-5 メインウィンドウのダイナミックディスプレイ

13. デザインを `MyWinApp.xd` というファイルに保存します。
14. デザインにもう 1 つのシェルを追加します。今度はダイアログシェルです。これに「sub_shell」という名前を付けます。
15. シェルにフォームを追加して「sub_form」という名前を付けます。
16. フォームにラベルとボタンを追加します。ボタンに「close_b」という名前を付けます。

    このサブダイアログの完成した階層を図 12-6 に示します。

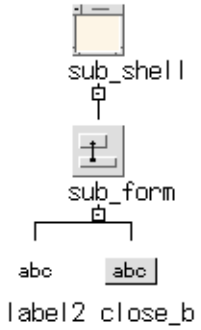


図 12-6 サブダイアログの階層

17. 図 12-7 に示すように、ボタンをダイアログの下部に置き、ダイアログがサイズ変更されたときにラベルのサイズも変更されるように、フォームの配置エディタを使用してウィジェットを配置します。

    ウィジェットを適切な場所に移動し、アタッチメントを指定する必要があります。ダイアログのサイズにあわせてラベルのサイズを変更するには、たとえば、ラベルの下部をボタンの先頭に接続し、他のすべての端をフォームの側面に接続します。この操作については、109 ページの第 4 章「配置エディタ」を参照してください。

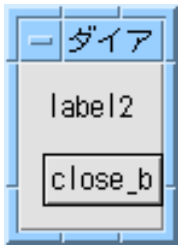


図 12-7 サブダイアログ

18. デザインを保存します。

    450 ページの手順 2 で用意されているデザインファイルを開いた場合でも、この保存操作は行なってください。ここで保存するファイルの名前は `MyWinApp.xd` とします。

コールバックの追加

ここで、サブダイアログを閉じる「close」ボタンにコールバック・メソッドを追加し、メインアプリケーションシェルのファイルメニューの「exit_b」にコールバック・メソッドを追加します。

1. 「close」ボタン「close_b」を選択します。
2. 「コールバック」ダイアログを表示し、コールバックのリストから活性化コールバックを選択します。
3. 定義されたコールバックの下にあるオプションメニューが「メソッドの名前」になっていることを確認します。

   C++ を生成するため、追加対象はメソッドになります。
4. テキストボックスに **`DoClose`** と入力し、「追加」ボタンを押します。
5. 「コールバック」ダイアログを閉じます。
6. ウィンドウ保持領域で「shell」というシェルを選択します。
7. ファイルメニュー (file_name) の下にある「exit_b」というボタンを選択します。
8. 「コールバック」ダイアログを表示し、コールバックのリストから活性化コールバックを選択します。
9. 定義されたコールバックのリストの下にあるオプションメニューが「メソッドの名前」になっていることを確認します。
10. テキストボックスに `DoExit` と入力し、「追加」ボタンを押します。
11. 「コールバック」ダイアログを閉じます。
12. デザインを保存します。

注 - Motif 専用のアプリケーションでは、ボタンからアプリケーションを閉じるコールバックを追加する必要はありません。sub_form の「自動アンマネージ」リソースを「はい」に設定するだけで十分です。一方、Microsoft Windows の場合には、コールバックを追加する必要があります。

リンクの追加

リンクを追加して、メインウィンドウからサブダイアログが表示されるようにします。Sun WorkShop Visual では、Motif と Microsoft Windows の両方で機能するリンクを実装するコードを生成できます。リンクを作成するときに考慮すべき Microsoft Windows での制限事項については、427 ページの「リンクの使用」を参照してください。ダイアログでのリンクの設定方法については、212 ページの「リンク」を参照してください。

1. メインアプリケーションシェル「shell」の「help_menu」というメニューで「subd_b」を選択し、「リンク編集」ダイアログを表示します。

   これを行うには、ツールバーのボタンを押すか、または「ウィジェット」メニューから「リンク編集」を選択してください。ツールバー上のリンク編集ボタンについては、16 ページの図 2-1 を参照してください。

   なお、宛先ウィジェットとしてメニューボタンが選択されているときには、追加ボタンがピンク色になります。表示、非表示、マネージ、およびアンマネージのリンクは、Motif のメニューボタンでのみ有効です。有効化と無効化のリンクは、Motif と Microsoft Windows の両方のメニューボタンで有効です。ただし、現在選択されているメニューボタンは、この学習での宛先ウィジェットではありません。

2. ウィンドウ保持領域で「sub_shell」を選択します。

   「リンク編集」ダイアログ内の「ウィジェット」というテキストボックスには、About ダイアログのシェルである「sub_shell」が追加されています。

3. 「リンク編集」ダイアログで「表示」リンクを選択し、「追加」ボタンを押して「sub_shell」に表示リンクを追加します。

4. 「リンク編集」ダイアログを閉じます。

   次に、メインアプリケーションシェルの「subd_pixmap」から同じリンクを追加します。

5. メインアプリケーションシェル「shell」に戻ります。

   ウィンドウ保持領域からシェルを選択します。

6. 「subd_pixmap」を選択し、「リンク編集」ダイアログを再表示します。

   選択したウィジェットが「リンク元」ウィジェットとなるよう、ダイアログを再表示する必要があります。さもなければ、これは「リンク先」ウィジェットとなります。

7. ウィンドウ保持領域で「sub_shell」を再び選択します。

8. 「リンク」ダイアログで「表示」リンクを選択し、「追加」ボタンを押して「sub_shell」に表示リンクを追加します。
9. 「リンク」ダイアログを閉じて、デザインを保存します。

ポップアップメニュー

どの言語で生成する場合も、描画領域ウィジェットにポップアップメニューを追加すると便利です。学習の次の段階では、その追加方法を示します。Sun WorkShop Visual ではメニューをポップアップするコードを自動生成しないため、特別なコードを少し記述する必要があります。Motif と Microsoft Windows ではメニューのポップアップ方法が異なるため、プラットフォームごとにコードを変えることが必要です。コードの追加については後述します。最初に、コールバックを設定しなければなりません。

1. メインアプリケーションシェルのスクロールウィンドウ「popup_window」の中から「draw_area」を選択します。
2. 「コールバック」ダイアログを表示します。
3. コールバックのリストから「入力」を選択し、**`DoInput`** というコールバックメソッドを追加します。

   「入力」コールバックタイプにはアスタリスク (*) が付いており、このコールバックは Microsoft Windows 版のアプリケーションに使用できないことを示します。
4. 「コールバック」ダイアログを閉じます。
5. 図 12-8 に示す描画領域リソースパネルを表示します。

   表示方法としてには、描画領域上でダブルクリックするのが最も簡単です。

   Motif XP ライブラリには、Microsoft Windows の描画モデルや入力モデルを模倣したものはありません。したがって、Microsoft Windows のリストはメッセージハンドラを対象としています。

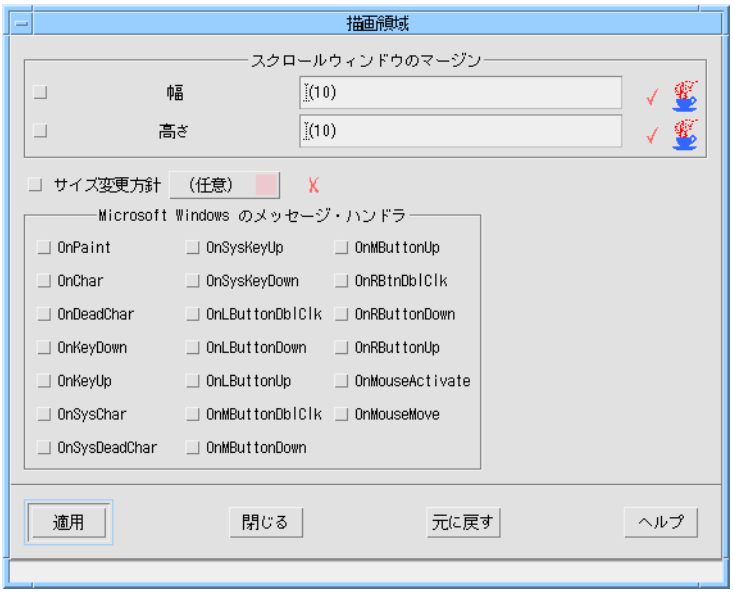


図 12-8 描画領域リソースパネル

6. 「OnRButtonDown」トグルをオンにし、「適用」を押してリソースパネルを閉じます。

   これにより、対応する Microsoft Windows メッセージへの応答で呼び出される、適切な Microsoft Windows メッセージハンドラと、該当するコールバックスタブが生成されます。

   メニューをポップアップするためのスタブの記入方法については、467 ページの「スタブへの記入」を参照してください。

7. デザインを保存します。

リソースの設定

Sun WorkShop Visual は、Motif と Microsoft Windows に対するリソースファイルを生成します。Motif では、そのウィジェットによる制御が広い範囲で可能です。Microsoft Windows でのリソースも存在しますが、Motif/X リソースに比べると制御可能な範囲が限られています。Microsoft Windows リソースは、実行可能ファイルに

コンパイルして組み込まれます。したがって、Motif の場合とは異なり、Microsoft Windows でリソースを変更するということは、アプリケーションを再コンパイルすることになります。

ラベルリソースの設定

このアプリケーション例には、文字列、ピックスマップ、およびキーボードのリソースがあります。最初に、ラベル、ボタン、およびシェルの文字列を設定します。

1. メインアプリケーションシェルの子であるフォームを選択します。
2. フォームのリソースパネルを表示します。
3. 「タイトル」というラベルの付いたテキストフィールドに「Tutorial」と入力して、ダイアログのタイトルを設定します。
4. 「適用」を押してリソースパネルを閉じます。
5. 「file_menu」カスケードボタンのリソースパネルを表示し、それにラベル文字列「File」を指定します。「適用」を押します。
6. 「edit_menu」にラベル「Edit」が表示され、「help_menu」にラベル「Help」が表示されるよう、他の 2 つのカスケードボタンにも同じ操作を行います。

   これを図 12-9 に示します。

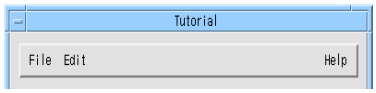


図 12-9 メニューバーラベルとシェルタイトル

7. 「file_menu」のボタンのラベルに「New」、「Open」、「Exit」を設定します。
8. 「edit_menu」のボタンのラベルに「Cut」、「Copy」、「Paste」を設定します。
9. 「help_menu」のボタンのラベルに「About...」、「Help」を設定します。

   これら 3 つのメニューを図 12-10 に示します。

図 12-10 メニューの項目ラベル

10. 描画領域の子であるポップアップメニューから最初のボタンを選択し、そのリソースパネルを表示します。
11. ボタンのラベルを「Cut」に設定します。
12. ポップアップメニュー内の他の 2 つのボタンをそれぞれ選択して、それらのラベルをそれぞれ「Copy」、「Paste」に設定します。
13. ウィンドウ保持領域内のダイアログシェル「sub_shell」を選択します。

    現在、デザイン領域にはこのサブダイアログの階層が表示されています。
14. ラベルを選択し、そのリソースパネルを表示します。
15. 次のような行を表示するよう、ラベルウィジェットの文字列を設定します。

    ```
    The quick brown fox
    jumped over
    the lazy dog
    ```
16. ラベルの文字列を中央揃えするには、リソースパネルの「設定」ページに移動して、「揃え方」オプションメニューを「中央揃え」に変更します。「適用」を押します。
17. ボタンのラベルを「Close」に設定します。
18. サブダイアログのタイトルに「Information」を設定します。

    そのためには、シェルの子であるフォームのリソースパネルで「タイトル」というテキストボックスに入力してください。完成したダイアログを図 12-11 に示します。

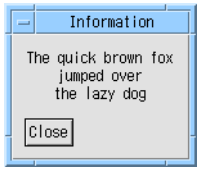


図 12-11 完成したサブダイアログ

デザインを保存します。

文字列リソースの生成

これらすべてのラベルは文字列リソースです。Motif 版のデザイン用のコードを生成する場合には、これらのリソースを生成する場所 (コード内またはリソースファイル内) を決めることができます。Microsoft Windows 版の場合には、すべての文字列リソースは自動的にソースコード内に生成されます。

ピックスマップの使用

デザインにピックスマップを使用している場合には、ピックスマップはコード生成時に、Sun WorkShop Visual によって Windows のビットマップに自動変換されます。このことを示すため、ボタンの 1 つにピックスマップラベルを指定します。

1. メインアプリケーションシェルで「subd_pixmap」というボタンを選択し、そのリソースパネルを表示します。
2. ボタンの「種類」リソースをピックスマップに設定します。

   これはボタンリソースパネルの「設定」ページにあります。
3. ボタンリソースパネルの「表示」ページに移動し、「ピックスマップ」というボタンを押します。

   これにより、ピックスマップセレクタのダイアログが表示されます。
4. ピックスマップセレクタのダイアログでは、既存のピックスマップを選択するか、または「編集」を押してピックスマップエディタを表示します。
5. ピックスマップエディタで新しいピックスマップを作成している場合には、上部のテキストフィールドにその名前を入力し、「バインド」を押してください。ピックスマップエディタを閉じます。

6. ピックスマップセレクタのダイアログで「適用」を押します。リソースパネルの「ピックスマップ」というボタンの横のテキストフィールドにオブジェクト名が表示されます。

   ピックスマップオブジェクトの作成方法と使い方については、167 ページの「ピックスマップの選択」を参照してください。

7. リソースパネルで「適用」を押すと、指定したピックスマップがボタンに表示されます。

8. デザインを保存します。

ピックスマップの命名

MS-DOS (および Microsoft Windows) の場合のファイル名は、拡張子の前部分が 8 バイトを超えてはならないことに注意してください。Sun WorkShop Visual は、Microsoft Windows モードでは、ピックスマップをバインドした名前がファイルのベース名として使われます。そのため、ファイル名が 8 バイトを超えないように注意してください。

以上でアプリケーションのデザインが終了しました。以降の学習では、UNIX と Microsoft Windows 用のコードの生成方法、および各プラットフォームでのアプリケーションの構築方法を示します。

アプリケーションの構築

Sun WorkShop Visual でのアプリケーション・ユーザーインタフェースのデザインが終わったので、Motif および Microsoft Windows コードを生成することができます。コードを生成したら、2 つのアプリケーションの構築準備ができたことになります。この節では、コールバックスタブへの記入方法も含めて、操作の詳細について説明します。

ソースの管理

この学習では、Motif 用と Microsoft Windows 用に 2 つの異なるアプリケーションを生成する必要があります。これらのアプリケーションは、上書きしたり、不正なヘッダーファイルをインクルードすることのないよう、2 つの別のディレクトリに生成してください。この学習では、ソースファイルの共有化は行いません。これについては 475 ページの「ソースの共有化」を参照してください。

Motif 用のコード生成

1. 「コード生成」ダイアログを表示します。

   ファイル名フィールドには、Sun WorkShop Visual の保存ファイル名に基づく名前が与えられています。

2. 「言語」オプションで「C++ (Motif XP)」が選択されていることを確認します。

   「コード生成」ダイアログの最上部には、ベースディレクトリの名前を入力するテキスト領域があります。すべてのファイル名は、このベースディレクトリからの相対パスで入力します。ベースディレクトリを確認するには「...」ボタンを使用してください。

3. コードファイルを生成したい領域をベースディレクトリに設定します。

   混乱を避けるには、Motif XP コードと Microsoft Windows MFC コードを別々のディレクトリに生成してください。

4. 「コード」ファイルに「生成」トグルが設定されていることを確認してください。

5. 「C++ コードのオプション」ダイアログを開きます。

   「コード」テキストボックスの横にある「オプション」というボタンを押してください。

6. 「ヘッダファイルをインクルード」トグルをオンにします。

7. 「Motif ヘッダファイルをインクルード」トグルがオンになっていることを確認します。

8. 「了解」ボタンを押します。

   変更が保存され、「C++ コードのオプション」ダイアログが閉じます。

9. 「外部宣言」の横にある「生成」トグルがオンになっていることを確認します。

10. 「スタブ」ファイルの横にある「生成」トグルをオンにします。

11. 「メインプログラム」テキストボックスの横にある「生成」トグルがオンになっていることを確認します。

12. 「コードオプション」ダイアログを表示します。

    そのためには、「生成」ダイアログの最下部にある、「オプション」というラベルのついたボタンを押します。

13. 「リンク」オプションメニューが「コードに生成」に設定されていることを確認します。

14. 「文字列」オプションメニューを「コード」に変更し、その他のすべてのリソースが「コード」に設定されていることを確認します。

    便宜上、この学習ではすべてのリソース設定が「ハードコード」されます。実際のアプリケーションでは、リソース設定を別個のリソースファイル内に生成するのが普通です。

15. 「了解」ボタンを押します。

    変更が保存され、「コードオプション」ダイアログが閉じます。

16. 「メークファイル」の横にある「生成」トグルをオンにします。

17. 「コード生成」ダイアログの「生成」ボタンを押します。

    ファイルは選択したディレクトリ内に生成されます。

18. デザインを保存します。これにより、「コード生成」ダイアログ内の設定が保存されます。

メークファイル

Sun WorkShop Visual のインストール方法と構成方法によっては、Motif XP コードにアクセスするためにメークファイル (Makefile) を編集する必要が生じる場合もあります。XPCLASSLIBS および CCFLAGS の定義が Motif XP ライブラリとインクルードファイルをアクセスすることを確認してください。VISUROOT は、Sun WorkShop Visual のインストールディレクトリのルートへのパスです。

```
XPCLASS = $(VISUROOT)/src/motifxp

XPCLASSLIBS = $(XPCLASS)/lib${ABIDIR}/libmotifxp.a

GEN_CFLAGS=-I. ${XINCLUDES} -I${XPMDIR} ${GROUP_COMPILEFLAGS}

CCFLAGS=${CFLAGS} ${ABICCFLAGS} -I${XPCLASS}/h ${GROUP_COMPILEFLAGS}
```

Microsoft Windows 用のコード生成

Microsoft Windows 用のコード生成手順は、上記の Motif 用の手順とほとんど同じですが、一部が異なっています。Sun WorkShop Visual は、様式ごとに異なる一連のファイルをそれぞれ記憶するため、異なる様式に対して一度ファイルの名前を指定すると、ツールバーの様式メニューおよびコード生成ボタンを使用することができます。

PC の場合には、ファイル名の拡張子前部分は 8 バイト以下であることが必要です。Visual C++ などいくつかの IDE (統合開発環境) では、C++ コードを含むソースファイルの拡張子として `.c` を指定しても受け入れられません。`.cpp` または `.cxx` を指定してください。

1. まだ画面に開いていない場合には、「コード生成」ダイアログを表示させます。
2. 「言語」オプションメニューから「C++ (Microsoft Windows MFC)」が選択されていることを確認します。
3. 「コード生成」ダイアログの最上部にあるベースディレクトリには、ファイルを生成したいディレクトリを設定します。

   混乱を避けるには、Motif XP コードと Microsoft Windows MFC コードを別々のディレクトリに生成してください。
4. 「コード」ファイルの横にある「生成」トグルがオンであることを確認します。
5. 「C++ コードのオプション」ダイアログを開きます。

   「コード」テキストボックスの横にある「オプション」というボタンを押します。
6. 「ヘッダファイルをインクルード」トグルをオンにします。
7. 「了解」ボタンを押します。

   変更が保存され、「C++ コードのオプション」ダイアログが閉じます。
8. 「外部宣言」ファイルと「スタブ」ファイルの「生成」トグルがオンになっていることを確認します。
9. 「メインプログラム」テキストボックスの横にある「生成」トグルがオンになっていることを確認します。

10. 「Microsoft Windows リソース」ファイルの横にある「生成」トグルをオンにします。

    このファイルにはダイアログテンプレートが含まれています。このトグルがオンにされると、432 ページの「ピックスマップ、 ビットマップ、アイコン」で説明するように、Sun WorkShop Visual は、ピックスマップを Windows ビットマップに (ファイルごとに 1 つずつ) 変換します。

11. 「コードオプション」ダイアログを表示します。

    「コードオプション」ダイアログの下部にある「オプション」ボタンを押します。

12. 「コードオプション」ダイアログで、「リソースとして生成」トグルがオンになっていることを確認します。

    これにより Sun WorkShop Visual は、442 ページの「ダイアログテンプレートの生成」で説明するように、Microsoft Windows リソースファイルであるダイアログテンプレートを生成します。

13. MFC 4 または 5 を使用している場合には、「MFC 4.X 拡張」トグルをオンにします。

    これにより 3D の見た目と使い心地が得られます。

14. 「リンク」のオプションメニューとすべてのリソースが「コードに生成」に設定されていることを確認します。

15. 「了解」ボタンを押します。

    変更が保存され、「コードオプション」ダイアログが閉じます。

注 - Microsoft Windows コードの場合には、メークファイルを生成する必要はありません。これらのファイルは別の方法で構築されます。これについては、470 ページの「Microsoft Windows 版のコンパイル」を参照してください。

---

16. 「コード生成」ダイアログで「生成」を押します。

    ピックスマップ用のビットマップファイルが生成されたことを知らせるメッセージが表示されます。ファイルのベース名は、ピックスマップオブジェクトの名前です。

生成されるファイル

MFC を生成すると、コード、スタブ、およびヘッダーファイルに加えて、特別なファイルも生成されます。これらのファイルは次のとおりです。

1. ビットマップファイル
   ピックスマップオブジェクトを作成した場合には、それらのオブジェクトは、ピックスマップごとに 1 つの Windows ビットマップファイルに変換されます。これらのファイルの接尾辞は「.ico」です。

2. プロジェクトファイル
   これらは Visual C++ バージョン 5 (以降) のファイルであり、メークファイルと同様に、アプリケーションの構築場所である Visual C++ にアプリケーションを読み込むために必要なすべての情報が含まれています。これらのファイルの接尾辞は「.dsw」と「.dsp」です。詳細については 444 ページの「プロジェクトファイル」を参照してください。

日本語テキスト

生成したコードに日本語テキストが含まれる場合には、そのコードを PC に転送する前に、Sun WorkShop Visual に付属のフィルタユーティリティ xdtosj を使用して、コードの後処理を行う必要があります。詳細については、445 ページの「日本語フォントの使用」を参照してください。

スタブへの記入

スタブファイルには、Motif 用と MFC 用の 2 つがあります。この学習では Motif アプリケーション用に Motif XP コードを生成したため、いくつかのコードを共有できます。これは、Sun WorkShop Visual の XP ライブラリが Motif 用に MFC の一部を模倣しているためです。しかし、単純化と速度向上のため、この学習ではこの 2 つを別個に扱います。ソースの共有の詳細については、475 ページの「ソースの共有化」を参照してください。

MFC スタブ

コールバックには､アプリケーションを終了するためのものと､描画領域からメニューをポップアップするためのものがあります｡スタブは下記のように記入してください。

```
void

shell_c::DoExit ( )

{
```

この行を追加 ⟶

```
                exit(0);
}

void
sub_shell_c::DoClose ( )
{
```

この行を追加 ⟶

```
                this->ShowWindow(SW_HIDE);
}

afx_msg void popup_window_c::OnRButtonDown( UINT nFlags, CPoint point
)
{
```

これらの行を追加

```
                ClientToScreen(&point);
                popup->TrackPopupMenu(
TPM_LEFTALIGN|TPM_RIGHTBUTTON,
                                        point.x, point.y, this,
NULL);
}
```

Motif スタブ

MFC スタブファイルには、Motif スタブファイルに合わせて 2 つのコールバックがあります。終了コールバックのシグニチャーは両方のプラットフォームで同じですが、内容は異なっています。

```
void
shell_c::DoExit ( )
{
```

この行を追加 ⟶

```
                exit (0);
}

void
```

```
sub_shell_c::DoClose ( )
{
この行を追加 ──→            this->ShowWindow(SW_HIDE);
}

void
popup_window_c::DoInput ( )
{
この行を追加 ──→            popup->TrackPopupMenu(0, 0, 0, this, NULL);
}
```

アプリケーションのコンパイル

デザインを作成し、コードを生成し、コールバックスタブを記入すると、Motif および Microsoft Windows のどちらでもアプリケーションをコンパイルできます。以降の節では、Motif アプリケーションの構築に関する簡単な説明の後に、Microsoft Windows 上でアプリケーションを構築する手順について記述します。

Microsoft Windows 上で Visual C++ バージョン 4.0 および 5 を使用してアプリケーションを構築する手順の詳細については、470 ページの「Microsoft Windows 版のコンパイル」で説明します。ただし、Microsoft Foundation Class を統合できる Symantec C++ などの C++ 開発環境であれば、生成コードを構築することができます。Microsoft Windows ベースの C++ コンパイラでアプリケーションを構築する方法に関する一般説明については、474 ページの「他のアプリケーションの使用」を参照してください。

Motif 版のコンパイル

すでにコードが生成されているため、Motif コードを生成したディレクトリのコマンドプロンプトから

make

と入力するだけで、アプリケーションを作成できます。メークファイルが生成してあるため、アプリケーション内のすべてのファイルが自動構築されます。

アプリケーションを実行する前に、251 ページの「X リソースファイルの設定」で説明したように、読み込むリソースファイルを準備してください。

Microsoft Windows 版のコンパイル

Visual C++ バージョン 5 を使用している場合には、Sun WorkShop Visual がユーザーに代わって Visual C++ のプロジェクトファイルをすべて生成するため、PC 上でのアプリケーションのコンパイルはきわめて簡単です。Visual C++ の古いバージョンを使用している場合には、プロジェクトファイルを作成する必要があります。これはあまり便利ではありませんが、操作は短時間で終わり、一度行うだけで済みます。Visual C++ バージョン 5 と Visual C++ バージョン 4.0 を使用するための詳細を以下に示します。Visual C++ を使用していない場合には、アプリケーションをコンパイルするためのヒントとして 474 ページの「他のアプリケーションの使用」を参照してください。

Visual C++ についての簡単な注意

Visual C++ は、Microsoft Windows 上で実行するアプリケーションの開発を支援する IDE であり、コンパイラ、デバッガ、および各種のエディタなど、多数のツールから構成されます。これは、アプリケーションのソースを構築、デバッグ、制御するための便利なツールです。しかし、Visual C++ でのコード変更は、Sun WorkShop Visual には反映できません。Visual C++ では、プロジェクトの概念を使用しています。アプリケーション内のファイルの管理に Visual C++ を使用する場合には、必ずプロジェクトが必要となります。バージョン 5 の場合には、Sun WorkShop Visual がユーザーに代わってプロジェクトファイルを生成します。

注 - サンから入手できる PC-NFS は、PC と Solaris システム間のファイル共有を簡単にするために設計されたツールです。

Visual C++ バージョン 5

Sun WorkShop Visual は、バージョン 5 以降の Visual C++ に対して、フル構成のプロジェクトファイルを生成します。これらは、「.dsw」と「.dsp」の接尾辞で終わるファイルです。これらのファイルは、すべての MFC ソースファイル (コードファイル、リソースファイル、ビットマップファイル、ヘッダーファイルを含む) と一緒に PC で使用してください。

1. Windows 環境下の PC で、Sun WorkShop Visual によって生成された「.dsw」ファイルをダブルクリックします。

   これによって、Sun WorkShop Visual は、プロジェクトを読み込んだ状態で Visual C++ を開きます。

2. 「構築」メニューから「Build <プロジェクト名>.exe」を選択します。

   Visual C++ がエラーを検出した場合には、エラーが表示され、コンパイルが停止します。F4 キーを押すと、エラーを含むファイルが開き、エラーに関連のある場所が報告されます。続けて F4 キーを押すと、次のエラーへと進み、関連のあるファイルが必要に応じて開きます。エラー項目をダブルクリックすると、関連するファイルが開き、エラーの発生箇所にカーソルが移動します。このようなファイルを表示するウィンドウでは、直接ファイルの内容を編集することもできます。「ファイル」メニューから「開く」を選択しても、ファイルの表示と編集を行うことができます。

3. アプリケーションを正しく構築したら、それを試してみることができます。「構築」メニューから「Execute <プロジェクト名>.exe」を選択します。

Visual C++ バージョン 4.0

Visual C++ バージョン 4 を使用している場合には、Sun WorkShop Visual によって生成されるプロジェクトファイルを使用できません。プロジェクトファイルの作成に必要な手順を以下に説明します。

1. すべての MFC ソースファイルを PC に取り込みます。

   「.dsp」ファイルと「.dsw」ファイルに関しては、Visual C++ バージョン 5 以降にだけ関連するファイルであるため、取り込む必要はありません。

2. Visual C++ を起動します。

3. 「ファイル」メニューから「新規」を選択します。

   下に示すように、作成できる新規項目のリストが表示されます。

図 12-12 新規項目リスト

4. 項目リストから「プロジェクトワークスペース」を選択し、「OK」を押します。

   「新規プロジェクトワークスペース」ダイアログが表示されます。

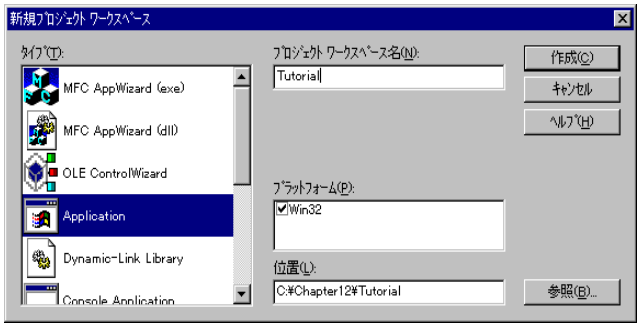


図 12-13 「新規プロジェクトワークスペース」ダイアログ

5. 「種類」リストから「アプリケーション」を選択し、「名前」フィールドに Tutorial などの名前を入力します。

6. 「作成」ボタンを押して、新規アプリケーションを作成します。

   Visual C++ のメインウィンドウが表示されます。次に、構築したいファイルをプロジェクトに追加します。

7. 「挿入」メニューから「ファイルをプロジェクトへ」を選択します。

   次のようなダイアログが表示されます。

図 12-14 「ファイルを挿入」ダイアログ

8. PC にコピーしたファイルのあるディレクトリを探し出します。
9. リストの最初のファイルをクリックし、Shift キーを押しながらリストの最後のファイルをクリックすると、必要なファイルが強調表示されます。
10. 「了解」ボタンを押して、選択したファイルをプロジェクトに追加します。
11. 「構築」メニューから「設定」オプションを選択します。

    次に示すような「プロジェクト設定」ダイアログが表示されます。

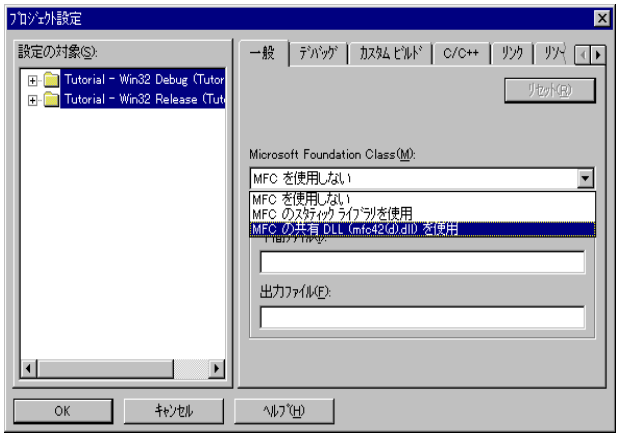


図 12-15 「プロジェクト設定」ダイアログ

12. 上に示すように、「Microsoft Foundation Class」オプションメニューから「MFC の共有 Dll を使用」オプションを選択し、「OK」を押します。

    最後に、次のようにしてアプリケーションを構築します。

13. 「構築」メニューから「Build Tutorial.exe」を選択します。

    Visual C++ がエラーを検出した場合には、エラーが表示され、コンパイルが停止します。F4 キーを押すと、エラーを含むファイルが開き、エラーに関連のある場所が報告されます。続けて F4 キーを押すと、次のエラーへと進み、関連のあるファイルが必要に応じて開きます。エラー項目をダブルクリックすると、関連するファイルが開き、エラーの発生箇所にカーソルが移動します。このようなファイルを表示するウィンドウでは、直接ファイルの内容を編集することもできます。「ファイル」メニューから「開く」を選択しても、ファイルの表示と編集を行うことができます。

14. アプリケーションを正しく構築したら、それを試してみることができます。「構築」メニューから「Execute <プロジェクト名>.exe」を選択します。

他のアプリケーションの使用

Sun WorkShop Visual によって生成されたコードは MFC コードです。これは特定の Microsoft Windows アプリケーション用に生成されたものではありません。Visual C++ を使用していない場合、Microsoft Windows 上でユーザーインタフェースを構築する際に注意すべき重要なポイントを以下に示します。この節はすべてのコンパイラに適用されます。

1. C++ コンパイラ、および Microsoft Foundation Class (MFC) のインクルードファイルとライブラリをシステムにインストールする必要があります。
2. Microsoft Windows の .EXE ファイルを構築するための構築ツールを設定します。
3. コンパイラには MFC ヘッダーファイルへの有効なインクルードパスが含まれていることを確認します。
4. コンパイラには、Sun WorkShop Visual が生成したソースファイルを含んでいるサブディレクトリへの有効なインクルードパスが含まれていることを確認します。
5. ラージ・メモリーモデル設定を使用して、コードをコンパイルします。

   リンカーが MFC ライブラリまたは DLL でリンクすることを確認します。

ソースの共有化

Sun WorkShop Visual の Motif XP ライブラリを使用すると、一部のコールバックコードを Motif と MFC で共有することができます。Motif で使用できるクラスについては、957 ページの付録 B「Motif XP リファレンス」を参照してください。この付録では、Motif と MFC の両方のプラットフォームで使用できる、MFC ツールキットへの呼び出しについて説明しています。

XP ライブラリを使用するには、Motif アプリケーション用に生成する言語として「コード生成」ダイアログで「C++ (Motif XP)」を選択してください。コードを生成する際には、UNIX と Windows の両方のプラットフォームで受け入れられる接尾辞は「.cxx」または「.cpp」だけであることを忘れないでください。以下の手順リストでは、構築済みのデザインとスタブファイルを使用します。このスタブファイルは、Motif XP を使用して Motif と Windows の両方で使用できるコールバックファイルを記述する方法を示すために、すでに記入されているものです。このために必要なファイルは `$VISUROOT/src/examples/tutorial` にあります。ここで、VISUROOT は Sun WorkShop Visual のインストールディレクトリです。

1. Windows モードで Sun WorkShop Visual を起動します。

   この詳細については、415 ページの「Microsoft Windows モードでの起動」を参照してください。

2. 「ファイル」メニューから「開く」を選択し、次のファイルを読み込みます。

   ```
   $VISUROOT/src/examples/tutorial/singlesource.xd
   ```

3. デザインが図 12-16 に示すようになっていることを確認します。

   これは、2 つのオプションメニュー、2 つのトグル、およびテキストフィールドから構成される簡単なダイアログです。

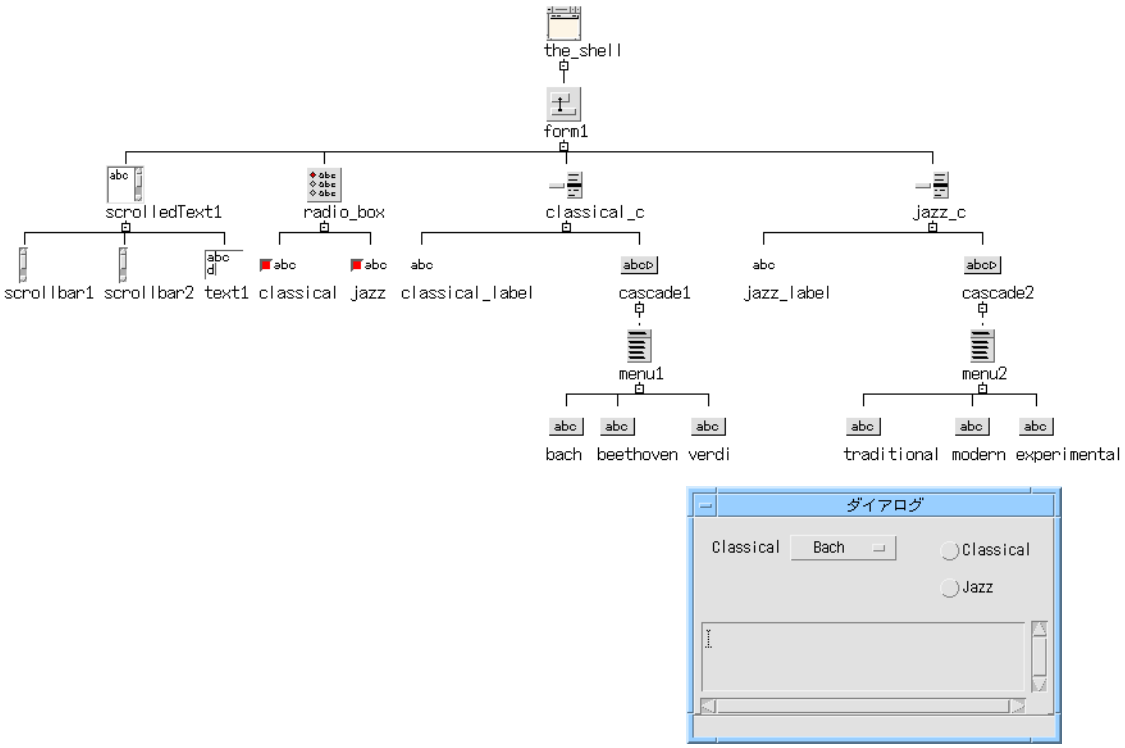


図 12-16 デザイン階層とダイナミックディスプレイ

4. 「コード生成」ダイアログを表示し、生成する最初の言語として「C++ (Motif XP)」を選択します。

   Motif 専用ソースのターゲットディレクトリがあることを確認します。

5. コードファイル、外部宣言ファイル、メインコードファイル、メークファイル、およびスタブファイルを生成します。「コード生成」ダイアログに表示されている名前を使用します。

   現在のスタブファイルは設定済みのスタブファイルによって上書きされますが、Sun WorkShop Visual を使用してメークファイル内にその依存情報を作成する必要があります。

6. 「言語」を「C++ (Microsoft Windows MFC)」に変更します。ターゲットディレクトリも変更することを忘れないでください。

7. コードファイル、外部宣言ファイル、メインコードファイル、および Windows リソースファイルを生成します。スタブファイルは生成しません。

   前述の同じ設定済みのスタブファイルを使用しますが、Sun WorkShop Visual を使用して、Visual C++ プロジェクトファイル内にその依存情報を作成する必要があります。

8. 以下のスタブファイルを Motif XP ディレクトリにコピーします。

   ```
   $VISUROOT/src/examples/mfc/singlesource_stubs.cpp
   ```

   このファイルは、生成した MFC ソースと一緒に PC にコピーする必要があることを忘れないでください。

   テキストエディタでスタブファイルを見ることができます。3 つのコールバックがあります。

   1. JazzChanged
      「Jazz」トグルの状態をチェックし、オンになっている場合は「Jazz」オプションメニューを表示し、「Classical」オプションメニューを非表示にします。

   2. ClassicalChanged
      「Classical」トグルの状態をチェックし、オンになっている場合は「Classical」オプションメニューを表示し、「Jazz」オプションメニューを非表示にします。

   3. DoSetText
      目に見えるオプションメニューから選択した項目を取り出し、テキストをテキストフィールドに書き込みます。

   このコードはすべて、Motif XP アプリケーションと MFC アプリケーションによって共有できます。

9. Motif アプリケーションをコンパイルするには、Motif 用のディレクトリで、コマンドプロンプトから次のように入力します。

   `make`

   コードの生成、コンパイル、および実行の詳細については、241 ページの第 7 章「コードの生成」を参照してください。

10. アプリケーションの構築後、それを Motif 上で実行して、コールバックが機能することを確認してください。2 つのトグルのオン / オフ、およびオプションメニューから何かを選択してみてください。

11. MFC アプリケーションを構築するには、次のようにします。

a. MFC ディレクトリに生成されたすべてのファイルを PC に取り込みます。

   Visual C++ バージョン 5 を使用している場合には、「.dsw」ファイルと「.dsp」ファイルを取り込むことを忘れないでください。これらは、Visual C++ のプロジェクトファイルです。

b. 設定済みのスタブファイルを Motif XP ディレクトリから PC に取り込み、最初のスタブファイルを上書きします。このスタブファイルは、強制的に正しい依存関係にすることだけを目的に生成されたものです。

c. Visual C++ を使用していない場合には、PC 上でのコンパイル時の注意点について 474 ページの「他のアプリケーションの使用」を参照してください。

d. PC 上では、プロジェクトワークスペースとして Visual C++ で「.dsw」ファイルを開きます。

   この場合には、Visual C++ バージョン 5 を使用しているものと想定します。これより古いバージョンや別のアプリケーションを使用している場合には、470 ページの「Microsoft Windows 版のコンパイル」を参照してください。

e. プロジェクトを構築します。

f. プロジェクトを実行します。

g. コールバックが機能することを確認します。トグルのオン / オフを切り替えたり、オプションメニューから何かを選択してみてください。

プラットフォーム間でソースを共有するその他の方法

コールバックファイルのソースを共有化するには、この他に 2 つの方法があります。

1. Java の使用

2. 取得/設定スマートコードの使用

Java プログラム言語を使用すれば、プラットフォームからの独立したコードを生成できます。Sun WorkShop Visual では、デザインを Java コードでも生成できます。さらに、デザインを Java アプリケーションビルダーである Visaj に取り込める形で保存できます。デザインから Java アプリケーションを作成する方法の詳細については、363 ページの第 10 章「Java 用のデザイン」を参照してください。

取得/設定スマートコードにより、特定のウィジェットの周りにツールキットから独立した「ラッパー」が提供されます。簡単な学習を含む詳細については、559 ページの第 16 章「取得と設定用のスマートコード」を参照してください。

第13章

デザインツール

はじめに

Sun WorkShop Visual には、ユーザーインタフェースを構築し、そのインタフェースをアプリケーションとリンクする作業を支援するツールが多数あります。本章では、AppGuru および Sun WorkShop Visual 捕獲機能について説明します。これらのツールは強力な独立したユーティリティで、Sun WorkShop Visual とともに機能し、デザイン作成の効率化に役立ちます。AppGuru (アップグルー) によって新規デザインの基礎を構築し、Sun WorkShop Visual 捕獲機能によって既存アプリケーションのデザインを取り込むことができます。

Sun WorkShop Visual のその他のツールに関する内容は、以下の章および節を参照してください。

- 第 14 章「Sun WorkShop Visual 再現機能」 (499 ページ)
  この章では、アプリケーションの使用状況の記録および再生を可能にする Sun WorkShop Visual 再現について説明します。
- 第 23 章「ユーザー定義ウィジェット」(719 ページ)
  Sun WorkShop Visual で独自のウィジェットを使用することを可能にする visu_config というツールについて説明します。
- 170 ページの「ピックスマップの編集」
  ピックスマップの作成と編集について説明します。
- 159 ページの「フォントの設定」
  フォントおよびフォントセットの選択について説明します。

- 154 ページの「色の設定」
  色の選択について説明します。

AppGuru

AppGuru では、テンプレートと呼ばれる再使用可能なデザインを使用することで、標準インタフェースを簡単に作成できます。独自のテンプレートを作成し、他のユーザーにそのテンプレートを使用させることもできます。テンプレートの作成と保存は AppGuru デザイナーのダイアログで行います。このダイアログからテンプレートを選択すると、そのテンプレートに記述されているデザインが Sun WorkShop Visual のセッションに自動的に追加されます。

AppGuru デザイナーのダイアログは、「ツール」メニューから「AppGuru デザイナー」を選択するか、またはツールバーの AppGuru ボタンを押すと表示されます。このボタンを図 13-1 に示します。

図 13-1　ツールバー上の AppGuru ボタン

AppGuruテンプレートを使用すると以下のような利点があります。

1. 社内でまたは特定のアプリケーションに対して使用しなければならないスタイルを、デザインの開始時から使用できる
2. 便宜上デザインが事前に定義されているため、エラーが発生しない
3. 類似したダイアログの基本デザインを再使用できる

AppGuru デザイナー

AppGuru デザイナーを使用すると、以下のことが行えます。

1. 新しいテンプレートを「ゼロから」作成する
2. 既存のテンプレートを編集する
3. 現在のデザインを使って、新しいテンプレートを仮に作ってみる

4. テンプレートを選択して Sun WorkShop Visual の現在のセッションに追加する

テンプレートを定義する際には、テキストフィールドにそのテンプレートの簡単な説明を入力することもできます。説明の入力は必須ではありませんが、テンプレートの機能などを簡単に記述しておくと、他のユーザーの参考になるため便利です。

AppGuru テンプレート

「ツール」メニューから「AppGuru デザイナー」を選択すると、図 13-2 に示すような「AppGuru テンプレート」ダイアログが表示されます。このダイアログに表示されるテンプレートが現在使用可能なテンプレートです。「テンプレート」メニューからコマンドを選択して、1 つのテンプレート (または指定したディレクトリのすべてのテンプレート) を「AppGuru テンプレート」ダイアログに読み込むことができます。テンプレートの読み込みを解除することもできます。その場合、テンプレートは「AppGuru テンプレート」ダイアログから削除されるだけで、削除されたテンプレートに関連付けられているファイルは残ります。

テンプレートを選択して「適用」を押すと、テンプレートで記述されるデザインが既存のデザインに追加されます。

テンプレートは「構成要素」で構成されており、各構成要素には任意の数のウィジェットを含めることができます。AppGuru の構成要素とは、「AppGuru テンプレート」ダイアログで選択および選択解除できるテンプレートの要素を指します。構成要素のウィジェットとは、構成要素の外観と動作を定義するものを指します。

テンプレートは、このダイアログの上部に小さなイメージとして表示されます。テンプレートを選択すると、選択したテンプレートに定義されている構成要素のリストがダイアログの下部に表示されます。デザインに追加したくない構成要素がある場合は、このリストから構成要素を選択して「オフ」にすることができます。1 つの構成要素でデザインの複数のウィジェットを表現できます。

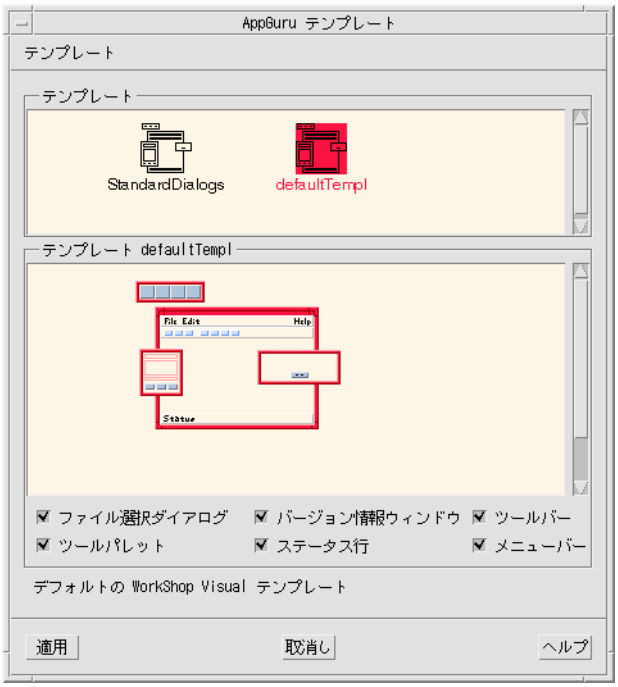


図 13-2 「AppGuru テンプレート」ダイアログ

テンプレートを編集するには「テンプレートを編集」を、テンプレートを作成するには「新規テンプレート」をそれぞれこのダイアログの「テンプレート」メニューから選択します。どのメニュー項目を選択しても、図 13-3 に示すような編集ダイアログが表示されます。

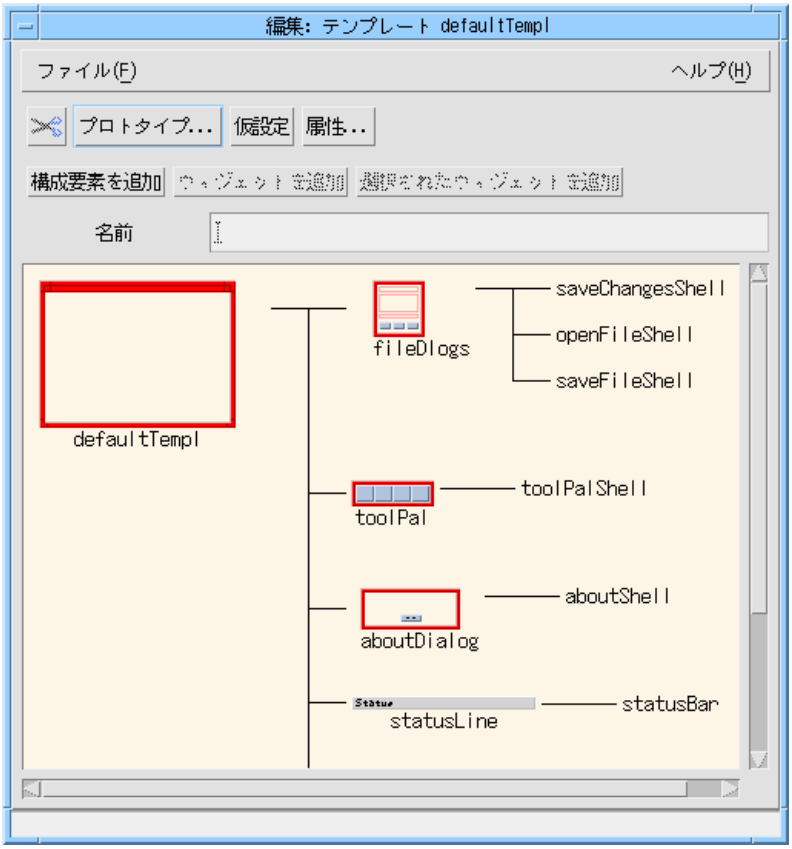


図 13-3 AppGuru のテンプレート編集ダイアログ

AppGuru のテンプレート編集ダイアログ

このダイアログでは以下のことが行えます。

1. テンプレートのすべての属性 (テンプレートに関連付けられているすべてのテキスト、デザインファイルとピックスマップファイルの位置を含む) を編集する
2. 作成中のテンプレートが「AppGuru テンプレート」ダイアログでどのように表示されるかを認する
3. テンプレートに構成要素を追加したり、テンプレートから構成要素を削除する
4. 構成要素にウィジェットを追加する

5. 構成要素の属性を編集する

6. テンプレートを保存する

これらについて、以下で詳しく説明します。

テンプレートの属性

テンプレートの選択時に「属性」ボタンを押すかテンプレートをダブルクリックすると、図 13-4 に示すようなテンプレートの属性ダイアログが表示されます。

図 13-4 テンプレートの属性ダイアログ

このダイアログで次の情報を編集できます。

- バックグラウンドピックスマップ
  このピックスマップは、フィルタすることのできないテンプレートの一部を表示します。つまり、「AppGuru テンプレート」ダイアログですべての構成要素が「オフ」になっているときでも表示されるピックスマップを指します。
- 小さなピックスマップ
  「AppGuru テンプレート」ダイアログの上部に表示される小さなイメージです。
- このテンプレートで参照されるウィジェットを含む Sun WorkShop Visual 保存ファイルの名前
  この情報がないと、Sun WorkShop Visual はウィジェットに関する情報が得られないので、ウィジェット情報を表示することもできません。
- テンプレートディレクトリの位置
  Sun WorkShop Visual は 1 つのアクションですべてのテンプレートをディレクトリに読み込むことができるので、すべてのテンプレートを同じディレクトリに置いておくと便利です。
- ピックスマップファイルを含むディレクトリ
- テンプレートの短い記述
  テンプレートを示す小さなイメージを選択すると「AppGuru テンプレート」ダイアログに表示されます。

プロトタイプ

テンプレートの編集または作成中に「プロトタイプ」ボタンを押すと、「AppGuru テンプレート」ダイアログで選択したときのテンプレートの表示を確認できます。

構成要素の追加と削除

テンプレートに構成要素を追加するには以下のようにします。

- テンプレートの編集ダイアログで「仮設定」ボタンを押す
- テンプレートの編集ダイアログで「構成要素を追加」ボタンを押す

「仮設定」は、現在のデザインのすべてのシェルを選択したテンプレートに追加します。各シェルはテンプレートの 1 つの構成要素です。「構成要素を追加」ボタンを押すと、構成要素が 1 つ追加されます。構成要素を削除するには、構成要素を選択してはさみのイメージの付いたカット用のボタン (「カット」ボタン) を押すだけです。ペーストや元に戻す機能はありません。

構成要素のウィジェットの追加と削除

構成要素にウィジェットを追加するには以下のようにします。

1. 構成要素を選択します。
2. Sun WorkShop Visual 階層でウィジェット (複数可) を選択します。
3. 「選択されたウィジェットを追加」ボタンを押します。

追加するウィジェットが別の Sun WorkShop Visual 保存ファイルにある場合、または現在のデザインで選択するのが困難な場合は、「ウィジェットを追加」を押し、ウィジェットを選択して「名前」テキストフィールドにウィジェットの名前を入力します。

ウィジェットを削除するには、ウィジェットを選択して、ツールバーの「カット」ボタンを押します。ペーストや元に戻す機能はありません。。

構成要素の属性の編集

テンプレートの編集ダイアログの「属性」ボタンを押すと、図 13-5 に示すような構成要素の属性ダイアログが表示されます。このダイアログに情報を入力して、構成要素の各種属性を編集できます。

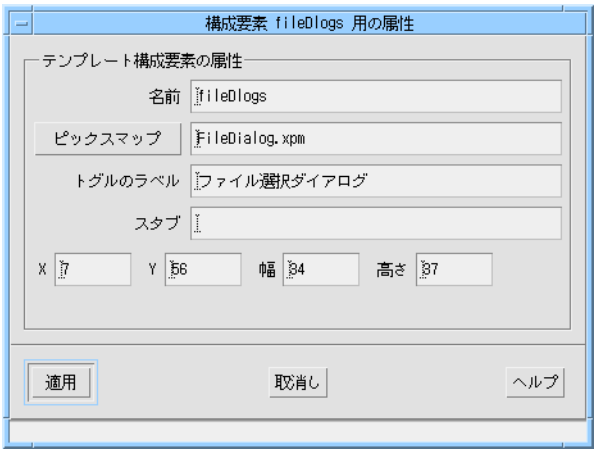


図 13-5 構成要素の属性ダイアログ

構成要素の属性ダイアログでは、構成要素がテンプレートの編集ダイアログに表示されるときの構成要素の名前を変更できます。他の残りのフィールドは、構成要素が「AppGuru テンプレート」ダイアログに表示されるときの構成要素に適用されます。構成要素の属性ダイアログで名前を指定したピックスマップは、指定された X 座標と Y 座標の位置に、指定された幅と高さで、テンプレートのバックグラウンドピックスマップの上に描画されます。「トグルのラベル」とは、「AppGuru テンプレート」ダイアログの構成要素を「オフ」にする際に使うトグルのラベルです。

注 - AppGuru への拡張は下位互換です。以前のリソースに基づいたテンプレートは、新しいシステムではすべて使用できます。

テンプレートの保存

AppGuru のテンプレートの編集ダイアログの「ファイル」メニューでは、以下のことが行えます。

1. 現在のファイル名 (存在する場合) を使用して現在のテンプレートを保存する
2. 別の名前を使用してテンプレートを保存する
3. テンプレートの編集ダイアログを閉じる

保存したファイルは X リソースファイルになり、このファイルが AppGuru で開かれます。リソースファイルは、テンプレートの記述を含む Sun WorkShop Visual デザインファイルも参照します。デザインファイルの名前は、属性ダイアログで変更してください。

Sun WorkShop Visual 捕獲機能

Sun WorkShop Visual 捕獲機能を使用すると、実行中の Motif アプリケーションからダイアログを捕獲し、それらのダイアログを Sun WorkShop Visual にドラッグすることができます。アプリケーションが Sun WorkShop Visual を使用してデザインされたかどうかにかかわらず、Sun WorkShop Visual 捕獲機能はアプリケーションのデザインの .xd ファイルを作成することができます。

Sun WorkShop Visual 捕獲機能は、「ツール」メニューから使用することができます。また、497 ページの「コマンド行からの Sun WorkShop Visual 捕獲機能の使用」で説明しているように、このツールをコマンド行から使用することもできます。

Sun WorkShop Visual 捕獲機能を使用する前に

Sun WorkShop Visual 捕獲機能が正しく動作するためには、捕獲したい Motif アプリケーションが Xt ライブラリ (libXt) と動的にリンクしている必要があります。多くの UNIX 実装上では、アプリケーションと libXt とのリンクが動的か静的かを調べるには、次のようにします。

```
ldd AnApplication
```

出力に libXt に関する内容がある場合は、そのアプリケーションは Xt ライブラリと動的にリンクしており、Sun WorkShop Visual 捕獲機能で使用することができます。出力にライブラリがない場合は、アプリケーションは Xt ライブラリと静的リンクしている可能性があります。Sun WorkShop Visual 捕獲機能を使用するには、アプリケーションを Xt 共有ライブラリに再度リンクする必要あります。

Sun WorkShop Visual 捕獲機能は完全に透過的で、アプリケーションのパフォーマンスにも動作にも影響しません。

Sun WorkShop Visual の捕獲機能の実行

「ツール」メニューから「Sun WorkShop Visual 捕獲」を選択すると「捕獲/再現」ダイアログが表示されます。「実行可能ファイル」というラベルが付いたボタンの横にあるテキストボックスに、捕獲する Motif アプリケーションの名前を入力します。また、このアプリケーションに引き渡す引数を入力できるテキストボックスもあります。図 13-6 にこのダイアログを示します。

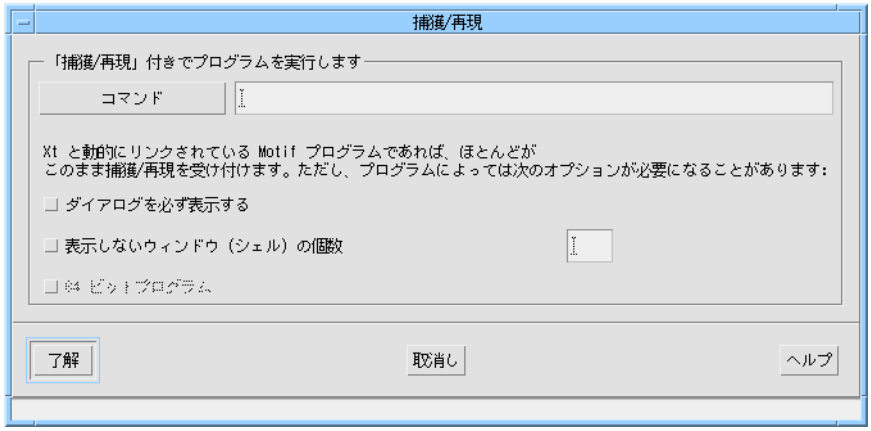


図 13-6 捕獲/再現のアプリケーション選択ウィンドウ

「了解」ボタンを押すと、指定された実行ファイルが PATH 変数に設定された実行パスの中にあるかどうか検索されます。このファイルがパス中にない場合、またはアプリケーションのある場所が分からない場合は、「実行可能ファイル」というラベルが付いたボタンを押します。図 13-7 に示すような、第 2 のスクロールリストを含むファイル選択ボックスが表示されます。各スクロールリストには、パスに含まれるディレクトリが表示されます。ただし、このパスは Sun WorkShop Visual によって設定されたものであるため、Sun WorkShop Visual に必要な特別なディレクトリを含んでいる場合があります。パスは、Sun WorkShop Visual の終了時には実行前と同じ状態になっています。「パス」スクロールリストからディレクトリを選択すると、そのディレクトリ内のファイルがすべて「ファイル」というラベルが付いたスクロールリストに表示されます。

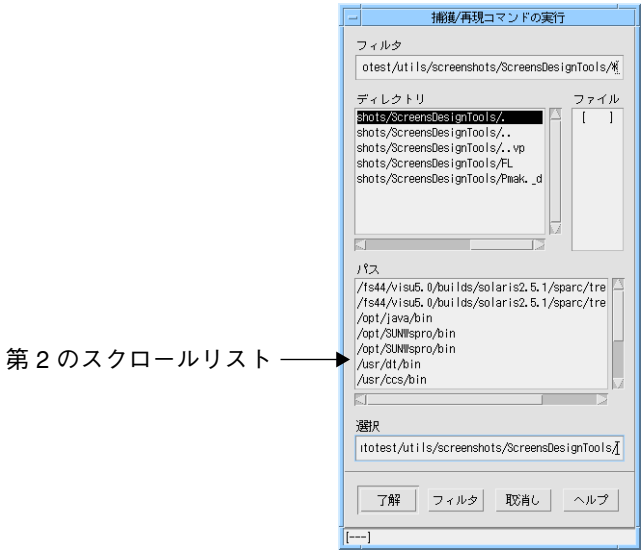


図 13-7 捕獲/再現のファイル選択ボックス

また、「Sun WorkShop Visual 捕獲」ダイアログはコマンド行からも実行することができます。詳細は、792 ページの「Sun WorkShop Visual 捕獲機能」を参照してください。

捕獲ダイアログ

図 13-8 に示す「Sun WorkShop Visual 捕獲」ダイアログには、Sun WorkShop Visual 捕獲機能用と Sun WorkShop Visual 再現用の 2 つのページがあります。ページを切り替えるには、「ページ」というラベルが付いたオプションメニューから「捕獲」と「再現」のどちらかを選択します。Sun WorkShop Visual 再現を使用する場合のダイアログについては、第 14 章「Sun WorkShop Visual 再現機能」を参照してください。

図 13-8　「Sun WorkShop Visual 捕獲」ダイアログ

「シェル」というテキストの右側に、アプリケーションで現在有効なシェルの名前が表示されています。このシェルが、「捕獲」ボタンを押すと捕獲されるシェルです。「捕獲」ボタンの下の領域には、現作業ディレクトリにある捕獲されたシェルが表示されています。

捕獲したシェルの保存およびアクセス

デフォルトでは、アプリケーションを捕獲すると、捕獲されたデザインを含むファイルが名前のない一時ディレクトリに保存されます。

注 - 一時ディレクトリおよびその保存内容は、Sun WorkShop Visual 捕獲機能を終了すると自動的に削除されます。作成した内容を保存したい場合は、名前を持ったディレクトリで作業してください (以降の内容を参照)。

Sun WorkShop Visual 捕獲機能の「ディレクトリ」メニューの「別名保存」オプションを使用して、現作業ディレクトリに新しい名前を付けて保存します。

ディレクトリを展開すると、ダイアログにはそのディレクトリに含まれる捕獲されたデザインがすべて表示されます。捕獲されたデザインは、捕獲されたシェルの縮小表示として表示されます。

「編集」メニューの操作を使用すると、あるディレクトリでカットまたはコピーしたデザインを別のディレクトリにペーストすることができます。「消去」コマンドを選択すると、選択したデザインが削除されます。

規約により、捕獲されたデザインには「.xd」というファイル名の接尾辞を付けます。ただし、この接尾辞は「Sun WorkShop Visual 捕獲」ダイアログには表示されません。

注 - 捕獲された各ダイアログデザインに対して .xpm ファイルが 1 つ作成されます。このファイルは、Sun WorkShop Visual 捕獲機能がウィンドウ保持領域にダイアログの縮小表示を行うために使用されます。

Sun WorkShop Visual 捕獲機能の使用

本節では、Sun WorkShop Visual 捕獲機能の使用例として、Sun WorkShop Visual に付属するユーザー定義ウィジェット構成ユーティリティ (visu_config) でデザインを捕獲する方法を説明します。

1. 「ツール」メニューから「Sun WorkShop Visual 捕獲」を選択します。
2. 「捕獲/再現コマンド実行」ダイアログの「実行可能ファイル」とラベルのついたテキストボックスに **`visu_config`** と入力し、「了解」ボタンを押します。

   Sun WorkShop Visual 捕獲ツールは、指定された実行ファイルが PATH に存在するか検索し、最初に見つけたディレクトリを呼び出します。visu_config ダイアログおよび「Sun WorkShop Visual 捕獲」ダイアログの両方が表示されます。

   「Sun WorkShop Visual 捕獲」ダイアログは事実上 visu_config プログラムに組み込まれているため、visu_config アプリケーションを終了すると Sun WorkShop Visual 捕獲機能も終了します。
3. 「Sun WorkShop Visual 捕獲」ダイアログの「捕獲」ボタンを押します。

   visu_config の有効なシェルの縮小表示が「Sun WorkShop Visual 捕獲」ダイアログに表示されます。他のダイアログを捕獲するには、visu_config から捕獲したいダイアログを起動して再度「捕獲」ボタンを押します。捕獲したダイアログがウィンドウ保持領域に追加されます。

4. Sun WorkShop Visual でデザインを見るには、「Sun WorkShop Visual 捕獲」ダイアログのウィンドウ保持領域にある縮小表示を Sun WorkShop Visual の構成領域まで (マウスボタン 2 を使用して) ドラッグします。

   これで、シェルおよびその階層が Sun WorkShop Visual に表示されます。

   この段階で Sun WorkShop Visual には、visu_config のウィジェット構造および関連するすべてのリソースを持つウィジェット階層が存在しています。また、visu_config と同じ外観を持つダイナミックディスプレイが表示されています。サブダイアログを捕獲した場合は、それらのサブダイアログも Sun WorkShop Visual の構成領域にドラッグすることができます。それらのサブダイアログは、独立したダイアログとして追加されます。

モード付きアプリケーションダイアログ

作成したアプリケーションでモード付きアプリケーションダイアログを使用している場合は、そのダイアログを閉じなければ Sun WorkShop Visual 捕獲インタフェースにアクセスすることができないため、「捕獲」ボタンを押すことができません。この場合は、「ホットキー」を押してください。デフォルトでは、Sun WorkShop Visual のリソースファイル中でホットキーは F5 機能キーに設定されています。ホットキーは、捕獲機能に関連する機能キーに翻訳されます。 リソースファイル中の設定は以下のとおりです。

```
*xdsTranslations:"<Key>F5: vcrInteractiveCaptureShell()"
```

すべてのユーザーに対してホットキーを別のキーに割り当てたい場合は、Sun WorkShop Visual のリソースファイルを編集してください。自分専用にホットキーを別のキーに割り当てたい場合は、ホームディレクトリの .Xdefaults ファイルを編集してください。

注 - トランスレーションの詳細は、O'Reilly and Associates 出版の『*X Toolkit Intrinsics Programming Manual*』などの資料を参照してください。

捕獲された情報

Sun WorkShop Visual 捕獲機能が visu_config の外観を捕獲したことは、すぐに確認できます。しかし実際は、外観以外の内容も捕獲されています。捕獲された内容は、以下のとおりです。

- 元のアプリケーションで使用されているウィジェットの階層
- 明示的に設定されたすべてのリソース
- フォームアタッチメントを含むすべての配置
- 寸法

Sun WorkShop Visual 捕獲機能を使用することで、必要なものはすべて既存の Motif アプリケーションのデザインから取り込むことができます。ただし、コールバックやリンクなどのアプリケーションに特有の動作は捕獲されません。

Java エミュレーションウィジェットの捕獲

新しい Java レイアウトエミュレーションウィジェットを使用したデザインに Sun WorkShop Visual の捕獲機能を使用したい場合は、xdsCaptureUserWidgets リソースが「true」に設定してあることを確認する必要があります。このリソースを使用すると、Motif 以外のウィジェットが完全に捕獲されます。このリソースを設定していない場合、ウィジェットは捕獲スクリプトにおける描画領域として現れます。
Java エミュレーションウィジェットの詳細については、第 10 章「Java 用のデザイン」の 375 ページの「Java クラス用の新しいウィジェット」セクションを参照してください。

ユーザー定義ウィジェットの捕獲

496 ページの「Java エミュレーションウィジェットの捕獲」で説明したように、リソース xdsCaptureUserWidgets を最初に「true」に設定すると、他社製のウィジェットを含むデザインを捕獲できます。これらのウィジェットは、Sun WorkShop Visual にすでに統合されている場合にのみ、Sun WorkShop Visual で表示することができます。他社のウィジェットの統合についての詳細は、719 ページの第 23 章「ユーザー定義ウィジェット」 を参照してください。

コマンド行からの Sun WorkShop Visual 捕獲機能の使用

Sun WorkShop Visual 捕獲機能は、`visu_capture` という独立したアプリケーションです。コマンド行から Sun WorkShop Visual 捕獲機能の使用方法を参照するには、次のように入力します。

```
visu_capture -x
```

Sun WorkShop Visual 捕獲機能の使用例を以下に示します。

1. 次のように入力します。

   ```
   visu_capture -f MyCaptureDesign.xd AnApplication
   ```

   `MyCaptureDesign.xd` は、捕獲されたアプリケーションが含まれるファイル名です。このファイルは、Sun WorkShop Visual の保存ファイル形式を使用します。`AnApplication` は、捕獲するアプリケーション名です。

2. 起動されたアプリケーションの上にマウスのポインタを置いた状態で、F5 キーを押します。

   アプリケーションが捕獲され、`.xd` ファイルに保存されます。

3. Sun WorkShop Visual 捕獲機能の使用方法に関するヒントは、924 ページの「Sun WorkShop Visual 再現、捕獲機能」を参照してください。

第 14 章

Sun WorkShop Visual 再現機能

はじめに

Sun WorkShop Visual 再現機能を使用して、Xt ベースのアプリケーションを記録および再生することができます。

記録モードでは、Sun WorkShop Visual 再現機能は「push hello_button, type Hello World」のように、ユーザーのアクションを高水準で記述したスクリプトを作成します。

再生モードでは、アプリケーション内の任意のウィジェットの状態を調べたり、再生速度を制御することができます。スクリプト内のアクションは、ユーザーがキーボードから操作しているかのように再生されます。

Sun WorkShop Visual 再現機能のコマンドセットはユーザーによる拡張が可能で、操作性に優れ、柔軟性に富んでいます。Sun WorkShop Visual 再現機能は、次のような幅広い用途に活用することができます。

- Motif アプリケーションの記録および再生
- 製品の宣伝用プログラムの作成
- アプリケーションのデバッグ
- アプリケーション内で検出された問題点の再現
- 製品の学習用デザインの開発
- アプリケーションのテストの自動化

再コンパイルや再リンクは必要ありません。また、特別なテスト環境も不要です。

Sun WorkShop Visual 再現機能は、Sun WorkShop Visual の「ツール」メニューから使用することができます。また、このツールは 515 ページの「コマンド行からの記録および再生」で説明するように、コマンド行からも使用することができます。

本章では、Sun WorkShop Visual 再現機能の使い方に慣れるために、まず学習手順の中で簡単な例をいくつか挙げながら Sun WorkShop Visual 再現機能の使用方法について説明します。Sun WorkShop Visual 再現機能の拡張については、528 ページの「Sun WorkShop Visual 再現ウィジェットセットの 拡張」 と 543 ページの「独自の Sun WorkShop Visual 再現機能コマンドの追加」で記述します。

Sun WorkShop Visual 再現機能の構文については 929 ページの付録 A「Sun WorkShop Visual 再現機能の コマンド構文」を参照してください。

Java アプリケーションの記録と再生

Sun WorkShop Visual 再現機能は、Java アプリケーションに使用できます。その場合は、528 ページの「Sun WorkShop Visual 再現機能を使用したデバッグ」セクションの説明に従って、最初のアプリケーションとして Java インタプリタを指定し、その後にターゲットアプリケーションを指定しなければなりません。「Sun WorkShop Visual 再現機能を使用したデバッグ」セクションでは、Sun WorkShop Visual 再現機能をコマンド行から使用する場合の間接参照の使い方が説明されています。このように、Java アプリケーションを記録または再生するためには、次のようにして Java インタプリタも指定しなければなりません。

```
visu_replay java MyJavaProgram
```

これは Java アプリケーションには当てはまりますが、アプレットには当てはまりません。Java コードの生成の詳細については、第 10 章「Java 用のデザイン」を参照してください。

Sun WorkShop Visual 再現機能を使用する前に

Sun WorkShop Visual 再現機能が正しく動作するためには、記録したい Motif アプリケーションが Xt ライブラリ (libXt) と動的にリンクしている必要があります。UNIX 実装上では通常、次のように入力してアプリケーションと libXt とのリンクが動的か静的かを判断することができます。

```
ldd AnApplication
```

出力に libXt に関する内容がある場合は、そのアプリケーションは Xt ライブラリと動的にリンクしており、Sun WorkShop Visual 再現機能で使用することができます。このライブラリが存在しない場合は、アプリケーションは Xt ライブラリと静的にリンクしている可能性があります。この場合に Sun WorkShop Visual 再現機能を使用するには、アプリケーションを Xt 共用ライブラリと再度リンクする必要があります。

Sun WorkShop Visual 再現機能を使用してアプリケーションを呼び出す方法

注 - Sun WorkShop Visual 再現機能をすぐに使用したい場合は、本節を省略して 510 ページの「学習例」に進んでください。

「ツール」メニューから「WorkShop Visual 再現」を選択すると、記録または再生したいアプリケーション名の入力を要求するダイアログが表示されます。図 14-1 に、そのダイアログを示します。

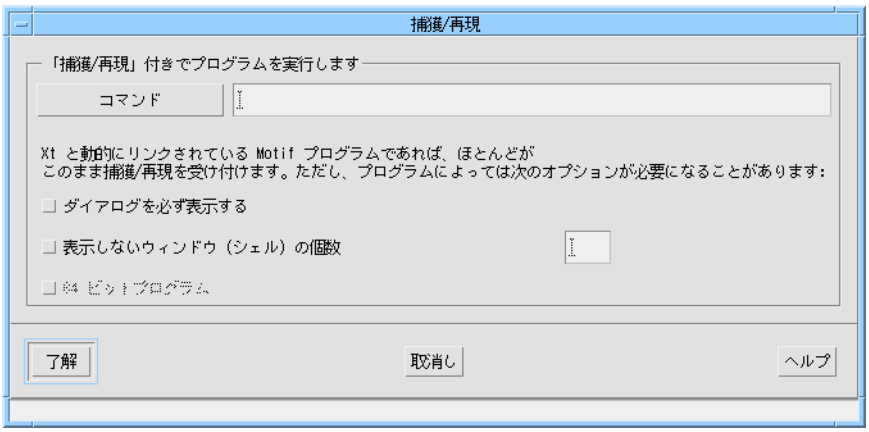


図 14-1 「捕獲/再現」の入力要求ダイアログ

「コマンド」テキストボックスに名前を入力します。アプリケーションの名前または保存場所が分からない場合は、「コマンド」というラベルが付いたボタンを押します。図 14-2 に示すような、「パス」スクロールリストを含むファイル選択ボックスが表示されます。

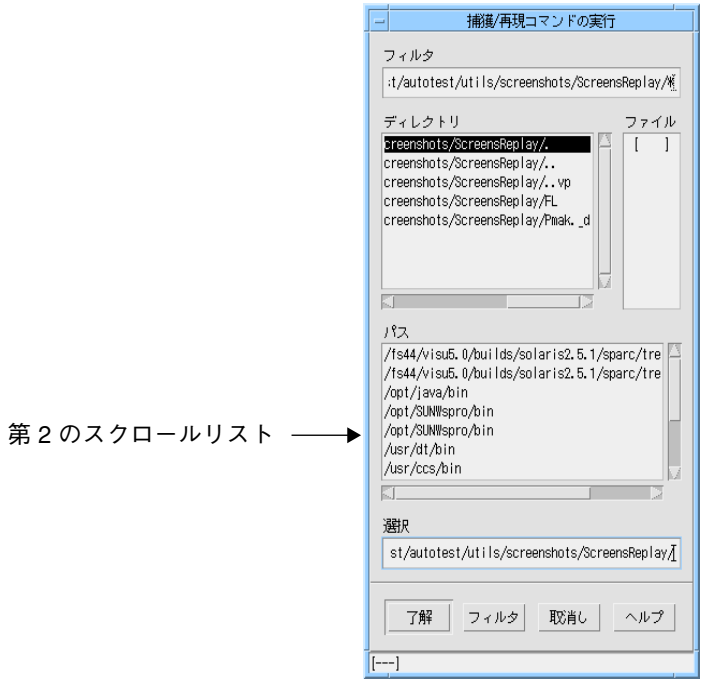


図 14-2 「捕獲/再現」のファイル選択ボックス

「パス」というラベルが付いた第 2 のスクロールリストには、PATH 環境変数で指定されている各ディレクトリが表示されます。 このリストから項目を選択すると、そのディレクトリの内容が「ファイル」リストに表示されます。

注 - 「パス」スクロールリストには、Sun WorkShop Visual 再現を実行したときに PATH 環境変数として設定されたディレクトリが表示されます。ただし、Sun WorkShop Visual に必要な特別なディレクトリが含まれている場合もあります。PATH 設定 は、Sun WorkShop Visual の終了時には実行前と同じ状態になっています。

ダイアログの「ファイル」リストから項目を選択して「了解」ボタンを押すと、その項目は「捕獲/再現」の入力要求ダイアログの「コマンド」フィールドに追加されます。このダイアログの「引数」というラベルが付いたテキストボックスに、アプリケーションに対するオプションや引数を入力します。「了解」ボタンを押すと、アプリケーションが Sun WorkShop Visual 再現を使用して実行されます。

ここでは、以下の 2 つの点に注意してください。

1. 「Sun WorkShop Visual 再現」ダイアログはアプリケーションの一部であるため、アプリケーションを終了すると画面から消去されます。
2. Sun WorkShop Visual 再現機能はアプリケーションの実行内容を監視するだけです。再生モードで動作を再現するだけで、アプリケーションの動作には決して影響を与えません。

記録対象

Sun WorkShop Visual 再現機能は、特に移植性と明確な記述を行うことに重点を置いています。 Motif インタフェースを効率的にテスト実行するように設計されています。

Sun WorkShop Visual 再現機能は、アプリケーション内のウィジェット間の移動およびユーザーとウィジェットとの対話を詳細に渡って記録します。以下の情報の記録および再生を行うことができます。

- すべてのウィジェットに対するアクション
- サブダイアログの表示および終了
- キーボードアクセラレータ、修飾子の使用
- すべてのキーボード入力 (テキスト、矢印キー、Delete など)
- どのマウスボタンが押されたかを含むすべてのマウスボタン操作

Sun WorkShop Visual 再現機能は、汎用の X テスト用機構として設計されているわけではありません。したがって、すべてのアプリケーションの動作が Sun WorkShop Visual 再現で記録されるわけではありません。ただし、Sun WorkShop Visual 再現機能を拡張することはできます。詳細は、528 ページの「Sun WorkShop Visual 再現 ウィジェットセットの 拡張」および 543 ページの「独自の Sun WorkShop Visual 再現機能コマンドの追加」を参照してください。

Sun WorkShop Visual 再現機能のインタフェース

「Sun WorkShop Visual 再現」ダイアログは、アプリケーションの横に表示されます。また、著作権メッセージも、Sun WorkShop Visual 再現機能の実行時に標準のエラー出力に表示されます。

著作権メッセージが表示されない場合は、アプリケーションが Xt ライブラリと動的にリンクしていない可能性があります。詳細は、500 ページの「Sun WorkShop Visual 再現機能を使用する前に」を参照してください。

図 14-3 に、「Sun WorkShop Visual 再現」ダイアログを示します。

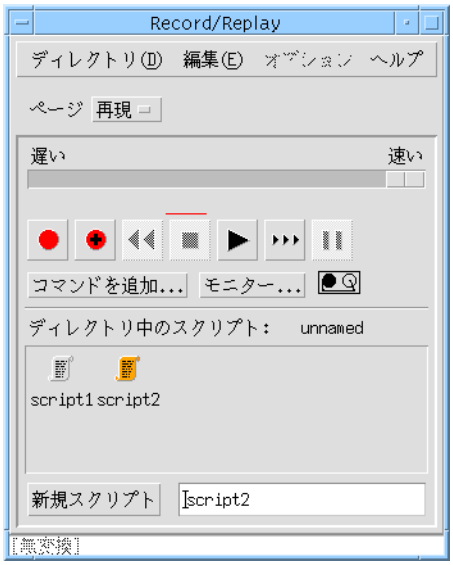


図 14-3 「Sun WorkShop Visual 再現」ダイアログ

また、Sun WorkShop Visual 再現機能の実行中にコマンド行からこのダイアログを表示することもできます。詳細は、515 ページの「コマンド行からの記録および再生」を参照してください。

このダイアログには、Sun WorkShop Visual 再現用とSun WorkShop Visual 捕獲用の 2 つのページがあります。ページを切り替えるには、「ページ」というラベルが付いたオプションメニューを選択します。Sun WorkShop Visual 捕獲機能を使用する場合のダイアログについては、492 ページの「捕獲ダイアログ」を参照してください。

機能および操作

「Sun WorkShop Visual 再現」ダイアログが表示されると、スクリプトの記録および再生を直ちに実行することができます。スクリプトの記録および再生はすべて、図 14-4 に示す「Sun WorkShop Visual 再現」ウィンドウ上のボタンを使用して行います。

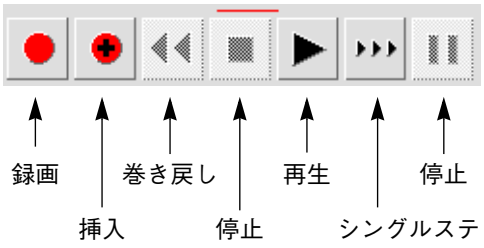


図 14-4 「Sun WorkShop Visual 再現」ウィンドウのボタン

ボタンの説明は、以下のとおりです。

●	録画	選択したスクリプトの現在の位置から、アプリケーションに対して行なったユーザーアクションを記録します。スクリプトを停止した後でこのボタンを押すと、スクリプトはその位置から上書きされます。スクリプトの最後でこのボタンを押すと、そのスクリプトに追加されます。
⊕	挿入	スクリプトが停止した位置に、アプリケーションに対して行なったユーザーアクションを記録します。スクリプト内のアクションは変更されません。
◀◀	巻き戻し	選択したスクリプトを先頭まで巻き戻します。

注 - スクリプトを正確に再生するには、録画開始時の状態にアプリケーションをリセットしなければならない場合があります。

■	停止	スクリプトの再生を停止します。
▶	再生	スクリプトの現在の位置からスクリプトが停止されるか末尾に達するまで、選択したスクリプトを再生します。
▸▸▸	シングルステップ	スクリプト内の次のコマンドを再生します。
❚❚	停止	記録または再生を一時停止します。このボタンを再度押すと、処理が続行されます。

有効なボタンだけが選択可能です。無効なボタンはグレー表示されています。

スクリプトを作成する前は、押すことができるボタンは「録画」だけです。このボタンを押すと「unnamed (名前未定)」のスクリプトが作成されます。スクリプトを作成したら、そのスクリプトの「巻き戻し」、「再生」、「シングルステップ」を実行することができます。

スクリプトを停止または一時停止すると、「挿入」ボタンが有効になります。挿入処理については、515 ページの「スクリプトへの挿入」を参照してください。

スクリプトの作成および命名

「新規スクリプト」ボタンを押すと、空のスクリプトが作成されます。スクリプトの命名または名前変更を行うには、次のようにします。

1. 「Sun WorkShop Visual 再現」ダイアログで関連するアイコンをクリックします。
2. 「新規スクリプト」テキストフィールドにスクリプトの名前を入力し、Return キーを押します。

   既存のスクリプトの名前と同じ名前を入力すると、新しく命名されたスクリプトには番号が付加されて元のスクリプトと区別されます。

注 - 「Sun WorkShop Visual 再現」ダイアログ内にスクリプトがない場合は、「録画」を押すと自動的に「unnamed (名前未定)」スクリプトが新規に作成されます。

スクリプトの選択およびステータスインジケータ

現在選択しているスクリプトは、「Sun WorkShop Visual 再現」ダイアログ内で強調表示されます。

Sun WorkShop Visual 再現のステータスインジケータは、記録中か再生中かを表し、またスクリプト内での現在位置を示します。ステータスインジケータが赤い場合は、記録中を示します。それ以外の場合は、再生中です。図 14-5 に、インジケータの表示状態としてあり得るものを示します。

図 14-5 Sun WorkShop Visual 再現インジケータのステータス表示

ボタンパネルで最後に選択したボタンの上には、赤い線が表示されます。

モニター

「モニター」ボタンを押すと、記録中および再生中に行なったアクションのログが表示されます。図 14-6 に示すように、記録個所または再生個所の先頭および末尾を示すコメントは、Sun WorkShop Visual 再現機能によって自動的に挿入されます。

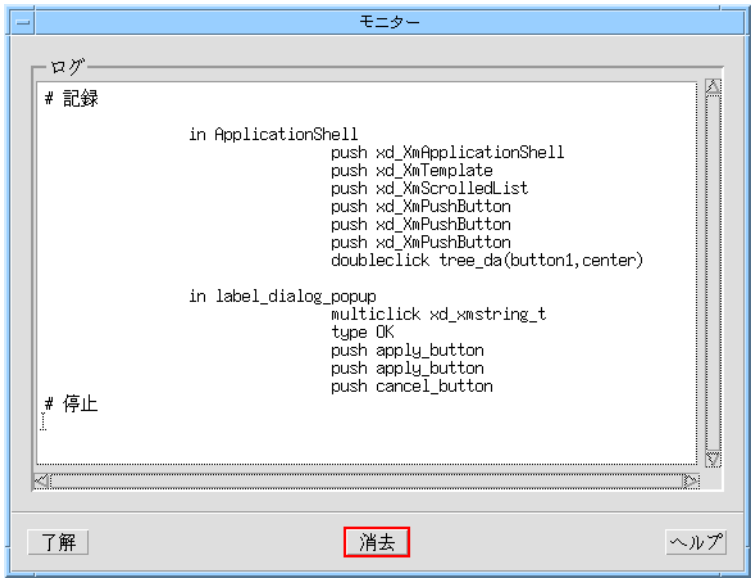


図 14-6 Sun WorkShop Visual 再現機能の「モニター」ウィンドウ

追加コマンドの挿入

アプリケーション以外のコマンドおよびコメントも、アクションと同じようにスクリプトに追加することができます。追加するには、手動でスクリプトを編集するか Sun WorkShop Visual 再現インタフェースを使用して編集します。本節では、インタフェースからスクリプトを編集する方法を説明します。

まず、追加コマンドを追加する位置でスクリプトを停止します。追加コマンドをスクリプトの先頭に追加するには、まずスクリプトを巻き戻します。スクリプト内の別の位置にコマンドを追加する場合は、シングル ステップ実行してその位置まで移動します。

次に「コマンドを追加」というラベルが付いたボタンを押します。テキスト編集ウィンドウが表示され、追加コマンドまたはコメントを入力することができます。図 14-7 に、このダイアログを示します。

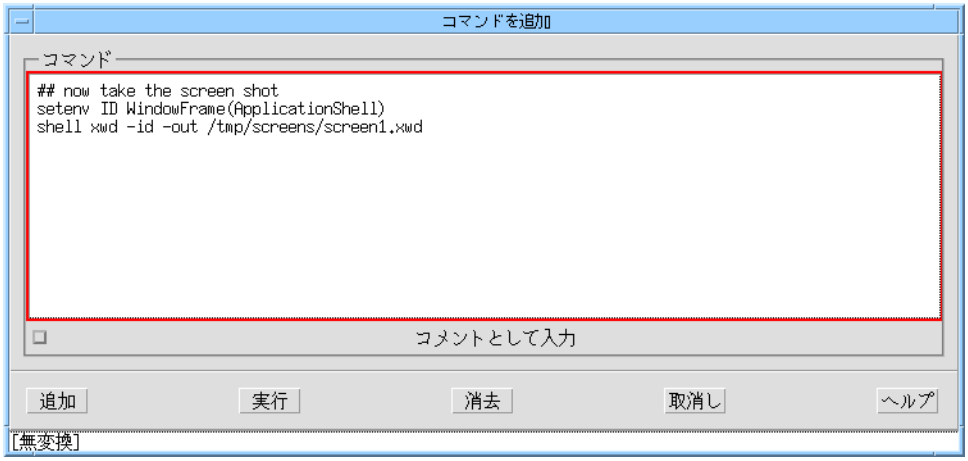


図 14-7　「コマンドを追加」ダイアログ

「コメントとして入力」トグルを設定すると、ダイアログの内容がコメントとして処理されます。スクリプトでは、各行の行頭に「#」文字が付加されます。

「実行」ボタンを押すと、記録されているスクリプトとは無関係に「コマンドを追加」ダイアログ内のコマンドだけが実行されます。コマンドが実行される様子は、「モニター」ウィンドウを使用して確認します。コマンドが希望どおりに実行されている場合は、「追加」ボタンを押してそれらのコマンドをスクリプトに保存します。

さらにコマンドまたはコメントを追加入力する場合は、その前に「消去」を押してください。「コマンドを追加」ダイアログの情報が削除されます。スクリプトの内容には影響しません。

再生速度の変更

「Sun WorkShop Visual 再現」ダイアログの「速い/遅い」スライダを使用すると、選択したスクリプトが再生される速度を変更することができます。デフォルトでは、スクリプトは最高速度で再生されます。

モード付きアプリケーションダイアログ

作成したアプリケーションでモード付きアプリケーションダイアログを使用している場合は、そのダイアログを閉じなければ Sun WorkShop Visual 再現機能のインタフェースにアクセスすることができません。つまり、モード付きアプリケーションダ

イアログ内では記録および再現を停止することはできません。シングルステップモードでは、モード付きアプリケーションダイアログで行なったすべてのアクションは、シングルステップとして認識されます。

スクリプトの保存およびアクセス

デフォルトでは、「Sun WorkShop Visual 再現」ダイアログで作成したスクリプトは名前のない一時ディレクトリに保存されます。

注意 - XDS_KEEPDIR 環境変数を定義していない場合は、一時ディレクトリおよびその保存内容は、Sun WorkShop Visual 再現を終了すると自動的に削除されます。XDS_KEEPDIR 環境変数を定義すると、一時ディレクトリとその保存内容は /tmp/XDS_SAVE に保存されます。作成した内容を保存したい場合は、名前を持ったディレクトリで作業してください (以下の内容を参照してください)。

Sun WorkShop Visual 再現機能の「ディレクトリ」メニューの「別名保存」オプションを使用して、現作業ディレクトリに新しい名前を付けて保存します。

「ディレクトリ」メニューの「開く」オプションを使用して、別のディレクトリからスクリプトを開きます。また、「別名保存」オプションを使用して、現在作業しているディレクトリの名前を変更することもできます。

「編集」メニューのオプションを使用すると、あるディレクトリでカットまたはコピーしたスクリプトを別のディレクトリにペーストすることができます。「消去」コマンドを選択すると、選択したスクリプトが削除されます。

規約により、ファイルシステムでは記録スクリプトには「.xds」というファイル名の接尾辞を付けてください。ただし、この接尾辞は、「Sun WorkShop Visual 再現」ダイアログに表示されるスクリプトの名前には使用されません。

学習例

この節では、Sun WorkShop Visual 再現機能を使用して visu_config ツールに対して行った操作を記録し、次に再生する方法を段階的に説明します。

注 - 以下の学習例では、visu_config ツールに関する知識は必要ありません。visu_config ツールの詳細については、728 ページの「visu_config のメインダイアログ」を参照してください。

---

1. 「ツール」メニューから「WorkShop Visual 再現」を選択します。
2. 「捕獲/再現」ダイアログの「コマンド」フィールドに `visu_config` と入力し、「了解」ボタンを押します。

   visu_config が実行され、その横に「Sun WorkShop Visual 再現」ダイアログが表示されます。
3. 「新規スクリプト」ボタンを押します。

   「unnamed (名前未定)」スクリプトが作成されます。
4. 「新規スクリプト」テキストフィールドにスクリプトの名前を入力し、Return キーを押します。

   入力したとおりに、スクリプトの名前が変更されます。
5. 「モニター」ボタンを押します。

   ダイアログが表示され、行なったすべてのアクションのログが表示されます。
6. 図 14-8 に示す「録画」ボタンを押します。

図 14-8 「録画」ボタン

7. visu_config で以下の操作を行います。
   a. 「選択」テキストフィールドに「**one**」と入力し、Return キーを押します。

      この名前が「ファミリ」リストに追加されます。
   b. 「選択」フィールドの「one」という名前をダブルクリックし、「**two**」と入力して Return キーを押します。

      これで、「ファミリ」リストの項目は 2 つになります。
   c. 「ファミリ」リストの「**one**」をクリックし、「編集」ボタンを押します。

「ファミリ」ダイアログが表示されます。

d. 「選択」フィールドに「**`WidgetOne`**」と入力し、Return キーを押します。

この名前が「ウィジェットクラス」リストに追加されます。

e. 「ウィジェットクラス」リストの「WidgetOne」をダブルクリックします。

「ウィジェット」ダイアログが表示されます。

f. 「ウィジェット」ダイアログの「閉じる」ボタンを押します。

g. 「ウィジェットクラス」ダイアログの「閉じる」ボタンを押します。

h. 「ファミリ」ダイアログの「編集」メニューから「停止リスト」オプションを選択します。

「停止リスト」ダイアログが表示されます。

i. 「停止リスト」ダイアログの「プルダウンメニュー」および「テキストフィールド」というラベルが付いたトグルを押します。

j. 「適用」ボタンを押し、その後で「閉じる」ボタンを押します。

k. 「ファミリ」ダイアログのファイルメニューから「新規」を選択します。

「変更を保存」警告ダイアログが表示されます。

l. 「変更を保存」ダイアログの「いいえ」ボタンを押します。

8. 「停止」ボタンを押します。

「録画」ボタンおよび「巻き戻し」ボタンが有効になります。それ以外のボタンはすべて無効になります。

アクションの記録を含むファイルが作成されました。このファイルは、いつでも再生することができます。ここでは内容を確認するために、すぐに再生してみましょう。

9. 「巻き戻し」ボタンを押します。

「録画」、「挿入」、「再生」、「シングルステップ」の各ボタンが有効になります。

10. 「再生」ボタンを押します。

これまで記録した内容を見ることができます。「WorkShop Visual 再現」ダイアログの「速い/遅い」スライダを使用すると記録内容を再生する速度を変更することができます。

11. 「巻き戻し」ボタンを押します。

12. 「シングルステップ」ボタンを押します。

    このボタンを使用すると、記録スクリプトの各コマンドを 1 ステップずつ実行することができます。このコマンドは、画面に「モニター」ウィンドウがあるとさらに詳細に内容を確認することができます。各ステップが再生されるたびに、該当するログが「モニター」ウィンドウに出力されます。

13. visu_config を終了します。

    変更を保存するかどうかを尋ねるメッセージに対しては、「いいえ」を選択します。アプリケーションが終了すると、記録内容の再生も終了し、「WorkShop Visual 再現」ダイアログも消去されます。これは、このダイアログが実際は visu_config プログラムの一部であるためです。

注意 - `XDS_KEEPDIR` 環境変数を定義していない場合は、一時ディレクトリおよびその保存内容は、Sun WorkShop Visual 再現を終了すると自動的に削除されます。`XDS_KEEPDIR` 環境変数を定義すると、一時ディレクトリとその保存内容は `/tmp/XDS_SAVE` に保存されます。作成した内容を保存したい場合は、名前を持ったディレクトリで作業してください (以下の内容を参照してください)。

スクリプトの内容

前述の例によって以下のスクリプトが作成されました。学習例で実行したステップごとに注釈が付いています。このスクリプトで、Sun WorkShop Visual 再現の構文の約 90 % を紹介します。

```
in ApplicationShell      (visu_config アプリケーション内)
```

「選択」テキストフィールドに

```
    push Text
    type one
    key Return
    doubleclick Text
```

「選択」テキストフィールドに

```
    type two
    key Return
```

「ファミリ」リストの「one」を

```
    push ItemsList('one',1)
    push family_selection.OK
```

「ウィジェット」ダイアログの「選択」テキストフィールドに「WidgetOne」と入力する。

```
in entity_dialog
    push Text#5
    type WidgetOne
```

「ウィジェットクラス」リストから「WidgetOne」を選択する。

```
    push widgetlist_selection.OK
    doubleclick ItemsList#5('WidgetOne',1)
in widgetedit_dialog
```

「ウィジェット」ダイアログおよび「ファミリ」ダイアログを閉じる。

```
    push widgetedit_closeb
in entity_dialog
    push widgetlist_selection.widgetlist_quitb
in ApplicationShell
```

「停止リスト」ダイアログを表示する。

```
    cascade family_editb
        select family_stop_b
in stop_list_shell
```

「停止リスト」ダイアログのトグルを設定する。

```
    push stop_pulldown_menu
    push stop_text_field
    push stop_apply
    push stop_close
```

「ファイル」メニューから「新規」を選択する。

```
in ApplicationShell
    cascade family_fileb
        select family_newb
```

「変更を保存」プロンプトに対して「いいえ」を選択する。

```
in savechanges_dialog
    push savechanges_messagebox.Cancel
```

注 - 作成したファイルは、正確にはこのとおりではない場合があります。多少異なる順序でアクションを実行した場合や、戻って入力ミスを訂正した場合があるためです。これらはすべて記録されています。

スクリプトへの挿入

スクリプトの先頭または途中 (たとえば、シングルステップ実行中) にアクションを挿入することができます。

注 - 「モニター」ウィンドウを開いていると、スクリプト内での現在の位置が正確にわかります。

スクリプトにアクションを追加するには、以下の作業を行います。

- スクリプトの先頭にいる場合は「挿入」ボタンを押す
- スクリプトの途中にいる場合は「停止」ボタンを押し、次に「挿入」ボタンを押す

いずれの場合も、続けてアプリケーションを使用します。

アプリケーションで行うことが、すべて既存のスクリプトの現在の位置に挿入される点を除けば、「挿入」ボタンを押すことは「録画」ボタンを押すことと同じです。挿入モードではないときに「録画」ボタンを押すと、スクリプトの内容が上書きされます。

注 - アクションをスクリプトに挿入する際は、挿入後もそのスクリプトは続けて実行可能である必要があります。続行できない場合は、再生はその位置で停止してしまいます。

コマンド行からの記録および再生

Sun WorkShop Visual 再現機能は、独立したアプリケーションとして提供されています。記録スクリプトおよび再生スクリプトを両方ともコマンド行から実行することができます。

Sun WorkShop Visual 再現機能を使用したスクリプトの記録

Sun WorkShop Visual 再現機能 (ユーザーアクションの記録に使用する場合) は、visu_recordという独立したアプリケーションとして提供されます。

ツールに関する基本情報を表示するには、次のように入力します。

```
visu_record -x
```

次の行は、visu_record の使用例です。

```
visu_record -f MyRecordScript AnApplication
```

MyRecordScript は、内容を記録したスクリプトが保存されるファイル名です。この引数を指定する必要はありません。指定しないと、スクリプトは標準出力に書き込まれます。AnApplication は、記録するアプリケーションの名前です。-i フラグを指定すると、ツールを対話形式で使用できます。この場合、503 ページの「Sun WorkShop Visual 再現機能のインタフェース」で示した「WorkShop Visual 再現」ダイアログが表示されます。

Sun WorkShop Visual 再現機能を使用したスクリプトの再生

Sun WorkShop Visual 再現機能 (記録したスクリプトの再生に使用する場合) は、visu_replayという独立したアプリケーションとして提供されます。

ツールに関する基本情報を表示するには、次のように入力します。

```
visu_replay -x
```

次の行は、Sun WorkShop Visual 再現機能の使用例です。

```
visu_replay -f MyRecordScript AnApplication
```

MyRecordScript は、内容を記録したスクリプトを含むファイル名です。この引数を指定する必要はありません。指定しないと、スクリプトは標準入力から読み取られます。AnApplication は、再実行するアプリケーションの名前です。-i フラグを指定すると、「WorkShop Visual 再現」ダイアログでツールを対話形式で使用できます。

Sun WorkShop Visual 再現機能の応用

本節では、Sun WorkShop Visual 再現機能の応用方法を説明し、以下の項目について説明します。

- 循環表示の作成
- 画面ダンプの実行
- テスト
- デバッグ

これらの内容は、Sun WorkShop Visual 再現機能の応用方法のすべてではありません。Sun WorkShop Visual 再現機能の幅広い機能を紹介するために、例として挙げています。

繰り返しデモの作成

Sun WorkShop Visual 再現機能を使用して作成したスクリプトは、以下に示すような簡単なシェルスクリプトを使用して「連続ループ」状態で実行することができます。

```
while (true)
{
                visu_replay -f mydemo.xds myapplication
}
end
```

注 - このような繰り返しデモを作成する際は、スクリプトの末尾に、アプリケーションを再実行できるような状態にするためのコマンドを必ず記述してください。

画面のダンプの撮影

アプリケーションの画面ダンプの撮影は、そのアプリケーションがたえず変わる傾向がある場合は特に複雑になることがあります。Sun WorkShop Visual 再現機能を使用すると、後で何度でも再利用できる画面ダンプ用スクリプトを作成することができます。また、画面ダンプ処理が自動化されるので、それらの処理を行うための費用を大幅に削減できます。

画面ダンプスクリプトは、アプリケーションの画面撮影を行う準備をする一連のアクションと、その後に続けて実際の画面撮影を行うアプリケーション外のコマンドで構成されています。以下に示すスクリプト例では、*current_shell* ダイアログの画面ダンプが撮られます。

```
in current_shell
 setenv ID WindowFrame(current_shell)
 shell xwd -id $ID -out /tmp/current_shell.xwd
```

最後の 2 行は非アプリケーションコマンドです。最初の行で変数 `ID` をウィンドウ装飾を含む現在のシェルウィンドウに設定します。2 番目の行では、`xwd` コマンドを使用してシェルウィンドウの画面ダンプを撮影し、保存します。当然、`xwd` の代わりに別の画面ダンプコマンドを使用することができます。変数の設定に使用されるキーワードについては、945 ページの「非アプリケーション操作」を参照してください。

テスト

Sun WorkShop Visual 再現機能は簡単に使用できて移植性のある、強力なウィジェットベースのテストツールです。Sun WorkShop Visual がサポートするすべてのプラットフォームに対して、テストツールとして使用できます。

ウィジェットベースのテストの役割

ほとんどの Motif/Xt プログラミングでは、Motif ウィジェットを再利用し、X ツールキットを利用しています。Sun WorkShop Visual 再現テストはテスト動作の制御とテストが成功したかどうかの判定の両方を行うために、Xt ウィジェット階層を主に使用します。

ウィジェット自体が正しいかどうかを調べているのではない、ということに注意してください。Sun WorkShop Visual 再現機能では、ユーザーがアプリケーション内のウィジェットに対して行なったアクションによって、希望する結果が得られたかどうかだけを調べます。

注 - このテスト上のテクニックおよび方法は、テストの「陳腐化」に対して非常に高い抵抗力があります。使用するディスプレイのサイズ、形状、品質とは関係なくテスト結果は同じになる必要があります。アプリケーション自体が変更された場合以外は、テストを追加または更新する必要はありません。また、これらのテストは、テスト担当者に報告されていない変更があれば、すぐに検出します。

すべてのテストをこのように自動化できるわけではありません。アプリケーションが正しく表示されているかどうか、また (描画領域内などで) グラフィックのプログラミングが動作しているかどうかを常に目で確認する必要があります。ただし、ウィジェットベースのテスト方法を使用すると、アプリケーションの必要な部分だけに集中することができます。

テストの方法

これまでの経験から、テストスクリプトを作成する方法には、以下の 3 つの段階があります。

- すでに記録したスクリプトの記録および再生
- 大規模なテストの細分化
- データ駆動型テスト

各方法の説明は次のとおりです。

すでに記録したスクリプトの記録および再生

ユーザーアクションが記録されたとおり正確に再生されているかどうかを調べるには、この方法が最も簡単です。ただし、スクリプトが非常に大きくなり、保守が困難になる場合があります。また、テストのどの部分が失敗したかを特定することが難しい場合もあります。特に、アプリケーションに変更が加わると、スクリプト全体を再度記録しなければなりません。

スクリプトの細分化

ここでは、大規模なスクリプトを小さく独立したスクリプトに分割し、各スクリプトでアプリケーションが識別可能な部分をテスト実行します。これは非常に効果的なテストテクニックなので、525 ページの「テストマクロの使用」に詳細な例を記載しています。プリプロセッサ (`m4` または `cpp` など) または使い慣れたプログラミング言語を使用して、細分化された各スクリプトを展開することができます。この方法で、以下のようなスクリプトを構築することができます。

```
StartApplication()

OpenFile(foo.c)

CloseApplication()
```

次に、プリプロセッサ、インタプリタ、コンパイラのうちのいずれかがこれらの細分化スクリプトを完全な Sun WorkShop Visual 再現コマンド群に変換します。

この簡単な方法によって段階的なプログラミングは必要なくなり、さらに、テストスクリプトも管理が非常に簡単になります。

テストを表現するために使用する言語は、注意して選択してください。主な選択基準は、次のとおりです。

- 式が簡単であること
  その言語を使用したインタフェースの記述が難しいと、モデルを書くことも読むことも、また理解することも難しくなります。
- 慣れていること
  使い慣れている言語を使用すると、モデルに集中することができます。

このためには、`Java` や `Python` などのクラスベースの言語が最適です。それ以外の有力候補としては、`Lisp` や `Prolog` などの記号処理向けのモデル化言語があります。`m4` などのプリプロセッサ、または C プリプロセッサでも高速に処理することができます。

または、アプリケーションが使用する言語でモデルを構築してもよいでしょう。こうすると、ソフトウェアを移植する場合に常に使用することができます。目安としては、一連のテストを設計しているのではなくプログラムを記述しているように感じたら、必ず他により簡単な方法があるということです。

このテスト方法は、小規模から中規模までの大部分のアプリケーションに適用できます。しかし、大規模アプリケーション (Sun WorkShop Visual がその一例) の場合は、細分化にも次のような限界があります。

- 細分化スクリプトが多数になってしまいがちである
- 多数の細分化スクリプトが異なるダイアログで同様のテストを行なってしまう
  アプリケーション内のダイアログごとに細分化スクリプトを開いているようになります。
- 細分化スクリプトは関数で記述的であるため、目的以外の方法では使用することができない
  細分化スクリプトにはダイアログに関する有益な情報を多く含めることができますが、目的どおりの方法でしか使用することができません。

次に、これらの問題を解決する方法を説明します。

データ駆動型テスト

これまで Sun WorkShop Visual 用のテストを計画してきた経験から、テストを記述する最も費用効果の高い方法は、各ダイアログの記述を用意してその記述をテストで使用することです。以下に示す Sun WorkShop Visualの「カラー」ダイアログが記述された例について考えてみます。

```
ColorDialog.shell               = my_color_shell
ColorDialog.helpbutton          = color_help
ColorDialog.applybutton         = color_apply
ColorDialog.quit                = color_quit
```

左側の名前でダイアログのウィジェットを間接参照できます。右側の名前は特定のウィジェットの名前です。これらの記述は、以下に示すようなダイアログの名前に渡すだけで、汎用関数に使用できます。

```
                CheckHelpFor(ColorDialog)
Close(ColorDialog)
```

以下に、`CheckHelpFor` と `Close` の定義を示します。

```
#define CheckHelpFor(dialog)
 in dialog.shell
                push dialog.helpbutton
```

```
                    #enddef
#define Close(dialog)
 in dialog.shell
                        push dialog.quit
#enddef
```

これらの関数を、ColorDialog にリストされた記述を含むダイアログに使用して、「ヘルプ」ボタンと「閉じる」ボタンが提供されているかどうかを確認することができます。

このテクニックを使用すると、インタフェースの記述とその記述を使用するアクションを分離することができます。また、インタフェースに変更を加えても、関連するデータ記述に変更を加える必要があるだけで、テストスクリプトは変更なくそのままです。新規ダイアログをアプリケーションに追加する際は、その記述およびダイアログで実行される非標準操作を作成するだけで十分です。

このような方法の最大の利点は、記述が簡単で明確であること、また、デザイン自体に非常に近いため、テストと製品開発の同期をとることで、確実で簡単なテストが実行可能であることです。

テストの成功と失敗の判定

よいテストとは、アプリケーションで調べる対象部分を分離するよう設計されたテストです。アプリケーションが停止しなければ、そのテストは成功です。それ以外の場合は失敗です。

自動再生それ自体が、1 個のユーザーアクションに対して起こり得る 1 つの結果をテストするという意味で、最小単位のテストと言えます。一連の動作がエラーなしで再生されると、予想した内容が実際に行われたということになります。

ファイルを開くというアクションを考えてみます。最小形式のテストでの予測される結果は、ファイルが開かれ、すべての処理が正常に行われることです。ただし、このテストは決して完全ではありません。以下のような別の結果も考慮する必要があります。

- ファイルを開くことができない場合はどうなるか
- ファイルの読み込み時にステータスインジケータの表示が変わるか
- ファイルが読み取り専用の場合に警告ダイアログは表示されるのか

テスト用スクリプトでの制御フローおよび式の使用

最も簡単なテストは、アプリケーションでの一連のアクションを記録し、次にそのスクリプトを再生してそれらのアクションを複製することです。スクリプトが正常に実行されると、そのアプリケーションには信頼性があると言えますが、基本スクリプトの機能を増やすために Sun WorkShop Visual 再現機能によって提供される制御フローおよび式の追加コマンドを利用すると、さらに信頼性を高めることができます。これらのコマンドを使用すると、各種ディスプレイへの対応、ウィジェットのリソース設定の検査、メッセージ出力などを行うことができます。

アプリケーションをモノクロディスプレイで実行している場合はメッセージが表示されるが、フルカラーディスプレイで実行している場合はメッセージが表示されない、という場合を考えてみます。

もちろん、ディスプレイごとの個別テストは行う必要はありません。代わりに、メッセージを表示したい位置にコマンドを挿入し、これらのコマンドを以下のように if 文で囲みます。

```
if !IsPseudoColor
                message Non PseudoColor display
                in warning_popup
                push warning.OK
endif
```

これと同じ検査処理は、使用しているディスプレイのハードウェアやウィンドウマネージャとは関係なく行われます。

また、アプリケーションのダイアログのサイズも重要です。2 つのダイアログを同時に表示する場合は、あるディスプレイでは両方とも完全に表示され、別のディスプレイでは重なりあって表示され、3 番目のディスプレイでは一方のダイアログの上に別のダイアログが置かれます。この結果、アプリケーションの中のあるモードの警告メッセージがメインダイアログの下に隠れてしまい、アプリケーションがロックして見えることがあります。

以下のテストスクリプトの一部で、このような問題の解決方法を説明します。

```
if !IsVisible(open_file_dialog)
                error The Open File dialog is off screen
endif
```

各種ディスプレイを処理する方法については、950 ページの「表示式」を参照してください。

アプリケーションが各種ディスプレイ上で異なる動作をする場合は、テストもこれに対応するように作成する必要があります。たとえば、ユーザーにディスプレイの品質が下がることを通知する警告ダイアログがアプリケーションから表示される場合があります。

今度は、オプションメニューからのオプションの選択について考えてみます。標準スクリプトが確実に選択を行うので、よいテストスクリプトの場合は、選択が行われたかどうかを検査します。

以下の例で、Sun WorkShop Visual の「コード生成」ダイアログで「言語」オプションが予想された値に設定されたかどうかをテストする方法を示します。

```
if !languageOption->menuHistory:'cppButton'
                message FAIL: Language option error.
                printres languageOption->menuHistory
                message expected cppButton
endif
```

テスト失敗時の対処

テストに失敗した場合にとる方法は、次の 3 つがあります。

- テストは中止するが、アプリケーションはそのままの状態にする
- テストを中止し、アプリケーションを終了する
- テストを強制終了し、テスト予定の中の次のテストを実行する

それぞれの処理は、特定の visu_replay コマンド行オプションに対応します。

- `-user-on-error` - アプリケーションは終了しない
- `-exit-on-error` - アプリケーションを終了する
- `-skip-on-error` - 次のテストまでスキップする

失敗に対しては、スクリプト内にその処理を作成することが最善の方法です。条件文を使用して、失敗時には正しいアクション (メッセージを出力するなど) をとってください。

上記以外で、テスト失敗の特定に役立つのが -v コマンド行オプションです。このオプションを使用すると、スクリプトからのコマンドは実行されたとおりに標準出力に表示されます。問題点を特定したら、スクリプトをさらに小さくして再度テストします。今度は (任意のデバッガとともに) このスクリプトを使用して問題点を特定します。また、バグが修正されていることを証明するために、一連の回帰テストにこのオプションを追加することもできます。

テストマクロの使用

繰り返し使用されるアクション (ダイアログを開く、アプリケーションを起動する、テキストフィールドへ入力するなど) を定義するマクロを使用してテストスクリプトをモジュール化する方法については、519 ページの「テストの方法」で説明しました。この方法によって、スクリプトはより簡単に手動で作成、修正、検査することができます。

マクロを使用したスクリプト例

マクロを使用してスクリプトをモジュール化する方法を説明するために、例として以下に Sun WorkShop Visual のテストスクリプトからの抜粋を示します。この短いスクリプトは以下のアクションを行います。

1. Sun WorkShop Visual の起動
2. シェルおよびフォームを含むデザインの作成
3. フォームに対する変数名の指定
4. ファイル名を指定したデザインの保存
5. Sun WorkShop Visual の終了

最上位スクリプトがより読みやすくなるように上記のテストを行うには、マクロプリプロセッサ `m4` を使用します。このプリプロセッサはすべての UNIX システムで使用することができます。

上記のテストを行う上位スクリプトは、次のようになります。

```
include(Defs.m4)

StartUp()

shell   date

Palette(xd_XmDialogShell)
```

```
Palette(xd_XmForm)
VariableName(myform)
SaveDesignAs(mydesign.xd)
message Test Sequence Over
shell   date
Finish()
```

前述のスクリプトの大部分がマクロ呼び出しで構成されています。構文中のキーワードについては、929 ページの付録 A「Sun WorkShop Visual 再現機能の コマンド構文」を参照してください。

Defs.m4 というマクロ定義スクリプトは、以下のとおりです。

```
define(HandleExpectedWarning,
        in warning_popup
                push warning.OK)
define(StartUp,
        if !IsPseudoColor
                message Non PseudoColor display
                HandleExpectedWarning()
        endif)
define(Palette,
        in ApplicationShell
                push $1)
define(VariableName,
        in ApplicationShell
                        multiclick nb_vn_t
                        type $1
                        key Return)
define(SaveDesignAs,
        in ApplicationShell
                cascade file_menu
                        select fm_menu.fm_saveas
```

```
                in save_dialog_popup
                        doubleclick Text
                type $1
                         push save_dialog.OK)
define(Finish,
        in ApplicationShell
                cascade file_menu
                                select fm_menu.fm_exit
                if in save_changes_dialog
                        push xd_question.xd_question_cancel_b
                endif)
```

次のコマンドは、Sun WorkShop Visual 再現機能に渡される最終スクリプトファイルを作成します。

```
m4 Test.in > Test.xds
```

ファイル *Test.in* が上位スクリプトです。*Test.xds* は出力ファイルで、このファイルに展開されたマクロを持つ最終スクリプトが含まれます。

前述したファイルに入力し、次に `m4` を使用して最終スクリプトファイルを作成し、この例をテストします。テストが終了したら、次のように Sun WorkShop Visual を使用し、*Test.xds* を再生対象のスクリプトとして指定して Sun WorkShop Visual 再現機能を実行します。

```
visu_replay -f Test.xds visu
```

さらに一歩進んで、個々のファイルにSun WorkShop Visual のウィジェットパレット上のウィジェットの名前を定義し、次に「shell」や「form」などのより抽象的な名前からウィジェット名が特定できるように「Palette」マクロを定義します。

```
define(shell, xd_XmDialogShell)
define(form, xd_XmForm)
```

これで、内部的な名前が一箇所に保持されるので、保守および変更をより簡単に行うことができます。

Sun WorkShop Visual 再現機能を使用したデバッグ

アプリケーションが間接アプリケーションの場合は、コマンド行から Sun WorkShop Visual 再現機能を実行することにより、複数のアプリケーション名を指定することができます。

たとえば、次のコマンドでは、`AnApplication` で指定したアプリケーションで `dbx` を実行します。

```
visu_replay -f MyScript dbx AnApplication
```

任意のデバッガを使用することができます。Sun WorkShop Visual 再現機能を使用すると、デバッグを開始したい位置にすばやく移動できます。デバッガに入るには、スクリプトの末尾に移動するか、スクリプトに「ブレークポイント」を入力します。「ブレークポイント」とは、その後にウィジェット名が続くキーワードです。指定されたウィジェットが活性化されると、アプリケーションはデバッガに入ります。

Sun WorkShop Visual 再現ウィジェットセットの拡張

概要

Sun WorkShop Visual 再現機能は、スクリプトに記録されたアクションはすぐにユーザーアクションとして認識できなければならないという方針に基づいています。

ユーザーとウィジェットとの対話方法 (マウスボタンの 1 つをクリックするなど)、またはキーボードからの入力内容など、Motif アプリケーション内で行われる大部分のアクションを記述することができます。この記述によって、Motif アプリケーションの記録および再現が非常に簡単になります。また、テストを行うユーザーもスクリプトを理解、プログラミング、保守することが簡単になります。アプリケーションで使用される非標準ウィジェットについても同様です。

この方法がすぐに適用できない Motif ウィジェット (ウィジェット内での位置が重要なもの) も多数あります。それらのウィジェットの多く (スケール、スクロールバーなど) は、1 つの機能がそのウィジェットのすべてのインスタンスに対して機能します。描画領域あるいはその他のカスタムウィジェットでは、ウィジェットのインスタンスごとにまったく動作が異なる場合があります。

たとえば、記録用ソフトウェアは描画領域内のクリックを認識しますが、ユーザーはこのアクションを別の見方でとらえています。ユーザーはアプリケーションが描画領域に描画したオブジェクトと対話しています。しかし、これらのオブジェクトはアプリケーションのコード内にのみ現われます。インタフェースの一部ではありません。

アプリケーションプログラマであれば、特定のカスタムウィジェットまたは描画領域内でのクリックの意味が正確に分かるので、ウィジェットの記録と再現を行うユーザーが理解および使用できるように、これらのアクションを記述するルーチンを簡単に作成することができます。

描画領域かそれ以外のカスタマイズされたウィジェットをプログラミングしている場合は、ウィジェット内の (x,y) 座標にあるイベントを、特定のアクションに変換するコードがあらかじめ作成されています。Sun WorkShop Visual 再現が提供するインタフェースを使用してコンバータを登録すると、専用のルーチンを使用してテストを行うことができます。

Sun WorkShop Visual 再現機能には 2 つのコンバータを設定する必要があります。1 つはある (x,y) 座標のイベントをアクションに変換する記録用コンバータで、もう 1 つはそのアクションを (x,y) 座標に変換する再現用のコンバータです。

変換ルーチンを使用すると、(x,y) 座標をユーザーとウィジェット自体の両方にとって意味のあるものにマップすることができます。

注 - 一部のウィジェットは、ウィンドウ (x,y) 座標とウィジェット内部の構造体との間の変換を簡単に行う方法 (*XmListYToPos* 関数など) を提供しています。これは望ましい方法です。これ以外のウィジェットはウィジェットの状態の判断および変更を行う方法を提供しています。たとえば、*XmScrollBarGetValues* を使用してスクロールバー上のユーザーアクションを記録したり、*XmScrollBarSetValues* を使用してそのアクションを再現します。イベントを実装することが難しい場合は、コンバータを使用してウィジェットを直接プログラミングすることができます。

同じ機能がウィジェットクラス (他社のウィジェットなど) とカスタムウィジェット (Motif XmDrawingArea ウィジェットなど) の両方に使用されています。

以下の 2 つの節で、コンバータルーチンについて説明します。また、Motif XmList ウィジェットクラス用のコンバータの作成例を示します。

イベントから名前または属性への変換ルーチン

名前

xdsXyToNameProc - イベントから名前/属性の記述への変換に使用される関数のインタフェース定義

形式

```
typedef int (*xdsXYToNameProc) (
                Widget widget,
                int    x,
                int    y,
                char**  name_p,
                char** attribute_p )
```

入力内容

widget ルーチンを使用するウィジェット

x イベントの x 座標

y イベントの y 座標

name_p ウィジェットの一部を識別する文字列への (戻り) ポインタ

attribute_p 名前に追加する文字列 (`center`、`left`、`right` など) への (戻り) ポインタ

用法

ルーチンは失敗すると 0 を、成功すると 1 を返します。*name_p* および *attribute_p* に割り当てられた文字列は、Sun WorkShop Visual 再現機能によっては解放されません。コピーが行われるため、静的記憶域を使用することができます。

ルーチンが失敗すると、エラーが報告されます。

名前または属性からイベントへの変換ルーチン

名前

xdsNameToXYProc - 名前/属性の記述からイベントへの変換に使用される関数のインタフェース定義

形式

```
typedef int (*xdsNameToXyProc) (
                    Widget widget,
                    char*  name,
                    char* attribute,
                    int*  x_p,
                    int*  y_p )
```

入力内容

widget ルーチンを使用するウィジェット

name ウィジェットの一部を識別する文字列

attribute 名前に追加する文字列 (`center`、`left`、`right` など)

x_p x 座標結果への (戻り) ポインタ

y_p y 座標結果への (戻り) ポインタ

用法

ルーチンは失敗すると 0 を、成功すると 1 を返します。

注意事項

再現したい結果をウィジェット上に直接プログラムすることが非常に簡単な場合もあります。たとえば、リソース値の設定や簡易関数の呼び出しなどによってプログラムする場合がそうです。ただし、このような場合は一連のイベントを正確に模倣するのは非常に困難です。その際には、このルーチンで処理することができます。

ただし、成功を返して、(x,y) 座標を負の値に設定する必要があります。通常は、ウィジェット内のクリック動作をシミュレートすると、結果の座標を正数と見なします。

例

以下の例は Sun WorkShop Visual 再現機能のソースに対するのものです。この例はウィジェットのクラス (ここでは Motif XmList ウィジェットクラス) 用のコンバータを登録する方法を示したものです。XmList ウィジェットでのクリックが、そのリスト内の要素の特定インスタンスの選択に変換される様子を示しています。これは、Sun WorkShop Visual 再現機能が XmList ウィジェットに対して実際に使用する機能です。この機能の使用例を以下のスクリプトに示します。

```
in my_shell
                        push my_list_widget(‘this line’,1)
```

このスクリプトが再現されると、ウィジェット内の適切な (x,y) 座標でクリックがシミュレートされます。

この例を実習すると、以下の 3 点が可能になります。

- 他社のウィジェットを統合する
- 変換機能を使用し、`widget(name,attribute)` 表記を理解する
- XmList ウィジェットクラスの実装方法を理解する

この例のソースファイルは、メークファイルとともに `$VISUROOT/src/examples/replay/cvtXm` ディレクトリに用意されています。`$VISUROOT` は、Sun WorkShop Visual のインストールディレクトリです。

このディレクトリの内容は、以下のとおりです。

ファイル	説明
`Makefile`	異なるプラットフォームに共有オブジェクトを構築することができる
`README`	共有オブジェクトの構築方法を説明する
`motif*.c`	変換ルーチン
`register.c`	登録ルーチン
`xds*`	サポートファイル

サポートファイルは、共有オブジェクトが Sun WorkShop Visual 再現メカニズムと通信するための枠組みを提供します。これらのファイルを変更する必要はありません。

この例では、XmList コンバータを含む *motif2.c* およびこれらのコンバータを登録するためのコードを組み込んでいる *register.c* について学習します。

この例では、Sun WorkShop Visual 再現のウィジェットセットを拡張する方法を 3 段階で説明します。

1. イベント (x, y) から名前/属性対への変換: `xdsXYToNameProc`
2. 一対の名前/属性からイベント (x, y) への変換: `xdsNameToXYProc`
3. コンバータの登録: `xdsRegisterContextHandler`

関連する 3 つのルーチンは以下のとおりです。

1. `xdsListXyToName()`
   (x,y) 位置を取得し、一対の名前/属性を返します。
2. `xdsListNameToXy()`
   一対の名前/属性を取得し、(x,y) 位置を返します。
3. `xdsRegister()`
   ウィジェットおよび関連するコンバータを Sun WorkShop Visual 再現に登録します。

重要なことは、コンバータコードの構造およびコンバータの登録方法であり、XmList コンバータの実装方法ではありません。

xdsListXyToName()

この関数は *motif2.c* にあり、(x,y) 座標を名前/属性対に変換します。ここでは、xdsXyToNameProc インタフェース定義の構造体を説明しています。この関数は Sun WorkShop Visual 再現が記録モードのときに使用されます。

```
int
xdsListXyToName( widget, x, y, namep, attrp)
                 Widget widget;
                 int x, y;
                 char ** namep;
                 char ** attrp;
{
                 extern Boolean XmStringCompare();

                 extern char *  xdsCvtXmStringToString();
                 extern Boolean xdsCvtSetListError();
                 extern int     xdsCvtListFailure();
                 extern Boolean xdsCvtGetXmListEntries();

                 extern int XmListYToPos();

                 static char name[255];
                 static char count[20];

                 int pos;
                 int len = 0;
                 int n;

                 int instance = 1;
                 XmString * list = (XmString*)0;
                 XmString   item;
```

```
    /* 要素を取得する */
    if (!xdsGetXmListEntries[1]( widget,&list, &len)) {
        return xdsListFailure();
    }

    /* XmListYToPos() を使用してリスト要素を取得する */
    pos = XmListYToPos[2]( widget, (Position)y);
    if (pos < 0 || pos > len) {
        xdsCvtSetListError(LIST_OUT_OF_BOUNDS);
        return xdsCvtListFailure();
    }

    item = list[--pos];
    for (n = 0; n < pos; n++) {
        if (XmStringCompare( item, list[n]) == True)
            instance++;
    }
    /* 記述を作成する */
    (void) sprintf ( count, "%d", instance);

    (void) strcpy  ( name, xdsXmStringToString[3](item));

    *namep  = name;
    *attrp  = count;

    return 1;
}
```

1. xdsGetXmListEntries() - リスト内の要素を返します。これは XmList の要素を取り出す簡単なルーチンです。
2. XmListYToPos() - Motif 簡易関数の XmListYToPos (widget,y) が必要な変換をすべて行います。
   イベントの y 座標を取得し、リスト内の位置を返します。
3. xdsXmStringToString() - XmString を String に変換するルーチンです。

xdsListNameToXy()

この関数も *motif2.c* にあり、名前/属性対を (x,y) 座標に変換します。ここでは、*xdsNameToXyProc* インタフェース定義の構造体を説明しています。この関数は *xdsListXyToName()* の補足関数です。Sun WorkShop Visual 再現機能の再生モードのときに使用されます。

```
int
xdsListNameToXy( widget, name, attr, xp, yp)
                 Widget widget;
                 char * name;
                 char * attr;
                 int  * xp;
                 int * yp;
{
                 extern char * xdsCvtXmStringToString();
                 extern Boolean xdsCvtSetListError();
                 extern int    xdsCvtListFailure();
                 extern int    xdsCvtSetListItem();
                 extern Boolean xdsCvtGetXmListEntries();

                 Position x, y;
                 Dimension w, h;
                 int pos;
                 int len = 0;
                 int n;
                 char * s;
                 int instance = 1;
                 XmString * list = (XmString*)0;
                 XmString   item;

                 if ((instance = atoi(attr)) == 0) {
                     xdsCvtSetListError(LIST_BAD_INSTANCE);
```

```
                    return xdsCvtListFailure();
                }
                instance--;

                if (!xdsCvtGetXmListEntries( widget, &list, &len))
{
                    xdsCvtSetListError(LIST_EMPTY_LIST);
                    return xdsCvtListFailure();
                }

                for ( n = 0; n < len; n++) {
                    s = xdsCvtXmStringToString(list[n]);
                    if (strcmp( name, s) != 0)
                        continue;
                    if (instance--)
                        continue;
                    break;
                }

                if (n == len) {
                    xdsCvtSetListError(LIST_ELEMENT_NOT_FOUND);
                    return xdsCvtListFailure();
                }
                (void) xdsCvtSetListItem( widget, n+1);

                if (!XmListPosToBounds[1]( widget, n+1, &x, &y, &w,
&h)) {
                    xdsCvtSetListError(LIST_OUT_OF_BOUNDS);
                    return xdsCvtListFailure();
```

1. XmListPosToBounds() - Motif 簡易関数の XmListPosToBounds() はリスト内の特定項目のウィンドウ境界ボックスを返します。このボックスを使用して、当該要素に対するクリックの考えられる (x, y) 座標を計算します。

```
                }


                *xp = x + (w/2);
                *yp = y + (h/2);
                return 1;
}
```

xdsRegister()

この関数は、*register.c* と呼ばれています。*motif2.c* の 2 つのコンバータおよび XmScrollBar、XmScale、XmDrawingArea の各ウィジェット用のコンバータを登録します。

```
void
RegisterWidgets()
{
                extern Boolean xdsRegister();
                extern int xdsListNameToXy();
                extern int xdsListXyToName();
                extern int xdsScrollBarNameToXy();
                extern int xdsScrollBarXyToName();
                extern int xdsScaleNameToXy();
                extern int xdsScaleXyToName();
                extern int xdsDaNameToXy();
                extern int xdsDaXyToName();


                (void) xdsRegister( "XmList", xdsListNameToXy,
xdsListXyToName);
                (void) xdsRegister( "XmScrollBar",
xdsScrollBarNameToXy, xdsScrollBarXyToName);
                (void) xdsRegister( "XmScale", xdsScaleNameToXy,
xdsScaleXyToName);
```

```
                (void) xdsRegister( "XmDrawingArea", xdsDaNameToXy,
xdsDaXyToName);
}
void RegisterThisListWidget(
                Widget w;
{
                xdsRegisterContextHandler(w, xdsListNameToXy,
xdsListXyToName);
}
```

この関数は xdsSetup.h に定義されています。ここでは、xdsRegisterContextHandler インタフェース定義の構造体を説明しています。

```
Boolean
xdsRegister( classname, name2xy, xy2name)
                char * classname;
                int_f  name2xy;
                int_f  xy2name;
{
                bool_f bf = xdsGetRegisterFunction();

                if (!bf)
                    return False;

                return (*bf)( classname, name2xy, xy2name);
}
```

例の構築

提供されている *Makefile* は、共有オブジェクトを構築するよう構成されています。多数のオペレーティングシステムがサポートされています。これらのシステムをリストするには、次のように入力します。

```
make
```

共有オブジェクトは、メークファイルの OBJECT 行を次のように変更するだけで構築することができます。

OBJECT = cvt<クラス名>

クラス名には、ウィジェットクラスの接頭辞を指定します。この例では、ウィジェットクラスは XmList なので、次のように Xm 接頭辞を入力します。

OBJECT=cvtXm

共有オブジェクトを構築するには、**make** <システム名> と入力します。たとえば、Solaris マシンの場合は、**make solaris** と入力します。これで、*libcvtXm.so* という共有オブジェクトが構築されます。

共有オブジェクトが構築されたら、そのオブジェクトを $VISUROOT/lib/xds というディレクトリにコピーまたはリンクします。これで、Sun WorkShop Visual 再現機能が必要に応じてこのオブジェクトを読み込むことができます。

コンバータを登録するためのソースファイルおよびメークファイルは、$VISUROOT/src/examples/replay/cvtTemplate に用意されています。$VISUROOT は、Sun WorkShop Visual のインストールディレクトリです。

カスタムウィジェット用のコンバータの追加

前述の例は、ウィジェットクラスに関するものです。カスタマイズ可能なウィジェット (すなわち、Motif XmDrawingArea などのウィジェットの特定のインスタンス) の場合は、変換ルーチンを登録するための機能が Sun WorkShop Visual とともに提供されています。この機能を使用すると、Sun WorkShop Visual 再現機能がウィジェットのインスタンス (Motif であるかどうかに関わらず) の中で行なったユーザーアクションを記録および再現できるように、Sun WorkShop Visual 再現機能の動作をカスタマイズすることができます。

コンバータ登録コードは、以下のとおりです。

```
int_f _xdsRegisterFunction = (int_f)0;

Boolean
xdsRegisterContextHandler( widget, name2xy, xy2name)
                Widget widget;
                int_f  name2xy;
                int_f  xy2name;
{
```

```
                if (!_xdsRegisterFunction)
                    return False;
                return (*_xdsRegisterFunction)( widget, name2xy,
xy2name, True);
}
```

この関数がアプリケーションの中で呼び出される必要があります。

注 - _xdsRegisterFunction 関数のポインタ変数は 0 に設定されています。これは、アプリケーションが Sun WorkShop Visual 再現機能を使用しないで実行される場合は、ルーチンは常にリターンし、アプリケーションでは何も行わないことを意味します。アプリケーションを Sun WorkShop Visual 再現機能とともに実行する場合は、変数は Register ハンドラを指すよう設定されるので、このルーチンが呼び出されます。

コンバータを登録するルーチンについては、以下で説明します。

コンバータの登録

名前

xdsRegisterContextHandler - コンバータの登録に使用する関数のインタフェース定義

形式

```
Boolean xdsRegisterContextHandler(
                Widget widget,
                xdsNameToXYProc name2xy,
                xdsXYToNameProc xy2name)
Boolean xdsRemoveContextHandler( Widget widget)
```

入力内容

widget ルーチンを使用するウィジェット

name2xy 再現用の変換関数へのポインタ

xy2name 記録用の変換関数へのポインタ

機能説明

このルーチンは、ウィジェットにおけるイベントの独自解釈を登録するためのものです。以下の例に示すように、ウィジェットをコード内に作成した後は、いつでも `xdsRegisterContextHandler` を呼び出すことができます。

```
                button1 = XmCreatePushButton ( shell1, "button1",
al, ac );

                xdsRegisterContextHandler(shell1, func1, func2)
```

Sun WorkShop Visual 再現機能は、再現時は `func1` ルーチンを呼び出し、記録時は `func2` ルーチンを呼び出します。

この機能は、ソースファイル (`client.c`) として、またはコンパイル済みライブラリモジュール (`libxdsclient.a`) として使用することができます。ソースファイルの場合はアプリケーションとともにコンパイルし、ライブラリモジュールの場合はアプリケーションと再リンクします。

この機能はアプリケーション自体には影響せず、アプリケーション内に残しても悪影響はありません。

まとめ

Sun WorkShop Visual 再現ウィジェットセットを拡張するには、以下の作業を行います。

- から名前/属性へのコンバータを作成する
- 名前/属性から (x,y) へのコンバータを作成する

ウィジェットクラスの場合は、以下の作業を行います。

- コンバータを登録する

- 共有オブジェクトを構築し、$VISUROOT/lib/xds ディレクトリにコピーまたはリンクする

個々のウィジェットには、以下の作業を行います。

- xdsRegisterContextHandler を呼び出して、ウィジェットを特定の関数に関連付ける
- client.c を使用してアプリケーションを構築するか、libxdsclient.a ライブラリとリンクする

提供されている examples ディレクトリの内容を参考にして、コンバータの記述、登録、構築を行なってください。

独自の Sun WorkShop Visual 再現機能コマンドの追加

概要

Sun WorkShop Visual 再現機能のコマンドセットの使用目的は、ユーザーアクションを再現したり、アプリケーションのウィジェット階層やリソース設定に関するアプリケーションの状態を検査することです。使用できるコマンドセットはこれだけではありません。以下に示すようにこのコマンドセットを拡張して、必要に応じて独自のコマンドを追加することもできます。

- 再現セッションのさまざまな位置で、画面ダンプを出力する場合
- ウィジェット階層に対して別の種類の一貫性検査を行う場合
  Doug Young の widgetlint ライブラリとのインタフェースをとることなどは、その一例です。
- 特定のデバッグ対象の問題に対してプローブまたはパッチを挿入する場合
  これは、ストリップされ最適化されたバイナリのように、デバッガの機能を完全に利用することが不可能な場合に最も役に立ちます。

これまでに、Sun WorkShop Visual 再現メカニズムによって暗黙に読み込まれるユーザー定義モジュールの例を学習してきました。また、このようなモジュールの構成および構築の方法についても学習しています。

import を使用すると、独自のコマンドのモジュールをスクリプトに明示的に読み込むことができます。一度モジュールを読み込むと、モジュール内のコマンドは user コマンドを使用して呼び出すことができます。本節では、このようなモジュールを構築する方法について説明します。この手順は、前節で説明したものと共通点が多くあります。

例

1 つのコマンドを含むモジュールを構築する方法について説明します。このコマンドは、標準エラーにメッセージおよび現在のシェルウィジェットの名前を出力します。独自のコマンドを構成する際に、この例をテンプレートとして使用することができます。

コマンドのソースファイルは、メークファイル (Makefile) とともに `$VISUROOT/src/examples/replay/usertemplate` ディレクトリに用意されています。`$VISUROOT` は、Sun WorkShop Visual のインストールディレクトリです。

このディレクトリの内容は、以下のとおりです。

ファイル	説明
`Makefile`	異なるプラットフォームに追加コマンドモジュールを構築することができる
`README`	コマンドモジュールの構築方法を説明する
`interface.c`	本節で説明する追加コマンドのコードを含む
`xds*`	サポートファイル

サポートファイルは、追加コマンドが Sun WorkShop Visual 再現メカニズムと通信するための枠組みを提供します。変更する必要があるファイルは、`xdsResources.h` ファイルだけです。**「`.xds`」という接頭辞が付いた残りのファイルは、決して変更しないでください。**

モジュールを構築するには、メークファイル中の `OBJECT` 行を変更します。変更後、次のように入力して構築します。

`make <`システム名**`>`**

次に、`$VISUROOT/lib/xds` ディレクトリに、共有オブジェクトをコピーまたはリンクします。

interface.c ファイルの内容は、次のとおりです。

```
#include <stdio.h>
#include <X11/Xos.h>
#include <X11/Xlib.h>
#include <X11/Intrinsic.h>
void
exampleHalloWorld( shell, message)
                Widget shell;
                char * message;
{
                if (!message)
                    message = "no message";
                (void) fprintf ( stderr, "Widget %s says
'%s'\n",XtName(shell), message);
}
```

この例にあるように、ユーザー定義関数は必ず以下の 2 つの引数をとります。

- `shell` - 現在のシェルウィジェット (Sun WorkShop Visual 再現によって引き渡されます)
- `message` - 文字列

メッセージは、スクリプト内の user コマンド構文がある行のコマンドの後に続く文字列すべてです。以下のスクリプトの一部は、コマンドにアクセスし、使用する例を示したものです。

```
import usertemplate
                in ApplicationShell
                    user HalloWorld I'm here
```

この場合、メッセージの文字列は「`I'm here`」です。

インタフェース

すべてのオブジェクトと Sun WorkShop Visual 再現メカニズムとの間のインタフェースは、標準の Xt リソース処理ルーチンを使用して実現されています。

注 - リソースの構造体については、『*X Toolkit Intrinsics Programming Manual*』(O'Reilly and Associates 刊) 第 4 巻の第 10 章などの文献を参照してください。

以下に示すように、新規コマンドの項目が *xdsResources.h* のリソース一覧に追加されます。

```
{
"HalloWorld", XtCCallback, XtRPointer, sizeof
(XtPointer), XtOffsetOf
(data_t,HalloWorld), XtRImmediate,
(XtPointer)exampleHalloWorld
}
```

上記コードで重要な項目は、次の 3 つです。

- `"HalloWorld"` はリソース名である
- `XtOffsetOf(data_t, HalloWorld)` はデータ構造体における関連項目からのオフセットである
- `(XtPointer)exampleHalloWorld` は関数のアドレスへのポインタである

リソースへのポインタは、このファイル内のデータ構造体に追加されます。

```
typedef struct {
                int       type;
                XtPointer setValues;
                XtPointer getValues;
                XtPointer engineSetValues;
                XtPointer engineGetValues;
                /*-----------------------*/
                XtPointer HalloWorld;
} data_t;
```

データ構造体のこの行より上の項目は、Sun WorkShop Visual 再現オブジェクトすべてに共通です。

このファイルの末尾で、関数の宣言を行います。

```
extern void exampleHalloWorld();
```

モジュールの構築

これまでに説明したように、モジュールの構築は、メークファイルの OBJECT 行を変更するだけで実行できます。この例では、次のように変更します。

```
OBJECT=usertemplate
```

そして、次のように入力してモジュールを構築します。

```
make solaris
```

最後に、構築した共有オブジェクトを $VISUROOT/lib/xds ディレクトリにコピーまたはリンクします。

```
cp libusertemplate.so $VISUROOT/lib/xds
```

これで、すべての準備が完了しました。

まとめ

新規コマンドを Sun WorkShop Visual 再現コマンドセットに追加するには、以下のようにします。

1. 関連関数を interface.c ファイルに追加します。
2. 項目を xdsResources.h のリソース一覧に追加します。
3. xdsResources.h にデータ構造体への関数ポインタを追加します。
4. xdsResources.h に関数の外部宣言を追加します。
5. 必要に応じて、メークファイルの OBJECT 行を変更します。
6. モジュールを構築します。
7. 構築したモジュールを $VISUROOT/lib/xds ディレクトリにコピーまたはリンクします。

構築するモジュールの数には制限はありません。また、それらのモジュールはいつでもスクリプトに読み込むことができます。

開発したアプリケーションの録画と再現の許可

自身で作成したアプリケーションを、そのアプリケーションのユーザーが Sun WorkShop Visual 再現機能を使用して録画および再現できるようにするには、次のようにします。

- コードに次の行を追加する

```
xdsAllowUserAccess()
```

- アプリケーションを `libxdsclient.a` ライブラリにリンクする

Sun WorkShop Visual 再現機能の使用方法のヒントについては、924 ページの「Sun WorkShop Visual 再現、捕獲機能」を参照してください。

第 15 章

グループ

はじめに

Sun WorkShop Visual では、1 つ以上のウィジェットを選択してグループにまとめることができます。グループにまとめることで、多数のウィジェットや類似した機能を持つウィジェットを、1 つのまとまりとして簡単に指定できます。また、グループにまとめたウィジェットは、スマートコードの基本的な構築ブロックとして使用できます。スマートコードはツールキットに依存しないコードの階層です。Sun WorkShop Visual では、このスマートコードを生成することによって、Motif アプリケーションを他のプラットフォームに移行してインターネット技術を最大限に活用することができます。

スマートコードでは、その基本データ構造としてグループを使用するため、まずグループの作成方法、カスタマイズ方法、および使い方を理解する必要があります。本章ではグループについて説明します。スマートコードについては以下の章を参照してください。

1. 第 16 章「取得と設定用のスマートコード」(559 ページ)
   この章ではスマートコードについて説明します。また、デザインでのグループの設定方法、「取得/設定」スマートコードの生成方法、および使用しているツールキットとは別の手段でグループメンバーにアクセスする方法についての簡単な学習も含まれています。

2. 第 17 章「thin クライアント用スマートコード」(577 ページ)
   この章では、デザインから thin クライアントおよび別個のサーバーアプリケーションを作成する方法を説明します。サーバーは *CGI* スクリプトであり、クライアントとサーバー間の通信はインターネットの規格を使用することによって実現されます。この章にも、基礎概念に習熟するための学習が組み込まれています。

3. 第 18 章「インターネット用スマートコード」(615 ページ)
   この章では、WWW 上のページにアクセスできるコードをデザインから生成する方法について説明します。実際に試すことのできる簡単な学習が含まれています。

グループの作成

グループを作成するには、デザイン内の任意の数のウィジェットを選択してから、(図 15-1 に示す) ツールバー上の「新規グループに追加」ボタンを押すか、またはウィジェットメニューから「新規グループに追加」を選択します。図 15-2 に示すように、グループエディタが表示されます。

図 15-1 「新規グループに追加」ツールバーボタン

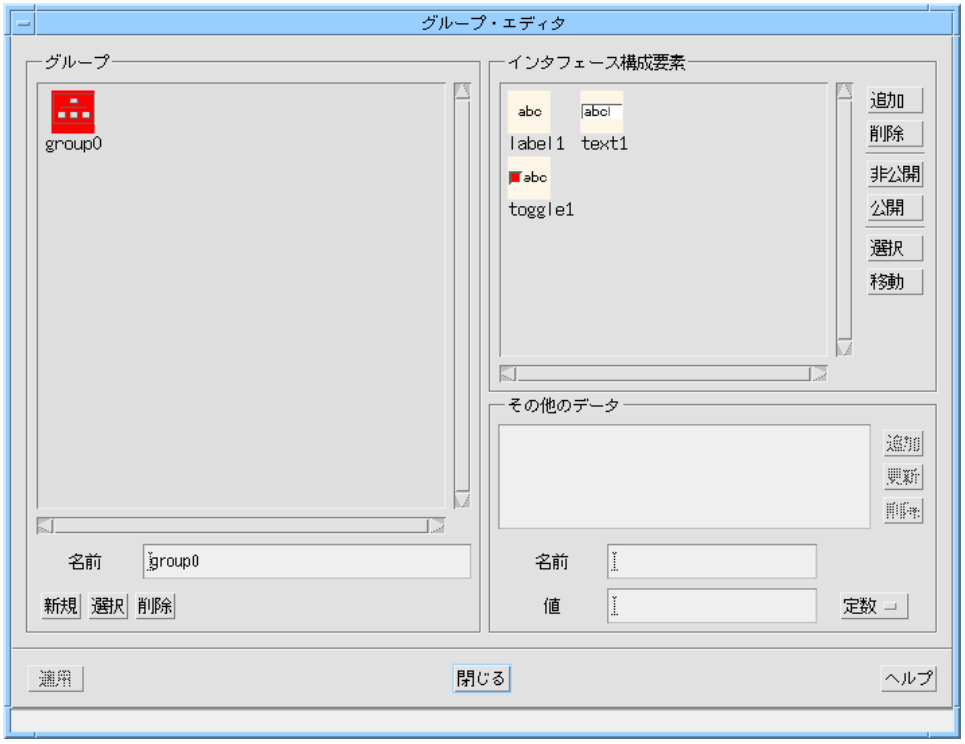


図 15-2 グループエディタ

グループエディタでは、以下の操作が可能です。

1. グループ名の変更
2. グループメンバーの指定

グループの名前を変更するには、グループを選択し、「名前」というラベルの付いたテキストフィールドに新しい名前を入力して Returnキーを押します。この操作の結果はすぐに有効になります。

グループが選択されると、そのメンバーが右側のリストに表示されます。グループメンバーには多くの関数を適用できます。

1. 追加
   デザイン領域で現在選択されているウィジェットを、このグループに追加します。

2. 削除
   選択したメンバーをグループから削除します。

3. 非公開
   選択したメンバーをこのグループに「専用」であると定義します。これはコードがグループ用に生成される方法を表し、thin クライアントのコールバックにだけ関連します。詳細については 554 ページの「公開メンバーと非公開メンバー」を参照してください。

4. 公開
   選択したメンバーを公開し、(サーバーから) アクセス可能であると定義します。これはコードをグループ用に生成する方法を表し、thin クライアントのコールバックにだけ関連します。詳細については 554 ページの「公開メンバーと非公開メンバー」を参照してください。

5. 選択
   デザイン領域で対応するウィジェットを選択します。

6. 移動
   この関数の場合には、グループエディタで選べるメンバーは 1 つだけです。「移動」を押すと、選択したウィジェットがデザイン領域に表示され、必要ならば階層内のノードが展開されます。

ショートカットとしてのグループ

グループは、スマートコード用に使用されたときに本領を発揮します。これについては、559 ページの第 16 章「取得と設定用のスマートコード」を参照してください。しかし、グループを使用すると、その他のこともより簡単に実行できるようになります。グループを使用できる 3 つの領域を次に示します。

高速な複数選択

グループエディタで、グループリストの下にある「選択」ボタンを押すと、デザイン領域でグループのすべてのウィジェットが強調表示されます。これにより、繰り返し選択する必要のあるウィジェットグループを設定できます。グループをこのように使用すれば、余分なコードはわずかしか生成されません。グループはウィジェットの配列として定義されます。

高速検索

グループエディタのグループメンバーリストの横にある「移動」ボタンを使用すれば、Sun WorkShop Visual は、選択したウィジェットをデザイン領域に表示し、必要ならば階層内のノードを展開します。特定のウィジェットだけをグループにまとめておくと、その後の検索が簡単になります。

リンク

図 15-3 に示すように、グループは「リンク編集」ダイアログでのリンク先として使用できます。

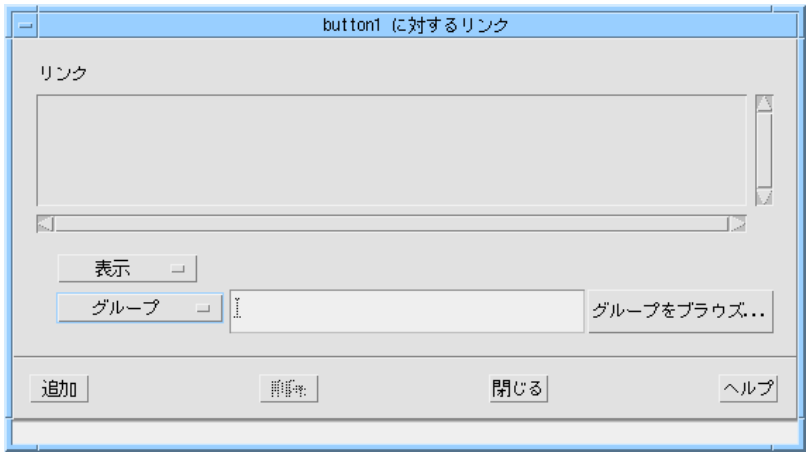


図 15-3 「リンク編集」ダイアログ

グループをリンク先として使用すれば、たとえば、ウィジェットのグループ全体を一度に非表示にしたり無効化する操作を高速かつ簡単に行うことができます。

スマートコード用のグループ

公開メンバー、非公開メンバー、およびその他のデータは、thin クライアントアプリケーションの開発を目的とした「グループ構造」の機能です。詳細については、577 ページの第 17 章「thin クライアント用スマートコード」で説明しています。thin クライアント用スマートコードもインターネット用スマートコードも使用しない場合には、これらの機能を使用する必要はありません。

公開メンバーと非公開メンバー

「公開」と「非公開」のトグルを使用すれば、コードの生成時に各メンバーのアクセスのレベルを制御できます。thin クライアント・サーバーアプリケーションで使用するグループを設定している場合には、これを変更するだけで済みます。

デフォルトでは、グループのメンバーは公開であると定義されます。つまり、グループがクライアントアプリケーションの外部 (たとえばリモートサーバー) に渡された場合には、受信側ルーチンはそれらのメンバーにアクセスできます。

ユーザーインタフェースであるアプリケーションの thin クライアントにだけ、グループの特定メンバーへのアクセス権を与えることもできます。その場合、そのメンバーを「非公開」にしてください。非公開のメンバーは、thin クライアントアプリケーションのどこにあっても認識されますが、サーバー内では認識されません。クライアントアプリケーションはユーザーインタフェースを制御しているため、サーバーの場合よりも多くのメンバーにアクセスが必要となります。これについては 557 ページの「その他のデータ ― 関数」を参照してください。

thin クライアントアプリケーションとサーバーアプリケーションの作成については、577 ページの第 17 章「thin クライアント用スマートコード」を参照してください。

その他のデータ

グループエディタには「その他のデータ」という領域があり、そこではグループに新しいメンバーを追加できます。これらのメンバーは、クライアントとサーバー間のデータのやりとりに使用され、名前と値を組み合わせて指定します。「その他のデータ」メンバーは常に文字列として取り扱われます。

スマートコードのコールバックにはグループだけが渡されるため、グループ内にその他のデータを追加することで、コールバックに渡す情報を増やすことができます。コールバックが独立したサーバーアプリケーション内で機能しており、残りのクライアントアプリケーションには直接アクセスしない場合には、この方法が特に便利です。

グループエディタの「その他のデータ」領域には、既存の定義、データの名前用と値用の 2 つのテキストフィールド、およびオプションメニューを示すリストが含まれており、その他のデータとして「定数」、「変数」、「関数」のどちらかを選択できます。

その他のデータを追加するには、名前と値を入力し、そのタイプを選択してから「追加」を押します。

その他のデータは、グループの他のメンバーとまったく同じ方法で取り扱われます。これらのメンバーには get 関数と set 関数が用意されています。この意味は、定数、変数、関数の場合で少し異なります。以下の項では各「型」の意味について説明します。

その他のデータ ― 定数

図 15-4 に示すようにその他のデータを定義すると、その他のメンバー「myNewConstant」に文字列「hello」が初期設定されます。

値「hello」を取得および設定するルーチンは、コード中に生成されます。グループメンバーの値を取得および設定する方法については、559 ページの第 16 章「取得と設定用のスマートコード」を参照してください。

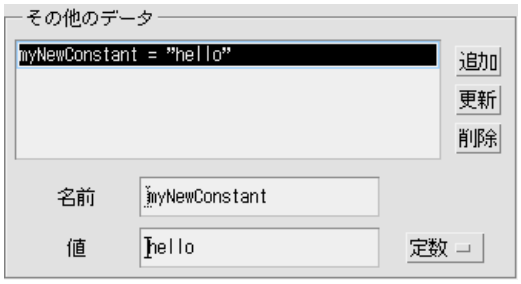


図 15-4 その他のデータ - 定数型

その他のデータ ― 変数

変数値を持つデータをグループに追加するには、既存の変数を「値」として使用し、ユーザー自身の名前を「名前」フィールドに入力します。既存の変数 (「値」フィールドで指定された変数) 用の生成コードファイルに「外部宣言」定義が追加されます。この例を図 15-5 に示します。

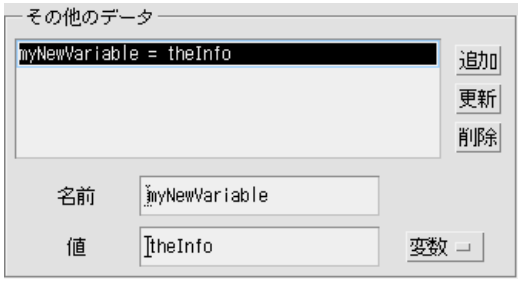


図 15-5 その他のデータ ― 変数型

「theInfo」が文字列変数であり、どこか他の場所で定義されていることを確認する必要があります。このケースで提供される get 関数と set 関数は、変数の値を取得および設定します。たとえば、「theInfo」を次のように定義したとします。

```
char  * theInfo = "hello";
```

さらに、「myNewVariable」を上記のように定義しました。次の行では、文字列「hello」を `str` に割り当てます。

C の場合:

```
char * str = SC_GET(Value, myGroup->myNewVariable);
```

C++ の場合:

```
char * str = myGroup->myMyNewVariable->getValue();
```

Java の場合:

```
String str = myGroup.myNewVariable.getValue();
```

グループメンバーの値を取得および設定する方法については、559 ページの第 16 章「取得と設定用のスマートコード」を参照してください。

その他のデータ — 関数

その他のデータを関数の形で追加できることにより、柔軟性が向上します。「関数」型のその他のメンバーを定義すると、ユーザーが記入した getter 関数と setter 関数を介してアクセスされる値を持つグループにメンバーが追加されます。「名前」フィールドに入力される文字列は、その他のメンバーの名前です。「値」フィールドに入力される文字列は、get ルーチンと set ルーチンの名前に使用されます。たとえば、「名前」フィールドに address と入力し、「値」フィールドに `myAddress` を入力した場合には、「address」というその他のメンバーがグループに追加され、その値は以下のルーチンを介してアクセスされます。

- get_myAddress
- set_myAddress

これを図 15-6 に示します。

図 15-6 その他のデータ — 関数型

Sun WorkShop Visual は、`callouts` サブディレクトリに、グループの名前と「値」フィールドの内容を連結した名前を持つファイルを作成し、これら 2 つのルーチンのスタブをそのファイルに生成します。たとえば、上述の「`myAddress`」ルーチンが「myGroup」というグループ内で定義されていると想定すると、それらを含んでいるファイルの名前は `myGroup_myAddress.c` となります (C コードの場合)。

次の C コード行では「address」の値を取り出します。

```
SC_GET(Value, group->address);
```

このコード行によって、新しいルーチン `get_myAddress` が呼び出されます。値が返されるように、`get_myAddress` を記入してください。その結果、「`SC_SET`」は `set_myAddress` を呼び出します。

注 - 「名前」と「値」のフィールドに同じ名前を入力してもかまいません。グループメンバー名は getter ルーチンと setter ルーチンによって綿密に照合されるため、これによる混乱はそうありません。

グループメンバーの値を取得および設定する方法については、559 ページの第 16 章「取得と設定用のスマートコード」を参照してください。

その他のデータを関数の形で定義のは、グループのデータをサーバーが要求する形に組み立てるためです。たとえば、使用するサーバーではユーザーインタフェースの構成要素が認識されないため、アドレスを 1 つの長い文字列の形にする必要があるとします。しかし、ユーザーインタフェースには、ユーザーがアドレスを入力するためのテキストフィールドがいくつか含まれることがあります。このような場合は、「address」という名前のその他のデータメンバーを関数として定義すると便利です。「名前」と「値」のフィールドに文字列「address」が入力された場合には、`get_address` ルーチンはすべてのアドレスフィールドを結合した値を返します。これにより、サーバー内のコールバックは、「address」の値を通常の方法で取り出すことができます。

```
char * value = SC_GET(myGroup->address);
```

同様に、対応する `set_address` ルーチンは、サーバーから文字列を取り出し、各テキストフィールドに関連する文字列を切り離すことができます。この例では、サーバーは 1 つの「address」メンバーだけに関心を持つため、テキストフィールドウィジェットを「非公開」に、「address」メンバーを「公開」にすることもできます。

グループから入手できるデータをサーバー用に手直しする必要がある場合もよくあります。たとえば、(テキスト全体ではなく) テキスト領域から選択したテキストだけを送信することもできます。

第 16 章

取得と設定用のスマートコード

はじめに

Sun WorkShop Visual では、スマートコードというツールキットに依存しないコードを生成できます。スマートコードは、単独のアプリケーションでも、クライアントとサーバーで構成される「クライアントサーバーアプリケーション」でも使用できます。クライアントサーバーアプリケーションに使用される場合、Sun WorkShop Visual は、Java、Motif (C または C++) あるいは Microsoft Windows の MFC コード、および別個のサーバーアプリケーションを使用して、*thin* クライアントを生成します。Sun WorkShop Visual は、トランスポートプロトコルとして HTTP を用いるコードを生成します。生成されたアプリケーションは、企業のイントラネットで使用したり、インターネット上のさまざまなサイトに接続することができます。

デザインからクライアントサーバーや「Web 対応」のアプリケーションを作成するため、Sun WorkShop Visual は、通信用の基本データ構造として「グループ」を使用し、グループにアクセスする手段としてスマートコードを使用します。グループ機能は、これ以外にもデザインの操作に役立てることができます。グループについては、549 ページの第 15 章「グループ」を参照してください。

スマートコード情報の構成

- 本章では「取得/設定」用スマートコードについて説明します。
- グループの定義と取得/設定用スマートコードの使い方については、568 ページの「取得と設定の学習」を参照してください。

- デザインを thin クライアントとサーバーアプリケーションに分割するように Sun WorkShop Visual に指示する方法については、577 ページの第 17 章「thin クライアント用スマートコード」を参照してください。
- World Wide Web (WWW) からデータを取り出せるアプリケーションの構築方法については、615 ページの第 18 章「インターネット用スマートコード」を参照してください。

スマートコードの使用

デフォルトでは、従来の Motif、MFC、または Java のコールバックが生成されます。スマートコードを生成するには、「コールバック」ダイアログで「スマートコード」トグルを選択してください。

スマートコードにより、コールバックは (言語独立ではなく) ツールキット独立になります。生成したい言語は選択できます。スマートコードは、指定の言語を使用してグループ内のウィジェットを「包み込む」コードの層です。

図 16-1 に示すオプションメニューにより、以下のどちらかのスタイルのスマートコードを選択できます。

1. **取得と設定**
   ウィジェットのグループに対して「getter」関数と「setter」関数を提供します。

2. **thin クライアント**
   getter と setter を提供するだけではなく、Sun WorkShop Visual に、ユーザーインタフェースを含んだ別個のクライアントアプリケーションと、スマートコードのコールバックを含んだサーバーアプリケーションを生成させます。さらに、この 2 つの間の通信プロトコルを処理するコードも生成されます。

3. **インターネット**
   このオプションは、上述の「thin クライアント」オプションにきわめてよく似ていますが、サーバーアプリケーションは生成されません。このオプションは、既存のサーバーに接続するアプリケーションや Web ページを取り出すアプリケーションを生成するために使用します。

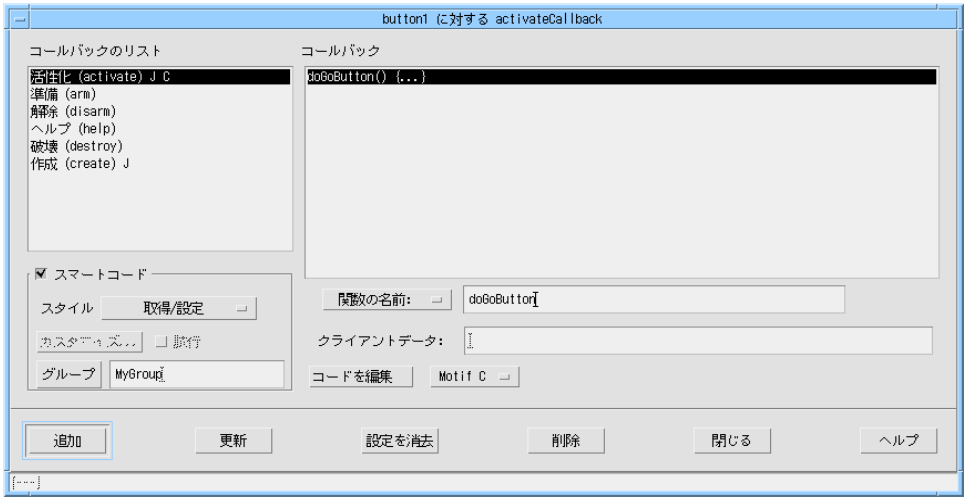


図 16-1 「コールバック」ダイアログでのスマートコード

スマートコードのコールバックにはグループが必要です。このグループは、ツールキットに依存しない方法でプログラムできるオブジェクト群へのハンドルをコールバックに提供します。グループを指定するには、次のどちらかを実行します。

1. グループの名前を「グループ」テキストフィールドに入力する
2. 「グループ」ボタンを押してグループエディタを表示し、グループを選択してから「適用」を押す

「スタイル」オプションメニューを使用すれば必要なスマートコードのタイプを選択できるため、作成するアプリケーションの構造を選択できます。各スタイルで生成されるコードは異なります。以下の項では各スタイルを個々に説明します。

取得と設定用のスマートコード

「コールバック」ダイアログのスマートコードの「スタイル」メニューから「取得/設定」を選択すると、Sun WorkShop Visual は、指定したグループ内の構成要素に対してツールキットに依存しないラッパーを生成します。これらは「getter」関数および「setter」関数とも呼ばれ、ツールキット固有のコードを書かずに、グループの構成要素の値を取得および設定することができます。

たとえば、C を使用しており、図 16-2 に示すように、「MyGroup」というグループに「text1」というウィジェットと「toggle1」というトグルがあるとします。

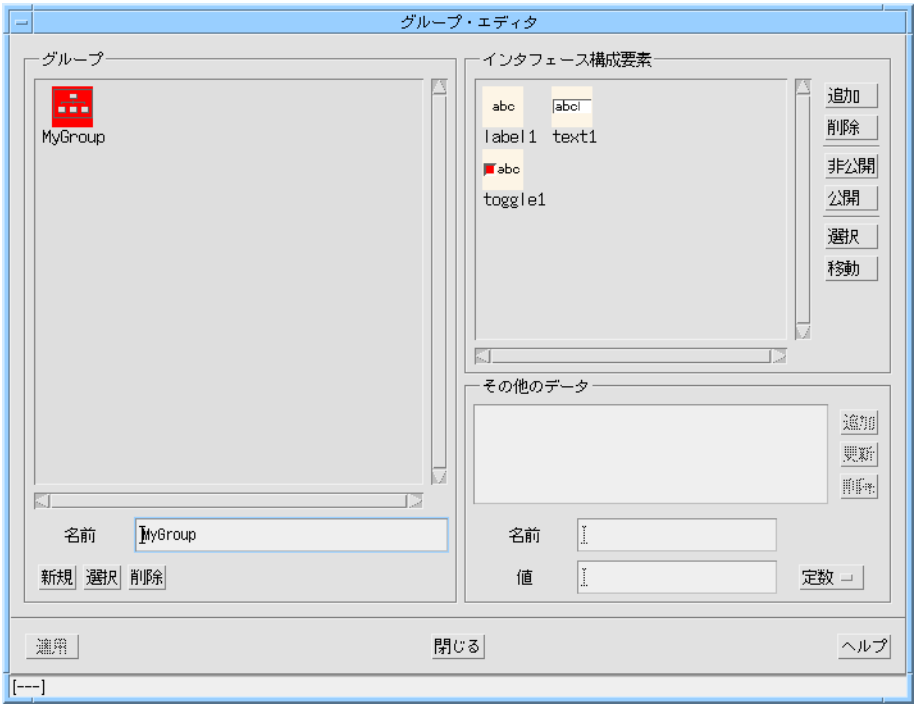


図 16-2 グループエディタに表示されたグループ「MyGroup」

コールバック内のこのグループにアクセスするには、スマートコードのコールバックを定義し、スマートコードのスタイルとして「取得/設定」を選択し、コールバックで使用するグループとして「MyGroup」を指定します。これを図 16-3 に示します。

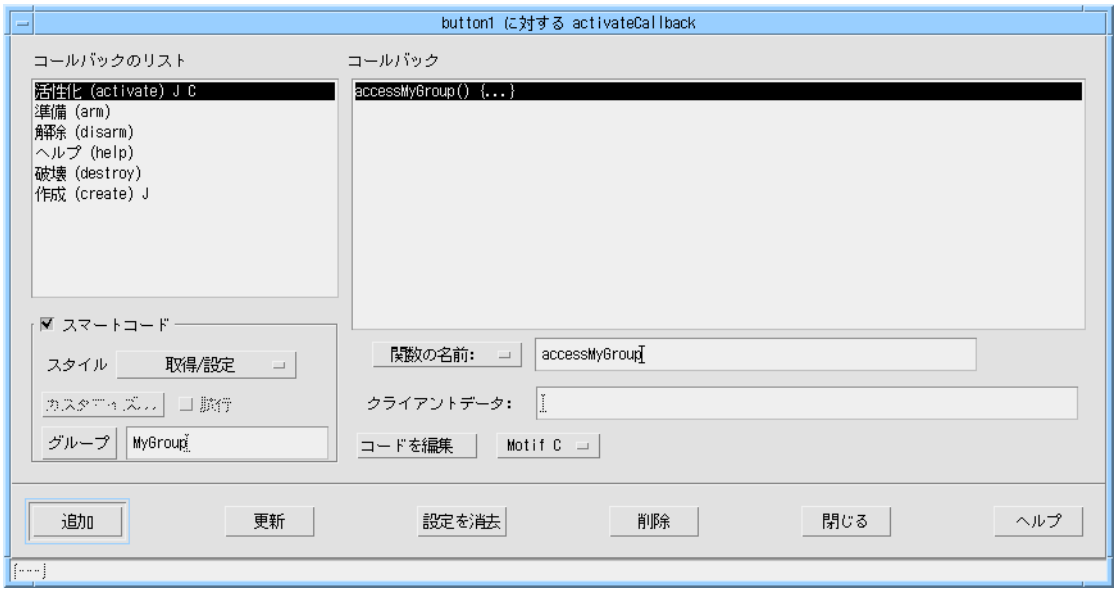


図 16-3 「MyGroup」を使用するコールバック

C によるスマートコードのコールバック

図 16-3 に示すコールバック用に生成されるコードは次のようになります。

```
#include <stdio.h>
#include “sc_groups_c.h”

void
accessMyGroup_user ( d, group, client_data)
        sc_data_t * d;
        MyGroup_t* group;
        void* client_data;
{
}
```

スマートコードのコールバックに渡されるデータ構造 sc_data_t には、主にクライアントとサーバー間の通信に関係する情報が含まれています。この構造の定義は、ヘッダーファイル sc_groups_<言語>.h に含まれています。Sun WorkShop Visual は、このファイルを「コード生成」ダイアログで指定したディレクトリに生成します。スマートコード用に生成されるコードの詳細については 606 ページの「生成コード」、デザインからクライアントとサーバーを作成する方法については 577 ページの第 17 章「thin クライアント用スマートコード」を参照してください。

「text1」の値を得るには、このコールバックに次の行を追加する必要があります。

```
char * val = SC_GET(Value, group->text1);
```

次に示すコードでは、トグルの状態を確認し、トグルがオンになっている場合には「text1」にテキストを入力します。

```
if (SC_GET(State, group->toggle1)
                    SC_SET(Value, group->text1, "The toggle is set");
```

この例では、C を使用しているものと想定しています。C の構文は扱いにくいため、プログラミングのしやすさを維持するためにマクロが使われています。このマクロは SC_SET と SC_GET であり、2 つの引数を受け取ります。最初の引数は参照の種類 (値、状態などの取得しようとしている対象) であり、2 番目の引数は (グループを介して) アクセスしたいウィジェットです。

C++ によるスマートコードのコールバック

「コード生成」ダイアログで言語として C++ を選択した場合には、図 16-3 に示す例に対して生成されるスマートコードのコールバックは、上の例の場合とは異なります。この場合には、Sun WorkShop Visual は、新しいクラス名としてコールバックの名前を使用して、定義済みの内部スマートコードクラス sc_data_c のサブクラスを生成します。クラス sc_data_c は、上述の C データ構造 sc_data_t に対応しています。このコールバックが起動されたときにメインアプリケーションから呼び出されるルーチンは、新しいクラスの doit() メソッドです。以下にディレクトリ callouts_cpp の accessMyGroup_user.cpp の内容を示します。accessMyGroup は、「コールバック」ダイアログに入力されたコールバック・メソッドの名前です。

getGroup() メソッドは、親 sc_data_c クラス内の簡易ルーチンです。

```
#include <stdio.h>

#include "sc_groups_cpp.h"
```

```
class accessMyGroup_user: public sc_data_c
{
        void doit() {
                        MyGroup_c * g = (MyGroup_c *)getGroup();
自分のコードをここに
追加します⟶
        }
};


sc_data_c *
getNew_accessMyGroup()
{
        return (sc_data_c*) new accessMyGroup_user;
}
```

C++ では、C コードで説明したマクロを使用する必要はありません。次のコード行は「text1」の内容を取り出します。

```
char * str = g->text1->getValue();
```

次の行はトグルの状態を確認し、トグルがオンになっている場合には「text1」の内容を設定します。

```
if (g->toggle1->getState())
                g->text1->setValue("The toggle is set");
```

Java によるスマートコードのコールバック

「コード生成」ダイアログで言語として Java を選択した場合には、図 16-3 に示すスマートコードのコールバックが、SCData スマートコードクラスのサブクラスとして生成されます。これは、上述の C++ の場合に似ています。クラスの `doit()` メソッドに、ユーザー自身のコードを追加します。スーパークラスである SCData には、グループへのハンドルを取得するための簡易関数が含まれています。SCData クラスは、生成ディレクトリの下位にある `utils_java` サブディレクトリのファイル `SCData.java` で定義されます。

図 16-3 に示す例の場合には、callouts_java サブディレクトリの accessMyGroup_user.java というファイルに以下のコードが生成されます。

```
package callouts_java;


import groups_java.* ;
import utils_java.* ;


public class accessMyGroup_user extends SCData
{
        public void doit() {
                group0_c g = (group0_c)getGroup();
```

自分のコードをここに追加します ⟶

```


                }


        public static accessMyGroup_user getNew()
        {
                return  new accessMyGroup_user();
        }
}
```

Java では、C コードで説明したマクロを使用する必要はありません。その代わり、C++ の場合と似た操作を行う必要があります。次のコード行は「text1」の内容を取り出します。

```
String str = g.text1.getValue();
```

次の行ではトグルの状態を確認し、トグルがオンになっている場合には「text1」の内容を設定します。

```
if (g.toggle1.getState())
                g.text1.setValue("The toggle is set");
```

作成コールバックによるダイアログの準備

上の例では、「活性化」用に定義されたコールバックを示します。つまり、「取得/設定」コードは、ボタンが押されるまで呼び出されません。ダイアログが最初に表示されたときにダイアログ内のウィジェットを「準備する」必要がある場合には、シェルの「作成」コールバックに対してスマートコードのコールバックを定義してください。このコールバックは、ダイアログの作成時に一度だけ行われます。

ウィジェットの作成コールバックは、すべてのウィジェットの子が作成された後で呼び出されます。スマートコードのコールバックからアクセスするには、グループ内のウィジェットを事前に作成しておく必要があります。シェルの作成コールバックを使用してダイアログを準備すれば、シェルは作成される最後のウィジェットであるため、作成コールバックが起動される前に、必ずすべてのウィジェットが作成されます。シェル上にコールバックを設定したくない場合には、グループ内のすべてのウィジェットを含んでいる任意のコンテナをデザインに追加します (まだの場合)。子ウィジェットは必ず親よりも前に作成されるため、このようにすれば、スマートコードのコールバックが起動される前に、スマートコードのコールバック用に指定したグループ内のウィジェットが確実に作成されます。

getter と setter の使用

ツールキットに依存しないラッパーは、手軽に使用できます。ウィジェットの種類ごとに提供される関数の一覧については、979 ページの付録 C「getter と setter」を参照してください。

HTML ファイル

コールバックを検索しやすくし、スマートコードの要求時に生成されるファイルに慣れていただくため、Sun WorkShop Visual は一連の HTML ファイルも生成します。主要な HTML ファイルは以下のものです。

- index.html

このファイルは、最上位ディレクトリ (つまり、「コード生成」ダイアログで指定したディレクトリ) にあります。

このファイルを Web ブラウザ (または HTML を読み込めるその他のソフト) で開くと、生成されるファイル名が簡単な説明とともに表示されます。また、生成ディレクトリから Sun WorkShop Visual のインストールディレクトリにシンボリックリンクが張られているため、このファイルからスマートコードの一般的な機能について記述する HTML ファイルを参照することもできます。

取得と設定の学習

ここでは、グループと取得と設定用のスマートコードの主要な機能を紹介します。この学習では、グループを設定し、スマートコードを使用してグループメンバーの値を取得および設定します。

完成したアプリケーションは、図 16-4 に示すようになります。

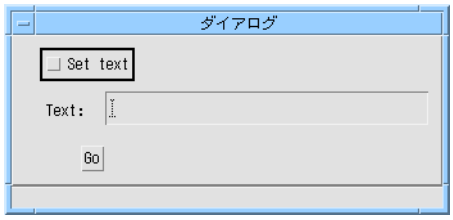


図 16-4 最終のアプリケーション

1. Sun WorkShop Visual を起動します。
2. アプリケーションシェルとフォームを作成します。ボタン、ラベル、テキストフィールド、およびトグルをフォームの中に入れます。

   この最初の階層を図 16-5 に示します。

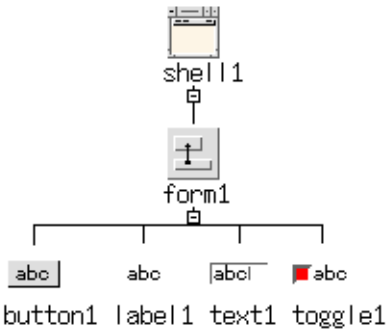


図 16-5 学習用の最初の階層

3. ラベル、テキスト、およびトグルのウィジェットを選択し、ツールバー上の「グループ編集」ボタンを押します。

   この操作は、ウィジェットメニューから「新規グループに追加」を選択するのと同じです。

   グループエディタは、図 16-6 に示すように表示されます。

4. 新しいグループの名前を「MyGroup」に変更するには、これを「名前」というテキストフィールドに入力して Return キーを押します。

   グループの名前を変更する必要はありませんが、特に多数のグループを作成する場合には、名前を変更すると後の操作が簡単になることがあります。

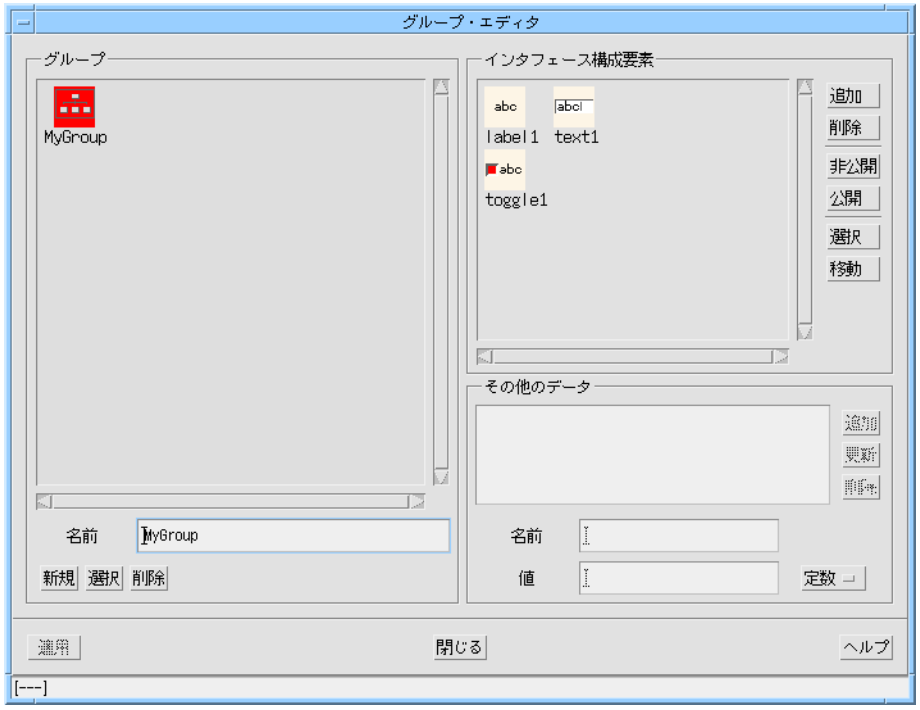


図 16-6 新規グループ用のグループエディタ

5. グループエディタを閉じます。

   後で見るように、「適用」ボタンが有効になるのは、コールバックに追加するグループを選択しているときだけです。

6. 図 16-7 を参考にして、フォーム内のウィジェットとボタンのラベルの配置、ラベル、およびトグルを変更します。

   これは単なる表面的な変更です。ボタンのラベルは「Go」、ラベルは「Text:」、トグルのラベルは「Set text」に設定します。

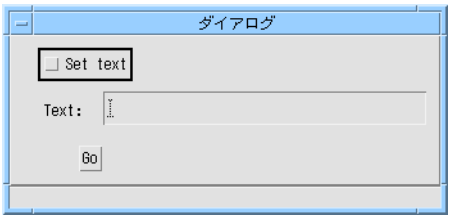


図 16-7 ウィジェットを配置し、ラベルを設定

7. ボタンの「コールバック」ダイアログを表示します。

   これを行うには、ツールバー上の「コールバック」ボタンを押すか、またはウィジェットメニューから「コールバック」を選択します。

8. コールバックのリストで「活性化」が選択されていることを確認します。

   「活性化」を選択するのは、ボタンが押されたときにコールバックが呼び出されるようにするためです。

注 - コールバックのリストでは、活性化コールバックの後に「J」が表示されています。これは Java コードに適用できることを示しています。これらの注釈については、198 ページの「コールバック」を参照してください。

---

9. 「スマートコード」トグルをオンにします。

10. 「スマートコード」オプションメニューから「取得/設定」を選択します。

11. 「グループ」というボタンを押します。

    これによってグループエディタが表示されます。なお、「適用」ボタンは現在有効になっています。

12. MyGroup を選択して「適用」を押します。

    エディタが消えて、グループボタンの横にあるテキストフィールドに名前「MyGroup」が追加されます。名前を直接テキストフィールドに入力することもできます。

13. コールバックに「doGoButton」という名前を付けて「追加」を押します。

    ピリオドを中括弧で囲んだ状態 ({...}) で新しいコールバックがリストに表示され、これはスマートコードのコールバックであることを示します。定義されたコールバックを図 16-8 に示します。

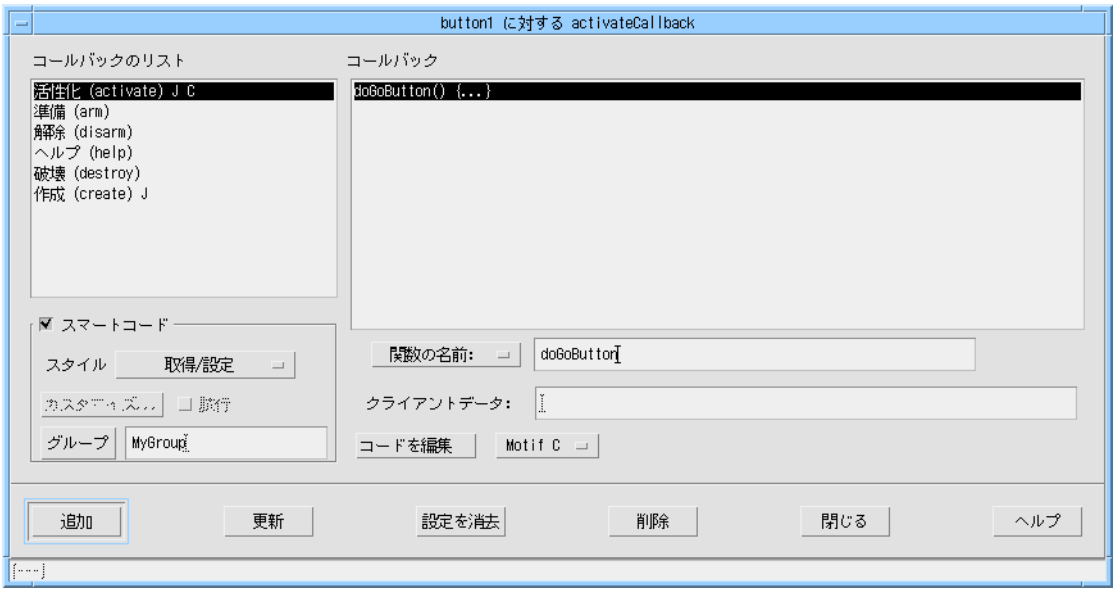


図 16-8 新しいスマートコードのコールバック

14. 「コールバック」ダイアログを閉じます。

    コールバックの定義が完成しました。次は、これに記入する必要があります。

15. デザインを `tutorial.xd` として保存します。

16. 「コード生成」ダイアログを表示し、ソースファイルのターゲットディレクトリを選択します。

    ターゲットディレクトリは、ダイアログの上部に表示されます。

17. 「スタブ」、「コード」、「外部宣言」、「メインプログラム」、および「メークファイル」の生成トグルがオンになっていることを確認します。

    「コード生成」ダイアログの正しい設定を図 16-9 に示します。この学習では、ファイルの名前をデフォルトの「untitled」から「tutorial」に変更しました。

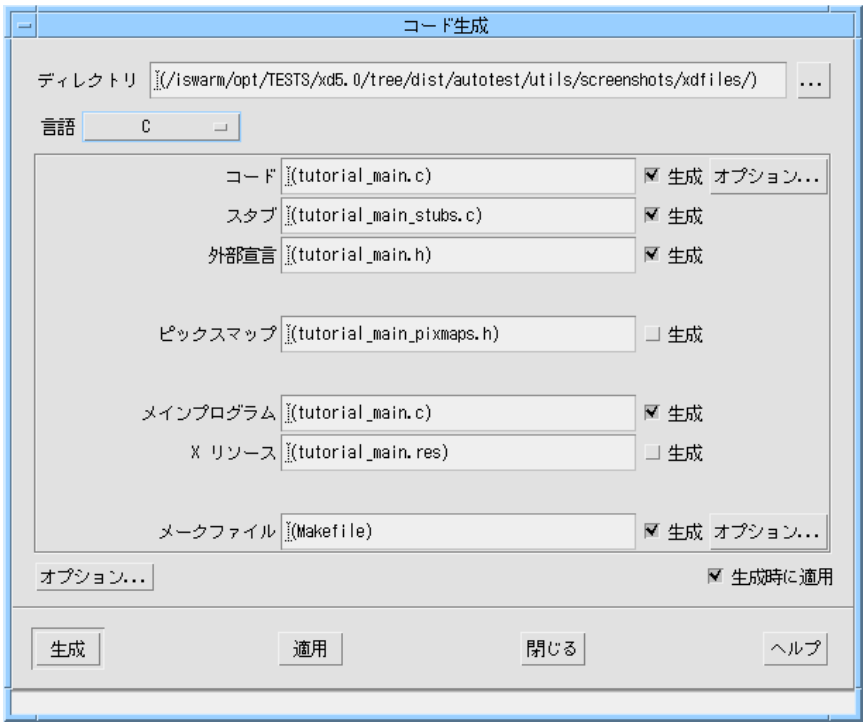


図 16-9 「コード生成」ダイアログ

18. 「コード生成」ダイアログの上の方にある「オプション」ボタンを押して「コード生成オプション」ダイアログを表示します。

19. 「文字列」オプションメニューに「コード」を設定し、このダイアログを閉じます。

    簡単な学習では、別個のリソースファイルを生成するよりも、ラベル文字列をコードファイルに生成する方が便利です。

20. 「生成」ボタンを押して C コードを生成します。

    もちろん、C++ コードも生成できます。後述するリスト例では C を使用しています。

    図 16-10 に示すディレクトリとファイルが生成されます。

参照 - 「取得/設定」スマートコード用に生成されるファイルについては、607 ページの「取得と設定の生成コード」を参照してください。

---

21. デザインを保存します。

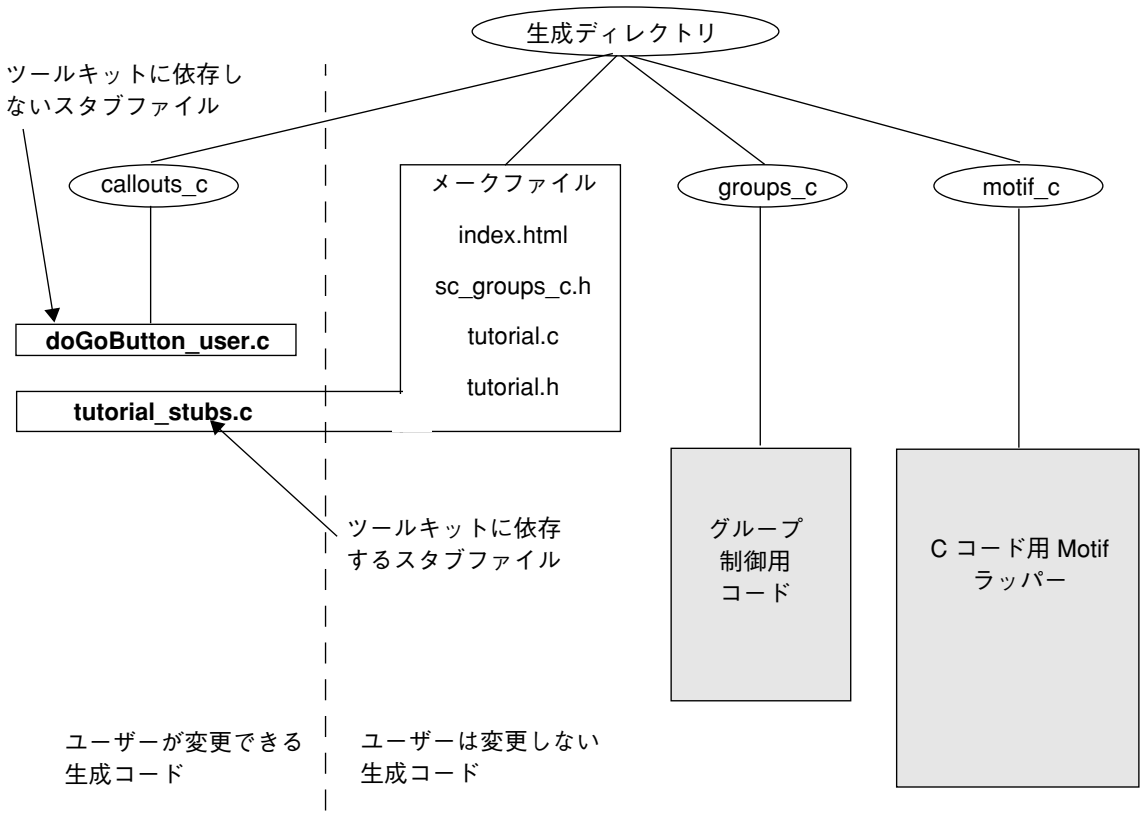


図 16-10 学習用に生成されたファイルとディレクトリ

22. 好みのエディタを使用して、ディレクトリ **callouts_c** 内のファイル **doGoButton_user.c** を編集します。

    これは図 16-10 で強調表示されているファイルです。

23. コールバック自身も「doGoButton_user」で呼び出されます。

    このリストを次に示します。

```
#include <stdio.h>
#include "sc_groups_c.h"

void
doGoButton_user ( d, group, client_data)
        sc_data_t * d;
```

```
        MyGroup_t* group;
        void* client_data;
{


}
```

注 - このルーチンは、tutorial_stubs.c のボタン活性化コールバックから呼び出されます。

---

24. 以下の行を「doGoButton_user」に追加します。

```
if (SC_GET(State, group->toggle1))
                SC_SET(Value, group->text1, "The toggle is set");
else
                SC_SET(Value, group->text1, "");
```

上記の行は、トグルがオンになっているかどうかを確認します。トグルがオンになっている場合には、テキストフィールドに「The toggle is set」と表示されます。オンでない場合には、テキストフィールドはクリアされます。

25. 編集内容を保存します。

26. 環境変数 VISUROOT に Sun WorkShop Visual のインストールディレクトリが設定されており、C コンパイラが PATH に含まれていることを確認します。

コンパイルの前に設定の詳細を確認するには、ユーザーガイドのコードの生成に関する章を参照してください。

27. ファイルが生成されたディレクトリで

make

と入力します。

28. 構築したアプリケーションを実行してみます。

29. トグルをオンにした状態とオフにした状態で、「Go」ボタンを押します。

これで学習が完了しました。この学習で追加した関数は非常に簡単なものですが、ここで確立した枠組みに従えば、自分のアプリケーションをきわめて簡単に拡張することができます。thin クライアント用スマートコードのコールバックを定義することに

より、デザインからクライアントとサーバーの両方を簡単に生成できます。これについては、577 ページの第 17 章「thin クライアント用スマートコード」を参照してください。具体的な方法については、581 ページの「thin クライアント用スマートコードの学習」を参照してください。

第17章

thin クライアント用スマートコード

はじめに

Motif と MFC のアプリケーションは、通常は大規模なアプリケーションです。最新のインターネット技術では、*thin* クライアント方式を推奨しています。この方式は、以下の要素が完全に分割されたアプリケーション構造を採用しています。

- ユーザーインタフェースだけを制御する軽量クライアント
- データをクライアントに返すリモートサーバー

注 - Sun WorkShop Visual によってデフォルトで生成されるサーバーアプリケーションは、Web サーバーを使用してクライアントと交信する CGI プログラムです。

本章では、Sun WorkShop Visual を使用して、大規模なアプリケーションから上述の構造にきわめて簡単に移行する方法について説明します。これを行うには、まずウィジェットを「グループ化する」ことによって移植可能なデータ構造を作成し、コールバックを「機能拡張」することによってリモートに実行したり、リモートサーバーと交信できるようにします。デザインに対しては、この他の変更は必要ありません。

Sun WorkShop Visual を使用すれば、「コードを生成する前に」、クライアントとサーバー間の接続を動的に試すこともできます。thin クライアントやインターネットのコールバックの「試行」トグルをオンにすると、指定した URL に接続することによって、そのコールバックが起動されます。これについては、580 ページの「試行」を参照してください。

グループは、その getter や setter とともに、あらゆるタイプのスマートコードの基本となります。したがって、thin クライアント (またはインターネット) 用スマートコードを使用するには、グループと取得/設定用スマートコードの両方の使い方を理解する必要があります。これらの内容については、以下を参照してください。

1. ウィジェットをグループ化する方法については、549 ページの第 15 章「グループ」を参照してください。
2. 559 ページの第 16 章「取得と設定用のスマートコード」では、デザイン内のウィジェットに対してツールキットに依存しないラッパーを提供する取得/設定用スマートコードについて説明します。

thin クライアント用スマートコードの使用

デザインから thin クライアントおよびサーバーアプリケーションを作成するには、特別なスマートコードのコールバックを設定します。そのためには、以下の操作を行う必要があります。

1. デザイン内のウィジェットの「コールバック」ダイアログを表示します。
2. 「スマートコード」トグルを選択します。
3. 「スマートコード」オプションメニューから「thin クライアント」を選択します。
4. グループを指定します。
5. コールバックに名前をつけて追加します。

グループの説明と作成方法については、549 ページの第 15 章「グループ」を参照してください。

スマートコードにより、コールバックは (言語独立ではなく) ツールキット独立になります。生成したい言語は、これまでどおり選択できます。スマートコードは、指定の言語を使用してグループ内のウィジェットを「包み込む」コード階層です。

スマートコードには 3 つの種類があります。

1. 取得/設定
   ウィジェットのグループに対する一連の「getter」関数と「setter」関数が使用できます。559 ページの第 16 章「取得と設定用のスマートコード」を参照してください。

2. thin クライアント
   getter 関数と setter 関数が使用可能になるだけでなく、Sun WorkShop Visual が、ユーザーインタフェースを含んだ別個の thin クライアントアプリケーションと、ユーザーが定義するコールバックを含んだサーバーアプリケーションを生成可能になります。さらに、この 2 つの間の通信を処理するコードも生成されます。本章では、thin クライアント用スマートコードの使い方を説明します。

3. インターネット
   getter 関数と setter 関数が使用可能になり、Sun WorkShop Visual がリモートサーバーといつでも通信できるクライアントアプリケーションを生成します。サーバーと通信するためのコードも生成されます。インターネット用スマートコードについては、615 ページの第 18 章「インターネット用スマートコード」を参照してください。

グループの指定

「thin クライアント」用と「インターネット」用のスマートコードでは、コールバックに渡すグループの名前が必要になります。リモートサーバーでコールバックが実行されていても、グループによって、ユーザーインタフェースでコールバックにアクセスできるようになります。グループの名前を「グループ」テキストフィールドに直接入力するか、「グループ」ボタンを押してください。これによって、グループエディタが表示されます。必要なグループを選択して「適用」ボタンを押すと、エディタが消えて、グループの名前がテキストフィールドに入力されます。

getter と setter

「thin クライアント」または「インターネット」が選択された状態で、Sun WorkShop Visual によって生成されたすべてのスタブは、指定のグループに対して「getter」と「setter」のルーチンの機能をすべて使用できます。取得/設定用スマートコードについては、559 ページの第 16 章「取得と設定用のスマートコード」を参照してください。サーバー内のコールバックの場合には、グループの構成要素に対するアクセスが若干異なります。このコールバックは、デフォルトの「値」にアクセスする単独の

get ルーチンまたは set ルーチンを呼び出します。この「値」は、ウィジェットの種類によって異なります。たとえば、テキストフィールドには文字列値 (その内容) があり、トグルにはブール値 (その状態) があります。

カスタマイズボタン

Sun WorkShop Visual は、「コールバック」ダイアログのスマートコード「スタイル」オプションメニューから「thin クライアント」や「インターネット」を選択すると、「カスタマイズ」ボタンが有効になります。このボタンを押すと表示される「カスタマイズ」ダイアログにより、サーバーとの間でデータの送受信方法を設定できます。Sun WorkShop Visual は、目的にかなったデフォルトを提供します。しかし、ユーザーは通信に使用するサーバーの URL アドレスを必ず変更する必要があります。

「カスタマイズ」ダイアログは、オプションメニューから「thin クライアント」を選択した場合と「インターネット」を選択した場合とでは、その表示が若干異なります。唯一の相違点は、送信カスタムデータハンドラ・テキストボックスのラベルです。その理由は、Sun WorkShop Visual が、2 種類のアプリケーションに対して異なる HTTP メソッドを使用するためです。「thin クライアント」が選択されると、Sun WorkShop Visual は POST プロトコルを使用します。デフォルトでは、「インターネット」が選択されると、Web ページだけを取り出すものと想定して、Sun WorkShop Visual は GET プロトコルを使用します。これを変更して POST を使用するには、カスタム送信データハンドラを使用します。詳細については 597 ページの「カスタムデータハンドラ」を参照してください。

「カスタマイズ」ダイアログについては、592 ページの「サーバー接続のカスタマイズ」を参照してください。

スタブファイルのリストも含めて、サーバーコールバックの定義時に生成されるコードの詳細については、609 ページの「thin クライアントとインターネットの生成コード」を参照してください。

試行

thin クライアントやインターネットのコールバックが定義された場合には、「試行」トグルが有効になります。このトグルをオンにすると、コードを生成する前に、Sun WorkShop Visual の内部からクライアントとサーバー間の接続をテストすることができます。詳細については、580 ページの「試行」を参照してください。

必要条件

thin クライアント用とインターネット用のスマートコードでは、生成されたクライアントアプリケーションとサーバー間の通信手段として、WWW の規格を使用します。生成およびコンパイルされたアプリケーションを実行し、「試行」機能を使用するためには、以下の条件が必要となります。

1. イントラネットまたはインターネットに接続されたネットワークコンピュータ
2. 動作中の Web サーバー

これらの詳細については、システム管理者に相談するか、または Sun WorkShop Visual のご購入先にご連絡ください。

thin クライアント用スマートコードの学習

この簡単な学習例では、「thin クライアント」スタイルのスマートコードを試すことができます。以下の手順に従えば、図 17-1 に示すユーザーインタフェースを含んだ thin クライアントと、トグルのオン/オフ状態に応じてテキストフィールドの内容を設定するサーバーを簡単に生成できます。このサーバーは、独自のイントラネット上でリモートに実行したり、Web 上の任意の場所で実行できます。

読者はすでに Sun WorkShop Visual の一般的な使い方を熟知しているものと想定して、最初の手順は手短かに説明しています。使い方に不慣れな方は、第 2 章から始まる主要な Sun WorkShop Visual 学習例をお試しください。

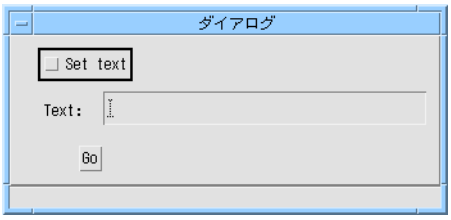


図 17-1 thin クライアントアプリケーションのユーザーインタフェース

1. Sun WorkShop Visual を起動します。

2. アプリケーションシェル、フォーム、ボタン、ラベル、トグル、テキストフィールドという簡単な階層を作成します。

   これを図 17-2 に示します。

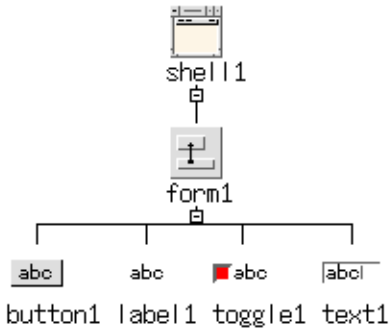


図 17-2 スマートコード学習用の階層

3. ボタンには「Go」、トグルには「Check me」、ラベルには「Text:」というラベルを付けます。

4. シェルに「Tutorial Example」というタイトルを付けます。

   これをシェルのリソースダイアログで行います。

5. フォームの配置エディタを表示し、図 17-3 に示すようにウィジェットを配置します。

   これは単なる表面的な変更です。外見が変化するだけで、アプリケーションの機能には影響ありません。

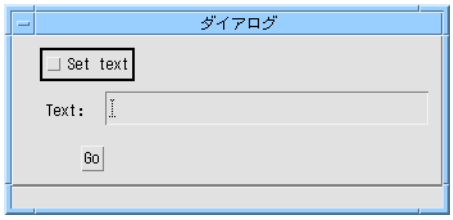


図 17-3 配置された学習用ウィジェット

6. トグルとテキストフィールドを「MyGroup」というグループにします。

   この操作を忘れた場合には、549 ページの第 15 章「グループ」を参照してください。

参照 - グループの詳細については、549 ページの第 15 章「グループ」を参照してください。グループの作成と使用についての簡単な例は、568 ページの「取得と設定の学習」を参照してください。

7. ボタンウィジェットを選択して、「コールバック」ダイアログを表示します。
8. 関数名テキストボックスに「doGoButton」と入力します。

   このコールバックは、実際には、アプリケーションのサーバー側に移動します。
9. 「スマートコード」トグルボタンをオンにします。
10. スマートコードのスタイルを「thin クライアント」に変更します。

    これによって「カスタマイズ」ボタンが有効になります。

参照 - 「thin クライアント」の使い方と意味の詳細については、590 ページの「thin クライアントのサーバーコールバック」を参照してください。

11. 「カスタマイズ」ボタンを押して「カスタマイズ」ダイアログを表示します。

    このダイアログを図 17-4 に示します。

参照 - 「カスタマイズ」ダイアログの各種フィールドについては、592 ページの「サーバー接続のカスタマイズ」を参照してください。

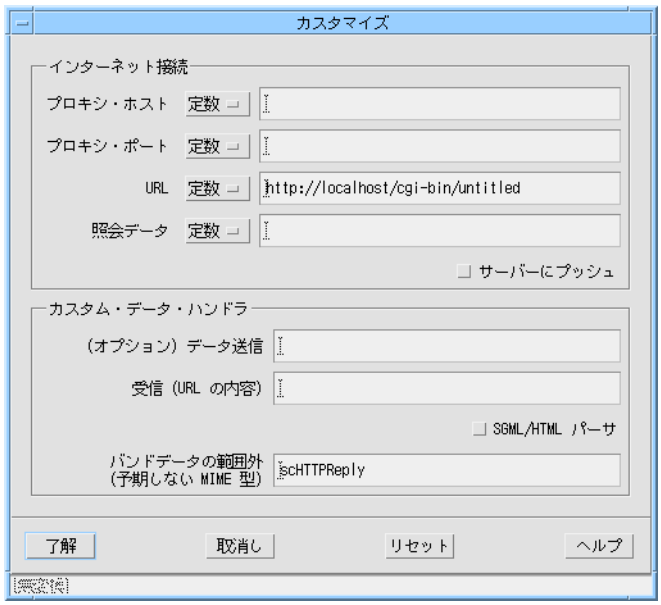


図 17-4 「カスタマイズ」ダイアログ

12. ファイアウォールの保護下にある場合には、「プロキシ・ホスト」フィールドと「プロシキ・ポート」フィールドに入力してください。

    入力内容がわからない場合には、システム管理者に問い合わせるか、Web ブラウザの設定を確認してください。

参照 - プロキシの調べ方については、595 ページの「プロキシの詳細」を参照してください。

---

13. 「URL」フィールドにサーバープログラムの名前と位置が入力されていることを確認します。

    たとえば、

    ```
    http://localhost/cgi-bin/untitled
    ```

    では、イントラネットの「cgi-bin」ディレクトリにある「untitled」というサーバーを表します。この学習に従った場合、サーバーアプリケーションの名前は「tutorial」となるため、ここではこの名前を使用します。「untitled」を「tutorial」に変更してください。「cgi-bin」は標準のディレクトリ名ですが、「localhost」はイントラネットの名前に変更してください。

注 - サーバーをファイルシステム上のどこに置くか決めるには、サーバーの「cgi-bin」領域の物理位置に関して Web サーバーがどのように構成されているかを確認してください。

14. 「カスタマイズ」ダイアログで「了解」を押します。

    この学習ではデフォルトのデータハンドラを使用するため、変更の必要はありません。「了解」を押すとダイアログが閉じます。

15. 「コールバック」ダイアログに戻り、「グループ」ボタンを押し、グループエディタから「MyGroup」を選択します。

    グループの名前は、テキストボックスに直接入力できますが、グループエディタから選択すれば入力ミスを防止できます。

16. 「追加」を押して「doGoButton」関数をコールバックのリストに追加します。

17. 「コールバック」ダイアログを閉じます。

18. デザインを「`tutorial.xd`」として保存します。

    デザインは定期的に保存するようお勧めします。

19. 「生成」メニューから「生成」を選択して、「コード生成」ダイアログを表示します。

20. 「言語」メニューから「C」を選択します。

21. ファイルの生成先ディレクトリを選択します。

22. 「スタブ」、「コード」、「外部宣言」、「メインプログラム」、および「メークファイル」の生成トグルがオンになっていることを確認します。

    デフォルトでは、「スタブ」ファイルは選択されません。したがって、生成する前に「スタブ」がオンになっていることを確認してください。

23. 「コード生成」ダイアログの下部にある「オプション」ボタンを押して、「コードオプション」ダイアログを表示します。

    このダイアログは、「コード」フィールドの横にある「オプション」ボタンを押したときに表示される基本ソースファイルオプションダイアログとは違います。

24. 「コードオプション」ダイアログで、「文字列」オプションメニューを「コード」に変更します。

    デフォルトでは、これには「リソースファイル」が設定されています。この簡単な学習では、文字列 (設定したすべてのラベル) をコード内に生成する方が簡単です。これは最終的なインタフェースの外見にのみ影響を与えます。

25. 「コードオプション」ダイアログで「了解」を押すと、変更内容が保存され、ダイアログが消えます。

26. 「生成」ダイアログに戻って「生成」を押します。

    メインコードファイル、メークファイル、および HTML インデックスが最上位ディレクトリに生成される他に、Sun WorkShop Visual によって以下のサブディレクトリも生成されます。

    - callouts_c
      スマートコードのコールバック用のクライアント側スタブが含まれています。
    - server_c
      スマートコードのコールバックスタブなど、サーバー用のすべてのコードが含まれています。
    - groups_c
      グループのコードが含まれています。
    - motif_c
      グループの構成要素を Motif から独立させる「ラッパー」が含まれています。
    - http_c
      HTTP プロトコルを使用するクライアントとサーバー間の通信用のすべてのコードが含まれています。

    生成されるすべてのファイルを図 17-5 に示します。スタブを含んでいるファイルは強調表示されています。

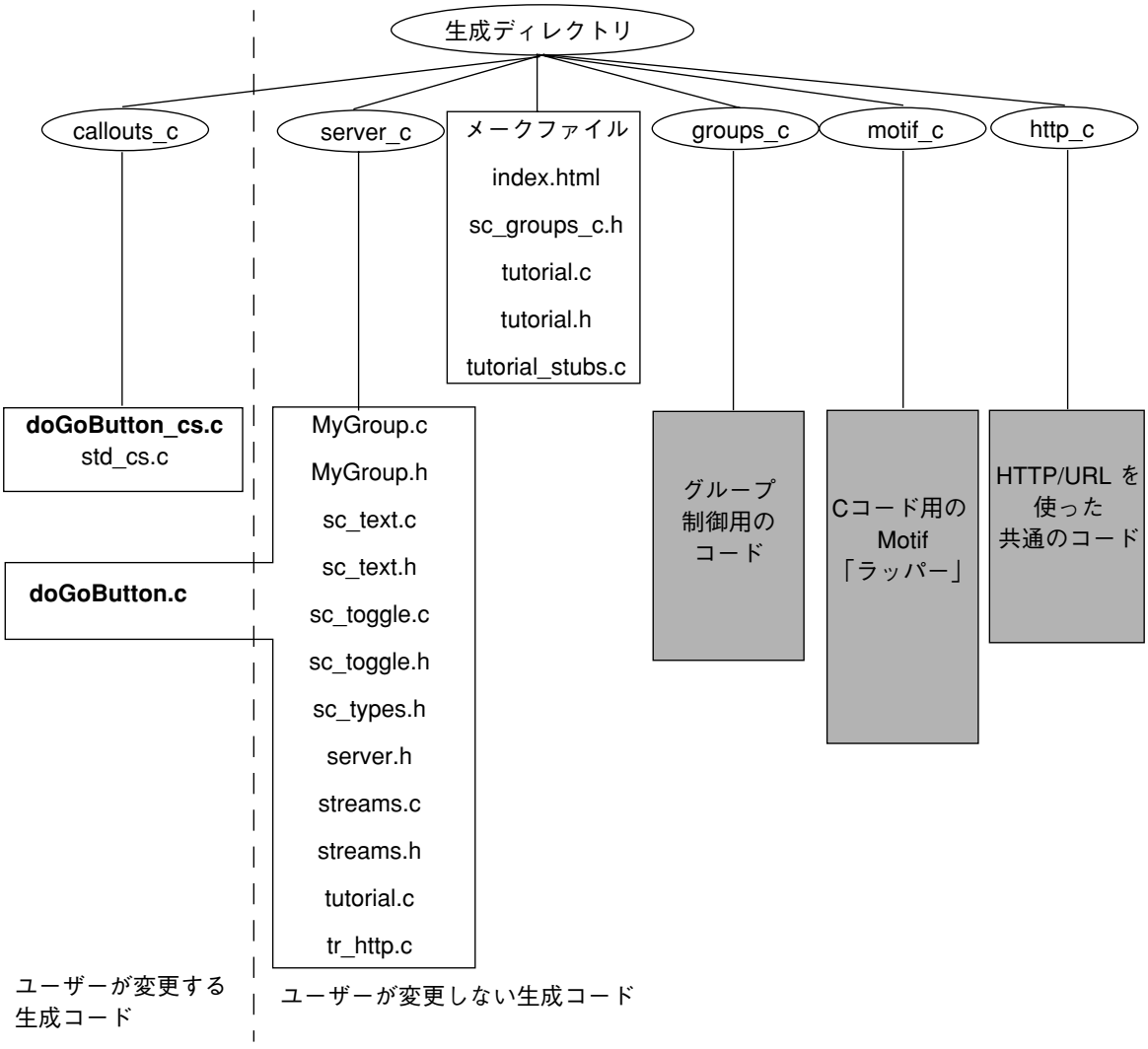


図 17-5 学習用に生成されるファイルとディレクトリ

27. デザインを保存します。

この学習例の残りの部分は、Sun WorkShop Visual の外部で行われます。この時点で、Sun WorkShop Visual を終了してもかまいません。

生成されたファイルのうち、次の 3 つのファイルについて説明します。

- callouts_c/doGoButton_cs.c
  このファイルは、アプリケーションのクライアント側にあります。ここには、サーバーコールバックが呼び出される前と後に直接呼び出される、「事前条件」ルーチンと「事後条件」ルーチンが含まれています。デフォルトでは、これらのルーチンは空です。たとえば、サーバーに送信されるグループ構造に前処理や後処理を行いたい場合には、これらのファイルにルーチンを設定するだけで済みます。
- callouts_c/std_cs.c
  このファイルには、バンドデータの範囲外を処理するための標準エラーハンドラが含まれています。また、サーバーと交信するためのルーチンも含まれています。事前条件ルーチンと事後条件ルーチンを呼び出すのはこのルーチンです。このファイルの内容は常に同じですが、必要ならばハンドシェークの手順をユーザーが確認できるよう、ここに置かれています。通常、このファイルを変更する必要はありません。
- server_c/doGoButton.c
  このファイルには、クライアントから呼び出されるスマートコードのコールバックスタブが含まれています。これはサーバーアプリケーション内に存在します。

参照 - thin クライアント用に Sun WorkShop Visual が生成するコードとインターネット用のスマートコードの詳細については、606 ページの「生成コード」を参照してください。

---

28. 「server_c」ディレクトリ内のファイル doGoButton.c を編集します。

    これには、スマートコードのコールバックスタブが含まれています。サーバーにその機能を実行させるには、ここにコードを追加する必要があります。

29. 以下に示す行を含むよう、メソッド「doGoButton_callback」を変更します。

```
int
doGoButton ( sc_data_t * data)
{
                        MyGroup_t * g = (MyGroup_t*)data->group;

    if (SC_GET(g->toggle1))
                            SC_SET(g->text1, "Server says: toggle is
set");
    else
```

これらの行を追加します

```
                                SC_SET(g->text1, "Server says: toggle is NOT
set");


        return 1;
}
```

30. 環境変数 VISUROOT には Sun WorkShop Visual のインストールディレクトリが設定され、C コンパイラが PATH に含まれていることを確認します。

    コンパイルの前に設定の詳細を確認するには、「メインユーザーガイドのコードの生成」に関する章を参照してください。

31. コマンドプロンプトから

    make

    と入力してクライアントアプリケーションを構築し、

    make server

    と入力してサーバーアプリケーションを構築します。

    「server_c」ディレクトリ全体を他の場所に移動してコンパイルすることもできます。その際はメークファイルもコピーしてください。メークファイルでは、ディレクトリ「server_c」がメークファイルの下のディレクトリにあるものと想定しています。

32. Sun WorkShop Visual 再現機能を使用する際のように、サーバーアプリケーション (「server_c」ディレクトリの「tutorial」という名前) を URL フィールドで指定された位置に移動します。

33. クライアントアプリケーション (生成ディレクトリ内の「tutorial」という名前) を実行します。

    トグルをオンにした状態と、オフにした状態でボタンを押してみてください。テキストボックスにメッセージが表示されます。このメッセージはサーバーによって生成されます。サーバーは、最初にトグルの値を検査しています。これは簡単な例ですが、ここで示す構造は簡単に拡張できます。

一歩進んだ学習

この学習を終了したら、thin クライアント用とインターネット用のスマートコードのさらに高度な機能を試してみることもできます。Sun WorkShop Visual のパッケージには、ユーザー用の学習例を実行する Sun WorkShop Visual 再現スクリプト実行用の

命令を含んだ HTML ファイルが付属しています。学習が行われる様子を見て、その結果を確認してください。この学習例を実行するには、HTML ブラウザで以下のファイルを開きます。

1. `$VISUROOT/lib/locale/<YourLocale>/sc/timex.html`
   このファイルでは、自動リモート更新によるアプリケーションの作成方法を示す「サーバーにプッシュ」学習について説明します。

2. `$VISUROOT/lib/locale/<YourLocale>/sc/parsex.html`
   このファイルでは、Web ページを取り出して解析するアプリケーションの作成方法について説明します。

注 - VISUROOT は Sun WorkShop Visual のインストールディレクトリであり、<YourLocale> は使用しているロケールの名前です。

---

thin クライアントのサーバーコールバック

「thin クライアント」用スマートコードのコールバックを含んでいるデザインからコードを生成すると、Sun WorkShop Visual は、その生成コードを、すべてのユーザーインタフェースコードを含む thin クライアントアプリケーションと、スマートコードのコールバックを含むサーバーに分割します。このサーバーは *CGI* スクリプトであるため、通信の手段として WWW の規格を使用します。HTTP を使用してクライアントとサーバー間で交信するコードも生成されます。

「thin クライアント」コールバックが定義された場合に Sun WorkShop Visual が生成するアプリケーションの構造図を図 17-6 に示します。サーバー内のスマートコードのコールバックに加えて、アプリケーションのクライアント側にはスタブも生成されるため、データがサーバーに送信される直前と直後に必要な操作を実行することができます。

サーバー内にあるスマートコードのコールバックには、グループへのハンドルがあります。これは、コールバックの定義時に指定されたグループです。グループによって、クライアントとサーバー間でデータを受け渡す手段が設けられ、サーバーにインタフェース構成要素へのアクセス権が与えられます。

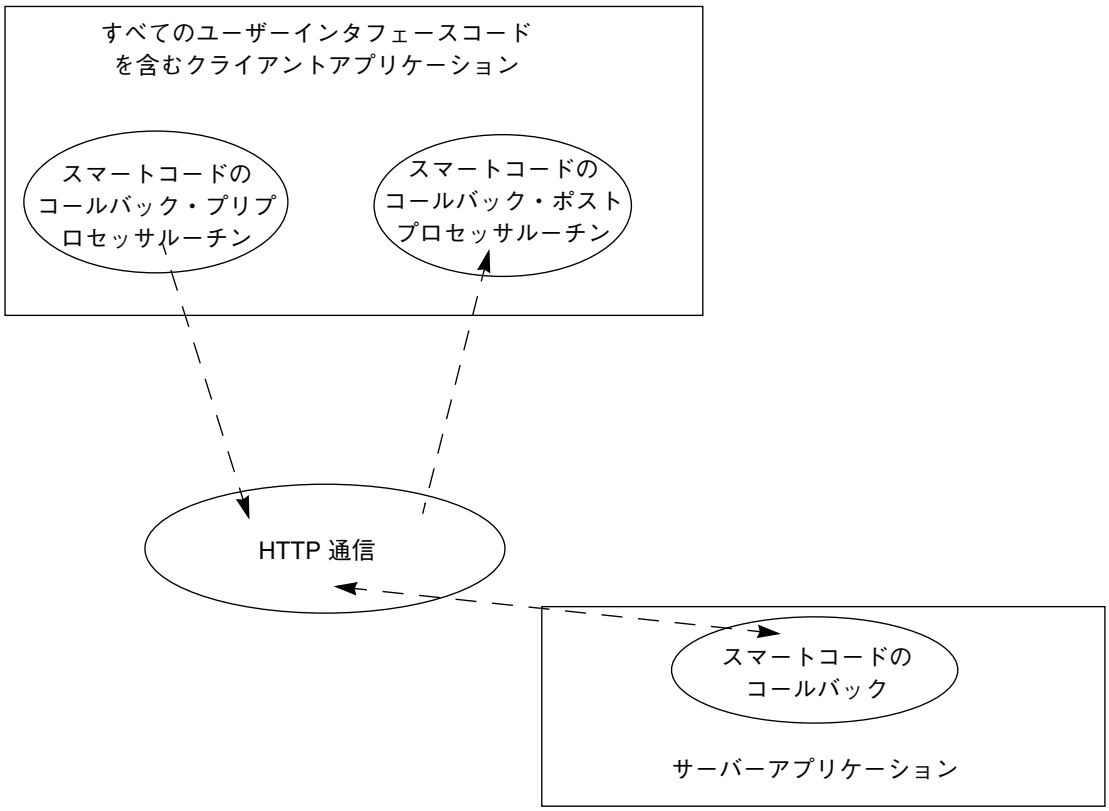


図 17-6 thin クライアントのコールバックアプリケーションの構造

サーバーアプリケーション

thin クライアント用スマートコードが要求されたときに Sun WorkShop Visual が生成するサーバーアプリケーションは、*CGI* スクリプトです。つまり、これはどんな言語 (またはシェルスクリプト) でも記述でき、HTTP 規格によってアクセスされるプログラムです。サーバーアプリケーションの URL は、クライアントアプリケーションから Web サーバーに渡されて、Web サーバーがサーバーアプリケーションを起動します。サーバーアプリケーションからの出力は、クライアントに返されます。このサーバーアプリケーションは、「コード生成」ダイアログで指定した言語で生成されます。

動作中の Web サーバーが存在しない場合には、クライアントとサーバーのアプリケーションは交信できません。インターネット用スマートコードでも、インターネット上の Web ページにアクセスするには、動作中の Web サーバーアプリケーションが存在することを想定しています。Sun の提供する Web サーバー製品については、Sun WorkShop のご購入先にお問い合わせください。Sun の Web ページ (http://www.sun.co.jp) もご覧ください。インターネット上には、無料で入手できる Web サーバーアプリケーションが多数存在します。Web の検索機能を使用すれば、簡単に Web サーバーアプリケーションを見つけることができるでしょう。

サーバー接続のカスタマイズ

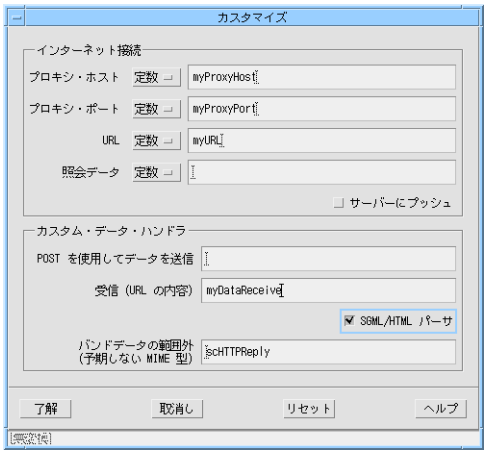
「コールバック」ダイアログで「thin クライアント」用または「インターネット」用スマートコードスタイルの「カスタマイズ」ボタンを押すと、図 17-7 に示す「カスタマイズ」ダイアログが表示されます。

このダイアログは、オプションメニューから選択したものが「thin クライアント」であるか、「インターネット」であるかによって、その表示が若干異なります。図 17-7 には両方の場合を示します。

このダイアログを使用すれば、サーバーとのデータの送受信方法を指定できます。このダイアログには 2 つの主要な領域があります。

1. 接続
   プロキシや URL など、ネットワーク接続の詳細。
2. カスタムデータハンドラ
   ネットワークを介して交信するために Sun WorkShop Visual によって提供されるルーチンの代わりに使用されるルーチンの名前。

これらを以下の節で説明します。



「インターネット」を選択して表示

「thin クライアント」を選択して表示

図 17-7 「カスタマイズ」ダイアログ

接続

Sun WorkShop Visual は、この節で説明する情報を使用して、ネットワークを介したリンクを確立するためのコードを生成します。このダイアログには以下のフィールドがあります。

1. **プロキシ・ホスト**
   これは、ファイアウォールの保護下にある場合にだけ必要となります。これはコンピュータとインターネット間のコンジットの名前です。

2. **プロキシ・ポート**
   これは、ファイアウォールの保護下にある場合にだけ必要となります。これはプロキシ・ホスト上で使用するポート番号です。

注 - プロキシに設定する値がわからない場合には、595 ページの「プロキシの詳細」を参照してください。

---

3. URL
   このフィールドだけは必ず入力しなければなりません。これは、HTML 文書や CGI スクリプトの場所を記述する文字列です。この「文書」が何であるかは、(上述のように) 選択した通信の「スタイル」によって決まります。

- thin クライアント用スマートコードを選択した場合には、おそらく URL は、サーバー側のアプリケーションである CGI プログラムを指します。
- インターネット用スマートコードを選択しても、受信ハンドラを指定しなかった場合には、URL は完全に別個のリモートサーバーを指します。そのサーバーからは受信通知だけが送られます。
- インターネット用スマートコードを選択し、受信ハンドラを指定した場合には、おそらく URL は完全に別個のリモートサーバー (またはそのサーバー上のファイル) を指します。そのサーバーからは特定フォーマットのデータが送られます。

URL は、確立された WWW プロトコルの一部です。

4. 照会データ
   これは照会文字列です。URL に疑問符 (?) が現われた場合、その残りの部分は、検索エンジンが必要とするような、文書やその内容を照合するためのアルゴリズムとみなされます。この文字列の解釈はサーバーに依存します。Sun WorkShop Visual は、疑問符文字を追加してから、その照会文字列を URL に追加します。

5. サーバープッシュ
   これはトグルであり、これを選択すると、サーバーから非同期入力を受け付ける用意ができたことを Sun WorkShop Visual に通知することになります。つまり、データは (要求の後ではなく) 用意ができた時点でクライアントアプリケーションに送られることになります。

URL ライブラリ

Sun WorkShop Visual に付属され、使用される URL ライブラリは、簡単に Java に移行移行できるように、また、C、C++、Java のいずれを使用しても共通のインタフェースが保たれるように、Java の URL クラスと URLConnection クラスを基にして作成されました。クライアントとサーバー間の接続を完全に制御したい場合に、ユーザーはこのライブラリを使用できます。URL ライブラリのプログラミングインタフェースは、生成された `index.html` ファイル中のシンボリックリンクから到達されるファイルに記述されます。このファイルは、612 ページの「HTML ファイル」で記述するように、生成ディレクトリの中にあります。

まだコードを生成していない場合には、HTML ブラウザで次のファイルを開きます。

```
$VISUROOT/lib/locale/<your_locale>/sc/URL.html
```

ここで、VISUROOT は Sun WorkShop Visual のインストールディレクトリであり、<your_locale> は使用しているロケールの名前です。ロケールがわからない場合には、端末ウィンドウに **locale** と入力してみてください。これによってロケール情報が出力されます。この出力の中の LANG に割り当てられた文字列を使用します。ロケールの例を次に示します。

- C (英語用)
- ja (日本語用)

プロキシの詳細

ネットワークプロキシは、ファイアウォールを通じてコンピュータ (またはイントラネット) をインターネットに接続するコンジットです。ファイアウォールは、外部からコンピュータへの不正なアクセスを防止する「セキュリティスクリーン」です。ファイアウォールの保護下からインターネットに接続するには、プロキシ・ホストとプロキシ・ポートを指定する必要があります。これらについてまだご存知でない場合には、以下のどちらかを行なってください。

1. システム管理者に問い合わせる。ネットワーク、電子メール、およびインターネットの接続を行なった担当者がいるはずです。
2. Web ブラウザの設定を調べる。インターネットに接続するには、プロキシを使用する必要があります。この詳細については次の段落で説明します。

Web ブラウザ内のどこか (「オプション」、「設定の変更」または「構成」の下など) に、ネットワーク設定を指定できるダイアログがあります。このダイアログを使用すれば、プロキシを使用する必要性の有無、および使用する場合にはプロキシが何であるかをブラウザに通知できます。ここには複数のオプションがあります。

1. プロキシを使用しない。この場合には、Sun WorkShop Visual の「カスタマイズ」ダイアログにプロキシを指定する必要はありません。
2. プロキシが、ダイアログまたはサブダイアログに手作業で指定されている。この場合には、指定されているプロキシを使用します。プロキシを選択できる場合には、HTTP プロキシを使用します。
3. 構成ファイルの URL が、プロキシの自動設定用に指定されている。この場合には、構成ファイルで指定されたプロキシを調べます。

定数、変数、または関数

「プロキシ・ホスト」、「プロキシ・ポート」、「URL」、「照会データ」には、それぞれ対応するオプションメニューがあり、入力する値が「定数」、「変数」、「関数」のいずれであるかを定義できます。その名前からわかるように、定数は入力された値を保持するだけです。「変数」を選択すると、Sun WorkShop Visual は指定した名前を持つ外部変数を宣言します。変数は、自分のコードで定義してください。

「関数」を選択すると、Sun WorkShop Visual は、ファイル名として関数の名前を使用して、「callouts」サブディレクトリ内にファイルを生成します。このファイルには、「get」と「set」という 2 つの関数が含まれています。これらのルーチンにはコメント付きのコード挿入用の領域があります。ここにはプロキシ・ホストを取得および設定するためのコードを挿入してください。

たとえば、「MyProxyHost」という関数として定義されたプロキシ・ホストを図 17-8 に示します。C コードを想定して、Sun WorkShop Visual は MyProxyHost.c というファイルの中に以下の 2 つのルーチンを生成します。

```
static char * MyProxyHost_value = (char*)0;

char *
get_MyProxyHost ( AnyGroup_t* ingroup)
{
                group0_t * group = (group0_t*)ingroup;
                if (!MyProxyHost_value) {
                    /* do something */
                    (void) fprintf( stderr, "Warning:
getMyProxyHost()
                                                returns NULL\n");
                }
                return MyProxyHost_value;
}


void
set_MyProxyHost (AnyGroup_t* ingroup, char* value)
{
                group0_t * group = (group0_t*)ingroup;
```

```
                /* set MyProxyHost_value here */
}
```

コメントは、プロキシ・ホストを取得および設定するためのコードを追加すべき場所を示します。たとえば、ユーザーインタフェース内にフォームを定義すれば、ユーザーがテキストフィールドにプロキシ・ホストを入力できます。次に、テキストフィールドの内容を取得して、上記の「MyProxyHost_value」に割り当てることができます。

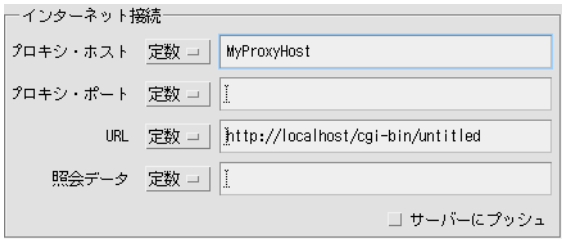


図 17-8 関数としての「MyProxyHost」

ネットワークとプロトコルについては、参考文献一覧に書籍を紹介しています。

カスタムデータハンドラ

「カスタマイズ」ダイアログの「カスタム・データ・ハンドラ」というセクションには、以下のデフォルトルーチンを無効にするオプションがあります。

1. 送信
   これはデータをサーバーに送信するルーチンです。
2. 受信
   これはサーバーからデータを受信するルーチンです。
3. バンドデータの範囲外
   これは、サーバーから返される予期しないタイプのデータを処理するルーチンです。

送信ハンドラや受信ハンドラのテキストボックスを空のままで残すと、Sun WorkShop Visual はデフォルトルーチンを生成します。バンドデータの範囲外の場合には、デフォルトルーチンの名前がテキストボックスに表示されます。これらのデフォルトが提供する基本的な機能を使用すれば、独自のハンドラをすぐに用意しなく

ても、実働可能な「クライアント - サーバー」アプリケーションを作成できます。以下の項では、独自のハンドラの記述に役立つ情報に加えて、デフォルトルーチンの内容について簡単に説明します。

送信ハンドラ

デフォルトでは、Sun WorkShop Visual は、サーバーにデータを送信するルーチンを生成します。このルーチンは以下の動作を行います。

- サーバーへの出力ストリームを開く
- MIME ヘッダーを出力する
- グループの各構成要素を連続した HTTP オブジェクトとして出力する
- MIME エンドを出力する

「カスタマイズ」ダイアログで送信ハンドラを指定した場合には、上述のルーチンは呼び出されません。その代わりに、ユーザーのルーチンが呼び出されます。ユーザーのルーチンは、整数を返すものとして定義されます。また、次の順序で 2 つの引数を受け取ります。

- 出力ストリームへのポインタ
- グループへのポインタ

送信ルーチンを無効にした場合でも、通信プロトコルには影響はありません。置き換えられたデータを送信するだけです。無効にした場合でも、デフォルトの送信ルーチンは生成されますが、呼び出されません。このルーチン (`httpSendOutputStream` という名前) による MIME ヘッダーや MIME エンドの送信方法、およびグループ構成要素の連続化方法を確認したい場合には、次のファイルを調べてください。

```
<YourGenerateDirectory>/http_c/tr_http.c
```

`<YourGenerateDirectory>` は、コードの生成先のディレクトリです。

ユーザーの送信ハンドラは、`Callouts` ディレクトリに置かれます。生成されるコードとその場所の詳細については、606 ページの「生成コード」を参照してください。

参照 - 601 ページの「試行」で説明した取得/設定の短縮形表記を使用してデザインからコードを生成した場合には、例で使用した送信ハンドラが生成されます。これを出発点として使用できます。

受信ハンドラ

「カスタマイズ」ダイアログで受信データハンドラを指定した場合には、Sun WorkShop Visual がデフォルトで生成するすべての通信プロトコルに加えて、このルーチンが呼び出されます。

C と C++ のコードでは、ユーザーのルーチンは、整数を返し 2 つの引数を受け取るものとして定義されます。

1. Sun WorkShop Visual によって定義される構造体へのポインタは、C コードの場合には `sc_stdcs_t` と呼ばれ、C++ の場合には `sc_stdcs_c` と呼ばれます。この構造体によって、ユーザーのルーチンがグループにアクセスできるようになります。内部使用専用の非公開フィールドもあります。これは、コードを生成するディレクトリのファイル `sc_groups_c.h` で定義されます。

2. Sun WorkShop Visual によって定義される構造体へのポインタは、C では sc_idata_t と呼ばれ、C++ では sc_idata_c と呼ばれます。この構造体によって、ユーザーのルーチンが入力ストリーム、その長さ、および MIME タイプにアクセスできるようになります。これは、`http` サブディレクトリ内のファイル `http_transport.h` で定義されています。

Java の場合には、指定する受信ハンドラは、Sun WorkShop Visual が定義したクラスである SCInputDataHandler のサブクラスにされます。独自のコードは doit() メソッドに追加してください。`utils_java` サブディレクトリの `SCIData.java` で定義されるように、このメソッドは「void」であり、SCIData クラスのインスタンスである 1 つの引数を受け取ります。

デフォルトで提供される受信ハンドラは、返されたデータを無視します。しかしほとんどの場合、返されたデータの解析は必要になります。Sun WorkShop Visual には完全な HTML パーサーが提供されています。このパーサーはデータ駆動型であるため、使いやすくなっています。つまり、ウィジェットが調べたいコールバックにインタレスト (関心) を登録するのと同じように、関心のある HTML タグをパーサーに通知することができます。この詳細については、624 ページの「HTML データから情報を抽出」を参照してください。

ユーザーの受信ハンドラは、`Callouts` ディレクトリに置かれます。生成されるコードとその場所の詳細については、606 ページの「生成コード」を参照してください。

参照 - 601 ページの「試行」で説明した取得/設定の短縮形の表記を使用して、デザインからコードを生成した場合には、例で使用した受信ハンドラが生成されます。これを出発点として使用できます。

バンドデータの範囲外ハンドラ

Sun WorkShop Visual は、MIME 規格に準拠した着信データを想定しています。予期しないデータが受信された場合には、Sun WorkShop Visual によって生成された、アプリケーションのクライアント側のルーチンが呼び出されます。このデフォルトルーチンは scHTTPReply と呼ばれ、着信データを標準出力にそのまま出力します。thin クライアント用やインターネット用のスマートコードのコールバックを定義した場合には、このデフォルトルーチンが (Callouts サブディレクトリに) 生成されます。これは、「コード生成」ダイアログで要求した言語で生成されます。次に示すのは C コードのルーチンです。

```
int
scHTTPReply ( csdata, data)
        sc_stdcs_t* csdata;
        sc_idata* data;
{
                extern DataInputStream * newDataInputStream();

                InputStream* i = (InputStream*)(*data-
>getInputStream)( data);
                DataInputStream *  d = newDataInputStream( i);
                char * s;

                printf("data from server:\n");
                if (!d)  {
                    printf("no data\n");
                    return 0;
                }
                while ((s = (*d->readLine)( d)) != (char*)0) {
```

```
        printf("%s\n", s);
    }
    (*d->delete)( d);

    printf("end data\n");
}
```

ユーザー独自のルーチンを「カスタマイズ」ダイアログに「バンドデータの範囲外」ハンドラとして指定した場合には、そのルーチンはデフォルトと同じように定義されます。そのルーチンが受け取る引数は、上述の受信ハンドラの場合とまったく同じです。

Java コードの場合、「バンドデータの範囲外」ハンドラのフォーマットは、受信ハンドラの場合とまったく同じです。これは、Sun WorkShop Visual が定義したクラスである SCInputDataHandler のサブクラスです。独自のコードは doit() メソッドに追加してください。

ユーザーの「バンドデータの範囲外」ハンドラは、`Callouts` ディレクトリに置かれます。生成されるコードとその場所の詳細については、606 ページの「生成コード」を参照してください。

試行

「試行」トグルにより、Sun WorkShop Visual のダイナミックディスプレイを、すべての機能を備えた thin クライアントインタフェースに変えることができます。このトグルは、thin クライアントとインターネットのどちらのコールバックにも設定できます。

「試行」トグルを有効にするには、コールバックの定義と追加を行う必要があります。thin クライアント用スマートコードの場合には、コールバックはサーバー内のルーチンの名前です。インターネット用スマートコードの場合には、これはクライアントアプリケーション内のコールバックの名前です。

Sun WorkShop Visual が提供する短縮形の getter や setter により、「カスタマイズ」ダイアログで動的なカスタマイズを行うことができます。この短縮形では、スプレッドシート表記規則 (「@」記号) を使用して「…の内容」を表します。たとえば、

```
@text1
```

では、「指定したグループでの text1 というウィジェットの内容」を意味します。指定したグループとは、「コールバック」ダイアログで指定したグループです。「接続」領域のフィールド (URL とプロキシ) でこの表記法を使用すると、コントロールの内容が取り出されます。これを「受信ハンドラ」フィールドで使用した場合には、text1 の内容にサーバーから受信したデータを設定するものとみなされます。

「試行」の学習

「試行」ボタンの使用方法を次の例に示します。ここでは、WWW を使用してリモート Web サイトにアクセスし、Web ページを読み込みます。この学習をすべて実行するには、Web サーバーを実行している必要があります。

1. 図 17-9 に示す階層を作成します。これは、シェル -> フォーム -> {ボタン、テキストフィールド、テキスト} で構成されています。

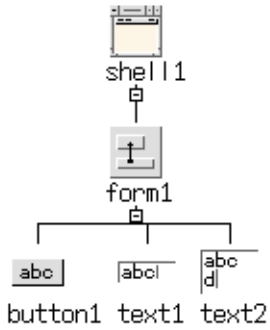


図 17-9 「試行」の例の階層

2. テキストフィールドとテキストウィジェットを含む、Group0 というグループを作成し、テキストフィールドを非公開にします。

   これを図 17-10 に示します。

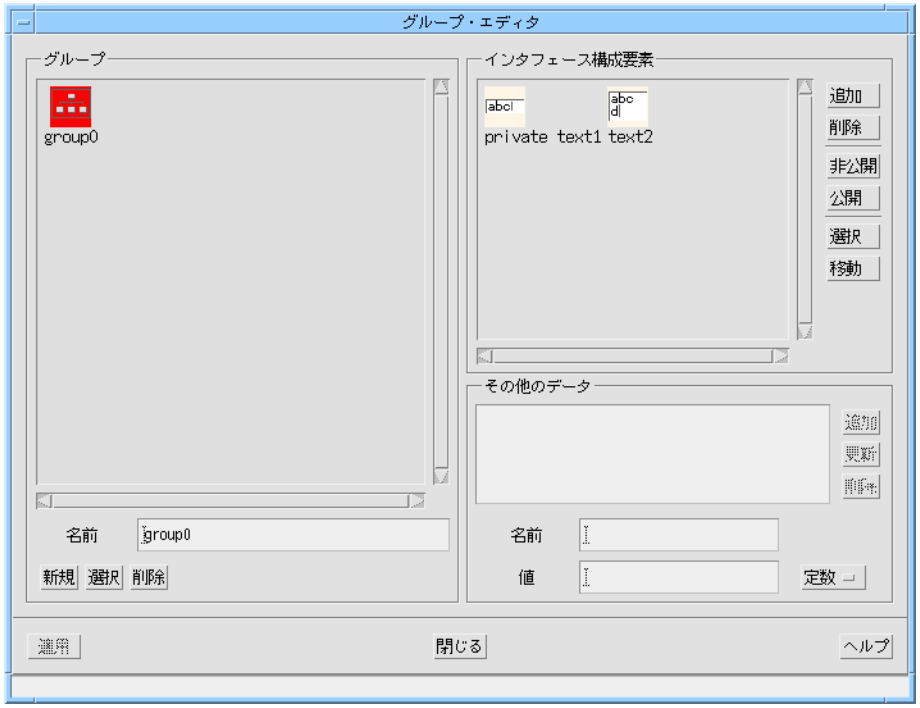


図 17-10 「試行」の例のグループ

3. button1 の「コールバック」ダイアログを表示します。
4. 図 17-11 に示すように、「doit」というインターネット用スマートコードのコールバックを設定します。

   このコールバックは、最初にカスタマイズする必要があるため、まだ追加しないでください。

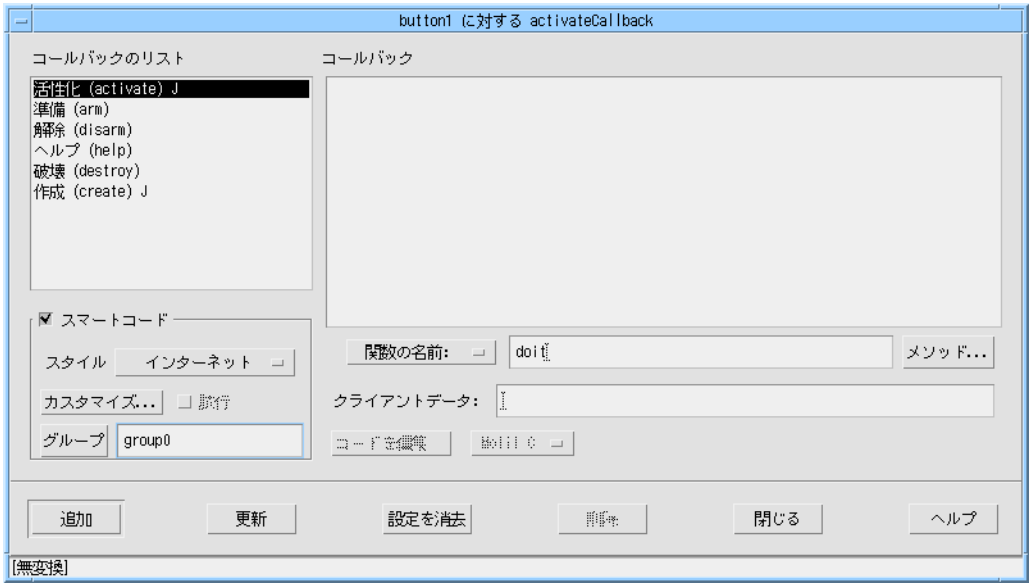


図 17-11 「試行」例のサーバーコールバック

5. 「カスタマイズ」ボタンを押します。

6. 「カスタマイズ」ダイアログでは、図 17-12 に示すように、ウィジェット text1 の内容に対する getter 短縮表記を「URL」フィールドに書き込みます。

   この短縮表記は「`@text1`」となります。

7. 図 17-12 に示すように、ウィジェット text2 に対する短縮表記を「受信ハンドラ」フィールドに書き込みます。

   この短縮表記は「`@text2`」となります。

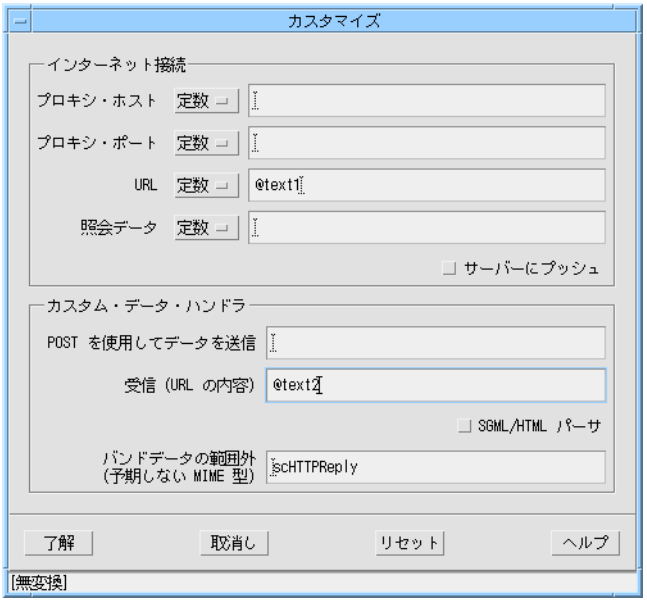


図 17-12 「試行」の例の「カスタマイズ」ダイアログ

参照 - ファイアウォールの保護下にある場合には、プロキシフィールドも設定する必要があります。詳細については 595 ページの「プロキシの詳細」を参照してください。

---

8. 「了解」を押して、設定内容を保存し「カスタマイズ」ダイアログを閉じます。
9. 「コールバック」ダイアログに戻って、「試行」トグルをオンにします。
10. 新しいコールバックを追加して「コールバック」ダイアログを閉じます。
11. ダイナミックディスプレイの「text1」というテキストフィールドで、次の URL を入力します。

    `http://www.sun.co.jp/workshop`

12. ダイナミックディスプレイで button1 を押します。

    テキストフィールドに入力された Web サイトのフロントページが text2 に表示されます。

13. text1 で他の URL を試し、もう一度 button1 を押します。

    有効な URL を入力していれば、text2 は URL の参照先となる HTML ページを表示します。たとえば、(存在する場合は) イントラネットの URL を入力してみてください。

    この例では、Sun WorkShop Visual の内部から実際のサーバーに動的な接続が行われています。

生成コード

ツールキットに依存しないラッパーとグループオブジェクトの定義を含むコードファイルは、「コールバック」ダイアログからスマートコードが選択された場合に生成されます。「thin クライアント」と「インターネット」の場合には、「取得/設定」の場合よりも多くのファイルが生成されます。その理由は、グループの getter と setter に加えて、サーバーとの通信用のコードもすべて生成されるためです。さらに、「thin クライアント」の場合には、サーバー側のアプリケーションも生成されます。

これらのファイルの一部は、特別な名前のディレクトリに置かれます。それらのディレクトリは、Sun WorkShop Visual によって生成され、「コード生成」ダイアログで選択したディレクトリの下位に置かれます。他のファイルは、選択されたディレクトリに置かれます。thin クライアント用スマートコードの学習のために生成されたファイルを、587 ページの図 17-5 に示します。これによって、生成されたファイルとサブディレクトリの仕組みがわかります。

最も重要なことは、編集が必要になる可能性のあるコールバックやその他のルーチン (カスタムデータハンドラ、前処理および後処理ルーチン、その他のデータ関数など) を簡単に見つけ出せるように、まとめておくことです。他のコードファイルを変更する必要はありません。

生成される内容は、コールバック用に「取得/設定」用スマートコードだけを要求したのか、それとも「thin クライアント」や「インターネット」を要求したのかによって異なります。それぞれの場合に生成されるコードを以下に説明します。

グループ

グループを使用し、スマートコードを使用していない場合には、基本ソースファイルにウィジェットの配列が定義されます。

しかし、スマートコードがない場合には、ユーザーズガイドの第 7 章「コードの生成」で記述したファイルだけが生成され、新しいディレクトリは作成されません。

取得と設定の生成コード

コードファイルを生成する際に、Sun WorkShop Visual は、選択された言語に適した拡張子を使用します。

1. **.c** は C コード用
2. **.cpp** は C++ コード用
3. **.java** は Java コード用

以下の例では、単に説明のために「.c」を使用します。

1 つ以上のコールバックに対して「取得/設定」スタイルのスマートコードを選択した場合には、以下のファイルは、「コールバック」ダイアログの最初の項目で指定されたディレクトリに生成されます。

- Makefile
  Sun WorkShop Visual が生成した標準のメークファイルであり、新しいサブディレクトリを使用して構築を行うように記述されています。
- sc_groups_c.h
  このヘッダーファイルには、関連するサブディレクトリの適切なヘッダーファイルが含まれています。また、sc_data_t データ構造体も定義します。この構造体を指すポインタは、スマートコードのコールバックとデータハンドラに渡されます。
- untitled.c
  メインソースコードファイルです。
- untitled.h
  メインヘッダーファイルです。
- untitled_stubs.c
  ここにコールバックスタブが生成されます。
- index.html
  生成されたすべてのファイルを表示し、簡単に説明する HTML ファイルです。ファイルとサブディレクトリの内容を調べるには、Web ブラウザやその他の HTML ブラウザを使って参照してください。

さらに、以下のサブディレクトリが作成されます。

- callouts_c
  関数として定義されているグループ用に定義された「その他のデータ」ごとのファイルが含まれています。
- groups_c
  定義したグループごとのソースファイル、ヘッダーファイル、および HTML ファイルが含まれています。より一般的なヘッダーファイルも含まれています。
- motif_c
  グループ内の構成要素ごとの Motif「ラッパー」が含まれています。このディレクトリには、グループごとの HTML ファイルも含まれています。

注 - 各ディレクトリ名の最後にある「c」は、コード生成用の言語として C が選択されたことを示します。他の言語が選択された場合には、この表記は言語に応じて変化します。

取得と設定用ユーザーコードの追加先

「取得と設定」用スマートコードでは、以下のスタブファイルが生成されます。

- <プログラム名>_stubs.cpp (<プログラム名>_stubs.c)
  ユーザーが定義したコールバックを含んでいる従来のスタブファイルです。このファイル内のスタブは、ツールキットと強い関連性を持っており、グループやスマートコードデータにはアクセスできません。ここには、ツールキット固有のコードだけを追加してください。このファイルは、「コード生成」ダイアログで指定されたディレクトリに生成されます。
- <コールバック名>_user.cpp (<コールバック名>_user.c)
  `Callouts` サブディレクトリに置かれます。取得/設定用スマートコードを使用する場合には、これには `<`コールバック名`>_user` というメインコールバックスタブも含まれています。このコールバックには、ツールキット固有のコードはありません。
- 「その他のデータ」関数のファイル
  グループエディタの「その他のデータ」領域を表します。その他のデータを追加してそれを「関数」と定義した場合には、ファイル名としてその他のデータの名前を使用してファイルが生成されます。このようなファイルには、それぞれ `get_<`その他のデータ`>` および `set_<`その他のデータ`>` という 2 つのルーチンが含まれています。ここで、`<`その他のデータ`>` は、グループエディタでユーザーが指定した名

前です。C++ と Java の場合には、これらのルーチンは「その他のデータ」関数の名前を使用して定義されたクラスのメソッドです。これらのルーチンの使い方については、557 ページの「その他のデータ ― 関数」を参照してください。

「取得/設定」用スマートコードの学習用に生成されたファイルの図式表示を、574 ページの図 16-10 に示します。

thin クライアントとインターネットの生成コード

スマートコードのコールバックを含んでいるデザインからコードを生成すると、607 ページの「取得と設定の生成コード」で説明したファイルとディレクトリがすべて生成されます。さらに、Sun WorkShop Visual は以下のディレクトリを作成します。

- http_c
  HTTP プロトコルを使用して URL を解析し、データを送受信するためのソースコードが含まれています。
- server_c
  サーバー側のアプリケーションです。

ディレクトリ「server_c」には、ユーザーインタフェースを含んでいる thin クライアントアプリケーションに接続するサーバーを構築するために必要なファイルがすべて含まれています。このディレクトリ全体を取り出して構築し、リモート Web サーバー上で実行することができます。

最上位ディレクトリの Makefile には、クライアント (ユーザーのデザイン) とサーバーの両方を構築するための規則が含まれています。クライアントを構築するには、コマンド行から

```
make
```

と入力するだけです。サーバーを構築するには、コマンド行から

```
make server
```

と入力します。

注 - サーバーコードをどこかに移動する場合には、Makefile も忘れずに移動してください。

---

thin クライアントとインターネット用コードの追加先

以下の種類のスタブは、thin クライアント用やインターネット用のスマートコードのコールバックを作成するときに生成できます。これらのいくつかは任意指定であり、「カスタマイズ」ダイアログとグループエディタでルーチンを指定したかどうかによって選択します。

- ツールキットに依存するクライアントコールバック
- ツールキットに依存しないクライアントコールバック
- サーバーコールバック (thin クライアント用スマートコード専用)
- 事前条件ルーチンと事後条件ルーチン
- インターネット接続関数
- グループの「その他のデータ」関数

以下、これらを個別に説明します。

参照 - 独自のデータハンドラを追加する方法の詳細については、597 ページの「カスタムデータハンドラ」を参照してください。グループに特別な関数を追加する方法については、557 ページの「その他のデータ ― 関数」を参照してください。

ツールキットに依存するクライアントコールバック

従来の、ツールキットに依存するクライアントコールバックを含んでいるファイルは次のとおりです。

- untitled_stubs.c (C++ コードを生成する場合には untitled_stubs.cpp)

これは、スマートコードの使用とは無関係に、任意のコールバックの定義時に生成されるコールバックファイルです。thin クライアント用またはインターネット用のスマートコードのコールバックを定義するとき、Sun WorkShop Visual は、このファイルにクライアント側のコールバックを自動生成します。これにはツールキットに依存するコードが含まれています。ここには、ツールキットを使用するコードだけを追加してください。

ツールキットに依存しないクライアントコールバック

ツールキットに依存しないクライアントコールバックを含んでいるファイルは次のとおりです。

■ <コールバック>_cs.c (C++ コードを生成する場合は <コールバック>_cs.cpp)

thin クライアント用またはインターネット用のスマートコードのコールバックごとに、Callouts サブディレクトリにファイルが生成されます。ファイル名としては、コールバックの名前に「_cs」(クライアントサーバーを表す) を付加した名前を使用します。このファイルにはコールバックルーチン (C++ と Java の場合にはメソッド) が含まれています。これには、グループにアクセスできる、ツールキットに依存しないコードが含まれています。

サーバーコールバック

サーバーコールバックは、次のディレクトリに置かれています。

■ server_c
このサブディレクトリには、定義したコールバックごとにソースファイルがあります。このソースファイルにはコールバックと同じ名前が与えられます。

インターネット接続関数

「関数」が選択された場合には、「カスタマイズ」ダイアログの「接続」ごとにファイルが生成されます。各ファイルには、get_<接続> と set_<接続> のルーチンが含まれています。ファイルは、ファイル名として指定する関数の名前を使用して、Callouts サブディレクトリに生成されます。

「カスタマイズ」ダイアログで指定したカスタムデータハンドラにもファイルが生成されます。ファイル名は、「カスタマイズ」ダイアログに入力した名前です。

その他のデータ関数

608 ページの「取得と設定用ユーザーコードの追加先」で説明したように、グループエディタで定義された「その他のデータ」関数に対してスタブルーチンが生成されます。

「thin クライアント」用スマートコードの学習用に生成されたファイルの図式表示を、587 ページの図 17-5 に示します。

HTML ファイル

コールバックを探したり、生成されたファイルの内容をよく知るためには、Sun WorkShop Visual によって生成される一連の HTML ファイルが役立ちます。主要なファイルは以下のものです。

- index.html

最上位ディレクトリ (つまり、「コールバック」ダイアログで指定されたディレクトリ) にあります。

Web ブラウザ (または HTML を読み込める他のソフト) でこのファイルを開くと、図 17-13 に示すように、生成されるファイルの一覧と、それぞれの簡単な説明が表示されます。

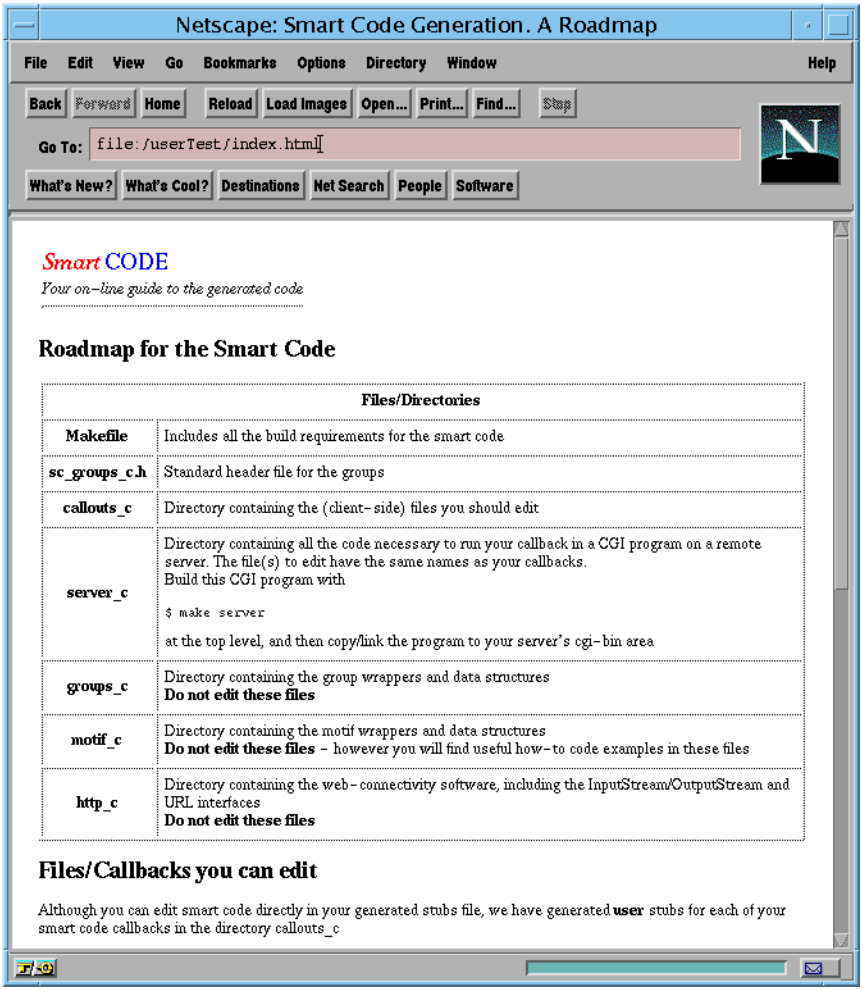


図 17-13 Netscape での index.html

他のファイルへのハイパーテキストリンクもあり、これも Sun WorkShop Visual によって生成されたものです。リンク先のファイルには、グループと Motif「ラッパー」ごとに生成されるコードが記述されています。

このファイルには、スマートコードをプログラミングするためのオンライン参考資料へのリンクも含まれています。この参考資料を直接表示するには、HTML ブラウザで次のファイルを開いてください。

```
$VISUROOT/lib/locale/<YourLocale>/sc/index.html
```

ここで、VISUROOT は Sun WorkShop Visual のインストールディレクトリであり、<YourLocale> はユーザーが使用しているロケールです。自分のロケールがわからない場合には、端末ウィンドウに **locale** と入力してみてください。これによってロケール情報が出力されます。出力の中で LANG に割り当てられた文字列を使用します。ロケールの例を次に示します。

- C (英語用)
- ja (日本語用)

第18章

インターネット用スマートコード

はじめに

コールバックに「インターネット」用スマートコードを選択すると、Sun WorkShop Visual は、デザインからクライアントアプリケーションを生成します。インターネット用スマートコードのプログラミングでは、WWW を介して公開サーバー上の既存の Web ページや CGI プログラムにアクセスします。スマートコードのコールバックは、(「callouts」というサブディレクトリの) クライアントアプリケーションに生成されます。「インターネット」のコールバックを定義した場合に、Sun WorkShop Visual によって生成されるアプリケーションの構造を図 18-1 に示します。thin クライアント用スマートコードの場合とは異なり、この種類のコールバックには、クライアントアプリケーションと通信コードだけが生成されます。

グループは、getter や setter とともに、あらゆる種類のスマートコードの基本となります。したがって、インターネット (または thin クライアント) 用スマートコードを使用するには、グループと取得/設定用スマートコードの使い方を理解する必要があります。これらの内容については、以下の章を参照してください。

1. ウィジェットのグループ化については、549 ページの第 15 章「グループ」を参照してください。
2. 559 ページの第 16 章「取得と設定用のスマートコード」では、デザイン内のウィジェットに対してツールキットに依存しないラッパーを提供する、取得/設定用スマートコードについて説明しています。

「試行」機能により、Sun WorkShop Visual のダイナミックディスプレイをプロトタイプクライアントとして使用できます。これによって、インタフェースの開発時に、インタフェースを生データでテストできます。618 ページから始まる学習では、きわめて簡単な例を通じて、この操作方法とアプリケーションの生成方法を示します。

着信データの処理や操作のためには独自の受信ハンドラを記述する必要があるため、インターネット用スマートコードのプロトタイプで「試行」を使用する際には制限があります。

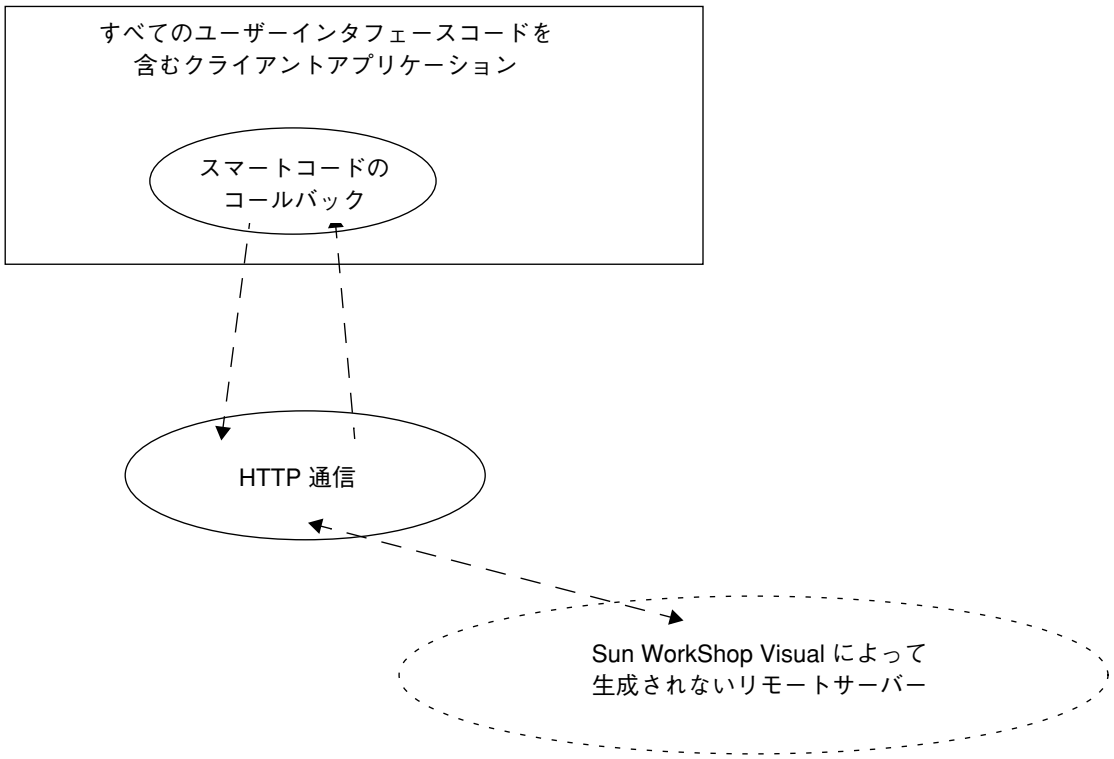


図 18-1 インターネット用コールバックアプリケーションの構造

インターネットと thin クライアント

インターネット用と thin クライアント用のスマートコードは次の点でよく似ています。

- ツールキットに依存しない getter と setter を使用する
- 通信の手段として HTTP (Sun WorkShop Visual URL API を介して生成) を使用する
- デザインをクライアントアプリケーションにする

このため、Sun WorkShop Visual の内部では、ユーザーインタフェースのいくつかの部分が共有されています。インターネットを使用するには、第 17 章の以下の節を参照してください。

- 592 ページの「サーバー接続のカスタマイズ」
- 601 ページの「試行」
- 606 ページの「生成コード」

インターネットと thin クライアントは以下の点で異なります。

- インターネット用スマートコードでは、サーバーアプリケーションは生成されない
- インターネット用スマートコードでは、(POST ではなく) GET HTTP プロトコルが使用される
- インターネットデザインは *thick* クライアントであるとみなされるため、デザインの組み立て方法が異なる

インターネット用スマートコードで生成されたアプリケーションは、以下の目的に使用できます。

- WWW ページの内容の取り出しと解析
- 既存のリモートサーバーへの接続
- 別の Sun WorkShop Visual デザインから生成されたサーバーとの交信

データの受信

インターネット用スマートコードで生成されたアプリケーションには、受信ハンドラが必要です。Sun WorkShop Visual は返却データへのポインタを提供しますが、そのデータの処理はユーザーが行います。データハンドラは「カスタマイズ」ダイアログの一部であり、592 ページの「サーバー接続のカスタマイズ」に記述されています。返却されたデータを利用するために、Sun WorkShop Visual はライブラリを提供しています。このライブラリを使用すれば、入力ストリームの特定項目にインタレスト (関心) を表明し、それらの項目が到着したら「選び出す」ことができます。これについては 624 ページの「HTML データから情報を抽出」を参照してください。

データをストリームとして処理したり、Sun WorkShop Visual が提供する InputData クラスやオブジェクトを使用して `getData()` や `getSize()` のメソッドを介してデータにアクセスすることができます。たとえば、gif や jpeg のイメージなど、表示ウィジェットに送信するデータをダウンロードしている場合には、これは特に便利です。C コードの場合には、InputData オブジェクトはデータ構造体です。C++ と Java の場合には、これはクラスです。InputData の詳細をオンライン参照資料で調べるには、HTML ブラウザで次のファイルを開き、適切なリンクに従ってください。

```
$VISUROOT/lib/locale/<YourLocale>/sc/index.html
```

ここで、VISUROOT は Sun WorkShop Visual のインストールディレクトリであり、`<YourLocale>` は使用しているロケールです。

着信データの処理方法に関する簡単な例として、「試行」トグルをオンにした状態でインターネット用スマートコードのコールバックを設定した後、デザインからコードを生成します。この場合には、受信ハンドラとして「@<ウィジェット名>」という短縮形表記を使用します。これは、618 ページの「インターネット用スマートコードの簡単な学習」で行う操作とまったく同じです。

通信プロトコル

「インターネット」用スマートコードを選択した場合には、Sun WorkShop Visual はインターネット上のサイトからデータを取り出すものと想定するため、GET HTTP プロトコルを使用します。「カスタマイズ」ダイアログで関数名を指定して送信ハンドラを無効にした場合には、Sun WorkShop Visual は POST プロトコルを使用します。

インターネット用スマートコードの簡単な学習

次の例では、インターネット用スマートコードについて簡単に紹介します。ここでは、実際の Web サイトに接続して、そこからデータをダウンロードします。その様子を見るには、Sun WorkShop Visual の「試行」を使用するか、生成したアプリケーションを実行してください。

インターネット用スマートコードの使い方に早く慣れていただくため、次の例では返却データの解析を行いません。HTML の解析については、624 ページの「HTML データから情報を抽出」を参照してください。

注 - この学習ではリモートサーバーへの接続方法を示します。したがって、リモートサーバーに接続できるよう構成されたコンピュータをご使用ください。

1. 図 18-2 に示すウィジェットを含んでいる階層を作成します。

   これは、上位階層から順にアプリケーションシェル、フォーム、{ボタン、スクロールテキスト} となります。

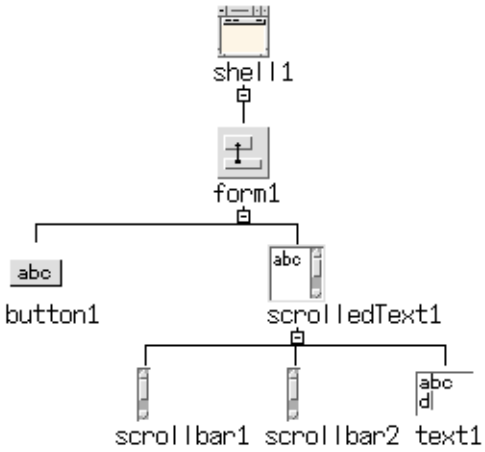


図 18-2 インターネット学習用の階層

2. 配置エディタで、スクロールテキストウィジェットをフォームの下端と右端に接続して、ウィンドウのサイズ変更に応じてこのウィジェットがサイズ変更できるようにします。

   これは、返却データを見やすくするための表面的な手順です。

3. text1 (スクロールテキストウィジェットのテキスト領域) を選択します。ツールバーの「新規グループに追加」ボタンを押します。

   このボタンを図 18-3 に示します。

図 18-3 「新規グループに追加」ツールバーボタン

4. 表示されたグループエディタで、図 18-4 に示すようにテキストウィジェットだけをグループのメンバーとして持つ Group0 というグループがあることを確認します。

図 18-4 グループエディタ

5. グループエディタを閉じます。

   何も変更する必要はありません。この学習では Sun WorkShop Visual によって作成されたグループを使用します。

6. button1 を選択し、「コールバック」ダイアログを表示します。

7. 左側のリストから「活性化」が選択されていることを確認し、コールバックの名前として `goInternet` を指定します。

   スマートコードを定義する必要があるため、まだこのコールバックを追加しないでください。

8. 「スマートコード」トグルをオンにします。

9. スマートコードの「スタイル」オプションメニューから「インターネット」を選択します。

10. このコールバックのグループとして Group0 を選択します。

    そのためには、「グループ」ボタンを押し、グループエディタで Group0 が選択されていることを確認してから「適用」ボタンを押します。

11. 「カスタマイズ」ボタンを押します。

    これによって「カスタマイズ」ダイアログが表示されます。

12. 「カスタマイズ」ダイアログで、URL フィールドに次の URL を入力します。

    `http://www.ist.co.uk/index.html`

13. ファイアウォールの保護下にある場合には、プロキシ・ホストとポートを設定します。

参照 - プロキシ設定の詳細については、595 ページの「プロキシの詳細」を参照してください。

---

14. 「受信 (URL の内容)」フィールドに次のように入力します。

    `@text1`

参照 - これらのフィールドでの「@」の使い方については、580 ページの「試行」を参照してください。

---

15. 「カスタマイズ」ダイアログで「了解」を押します。

    完成した「カスタマイズ」ダイアログを図 18-5 に示します。

注 - この「カスタマイズ」ダイアログは、一例として架空のプロキシを示します。595 ページの「プロキシの詳細」で説明するように、使用するネットワークに適切なプロキシを入力する必要があります。

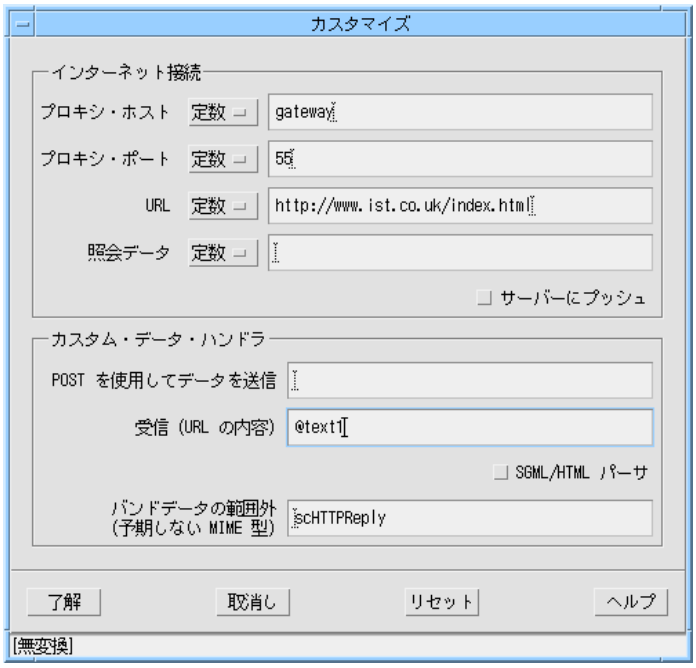


図 18-5 完成した「カスタマイズ」ダイアログ

16. 「追加」を押して新しいコールバックを追加します。

17. 「コールバック」ダイアログで、「試行」トグルをオンにします。

    「試行」をオンにした場合には、コールバックを「更新」する必要はありません。コールバックはその時点で試行可能になります。

18. ダイナミックディスプレイで、button1 を押します。

    リモートサーバーとの接続が行われている間は一時停止状態になります。その後、図 18-6 に示すように、返却データ (「カスタマイズ」ダイアログで指定した Web ページ) が text1 に表示されます。

図 18-6 試行中のダイナミックディスプレイ

この学習の最終段階では、生成されたアプリケーションでも同じものが表示されます。

19. 作成したデザインのコードを生成します。

    コンパイル可能であればどんな様式のコードでも生成できます。

20. 生成されたコードをコンパイルします。

21. クライアントアプリケーションを実行します。

    アプリケーションがリモートサーバーに接続され、指定した Web ページが表示されます。

さらに進んだ学習

引き続き、thin クライアント用とインターネット用のスマートコードのさらに高度な機能を試してみることもできます。Sun WorkShop Visual のパッケージには、ユーザー用の学習を実行する「Sun WorkShop Visual 再現」スクリプト実行用の命令を含んだ HTML ファイルが付属しています。HTML ブラウザで以下のファイルを開いて学習が実行される様子を観察し、その結果を確認してください。

1. `$VISUROOT/lib/locale/<YourLocale>/sc/timex.html`
   このファイルでは、自動リモート更新によるアプリケーションの作成方法を示す「サーバーにプッシュ」学習について説明します。

2. `$VISUROOT/lib/locale/<YourLocale>/sc/parsex.html`
   このファイルでは、Web ページを取り出して解析するアプリケーションの作成方法について説明します。

注 - VISUROOT は Sun WorkShop Visual のインストールディレクトリであり、<YourLocale> は使用しているロケールの名前です。

HTML データから情報を抽出

インターネット用スマートコードで開発したアプリケーションでは、Web ページを取り出し、それを解析し、その結果を表示できます。サーバーから返された HTML データを整理してそのプロセスを大幅に単純化し、さらに使いやすくできるように、Sun WorkShop Visual には HTML パーサーが付属しています。

WWW の概念として、Web サーバーから取り出されるデータの大部分は HTML 形式になります。パーサーは、SGML User Groupの基準 SGML パーサー資料[1] をベースにしています。このパーサーは、汎用の SGML パーサーエンジンを生成するよう改造されたものです。SGML は、DTD (Document Type Definition: 文書タイプ定義) と連結して、マークアップ言語を定義します。HTML 用の DTD は、Sun WorkShop Visual に付属しています。

DTD を別途追加することにより、他の規格のマークアップ言語や社内用のマークアップ言語でもパーサーを使用できます。また、HTML の将来のバージョンや XML に対しても、アプリケーションをアップグレードすることができます。

SGML パーサーには、簡単で便利なプログラミングインタフェースがあります。入力ストリーム (つまり、HTML タグ) の 1 つ以上の項目にインタレスト (関心) を登録すれば、パーサーがこれらの項目を検出するたびに、選択したルーチンが呼び出されます。これは、ウィジェットのコールバック方式と似ています。ウィジェットは特定のアクションにインタレストを登録します。そのようなアクションが発生すると、所定のルーチンが呼び出されます。

1. Standard Generalized Markup Language User's Group (SGMLUG) SGML パーサー資料。James Clark 著。

注 - ここで使用される Web 技術についてよくご存じない (または頭字語の意味がわからない) 場合には、背景となる知識を得られるようお勧めします。推奨書籍の一覧については、1014 ページの「ネットワーキングとWWWに関する資料」を参照してください。

パーサーを使用して着信データから必要な情報を抽出したいことを Sun WorkShop Visual に通知するには、「カスタマイズ」ダイアログで「SGML/HTML パーサ」トグルをオンにします。受信ハンドラでは、必要条件に応じてパーサーを設定してから、データをパーサーに送信します。この操作 (およびその結果) の詳細を以下に説明します。

パーサーの使用

「カスタマイズ」ダイアログで「SGML/HTML パーサ」トグルをオンにして、SGML/HTML を処理したいことを Sun WorkShop Visual に通知した後は、以下の 4 つの手順が必要となります。各手順は、受信ハンドラの内部で行われます。

1. ルーチン `scRegisterSGMLMimeType` (または HTML `scRegisterHTML` のショートカット) を呼び出すことによって、MIME 型を登録します。
2. ルーチン `scRegisterSGMLMimeErrorHandler` を呼び出すことによって、エラーハンドラを登録します。この手順は省略可能です。
3. ルーチン `scAddTagCallback` と `scAddAttrCallback` を呼び出すことによって、入力ストリームの 1 つ以上の項目にインタレストを登録します。あるいは、この時点で従来の構文解析ツリーを要求することもできます。
4. ルーチン `scProcessSGML` を使用してパーサーを呼び出します。

これらの手順を行う際は、次のことに注意してください。

パーサーへのインタフェースをプログラムする前に、次のヘッダーファイルをインクルードしていることを確認してください。

```
#include <SGML.h>
```

このヘッダーファイルは Sun WorkShop Visual に付属しています。このヘッダーファイルのディレクトリは、自動的に Makefile に組み込まれます。

さらに、DTDDIR 環境変数には次の値を設定する必要があります。

```
$VISUROOT/src/sgml/dtds
```

SGML パーサーの位置と使用するファイルの詳細については、637 ページの「パーサーを使用するための実用情報」参照してください。

MIME 型の登録

パーサーを構成するには、最初に SGML オブジェクトを作成する必要があります。その後、このオブジェクトは、呼び出す必要のある他のルーチンに渡されます。SGML オブジェクトは、次に示す、データの MIME 型を登録するために呼び出すルーチンから返されます。

```
SGML_t *
scRegisterSGMLMimeType( mimetype, dtd)
        char * mimetype;
        char * dtd;
```

これを使用して、MIME 型と SGML DTD を関連付けます。最も一般的な方法を次に示します。

```
SGML_t * sgm = scRegisterSGMLMimeType("text/html", "HTML32.soc");
```

これは最もよく使用されるため、次のショートカットを使用することもできます。

```
SGML_t *
scRegisterHTML( mimetype)
        char * mimetype;
```

この動作は上とまったく同じであり、「text/html」と HTML32 DTD を関連付けます。独自の DTD を追加するには、それを DTDDIR 環境変数によって参照されるディレクトリに置いてください。

次に、「SGML/HTML パーサ」トグルをオンにした状態で、「受信 (URL の内容)」フィールドに「processMyData」を指定した場合に生成されるものを示します。MIME 型を登録するために 1 つの行が追加されています。

```
int
processMyData ( data, idata)
        sc_stdcs_t* data;
        sc_idata* idata;
{
                extern InputData * newInputData();
```

```
                group0_t * group = (group0_t*)data->group;
                InputStream * i   = (*idata->getInputStream)(
idata);
#if 0 /* example usage */
                char      * type  = (*idata->getMimeType)(idata);
                int         len   = (*idata->getContentLength)(
idata);

                InputData * id    = newInputData( i);
                char *   d        = (*id->getData)( id);
#endif
追加された行──▶ sgm = scRegisterHTML( type);
                ...
                return 0;
}
```

エラーハンドラの登録

デフォルトのエラーハンドラは、標準出力にエラーメッセージを出力します。これを無効にするには、次のルーチンを使用して、独自のエラーハンドラを登録してください。

```
int
scRegisterSGMLErrorHandler( errorhandler)
        void_f errorhandler;
```

ユーザーのエラーハンドラは、次の形式にしてください。

```
void
errorhandler( s)
        char * s;
```

入力ストリーム項目にインタレストを登録

解析されたデータを利用するため、入力の 1 つ以上の項目 (たとえば、HTML 内の特定のタグや属性) にインタレストを登録します。

入力項目へのインタレストの登録は、ウィジェットのコールバック方式によく似ています。ウィジェットのコールバックの場合、ウィジェットは特定のアクションにインタレストを登録します。そして、そのアクションが起こると、指定したルーチン (コールバック) が呼び出されます。ここでは、言語の項目にインタレストを登録します。パーサーは、これらの項目のどちらかを検出すると、ユーザーのコールバックルーチンを呼び出します。

HTML には、タグと属性という 2 つの主要項目があります。以降の節で説明する 2 つのルーチンのどちらかを使用して、これらの項目にインタレストを登録できます。最初に、タグと属性を簡単に説明しておきます。

タグ

タグは、以下のテキスト要素の書式を記述する HTML の項目です。タグは山括弧で囲まれて表示されます。たとえば、<menu> は中黒付きリストを示し、<code> はコードのリストを示します。次の例では、個々のリスト項目を含んだ「menu」ブロックを示します。

```
<menu>
                <li>The first item in the list
                <li>The second item in the list
</menu>
```

属性

属性は HTML のもう 1 つの項目です。属性も山括弧で囲まれて表示されます。属性はタグの後に置かれ、参照を示すために使用されます。参照先としては、外部ファイル、イメージ、文書内の他の位置などがあります。属性は常に、予約文字列である等号記号 (=) と参照から構成されています。次の例では 2 つの属性を示します。最初の属性「href」はリンクの宛先 (「bottom」という場所) を指定します。ブロック内の「Go to bottom of page」というテキストは、実際に「目に見える」部分です。2 番目の属性は「name」であり、場所を指定します (この場合は「bottom」)。したがって、ユーザーの側から見れば、「Go to bottom of page」を選択すると、指定された位置である「bottom」にある内容が表示されます。

```
<a href="#bottom"><b>Go to bottom of page</b></a>

...

<a name="bottom"></a>
```

特定の HTML 言語項目にインタレストを登録するために使用できる、2 つのルーチンがあります。これらは次のとおりです。

1. scAddTagCallback
   特定のタグにインタレストを登録します。

2. scAddAttrCallback
   属性にインタレストを登録します。これは scAddTagCallback と同じですが、属性内だけのインタレストを表示するよう設定されています。

以下の節では、これらの登録ルーチンについて説明します。

タグにインタレストを登録

SGML パーサーは、HTML 入力のどの部分にユーザーが関心を持っているかを知る必要があります。また、選択した HTML ブロック内のどこでコールバックルーチンを呼び出すかも知る必要があります。

タグ要素にインタレストを登録するためのルーチンは次のとおりです。

```
int
scAddTagCallback( sgm, tagname, type, callback, data)
                  SGML_t * sgm;
                  char * tagname;
                  int type;
                  void (*callback)();
                  void * data;
```

このルーチンへの引数には、さらに説明が必要です。これについては以下の項を参照してください。

SGML_t * sgm

これは、データの MIME 型を登録するルーチンである、`scRegisterSGMLMimeType` から返される SGML オブジェクトです。`scRegisterSGMLMimeType` については、626 ページの「MIME 型の登録」を参照してください。

char * tagname

これは、ユーザーが関心を持っているタグです。山括弧は含めずに、たとえば「A」、「MENU」、「LI」のように、タグ自身を大文字で記述してください。

int type

この引数は、選択されたタグ内でユーザーのコールバックルーチンをいつ呼び出すかをパーサーに通知します。さらに、HTML の選択したブロックの内部からコールバックルーチンにデータが渡されるかどうかは、選択した「型」によって決まります。ユーザーのコールバックルーチンは、任意の数の異なるタグと型に同じルーチンを使用できるよう、常にタグ名と型名を受け取ります。しかし、「ON_ATTR」と「ON_DATA」の場合には、より多くの情報が返されます。

図 18-7 に示すように、タグブロック内のどこでコールバックルーチンを呼び出すかに応じて、4 つの定義済みの型を選択できます。4 つの型は次のとおりです。

- ON_ENTRY
  ブロックの先頭を表します (たとえば <a> や <menu>)。データ (632 ページの「ユーザーのコールバックルーチン」では *call_data* と呼ぶ) はユーザーのルーチンに渡されません。
- ON_EXIT
  ブロックの最後を表します (たとえば や </menu>)。データ (632 ページの「ユーザーのコールバックルーチン」では *call_data* と呼ぶ) はユーザーのルーチンに渡されません。
- ON_ATTR
  タグ定義の内部に現われる属性を表します (たとえば、href="mylink")。ユーザーのコールバックルーチンは、call_data 引数として引用符内部のテキストを受け取ります (632 ページの「ユーザーのコールバックルーチン」で説明します)。
- ON_DATA
  タグの後にあるテキスト (またはデータ) を表します。ユーザーのコールバックルーチンは、タグの先頭と最後の間にあるテキストを call_data 引数として受け取ります (632 ページの「ユーザーのコールバックルーチン」で説明します)。

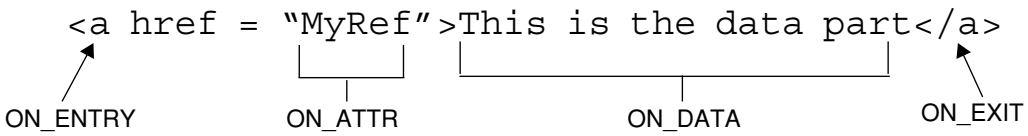


図 18-7 型

void (*callback)()

これは、パーサーがユーザーの関心のあるタグを検出したときに呼び出すルーチンの名前です (コールバックルーチン)。これはユーザーが定義するルーチンです。このルーチンの形式については、632 ページの「ユーザーのコールバックルーチン」を参照してください。

void *data

この引数によって、ユーザーのコールバックルーチンにデータを渡すことができます。これは、ユーザーのルーチンに「クライアントデータ」引数として渡されます。

タグごとに 1 回このルーチンを呼び出すことによって、任意の数のタグにインタレストを登録できます。

属性にインタレストを登録

属性にインタレストを登録するルーチンは次のとおりです。

```
int
scAddAttrCallback( sgm, tagname, attrname, callback, data)
                SGML_t * sgm;
                char * tagname;
                char * attrname;
                void (*callback)();
                void * data;
```

以下の引数だけは、上記のタグ登録ルーチン用に記述されたものとは異なります。

- `char * attrname`
  これは関心のある属性の名前です。

任意の数の属性にインタレストを登録できます。

ユーザーのコールバックルーチン

ユーザーのコールバックルーチンにはスタブファイルがありません。すべて自分で記述する必要があります。ルーチンの名前は、`scAddAttrCallback` または `scAddTagCallback` への 4 番目の引数 (つまり「callback」という引数) として指定された名前です。

パーサーは、インタレストの登録されたタグや属性を検出すると、ユーザーのコールバックルーチンを呼び出します。次の例では、ユーザーのコールバックの内容と渡される引数の一覧を示します。

```
int
mycallback( tag, attribute, type, call_data, client_data)
        char * tag;
        char * attribute;
        int    type;
        void * call_data;
        void * client_data;
```

ユーザーのルーチンに渡される引数は次のとおりです。

1. `char * tag`
   パーサーが検出したタグです。

2. `char * attribute`
   パーサーが検出した属性です。タグだけに関心があった場合には、これはヌルです。

3. `int type`
   ルーチンの呼び出し型が「ON_ENTRY」、「ON_EXIT」、「ON_DATA」、「ON_ATTR」のいずれであるかを示します。これらについては、630 ページの「int type」を参照してください。属性に関心がある場合には、この引数は関係ありません。

4. `void * call_data`
   属性に関心がある場合には、これは等号記号の後に記述される属性部分です。たとえば、「href」属性にインタレストを登録し、パーサーが次の行を検出した場合、この引数は「`#regmime`」となります。

```
<a href="#regmime">
```

インタレストのタグとして「ON_DATA」を指定した場合には、この引数はタグの後のデータです。実例については、631 ページの図 18-7 を参照してください。

5. `void * client_data`
   これは登録ルーチンに渡される「データ」引数であり、ユーザー独自のデータをコールバックに渡すことができます。これは、「コールバック」ダイアログでの Xt コールバック用のクライアントデータに似ています。

任意の数のタグや属性にインタレストを登録できるため、任意の数のコールバックルーチンを持つことができます。しかし、タグと属性にそれぞれ 1 つのコールバックルーチンを持つことが、おそらく最も便利な組み合わせとなります。

入力ストリームの解析

MIME 型を指定し、エラーハンドラを登録し、特定の入力ストリーム項目にインタレストを登録して SGML オブジェクトを設定すると、パーサーを呼び出す準備ができたことになります。そのためのルーチンを次に示します。

```
int

scProcessSGML( sgm, istream)
```

```
                SGML_t * sgm;/* the parser handle
scRegisterSGMLMimeType */
                InputStream * istream;/* the input stream from the
server */
```

最初の引数は、前の節で説明されています。2 番目の引数である入力ストリームは、受信ハンドラに渡されます。

パーサーの使用例

この節では、SGML パーサーの使い方を説明します。「カスタマイズ」ダイアログで「SGML/HTML パーサ」トグルをオンにした場合には、受信ハンドラの名前も与える必要があります。Sun WorkShop Visual によって生成されたハンドラのスタブを次に示します。

```
int
processMyData ( data, idata)
        sc_stdcs_t* data;
        sc_idata* idata;
{
        extern InputData * newInputData();

        group0_t * group = (group0_t*)data->group;
        InputStream * i   = (*idata->getInputStream)( idata);
#if 0 /* example usage */
        char      * type  = (*idata->getMimeType)(idata);
        int         len   = (*idata->getContentLength)( idata);

        InputData * id    = newInputData( i);
        char *   d        = (*id->getData)( id);
#endif
        return 0;
}
```

SGML パーサーを使用するには、SGML オブジェクトを作成するためにこのルーチンにいくつかのコードを追加し、SGML オブジェクトを設定してから、着信データをパーサーに送信する必要があります。着信 HTML を解析するためのコードを追加した受信ハンドラを次に示します。

```
int
processMyData ( data, idata)
        sc_stdcs_t* data;
        sc_idata* idata;
{
                extern InputData * newInputData();

                group0_t * group = (group0_t*)data->group;
                InputStream * i   = (*idata->getInputStream)(
idata);
削除された行→#if 0 /* example usage */
                char      * type  = (*idata->getMimeType)(idata);
                int         len   = (*idata->getContentLength)(
idata);

                InputData * id    = newInputData( i);
                char *   d        = (*id->getData)( id);
削除された行→#endif

                SGML_t * sgm;
                if ( strcmp( type, "text/html") != 0)
追加された行→       return -1;
                sgm = scRegisterHTML( type); /* the parser object */
                (void) scAddTagCallback(sgm, "A", ON_ENTRY,
getanchor,
                       "a-call");
                (void) scAddAttrCallback(sgm,  "A", "HREF",
                       getlinkinfo, "href");
```

```
                (void) scProcessSGML( sgm, i);

追加された行 ─

                return 0;
}
```

このルーチンでは、パーサーがアンカータグ (<a>) を検出するたびに getanchor を呼び出し、「href」属性が検出されるたびに getlinkinfo を呼び出すことを指定しています。以下のルーチンは、ユーザーが記述してください。

```
int
getanchor( tag, attr, type, call_data, client_data)
                char * tag;
                char * attr;
                int    type;
                void * call_data;
                void * client_data;
{
                printf("anchor-start(%s)\n", client_data);
}

int
getlinkinfo( tag, attr, type, call_data, client_data)
                char * tag;
                char * attr;
                int    type;
                void * call_data;
                void * client_data;
{
                printf( "%s=%s\n", client_data, call_data);
}
```

注 - このパーサーを使用する、Sun WorkShop Visual 再現機能によるオンラインスクリプトを実行する方法については、623 ページの「さらに進んだ学習」を参照してください。

パーサーを使用するための実用情報

SGML パーサーを使用するには、コンパイル済みのコードとリンクする必要があります。ソースの場所を次に示します。

```
$VISUROOT/src/sgml
```

ライセンス条項により、ユーザーはこれらのソースを自由に使用できます。

最初は Sun WorkShop Visual に付属のコンパイル済みバージョンを使用する方が簡単です。

SGML パーサーは、以下のファイルとディレクトリを使用します。

1. `$VISUROOT/lib`
   ここには、アーカイブと SGML ライブラリの共有バージョンが含まれています。Sun WorkShop Visual が `Makefile` に生成した構築ルールでは、`libsgml.so` を使用します。しかし、好みに応じて `libsgml.a` とリンクすることもできます。

2. `$VISUROOT/src/sgml/hdrs/SGML.h`
   これは、パーサーエンジンを使用するために必要なインクルードファイルです。スマートコードの「カスタマイズ」ダイアログで「SGML/HTML パーサ」トグルをオンにした場合には、`Makefile` では API が参照されます。

3. `$VISUROOT/src/sgml/dtds`
   これは、HTML 3.2 DTD および他の関連するデータファイルを含んでいるディレクトリです。パーサーはこれを見つける必要があるため、DTDDIR 環境変数に次の値を設定する必要があります。

   ```
   $VISUROOT/src/sgml/dtds
   ```

コンパイルする前に、以下のことを確認してください。

1. `$VISUROOT/bin` が PATH に含まれていること

2. `$VISUROOT/lib` がライブラリのパス環境変数 (たとえば、32 ビットアプリケーション用の LD_LIBRARY_PATH、64 ビットアプリケーション用の LD_LIBRARY_PATH64) に追加されていること

第19章

メークファイル生成

はじめに

本章では、Sun WorkShop Visual のメークファイル生成機能について説明します。Sun WorkShop Visual は、単純メークファイルあるいはテンプレート付きメークファイルの 2 種類を作成することができます。単純メークファイルは局所的な `.xd` ファイルに対する構築規則だけを含んでいるため、単一のファイル構成のアプリケーションにのみ役立ちます。テンプレート付きメークファイルを更新して、以前の作業を書き換えることなくファイルを追加することができるため、多くの生成ファイルとは異なり、編集および再生成によって以前の作業内容が失われることはありません。

本章では、デザインに対してコードを生成するときに使用できるメークファイルオプションについて説明します。また、簡単な学習例を使用して、テンプレート付きメークファイルを作成し、アプリケーションに第 2 のデザインファイルが追加された場合にメークファイルを更新するための指示について、段階を追って説明していきます。この学習例によって、Sun WorkShop Visual のメークファイル生成機能について十分理解することができます。

メークファイル生成オプション

「メークファイル」テキストボックスの横の「オプション」ボタンを押すと、図 19-1 に示すようなダイアログが表示されます。

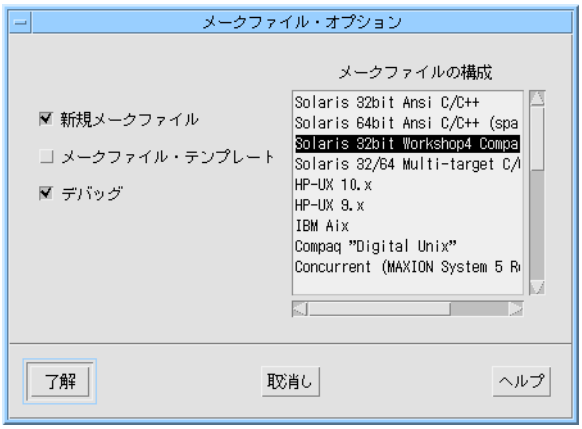


図 19-1 「メークファイル・オプション」ダイアログ

このダイアログには、「新規メークファイル」、「メークファイル・テンプレート」、「デバッグ」という 3 つのトグルとスクロールリストがあります。「デバッグ」をオンにするとコンパイラに送られるフラグリストに -g フラグが追加されて、デバッグ用のアプリケーションのバージョンを構築することができます。

「新規メークファイル」トグルと「メークファイル・テンプレート」トグル

「新規メークファイル」と「メークファイル・テンプレート」は、ユーザーが生成することのできる 2 種類のメークファイル (単純なメークファイルとテンプレート付きメークファイル) と関連しています。どちらのメークファイルが生成されるかは、このダイアログのどのトグルがオンになっているかによって異なります。2 つのトグルは相互関連して動作します。トグルの設定方法には次の 4 通りがあります。

1. 「新規メークファイル」トグルをオンにして、「メークファイル・テンプレート」トグルをオフにする

単純メークファイルが生成されます。メークファイルが構築するアプリケーションに他のデザインファイルを追加しない場合は、このオプションを使用します。

2. 両方のトグルをオンにする

テンプレート付きメークファイルが生成されます。たとえば、次回からの生成でファイルを更新するためのテンプレートとなる構造化コメント付きのメークファイルなどが生成されます。メークファイルが構築するアプリケーションに他のデザインファイルを追加する場合は、このオプションを使用します。

3. 「新規メークファイル」トグルをオフにして、「メークファイル・テンプレート」トグルをオンにする

2 で生成されたメークファイルに現在のデザインコードファイルが追加されます。

4. 両方のトグルをオフにする

3 と同じ結果になります。

メークファイルの種類のリスト

「メークファイル・オプション」ダイアログのスクロールリストを使用して、特定のプラットフォーム用のメークファイルを生成できます。プラットフォームによっては複数のオプションを使用することができるため、さまざまな環境に適用できます。たとえば、Solaris では、アプリケーションを 32 ビットアーキテクチャにも 64 ビットアーキテクチャにもコンパイルできます。プラットフォームのデフォルトは最初から選択されています。アプリケーションを構築するメークファイルを別のプラットフォームに生成する場合や、デフォルト以外のコンパイラを使用する場合は、このデフォルトを変更します。アプリケーションを実行する環境に合わせて任意の数の各種メークファイルを生成できます。

指定のプラットフォームに対して複数のオプションがある場合などは、「マルチターゲット」メークファイルを生成できます。メークファイル 1 つで、複数の任意のターゲット (使用可能なもののみ) を指定できます。たとえば、「メークファイル・オプション」ダイアログから次を選択するとします。

```
Solaris 32/64 Multi-target C/C++ (sparc)
```

この場合、「メークファイル・オプション」ダイアログから使用可能な他のすべての Solaris メークファイル (以下参照) と同じことを実行できるメークファイルが生成されます。

```
Solaris 32bit Ansi C/C++

Solaris 64bit Ansi C/C++ (sparc)

Solaris 32bit Workshop4 Compatible C++
```

マルチターゲットメークファイルとは、「メークファイル・オプション」ダイアログのスクロールリスト中で、「Multi-target」という語を含むメークファイルのことです。使用できるコマンド行オプションについては、メークファイルを調べるとわかります。

アプリケーションのバージョンを 1 つだけ構築する場合は、適切なメークファイルタイプを選択します。アプリケーションを何度も構築する場合で、1 回ごとに指定のプラットフォームに異なるターゲットを指定するには、「マルチターゲット」メークファイルが最も便利です。

メークファイル生成時の諸注意

メークファイルを生成する場合、以下の点に注意してください。

- テンプレート付きメークファイルを選択する場合は、メインデザイン (通常、アプリケーションシェルを含むデザイン) に対して最初のメークファイル (「新規メークファイル」と「メークファイルテンプレート」の両方のトグルをオンにして生成するメークファイル) を生成します。Sun WorkShop Visual では、アプリケーションの名前として、「新規メークファイル」を生成するデザインの一次ソースファイルの名前を使用します。
- 「メークファイル・テンプレート」だけを選択する場合は、生成済みのメークファイルの名前を指定してください。別の名前を指定するとエラーが報告されます。
- テンプレート付きメークファイルを編集し、変更内容を適用して、そのメークファイルを再生成できます。これは単純メークファイルには適用されません。

初期メークファイルの作成

第 1 段階では、デザインを作成し、デザインに対する C コードを生成して初期メークファイルを作成します。ここでは、Sun WorkShop Visual の一般的な使用方法についての知識があることを前提に説明を進めます。

1. 新しいディレクトリ `myapp` を作成します。このディレクトリを現作業ディレクトリとして Sun WorkShop Visual を起動します。
2. 図 19-2 に示すウィジェット階層を構築します。

   これは、ボタンを含むフォームを持つアプリケーションシェルです。

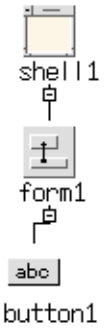


図 19-2 メインプログラムのウィジェット階層

3. プッシュボタンに活性化コールバックである *button_pressed* を指定します。

   メークファイルを作成する前に、それに含めるコードファイルを生成する必要があります。コードファイルを生成すると、デザインファイルでそのコードファイルの名前が設定されます。この作業が行われるまでは、メークファイル生成機能はファイルの名前を識別することができません。したがって、ファイルをメークファイルに追加することもできません。

4. 「コード生成」ダイアログを表示して、「言語」メニューで「C」が選択されていることを確認します。

   「コード生成」ダイアログには保存ファイル名が表示されます。デザインが保存されていない場合は、untitled と表示されます。

5. 「コード」テキストボックスに **`myapp.c`** と入力して、「生成」トグルをオンにします。

6. 「メインプログラム」テキストボックスに **`myapp.c`** と入力して、「生成」トグルをオンにします。

   「コード」テキストボックスと「メインプログラム」テキストボックスに表示されるファイル名が同一の場合は、コードファイルは main() 手続きとともに生成されます。

7. 「コードオプション」ダイアログで、「ヘッダーファイルをインクルード」トグルをオンにします。

8. 「コード生成」ダイアログの「スタブ」テキストボックスに **`app_stubs.c`** と入力して、「生成」トグルをオンにします。

9. 「コードオプション」ダイアログの「リンク」オプションメニューから「生成しない」を選択します。

   これで *myapp.c* および *app_stubs.c* のコンパイルとリンクを行うメークファイルを生成できる状態になりました。

10. 「メークファイル」テキストボックスに **Makefile** と入力して、「生成」トグルをオンにします。

11. 図 19-3 に示すように、「新規メークファイル」および「メークファイル・テンプレート」の両方のトグルをオンにします。

    ダイアログの右側のスクロールリストについては、641 ページの「メークファイルの種類のリスト」 で説明しています。

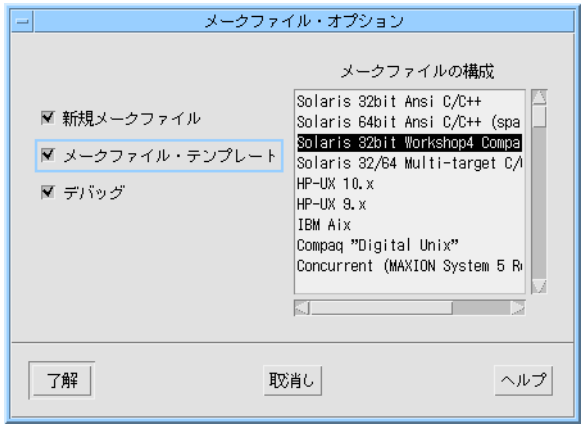


図 19-3 初期メークファイル生成

12. 「コード生成」ダイアログの「生成」ボタンを押します。

    生成されたメークファイルには、必要な作成規則や詳細な修正を行うためのテンプレート行が含まれています。テンプレート行以外に、生成されたメークファイルには以下の規則が含まれています。

```
XD_C_PROGRAMS=\
                        myapp
XD_C_PROGRAM_OBJECTS=\
                        myapp.o
XD_C_PROGRAM_SOURCES=\
```

```
                    myapp.c
XD_C_STUB_OBJECTS=\
                    app_stubs.o
XD_C_STUB_SOURCES=\
                    app_stubs.c
myapp: myapp.o $(XD_C_OBJECTS) $(XD_C_STUB_OBJECTS)
                $(CC) $(CFLAGS) $(CFPPFLAGS) $(LDFLAGS) -o myapp
myapp.o
$(XD_C_OBJECTS) $(XD_C_STUB_OBJECTS) $(MOTIFLIBS) $(LDLIBS)
myapp.o: myapp.c
                $(CC) $(CFLAGS) $(CPPFLAGS) -c myapp.c
app_stubs.o: app_stubs.c
                $(CC) $(CFLAGS) $(CPPFLAGS) -c app_stubs.c
```

13. 現在のデザインを *myapp.xd* に保存します。

初期メークファイルの更新

初期メークファイルを生成した場合には、その後の作業を反映させるためにメークファイルを更新することができます。 メークファイルの更新は、以下の手順に従って行います。

まず、別のファイルにポップアップダイアログを構築し、そのためのコードを生成してから、メークファイルを更新して新しいモジュールを反映させます。

1. ファイルメニューから「新規」を選択し、新しいデザインを開始します。
2. 図 19-4 に示すような階層を構築します。

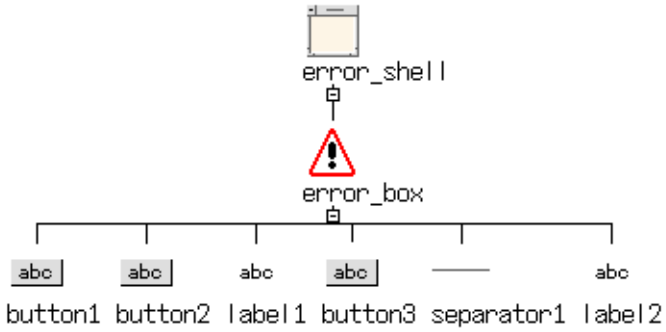


図 19-4 第 2 のポップアップダイアログ

3. シェルおよびメッセージボックスにそれぞれ *error_shell*、*error_box* と名前を付けます。
4. メッセージボックスに取り消しコールバックである *cancel_error* を指定します。
5. 「コード生成」ダイアログを表示して、「言語」メニューに「C」が選択されていることを確認します。
6. 「外部宣言」テキストボックスに **error.h** と入力して、「生成」トグルをオンにします。
7. 「コード」テキストボックスに **error.c** と入力して、「生成」トグルをオンにします。
8. 「コードオプション」ダイアログで「ヘッダーファイルをインクルード」トグルをオンにして、「ヘッダーファイルをインクルード」テキストボックスに **error.h** と入力します。
9. 「スタブ」テキストボックスに **error_stubs.c** と入力して、「生成」トグルをオンにします。
10. 「メインプログラム」の横にある「生成」トグルがオフになっていることを確認してください。
11. 「コードオプション」ダイアログの「リンク」オプションメニューから「生成しない」を選択します。
12. 「メークファイル」テキストボックスに **Makefile** と入力して、「生成」トグルをオンにします。

13. 図 19-5 に示すように「メークファイル・オプション」ダイアログで「新規メークファイル」トグルをオフにし、「メークファイル・テンプレート」トグルはオンのままにします。

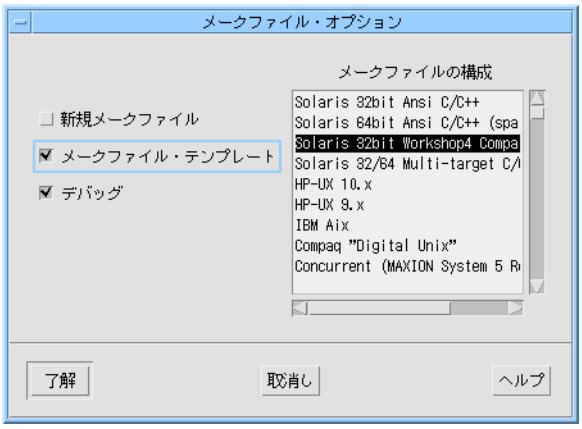


図 19-5 メークファイルの更新

14. 「コード生成」ダイアログの「生成」ボタンを押します。

生成されたメークファイルは、新しいモジュールで更新されます。

```
XD_C_PROGRAMS=\
myapp

XD_C_PROGRAM_OBJECTS=\
myapp.o

XD_C_PROGRAM_SOURCES=\
myapp.c

XD_C_OBJECTS=\
error.o

XD_C_SOURCES=\
error.c

XD_C_STUB_OBJECTS=\
error_stubs.o\
app_stubs.o

XD_C_STUB_SOURCES=\
error_stubs.c\
app_stubs.c
```

```
myapp: myapp.o $(XD_C_OBJECTS) $(XD_C_STUB_OBJECTS)
$(CC) $(CFLAGS) $(CPPFLAGS) $(LDFLAGS) -o myapp\
myapp.o $(XD_C_OBJECTS)\
$(XD_C_STUB_OBJECTS) $(MOTIFLIBS)\
$(LDLIBS)

myapp.o: myapp.c
$(CC) $(CFLAGS) $(CPPFLAGS) -c myapp.c

error.o: error.c
$(CC) $(CFLAGS) $(CPPFLAGS) -c error.c

app_stubs.o: app_stubs.c
$(CC) $(CFLAGS) $(CPPFLAGS) -c app_stubs.c

error_stubs.o: error_stubs.c
$(CC) $(CFLAGS) $(CPPFLAGS) -c error_stubs.c
```

アプリケーションの構築

Sun WorkShop Visual はこの時点で 2 個のダイアログに対してのそれぞれのコードファイル、2 個のスタブファイル、 1 個のメークファイルを生成しています。最後に 2 個のスタブを記入し、アプリケーションを構築します。

1. `app_stubs.c` を編集して、以下に示す変更を行います。

   生成されたインクルードのリストの最後尾に、次の行を追加します。

   ```
   #include <error.h>
   ```

   button_pressed コールバック・スタブに機能性を与えます。このコールバックは、ボタンが押された場合にエラーダイアログをポップアップするものです。

   ```
   void
   button_pressed (Widget w, XtPointer client_data, XtPointer
                                            call_data )
   {
       ...
       if ( error_shell == NULL )
             create_error_shell (XtParent (XtParent (w) ) );
       XtManageChild ( error_box );
   }
   ```

2. error_stubs.c を編集し、次に示す変更を行います。

   cancel_error コールバックに機能性を与えます。この関数は、ユーザーが「取消し」ボタンを押した場合にエラーダイアログを消去します。

```
void
Widget w, XtPointer client_data, XtPointer{
     ...
     XtUnmanageChild ( error_box );
}
```

3. 現在のデザインを error.xd に保存します。
4. ファイル myapp.xd を Sun WorkShop Visual で開きます。
5. ファイル myapp.res に X リソースを生成します。
6. これらのリソースを呼び出すために、コマンド行に以下の操作を行います。
7. C シェルを使用している場合は、コマンド行に以下のように入力します。

```
setenv XENVIRONMENT myapp.res
```

8. Bourne シェル (sh) または Korn シェル (ksh) を使用している場合は、コマンド行に、以下のように入力します。

```
XENVIRONMENT=myapp.res; export XENVIRONMENT
```

9. $VISUROOT を Sun WorkShop Visual のインストールディレクトリに設定します。

   $VISUROOT は、メークファイルがファイルおよびライブラリを参照する際に、Sun WorkShop Visual のインストールディレクトリへの相対パスとして使用します。

10. アプリケーションを構築します。コマンド行に以下のように入力します。

```
make
```

11. アプリケーションを実行させるために、以下のように入力します。

```
myapp
```

生成されたメークファイルの編集

メークファイルは、情報を失うことなく編集および再生成できます。一般的な変更を行うには、ファイルの先頭にあるメークファイルオプションを編集します。たとえば、コンパイラに ../hdrs ディレクトリ内でヘッダーファイルを検索させる場合には、以下に示す内容をメークファイル内の CFLAGS 行の最後に追加します。

```
-I../hdrs
```

CFLAGS に対する変更は、「新規メークファイル」トグルをオフにしてメークファイルを再生成する場合には保持されます。変更は、新しいメークファイルを生成する場合にのみ失われます。

テンプレート行の編集

生成されたメークファイルの大部分は、テンプレート行で構成されています。テンプレート行は、メークファイルへの情報の生成を制御するコメントです。テンプレート行には、#Sun WorkShop Visual: という接頭辞が付きます。たとえば、以下のテンプレート行は、C ソースファイル (XDG_C_SOURCE) から C オブジェクトファイルを作成するメークファイル行の生成方法を Sun WorkShop Visual に伝えます。

```
#Sun WorkShop Visual:XDG_C_OBJECT: XDG_C_SOURCE
#Sun WorkShop Visual:    $(CC) $(CFLAGS) $(CPPFLAGS) -c XDG_C_SOURCE
```

メークファイルを更新して、アプリケーションにファイルを追加するごとに、

Sun WorkShop Visual は各関連テンプレートに対してのテンプレートインスタンスを生成します。これらのインスタンスは、アプリケーションに対しての実際の構築コマンドを含んでいます。テンプレートインスタンスの開始および終了部分には、「DO NOT EDIT」コメントで印が付けられます。以下は前述のテンプレートの典型的なインスタンスの例です。

```
#DO NOT EDIT >>>
error.o: error.c
                    $(CC) $(CFLAGS) $(CPPFLAGS) -c error.c
#<<< DO NOT EDIT
```

テンプレートインスタンスは、編集しないことをお勧めします。編集しても次回のメークファイル生成時に、行なった編集が失われる場合があります。構築コマンドを変更するには、テンプレートインスタンスを編集する代わりに、対応するテンプレート行を編集します。テンプレート行を編集した後は、メークファイルにすでに存在するテンプレート行のインスタンスを削除します。インスタンスは、テンプレート行の直後にあります。

デバッグするために、すべての C ファイルを構築する例を示します。

1. テンプレート行を編集します。

   ```
   #Sun WorkShop Visual: $(CC) $(CFLAGS) $(CPPFLAGS) -c XDG_C_SOURCE
   ```

   を、次のように変更します。

   ```
   #Sun WorkShop Visual: $(CC) $(CFLAGS) $(CPPFLAGS) -g -c XDG_C_SOURCE
   ```

2. テンプレート行の次に続くインスタンスを削除します。

3. 「新規メークファイル」トグルをオフにして、アプリケーション内の各デザインに対しメークファイルを再生成します。

   この手順により、修正されたテンプレートを使用して新しいインスタンスが生成されます。

テンプレート構成

オリジナルのテンプレートは、以下のリソースによって示されるファイルにより指定されます。

```
visu.motifMakeTemplateFile: $VISUROOT/make_templates/motif
```

```
visu.mmfcMakeTemplateFile: $VISUROOT/make_templates/mfc
```

2 個のリソースが存在するため、異なるテンプレートをカスタマイズして、適切なクラスライブラリを選択することができます。リソースの値には、`/bin/sh` によって拡張される環境変数を持たせることができます。

Sun WorkShop Visual が、指定されているファイルを見つけることができない場合には、*makefileTemplate* というアプリケーションリソースを使用して、Sun WorkShop Visual リソースファイルに指定されているテンプレートにフォールバックします。すべての新しいメークファイルに大域的に適用されるテンプレートを変更するには、リソースファイルを編集します。詳細は、第 25 章「構成」を参照してください。

Sun WorkShop Visual は、新しいメークファイルの生成時にのみテンプレートリソースを参照します。既存のメークファイルにおけるテンプレートを変更するには、前節で説明したように手作業でファイルを編集するか、ファイルを削除してテンプレートを作成し直します。

依存情報

Sun WorkShop Visual は、メークファイルに依存情報を生成しません。デフォルトテンプレートには、depend (依存) 対象が含まれています。これは以下のコマンドを使用して呼び出すことができます。

```
make depend
```

この操作は、既存のメークファイルを走査して標準依存リストを追加する makedepend ユーティリティを呼び出します。詳細は、makedepend のマニュアルページを参照してください。

自身のメークファイルの使用

$VISUROOT/make_template ディレクトリには、メークファイルを生成するために使用するテンプレートがあります。このテンプレートには、サポートされている各システム構成で必要なインクルードディレクトリとライブラリが含まれます。「メークファイル・オプション」ダイアログから選択した構成によって、テンプレート内で使用する「ブロック」が決まります。システムには、常に、デフォルト構成が設定されているので、必ずしもこのダイアログから選択を行う必要はありません。詳細は、641 ページの「メークファイルの種類のリスト」を参照してください。一般的に使用する「ブロック」を以下に示します。

```
##: system So##: system Solaris 32bit Ansi C/C++
##: default cpp
#UILFLAGS=-I${MOTIFHOME}/include/uil
#MRMLIBS=-L${MOTIF_LIB_DIR} -lMrm
#CPPFLAGS=-Ddrem=remainder -DS_SUNOS5
#MOTIFHOME=/usr/dt
#MOTIFINCLUDES=-I${MOTIFHOME}/include
#MOTIF_LIB_DIR=${MOTIFHOME}/lib${SYSDIR} -R${MOTIFHOME}/lib${SYSDIR}
```

```
#XINCLUDES=${MOTIFINCLUDES} -I/usr/openwin/include -I/usr/openwin/
include/X11
#X11_LIB_DIR=/usr/openwin/lib${SYSDIR} -R/usr/openwin/lib${SYSDIR}
#XSYSLIBS=-L${X11_LIB_DIR} -lXt -lX11 -lXext -lnsl -lsocket -lgen
#MOTIFLIB=-L${MOTIF_LIB_DIR} -lXm
#CC=cc
#CCC=CC
#ABI2CFLAGS=
#ABI2CCFLAGS=-compat=5
#ABI2LDFLAGS=
#ABI2ABIDIR=/ansi32
#ABI2SYSDIR=
#ABICFLAGS=${ABI2CFLAGS}
#ABICCFLAGS=${ABI2CCFLAGS}
#ABILDFLAGS=${ABI2LDFLAGS}
#SYSDIR=
#ABIDIR=/ansi32
##: end
```

XINCLUDES および XLIBS は、それぞれインクルードするディレクトリとライブラリのパスを示す拡張子です。

第20章

複雑な配置

はじめに

本章では、複雑なウィジェット配置について説明します。

ローカラムを使用した列配置

ダイアログ中の要素は、多くの場合列または行に沿って配列されます。単一の列だけを必要としている場合には簡単ですが、複数の列を作成する場合はより多くの作業が必要になります。

単一の列配置を作成する最も簡単な方法は、フォームの代わりにローカラム (RowColumn) ウィジェットを使用するものです。ローカラムは、ほとんどのウィジェットを子に持つことが可能であり、また、異なるウィジェット型を同じローカラムの子にすることもできます。 ローカラムの配置方向が垂直の場合は単一の列が作成され、水平方向の場合は単一の行が作成されます。それぞれの結果を図 20-1 に示します。

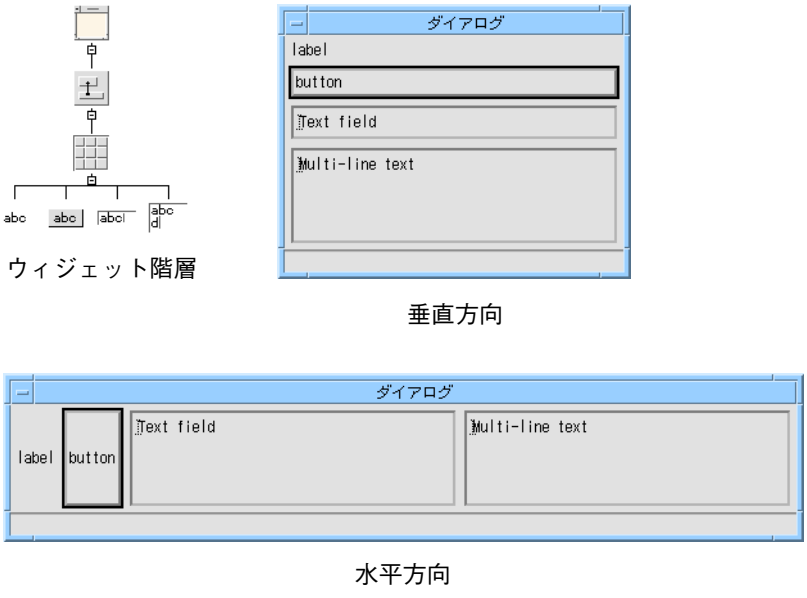


図 20-1 単一の列配置

図 20-1 は、間隔設定方法リソースが「密」に設定されている場合の、ローカラムウィジェットのデフォルト動作を示したものです。「密」な配置では、常に 1 つの列または行が作成されます。列配置の場合では、すべてのウィジェットは異なる高さを持つことができますが、同じ幅に強制されます。行配置の場合では、すべてのウィジェットは異なる幅を持つことができますが、同じ高さに強制されます。

間隔設定方法を「列」に設定すると、図 20-2 に示すように、すべてのウィジェットの幅と高さは同じになります。

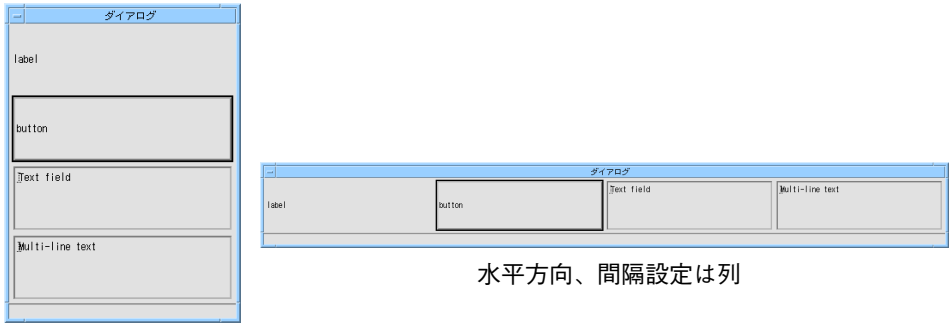


図 20-2 列配置

ローカラムのサイズ変更動作

ダイアログはサイズ変更が可能であるため、サイズ変更の際の構成要素の動作は常に重要です。サイズ変更に対しての一般的な記述がある資料は存在しません。したがって、実際に操作を行いながらサイズ変更の一般的な規則を見つけていく必要があります。 間隔設定方法が「密」に設定されている垂直ローカラムは、図 20-3 に示すように動作します。

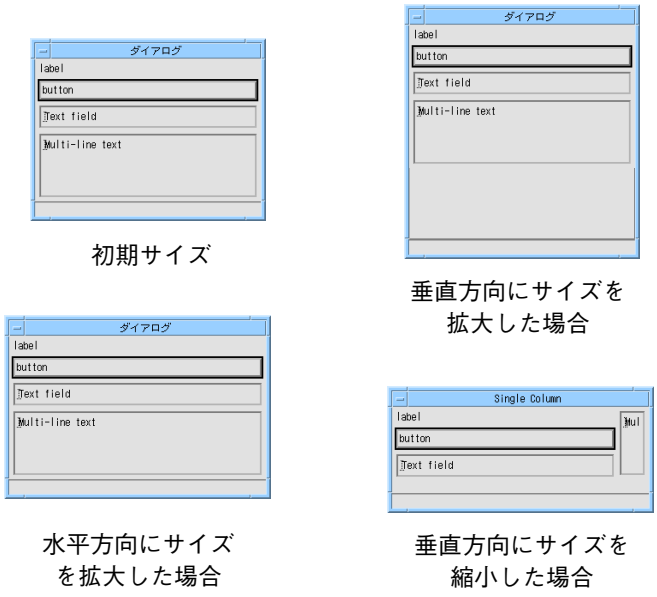


図 20-3 ローカラムのサイズ変更

可能な場合には、ローカラムウィジェット内のすべての子が表示されます。すべての子を表示できるほどローカラムが大きくない場合には、収まらない子は表示されません。特殊な場合を除いては、子ウィジェットが部分的にしか表示されないということはありません。

複数の列

ローカラムウィジェットを使用して、図 20-4 に示すように、ウィジェットを複数の列に配置することができます。

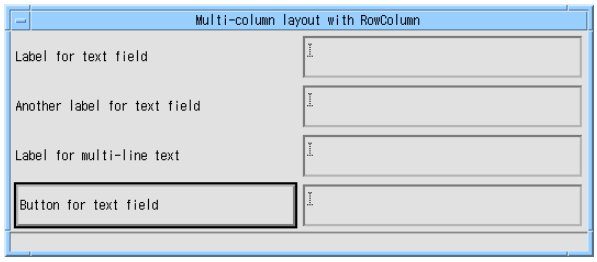


図 20-4 ローカラムウィジェットを用いた複数列の配置

ただし、この配置には制限があります。複数の列を使用するために、間隔設定方法を「列」に設定しなければなりません。この設定を行うと、すべての子は強制的に同じサイズにされます。この場合、テキストボックスの 1 つが 2 行分の高さであるため、すべてのテキストボックスはそのサイズと同じでなければなりません。その結果、空間が無駄に使用されるうえ、ユーザーを混乱させることにもなりかねません。図 20-5 に、改良を加えた配置を示します。

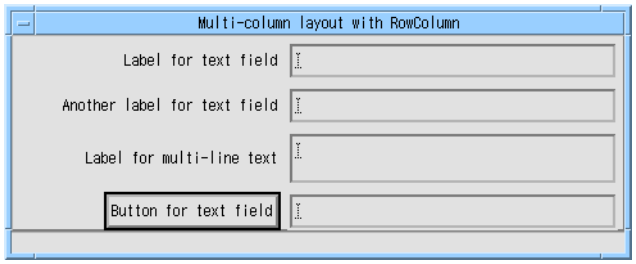


図 20-5 フォームウィジェットを使用した複数列の配置

この例は、各行の高さが異なるため、ローカラムを使用することはできません。このような場合には、フォームを使用してください。

フォームを使用した列配置

ここでは、複雑な列配置を作成するための手順を説明します。最初の例は、ラベルとテキストフィールドウィジェットを 1 つずつ含んでいる、1 行 2 列の配置です。特に記載のない限り、本章で使用されるすべてのアタッチメントのオフセットはゼロとします。

初めてフォームを作成し、その子を追加すると、Motif はフォームの左側に各ウィジェットの左側を接続する形で、子を下方向に配置します。最初のウィジェットの上部は、フォームの上部に接続されており、後に続く各ウィジェットの上部は上のウィジェットの下部に接続されます。したがって、ラベルおよびテキストフィールドを含むフォームを作成すると、図 20-6 のように配置されます。

この例で使用されるフォームでは、水平および垂直間隔が 5 ピクセルに設定されています。

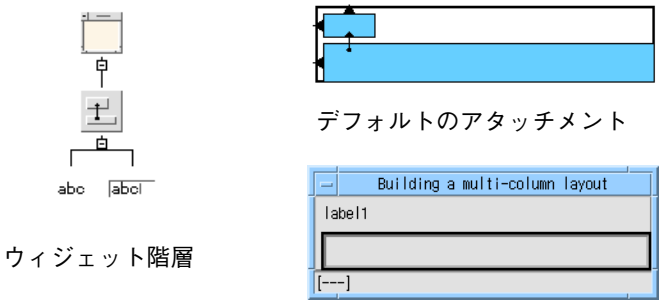


図 20-6 複数列の配置の構築

図 20-7 から図 20-11 に、この配置を 2 列配置に変更するための手順を示します。

まず、 テキストフィールドを図 20-7 に示すような位置に移動します。

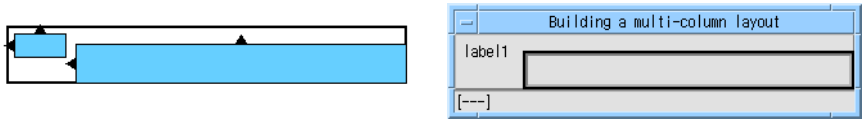


図 20-7 テキストフィールドウィジェットの配置

次に、図 20-8 に示すように、ラベルの上下とテキストフィールドの上下を揃えます。

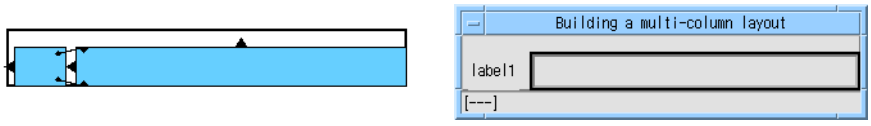


図 20-8 ウィジェットの整列

図 20-9 に示すように、テキストフィールドの上および右側をフォームの上および右側に接続します。

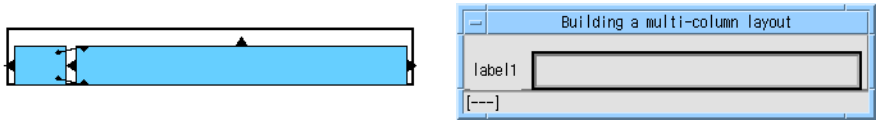


図 20-9 テキストフィールドのフォームへの接続

図 20-10 に示すように、ラベルの右側とテキストフィールドの左側を接続します。

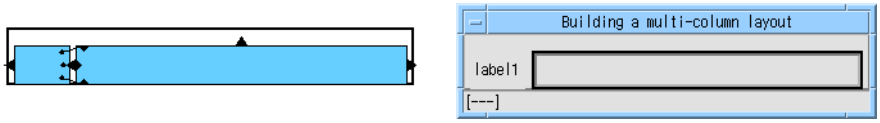


図 20-10 ラベルのテキストフィールドへの接続

最後に、図 20-11 に示すように、テキストフィールドの左側の位置を 25 % に設定します。

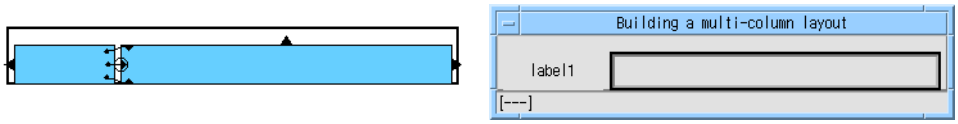


図 20-11 テキストフィールドの位置の設定

ラベルの右側がテキストフィールドの左側に接続されているにも関わらず、位置を設定してテキストフィールドの左側を固定することは不自然に思われるかもしれません。むしろ、テキストフィールドの左側を固定して、ラベルの右側へ接続する方が自然に思えるでしょう。しかし、この方法では、 フォームでは認められない循環アタッチメントが生じることになってしまいます。位置に使用された 25 % という値は、この段階では任意の値です。全ウィジェットの幅が確定するまでは、最終の位置を指定することはできません。

複数の行

上記の手順を複数の行に応用するには、単に同じ手順を繰り返してください。ただし、最初の行の左端および右端の位置 (ラベルの左側とテキストフィールドの右側) は、フォームの両側へのアタッチメントによって設定されていますが、後に続く各行の左端と右端の位置は、上にある行にウィジェットを揃えることによって設定されている、ということが異なります。

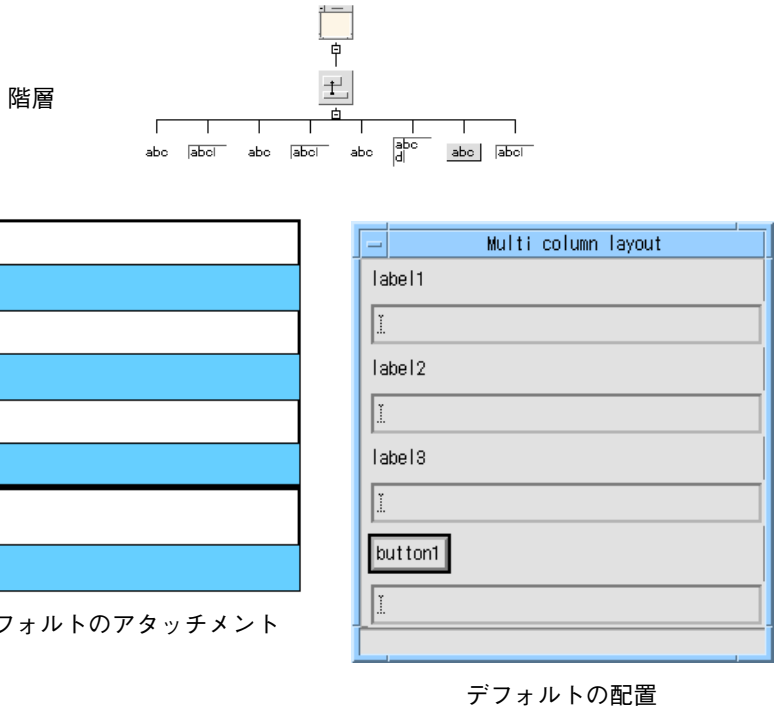


図 20-12 複数列の配置 - 初期状態

図 20-12 は、図 20-5 で示されているダイアログの初期状態を示します。図 20-5 に示すダイアログでは、左の列は 3 個のラベルと 1 個のプッシュボタンで構成されており、右の列は 3 個のテキストフィールドと 1 個のテキストウィジェットで構成されています。

まず、ダイアログを構成するウィジェットにリソースを設定します。つまりラベルおよびプッシュボタンにラベルテキストを、複数行テキストウィジェットに行数および編集モードを設定することができます。

これらのリソースは、この段階で設定しなくてもかまいません。後で設定することもできます。しかし、この段階でリソースを設定すると、ダイアログが表示される方法、および、意図したとおりの動作が行われるかどうかを早く確認することができます。

リソースを設定し、図 20-13 に示す配置を作成します。

図 20-13 リソースが設定されている複数列の配置

ここで、図 20-7 から図 20-11 に示した手順を列の各行に対して適用します。まず、テキストとテキストフィールドウィジェットを図 20-14 に示すような位置に移動します。

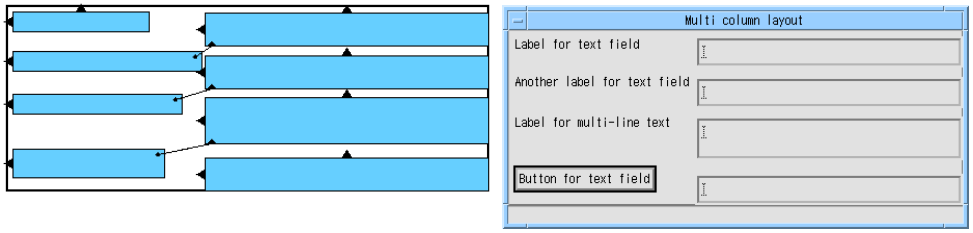


図 20-14 おおまかな位置

次に、図 20-15 に示すように、ラベルとプッシュボタンの上下を、対応するテキストまたはテキストフィールドウィジェットの上下に揃えます。

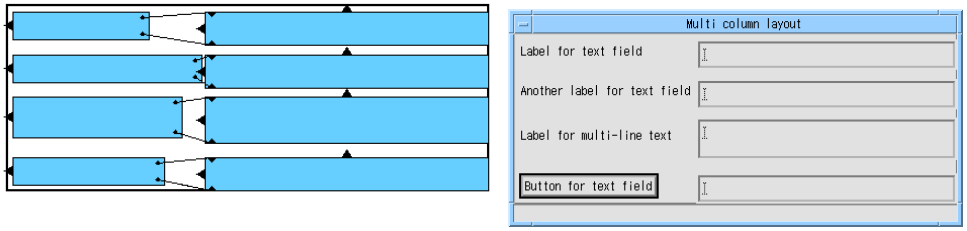


図 20-15 ラベルとテキストウィジェットの整列

図 20-9 に示すように、5 ピクセルのオフセットを使用して、最初の行のテキストフィールドウィジェットをフォームの上および右側に接続します。その他の各テキストフィールドおよびテキストウィジェットの上部は、5 ピクセルのオフセットを使用して、上にあるウィジェットの下部に接続します。

次に、テキストウィジェットを一番上のウィジェットに揃えます。右側の配置については、各テキストウィジェットの右側を上にあるウィジェットの右側に揃えます (または、0 ピクセルのオフセットを使用して接続します)。あるいは、下から上にテキストウィジェットを選択して「グループ整列方法」を使用します。左側に対しては、50 % などの任意の位置を設定します。

次に、各ラベルとプッシュボタンの左側を、上のウィジェットの左側に揃えます。下から上にウィジェットを選択して「グループ整列方法」を使用することができます。この時点での配置は、図 20-16 のようになります。

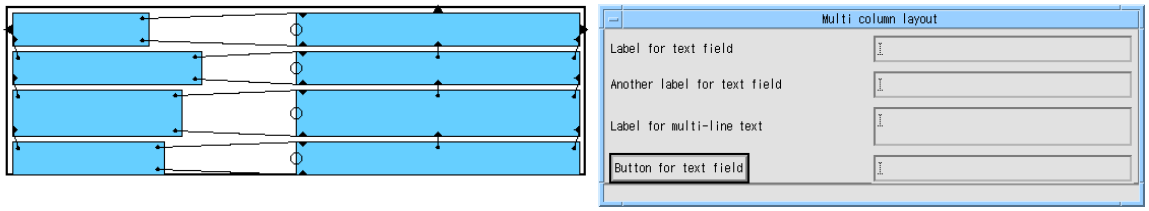


図 20-16 列方向の整列

図 20-17 に示すように、各ラベルおよびプッシュボタンの右側を、対応するテキストまたはテキストフィールドウィジェットの左側に接続します。

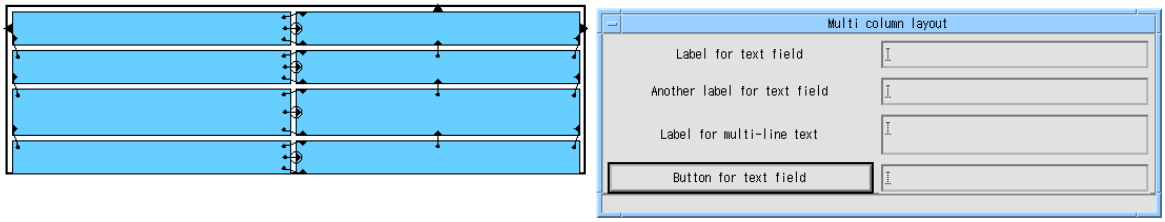


図 20-17 テキストとテキストフィールドウィジェットに接続されたラベル

最後の手順として、テキストウィジェットの左側の位置を調整します。これは、ある程度の試行錯誤を必要とします。パーセント (%) が大きすぎる、または小さすぎる場合には、フォームは必要以上に広くなり、画面スペースを無駄に使用することになります。図 20-18 にいくつかの例を示します。

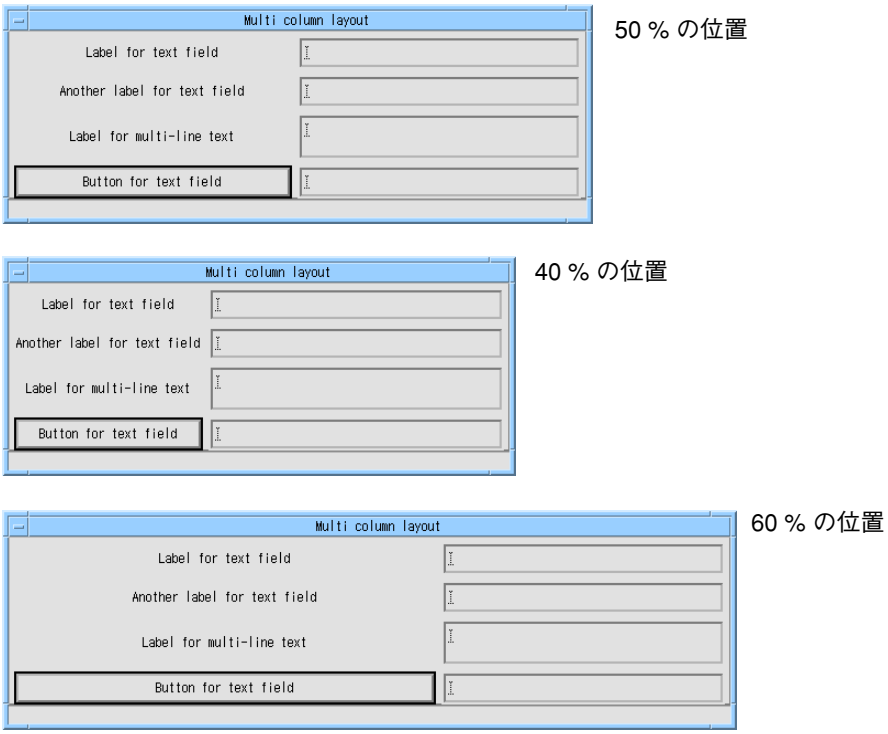


図 20-18 位置の設定

リセット

さまざまな位置の値を試す際には、値を変更するたびに (サイズ変更方針のリソース設定によっては) フォームが大きくなることに注意してください。これは、フォームが作成された後に変更された制約に対して、フォームが対応しきれないためです。アプリケーションで実際にどのようにフォームが表示されるかを確認するためには、フォームをリセットします。

複数列の配置に使用されるアタッチメント、整列方法、および位置の調整は複雑に見えるかもしれませんが、この調整方法は柔軟性が高く、変更も可能です。特に、ダイアログに対しての新しい行の追加は比較的簡単に行うことができます。

新しい行の追加

ダイアログの一番下に新しい行を追加するには、デザインに新しいウィジェットを追加し、上にある行と同様にアタッチメントを設定します。ただし、ダイアログの中程に新しい行を追加する場合は、作業がやや複雑になります。

ダイアログ中への行の追加

最初の手順として、新しい行のためのスペースを空けます。各行は 1 つ上の行にのみ接続されており、左の列のウィジェットは右の列のウィジェットの上下に接続されているため、右の列のウィジェットをドラッグするだけで、ダイアログの下部全体を移動させることができます。しかし、これはすべてのアタッチメントを切り離してしまうため、後で接続し直す必要があります。ただし、他のウィジェットから設定されたアタッチメントには影響はありません。図 20-19 に、テキストウィジェットを下へ移動した場合の結果を示します。

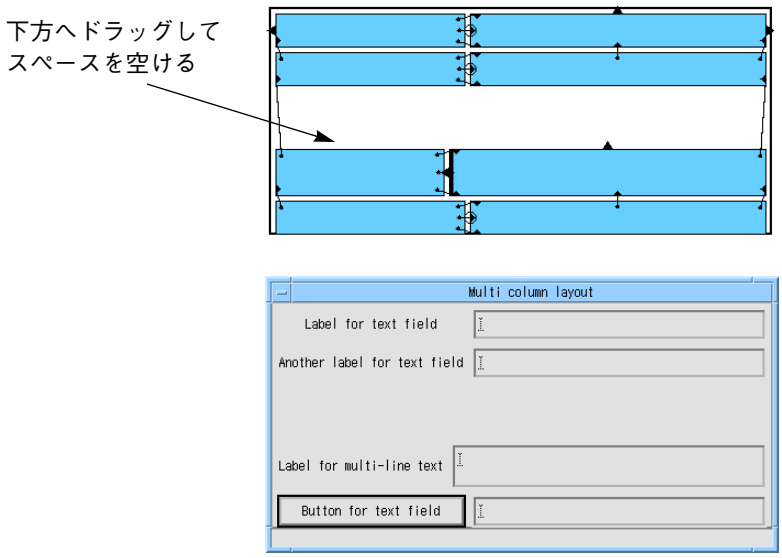


図 20-19 新しい行のためのスペースの作成

この時点で、新しい行のウィジェットを追加し、必要に応じてリソースを設定し、図 20-20 に示すように適当な位置に移動させることができます。

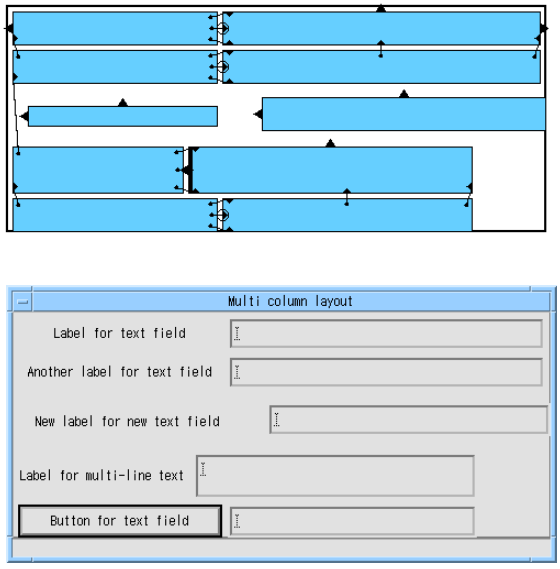


図 20-20 新しいウィジェットの追加

次に、図 20-21 に示すようにウィジェットが 2 つの列になるように位置を揃えます。

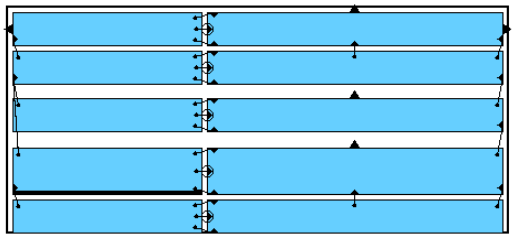


図 20-21 整列方法

最後に、各テキストまたはテキストフィールドを 1 つ上にあるものに接続し、図 20-22 のようにテキストまたはテキストフィールドウィジェットの左側の位置を設定します。

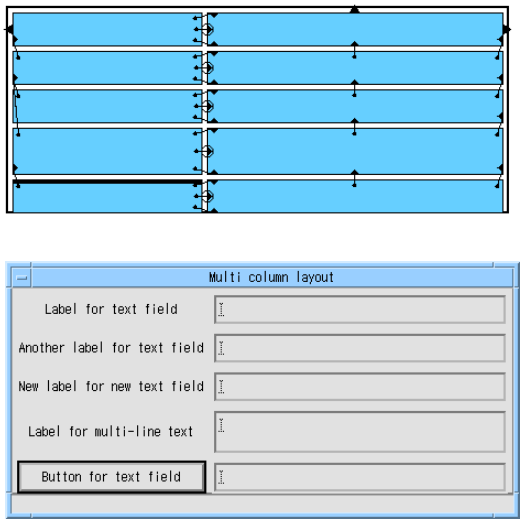


図 20-22 アタッチメント

新しいラベルは少し狭くなっているため、テキストウィジェットの左側の位置を 55% に設定してダイアログをリセットします。すると、図 20-23 に示されるような結果になります。

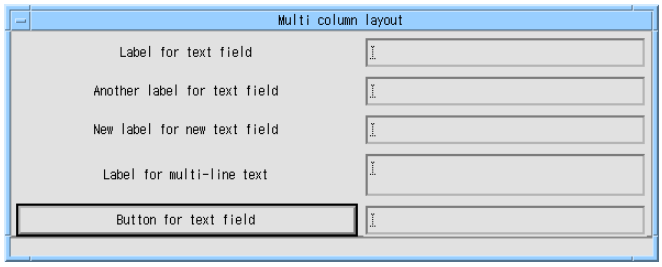


図 20-23 55% に設定された位置

4 列への変更

一部が 2 列からなるダイアログは、一般的によく使用されます。項目が多すぎて 2 列配置では列が高くなりすぎてしまう場合は、4 列配置に変更することができます。

たとえば、図 20-23 の例の 4 番目および 5 番目の行を、それぞれ別の 2 つの列 (3 と 4) に移動することができます。最初の手順では、図 20-24 に示すように、行とその上にある行を揃えるためのアタッチメントを切り離します。

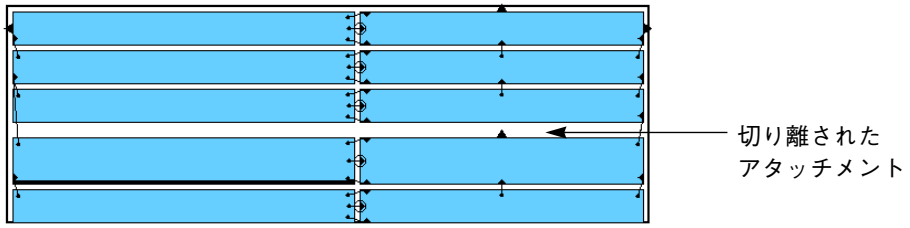


図 20-24 アタッチメントの切り離し

次に、新しい列のために右側のスペースを空ける必要があります。そこで、最終的にフォームでアタッチメントを持つ可能性がある最初の 3 行の位置を設定します。図 20-25 に、この結果を示します。位置の変更が行われると、フォームの幅が広くなるため、位置の変更後にはフォームをリセットする必要があります。

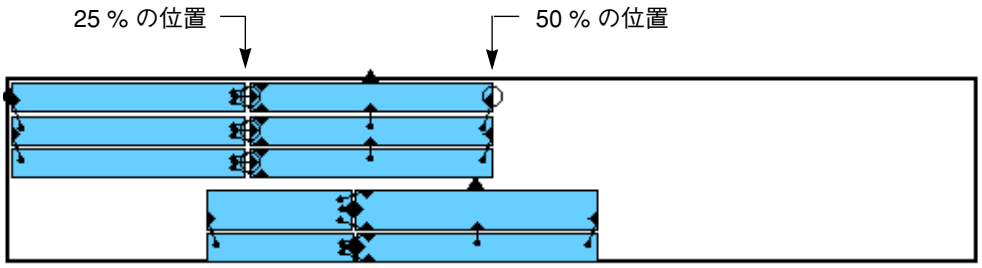


図 20-25 位置のリセット

同様に、図 20-26 のように下の 2 行の位置を設定します。フォームが一時的にも一貫性のない状態にならないように、順序通り位置の設定を行う必要があります。間違った順序で操作を実行した場合は、「Bailed out of synchronization」(端の同期の打ち切り) というメッセージが表示されます。このエラーメッセージは無視してかまいませんが、操作を続行する前にメッセージボックスを消去してください。また、位置の変更後にはフォームを忘れずにリセットしてください。

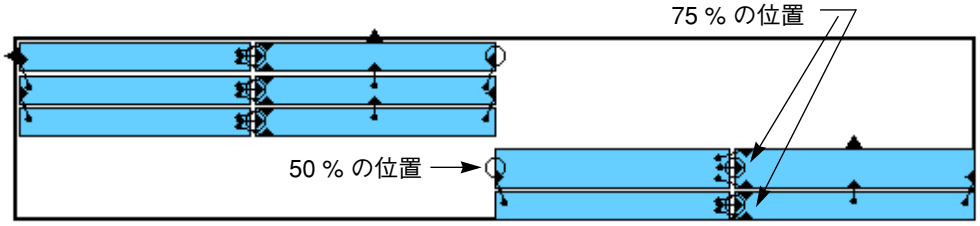


図 20-26 右側の列の位置決め

この時点で、一番上のテキストウィジェットをフォームの上および右側に接続して、右側の 2 列を正しい位置に移動することができます。結果は図 20-27 のようになります。

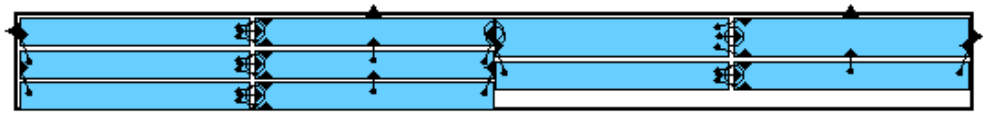


図 20-27 フォームの上および右側への接続

これで、配置はほぼ終了しました。列 2 の右側は、現在 50 % の位置にあります。この位置を、50 % の位置にある列 3 の左側へのアタッチメントに置き換えます。図 20-28 に、最終的な配置を示します。

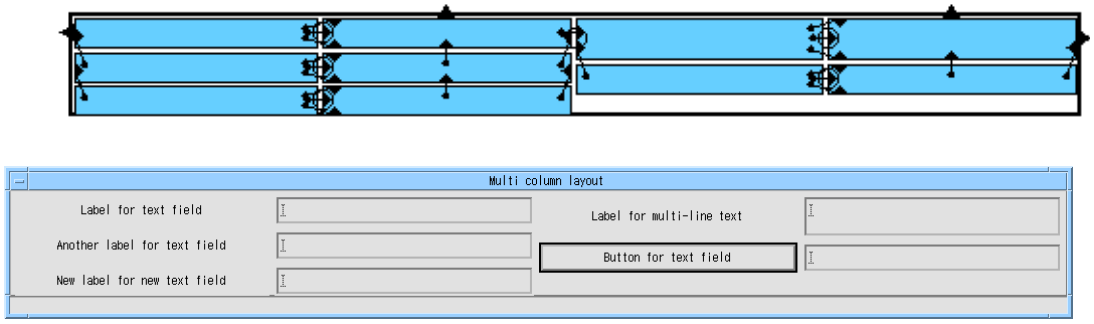


図 20-28 最終的な配置

端の問題

フォームはシェルの子なので、内側の端周辺にマージンラインを描きます。しかし、フォームの端に沿って拡張するフォームの子ウィジェットによって、このマージンラインの一部が隠されてしまいます (図 20-29)。

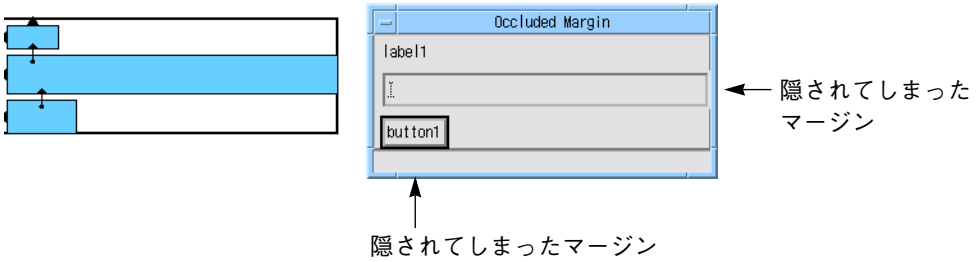


図 20-29 隠されてしまったマージン

この問題を解決するには、マージンに重なるウィジェットのオフセットを小さく設定して、フォームに接続するという方法が最も簡単です。しかし、この方法でも図 20-30 に示すようにサイズ変更動作に問題が生じる可能性があります。

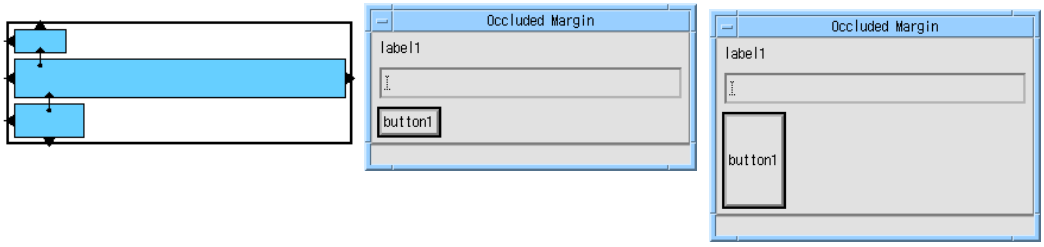


図 20-30 フォームへの追加アタッチメントとサイズ変更動作

この問題を解決するためには、2 通りの方法があります。これらは両方ともデザインに別のウィジェットを取り込む必要があります。

見えないウィジェット

図 20-30 に示す単純なアタッチメントの問題は、 フォームのサイズが変更されると同時にボタンのサイズ変更も行われるために、予想外の外観が生じてしまうことです。代替方法としては、目に見えないウィジェットを導入します。このウィジェットもまたフォームと一緒にサイズが変更されますが、目に見えないため違和感はありません。使用するウィジェットは、「種類」リソースを「線なし」に設定したセパレータガジェットが最も効果的です。見えないウィジェットを図 20-30 の例に追加したウィジェット階層およびアタッチメントを、図 20-31 に示します。

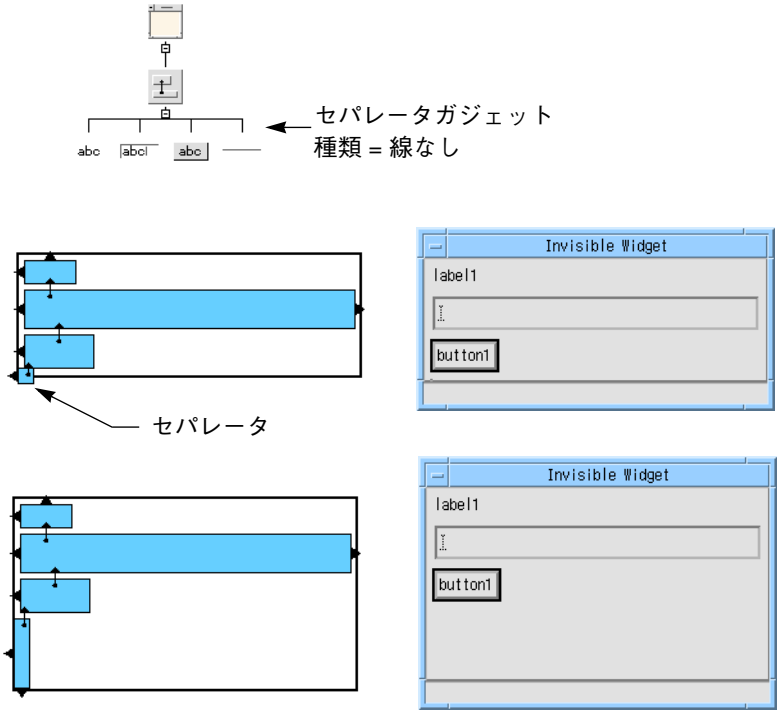


図 20-31 見えないウィジェット

セパレータの下部は、マージンラインを隠さないようにオフセットを小さくしたフォームの下部に接続されます。セパレータの上部は、オフセット 0 でプッシュボタンの下部に接続されます。セパレータはフォームと一緒に縦方向にサイズ変更されますが、他の構成要素のサイズは変わりません。

フォーム配置エディタは、セパレータ (および一定サイズ以下の他のウィジェット) の高さと幅を強調表示するため、マウスを使用してアタッチメントを設定することができます。これにより左下角のマージンが隠されているように見えますが、実際は隠されていません。

また、第 2 のセパレータを使用して、内部にあるテキストフィールドウィジェットの右側がフォームの右側を隠さないようにすることもできます (あるいは、両方に同じセパレータを使用することができます)。しかし、図 20-30 に示されているようなアタッチメントでの水平方向のサイズ変更動作は、ほとんどの場合に問題を発生させません (テキストフィールドウィジェットは、フォームと一緒に水平にサイズ変更します)。

二重フォーム

もうひとつの方法では、第 2 のフォームを見えないウィジェットとして使用します。この方法は、 フォームがシェルの直接の子である場合に限り、マージンラインを描くことを利用しています。シェルの子ではない中間のフォームはマージンラインを描かないため、その子を端まで拡張してもラインを隠してしまうという問題は発生しません。中間のフォームを使用すると、4 辺すべてについてマージンが隠れてしまうという問題を防ぐことができます。

図 20-32 は、この方法を用いた場合のウィジェット階層と適切なアタッチメントを示します。子フォームは、マージンが見えるように、オフセットを小さくして親フォームの 4 辺すべてに接続されています。

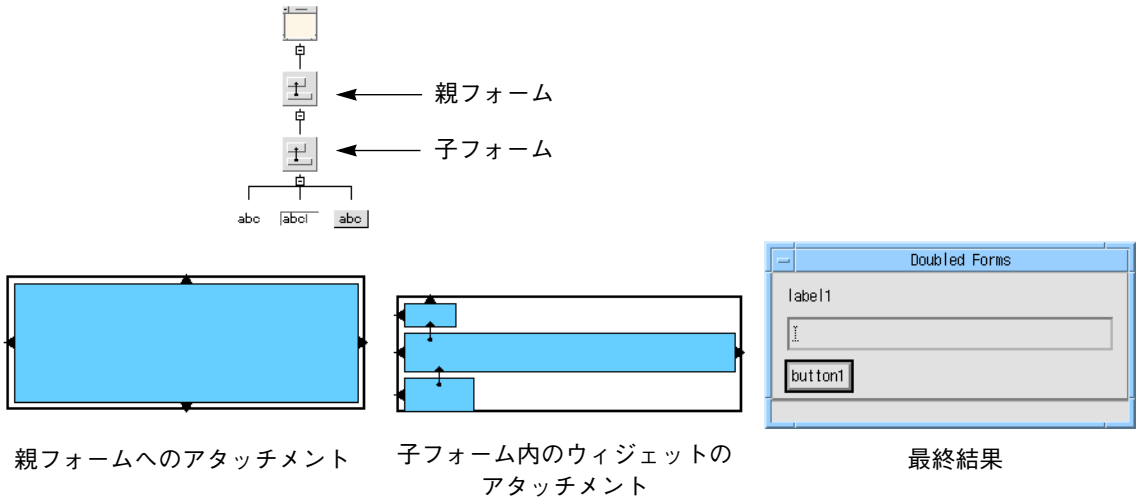


図 20-32 二重フォーム

マージンの幅と高さを 5 ピクセルなどの比較的小さな値に設定すると、親フォームの代わりとして、ブリテンボードを使用することができます。シェルはその子がブリテンボード (またはフォームなど、ブリテンボードの子孫) であると見なすため、この位置にフレームなどの異なる種類のコンテナウィジェットを使用しないでください。

フォームのサイズ変更

ユーザーがフォームをサイズ変更した際に引き起こされる動作は、アタッチメントと位置の設定の方法によって異なります。一般に、ダイアログは、ユーザーがそれを大きくすると、より多くの重要な情報が表示されるようにデザインします。たとえば、ダイアログがスクロールリストを含んでいる場合、ユーザーがウィンドウを高くすると、リストの可視部分が長くなるようにします。一方、ラベルやボタンなどのウィジェットは、別の情報を持っていないため、ウィンドウと一緒に大きくする必要はありません。

実際には、どのフォーム配置もサイズ変更動作、ウィジェット内でのサイズ変更 (フォント変更など) に対しての反応の信頼性、実装の容易さおよび保守の容易さとの間で妥協せざるをえません。本節では、望ましいサイズ変更動作を持つフォームを作成するためのガイドラインを示します。

ウィジェットのサイズ変更

フォーム内のウィジェットは、左右両側に制約がある場合には、フォームが横に広がると同時に横に広がります。また、 フォーム内のウィジェットの上下両端に制約がある場合には、 フォームが縦に長くなると内部のウィジェットも同時に縦に長くなります。

ダイアログを配置する最も簡単な方法は、フォームの左上角から右下角に向かって作業を進めることです。この場合、各ウィジェットの上および左側を、前のウィジェットの下および右側に接続します。この方法を実行すると、ウィジェットはフォームのサイズ変更時に、そのサイズの変更を行いません。フォーム内のウィジェットを的確にサイズ変更させるためには、少し高度な知識が必要です。

水平方向のサイズ変更

ローカラムウィジェットを使用した水平方向のサイズ変更の例は、657 ページの「ローカラムのサイズ変更動作」ですでに紹介してあります。本章の残りの部分では、その他の方法を実例を挙げて説明します。

2 個のウィジェットの配置

フォームの幅を 2 個だけのウィジェットが占めるように配置することは、比較的単純です。ここで必要とされるのは、フォームがサイズ変更された場合に生じる余分な幅をどちらのウィジェットに与えるか、あるいは両方のウィジェットで共有するかを指定することだけです。

図 20-33 に例を示します。 widget 1 は、その左側でフォームに接続されていますが、右側は自由です。そのため、本来の幅を持ち、テキストラベルの表示が可能です。widget 2 は、左側を widget 1 に、右側をフォームに接続されています。したがって、そのサイズは widget 1 が占有していないフォームの幅の部分を占めるように変化します。つまり、widget 2 は余ったスペースを使用します。

フォームをリセットする場合、 フォームは両方のウィジェットを収めるための独自の幅を設定し、図 20-33 に示すような初期状態を生成します。

図 20-33 2 個のウィジェット: 右優先

2 個のウィジェット間のアタッチメントは、図 20-34 では逆になっています。つまり、widget 1 の右側が widget 2 の左側に接続されています。この結果、widget 2 のサイズは変化せず、widget 1 が残りの幅をすべて得ることになります。

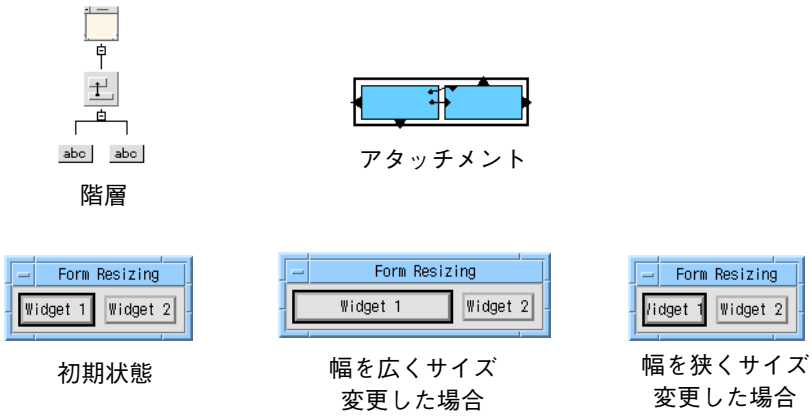


図 20-34 2 個のウィジェット: 左優先

アタッチメント反転時の循環の防止

2 個のウィジェット間の右から左へのアタッチメントは、同じ場所に左から右へのアタッチメントを新たに加えるだけで反転することができます。ユーザーが新しいアタッチメントを追加すると、配置エディタはこの状況を検出して古いアタッチメントを取り除きます。しかし、ここで示している例では、上記の動作を行うと循環エラーメッセージが表示されます。これは、変更されるべきアタッチメントが 2 つ存在するためです。循環アタッチメントを取り除くためには、2 個のウィジェットの上部を揃えているアタッチメントも入れ替える必要があります。両方のウィジェットとも同じ高さであるため、この入れ替えによって配置の外観が変化することはありません。

均等なスペース配分

余分なスペースを 2 個のウィジェット間で等分するためには、均等配置の使用が必要です。widget 1 に位置を設定して widget 2 に接続する、widget 2 に位置を設定して widget 1 に接続する、あるいは両方に位置を設定するという 3 通りの方法があります。これらの設定方法は、循環アタッチメントが起こらない限り有効です。図 20-35 に 3 通りの例を示します。

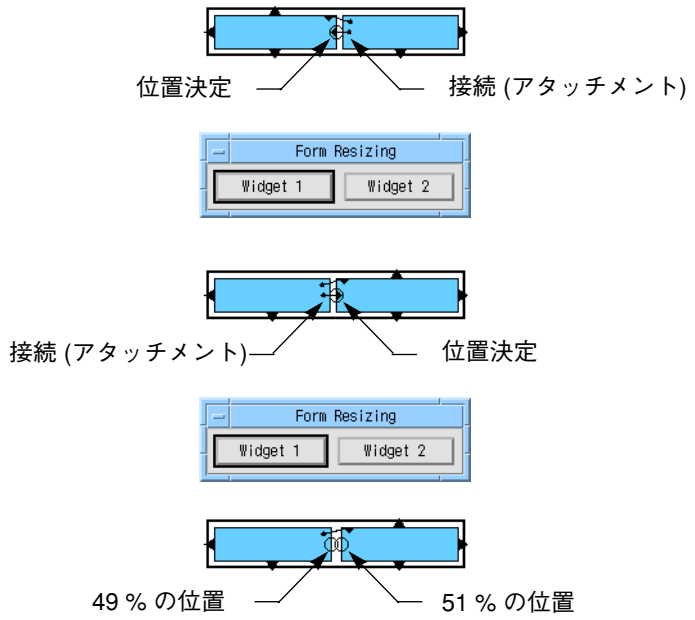


図 20-35 2 個のウィジェットによる等分配置

図 20-35 では、位置は約 50 % に設定されており、2 個のウィジェットはほぼ均等にフォームの幅を分割します。異なるパーセントを使用すると、どちらかのウィジェットのサイズを他方よりも大きくすることができます。この例の場合、余分なスペースはウィジェットの初期サイズに比例して分割されます。サイズ変更をして幅を小さくした場合のフォームの初期状態と動作は、すべてのクラスに共通しています。

3 個のウィジェットの配置

3 個のウィジェットを使用する場合には、配置方法も多くなります。余分なスペースを 3 個のウィジェットのどれか 1 つに与える、あるいは 3 個で分割することができます。図 20-36 は、余分なスペースすべてが 3 個のウィジェット中のひとつに与えられるという簡単な例を示しています。どの場合においても、両端が接続されているウィジェットが、フォームと一緒にサイズ変更を行うことに注意してください。

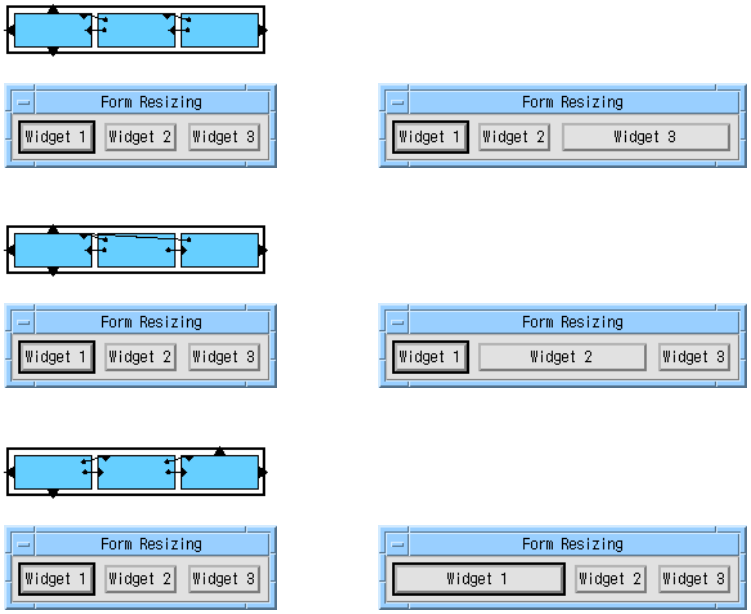


図 20-36 3 個のウィジェットのうち 1 個を優先

3 個のウィジェット間でスペースを分割する場合は、図 20-37 に示すように、単純な均等配置を使用することができます。

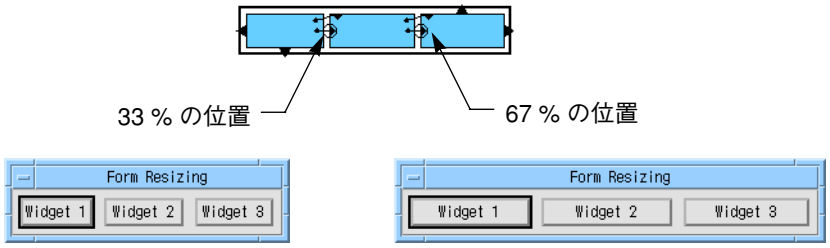


図 20-37 3 個のウィジェットによる均等分割

2 個のウィジェットの場合も、異なるパーセントを使用して、1 つのウィジェットを他のウィジェットよりも優先することができます。

アタッチメントと位置の組み合せを使用して、1 つのウィジェットのサイズを変更せずに他の 2 個のウィジェットのサイズを変更する、という配置を作成することができます。図 20-38 にいくつかの例を示します。

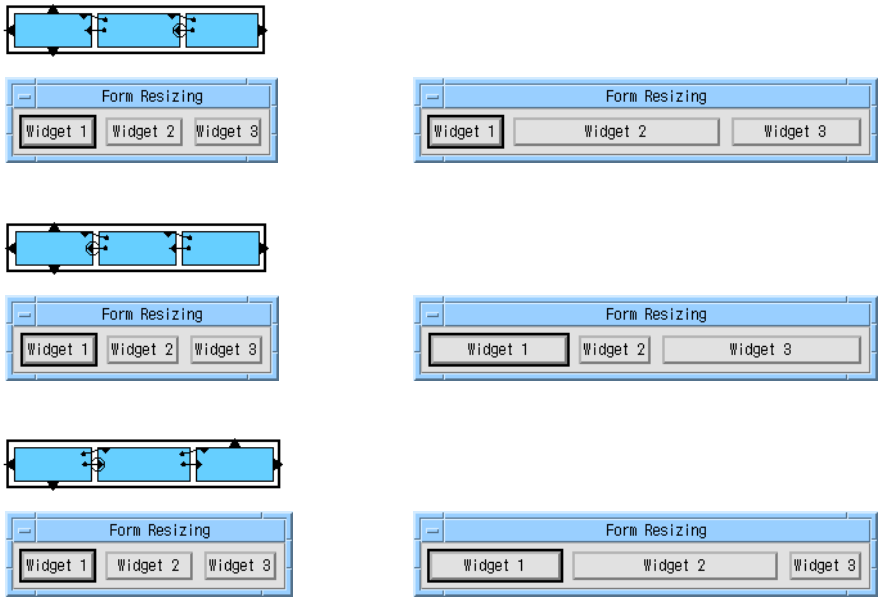


図 20-38 アタッチメントと位置の組み合せ

高さの異なるウィジェット

これまでのすべての例では、フォームの幅を分割するウィジェットの高さはすべて同じでした。このため、ほとんど制約なくウィジェットの上下におけるアタッチメントを調整し、容易に循環アタッチメントを回避することができました。しかし、ウィジェットの高さが異なる場合は、循環アタッチメントの回避はやや難しくなります。

図 20-39 に示すような階層において、図 20-33 の右優先配置と同様に、右のテキストウィジェットに利用可能スペースを最大限使用する配置を行うことにします。この場合、右のウィジェットを左のウィジェットに接続する必要があります。しかし、図 20-39 に示す例では、左側にあるラベルウィジェットがすでにテキストの上下に接続されることにより正しい垂直整列が生成されているため、このような接続は実行できません。

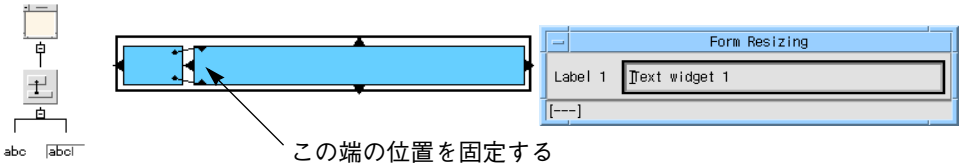


図 20-39 ラベルとテキストウィジェットを使用した 2 個のウィジェットの配置

この矛盾した状況を解決するには、以下の 3 通りの方法があります。

1. テキストウィジェットがラベルを覆い隠すことのないように十分に大きなオフセットを使用して、フォームの左側にテキストウィジェットの左側を接続します。ラベルの右側をテキストウィジェットの左側に接続します (図 20-40)。
2. テキストウィジェットの左側を指定したパーセントに設定し、ラベルの右側をその場所に接続します (図 20-41)。
3. 単一行の場合には、ラベルの上下の整列をフォームへのアタッチメントに置き換え、テキストウィジェットの左側をラベルの右側に接続することができます (図 20-42)。

それぞれの図には、これらの 3 通りの方法と、フォームがサイズ変更され、ラベル、フォントが変更される場合の動作を示します。

図 20-40 では、テキストウィジェットは両端でフォームに接続されます。この配列の長所は、 フォームの幅を広くした場合に生じる余分のスペースを、すべてテキストウィジェットが使用することです。これはユーザーがアプリケーションの構成を大幅に変更しない場合に限り、満足のいく動作が期待できます。ラベルまたはフォントが変更された場合には、正しく動作しません。

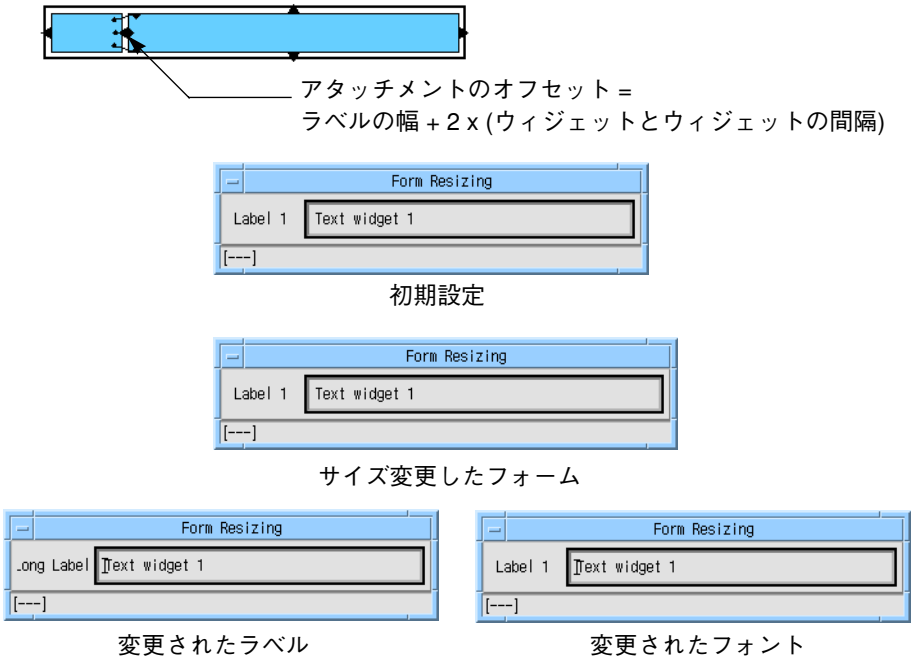


図 20-40 接続されたテキストウィジェット

図 20-41 では、テキストウィジェットの左端の位置が設定されています。これは、ラベルまたはフォントが変更される場合には、ラベルに余分なフォーム幅の一部が与えられるため、いっそう確実な配置となります。前述の列配置の場合には、この方法が最も有効です。

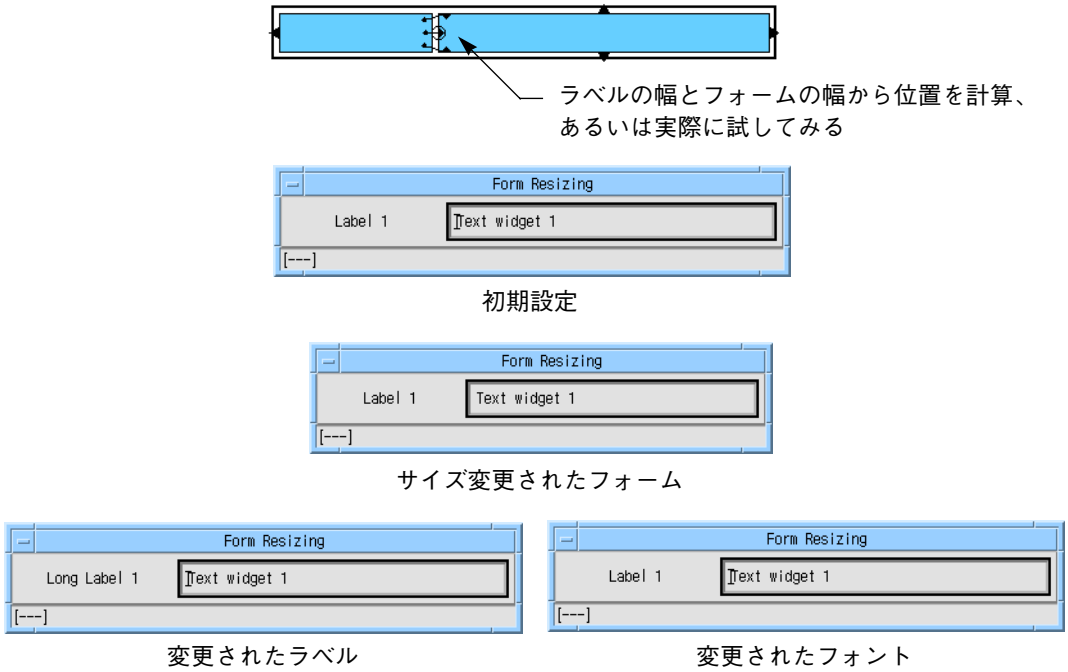


図 20-41 位置設定されたテキストウィジェット

図 20-42 では、ラベルの上下をテキストウィジェットの上下に接続する代わりに、フォームに接続することによって、循環アタッチメントの問題が解決されています。この後、テキストウィジェットの左側をラベルの右側に接続して、図 20-33 に示すような右優先の配置を作成することができます。

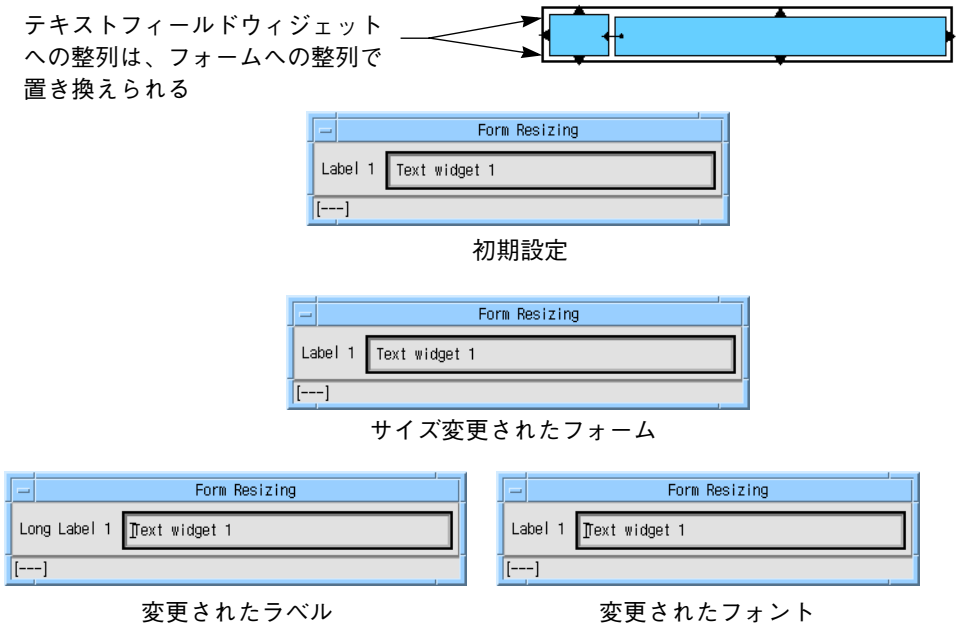


図 20-42 循環の除去

この方法を使用すると、最適な動作を実現することができます。しかし、この方法は各行を独立したフォームで囲む必要があると同時に、列を垂直に揃える手段がないため、列配置に使用することはできません。また、各行に対して余分なフォームウィジェットを必要とするため、かなりの量のオーバーヘッドが発生する可能性があります。

垂直サイズ変更

垂直のサイズ変更方法は、基本的には水平サイズ変更の場合と同じです。しかし、通常は垂直サイズ変更の方が循環アタッチメントの問題が少なくなっています。これは、オブジェクトの左側にあるラベルを、使用可能なスペースの中央に配置する必要があるのに対し、オブジェクトの上に位置されているラベルは、オブジェクトの左端に揃えることができるためです。図 20-43 にこの例を示します。

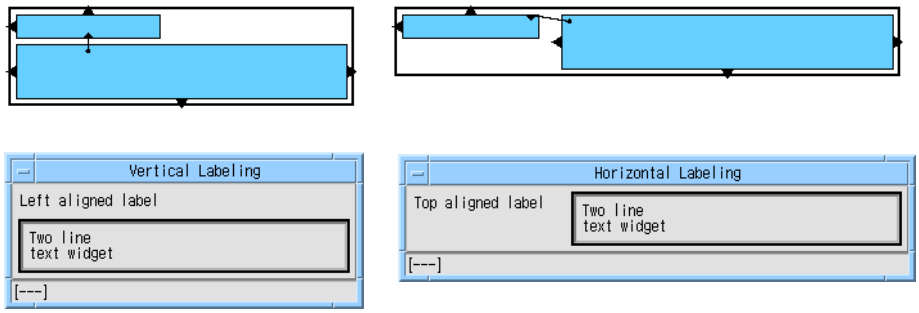


図 20-43 垂直および水平のラベル配置

図 20-44 から図 20-46 に、図 20-36 に示す水平配列に似ている垂直配置の例を示します。それぞれの例では、余分の高さは複数行のテキストウィジェットに与えられます。

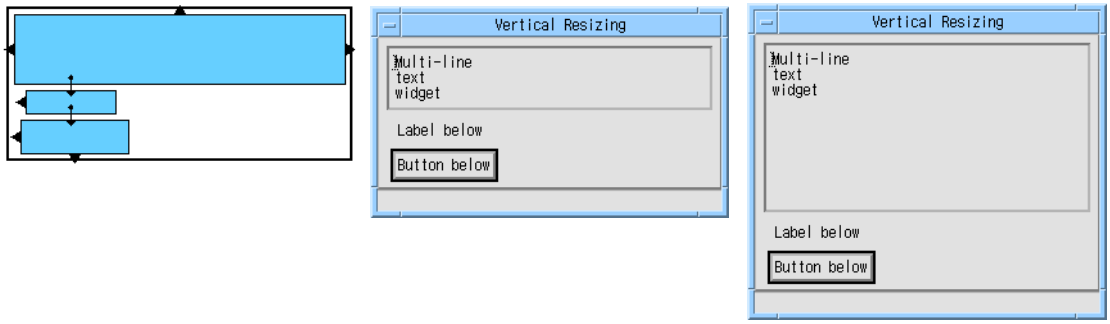


図 20-44 一番上のウィジェットのサイズ変更

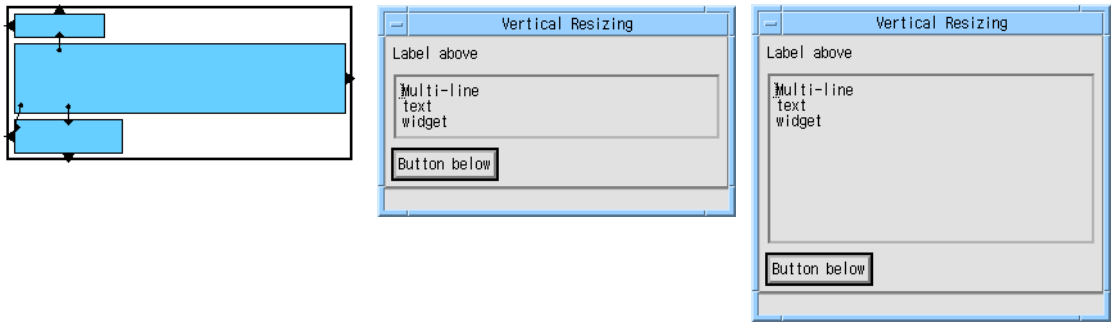


図 20-45 中央のウィジェットのサイズ変更

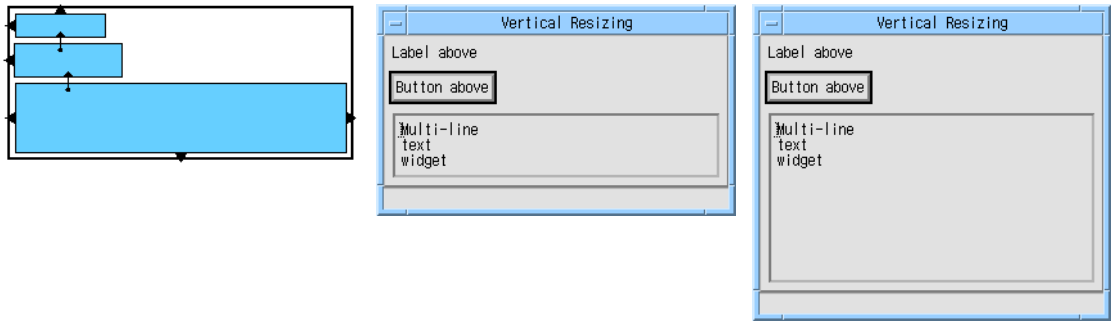


図 20-46 一番下のウィジェットのサイズ変更

初期サイズ

ダイアログの初期サイズは、シェル、フォーム、フォーム内のウィジェット間の交渉の結果によって決定されます。通常、フォームはその子についての制約をすべて満たす配置を見つけようとし、それぞれの要求をできるだけ「自然な」サイズで実現させようとします。その後、フォームはフォーム自身がその配置を含むようにサイズを決め、シェルはフォームまたはブリテンボードを含むようにそのサイズを決めます。

ほとんどのウィジェットには、合理的で自然なサイズというものがあります。たとえばラベルの自然なサイズは、その中に含まれるテキストとフォントによって決定されます。また、テキストウィジェットは通常、リソースで指定されている行および列の数に従ってウィジェット自身でサイズを決定します。

ダイアログに、許容できる大きさの自然サイズのウィジェットのみが含まれている場合は、フォームが初期サイズを決めることができます。ダイアログ中にあるウィジェットの自然サイズが大きすぎる場合は、制約を使用してサイズを固定する必要があります。その際、ダイアログの初期サイズがかなり小さく見える場合があります。

この問題が生じた場合、最上位フォームの幅と高さのリソースを設定してダイアログの初期サイズを設定します。適切なシェルリソースを使用して最小サイズを設定することはできますが、シェルの幅および高さのリソースを設定して初期サイズを設定することはできません。

第21章

ハイパーテキスト・ヘルプ

はじめに

Sun WorkShop Visual には、アプリケーションのさまざまな場所から呼び出すことができる、広範囲に渡るオンラインヘルプが用意されています。本章では、アプリケーションのヘルプ構築のために Sun WorkShop Visual が提供するサポートについて説明します。

ヘルプモデル

アプリケーションでのヘルプをサポートするために使用されているモデルは、非常に単純です。すべての Motif ウィジェットで使用できるヘルプコールバック、またはヘルプボタン上の活性化コールバックを使用することができます。したがって、文脈依存ヘルプを提供するために必要なものは、コールバックが呼び出された場合に表示されるヘルプ (あるいはヘルプへのパス) をクライアントデータとして使用する、汎用のコールバックです。Sun WorkShop Visual では、このヘルプ指定はドキュメントのパスおよびそのドキュメント内の参照個所を示すマーカーとして定義されます。Sun WorkShop Visual には、Sun WorkShop Visual のヘルプシステムのインタフェースで使用可能なコールバックが、1 つ用意されています。ヘルプシステムの詳細は、688 ページの「ヘルプビューア」を参照してください。

また、アプリケーションで使用するコールバックを再実装して、選択したヘルプシステムとインタフェースさせることについての制約などは一切ありません。さらに、通常のコールバックメカニズムを使用して同じ効果を実現することも、あるいはオンラインヘルプをまったく提供しないようにすることも自由に行うことができます。

Sun WorkShop Visual は、ヘルプを表示するために次の 2 つのシステムを使用しています。

- ✳◆■ ✶❑❒✷✳❄❑❒ ✶❄▲◆❁● ヘルプ
  項目間を行き来するための操作コマンドと、関連するヘルプ項目へ移動するためのリンク機能があります。

- Netscape
  HTML 形式のヘルプファイルを表示するブラウザとして使用します。

Sun WorkShop Visual でヘルプを参照するには、上記のいずれかを使用する必要があります。

Motif のヘルプコールバック

Motif のデフォルトでは、各ウィジェットに対して指定されているヘルプのアクションによってコールバックが呼び出されます。このアクションは、すべての Motif クラスのデフォルトのトランスレーションで *<osfHelp>* キーに割り当てられます。

ヘルプビューア

Sun WorkShop Visual では、以下のビューアを使用することができます。

- Sun WorkShop Visual ヘルプ
- Netscape
- FrameMaker

これらのビューアを使用して、アプリケーションのヘルプを表示することができます。詳細は、以降の節で説明します。

アプリケーションには、どのビューアを使用する場合でも同じようにヘルプを設定することができます。次に、そのアプリケーションを該当する供給ライブラリとリンクします。詳細は、703 ページの「ヘルプの実装」を参照してください。

Sun WorkShop Visual ヘルプ

図 21-1 は、Sun WorkShop Visual が使用するヘルプビューアです。

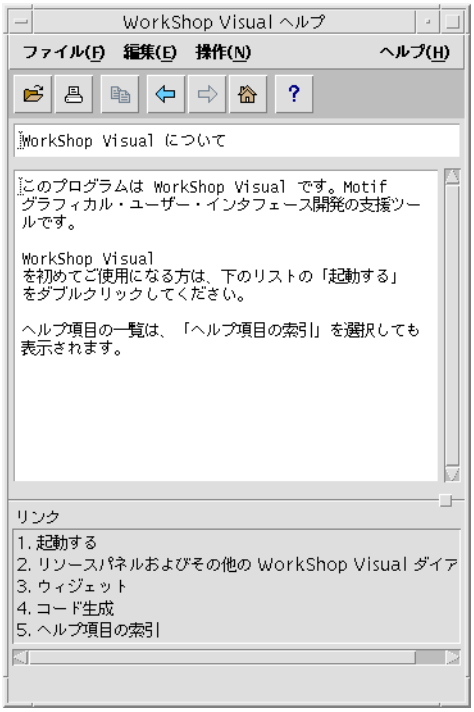


図 21-1 Sun WorkShop Visual ヘルプのヘルプビューア

Sun WorkShop Visual ヘルプのヘルプビューアには、「ファイル」、「編集」、「操作」、「ヘルプ」メニューを持つメニューバーがあり、それぞれファイル、編集およびヘルプ項目間を行き来するためのコマンドがあります。「ヘルプ」メニューでは、バージョン情報と Sun WorkShop Visual の使用方法に関するヘルプ情報を見ることができます。また、ツールバー上のボタンをクリックすると、大部分のメニュー項目に簡単にアクセスすることができます。ツールバーのボタン上でマウスのポインタを動かすと、ビューアウィンドウの最下部にあるステータス行に、該当するボタンの機能の説明が表示されます。

ヘルプビューアには、テキスト領域が 2 つあります。最上部にあるテキスト領域には、現在選択している項目についてのヘルプが表示されます。最下部にある領域には、関連項目へのリンクがあります。リンクの 1 つをダブルクリックすると、その項目のヘルプが表示されます。

Sun WorkShop Visual ヘルプが使用するファイルは、ハイパーテキストマークアップ言語 (HTML) 形式です。これはプリント可能文字のみを使用するパブリックドメイン形式なので、任意のテキストエディタで作成することができます。また、多くのアプリケーションが使用する標準でもあります。

Sun WorkShop Visual ヘルプは完全な HTML インタプリタではないため、HTML のサブセットを認識し、それらを単純な方法で解釈します。Sun WorkShop Visual ヘルプが認識する HTML キーワードとその解釈の方法については、691 ページの「HTML タグ」で説明します。

HTML の参考文献については、付録 E「参考資料」を参照してください。

Netscape

Netscape とは HTML ファイルを受け取って、それらのファイルに指定されたハイパーテキストリンクや特別な形式を含めて表示するブラウザです。

また、Netscape にはファイル編集および操作コマンドも用意されているため、ユーザーは、自分が作成したヘルプファイルもブラウズすることができます。

FrameMaker

FrameMaker は、独自の内部形式を使用するデスクトップパブリッシング用のパッケージです。また、FrameMaker では、読み取り専用ドキュメントを表示およびブラウズすることもできます。FrameMaker をヘルプビューアとして使用する方法についての詳細は、696 ページの「FrameMaker の使用」を参照してください。

ヘルプドキュメントでの HTML の使い方

Sun WorkShop Visual ヘルプと Netscape は、ともに HTML ファイルを読み取ります。リンクのソースおよび宛先となる位置に HTML アンカーを指定するように、ファイルを作成する必要があります。これらのアンカーの名前は、個々のウィジェットに対してヘルプアクションを指定する際に使用するので、覚えておくと便利です。

これらのアンカー位置は Sun WorkShop Visual では「マーカー」と言い、ヘルプコールバックで指定します。ヘルプコールバックのマーカーを設定する方法については、697 ページの「Sun WorkShop Visual でのヘルプの設定」で説明します。

Sun WorkShop Visual ヘルプまたは Netscape のいずれかをアプリケーションのヘルプシステムとして使用するには、以下の作業を行う必要があります。

1. HTML 形式でヘルプファイルを作成します。

   HTML では同じドキュメント内にリンクを持たせることができるので、項目ごとに個別のファイルを作成する必要はありません。ただし、ヘルプテキストを管理するには、個別のファイルにしておくことをお勧めします。Sun WorkShop Visual ヘルプは HTML のサブセットを使用します。Sun WorkShop Visual ヘルプが認識する HTML キーワードについては、691 ページの「HTML タグ」を参照してください。

2. 各ウィジェットまたはモジュール全体に対してヘルプコールバックを指定します。

   デザイン内のウィジェットにヘルプコールバックを追加する方法については、697 ページの「Sun WorkShop Visual でのヘルプの設定」を参照してください。ヘルプコールバックをアプリケーション内に定義するには、701 ページの「アプリケーションへのリンク」の指示に従う必要があります。

3. 生成されたコードを Sun WorkShop Visual ヘルプか Netscape で提供されているライブラリにリンクします。

   703 ページの「ヘルプの実装」を参照してください。

HTML タグ

本節では、タグとして知られているおもな HTML キーワードについて簡単に説明します。ここに記載されるタグは、すべて Sun WorkShop Visual ヘルプシステムによって解釈されます。ここでは、大部分の完全な HTML インタプリタ (Netscape など) によるタグの解釈方法を示し、Sun WorkShop Visual ヘルプによるタグの解釈方法と異なる場合は、その解釈方法も示します。

HTML では、タグは山括弧 (< と >) で囲みます。テキストファイルでは、これらの括弧を使用する必要があります。ドキュメント内での HTML タグの使い方については、695 ページの「HTML ドキュメントの例」を参照してください。

タグはその使用法に応じて分類されています。ここでは、タグは大文字で記載されていることに注意してください。ただし、HTML は大文字小文字を区別する言語ではないので、大文字も小文字も使用できます。

開始と終了

<HTML>
すべての HTML ドキュメントは、このタグで開始します。

</HTML>
HTML ドキュメントは、このタグで終了します。

アンカーとリンク

<A NAME="アンカー名">テキスト</A>
これは、リンクの宛先として使用されるアンカーです。「アンカー名」は、内部で使用するアンカーの名前です。「テキスト」には、ユーザーにクリックさせたいテキストを指定します。このテキストをクリックすると、リンク先へ移動することができます。

<A NAME="#アンカー名">テキスト</A>
これは、同じドキュメントの別の場所へのリンクのソースです。「アンカー名」は、アンカーに対して内部で付けられた名前です。

相対パス名か絶対パス名を使用して、別ファイルにリンクさせることができます。外部ファイルは別サーバーに常駐させることもできます。外部リンクの場合は、「アンカー名」がファイル名となり、先頭にある '#' 記号を省略します。「テキスト」は、ユーザーにクリックさせたいテキストです。

パラグラフ

<P>
このタグは、パラグラフの始まりを示すために使われます。パラグラフは、空の行で区切られます。

</P>
このタグは、パラグラフの終りを示すために使われます

ブレーク

このタグは、改行を示します。改行がなければ、テキストはブラウザウィンドウの幅に合わせて流し込まれます。

リスト

Sun WorkShop Visual ヘルプは、大部分の HTML インタプリタが行うようには、さまざまなリストを処理することができません。リストは互換用に解釈されるので、既存の HTML ドキュメントを使用することができます。

<OL>

以降の行が番号付きリストであることを示します。大部分の HTML 実装では、リスト項目の前に番号が付きますが、Sun WorkShop Visual ヘルプでは '-' (ダッシュ) だけが出力されます。

</OL>

番号付きリストの終了を示します。

<UL>

以降の行が番号なしのリストであることを示します。大部分の HTML 実装では、リスト項目の前にドットマークが付きますが、Sun WorkShop Visual ヘルプでは '-' (ダッシュ) だけが出力されます。

</UL>

番号なしリストの終了を示します。

<DL>

以降の行が定義リストであることを示します。大部分の HTML 実装では、用語集のように「用語」と「定義」があると見なしますが、Sun WorkShop Visual ヘルプでは、リストの各項目の前に '-' (ダッシュ) だけが出力されます。

</DL>

定義リストの終了を示します。

<LI>

リスト項目を示します。行の先頭に '-' (ダッシュ) が出力されます。

<DT>

定義リストの「用語」を示します。Sun WorkShop Visual ヘルプでは、これらの用語を通常のリスト項目と見なします。

<DD>

定義リストの「定義」を示します。Sun WorkShop Visual ヘルプでは、これらの定義を '-' (ダッシュ) の付かないインデントされたリスト項目と見なします。

文字形式

<EM>
以降のテキストを強調表示するように指定します。HTML インタプリタには、イタリックを使用するものも、ボールドを使用するものもあります。Sun WorkShop Visual ヘルプでは、該当するテキストの前後に '*' (アスタリスク) を出力します。

</EM>
テキストの強調表示を終了します。

<B>
以降のテキストをボールドで出力するよう指定します。Sun WorkShop Visual ヘルプでは、該当するテキストの前後に '*' (アスタリスク) を出力します。

</B>
テキストのボールドでの出力を終了します。

<I>
以降のテキストをイタリックで出力するよう指定します。Sun WorkShop Visual ヘルプでは、該当するテキストの前後に '*' (アスタリスク) を出力します。

</I>
テキストのイタリックでの出力を終了します。

見出し

HTML は、見出しを「H1」から「H6」までの範囲で定義します。この見出しの範囲は、大部分の HTML インタプリタに従って、大きいボールドの見出しで始まり、徐々にサイズと太さが小さくなっていきます。Sun WorkShop Visual ヘルプでは、見出しの前に空白行を 2 行出力するだけです。以下に示す見出しタグは、見出しテキストの開始と終了を示します。

<H1>最初のレベルの見出し</H1>

<H2>2 番目のレベルの見出し</H2>

<H3>3 番目のレベルの見出し</H3>

<H4>4 番目のレベルの見出し</H4>

<H5>5 番目のレベルの見出し</H5>

<H6>6 番目のレベルの見出し</H6>

事前に形式設定されたテキスト

<PRE>
大部分の HTML インタプリタは、余白、タブまたは復帰行を無視して、HTML ドキュメントのテキストを形式設定します。事前に形式設定されたタグにより、インタプリタはクーリエなどのモノスペースフォントを使用して、テキストをユーザーが入力したとおりに出力します。Sun WorkShop Visual ヘルプは、出力関数がスペースを削除しないようにするだけです。

</PRE>
事前に形式設定したテキストの終了です。

HTML ドキュメントの例

Sun WorkShop Visual ヘルプ用に HTML で記述したヘルプドキュメントの例を以下に示します。

```
<html>

<head>

Sun WorkShop Visual について

</head>

<h1>Sun WorkShop Visual について </h1>

<p>このプログラムは Sun WorkShop Visual です。Motif グラフィカル・ユーザー・インタフェース開発の支援ツールです。

<p>

Sun WorkShop Visual を初めてご使用になる方は、下のリストの「起動する」をダブルクリックしてください。

<p>
ヘルプ項目の一覧は、「ヘルプ項目の索引」を選択しても表示されます。


関連項目:

<ul>

  <li><a href="get_started.html">起動する</a><br>

  <li><a href="dialogs.html">リソースパネルおよびその他の Sun WorkShop Visual ダイアログ </a><br>

  <li><a href="widget_list.html">ウィジェット</a><br>
```

```
  <li><a href="code_gen.html">コード生成</a><br>
  <li><a href="index.html">ヘルプ項目の索引</a><br>
</ul>
</html>
```

このファイルは、図 21-1 の Sun WorkShop Visual ヘルプに表示されています。

FrameMaker の使用

FrameMaker を使用してヘルプシステムを構築する場合は、FrameMaker ハイパーテキストシステムの動作についての知識が必要です。 詳細は、FrameMaker のマニュアルを参照してください。以下に、概要を説明します。

FrameMaker を使用すると、テキスト内の場所をソースマーカー、あるいは宛て先マーカーとしてマークすることができます。宛て先マーカーは、ジャンプして移動できるヘルプドキュメント内の場所を指します。ソースマーカーとは、ユーザーが選択してアクション (通常は宛て先マーカーへのジャンプ移動) を起こすことができるヘルプドキュメントまたはアプリケーション内の場所を指します。ヘルプドキュメント中のソースマーカーには、ユーザーにクリック可能であることを知らせるための視覚的な目印が必要です。斜体や下線付けなどの文字形式を指定して、ソースリンクを示すことが最も効果的です。

ドキュメント内の特殊文字形式が設定されている場所をユーザーがクリックすると、FrameMaker はその領域のソースマーカーを調べます。ソースマーカーが見つかった場合には、グラフィック領域全体またはテキストの部分を強調表示して、マーカーで指定されているアクションを実行します。

これらの特殊マーカーは、「スペシャル」メニューの「マーカー」コマンドを使用して、FrameMaker のドキュメントに挿入することができます。

1. 「マーカータイプ」リストから「Hypertext」を選択します。
2. 宛て先マーカーを指定する場合には、マーカーテキストフィールドに次のように入力します。

   newlink <タグ名**>**
3. ソースマーカーを指定する場合には、次のように入力します。

   gotolink <タグ名**>**

注 - 現在のドキュメントにおけるタグは <タグ名>、別のドキュメントにおけるタグは <ドキュメント名>:<タグ名> として指定します。

---

4. Hypertext マーカーは、ロックされているドキュメントに限って有効です。ドキュメントのロックをオンあるいはオフにするためには、以下のように入力します。

   <escape> <F> <l> <k>

5. ロックされたドキュメントを終了するには、次のように入力します。

   <escape> <f> <c>

   このコマンドにより、ドキュメントをロックした状態で保存することができるようになります。

Sun WorkShop Visual でのヘルプの設定

タグという用語は、ドキュメントとマーカーの組み合せを意味します。ヘルプタグは、コアリソースパネルの「コード生成」ページで指定されます。指定できるヘルプタグには、任意の Motif ウィジェットに対して指定できるヘルプ・コールバックと、プッシュボタンおよびカスケードボタンウィジェットに対してのみ指定できる活性化コールバックの 2 種類があります 。

注 - カスケードボタンの活性化コールバックは、カスケードボタンに関連するプルダウンメニューが存在しない場合にのみ読み出されます。

タグを指定するためには、ドキュメントの名前とマーカーの名前を入力し、「適用」を押します。この時点で、活性化コールバックに対するボタンをクリック、 あるいはヘルプコールバックに対する *<F1>* を押してコールバックを呼び出して、ドキュメントをダイナミックディスプレイに表示することができます。

コアリソースパネルにおいてタグ情報を指定する場合、Sun WorkShop Visual は自動的にコールバック関数 *XDhelp_link()* をそのウィジェットの適切なコールバックリストに追加します。ウィジェットに対するコールバックダイアログで、コールバックを指定する必要はありません。

継承されるドキュメント

ヘルプファイルの名前を再入力せずに、マーカーのみを入力して、ヘルプ情報の一部を指定することができます。 ヘルプコールバックのドキュメント名を指定せずにマーカーを指定する場合には、Sun WorkShop Visual は最も近い親ウィジェットのヘルプコールバックドキュメントを使用します。ウィジェットの祖先がヘルプコールバックドキュメントを持っていない場合は、次の「モジュールのデフォルト」で説明するように、「ヘルプのデフォルト」ダイアログで指定されているデフォルトのドキュメントが使用されます。

活性化コールバックの場合は、明示的に設定されているドキュメントがないと、ヘルプコールバックのドキュメントが使用されます。このドキュメントを設定していない場合には、前述したように、同じヘルプドキュメントの継承が行われます。

モジュールのデフォルト

モジュールごとに指定されるヘルプのデフォルトは数多く存在します。これらのデフォルトを指定するためには、モジュールメニューをプルダウンして「ヘルプのデフォルト」を選択します。図 21-2 に結果のダイアログを示します。

図 21-2 「ヘルプのデフォルト」ダイアログ

「ヘルプのデフォルト」ダイアログを使用すると、Sun WorkShop Visual のヘルプシステムの構築および完成したアプリケーションの両方に使用するデフォルトを指定することができます。

- デフォルトのドキュメント
  このフィールドは、個々のウィジェットに他のドキュメントが指定されていない場合に使用するデフォルトドキュメントの名前を指定します。

- デフォルトのパス
  すべてのヘルプタグドキュメント名は、「/」で始まっていない限り相対パスであると見なされます。このフィールドは、相対パスで示されるドキュメント名の始めに追加されるデフォルトパスを指定します。デフォルトパスは、Sun WorkShop Visual 内のヘルプをテストする場合にドキュメントを検索するためにも使用されます。

- パスリソース
  アプリケーション内で、アプリケーションリソースを設定することにより、デフォルトパスを書き換えることができます。このフィールドは、アプリケーションリソースの名前を指定します。

- パス環境変数
  環境変数を設定することによって、リソース設定を書き換えることができます。このフィールドは、環境変数の名前を指定します。

- デフォルトのトランスレーション
  このフィールドは、ヘルプコールバックに指定されたマーカーを持ち、すべてのウィジェットへのトランスレーションとして追加されるイベントを指定します。トランスレーションは、*Help()* アクションを呼び出します。これにより、アプリケーションヘルプキーとしてキーの組み合せを指定することができます。

- 試行用ビューア
  このオプションメニューを使用すると、ユーザーがヘルプドキュメントを試行する際に Sun WorkShop Visual が使用するヘルプビューアを指定することができます。オプションには、Sun WorkShop Visual ヘルプ、Netscape、および FrameMaker の 3 つがあります。

- 常に固有のウィンドウを持つ
  このトグルは、FrameMaker 統合コールバックが、ドキュメントを独自のウィンドウに表示するようにします。このトグルがオフである場合、新しいページの表示に

は既存のウィンドウが使用されます。「ヘルプ用ドキュメントとマーカー」ダイアログの「固有のウィンドウを持つ」を使用することにより、ページごとに新しいウィンドウを使用するように指定することもできます。

ヘルプドキュメントとマーカーの検索

コアリソースパネルの「コード生成」ページにある「活性化コールバック (activate)」および「ヘルプ・コールバック」ボタンを使用して、現在デザインが参照しているすべてのドキュメントおよびマーカーを示すダイアログを表示することができます。

図 21-3 「ヘルプ用ドキュメントとマーカー」ダイアログ

- 追加
  適切なフィールドに名前を入力して「追加」を押すと、ドキュメントあるいはマーカーを追加することができます。

- 削除
  リストから名前を選択して「削除」を押すと、ドキュメントまたはマーカーを削除することができます。デザインにおいてウィジェットが参照しているドキュメントやマーカーは、削除することができません。

- 検索
  このボタンは、ファイルシステム内でのファイルの検索を行うための「ファイル選択」ダイアログをポップアップします。このダイアログは「ヘルプのデフォルト」ダイアログの「デフォルトパス」または「デフォルトのドキュメント」ボタンを選択しても表示することができます。

- 試行
  このボタンは、Sun WorkShop Visual に選択されたヘルプビューアへの接続を試行させ、指定されたドキュメント中の指定されたマーカーを表示します。「ヘルプのデフォルト」ダイアログで、使用したいビューアを選択してください。詳細は698 ページの「モジュールのデフォルト」 を参照してください。

- 固有のウィンドウを持つ
  各ドキュメントには、それに関連付けられている「固有のウィンドウを持つ」オプションがあります。このオプションは、システムがこのドキュメントを独自のウィンドウ内に表示し、他のドキュメントとウィンドウを共有しないように強制します。このトグルを使用すると、選択されたドキュメントに対してのオプションの状態を指定することができます。

- *<Open>*
  ドキュメントの最初のページを開かせる、特殊なデフォルトマーカーです。

- *<Widget name>*
  ドキュメント内でジャンプ移動するためのマーカーとしてウィジェット名を使用する特殊なデフォルトマーカーです。

アプリケーションへのリンク

Sun WorkShop Visual のデフォルトのヘルプコールバック関数を使用するには、Sun WorkShop Visual のインストールディレクトリ中の次に示すオブジェクトファイルをリンクする必要があります。`$VISUROOT` は、Sun WorkShop Visual のインストールディレクトリです。

ディレクトリは各ヘルプビューアにつき 1 つずつで、合計 3 つあります。Sun WorkShop Visual のヘルプおよび FrameMaker に関しては、さらに別のファイルへのリンクを行う必要があります。

Sun WorkShop Visual ヘルプ

Sun WorkShop Visual ヘルプを使用するには、ファイル `helplink.o` を自分のアプリケーションにリンクする必要があります。このオブジェクトのソースファイルである `helplink.c` は、次の場所にあります。

```
$VISUROOT/src/libhelplink/helplink
```

また、`helplink.o` を構築する `Makefile` と、Sun WorkShop Visual ヘルプでのリンクについての情報が含まれている `README` という名前のファイルもあります。

これ以外に、次の場所にあるライブラリをリンクする必要があります。

```
$VISUROOT/src/help/client
```

このディレクトリには、ライブラリを構築する `Makefile` があります。

Netscape

Netscape を使用するには、ファイル `helplink.o` を自身のアプリケーションにリンクする必要があります。このオブジェクトのソースファイルである `helplink.c` は、次の場所にあります。

```
$VISUROOT/src/libhelplink/nshelplink
```

また、`helplink.o` を構築する `Makefile` と、Netscape のリンクについての情報が含まれている `README` という名前のファイルもあります。

FrameMaker

FrameMaker を使用するには、ファイル `helplink.o` を自分のアプリケーションにリンクする必要があります。このオブジェクトのソースファイルである `helplink.c` は、次の場所にあります。

```
$VISUROOT/src/libhelplink/fmhelplink
```

また、`helplink.o` を構築する `Makefile` と、FrameMaker のリンクについての情報が含まれている `README` という名前のファイルもあります。

これ以外に、次の場所でファイルを構築する必要があります。

```
$VISUROOT/src/libhelplink/fmhelplink/libframe/fmclient
```

このディレクトリには、ライブラリの構築手順を記述した `Makefile` が用意されています。

その他

独自のヘルプコールバック関数を提供することもできます。この関数も、*XDhelp_link()* と呼ばれ、Sun WorkShop Visual の関数と同じ形式をとるようにします。ヘルプコールバック関数をカスタマイズする方法については、次の「ヘルプの実装」を参照してください。

ヘルプの実装

これまで、Sun WorkShop Visual 内でヘルプシステムが動作する方法について説明してきました。Sun WorkShop Visual の提供しているデフォルトヘルプコールバックを使用する場合は、アプリケーション内でもヘルプは同様の動作を行います。しかし、ヘルプコールバック実装のカスタマイズが必要な場合もあります。以下に FrameMaker を使用して、ヘルプコールバック実装をカスタマイズする方法について記述します。

Sun WorkShop Visual のディレクトリ `libhelplink` には、FrameMaker 統合コールバック *XDhelp_link* に対するすべてのソースが含まれています。サブディレクトリ *libframe* には、FrameMaker のリリースから採用されたさまざまなソースファイルが含まれています。元のファイルを調べる場合は、 FrameMaker の `$FMHOME/source/openmaker` にあります。

XDhelp_link() は、*_XDHelpPair_t* 構造体へのポインタである *client_data* 引数を受け取ります。

```
typedef struct _XDHelpPair_s {
_XDHelpDoc_p doc;
char** tag;
Bool open_doc;
} _XDHelpPair_t, *_XDHelpPair_p;
```

これには、*_XDHelpDoc_t* 構造体へのポインタ、マーカー (タグ) へのポインタ、*open_doc* フラグが含まれています。タグが *NULL* である場合は、開発者が *<Widget name>* (*open_doc* が *FALSE* の場合)、あるいは *<Open>* (*open_doc* が *TRUE* の場合) のどちらかのデフォルトマーカーを指定しています。以下にドキュメント構造を示します。

```
typedefstruct_XDHelpDoc_s{

char *doc;

char** path;

int handle;

Bool new_window;

}_XDHelpDoc_t,*_XDHelpDoc_p;
```

doc フィールドは、ドキュメントの名前です。*path* は、デフォルトパスが存在する場合のポインタです。このパスは、上記のように、ヘルプパスアプリケーション・リソースおよび環境変数の設定を考慮して計算されます。*handle* は、FrameMaker から返されるドキュメントハンドルであり、*new_window* はドキュメントが独自のウィンドウを持つかどうかを指定します。

メインコードファイルには、ドキュメントとマーカーの静的配列、および他の配列を指すタグのペアの配列が含まれています。タグのペア配列の要素へのポインタは、クライアントデータとして渡されます。

XDhelp_link() は、 FrameMaker と通信するための適切な関数を呼び出して、必要に応じてドキュメントを表示します。*XDhelp_link()* のその他の実装は、異なるヘルプシステムと通信します。Netscape と Sun WorkShop Visual ヘルプと統合するためのライブラリがあります。

第 22 章

国際化

はじめに

Sun WorkShop Visual は、さまざまな言語で使用できるアプリケーションの開発を支援するために、X および Motif が提供している特別な機能を活用する手助けをします。国際化は複雑な問題ですが、本章では、ソフトウェアを完全に国際化するために調べる必要のある分野に関して一定の方向性を示します。

使用言語以外の言語のテキストを入力するためには、その言語をサポートする X サーバーが必要です。X サーバーでその言語がサポートされている場合は、システムにロケールが存在します。確認方法のヒントについては、707 ページの「ロケール名」を参照してください。

問題点

あなたと別の言語を使用するユーザーが、あなたの作成したアプリケーションを使用する場合は (その別の言語が「米語」ではなく「イギリス英語」であっても) 、自分の言語および規約をユーザーが理解するとは考えられません。言語の違い以外にも住所の使い方や様式、日付や時刻の形式、個人名、用紙サイズもアプリケーションの開発者とは異なっていることがあります。

特に文字の種類の違いは、アプリケーションの外観および動作に大きな影響を与えます。韓国語、日本語、中国語などの一部の言語は表意文字を使用します。ラテン語系の言語で使用されるアルファベットは使用しません。各語が視覚的に訴えるような形

式で表わされています。そのような種類の文字 (字種) を表示するために使用するフォントは、その言語の文字を多数処理できる必要があります。この問題については、711 ページの「国際的テキストの作成」で説明します。

ヘブライ語、タイ語などの言語は、独自のアルファベットを使用していますが、ラテン語系の言語の字種とはまったく異なります。文脈に応じて一部の文字が変化します。単語同士を区別するためのスペースがないこともよくあります。また、一部の言語では右から左へ書かれ、別の言語では左から右へ書かれます。

フランス語、英語、ドイツ語のようによく似ている言語でも、文字とともに使用する記号 (すなわち発音区別符) および小文字から大文字へ、または大文字から小文字への変わり方に差があります。

X11 はフォントセットを提供することによって、このような問題の処理を支援します。詳細は、708 ページの「フォントセット」で説明します。さらに本章では、アルファベット以外の字種に必要な「入力方式」をアプリケーションで使用できるようにする機能についても説明します。

本章では、大部分の例に日本語を使用していますが、どの言語でも原理は同じです。

本章の目的は国際化という広いテーマを紹介し、Sun WorkShop Visual で国際化がサポートされている方法を説明することです。詳細については、別のマニュアルも是非参照してください。システムマニュアル、Motif マニュアル、必要に応じて 1011 ページの付録 E「参考資料」に記載されている参考文献も参照することをお勧めします。

ロケール

アプリケーションに関連した国際化の問題に取り組むにあたって、特定の場所向けに環境を設定する方法を理解する必要があります。これには「ロケール」の理解も含まれます。ロケールとは、ANSI C の概念で、一連の地域的情報を識別するために使用される名前です。たとえば、ロケールが「en_UK」に設定されている場合は、場所は英国で言語は英語という意味です。「en_US」に設定されている場合は、場所は米国で言語は英語という意味です。「ja」の場合は、日本語が使用されているという意味です。ロケールを設定することによって、ソート順、日付形式の定義などの特定の C ライブラリ関数が異なった方法で実行されます。

ロケール名

使用したいロケールのシステム名を調べ、次にそのシステム名に応じて LANG 環境変数を設定し、アプリケーションが表示すべき言語を特定できるようにします。ロケールの名前は、使用しているシステムに応じて異なることがあります。使用している UNIX システムがサポートしているロケール名のリストは、`/usr/lib/locale` ディレクトリにあります。このディレクトリには、使用できる言語ごとのディレクトリが含まれています。ディレクトリ名はロケール名として使用されます。この名前は、LANG 環境変数にも使用します。次にように入力すると、現在システムにインストールされているすべてのロケールを画面に出力することができます。

```
locale -a
```

このコマンドによって、システムに現在インストールされているロケールがすべてリストされます。

ロケールの指定

国際化サポートを使用して Sun WorkShop Visual を実行するには、正しいロケールを設定する必要があります。この設定を行うには、X サーバーを実行する前に *LANG* 環境変数を設定します。たとえば、次のように入力します。

```
setenv LANG ja
```

日本語がシステムにインストールされている場合は、このコマンドで X に対して表示フォントおよびキーボード入力に日本語を使用するよう指定します。環境変数の値は、システムによって異なる場合があります。詳細は、上記の「ロケール名」を参照してください。

キーボードとテキストウィジェットの間に入力方式を使用する日本語などの言語の場合は、正しいバージョンの Sun WorkShop Visual (この場合は日本語版) を実行していることを確認してください。LANG 環境変数とともに日本語フォントが存在することによって、入力方式が必要であることが初めて X サーバーに対して指定されるためです。

フォントセット

ロケールの定義の一部に、表示できるすべてのテキストを表示するために必要なフォントのコード化の指定が含まれます。たとえば、日本語ロケールでは、日本語のすべてのテキストを表示するには、コードセット ISO 8859-1、JIS X0208-1983、JIS X0201-1976 を持つフォントが必要です。これは、日本語にはローマ字、五十音字 (かな)、表意文字 (漢字) が含まれているためです。フォントセットはフォントの集まりで、現在のロケールに必要な文字セットがすべて含まれています。

フォントリスト

フォントリストはフォントの集まりで、ラベルを複数のフォントやスタイルで描画するために使用されます。フォントリストは Motif 固有のものであるのに対し、フォントセットは X によって定義されます。

たとえば、文字列にボールド体とイタリック体の両方を含めるよう指定するには、使用するフォント (ボールド体またはイタリック体) と使用する場所を指定したマーカーを持つコンパウンド文字列を作成します。これは、Sun WorkShop Visual の XmString エディタを使用すると、簡単に作成することができます (詳細は 186 ページの「コンパウンド文字列」を参照してください)。ただし、そのためには、フォントエディタでフォントのリスト (1 つはボールド体でもう 1 つはイタリック体) を含むフォントオブジェクトを定義する必要があります (詳細は 159 ページの「フォントの設定」を参照してください)。こうして独自のフォントリストを作成します。リスト内の各フォントには、フォントエディタで指定した名前付きのタグが付けられています。これらのタグがコンパウンド文字列に組み込まれて、X サーバーに使用するフォントを指示します。

フォントセットとフォントリストの関係

フォントセットを組み込むために Motif のフォントリストの定義が拡張された結果、フォントリスト内の各項目が単独のフォント (フォント構造体) にも、フォントセットにもなることができます。

Sun WorkShop Visual は、フォントセット中のフォントを単純にフォントリストの直接のメンバーとして表示することによって、事態を単純化します。ただし、「フォント選択」ダイアログでフォントセットトグルを設定する場合は、Sun WorkShop Visual に対してこのフォントはフォントセットのメンバーであること

を告げていることになります。Sun WorkShop Visual は内部では、フォントセットのすべてのメンバーに同じフォントリストタグを設定しています。次に Motif は、同じフォントリストタグを持つフォントをすべてフォントセットのメンバーとして処理します。

フォントオブジェクトについては、165 ページの「単純フォントオブジェクト」を参照してください。フォントリストについては、188 ページの「複合フォントオブジェクトの作成」を参照してください。

フォントセットの使用例

次の例題で、フォントの使用方法を説明します。

前述の日本語ロケールには、以下の 3 つのコード化形式が必要です。

- ラテン文字 (ローマ字) 用の ISO 8859-1
- 表意文字 (漢字) 用の JIS X0208-1983
- 五十音字 (かな) 用の JIS X0201-1976

1. 「フォント選択」ダイアログをポップアップします。
2. フォントオブジェクト (例: `f1`) およびタグ名 (例: `t1`) を入力します。
3. 「フォントセット」トグルをオンにします。
4. フィルタメニューを使用して、ISO 8859-1 フォントをすべて表示します。
5. 該当する ISO 8859-1 フォントを選択して「バインド」を押します。

   このフォントセットをデフォルトとして使用したい場合は、「デフォルト」ボタンを押して必ずタグ名を <Default> にしてください。このボタンを押さない場合は、ウィジェット内で必ずこのフォントセットが使用されるように、コードを記述する必要があります。

   この段階では、フォントセットが現在のロケールでテキストを表示するために必要な文字セットを十分持っていないことを警告するメッセージが Xlib から表示されます。これは、まだ 2 つのフォントをバインドしていないためです。

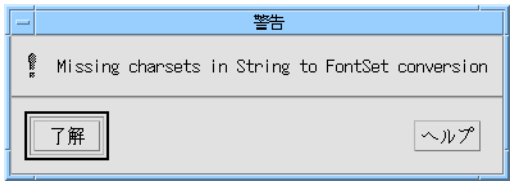


図 22-1　文字セット不足の警告

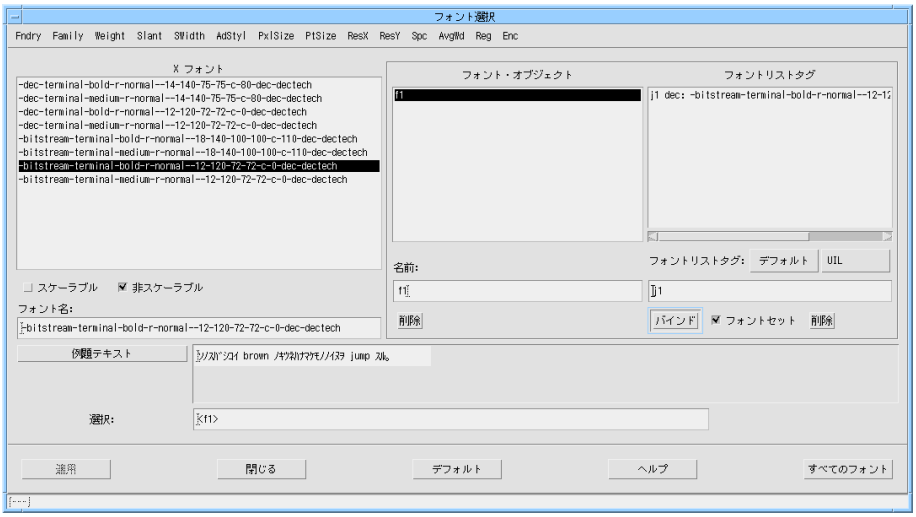


図 22-2　フォントセットの初期フォント

6. JIS X0208-1983 レジストリを持つフォントを選択し、バインドします。

   Xlib から文字セット不足を警告するメッセージが再度表示されます。

7. JIS X0201-1976 レジストリを持つフォントを選択し、バインドします。

   これでフォントセットが完全に揃ったので、今回は Xlib からの警告メッセージはありません。

   フォントオブジェクト項目は、フォントセットの一部であるかどうかには関係なく、削除および再バインドを行うことができます。フォントリストには、フォントセットにも単純フォントにもなることができる項目を多数持たせることができます。これは、さまざまなフォントリストタグを指定し、必要に応じて「フォントセット」トグ

ルを設定して、容易に行うことができます。以下の例では、*t1*、*big*、*t2* という 3 つの項目を持つ単独フォントオブジェクトを示します。*t1* および *t2* は ja ロケールの完全なフォントセットで、*big* は単純フォントです。

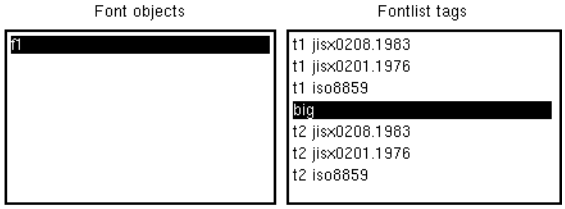


図 22-3 3 つの項目を持つフォントオブジェクト

国際的テキストの作成

日本語、中国語、韓国語などの言語の場合は、キーボード上のキーの数、すなわち 8 ビットで表される数よりも多い文字数を表示することができます。さらに、日本語文字で表示されるテキストには、ローマ数字およびローマ字のテキストも混在させる必要があります。これらの問題を解決するには、テキストを特別な方法でコード化します。このような言語の文字より大きな文字セットを使用するためには、よく知られている ASCII マッピングではなくマルチバイトテキスト文字列を使用します。テキスト内の複数の文字が、一致するフォントの 1 文字を表します。1 バイト文字とマルチバイト文字を区別する方法は、コード化の方法に依存します。さらに、このような言語は音 (キーボードに入力される音) と表意文字が一対多でマッピングします。この問題を解決するために入力方式が提供されています。

ヘブライ語、アラビア語などの言語でも入力方式が必要です。文字が単語内の位置に応じて変化したり、母音が子音を囲んで記述されるためです。

正しく日本語化されたバージョンの Sun WorkShop Visual を実行している場合は、テキストをテキストフィールドに入力する際に特殊な一連のキー操作によって入力方式が使用できるように切り替わります。入力方式によって、入力された文字が目的とする言語の対応する文字に変換されます。

Sun WorkShop Visual での入力方式の使用

Sun WorkShop Visual で英語以外のテキストをあらゆるテキスト入力領域に入力するには、まず以下の必要条件を満たしているかどうかを必ず確認してください。

1. LANG 環境変数が目的の言語の種類 (ロケール) に設定されている

   詳細は、706 ページの「ロケール」を参照してください。

2. 目的の言語をサポートする X サーバーがある
3. 正しいバージョンの Sun WorkShop Visual である
4. 指定言語に入力方式が必要な場合は、その入力方式がシステム上に存在し、自動的に起動されているか、ユーザーによって起動されている

以下の例で、入力方式が必要な言語でテキストを入力する方法を示します。使用言語は日本語です。CDE を使用している場合は、次を実行します。

1. ログインするときにダイアログボックスでロケールを設定してから通常のログインを行います。

CDE を使用していない場合は、次を実行します。

1. LANG 環境変数を「ja」に設定します。

   OpenWindows を使用している場合は何もする必要はありません。ただし、Motif Window Manager を使用している場合は、コマンド行プロンプトで **`htt`** と入力して入力方式サーバーを起動します。

2. X サーバーを起動します。

   テキストフィールドを含むウィンドウに現在のフォーカスがある場合は、必ずウィンドウの最下部にステータス行が表示されます。これは、現在の入力モードを表示するテキスト領域です。

CDE の使用に関係なく、以下の手順は同じです。

3. Sun WorkShop Visual を起動します。

   保存したデザインを読み込んでも、新規のデザインを開始しても構いません。必ずテキストウィジェットがあるものにしてください。

4. テキストウィジェットを選択し、そのリソースパネルを呼び出して「ラベル」フィールドを選択します。

5. Ctrl-Space を押します。入力方式のドキュメントで正しいキーを確認してください。

   ステータス行は、かな入力が可能であることを示します。

6. Ctrl-W を押します。

   かなが漢字に変換され、候補語リストが表示されて、テキストフィールドに入力したい漢字を選択することができます。

7. 候補語リストから漢字を選択して、<Space> を押します。

   選択した漢字が Sun WorkShop Visual に送信されます。

   他の言語の場合も手順は似ています。まず、LANG 環境変数を設定し、X サーバーを起動し、次に Sun WorkShop Visual を呼び出します。一部の言語および一部のシステムでは、さらに入力方式またはサーバーも起動する必要があります。システムのマニュアルを参照して、この必要があるかどうかを調べてください。テキスト領域を持つウィンドウに現在のフォーカスがあるときは、必ず入力方式が現われます。

   Sun プラットフォームでの国際的テキストについての詳細は、Sun のマニュアルを参照してください。

アプリケーションへの日本語化された入力

前節では、Sun WorkShop Visual (例ではラベルリソース) に 非 ASCII テキストを入力する方法について説明しました。本節では、日本語化されたテキストを自分の生成したアプリケーションで入力する方法について説明します。「変換後のテキスト」という用語を使用することに注意してください。これは、標準のキーボード上のテキストとは異なって表示されている画面上のテキストを意味します。

テキストの変換方法

X ツールキットが大部分の処理を行います。これは、特定ウィジェットが ASCII 以外のテキストを受け取ることを X ツールキットに指定するだけで、簡単に行うことができます。必要条件は以下のとおりです。

1. ロケールが設定されている

707 ページの「ロケールの指定」を参照してください。

2. 現在のロケールと一致するフォントセットを含むテキストウィジェットに対してフォントオブジェクトを指定済みである

前述の 2 番目の条件が満たされている場合は、テキストウィジェットは包含しているシェルに入力方式との接続を確立するよう要求します。次に X ツールキットがこの処理を引き継ぎ、そのテキストウィジェットに対するテキスト入力を入力方式に送り、変換後のテキストをテキストウィジェットに返します。

テキストウィジェットによる変換テキストの受信の設定

前述したように、アプリケーション内のテキストウィジェットが入力方式を使用するために必要な作業は、作成時に必ずテキストウィジェットに対して、ロケールと一致するフォントセットを含むフォントリストを指定させることだけです。最も簡単に行うには、Sun WorkShop Visual を使用して、該当するフォントセットを含むフォントオブジェクトを指定し、そのオブジェクトをウィジェットのフォントリソースとして使用します。この方法を以下の手順で説明します。

1. テキストの子を持つダイアログを作成します。
2. シェルをアプリケーションシェルに指定します。
3. このテキストを選択し、リソースパネルをポップアップします。
4. 「フォント」ボタンをクリックします。
5. 該当するフォントセットを持つフォントオブジェクトを指定します。

   この方法については、709 ページの「フォントセットの使用例」を参照してください。フォントセットのフォントには必ず「Default」タグを指定してください。指定しないと、X サーバーは入力方式が必要であることを認識できません。
6. コードを生成し、コンパイルします。
7. 生成されたアプリケーションを、テキストウィジェットに指定された言語をサポートする X サーバー上で実行します。

ダイナミックディスプレイでの入力方式の表示

前述の例によって、生成されたアプリケーションには入力方式が正しく設定されますが、問題点もあります。Sun WorkShop Visual は、常にリソースが適用される前にウィジェットを作成します。これは、ウィジェットのリセットやペーストを行なった場合でも同様です。したがって、ダイナミックディスプレイウィンドウで入力方式を

動作させるには、作成時にフォントを強制的に指定する必要があります。その最適な方法は、親のブリテンボードまたはシェルウィジェット上でテキストフォントリストリソースを設定することです。ただし、この方法では子テキストウィジェットのすべてが、入力方式を持つことになるので注意してください。

Sun WorkShop Visual 内でテキストウィジェットが入力方式を使用して動作していることを確認するには、以下の作業を行います。

1. フォームの子を持つダイアログを作成します。
2. シェルをアプリケーションシェルに指定します。
3. このフォームを選択し、リソースパネルの「フォント」ページをポップアップします。
4. 「テキスト用フォント」ボタンをクリックします。
5. 該当するフォントセットを持つフォントオブジェクトを指定します。

   この方法については、709 ページの「フォントセットの使用例」を参照してください。フォントセットのフォントには必ず「Default」タグを指定してください。指定しないと、X サーバーは入力方式が必要であることを認識できません。
6. テキストウィジェットの子を作成します。
7. 必要に応じて別のウィジェットを追加します。

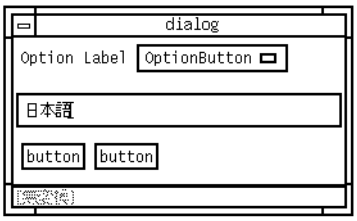


図 22-4 入力方式付きの単純なダイアログ

デフォルトの *main()* プログラムには、ロケール処理ルーチンを初期化する *XtSetLanguageProc()* 呼び出しが含まれています。独自の *main()* プログラムを記述していて、国際化機能を使用している場合は、このルーチンへの呼び出しを追加する必要があります。作成した *main()* ルーチンの最初に、次の行を追加してください。

```
XtSetLanguageProc ( (XtAppContext) 0, (XtLanguageProc) 0, (XtPointer) 0 );
```

アプリケーションのフォントリソースの設定

国際的テキストのフォントを Sun WorkShop Visual のダイナミックディスプレイまたは生成されたアプリケーション上で正しく表示するには、国際的テキストの FontList リソースを正しく設定する必要があります。

フォントリソースをアプリケーション全体に正しく設定するには、緩い結合を使用して FontList リソース設定を定義することをお勧めします。以下に例を示します。

注 - 以下の例は、見やすくするために改行していますが、実際の設定では改行は行わないでください。改行を使用すると、そのリソース設定は無視されます。

```
XApplication*FontList:
                -misc-fixed-medium-r-normal-*-14-130-75-75-c-140-
                    jisx0208.1983-0;
                -*-*-medium-r-normal-*-14-130-75-75-c-70-
jisx0201.1976-0;
                -misc-fixed-medium-r-normal-*-14-110-100-100
                    -c-70-iso8859-1:default
```

XApplication は、作成するアプリケーションの名前です。この例は、日本語のフォントを設定するためのものです。この例のリソースは、$VISUROOT/src/examples/loose_bindings/ja_fontlist.res に用意されています。「緩い結合」ダイアログを使用して、このリソースをデザインに読み込むこともできます。

緩い結合を使用したリソースの設定方法の詳細は、97 ページの「緩い結合」を参照してください。

シェルタイトルでの 8 ビット文字の使用

シェルウィジェットのタイトルのテキストに 8 ビットの文字を使用する場合は、*titleEncoding* リソースが `STRING` に設定されていることを確認してください。リソースは、 Sun WorkShop Visual で緩い結合を使用して設定します。以下に例を示します。

```
XApplication*titleEncoding:STRING
```

(XApplication は、作成するアプリケーションのクラス名です。)

または、次のように設定します。

```
*titleEncoding:STRING
```

リソース設定を適用する対象が自分で作成したアプリケーションのみの場合は前者を、すべてのアプリケーションの場合は、後者を設定してください。

詳細は 104 ページの「密な結合」を参照してください。

このリソース設定をデザインに「ハードコード」したい場合は、メインシェルのマネージ前プレリュード次のように設定してください。

```
XtSetArg ( al[ac], XmNtitleEncoding, XA_STRING ); ac++;
```

マネージの前プレリュードの追加方法については、285 ページの「シェルの作成の前プレリュード」を参照してください。

サポートされていないロケール

LANG 環境変数を使用してロケールを指定する方法を、712 ページの「Sun WorkShop Visual での入力方式の使用」で説明しました。Sun WorkShop Visual の適切なバージョンを使用することで、デザインしているユーザーインタフェースの文字列にロケールの言語を使用することが可能でした。Sun WorkShop Visual のご使用のバージョンでサポートしていないロケールに設定した場合は、起動時にメッセージが表示されて、現在の LANG 設定がサポートされておらず、強制的に「C」に変換されることが通知されます。この設定はデフォルトで、米語が使用されます。ご使用の Sun WorkShop Visual がラテン語のスクリプト[1] に対応している場合であれば、正し

1. この場合の「ラテン語のスクリプト」とは ISO ラテン 1 文字セットを意味します。

く表示して機能します。問題となる領域は、生成されたコード内の文字列だけです。以下の場合であれば、指定したロケールでラベルやテキスト文字列などを追加できます。

- LANG に指定したロケールがラテン語の文字セットを使用している
- 適切なフォントが使用できる
- 指定したロケールの言語が特別な処理 (たとえば、入力方式を使用しなければ左から右に表示されないなど)を必要としていない

ただし、上記のいずれにも当てはまらない場合は、指定した言語で文字列を入力することも表示することもできません。

第23章

ユーザー定義ウィジェット

はじめに

Sun WorkShop Visual は、Motif ウィジェットセットによってあらかじめ構成されていますが、デフォルトセットに加え、他のソースからのウィジェットをサポートするように拡張することができます。Sun WorkShop Visual にユーザーによって追加されるウィジェットは、ユーザー定義ウィジェットと呼ばれます。これらのウィジェットは Sun WorkShop Visual ウィジェットパレットに表示されます。ユーザーがウィジェットを作成すると、そのリソースを設定して、それらを含むデザインのコードを生成することができます。

Sun WorkShop Visual は、ユーティリティ visu_config を備えています。このユーティリティは、ユーザー定義ウィジェットをサポートするために Sun WorkShop Visual が必要とする情報を提供する際に役立ちます。ユーザーはこのユーティリティを使って使用するウィジェットを指定し、そのウィジェットに必要な非標準のリソース型についての情報を与えます。すると、visu_config は Sun WorkShop Visual とユーザー定義ウィジェットの間の橋渡しとして機能する 3 個の C ファイルを生成します。

visu_config ファイルの生成後、以下に示す構成要素を使用して Sun WorkShop Visual の新バージョンを構築してください。

- Sun WorkShop Visual オブジェクトファイル `visu.o`
- 追加ウィジェットを含むオブジェクトファイルまたはアーカイブライブラリ
- visu_config により生成されるコードファイル

- visu_config により生成される構成ファイル
- ウィジェットアイコンのためのビットマップファイル (必要に応じて)
- ウィジェットアイコンのためのピックスマップファイル (必要に応じて)
- 補助関数を含む手書きコードファイル (必要に応じて)

アイコンを使用することをお勧めしますが、必須要素ではありません。手書きコードは、カスタマイズされたポップアップダイアログを提供しようとする場合、あるいは 773 ページの「構成関数」に記述されている特殊な問題がある場合にのみ必要です。

事前構成済み統合キットの使用

本章では、visu_config の使用について説明します。ウィジェットを Sun WorkShop Visual に統合する手続きはやや複雑です。visu_config を使用すれば、この手続きを簡素化できますが、visu_config 自体がやや複雑なツールでもあります。そこで、Sun WorkShop Visual では、いくつかのウィジェットセット用に事前構成済みの統合キットを用意しました。Sun WorkShop Visual リリースディレクトリの `user_widgets` ディレクトリで、サポートされているウィジェット統合を確認してください。必要に応じて多くのセットが使用できるため、セットに組み込まれている統合キットで十分に対応できるはずです。すべてのセットの詳細については、Sun WorkShop Visual のご購入先までお問い合わせください。各統合キットの使用方法を説明した詳細情報は `README` ファイルにあります。

visu_config は、ユーザーが所有しているウィジェットセットを統合する場合、または標準 Motif セットの他に 1 つ以上のウィジェットセットを含むパレットを構築する場合に必要です。

必要条件

ユーザー定義ウィジェットは、X11 リリース 5X またはリリース 6X ツールキット組み込み関数を構築し実行する必要があります。各ウィジェットクラスには、公開ヘッダーと、ウィジェットクラスを実現するオブジェクトファイルが必要です。これらの

ファイルはどちらもウィジェットの供給元から提供されます。オブジェクトファイルはアーカイブライブラリに保存されます。また、ウィジェットクラスに対しての非公開ヘッダーファイルが必要な場合もあります。

C コンパイラ、リンカー、および *make* を含む標準 UNIX 開発ツールへのアクセスが必要です。

visu_config は、ウィジェットクラス記述等の標準ウィジェット情報を必要とします。したがって、ウィジェットの資料を手元に用意しておくと良いでしょう。ウィジェットが非標準リソース型を持つ場合、あるいはカスタマイズされたポップアップダイアログを提供する場合には、ヘッダーファイルやウィジェットソースコードを参照して必要な情報を獲得する必要があります。726 ページの「ウィジェット情報の獲得」を参照してください。

UILの生成

通常、他社のウィジェットに対しては C または C++ を生成しますが、UIL を生成することもできます。それには、Sun WorkShop Visual にその実行方法を伝えるために、さらに情報を追加する必要があります。詳細は、780 ページの「UIL の生成」を参照してください。

Java コードの生成

デザインにユーザー定義のウィジェットが含まれているとき、このウィジェットが Java に対応していなくても、Java コードを生成することができます。Sun WorkShop Visual は、Java Panel クラスでコンテナウィジェットを代用し、その他のタイプのウィジェットには Java Canvasクラスを使用します。Java コードの生成の詳細については、第 10 章「Java 用のデザイン」を参照してください。

注意事項

Sun WorkShop Visual ダイナミックディスプレイは、ウィジェットの実際のインスタンスを作成することによって有効となるため、Sun WorkShop Visual に組み込まれるウィジェットはすべてツールの一部となります。ウィジェットが意図したとおりに機能しない場合には、ユーザーがデザインにそのウィジェットを追加すると Sun WorkShop Visual が異常終了する場合もあります。ウィジェットにおけるメモリーリークは、Sun WorkShop Visual に影響を与え、処理速度が徐々に低下したり、あるいはコアダンプの原因となる可能性があります。標準ベンダーから提供される汎用ウィジェットであっても問題を生じる場合もあります。したがって、ウィジェットを Sun WorkShop Visual に追加する前には、入念なテストを行なってください。

前提条件

Sun WorkShop Visual を構成するためには、C についてのある程度の知識、および make などの共通 UNIX 開発ツールについての理解が必要です。X のエキスパートである必要はありませんが、X および X ツールキット組み込み関数についてのある程度の知識は必要です。visu_config には、ウィジェットクラスポインタおよびリソース型を表わす記号定数など、ウィジェットについての情報を供給しなければなりません。ウィジェットの資料またはソースコードから必要な情報を獲得する方法については、726 ページの「ウィジェット情報の獲得」を参照してください。

Sun WorkShop Visual の動作

ウィジェットの構成を開始する前に、Sun WorkShop Visual が選択されたウィジェットをどのようにダイナミックディスプレイに適用するかを理解しておくと役立ちます。本節では、Sun WorkShop Visual の内部動作について説明します。

ウィジェットの作成

Sun WorkShop Visual は、シミュレーションではなく実際のウィジェットを使用してダイナミックディスプレイを構築します。ウィジェットの作成は、ユーザーが visu_config で指定するウィジェットクラスポインタを *XtCreateWidget()* に渡すことによって実行されます。また、ウィジェットクラスポインタによって、ウィジェットのリソースおよびその型についての情報へアクセス可能になり、Sun WorkShop Visual がウィジェットのリソースパネルを構築することができるようになります。

ダイナミックディスプレイにおいて作成およびマネージされるウィジェットの順番は、アプリケーションでの標準的な操作順序とは異なります。ユーザーがアイコンをクリックして階層にウィジェットを追加する時、Sun WorkShop Visual はウィジェットのインスタンスを作成し、リアライズ (実体化) し、リソースが設定される前にそれをマネージします。一方、Sun WorkShop Visual がファイルから階層を読み出す場合は、階層は下から順番に作成されます。各ウィジェットは、まず作成され、次にそのリソースが設定されてから最後にマネージされます。どちらの場合においても、Sun WorkShop Visual はリソースを設定する前にウィジェットを作成するため、作成時のみ設定可能なリソースは通常の方法ではダイナミックディスプレイには設定できません。

Athena Form ウィジェットなどウィジェットによっては、作成時リソース設定を必要としたり、あるいは子を持たずにマネージされると正しく動作しない場合があります。このような場合は、visu_config にリアライズ関数を指定して、階層にウィジェットを追加する Sun WorkShop Visual の手順をカスタマイズすることができます (詳細は、774 ページの「リアライズ関数」を参照してください)。

注 - 上記のような問題は生成コードには影響しません。Sun WorkShop Visual は、階層を下から順番に作成するコードを生成し、作成時にすべてのリソースを設定します。

選択されたウィジェットの強調表示

ツリー内のウィジェットを選択すると、選択されたウィジェットのアイコンは強調表示されます。また、ダイナミックディスプレイ内のウィジェット自体も、前景と背景の色を入れ替えることにより強調表示されます。ウィジェットが作成時にのみ設定可能な前景リソースを持っている場合は、ウィジェットの強調表示により問題が生じる

可能性があります。この場合は、733 ページの「ウィジェットの属性」に説明されているように、ウィジェット編集ダイアログの前景スワッピングを使用不可能とすることができます。

無効な階層の防止

Sun WorkShop Visual は、無効な階層が作成されないように、選択されたウィジェットに子としてアタッチすることのできないウィジェットのアイコンをすべて使用不可能にします。また、階層に新しく追加されたウィジェットが子を持つことができない場合には、Sun WorkShop Visual はそのウィジェットを自動選択しません。

ユーザー定義ウィジェットの場合、Sun WorkShop Visual はウィジェットのスーパークラスを調べて、有効な階層を決定します。また、構成関数を指定することにより、有効な子および親ウィジェットに対してのウィジェットの必要条件をカスタマイズすることも可能です。詳細は、773 ページの「構成関数」を参照してください。

リソースパネルの構築

Sun WorkShop Visual は、ウィジェットクラスレコードにあるリソース名およびリソース型にもとづき、ウィジェットのリソースパネルを構築します。コアウィジェットまたは Motif 親クラスなどの既知のスーパークラスから継承されているリソースは、スーパークラスに対してのリソースパネル上で設定することができるので、当該ウィジェットのリソースパネルからは除かれています。

Sun WorkShop Visual は、自動的に標準のリソース型をリソースパネル上の適切なページに割り当てますが、必要に応じて明示的に他のページに割り当てることもできます。ほとんどのウィジェットリソースは、標準的カテゴリのいずれかに分類されます。その内容については、740 ページの「リソース」を参照してください。

リソースパネルが表示されると、Sun WorkShop Visual は *XtGetValues()* を使用してウィジェットの全リソースの現在値を獲得します。列挙型を除くすべてのリソースはテキスト文字列に変換され、リソースパネル上のテキストフィールドに表示されます。ユーザーはテキスト文字列を編集してリソースを設定できます。フォント、色およびコールバックなどのように、ユーザーがポップアップダイアログを使用して間接的にテキストを指定できる場合もあります。Sun WorkShop Visual の中でポップアップダイアログを持っていないリソースに対しては、ポップアップを提供することができます。詳細は、755 ページの「ポップアップ」を参照してください。

リソースの設定

ユーザーが新しいリソース値を適用すると、Sun WorkShop Visual はテキスト文字列を値に変換し、それを *XtSetValues()* を使用してウィジェットに適用します。すると、即座に *XtGetValues()* を使用してそのウィジェットのすべてのリソース値を検索し、それらの値を変換してテキストに戻し、戻したテキストをリソースパネルに表示します。この手順は、ツールキットにとって新しい値が有効かどうか、および新しい値によって他のリソースが変更されたかどうかを即座に示します。

テキスト文字列をリソース値に変換したり戻したりするためには、リソースコンバータと呼ばれる関数が使用されます。標準リソース型にはコンバータ関数が組み込まれているため、ユーザーは visu_config を使用する必要はありませんが、ウィジェットが非標準のリソース型を持っている場合は、visu_config を使用してコンバータ関数についての情報を指定することができます。詳細は、753 ページの「コンバータ」を参照してください。この情報を指定しない場合でも、ユーザーは生成コード内のリソース、あるいはリソースファイルにテキスト文字列を設定することができます。しかし、Sun WorkShop Visual はテキスト文字列を値に変換することができないため、ダイナミックディスプレイ上でリソースを設定することはできません。

列挙型リソースに対しては、Sun WorkShop Visual はオプションメニューを構築します。オプションメニューは、デフォルトでリソースパネルの「設定」ページに配置されます。Sun WorkShop Visual は、ユーザー定義ウィジェットに対しての Boolean 列挙型リソースを処理することができます。その他の列挙型に対しては、Sun WorkShop Visual がオプションメニューを構築することができるように、visu_config に有効な値のリストを提供する必要があります。詳細は、746 ページの「列挙型」を参照してください。

デザインの保存とコード生成

Sun WorkShop Visual では、デザインの保存および保存ファイルの解析を簡単に行うことができます。リソース設定の場合、Sun WorkShop Visual はリソース名およびその値のテキスト表示を `.xd` ファイルに書き込みます。リソース名および値は、リソースパネルに表示される通りに保存されます。

コードを生成する場合、Sun WorkShop Visual はそれぞれのリソースについて 2 通りのバージョンを使用します。生成される X リソースファイルにおいては、リソースは *label* のような名前で識別され、Sun WorkShop Visual はウィジェットクラスレコードからリソース名を獲得します。生成コードにおいては、リソースは *XtNlabel* のような定義名で識別され、Sun WorkShop Visual はリソース名に接頭辞 `XtN` を追加すること

により定義名を構成します。ウィジェットがこの命名規則に沿っていない場合は、visu_config を使用して定義名を構成するための関数を指定することができます。詳細は、773 ページの「構成関数」を参照してください。

列挙型の場合も、Sun WorkShop Visual は、*center* のようなリソースファイル記号と *XtJustifyCenter* のようなコード記号というように、それぞれの 2 通りのバージョンを使用します。列挙型を構成する場合は、各値の両方のバージョンを提供する必要があります。

Sun WorkShop Visual がウィジェットクラスに対してのコードを生成する場合、そのクラスの公開ヘッダーファイルに対して *#include* を生成します。visu_config を使用すると、各ユーザー定義ウィジェットに対して *#include* ファイルを指定することができます。

ウィジェット情報の獲得

ウィジェットを構成するためには、ウィジェットのコードからひとつあるいは複数の変数名および記号定数を抜き出す必要があります。この節では、ユーザーが指定する必要がある情報について説明します。必要な情報のほとんどは、ウィジェットの資料から利用することができます。そこから情報が得られない場合は、ウィジェットの公開および非公開ヘッダーファイル、ウィジェットソースコード (使用可能な場合)、あるいはウィジェットの開発元のテクニカルサポートサービスから情報を得ることができます。

これから、多くのウィジェット開発元が準拠している命名規則について説明します。ただし、命名規則は変更される可能性があるので、資料またはソースコードで常に名前を確認してください。

ウィジェットクラスポインタ

visu_config にウィジェットクラスを追加する場合、ウィジェットクラスポインタを指定する必要があります。ウィジェットクラスポインタは、*WidgetClass* 型のポインタ変数名です。このポインタにより、リソースのリストおよびその型を含む、ウィジェットクラスについての情報を持っている構造へアクセスすることができます。
Sun WorkShop Visual は、この情報を使用して、ウィジェットクラスのリソースパネルを構築します。規約により、ウィジェットクラスポインタは、
<クラス名> *WidgetClass* という形式をとります。

資料でウィジェットクラスポインタを見つけることができない場合は、以下に示すような行を公開ヘッダーファイル内で探してください。

```
externalref WidgetClass fredWidgetClass
```

または、

```
extern WidgetClass fredWidgetClass
```

どちらの場合においても、ウィジェットクラスポインタは *fredWidgetClass* です。

リソース情報

リソース名とは、生成される X リソースファイル内およびリソースパネルでリソースを識別するために使用される文字列です。Sun WorkShop Visual はこの名前をウィジェットクラスレコードから直接得るため、ユーザーが指定を行う必要はありません。この文字列は通常、 *label* のように接頭辞のない単純な名前です。

定義名は、ソースコードにおいてリソース名を識別するために使用される記号定数です。規約により、定義名は「<接頭辞>*N*<名前>」の形式をとります。ここで <名前> はリソース名です。定義名を見つけるには、ウィジェット資料、あるいは *#define* 宣言の公開ヘッダーファイル内を調べます。たとえば、以下に示す Athena Form ヘッダーファイルにある行では、定義名 *XtNtop* および *XtNbottom* が定義されています。

```
#define XtNtop "top"
#define XtNbottom "bottom"
```

非標準型リソース

標準型以外のリソースを構成するためには、リソース型を理解している必要があります。リソース型というのは、*unsigned char* のような型ではなく、ウィジェットクラスがリソース型を識別することのできる、文字列として定義されている記号定数です。規約により、リソース型は「<接頭辞>*R*<型> 」の形式をとります。非標準リソース型、特に列挙型では、<型> はリソース名と同じ場合があります。

ウィジェットの資料が *foo* のようなリソース型を示している場合は、以下に示すような公開ヘッダーファイル内の行を探してください。

```
#define XtRFoo "foo"
```

この例では、 visu_config がリソース型を要求した場合、*XtRFoo* と入力します。このリソース型はウィジェットのソースコードでも見つけることができます。次に示す構造は、リソース型が *XtRFoo* であるリソースを定義します。この構造からは、リソースの定義名 *XtNfoo* も得ることができます。

```
{
                XtNfoo, XtCFoo, XtRFoo,
                sizeof(foo), XtOffset( FooWidget, foo),
                XtRImmediate, (XtPointer) NULL,
}
```

標準以外の列挙型

ウィジェットが標準以外の列挙型リソースを持っている場合、指定可能な値のリストを指定する必要があります。必要な名前を獲得するためには、ソースコードを読まなければいけない場合もあります。詳細は、746 ページの「列挙型」を参照してください。

visu_config のメインダイアログ

以下のように入力して、visu_config を実行します。

```
visu_config
```

図 23-1 に示すメインダイアログが表示されます。

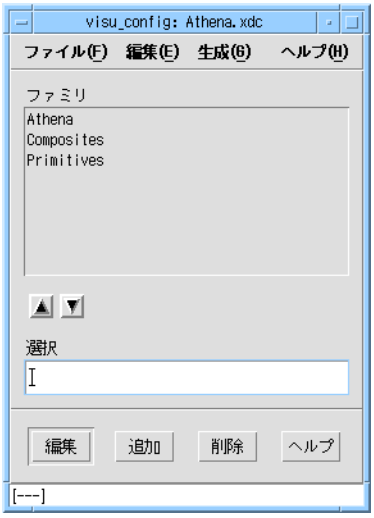


図 23-1 visu_config メインダイアログ

メニューコマンド

visu_config の「ファイル」メニューには、構成データを含むファイルの保存、および読み出しを行うオプションがあります。規約により、これらのファイルは接尾辞 .xdc を持ちます。既存ウィジェット仕様ファイルを開くためには「開く」を使用します。現在編集中のファイルと別のファイルを組み合せるためには「読む」を、ファイルの保存には「保存」および「別名保存」を、そして編集領域をクリアするためには「新規」を使用します。

「編集」メニューは「停止リスト」ダイアログを表示するために使用します。「停止リスト」ダイアログを使用すると、選択した Motif ウィジェットを Sun WorkShop Visual パレットから削除することができます。詳細は 766 ページの「Motif ウィジェットの停止リスト」を参照してください。

生成メニューには、追加されたウィジェットを使用して Sun WorkShop Visual を構築するために必要となる 2 種類のコードファイルを生成するためのオプションがあります。visu_config からのコードの生成および Sun WorkShop Visual の構築についての詳細は、767 ページの「コードの生成とコンパイル」を参照してください。

ファミリ

メインダイアログは、ファミリのリストを表示します。ファミリとは、ウィジェットパレット内に一緒に表示されるウィジェットの組です。プログラムを起動した時点では、このリストは空になっています。図 23-1 は、Sun WorkShop Visual と一緒に供給される `Athena.xdc` ファイルを読み込んだ後のダイアログを示しています。このファイルを開いて、内容を確認してください。

ユーザー定義ウィジェットは、必要に応じてファミリに編成することができます。ウィジェットをファミリにすることには、2 つの利点があります。第 1 に、ウィジェットパレットのサイズが大きくなりすぎないように適当な大きさに保つことができます。Sun WorkShop Visual は、デフォルトの Motif ウィジェットとユーザー定義ウィジェットのファミリ 1 個のアイコンをウィジェットパレットに表示します。オプションメニューを使用すると、リソースパネルを使用する場合と同様に、別のファミリに切り換えることができます。第 2 には、ウィジェットをファミリにグループ化することにより、異なるファミリの組み合せを持つ Sun WorkShop Visual の別バージョンを容易に生成することができます。コード生成時には、リストから任意のファミリの組み合せを選択することができます。このため、Sun WorkShop Visual をカスタマイズして、用途や技術レベルの異なるユーザーをサポートすることが可能になります。

ファミリリストの編集

リストに新しいファミリを追加するには、「選択」フィールド内にファミリ名をタイプ入力し、その後「追加」をクリックします。ファミリには任意の名前を指定することができます。ファミリ名は visu_config および Sun WorkShop Visual ウィジェットパレットにおいてファミリを識別するために使用されます。

ファミリを削除する場合は、リストから削除するファミリを選択し、「削除」をクリックします。また、リストを並べ替える場合は、ファミリを選択した後に矢印ボタンを使用して上下にファミリを移動します。リストの順序によって、ウィジェットパレット内のオプションメニュー項目の順序が決定されます。

ファミリの分け方のヒント

同一のウィジェットを複数のファミリに含めることができます。たとえば、`Athena.xdc` の場合、「Athena」という名のファミリはすべての Athena ウィジェットを含んでおり、「Composites」と「Primitives」という 2 個の小ファミリはそれぞ

れ Athena グループの部分集合です。 visu_config からコードを生成する際には、ウィジェットパレット上での表示方法を決定することができます。つまり、大ファミリを使用してすべての Athena ウィジェットを同時に表示したり、あるいは 2 個の小ファミリを使用してサブセットを表示することができます。

頻繁に使用するウィジェットを複数のファミリに含めることにより、表示されているパレットのページとは関係なく、常にそれらのウィジェットへアクセスすることができます。ただし、そのためにはウィジェット構成情報のコピーを 2 個別々に入力し、維持する必要があります。また、パレットの各ページにおいてアイコンを別々にテストする必要もあります。

visu_config からコードを生成する際には、現在開いているファイルから任意のグループを選択することができますが、その他のファイルからは選択できません。複数の `.xdc` ファイルのウィジェットファミリを持つ Sun WorkShop Visual を構成するためには、コードを生成する前に「読む」オプションを使用してファイルを組み合わせます。

ファミリへのウィジェットの追加および編集

ウィジェットをファミリへ追加あるいは構成するには、ファミリを選択した後「編集」をクリックしてファミリ編集ダイアログを表示します。ファミリ編集ダイアログを使用すると以下の作業を行うことができます。

- 選択されたファミリでのウィジェットの追加または削除
- 選択されたファミリでのウィジェットクラスの仕様の編集
- 非標準リソース型の処理方法の指定
- リソースに対してのポップアップダイアログの指定

ファミリ編集ダイアログには複数のページがあり、それらを「表示」メニューで選択することができます。現在選択されているファミリの名前がダイアログのタイトルバーに表示されます。

ファミリ編集ダイアログの詳細については、以下の節を参照してください。

ウィジェットクラス

そのファミリ内にあるウィジェットクラスのリストを表示するには、「表示」メニューから「ウィジェット」を選択します。図 23-2 に Athena の「Composites」ファミリの「ウィジェット」ページを示します。

図 23-2 ファミリ編集ダイアログの「ウィジェット」ページ

ウィジェットクラスの追加

ファミリに新しいウィジェットクラスを追加するには、「選択」フィールドにウィジェットクラスポインタを入力し、その後「追加」をクリックします。完了するためには、733 ページの「ウィジェットの属性」に記述されているように、クラスの属性を指定します。

ウィジェットクラスリストの編集

ファミリからウィジェットクラスを削除するには、まずリストからそのウィジェットクラスを選択し、「削除」をクリックします。リストを並べ替える場合は、項目を選択し、矢印ボタンを使用して項目を上下に移動します。リストの順序によって、ウィジェットパレット内のオプションメニューの順序が決定されます。

ウィジェットの属性

ウィジェットクラスの属性を指定する場合は、まず、 ファミリ編集ダイアログでウィジェットクラスを選択し、その後「編集」をクリックします。すると、ウィジェット編集ダイアログ (図 23-3) が表示されます。タイトルバーには編集中のウィジェットクラスの名前が表示されます。

この節は、 ウィジェット編集ダイアログの左側にある一連の属性について説明します。ダイアログの右側は、既存または新しいページにリソースを割り当て、ウィジェットリソースにカスタムポップアップを指定し、デフォルトリソースメモリー管理を書き換えるために使用します。詳細は、740 ページの「リソース」を参照してください。

変更の適用

ウィジェットの属性の入力が終了したら、「適用」をクリックして新しい値を設定します。「元に戻す」をクリックすると、最後に適用された変更に戻ります。

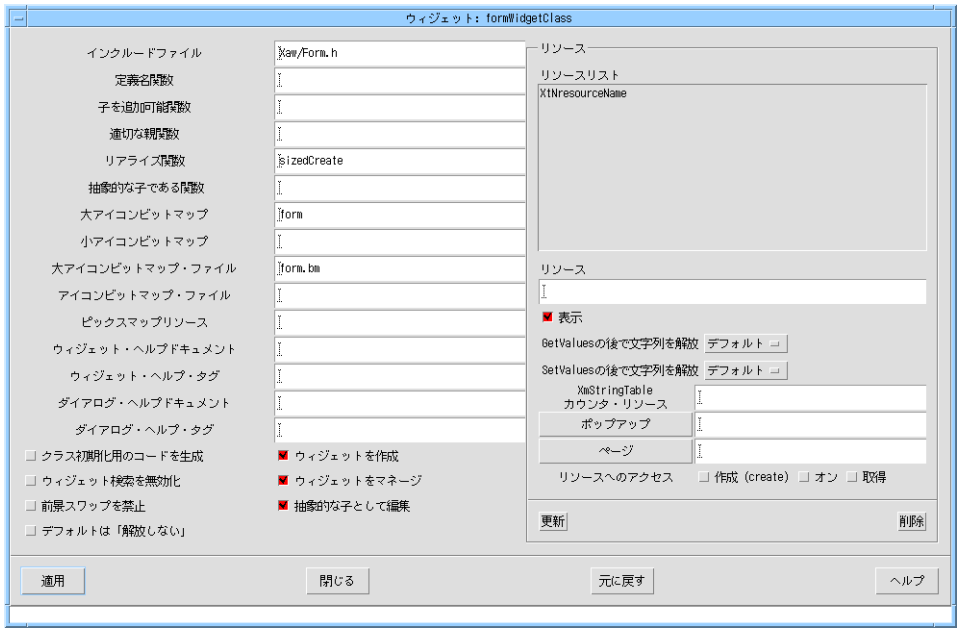


図 23-3 Athena Form の属性を示すウィジェット編集ダイアログ

インクルードファイル

「インクルードファイル」フィールドには、ウィジェットクラスに対しての公開ヘッダーファイルの名前を指定します。ファイル名 (通常は `/usr/include` に対する相対指定) には、引用符や括弧を使用しないでください。このファイルは、 visu_config によって生成されるコードファイルと Sun WorkShop Visual によって生成されるアプリケーションコードの 2 箇所に組み込まれます。

アイコン

アイコンは、Sun WorkShop Visual ウィジェットパレットおよびデザイン階層内にあるユーザー定義ウィジェットを表わす X ピックスマップまたはビットマップです。各ウィジェットに対してピックスマップとビットマップの両方を指定することができます。アイコンピックスマップは別のファイルに保存されており、Sun WorkShop Visual リソースファイル中の設定内容によって指定されます。

アイコンビットマップは Sun WorkShop Visual に組み込まれており、ピックスマップリソースが設定されていない、あるいはピックスマップファイルが見つからない場合にのみ使用されます。どちらの形式のアイコンも指定しない場合、あるいは指定されたアイコンが見つからない場合、Sun WorkShop Visual はウィジェットクラス名を中に表示したボタンをウィジェットパレット内に、×印のついた四角形のアイコンを階層内に表示します。

使用する色を減らすために、Sun WorkShop Visual がアイコンに対して使用するカラーパレットを使用することをお勧めします。「パレット」メニューの「パレットを読み取り」を選択して、以下のファイルをピックスマップ・エディタに読み込んでください。

```
$VISUROOT/lib/palettes/icon.xpm
```

`$VISUROOT` は Sun WorkShop Visual のインストールディレクトリです。ピックスマップ・エディタへのパレット読み込みについては 177 ページの「ピックスマップ・エディタのパレットメニュー」を参照してください。

ピックスマップリソース

「ピックスマップリソース」フィールドは、ピックスマップリソースの名前を指定するために使用します。Sun WorkShop Visual を構築した後、このリソースを設定してピックスマップファイルを指定することができます。

たとえば、ユーザー定義ウィジェットのピックスマップリソースとして *myWidgetPixmap* を指定する場合は、以下のエントリを使用して Sun WorkShop Visual リソースファイルにピックスマップを指定することができます。

```
visu.myWidgetPixmap: /usr/local/newwidget.xpm
```

これにより、XPM ファイルの位置として `/usr/local/newwidget.xpm` が指定されます。パス名は、絶対あるいは相対のどちらでも使用することができます。ピックスマップリソースの使用方法についての詳細は、807 ページの「パレットアイコン」を参照してください。

ビットマップ

組み込みアイコンビットマップを指定するためには、ビットマップ名と対応するビットマップファイル名の 2 項目を指定する必要があります。大型画面 (ワークステーション) 用の Sun WorkShop Visual に対してアイコンを指定する場合は、「大アイコンビットマップ」および「大アイコンビットマップ・ファイル」フィールドを使用し

ます。小型画面 (VGA) 用の場合は「小アイコンビットマップ」および「小アイコンビットマップ・ファイル」フィールドを使用します。アイコンは、大型画面用または小型画面用のどちらか一方だけを指定することも、両方指定することも、どちらにも指定しないでおくことも可能です。

大型画面アイコンは 32×32 ピクセルの X ビットマップ、小型画面アイコンは 20×20 ピクセルにします。アイコンビットマップは X11 ビットマップユーティリティなどのツールを使用して作成することができます。ピックスマップは使用できません。

ビットマップ名を「大アイコンビットマップ」または「小アイコンビットマップ」フィールドに入力します。ビットマップ名は、以下に示すようにビットマップファイルの最初の行の定義中に見つけることができます。

```
#define <ビットマップ名>_width 32
```

アイコンファイル名を「大アイコンビットマップ・ファイル」または「小アイコンビットマップ・ファイル」フィールドに、引用符や角括弧を使用せずに入力します。このファイルは Sun WorkShop Visual の組み込み時に読み出されるため、Sun WorkShop Visual メークファイルはそのファイルが保存されているディレクトリを参照しなければなりません。なお、エンドユーザーがこのファイルを使用できるようにする必要はありません。

ヘルプ

上述のような属性を使用して、ユーザーがパレット上で、あるいはウィジェットのリソースパネルにおいて、ウィジェットに対してのヘルプを呼び出した際に表示されるオンラインヘルプを指定することができます。それぞれのヘルプ項目に対して、(接尾辞を付けない) ドキュメント名を、該当するウィジットのヘルプが大きなドキュメント中に含まれている場合はタグを指定します。Sun WorkShop Visual ヘルプシステムが、Sun WorkShop Visual.*helpDir* リソースで指定されているパスのリストに従って、該当するファイルを検索します。ファイル名には接尾辞 `.html` が必要です。タグとは、ドキュメントの HTML ハイパーテキストを構成するアンカー記号を指します。

構成関数

構成関数を指定するフィールドとして、「定義名」関数、「子を追加可能」関数、「適切な親」関数および「リアライズ関数」の 4 つの関数があります。これらの関数は Sun WorkShop Visual のダイナミックディスプレイにおけるウィジェットの処理方法を微調整するために使用することができます。ウィジェットが構成関数を使用する場合は、対応するテキストフィールドに関数名をタイプ入力します。

構成関数は以下の処理を実行します。

- 定義名関数
  リソース名を適切な定義名に翻訳します。ウィジェットが標準の Motif 命名規約に従わない場合に限り必要です。
- 子を追加可能関数
  あるウィジェットの子として、他のクラスのウィジェットを追加できるかどうかを決定します。
- 適切な親関数
  他のクラスのウィジェットが、あるウィジェットに対して適切な親であるかどうかを決定します。
- リアライズ関数
  ウィジェットが、ダイナミックディスプレイにおいて作成される際の初期化手順を追加します。そのウィジェットに子がない状態ではうまく作成またはマネージできない場合、あるいは作成時にリソースの設定を要する場合にのみ必要です。

「子を追加可能」および「適切な親」関数は、有効な階層について特別な条件を持つウィジェットに対してのみ必要となります。多くの場合、これらの関数は必要ありません。これらの関数を指定しない場合、Sun WorkShop Visual はウィジェットクラスの最初に認識した祖先に対しての規則を使用します。たとえば、ユーザー定義ウィジェットが Primitive クラスから派生している場合、Sun WorkShop Visual は Primitive クラスの規則を使用するため、ユーザーによるウィジェットへの子の追加は許可しません。

構成関数の完全な定義、同義語および例については、773 ページの「構成関数」を参照してください。多くのウィジェットは構成関数を必要としないので注意してください。

その他のトグルボタン

ダイアログの下部に、ウィジェットの使用に関する各種オプションのトグルボタンがあります。各ボタンの詳細については、以下で説明します。「ウィジェット編集」ダイアログの「リソース」パネルのトグルについては、740 ページの「リソース」を参照してください。

クラス初期化用のコードを生成

初めてウィジェットを作成するときは、通常、指定のウィジェットインスタンスとは反対に、ウィジェットの特定のクラスに関連付けられた初期化が行われます。

ウィジェットセットによっては、指定のウィジェットを初めて作成するまでにこの初期化を行わないと問題が発生することがあります。ウィジェット間に暗黙的な依存関係がある場合は、指定のウィジェットセットに含まれるウィジェットを作成する前に、ウィジェットクラスを手作業で確実に初期化してください。このような場合、たとえば、テーブルウィジェットに子としてテーブルラベルウィジェットを指定できますが、テーブルのコードによっては、ラベルクラスが初期化されたと見なすことがあります。このため、ラベルをテーブルにアタッチしないときでも、テーブルを使用する前にラベルクラスを初期化しておく必要があります。

このトグルをオンにすると、Sun WorkShop Visual は、作成コードを呼び出す前に、メインモジュールに次の行を生成します。

```
XtInitializeWidgetClass(widget_class)
```

このトグルはデフォルトでオンになっています。そのため、クラスを事前に初期化していなくても問題は発生しませんが、ウィジェットによっては問題が発生することもあります。

ウィジェット検索を無効化

ウィジェットのスピード検索機能を使用しない場合には、このトグルをオフにします。ダイナミックディスプレイでスピード検索機能が正しく動作しない場合は、このオプションをオフにしてください。

前景スワップを禁止

通常、Sun WorkShop Visual はダイナミックディスプレイにおいて現在選択されているウィジェットを強調表示します。ウィジェットクラスに対しての強調表示をやめるには、「前景スワップを禁止」トグルをオンにします。この操作は、ウィジェットクラスが、ウィジェット作成後に設定できない前景リソースを持っている場合に限って行います。Sun WorkShop Visual の強調表示の方法は、このようなウィジェットに問題を生じさせる可能性があるため、「前景スワップを禁止」をオンにする必要があります。他の場合には、トグルをオフにしておきます。

デフォルトは「解放しない」

Sun WorkShop Visual が XmString と String(char *) ウィジェットリソースのメモリーを管理する方法のデフォルトを設定できます。このトグルの使用方法も含めて、詳細は、763 ページの「リソースメモリー管理」を参照してください。

ウィジェットを作成

「ウィジェットを作成」トグルがオンである場合、ユーザーは階層に直接ウィジェットを組み込むことができます。ウィジェットのアイコンをパレットに表示しようとする場合には、トグルをオンのままにしておきます。

ウィジェットクラスをコアウィジェットのような目に見えないスーパークラスにする場合は、トグルをオフにします。トグルをオフにすると、ウィジェットはウィジェットパレットに表示されず、ユーザーはそのインスタンスを直接作成することができません。Sun WorkShop Visual はこのクラスに対しては別のリソースパネルを作成します。リソースパネルには任意の派生ウィジェットクラスからアクセスすることができます。

ウィジェットをマネージ

すべてのウィジェットが GUI の構成要素というわけではありません。たとえば、INT ChartObject ウィジェットセットの各種データオブジェクトクラスは、MVC (Model View Controller) モデルではグラフの構成要素 (円、方形、軸など) を表す場合に使用します。この場合、これらのオブジェクトを直接描画することはしませんが、グラフの親ですべてを内部処理することができます。

Sun WorkShop Visual には管理できる事項と管理できない事項に関する内部ルールがあります。そのルールによると、WidgetClass 以外の ObjectClass や派生クラスを管理することは、ツールキットでは禁止されており、こうしたクラスを管理すると、クラスが破壊されてしまいます。一方、ウィジェットセットのデータオブジェクトには、WidgetClass から派生したものもありますが、こうしたオブジェクトが管理可能とは限りません。Sun WorkShop Visual は、こうした例外を認識できません。つまり、オブジェクトの管理に関するルールは、内部のオブジェクトに対しても生成されたオブジェクトに対しても正しく適用されない可能性があります。非常にまれなケースですが、構成要素の Sun WorkShop Visual による管理の中止が必要な場合に、このトグルを使用します。

抽象的な子として編集

複合ウィジェットの抽象的な子に対してコードを表示、構成、生成する機能です。複合ウィジェット MotifScrolledWindow の場合は、スクロールバーが抽象的な子となります。ウィジェットが複合ウィジェットで、ユーザーが抽象的な子にアクセスできないようにする場合は、このトグルをオフにします。Sun WorkShop Visual で他社の複合ウィジェットの抽象的な子にアクセスする方法の詳細については、769 ページの「抽象的な子のアクセス」を参照してください。

リソース

デフォルトでは、Sun WorkShop Visual がウィジェットのリソースダイアログを作成する場合、リソースの名前および型を直接ウィジェットコードから獲得し、リソースパネルを構築します。Sun WorkShop Visual は各リソースをリソース型に基づいて、リソースパネルのページに割り当て、適当な型にポップアップダイアログを割り当てます。たとえば、*XtRPixel* 型のリソースは、Sun WorkShop Visual カラーエディタをポップアップさせるボタンとともに、リソースパネルの「表示」ページに置かれます。

このデフォルト動作を変更する場合、あるいはリソースが標準型の表にリストされていない場合を除き、リソース構成のために何かを行う必要はありません。

標準リソース型のデフォルト処理

標準型のリソースは、以下の表に示すようにリソースパネルのページに割り当てられます。

表 23-1 Sun WorkShop Visual 標準リソース型

リソース型記号	値	ページ
XmRDimension	“Dimension”	マージン
XmRFontStruct	“FontStruct”	表示
XmRHorizontalDimension	“HorizontalDimension”	マージン
XmRInt	“Int”	マージン
XmRPixel	“Pixel”	表示
XmRPixmap	“Pixmap”	表示
XmRPosition	“Position”	マージン
XmRPrimForegroundPixmap	“PrimForegroundPixmap”	表示
XmRShort	“Short”	マージン
XmRString	“String”	表示
XmRVerticalDimension	“VerticalDimension”	マージン
XmRWidget	“Widget”	表示
XmRXmString	“XmString”	表示
XtRBoolean	“Boolean”	設定
XtRCallback	“Callback”	コールバック
XtRUnsignedChar	“UnsignedChar”	設定
XmRFontList	“FontList”	表示
XtRFontStruct	“FontStruct”	設定
XmRXmStringTable	"XmStringTable"	表示

ウィジェットの属性の変更

ウィジェット編集ダイアログの右側を使用すると、個々のウィジェットリソースに対して以下のような動作を行うことができます。

- リソースがリソースパネル上に表示されないようにする

- リソースをリソースパネルの選択されたページに割り当てる
- リソース設定用のポップアップダイアログを指定する
- リソースのメモリー管理を制御する
  763 ページの「リソースメモリー管理」を参照してください。
- ウィジェットのリソースの「CSG」(Create/Set/Get: 作成/オン/取得) アクセス可能性を制御する

リソースの属性をカスタマイズするためには、リソースフィールドにリソース名を入力します。この場合のリソース名は、 *XtNcursorName* のような定義名とします。

リソースパネルからリソースを取り除くには、「表示」トグルをオフにします。

リソースパネルの異なるページを指定する場合は、「ページ」ボタンをクリックして現在のページのリストを表示します。リストは、デフォルト Sun WorkShop Visual ページにあらかじめ設定されています。ページを追加するには、「ページ」フィールドに新しいページの名前を入力し、「更新」をクリックします。現在のリソースをページに設定するには、リストにおいてページを選択し、その後「適用」をクリックします。

visu_config は、次の表に示されている、特定のリソース型に対して Sun WorkShop Visual 事前定義ポップアップを自動的に呼び出します。これらの型のリソースに対してのポップアップを使用するために、ポップアップを明示的に指定する必要はありません。

表 23-2 標準リソースポップアップ

リソース型	ポップアップ
XmRFontList	フォント選択パネル
XtRFontStruct	フォント選択パネル
XmRPixel	カラー選択パネル
XmRPixmap	ピックスマップエディタ
XmRPrimForegroundPixmap	ピックスマップエディタ
XmRXmString	コンパウンド文字列エディタ
XtRCallback	コールバックダイアログ

ポップアップダイアログは、どのようなリソースにも指定することができます。事前定義されているポップアップを指定、あるいは独自のポップアップを作成することが可能です。詳細は、755 ページの「ポップアップ」を参照してください。

「カウンタ・リソース」テキストフィールド

リソースによってはペアになっているものがあります。XmString の配列が Motif リストに渡されると、配列内の文字列数を指定する必要があります。たとえば、次のコードで、アプリケーションはコアダンプします。

```
XmString *xmstrings = ... ;

XtVaSetValues(list, XmNitems, xmstrings, NULL) ;
```

配列リソースには、NULL で終わっているリストのように、配列の終わりを示すマーカーがありません。そのため、次の例で示すように、明示的にカウントを指定する必要があります。

```
int n = ... ;

XmString *xmstrings = ... ;

XtVaSetValues(list, XmNitems, xmstrings, XmNitemCount, n, NULL) ;
```

「XmStringTable カウンタ・リソース」テキストフィールドは、配列と配列カウンタ・リソースを組み合わせる場合に使用します。その結果、Sun WorkShop Visual で、配列と配列カウンタ・リソースを組み合わせて内部使用でき、さらに、一度に両方のリソースを使用して正確なコードを生成できます。

このテキストフィールドは、ペアになっているカウンタ・リソースを必要とする XmString 配列リソースと char ** 配列リソース用です。

作成/オン/取得

ウィジェットのリソースのアクセス可能性を制御できる「作成」、「オン」、「取得」という 3 つのトグルがあります。これらのトグルは Motif ドキュメントの「CSG」に直接関連しています。「オン」トグルをオフにすると、Sun WorkShop Visual と Sun WorkShop Visual が生成したコードでは、リソースは読み取り専用になります。「取得」トグルをオフにすると、ユーザーのコードでリソースの値をフェッチすることはできません。「作成」トグルをオフにすると、リソースをユーザーが作成しなければなりません。

他社のウィジェットに関するドキュメントでは、「CSG」(Create/Set/Get) の意味がここでの説明と異なっていることがあります。その場合は、このトグルの機能をオフにします。次のようにリソースを設定してください。

```
visu.xwResourceAccessControl: false
```

非標準リソース型

デフォルトでは、表 23-1 の Sun WorkShop Visual 標準リソース型以外のリソース型は、リソースパネルの「その他」のページに示されています。ユーザーは、テキストフィールドに文字列を入力することによって非標準のリソース型を設定することができます。文字列は、ユーザーが入力したとおり正確にコード生成されます。生成コードがコンパイルされる際に、リソース設定が有効になります。

このデフォルト動作には好ましくない点も含まれています。Sun WorkShop Visual はリソース型を識別しないため、ダイナミックディスプレイにおいてリソースを設定することができません。列挙型値が正確に大文字と小文字を区別して (接頭辞も含む) 入力されない場合は、生成コードのコンパイルは行われません。

visu_config では、Sun WorkShop Visual で非標準リソース型の処理方式を簡便化する方法をいくつか提供しています。

- 標準型のように動作するリソース型に対して別名を指定できる
  ある非標準リソース型が *XmRInt* と同様に動作する場合は、Sun WorkShop Visual にそのリソースを *XmRInt* として扱うように指示することができます。

- Sun WorkShop Visual が、*XtRBoolean* ではなく列挙型を処理するように構成できる
  このような構成を行うと、リソースパネルの「設定」ページに列挙型リソースが配置されます。ユーザーは、オプションメニューを使用してこれを設定することができます。ダイナミックディスプレイ上で、入力を誤ることなくリソースを有効にすることができます。

- その他の型に対しては、コンバータを指定できる
  コンバータは、Sun WorkShop Visual にダイナミックディスプレイのリソース設定を実行させます。

- どのようなリソースに対してもポップアップダイアログを指定することができる
  ポップアップダイアログを持つリソースのリソースパネル上には、ダイアログを呼び出すためのプッシュボタンが作成されます。

以下の節では、これらの手順についての詳細な説明を行います。

別名

ある定義ウィジェットが、標準リストに含まれてはいないものの、標準型と同様の意味を持つリソースを使用する場合は、visu_config にこのリソース型が標準型の別名であることを指定することができます。たとえば、型 *XtRDegrees* を 0 から 359 までの整数として定義する場合、 *XtRDegrees* が *XmRShort* と等価であることを指定する別名を設定することができます。その後、リソースパネルの「マージン」ページに *XtRDegrees* リソースを設定すると、即座にダイナミックディスプレイで結果を確認することができます。

必要条件

新しいリソース型は、標準型と同じ意味を持っている必要があります。特に、テキスト文字列のリソース値への変換と逆変換のしかたは、両方の型に対して同じでなければなりません。

別名の指定

ファミリ編集ダイアログにおいて、表示メニューをプルダウンし、「別名」を選択します。すると、図 23-4 に示されるダイアログが表示されます。

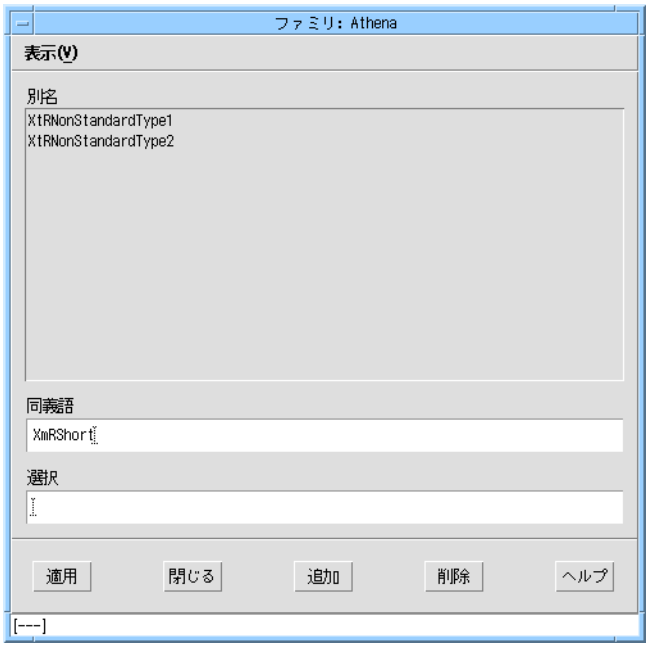


図 23-4 ファミリ編集ダイアログの「別名」ページ

別名を指定するためには、標準以外のリソース型を「選択」フィールドに入力し、それに対応する標準型の名前を「同義語」フィールドに入力します。その後「適用」をクリックして別名を登録します。

列挙型

列挙型リソースは、有効な値が限られているリソースです。ウィジェットが列挙型リソースを持っているかどうかは、そのウィジェットの資料を調べると判断することができます。資料にリソースに対してのすべての可能値がリストされている場合、それは列挙型であると判断できます。型 *XtRBoolean* の列挙型は自動的に処理されるため、この節で説明する構成手順は必要ありません。

デフォルトでは、Sun WorkShop Visual はその他すべての列挙型を、未知のリソース型を扱うように処理します。列挙型リソースはリソースパネルの「その他」ページに配置されます。テキストフィールドに新しい値を入力すると、それらの列挙型が設定されます。設定は生成コードに渡されますが、ダイナミックディスプレイには反映されません。

以降の指示に従って、Sun WorkShop Visual に必要な情報としてリソースパネルの「設定」ページのリソースに関するオプションメニューを構築し、ダイナミックディスプレイにリソースを設定してください。

列挙型の構成

ファミリ編集ダイアログの「表示」メニューから、「列挙型」を選択し、「列挙型」のページを表示します。図 23-5 には、 Athena の Primitives ファミリの列挙型リストを示します。

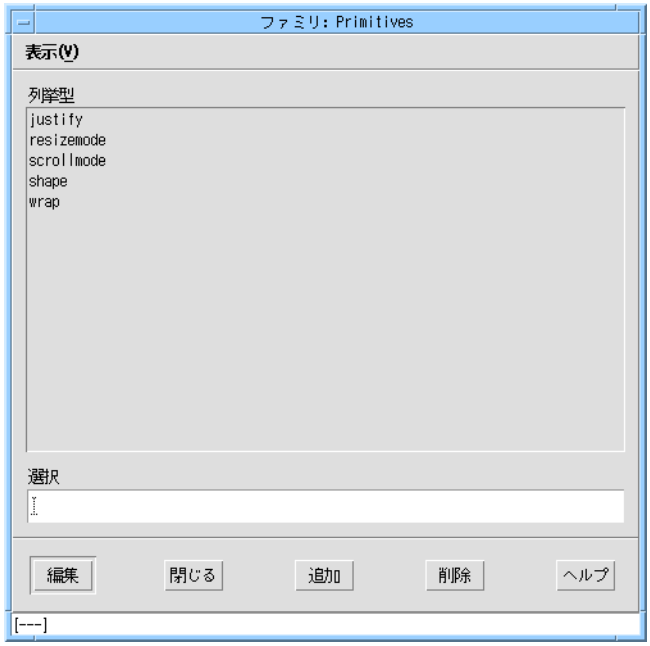


図 23-5 ファミリ編集ダイアログの「列挙型」ページ

新しい列挙型を追加するためには、「選択」フィールドに名前を入力した後、「追加」をクリックします。この名前は、visu_config における便宜上の名前にすぎません。リソース名はウィジェット資料にあるもの、あるいはリソースを特定できる文字列ならどのようなものでも使用することができます。そのファミリに対しての列挙型は、すべてが 1 つのリストに保持されるため、リソース名とともにウィジェット名を含めるとよいでしょう。

列挙型値の構成

処理を完了するには、列挙型リソースについての以下の情報を指定する必要があります。

- リソースの種類
- 可能値のリスト
- デフォルト値
- 各値に対してのコード記号
- 各値に対してのリソースファイル記号

コード記号およびリソースファイル記号の獲得については、751 ページの「リソースファイル記号の獲得」を参照してください。列挙型を構成する場合は、列挙型リストで選択を行い、「編集」をクリックします。すると、図 23-6 に示す列挙型エントリダイアログが表示されます。タイトルバーには列挙型の名前が表示されます。

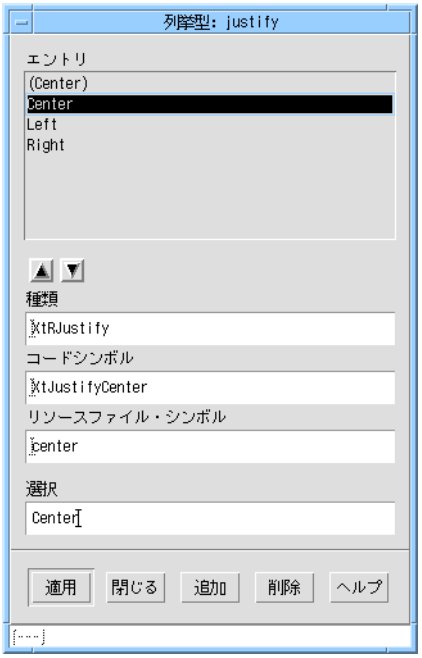


図 23-6 列挙型エントリダイアログ

リソース型を指定し、以下の説明に従って各値を構成します。構成の終了後、「適用」をクリックして新しい値を設定します。

種類の指定

「種類」フィールドにリソースの種類を入力します。

値の指定

列挙型の値のリストを獲得するには、ウィジェットの資料または公開ヘッダーファイルを調べます。それぞれの可能値に対して「エントリ」リストにエントリを作成します。リスト内のエントリは、リソースパネルのオプションメニューでしか使用されないため、どのような名前でも指定することができます。Sun WorkShop Visual は、*XmHORIZONTAL* ではなく *Horizontal* のように、意味のわかる名前を使用します。

各値に対して、「選択」フィールドに名前を入力し、「追加」をクリックします。値の指定を省略しても致命的なエラーとはなりませんが、ユーザーは値を設定することができなくなるので注意してください。

値の構成

各値に対して、列挙型エントリのダイアログでコード記号およびリソースファイル記号を指定する必要があります。コード記号は、生成コードの値を表わす記号定数です。また、リソースファイル記号はリソースファイルの値を表わす文字列です。

コード記号は、ウィジェットの資料の値リストあるいはウィジェット・ソースコードを調べて見つけます。コード記号は「コード記号」フィールドに入力します。

値に対してのリソースファイル記号は、「リソースファイル記号」フィールドに入力します。多くの場合、リソースファイル記号は、コード記号を小文字にして接頭辞を取り除いたものですが、これ以外の場合もあります。たとえば、*XtJustifyCenter* のリソースファイル記号は *center* です。

リソースファイル記号は、常にウィジェット資料にリストされているとは限りません。資料に記載がない場合、例として挙げられているリソースファイル、あるいはウィジェットの開発元から入手することができます。リソースコンバータに対してのソースコードを持っている場合、コードからリソースファイル記号を獲得することができます。この方法については、751 ページの「リソースファイル記号の獲得」で説明します。

デフォルト値の指定

「列挙型」ページの値リストの最初のエントリは、デフォルト値に予約されています。このエントリは、明示的に設定されていないリソースを示すために使用されます。オプションメニューには、同じ値を明示的な名前で設定する必要があります。規約では、以下のリストに示すように、デフォルト値はその名前を括弧で囲むことによって識別します。

```
(Center)
Center
Left
Right
```

対応する明示的な値に対しては、同一のコード記号とリソース記号を指定します。

Sun WorkShop Visual は、列挙型に対してのオプションメニューをひとつ作成することに注意してください。ウィジェットが同一の列挙型のリソースを複数持っている場合、それらのリソースは同じオプションメニューを共有します。リソースが異なるデフォルト値を持っている場合は、一般的なデフォルト値 (*Default*) をオプションメニューに入力することをお勧めします。

デフォルト値の 1 つに対して、コードおよびリソース記号を入力します。これは、生成コードにおいて、あるいはウィジェットが最初にダイナミックディスプレイで作成される場合には、エラーにはなりません。ユーザーがこのリソースを明示的に設定し、その後で明示的にデフォルト値を要求する場合には、ダイナミックディスプレイは正確に動作しない場合があります。しかし、このような問題はウィジェットをリセットすると解消します。

エントリの順序の指定

「列挙型」ページリスト内の順序に従って、リソースパネルのオプションメニューにそれらのエントリが表示されます。デフォルト値は最初にリストされている必要がありますが、その他はどのような順序でもリストすることができます。エントリを異なる位置に移動するためには、リスト内で選択を行い、矢印ボタンを使用して上下に移動します。

リソースファイル記号の獲得

ここでは、 列挙型エントリダイアログに入力するために必要なリソースファイル記号を、ウィジェットのリソースコンバータ・コードから獲得する方法を説明します。リソースコンバータは、リソースファイルから読み出された文字列を、対応する値に変換するために使用される関数です。単純なコンバータには、以下のようなコードの断片が含まれています。

```
if (StringsAreEqual (in_str, "vertical"))
                    i = XmVERTICAL;
else if (StringsAreEqual (in_str, "horizontal"))
                    i = XmHORIZONTAL;
```

この例では、文字列 *horizontal* は値 *XmHORIZONTAL* に、*vertical* は *XmVERTICAL* に変換されます。*horizontal* および *vertical* はリソースファイル記号です。また、*XmHORIZONTAL* および *XmVERTICAL* はコード記号です。

この変換は、以下に示すコードにおいて間接的に確認することができます。

```
if (!haveQuarks) {
                XtQEhorizontal = XrmStringToQuark(XtEhorizontal);
                XtQEvertical = XrmStringToQuark(XtEvertical);
                haveQuarks = 1;
}
XmuCopyISOLatin1Lowered(lowerName, (char *)fromVal->addr);
q = XrmStringToQuark(lowerName);
if (q == XtQEhorizontal) {
                orient = XtorientHorizontal;
                done(&orient, XtOrientation);
                return;
}
if (q == XtQEvertical) {
                orient = XtorientVertical;
                done(&orient, XtOrientation);
                return;
}
```

このコードの場合、リソースファイル記号は記号定数 *XtEhorizontal* および *XtEvertical* で表されています。リソースファイル記号である *horizontal* と *vertical* を獲得するためには、以下のような行の関連ヘッダーファイルを調べます。

```
#define XtEhorizontal "horizontal"
#define XtEvertical "vertical"
```

コードファイル記号である *XtorientHorizontal* と *XtorientVertical* は、変換の最終結果です。この例においては、文字列がクォークに変換され、そのクォークが比較に使用される、という間接的なステップがあることに注意してください。変換関数のしくみは、visu_config に影響しません。

コンバータ

ユーザー定義ウィジェットが列挙型以外の非標準型のリソースを持っている場合、Sun WorkShop Visual はデフォルトでそれらのリソースをリソースパネルの「その他」のページに配置します。ウィジェット編集ダイアログを使用して、これらを別のページに割り当てることができます。テキストフィールドに文字列を入力して、設定できます。デフォルトでは、文字列はコードまたはリソースファイルに生成されますが、Sun WorkShop Visual はダイナミックディスプレイでそれらの設定を行いません。ダイナミックディスプレイでリソースを動作させる場合は、このリソースにコンバータ関数を使用すると、Sun WorkShop Visual を構成することができます。コンバータ関数は、テキスト文字列をリソース値に変換する関数です。

コンバータを構成するには、「表示」メニューから「コンバータ」を選択します。すると、図 23-7 に示されるページが表示されます。「エントリ」リストは、コンバータが指定されているファミリのリソース型のリストを含んでいます。

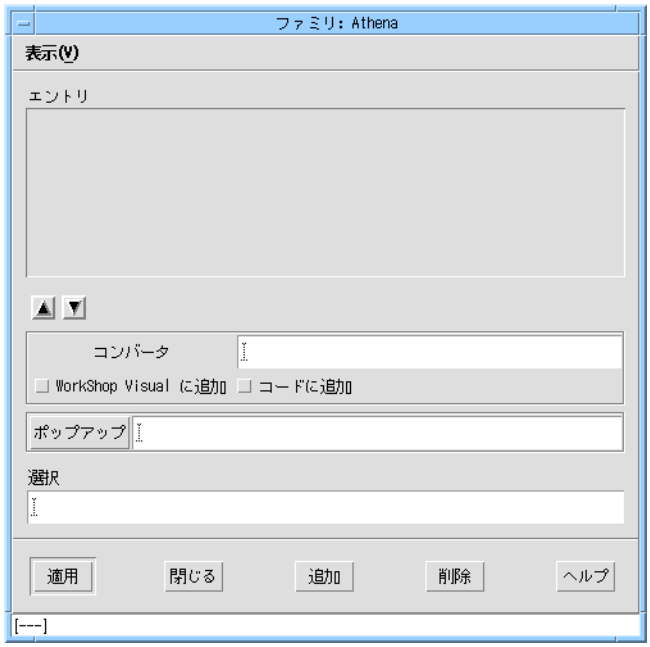


図 23-7　ファミリ編集ダイアログの「コンバータ」ページ

リソース型

「選択」フィールドにリソース型を入力し、「追加」をクリックします。

内部で追加されるコンバータ

多くのウィジェットは、クラスが初期化されると独自のコンバータを内部で追加します。この場合、ユーザーが実行すべき作業は、「コンバータ」ページのリストにリソース型を追加することだけです。リストにリソース型が追加されると、コンバータが使用可能であることが Sun WorkShop Visual に知らされます。これが行われない場合、Sun WorkShop Visual はダイナミックディスプレイにリソースを設定しません。

ウィジェットクラスが独自のコンバータを追加するかどうかを判断するためには、ウィジェットの資料を調べる、あるいはウィジェットのクラス初期化メソッドに対してのコードで *XtSetTypeConverter()* への呼び出しを探します。コンバータが内部で追加されない、あるいはその可能性がある場合には、以下の説明に従って、明示的にコンバータを追加するように、Sun WorkShop Visual に指示します。

明示的に追加されるコンバータ

ウィジェットが内部的にコンバータを追加しない場合、明示的に追加を行うように Sun WorkShop Visual に指示することができます。これを行うには、まず、「コンバータ」テキストボックスにコンバータ名を指定します。コンバータは型 *XtTypeConverter* の関数でなければなりません。「コンバータ」ページには、「Sun WorkShop Visual に追加」(ダイナミックディスプレイで使用) および「コードに追加」(生成コードで使用) の 2 つのトグルボタンがあります。一般に、コンバータを明示的に追加する必要がある場合は、両方のトグルをオンにします。

ポップアップダイアログ

「ポップアップ」ボタンを使用すると、その型のすべてがリソースの設定に使用するためのポップアップダイアログを指定することができます。詳細は、次の「ポップアップ」を参照してください。

ポップアップ

リソースポップアップは、Sun WorkShop Visual のカラーおよびフォント選択パネル等のリソースを設定するために使用するダイアログです。ポップアップを持っているリソースには、通常のテキストフィールドに加え、ポップアップを呼び出すためのボタンがリソースパネル上に作成されます。図 23-8 に、Sun WorkShop Visual コアリソースパネルのポップアップボタンを示します。

図 23-8 コアリソースパネル上のポップアップボタン

個々のリソースに対するポップアップ

ポップアップダイアログは、どのようなリソースに対しても指定することができます。ポップアップダイアログの指定を行うには、図 23-9 に示されているウィジェット編集ダイアログの右側を使用し、リスト内でリソース名を選択します。リストにリソースを追加していない場合は、*XtNresourceName* などのリソースの定義名を「リソース」フィールドに入力し、その後「更新」をクリックします。

図 23-9 ウィジェット編集ダイアログのポップアップ部分

このダイアログは、リソースの種類ではなくリソース名を使用します。したがって、同じ種類の異なるリソースに異なるポップアップを指定することができます。たとえば、型 *XmRInt* の特定のリソースに対してポップアップダイアログを指定する場合、そのポップアップは同一種類の他のリソースに対しては表示されません。

リソース名を入力した後、「ポップアップ」ボタンをクリックしてポップアップダイアログを表示します。754 ページの「ポップアップダイアログ」に記述されている指示に従って、ポップアップを選択します。

リソース型に対するポップアップ

あるリソース型に対してコンバータを指定すると、その型に対してポップアップダイアログを指定することができます。これを行うには、 ファミリ編集ダイアログの「コンバータ」ページにある「ポップアップ」をクリックします。すると、ポップアップダイアログが表示されます。ポップアップの選択は、次の「ポップアップダイアログ」の手順に従って行います。

「ポップアップ」ダイアログ

図 23-10 に「ポップアップ」ダイアログを示します。

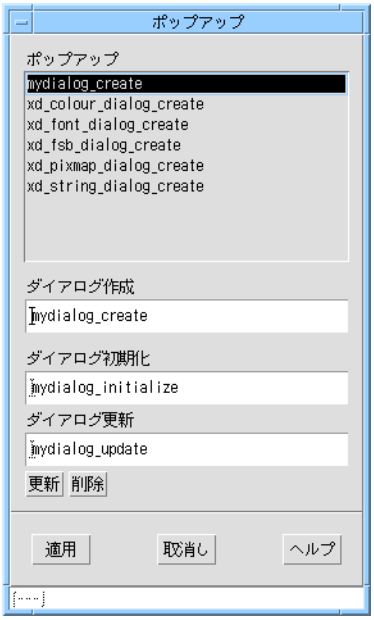


図 23-10 「ポップアップ」ダイアログ

「ポップアップ」ダイアログは、現在のポップアップのリストを表示します。リストは、組み込まれている Sun WorkShop Visual ポップアップ作成関数に事前設定されています。

表 23-3 組み込みポップアップ

名前	ポップアップ
xd_colour_dialog_create	カラー選択
xd_font_dialog_create	フォント選択
xd_fsb_dialog_create	ファイル選択ダイアログ
xd_pixmap_dialog_create	ピックスマップエディタ
xd_string_dialog_create	コンパウンド文字列エディタ

現在選択されている個別のリソース、あるいはコンバータ型に対して既存のポップアップを適用するには、リスト内でポップアップ作成関数を選択して「適用」をクリックします。すべてのポップアップダイアログはリソース値を「*red*」または「*<big_font>*」などの文字列の形式で返すので注意してください。これによって、ポップアップダイアログは文字列として宣言されていても、フォント、色あるいはファイル名を指定するために使用されているリソースに対して動作するのです。

カスタム・ポップアップダイアログ

ポップアップダイアログを独自に作成し、リストに追加してリソースまたはコンバータ型に適用することができます。各ポップアップダイアログには、作成関数、初期化関数、更新関数の 3 種類の関数が必要です。各関数の名前を適切なテキストフィールドに入力し、「更新」をクリックしてポップアップをリストに追加します。

コード要件

ポップアップ関数は、それぞれ以下に示すようなタイミングで呼び出されます。

表 23-4 ポップアップ関数が呼び出されるタイミング

関数	呼び出されるタイミング
作成関数	ダイアログが最初にポップアップされる時
初期化関数	ユーザーがリソースボタンをクリックする時
更新関数	新しいウィジェットが選択される時

新しい値の設定、値の編集、ヘルプの呼び出しおよびダイアログのポップダウン等の追加機能に対してフックを設定するためには、ポップアップダイアログのウィジェットにコールバックを追加します。ダイアログには、少なくとも以下の標準コールバックを持たせます。

表 23-5 ダイアログに追加するコールバック

コールバック	機能
Apply コールバック	ソース・テキストウィジェットに新しいリソースを設定する
Close コールバック	アンマネージしてダイアログをポップダウンする

リソースパネル上のテキストまたはテキストフィールドウィジェット (ソース・テキストウィジェットと呼ばれる) は、ダイアログからリソースパネルに情報を渡すために使用されます。ユーザーがポップアップダイアログに値を設定した場合、値はテキストに変換され、ソース・テキストウィジェットに設定されます。その後ユーザーは、リソースパネル上の「適用」をクリックして、ダイナミックディスプレイに、テキストの入力が行われたように値を設定することができます。

Sun WorkShop Visual でダイアログを構築する、あるいは手作業でダイアログを構築することが可能です。760 ページの「ポップアップ例」では、Sun WorkShop Visualで構築されたポップアップダイアログを示しています。

作成関数

作成関数は、ユーザーが初めてリソースボタンをクリックしてダイアログをポップアップする時に呼び出されます。この関数は、ダイアログに対してのウィジェット階層を作成します。

```
void popup_create (Widget parent)
```

parent 引数は、Sun WorkShop Visual のアプリケーションシェル・ウィジェットで、ダイアログに対しての親ウィジェットとして使用されます。この引数は、*XmCreateDialogShell()* などの関数を呼び出すために必要です。ポップアップダイアログを Sun WorkShop Visual で構築し、Sun WorkShop Visual の生成した作成手続きを作成関数として使用することも、あるいはそれを呼び出す作成関数のコードを書くことも可能です。この場合、Sun WorkShop Visual が生成した関数に *parent* を渡します。

初期化関数は作成関数の直後に呼び出されるため、初期化関数でウィジェットをマネージしている場合には、ウィジェットをマネージする必要はありません。

初期化関数

初期化関数は、ユーザーがダイアログをポップアップするためにリソースボタンをクリックするたびに呼び出されます。ダイアログが初めて呼び出されると、作成関数の後に初期化関数が呼び出されます。関数は、ダイアログをマネージして目に見えるようにし、ダイアログ内のフィールドを初期化します。初期化関数にはソース・テキストウィジェット、現在選択されているウィジェットおよびリソース名が渡されます。

```
void popup_initialize(Widget source_text, Widget current, char
*resource_name )
```

source_text 引数は、リソースパネル上のソース・テキストウィジェットです。このウィジェットは、リソースパネルに現在表示されているリソース値を獲得するために使用できます。ソース・テキストウィジェットは、ダイアログから新しい値を返すために使用されるため、初期化関数では、ソース・テキストウィジェットを静的変数に保存して、後で使用できるようにしなければなりません。

current は、現在選択されているウィジェットを表わします。初期化関数は現在選択されているウィジェットが予期したとおりのリソースを持っているかどうかを検査します。これは、ユーザーが異なる種類のウィジェットを選択した後で、リソースパネルからポップアップダイアログを呼び出すことができるようにするためです。ユーザーが階層からすべてのウィジェットを削除した場合には、*current* は *NULL* になります。

resource_name には、定義名ではなく、リソース名 (「label」等の文字列) を渡します。この引数は、複数のリソースを設定するために同じポップアップを使用する場合に特に役立ちます。

更新関数

更新関数は、ウィジェット階層において選択が変更された後に、すべてのポップアップダイアログに対して呼び出されます。

```
void popup_update( Widget current )
```

current は、新しく選択されたウィジェットを表します。更新関数は、新しく選択されたウィジェットが異なるクラスである場合、あるいは *current* が *NULL* である場合には、「適用」ボタンが応答不能になります。複数のリソースの設定に同一のダイアログを使用する場合、安全性を考慮して、すべてのクラスにおいて「適用」ボタンを応答不可能にすることをお勧めします。そうすると、ユーザーはポップアップを使用するためにリソースボタンを再度クリックしなければなりません。この手順により初期化関数が呼び出され、意図されたリソースが確実に設定されるようになります。

ポップアップ例

この例は、整数値を選択するために使用するスライダを持つ、単純なリソースポップアップを示します。ユーザーがポップアップにおいて「適用」をクリックすると、スライダの値はテキスト文字列に変換され、リソースパネルのテキストウィジェットに配置されます。

ダイアログそのものは、Sun WorkShop Visual で構築しました。そのダイアログのシェルを、データ構造体としてデザインして、以下に示す構造になっています。ダイアログ内にある名前付きのウィジェットは、構造ポインタ *foo_dialog* を使用して簡単に呼び出すことができます。たとえば、*foo_dialog->scale* はスケールウィジェットを呼び出します。

```
typedef struct foo_dialog_s {
                Widget foo_dialog;
                Widget form;
                Widget scale;
                Widget apply;
                Widget close;
} foo_dialog_t, *foo_dialog_p;
static foo_dialog_p foo_dialog = (foo_dialog_p) NULL;
```

以下の関数は、シェルおよびそのすべての子を作成するために生成されます。生成される関数の主要部は省略します。

```
foo_dialog_p create_foo_dialog (Widget parent)
{
/* Sun WorkShop Visual が生成したダイアログ生成コードは省略します。*/
}
```

モジュール・プレリュードにおいては、ソース・テキストウィジェットを保存するために静的変数が作成されます。初期化関数は、ソース・テキストウィジェットを提供します。

```
static Widget source_text;
```

ダイアログには、「適用」ボタンおよび「閉じる」ボタンがあり、それぞれのボタンは活性化コールバックを持ちます。「適用」ボタンは、以下に示すコールバック関数を呼び出します。コールバック関数はスケールの現在値を獲得するとそれをテキスト文字列に変換し、ソース・テキストウィジェットに設定します。これだけではリソースは設定されないので注意してください。ユーザーは、リソースパネルの「適用」をクリックする必要があります。

```
static void
foo_do_apply (Widget w, XtPointer client_data, XtPointer call_data )
{
```

```
                int i;
                char buf[52];
                XmScaleGetValue ( foo_dialog->scale, &i );
                sprintf ( buf, "%d", i );
                XmTextSetString ( source_text, buf );
}
```

ダイアログは「閉じる」ボタンも持っています。このボタンの活性化コールバック関数は、ダイアログのシェルウィジェットの子をアンマネージします。

```
static void
foo_do_close (Widget w, XtPointer client_data, XtPointer call_data )
{
    XtUnmanageChild ( foo_dialog->form );
}
```

作成関数は Sun WorkShop Visual が生成した作成関数を呼び出し、ウィジェット構造体を保存します。

```
foo_create(Widget parent)
{
                foo_dialog = create_foo_dialog( parent );
}
```

初期化関数は、ソース・テキストフィールドからテキストを抽出して整数に変換し、スケールを設定して現在の値を反映させます。この関数は、ソース・テキストフィールドを保存するため、スケールを使用して設定される新しい値をソース・テキストフィールドに適用することができます。また、現在のウィジェットのクラスに基づいて「適用」ボタンを有効化あるいは無効化し、ダイアログを可視状態にします。

```
foo_initialize( Widget text, Widget current)
{
                char *source_value;
                int i;
                source_text = text;
                source_value = XmTextGetString( source_text );
                i = atoi( source_value );
                XtFree( source_value );
```

```
                XmScaleSetValue( foo_dialog->scale, i );
                XtSetSensitive( foo_dialog->apply,
                    current && XtIsSubclass (current,
fooWidgetClass ) );
                XtManageChild( foo_dialog->form );
}
```

更新関数は、現在のウィジェットのクラスに基づいて、「適用」ボタンを有効化あるいは無効化します。

```
 foo_update( Widget current )
{
                XtSetSensitive( foo_dialog->apply,
                    current && XtIsSubclass ( current,
fooWidgetClass ) );
}
```

リソースメモリー管理

Sun WorkShop Visual は、XmString および文字列 (char *) 型のリソースに対してのデフォルトメモリー管理モデルを設定しています。XmString 型リソースの場合、このモデルは、 SetValues および GetValues の両方において、ウィジェットが XmString をコピーする (つまり、アプリケーションは GetValues あるいは SetValues の後に XmString を解放することができる)と仮定しています。

文字列リソースの場合、ウィジェットは文字列を SetValues でコピーしますが、GetValues ではコピーしないことになっています (つまり、アプリケーションは SetValues の後にのみ XmString を解放します)。このモデルに適合しないリソース (典型的なところでは、XmString リソース。これは GetValuesでコピーされません。) を持っている場合には、「リソース」の領域にあるオプションメニューを使用して Sun WorkShop Visual のデフォルト動作を書き換えることができます。XmString が GetValues でコピーされない場合は、ウィジェットの編集ダイアログ上の「GetValues」オプションメニューを「解放しない」に設定します。同様に、ウィジェットによって GetValues でコピーされた文字列リソース (ラベルウィジェット内の XmNmnemonicCharset など) を持っている場合には、「GetValues」オプションメニューを「解放」に設定します。

Sun WorkShop Visual がデフォルトでリソースを解放しないようにするには、「デフォルトは解放しない」トグルをオンにします。

不適切な状況でメモリーを解放すると Sun WorkShop Visual はクラッシュするため、注意してください。メモリーを解放すべき状況で Sun WorkShop Visual がメモリーを解放しない場合は、単にメモリーリークとして蓄積するだけなので、危険度が少ないと言えます。

XmStringTable リソース

Sun WorkShop VisualがXmStringTable リソースを正しく扱うためには、Table のエントリ数をカウントに使用する整数型のリソースも指定しなければなりません。XmStringTable のためリソース指定を追加してください。

ヘッダー

visu_config は、構成ファイルおよびコードファイルの 2 つのコードモジュールを生成します。visu_config では、これらの各ファイルに対してヘッダーリストの指定を行うことができます。ヘッダーの指定には、ファミリ編集ダイアログを使用します。図 23-11 に示すように、表示メニューから「コード統合ヘッダー」を選択すると「コードヘッダー」ページが表示され、「構成統合ヘッダー」を選択すると「構成ヘッダー」が表示されます。

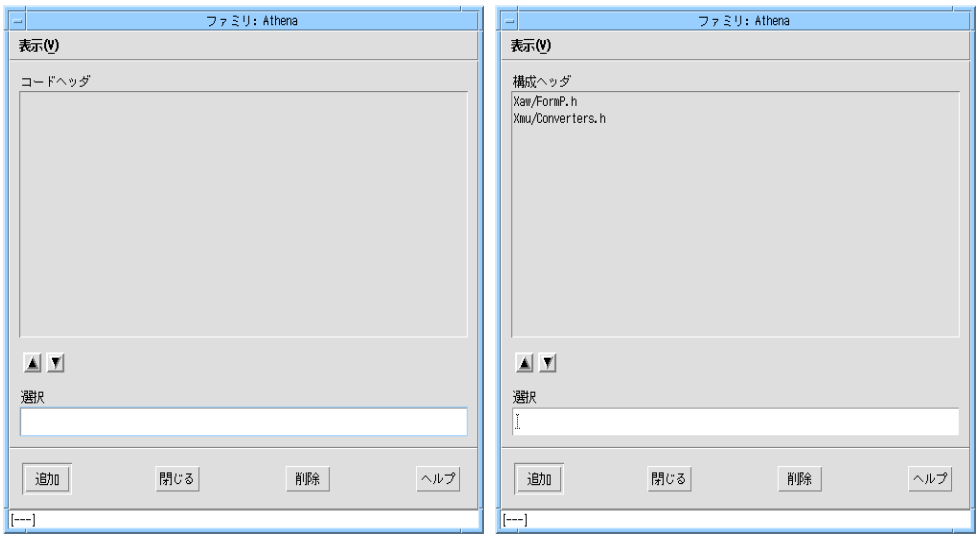


図 23-11 ファミリ編集ダイアログの「コードヘッダー」ページと「構成ヘッダー」ページ

ヘッダーリストは、各ファイル別に存在します。ヘッダーを追加するには、引用符または各括弧を使用せずにファイル名を入力し、「追加」をクリックします。ヘッダーを削除するには、ヘッダーを選択して「削除」をクリックします。リストの順序を変更する場合は、ヘッダーを選択し、矢印ボタンを使用してそのヘッダーを上下に移動します。

コードファイルは、ユーザー定義ウィジェットクラスのウィジェットクラス・レコードを定義します。visu_config は、ウィジェット編集ダイアログから取り出すウィジェットクラス・ヘッダーに対しての *#include* を自動的に生成します。多くの場合、追加のコードヘッダーは必要ありません。

構成ファイルには、ユーザー定義のウィジェット、列挙型および別名のリストがあります。ユーザー定義クラスのウィジェットヘッダーは、このファイルには自動的に生成されません。非標準列挙型またはリソース型を使用して visu_config を構成する場合は、それらが定義されているヘッダーファイルを構成に含めます。

コードを生成してコンパイルすると、簡単に必要なヘッダーファイルを探し出すことができます。コンパイラが未定義のリファレンスを返す場合には、必要な定義を含んでいるヘッダーを探し出し、そのヘッダーをファイルのヘッダーリストに追加します。その後、コードを再生成して再試行します。

Motif ウィジェットの停止リスト

visu_config を使用して、選択された Motif ウィジェットが Sun WorkShop Visual に表示されないようにする (停止する) ことができます。停止となったウィジェットは、ウィジェットパレットには表示されません。既存のデザインファイルから読み出されている場合は、これらのウィジェットは正確に動作しますが、これらを階層で選択したり、あるいは対話的に作成することはできません。たとえば、プログラムを作成する会社の方針に区画ウィンドウウィジェットが適合しない場合には、この機能を使用してユーザーが区画ウィンドウウィジェットを使用しないようにすることができます。

ウィジェットを停止するためには、メイン visu_config ダイアログの編集メニューをプルダウンして「停止リスト」を選択し、図 23-12 に示すようなダイアログを表示します。

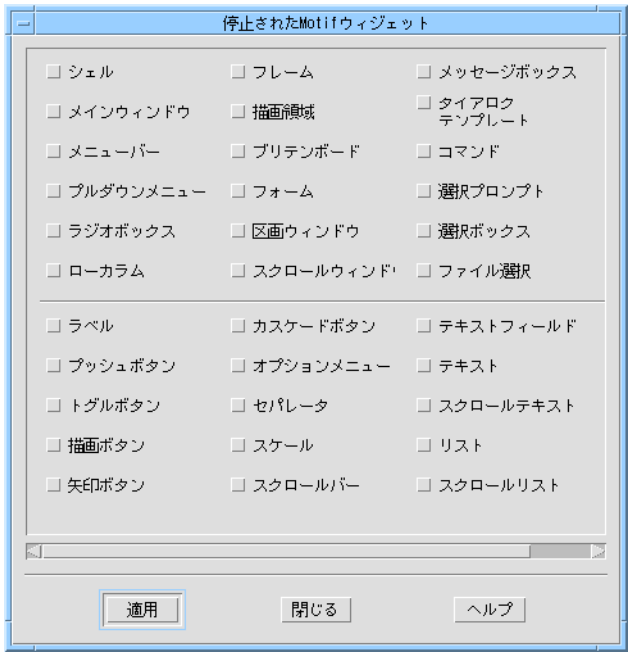


図 23-12 「停止された Motif ウィジェット」ダイアログ

ウィジェットパレットから Motif ウィジェットを取り除くためには、停止させたいウィジェットのトグルを選択して「適用」をクリックします。

visu.*stopList* リソースを設定することによっても、ウィジェットを停止することができます。詳細は、1009 ページの「FrameMaker」を参照してください。リソースを使用して停止したウィジェットは、リソースファイルを使用すれば簡単に活性化することができますが、 visu_config で停止されたウィジェットは、Sun WorkShop Visual を再構築することによってのみ活性化することができます。

ユーザー定義ウィジェットは、このダイアログを使用して停止することはできません。Sun WorkShop Visual で使用できるユーザー定義ファミリのセットは、次の「コードの生成とコンパイル」で説明するように、visu_config の生成ダイアログで選択することができます。

コードの生成とコンパイル

「生成」メニューには、「構成ファイル」、「コードファイル」、「スタブ」という 3 つのオプションがあります。このうち、「構成ファイル」と「コードファイル」のオプションは、2 つの構成ファイルを生成するために使用されます。各オプションで表示されるページは、図 23-13 に示されているようによく似ています。組み込むファミリの選択にはトグルボタンを使用します。それぞれに対して同じファミリセットを使用して、両方のファイルを生成します。

図 23-13 「構成ファイル」および「コードファイル」ダイアログ

コンパイル

`$VISUROOT/user_widgets/Athena` にあるサンプルのメークファイルは、`config.c` という名の構成ファイル、 `Athena.c` という名のコードファイル、そして構成関数を含んでいる `Athenaextras.c` という名のファイルをコンパイルします。これらのファイルおよびすべての Athena アイコンビットマップは、`$VISUROOT/user_widgets/Athena` に配置されています。このメークファイルを使用して Athena の例をコンパイルすることができます。その結果、`visu.`*bin*という実行ファイル が生成されます。このファイルは Sun WorkShop Visual 実行ファイルの構築に使用されます。

他のウィジェットを使用して構成する場合は、出発点として例題のメークファイルを使用してください。メークファイルが、アイコンビットマップ・ファイルおよび必要なヘッダーファイルを含んでいるすべてのディレクトリを確実に参照するようにしてください。

コンパイルに失敗した場合は、生成されたコードを調べて原因をつきとめます。原因がヘッダーの欠落である場合は、ファミリ編集ダイアログの「コードヘッダー」と「構成ヘッダー」ページに、別のヘッダーを追加してください。コンパイラは、visu_config ダイアログのスペルミスも検出します。問題を解決してコードと構成ファイルを再度生成し、コンパイルします。

リンカーはスペルの誤った関数名を検出します。 visu_config においてスペルミスがある場合には、そのミスを正し、コードおよび構成ファイルを再生成して再度コンパイルしてください。

Sun WorkShop Visual でのウィジェットの使用

環境変数 USER_WIDGETS はユーザー定義ウィジェットを含むディレクトリ (この例では `Athena`) に正しく設定して、`$VISUROOT/bin` の標準 `visu` コマンドを実行することで、実行可能ファイルが呼び出されます。これによって、`$XBMLANGPATH` にユーザーの使用するビットマップまたはカラーアイコンを含むディレクトリが追加され、`$XFILESEARCHPATH` の `$VISUROOT/lib/locale/<ロケール>/app-defaults` がユーザーの使用する `app-defaults` によって上書きされます。

`$VISUROOT/user_widgets` ディレクトリに、ユーザーウィジェットのディレクトリへのシンボリックリンク「local」を作成すると、ユーザーすべてに対してデフォルト設定を行うことができます。この場合は、USER_WIDGETS 設定によって明示的に上書きしない限り、すべてのユーザーがデフォルトで指定したバージョンの Sun WorkShop Visual を使用することになります。USER_WIDGETS に認識できない値が含まれていた場合、またはユーザーウィジェットを含むディレクトリで `visu.bin` が構築されなかった場合は、シンボリックリンクによって設定されたデフォルト設定、または標準設定の Sun WorkShop Visual が起動されます。

抽象的な子のアクセス

Sun WorkShop Visual では、他社の複合ウィジェットの抽象的な子への完全なアクセスが提供されています。複合ウィジェット MotifScrolledWindow の場合、抽象的な子はスクロールバーです。

次のリソースで抽象的な子へのアクセスを制御できます。

```
visu.abstractObjects: true
```

リソースはデフォルトで「`true`」です。この場合、他社のウィジェットの子にアクセスできます。「`false`」の場合は、子は表示されません。

visu_config で複合ウィジェットのアクセス可能性を構成できます。詳細は、740 ページの「抽象的な子として編集」を参照してください。

他社のウィジェットの子であっても、リソースパネルを介して完全に構成することができます。ただし、デザインから取り除くことはできません。

生成したコードで他社のウィジェットの構成要素にするために、Sun WorkShop Visual は `XtNameToWidget` を呼び出します。この構成要素には、通常、固有の名前が付けられています。ウィジェットに固有の名前が付けられていない場合に `XtNameToWidget` を使用しても正しく動作しません。このような場合は、生成したコードを編集して、別の方法でウィジェットにアクセスしなければなりません。

Sun WorkShop Visual は、他社のウィジェットの子に対しては、クロスプラットフォームコード (MFC または Java) を生成しません。ルートウィジェットの代わりに、キャンバスなどのプレースホルダーが追加されます。

構成のテスト

ここでは、ユーザー定義ウィジェットに対しての Sun WorkShop Visual インタフェースの推奨テスト手順を説明します。指示通りに visu_config を使用することで問題箇所を修正することができます。これらのテストは、ウィジェットが Sun WorkShop Visual に設定された方法についての問題を検出するためのものです。したがって、ウィジェット自体のテストは行いません。

ウィジェットの作成

注 - このテキストは、ウィジェット属性パネルの「ウィジェットを作成」トグルがオフの場合には適切ではありません。

Sun WorkShop Visual を実行し、ユーザー定義ウィジェットのアイコンが正しく表示されているかどうかを確認します。小画面 Sun WorkShop Visual を使用している場合は、smallxd という名前を使用して Sun WorkShop Visual を起動し、正しいアイコンが表示されていることを確認します。

ユーザー定義ウィジェットのインスタンスを含んでいる階層を作成します。ウィジェットが追加される際に Sun WorkShop Visual を実行できない場合は、リアライズ関数が必要になることもあります。詳細は、737 ページの「構成関数」を参照してください。Sun WorkShop Visual が実行できず、高さ/幅がゼロについての X エラー・メッセージが表示された場合は、ウィジェットのリアライズ関数を *sizedCreate()* (`Athenaextras.c` にある) にして再試行します。

それでも実行できない場合は、visu_config の「前景スワップの使用禁止」トグルを設定します。

前景スワップ

前景スワップを使用禁止にしない場合は、ユーザー定義ウィジェットを含んでいる階層が作成されます。階層内においてシェルを選択し、次にユーザー定義ウィジェットを選択します。ウィジェットを選択すると、そのウィジェットがダイナミックディスプレイにおいて正しく強調表示されることを確認してください。問題が生じた場合は、visu_config の「ウィジェット」ページの「前景スワップを禁止」トグルを設定してください。

定義名

すべてのリソースが設定されているユーザー定義ウィジェットのインスタンスを含んでいるダイアログを作成します。Sun WorkShop Visual から C を生成してコンパイルします。コンパイルに失敗した場合は、以下のようにメッセージが表示されます。

```
XtNfoo undefined
```

このようなメッセージを受け取った場合は、まず、生成コードがウィジェットクラスに対して正しい公開ヘッダーを含んでいることを確認します。正しい公開ヘッダーが含まれていない場合は、visu_config のウィジェット編集ダイアログの「インクルードファイル」フィールドで修正してください。

ヘッダーが正しく生成されたにもかかわらずコンパイルできない場合は、定義名関数が必要です。公開ヘッダーにおいて以下のような行を探してください。

```
#define <something> "foo"
```

<*something*> が *XtNfoo* 以外の場合、定義名関数が必要です。詳細は、737 ページの「構成関数」を参照してください。

ページ

ユーザー定義ウィジェットのリソースが特定のページに表示されるように指定した場合は、その指定が実行されていて、かつ必要なページがすべて存在していることを確認します。

コンバータ

ユーザー定義ウィジェットに対するソースパネルの「その他」ページを表示します。テキストウィジェットに有効なリソース値が入力でき、それらの値がウィジェットに正しく適用されることを確認します。リソースコンバータが存在しないことを示すメッセージを受け取った場合は、「コンバータ」ページの「WorkShop Visual に追加」を設定する必要があります。

列挙型

ユーザー定義ウィジェットに対するリソースパネルの「設定」ページを表示します。オプションメニューが、すべてのメニュー上部に括弧で囲まれたデフォルト値を持っていることを確認してください。

「その他」のページが存在する場合は、そのページを表示します。列挙型リソースが、visu_config において構成されていない場合は、「その他」ページに表示されます。このページに列挙型リソースが表示された場合は、visu_config に戻り、それらのリソースを追加します。

デフォルトを含め、列挙型に対して各値を順番に設定してください。各値がダイナミックディスプレイにおいて意図した通りに動作し、生成コードが正確にコンパイルを行うことを確認します。

ポップアップダイアログ

リソースにカスタム・ポップアップダイアログを指定した場合は、各リソースが表示されるリソースパネルのページを表示します。リソースパネルが、ポップアップダイアログを持つそれぞれのリソースに対してボタンを表示することを確認してください。表示されているボタンをクリックしてみます。ダイアログが表示され、そのリソースの現在の値でダイアログが正しく初期化されていることを確認してください。

ダイアログにリソースを設定します。リソースパネル上のテキストウィジェットが正確に更新されることを確認してください。リソースパネルから設定を適用し、ダイナミックディスプレイで結果を確認します。

ダイアログに「閉じる」ボタンがある場合、それが意図した通りに作動し、リソースパネル上のボタンがクリックされた際にダイアログが再度表示されることを確認してください。

コード検査

最後に、生成コードが正確であることを確認します。生成コードをチェックするには、各リソースを順番に設定して、C コードファイルと X リソースファイルを生成し、それらが意図したとおりのものであることを確認します。

すべての Motif ウィジェットに対してのコード検査は、Sun WorkShop Visual リリースプロセスの一部として行われます。したがって、Motif ウィジェットから派生したユーザー定義ウィジェットを持っている場合は、ユーザーはユーザー定義ウィジェット独自のリソースのテストに集中することができます。また、テスト済みである別のユーザー定義ウィジェットから派生したユーザー定義ウィジェットを持っている場合でも同様です。

構成関数

visu_config は、Sun WorkShop Visual のユーザー定義ウィジェットの処理をカスタマイズするための構成関数の指定を要求します。ここでは構成関数の定義および例を述べます。

構成関数を追加するには、ウィジェットクラスに対する visu_config のウィジェット編集ダイアログに関数の名前を指定した後、visu_config からコードおよび構成ファイルを再生成します。メークファイルを編集して、構成関数に対してのコードを含んでいるファイルをコンパイルしリンクします。

Athena ウィジェットを Sun WorkShop Visual に統合するために使用される構成関数の例については、`$VISUROOT/user_widgets/Athenaextras.c`[1] を参照してください。

1. `$VISUROOT` は、Sun WorkShop Visual のインストールルートディレクトリのパスです。

リアライズ関数

デフォルトでは、Sun WorkShop Visual は *XtCreateWidget()* を呼び出すことによってダイナミックディスプレイにウィジェットを作成しますが、その代わりとしてリアライズ関数を提供することもできます。リアライズ関数は、Sun WorkShop Visual においての作成時に問題が生じるウィジェットに対してのみ必要です。

リアライズ関数のプロトタイプ

リアライズ関数は、以下の形式をとります。この関数は、*XtCreateWidget()* と同じ引数を使用し、同じ結果を返します。

```
Widget realize( char *name, WidgetClass class, Widget parent,
ArgList args, Cardinal arg_count )
```

リアライズ関数に渡される `ArgList` は、常に空です。

リアライズ関数の例

Athena Form ウィジェットのようなコンポジットウィジェットには、作成時にその寸法が明示的に設定されない限り、子なしでは実体化できないものがあります。寸法が明示的に設定されない場合は、ウィジェットはゼロサイズで作成され、X エラーが生じます。ダイナミックディスプレイにおいてこの問題を解消するために、下に示すようなリアライズ関数 (`Athenaextras.c` にあります) を提供することができます。この関数はウィジェットの幅および高さのリソースをゼロ以外の値に初期化して、*XtCreateWidget()* を呼び出し、その結果を返します。

```
WidgetsizedCreate( char *name, WidgetClass class, Widget parent,
ArgList args, Cardinal arg_count )
{
                Arg al[2];
                int ac=0;
                XtSetArg(al[ac], XtNheight, 20); ac++;
                XtSetArg(al[ac], XtNwidth, 20); ac++;
                return XtCreateWidget ( name, class, parent, al,
ac);
}
```

リアライズ関数は、Sun WorkShop Visual がダイナミックディスプレイにウィジェットを作成する際にのみ使用されます。生成コードには反映されません。

定義名関数

コードファイルおよび X11 リソースファイルの両方を生成するために、Sun WorkShop Visual は *label* のようなリソース名と、それに対応する *XtNlabel* のような定義名の両方を使用します。Sun WorkShop Visual は、ウィジェットクラスのレコードから直接名前を取り入れます。デフォルトでは、Sun WorkShop Visual は接頭辞 `XtN` を追加することによって、名前から記号定数を派生させます。

ウィジェットがこの規則に従っていない場合は、 定義名関数を使用して Sun WorkShop Visual を構成することができます。定義名関数は、リソース名を対応する記号定数に変換するカスタム関数です。ウィジェットに定義名関数が必要かどうかを判断するには、そのウィジェットクラスの公開ヘッダーを調べます。ヘッダーファイルは、以下に示すような記号定数とその値を定義する行を含んでいます。

```
#define XtNlabel "label"

#define XtNfont "font"

#define XtNinternalWidth "internalWidth"
```

なんらかの定義名が命名規則に従っていない場合は、定義名関数が必要です。たとえば、Motif を含む多くのウィジェット・ツールキットは、以下に示すようなさまざまな接頭辞を使用しています。

```
#define XmNbuttons "buttons"

#define XmNbuttonSet "buttonSet"

#define XmNbuttonType "buttonType"
```

定義名関数のプロトタイプ

定義名関数は以下の形式をとります。

```
char *defined_name ( char *name )
```

定義名関数に、リソース名を含む文字列が渡されると、対応する定義名を含む文字列が返されます。定義名関数は、Sun WorkShop Visual の内部定義名関数である *def_defined_name()* を参照することができます。この関数は、単純にリソース名にデフォルトの `XtN` 接頭辞を追加します。

注 - 関数名 `defined_name` および `def_defined_name` はすでに Sun WorkShop Visual で使用されているため、これらの名前は使用しないようにしてください。

定義名関数の例

Athena の Clock ウィジェットリソース *hands* の定義名は、*XtNhands* ではなく *XtNhand* です。したがって、Clock ウィジェットは、以下に示す定義名関数を必要とします。

```
char *clock_defined_name( name )
                char *name;
{
                /*
                 * XtNhand は hands として定義されているので単純に
                 *  XtN を前に補足するだけでは不十分です。
                 */
                if ( strcmp ( name, "hands" ) == 0 )
                    return "XtNhand";
                return def_defined_name ( name );
}
```

hands を除くすべての Clock リソースは命名規則に従っているため、それらのリソースの変換には *def_defined_name()* が使用されます。

「子を追加可能」および「適切な親」関数

「適切な親」および「子を追加可能」関数を指定し、ユーザー定義ウィジェットに関する有効な親子関係の規則を定義することができます。これらの規則は、パレットアイコンのグレー表示、新しく作成されたウィジェットの自動選択、および構成領域におけるアイコンのドラッグなどの Sun WorkShop Visual の機能を制御します。

多くの場合、これらの関数は必要ありません。これらの関数を指定しない場合、Sun WorkShop Visual はウィジェットクラスの最初に認識した祖先に対しての規則を使用します。たとえば、ユーザー定義ウィジェットが Primitive クラスから派生している場合、Sun WorkShop Visual は Primitive クラスの規則を使用するため、ユーザーによるウィジェットへの子の追加は許可しません。

「適切な親」および「子を追加可能」関数は、他のウィジェットの規則と組み合わされます。たとえば、Sun WorkShop Visual はすでに、メニューバーウィジェットは子としてカスケードボタンのみを持つことができる、という規則と組み合わされているため、ユーザーがウィジェットをメニューバーの子に指定することを禁止するための「適切な親」関数は必要ありません。ウィジェットクラスに追加の規則がある場合に限り、これらの関数を指定する必要があります。

Motif 以外のコンポジットウィジェットを使用して Sun WorkShop Visual を構成する場合は、ウィジェットクラスに「子を追加可能」関数を指定して Motif の子を持たないようにする必要があります。Motif ウィジェットは、その親が Motif ウィジェットであることを前提としているため、Motif ウィジェット以外の子に指定されると、Sun WorkShop Visual はコアダンプする可能性があります。

「適切な親」関数のプロトタイプ

「適切な親」関数は、ユーザーがユーザー定義クラスのウィジェットを階層に追加しようとする、あるいはユーザー定義のウィジェットを他の親にドラッグまたはコピーしようとする場合に呼び出されます。この関数は、ユーザー定義ウィジェットを選択したウィジェットの子として追加することができるかどうかを判断します。

「適切な親」関数は以下の形式をとります。

```
Boolean is_appropriate_parent ( parent, childclass )

Widget parent;

WidgetClass childclass;
```

最初の引数は、指名された親ウィジェットである、階層内のウィジェットのインスタンスです。2 番目の引数は、新しいウィジェットクラスへのポインタです。指名された親ウィジェットへの新しいクラスの子の追加が有効な場合は、関数は *TRUE* を返し、その他の場合には *FALSE* を返します。

「適切な親」関数には指名された親ウィジェットのインスタンスが渡されるため、親ウィジェットのクラス、あるいは状態に基づいて規則を作成することができます。たとえば、ユーザーがユーザー定義クラスの子を追加する前に、親ウィジェットの寸法、祖先ウィジェット、あるいはその他の子をチェックすることができます。

「適切な親」関数の例

デフォルトでは、Sun WorkShop Visual はフォームおよびローカラムのような Motif マネージャウィジェットにどのような子でも持たせることができます。「適切な親」関数を使用すると、ウィジェットを特定のクラスの子に限定することができます。たとえば、ウィジェットを描画領域の子に限定する場合は、指名された親ウィジェットが描画領域である時には *TRUE* を返し、それ以外は *FALSE* を返す、「適切な親」関数を指定します。この場合のコードは、以下に示すように非常に単純なものです。

```
Boolean drawing_area_parent ( w, class )
Widget w;
WidgetClass class;
{
                if ( XtClass ( w ) == xmDrawingAreaWidgetClass )
                    return True;
                return False;
}
```

「子を追加可能」関数のプロトタイプ

「子を追加可能」関数は以下の形式をとります。

```
Boolean can_add_child (parent, childclass)
XWidget_p parent;
WidgetClass childclass;
{
...
}
```

この関数は、2 つの目的に使用されます。Sun WorkShop Visual は「子を追加可能」関数を呼び出して、特定のクラスのウィジェットが有効なユーザー定義ウィジェットの子であるかを判断します。また、Sun WorkShop Visual は「子を追加可能」関数を呼び出して、新しく作成されたユーザー定義ウィジェットのインスタンスを自動的に選択するかどうかも決定します。

最初の引数は、ユーザー定義ウィジェットクラスの既存のインスタンスへのポインタです。親ウィジェットインスタンスは、ウィジェットインスタンスを表わす内部 Sun WorkShop Visual データ型である *XWidget_s* 構造体に渡されます。この構造体のフィールドの 1 つは、ウィジェットインスタンスへのポインタです。*XWidget_s* 構造体の資料については、`$VISUROOT/user_widgets/hdrs/xwidget.h` を参照してください。

2 番目の引数は、指名された子ウィジェットクラスへのポインタ、あるいは ***NULL*** です。第 2 引数が ***NULL*** 以外である場合、Sun WorkShop Visual は指名された子クラスについての問い合わせを行います。子が追加できる場合には、関数は ***TRUE*** を返し、追加できない場合には ***FALSE*** を返します。子ウィジェットはインスタンス化されていないため、親インスタンスへのポインタはあっても、子に対してのポインタは存在しないので注意してください。関数は、ユーザー定義ウィジェットのインスタンスが現在の状態に基づいて規則を作成することがあります。たとえば、ウィジェットが限られた数の子だけを受け取るようにする関数を作成することができます。第 2 引数が ***NULL*** である場合は、ユーザーはそのクラスのウィジェットしか作成していません。

「子を追加可能」関数が ***TRUE*** を返す場合、Sun WorkShop Visual は階層内に新しく作成されたウィジェットを選択します。それ以外の場合は、親ウィジェットが選択されたままとなります。この場合、「子を追加可能」関数は、ウィジェットがすべての種類の子を持つことができる場合は通常 ***TRUE*** を返し、それ以外は ***FALSE*** を返します。

「子を追加可能」関数の例

以下の例は、「子を追加可能」関数を示しています。

```
Boolean paned_can_add_child (XWidget_p xw, WidgetClass class) {

/* このウィジェットクラスの新しく作成されたインスタンスに対し、新しく作成されたウィジェットを現在階層で選択されているウィジェットにします。*/

if (class == NULL)

                    return TRUE;
```

```
/* 描画領域およびスクロールバーを除くすべての子を許可します。 */
if                ( class == xmDrawingAreaWidgetClass ||
                  class == xmScrollBarWidgetClass )
                  return False;
else
                  return True;
}
```

UIL の生成

他社のウィジェットに対しても UIL を生成することができます。UIL を生成するうえで Sun WorkShop Visual に必要な情報は、ウィジェット統合キット[1] に納められています。新しい他社のウィジェットセットを統合する場合は、Sun WorkShop Visual のリソースに必要な情報をさらに指定する必要があります。

他社のウィジェットの UIL コードを生成するリソース

他社のウィジェットに対して UIL コードを生成するには、ヘッダーファイル、作成関数、テンプレートファイルに関する必要な情報を Sun WorkShop Visual に指定する必要があります[1]。頻繁に使用するウィジェットセットについては、この情報はすでに提供されています。詳細は、Sun WorkShop Visual のご購入先までお問い合わせください。

通常はほとんど使用しないウィジェットセットを使用する場合は、次の情報が必要です。

1. 「UIL ヘッダーファイル」
2. 「UIL 作成関数」

UIL ヘッダーファイル

他社のウィジェットに UIL ヘッダーファイルが関連付けられている場合は、次のように指定します。

1. 使用できるウィジェット統合セットについては、Sun WorkShop Visual のご購入先にお問い合わせください。

```
visu.xw_<他社のウィジェットクラス>.uilHeaderFile: foo.uil
```

たとえば、XRT 3D ウィジェットは次のように定義します。

```
visu*xw_XtXrt3d.uilHeaderFile: Xrt3d.uil
```

注 - UIL ヘッダーファイルは XRT ウィジェットセットに組み込まれています。

UIL ヘッダーファイルに組み込まれていない多くのウィジェットセットでは、Sun WorkShop Visual からの UIL コード生成を可能にするためにヘッダーファイルが作成されています。ヘッダーファイルは次の場所にあります。

```
$VISUROOT/user_widgets/USER_WIDGET_NAME/code_templates/UIL
```

`$VISUROOT` は、Sun WorkShop Visual のインストールディレクトリです。

他社のウィジェットに対して生成された `make` テンプレートには、通常のウィジェットベンダーの位置の他に、この `code_templates` ディレクトリを自動的にインクルードできる UILFLAGS セットがあります。そのため、XRT の場合、メークファイルには次の行が含まれます。

```
UILFLAGS=-I${XRTHOME}/include/Xm ...
```

このようにして、UIL ヘッダー付きの他社のウィジェットが構築されます。

UIL 作成関数

Motif 以外のウィジェットは UIL ヘッダーファイルで作成関数の事前宣言を行う必要がありますが、他社製のすべてのウィジェットでこの宣言が行われているとは限りません。そのため、Sun WorkShop Visual では、UIL を取得して他社のウィジェットを作成する場合に作成関数を制御する、 2 つのリソースを準備しています。次のような場合は、使用するリソースを考慮してください。

1. 指定のヘッダーファイルですでに関数が宣言されている
2. ウィジェットクリエータ名が指定されていない
3. 事前定義されたクリエータが存在しない
4. ウィジェットの作成に特別な処理が必要

注 - UIL では同一関数を複数回宣言することは避けるべきです。したがって、状況がどうであれ、宣言を単純に生成することはできません。そのため、Sun WorkShop Visual が宣言の生成時期を判断できるように、2 つのリソースが用意されています。

上記の 4 点については、以下で詳細に説明します。

すでに関数が宣言されている

ヘッダーファイルですでに関数が宣言されている場合は、Sun WorkShop Visual で生成が必要な名前を次のように指定します。

```
visu*xw_<他社のウィジェットクラス>.uilBuiltinProcedureName:
the_procedure_name
```

たとえば、XRT の UIL クリエータ XtCreateXrt3d が Xrt3d.uil ですでに宣言されている場合、Sun WorkShop Visual には生成する名前を指定するだけです。

```
visu*xw_XtXrt3d.uilBuiltinProcedureName: XtCreateXrt3d
```

生成したUILには、次のようなコードが表示されます。

```
object some_third_party_object: user_defined
                    the_object_creator_procedure_name
```

この XRT の例では、次のコードが生成されます。

```
include_file "Xrt3d.uil";
object my_xrt_3d_variable: user_defined XtCreateXrt3d;
```

ウィジェットクリエータ名が指定されていない

ウィジェットの作成者がウィジェットクリエータ名を指定しなかった場合は、クリエータの呼び出しだけでなく、宣言も Sun WorkShop Visual で生成する必要があります。

次のように指定した場合、

```
visu*xw_<他社のウィジェットクラス>.uilProcedureName:
the_procedure_name
```

次のコードが生成されます。

```
include_file “AnySpecifiedUilHeader.uil”;

/* AnySpecifiedUilHeader.uil にすでに指定されている場合、次の行は無効 */
procedure the_procedure_name();
...
object my_third_party_variable: user_defined
                          the_procedure_name;
```

事前定義されたクリエータが存在しない

事前定義されたクリエータが存在しないということは、下位レベルの汎用 XtCreateWidget( ) 呼び出しを置換する XmCreatePushButton が存在しないのと同じことです。

このような場合、Sun WorkShop Visual は、UIL ファイルに以下のものを生成します。

1. 名前がウィジェットクラスから構成されているクリエータ
2. 宣言
3. C と同じ名前の実関数
   この実関数はラップされます。

生成されるコンストラクタの形式は xdUilCreate<ウィジェットクラス名> です。

UIL のリソースが設定されていない他社のウィジェットを Sun WorkShop Visual で使用すると、このような事態が自動的に発生します。

たとえば、ウィジェットクラス `FredWidgetClass` の場合、UIL ファイルには以下の内容が含まれます。

```
procedure XdUilCreateFredWidgetClass();
...
object fred_variable : user_defined XdUilCreateFredWidgetClass
```

さらに、UIL に対する C には以下の内容が含まれます。

```
...
Widget XdUilCreateFredWidgetClass(parent, name, argv, argc)
```

```
Widget   parent ;
String   name ;
Arg     *argv ;
Cardinal argc ;
{
                  return XtCreateWidget(name, fredWidgetClass,
parent, argv,
                      argc) ;
}
```

ウィジェットの作成に特別な処理が必要

他社のウィジェットの 1 つを作成する際に特別な処理が必要であるのに、UIL 作成関数が宣言されていない場合は、内容が事前構成されている標準形式の関数が必要です。

この処理には、次のリソースを使用します。

`visu*xw_<ウィジェットクラス>.uilProcedureTemplate: some_file_name`

Sun WorkShop Visual が UIL 用に生成する C コードの UIL 関数の角括弧 < > 内に、*some_file_name* がそのまま書き出されます。

たとえば、fred ウィジェットのサイズを作成前に設定しなければならない場合、以下のようにします。

1. `fred.uil_template` というファイルを作成します。
2. 作成したファイルを $USER_WIDGETS/code_templates/UIL に置きます。
3. 次のようにリソースを設定します。
   XDesigner*xw_fred.uilProcedureTemplate:

```
                    $USER_WIDGETS/code_templates/UIL/
fred.uil_template
```

注 - このリソースの値には、シェル変数を使用できます。シェル変数は、自動的に拡張されます。

次に、このテンプレートファイル `fred.uil_templates` を作成します。

```
/* UILテンプレートファイルの開始 */
Arg     *av ;
Cardinal ac ;
Widget   w ;
/* 渡された引数をコピーする */
                av = (Arg *) XtMalloc((unsigned) (argc + 2) *
sizeof(Arg)) ;
                for (ac = 0 ; ac < argc ; ac++) {
                    av[ac].name  = argv[ac].name ;
                    av[ac].value = argv[ac].value ;
                }
/* 必要な幅と高さを追加する */
                XtSetArg(av[ac], XmNwidth,  100) ; ac++ ;
                XtSetArg(av[ac], XmNheight, 100) ; ac++ ;
                w = XtCreateWidget(name, fredWidgetClass, parent,
av,
                                                    ac) ;
/* 整理する */
                XtFree((char *) av) ;
                return w ;
/* UIL テンプレートファイルの終了 */
```

この場合、X デザイナーは通常のアルゴリズムを使用して必要な UIL 関数の実名を決定し、UIL ファイルの C コードに含まれる UIL 関数の角括弧 < > 内に入力します。

第24章

コマンド行の操作

はじめに

本章では、Sun WorkShop Visual で使用可能なコマンド行について説明します。コマンド行オプションには、Sun WorkShop Visual を対話的に使用するためのものと、コマンド行からコード生成を行うためのものの 2 種類があります。本章では Sun WorkShop Visual 捕獲機能、Sun WorkShop Visual 再現機能、UILおよび GIL コードを Sun WorkShop Visual の保存ファイルに変換するためのコマンドについて説明します。

対話的使用のためのオプション

以下に示すコマンド行オプションを使用することができます。

表 24-1 対話的使用のための visu オプション

オプション	意味
`-windows`	Sun WorkShop Visual を Microsoft Windows モードで起動
`-f` *file*	入力ファイルを指定
`-L`	ユーザー独自のカラーマップを使用
`-x`	この説明を表示 (および終了)
`-V`	Sun WorkShop Visualのバージョン情報を表示 (プログラムは実行しない)

Sun WorkShop Visual を Microsoft Windows モードで起動する方法については、第 11 章「Microsoft Windows 用のデザイン」、415 ページの「Microsoft Windows モードでの起動」を参照してください。

ピックスマップ・エディタで多くの色を使用したい場合は、独自のカラーマップを使用すると便利です。ピックスマップ・エディタについては、170 ページの「ピックスマップの編集」を参照してください。ただし、この -L オプションを使用すると、他のウィンドウの色が指定した色とは異なる色で表示される場合があります。

コマンド行からのコード生成

コマンド行からの入力形式は次のとおりです。

```
visu [-csepAKCSEulbarmRMFWX [code_file]] [-windows] -f filename
```

表 24-2 visu コマンド行オプション

オプション	生成されるコードファイル
-c	C
-s	C スタブ
-e	C 外部宣言
-p	C/C++ ピックスマップ
-A	ANSI C を強制 (**-c**、**-s**、 **-e** を使用)
-K	K&R C を強制 (**-c**、**-s**、 **-e** を使用)
-C	C++
-S	C++ スタブ
-E	C++ 外部宣言
-u	UIL
-l	UIL のための C
-b	UIL のための C 外部宣言
-a	UIL ピックスマップ
-r	X リソースファイル
-m	メークファイル
-X	X リソースファイル (-r と同意)

表 24-2 visu コマンド行オプション (続き)

オプション	生成されるコードファイル
-M	Motif 様式の C++ を生成
-F	Motif XP 様式の C++ を生成
-J	code_file で指定したディレクトリに Java を生成
-W	Microsoft Windows MFC 様式の C++ を生成
-pixmaps	ピックスマップを、code_file で指定したディレクトリの .xpm ファイルに生成
-R	Microsoft Windows リソースファイル
-windows	Sun WorkShop Visual を Microsoft Windows モードで起動
-f *file*	入力ファイル (*file*) を指定

code_file は、生成されるファイルを表します。*code_file* を指定しない場合、Sun WorkShop Visual は指定された言語に対しソースファイル内で最後に指定された対象となるファイルにコードを生成します。

Java コード (-J) やピックスマップ (-pixmaps) については、複数のソースファイルが生成されます。コードファイルの名前はクラスの名前 (Javaの場合) またはデザインで使用したピックスマップの名前になるため、*code_file* にはこれらのソースファイルのターゲットディレクトリを指定します。

filename は、コード生成のためのソースとして使用するデザインファイル (.xd) を表わします。ファイル名はいつも指定しなければなりません。*code_file* を指定しない場合は、指定するファイル名が 1 つだけであることを示すために、区切り文字 -f を使用します。

-windows オプションは、Microsoft Windows モードを指定します。Sun WorkShop Visual を Microsoft Windows モードで起動する方法についての詳細は、第 11 章「Microsoft Windows 用のデザイン」の 415 ページの「Microsoft Windows モードでの起動」を参照してください。

-M、-F、-W、-R オプションは、必ず -windows オプションと組み合わせて使用します。

例

以下のコマンドは、foo.xd 内のデザインからファイル foo.c に C コードを生成します。

```
visu -c foo.c -f foo.xd
```

次のコマンドは、foo.xd から対象となるファイルに C コードを生成します。

```
visu  -c -f foo.xd
```

この対象となるファイルは、最後に「コード生成」ダイアログを使用して *foo.xd* から C コードが生成された際に指定されたものです。

以下に示す形式のどちらかを使用すると、単一のコマンドを使用して複数のファイルを生成することができます。

```
visu -c -e -s -f foo.xd
```

または

```
visu -c <c_ファイル> -e <外部ファイル> -s <スタブファイル> -f foo.xd
```

Sun WorkShop Visual は、成功するとステータス 0 で終了し、何らかの理由によりコード生成に失敗した場合には 0 以外のステータスで終了します 。

諸注意

コマンド行からコードを生成するためには、Sun WorkShop Visual が X サーバーと接続されている必要があります。通常、コマンド行からのコード生成では目に見えるウィンドウの作成は行われませんが、Windows コードの生成時には、スクロールリストやスクロールテキストなどの特定の種類のウィジェットを含むデザインに対して、サーバー画面上にウィンドウが一時的に表示されます。

code_file を指定しない場合、Sun WorkShop Visual はデザインファイルに保存されているファイル名を使用します。「コード生成」ダイアログで指定してそのファイルを保存しない限り、ファイルの名前は保存されません。したがって、「コード生成」ダイアログを使ってデザインファイルからこのようなコードを生成したことがない場合は、Sun WorkShop Visual はエラーメッセージを生成します。

すべての場合において、最後にデザインファイル内で保存されたとおりに生成トグルが設定されます。デザインファイルからこの種類のコードを生成したことがない場合には、デフォルトのトグル設定が使用されます。

Sun WorkShop Visual 再現機能

ユーザーのアクションを記録するために使用する Sun WorkShop Visual 再現機能は、visu_record という独立したアプリケーションとして提供されています。

以下に visu_record の使用方法を示します。

`visu_record -f` *MyRecordScript AnApplication*

MyRecordScript は、一連のアクションを記録したスクリプトが保存されるファイルの名前です。*AnApplication* は、記録する対象であるアプリケーションの名前です。

以下に `visu_replay` の使用方法を示します。

Sun WorkShop Visual 再現機能と XD/ 記録機能のファイル名引数は省略できます。この場合、XD/ 記録機能は標準出力から出力し、Sun WorkShop Visual 再現機能は標準入力から読み取ります。

次の表に、visu_record と visu_replay の両方で使用できるすべてのコマンド行オプションを示します。

表 24-3 visu_record と visu_replay のコマンド行オプション

オプション	意味
`-x`	Sun WorkShop Visual 再現機能の詳細情報を表示。
`-f` *file*	記録の場合はファイルへ保存 (指定のない場合は標準出力)。再生の場合はファイルから読み出し (指定のない場合は標準入力)。
`-use` *n*	プログラムのメインシェルの前に表示されるシェルを n 個無視
`-lang` *locale*	GUI (グラフィカルユーザーインタフェース) やエラーメッセージを含む visu_record または visu_replay を、指定したロケールで起動。LANG 設定は無視されます。
`-p`	スクリプトを再現する前に、C プリプロセッサで事前処理。
`-v`	詳細出力 (verbose)。
`-V`	visu_record または visu_replay のバージョン情報を表示。
`-w`	ディスプレイ、サーバー、ウィンドウマネージャの概要情報を表示。
`-O`	上書き (非 Motif アプリケーションに対してはプログラム終了)。

表 24-3 visu_record と visu_replay のコマンド行オプション (続き)

オプション	意味
`-i`	対話的 (interactive)。`-f` を無視し、捕獲/再現ダイアログを使用。
`-I`	強制的にダイアログを表示。
`-exit-on-error`	再現不可能なコマンドがあった場合に、visu_replay スクリプトとアプリケーションを終了。
`-user-on-error`	再現不可能なコマンドがあった場合に、visu_replay スクリプトを終了。アプリケーションは終了しません。
`-skip-on-error`	再現不可能なコマンドがあった場合に、visu_replay スクリプト内の次のシーケンスへジャンプ。

Sun WorkShop Visual 捕獲機能

Sun WorkShop Visual 捕獲機能は、 visu_capture という独立したアプリケーションとして提供されています。

以下に visu_capture の使用方法を示します。

`visu_capture -f` *AnApplication*

「捕獲」ダイアログが表示され、アプリケーション *AnApplication* が実行されます。

次に visu_capture で使用できるすべてのコマンド行オプションを示します。

表 24-4 `visu_capture` コマンド行オプション

オプション	意味
`-x`	visu_capture の詳細情報を表示。
`-f` *file*	ファイルへ保存。このオプションで「捕獲」ダイアログは表示されません。
`-lang` *locale*	GUI (グラフィカルユーザーインタフェース) やエラーメッセージを含む visu_capture を、指定したロケールで起動。LANG 設定は無視されます。
`-use` *n*	プログラムのメインシェルの前に表示されるシェルを *n* 個無視。
`-v`	詳細出力 (verbose)。
`-V`	visu_capture のバージョン情報を表示。

表 24-4 `visu_capture` コマンド行オプション (続き)

オプション	意味
`-w`	ディスプレイ、サーバー、ウィンドウマネージャの概要情報を表示。
`-O`	上書き (非 Motif アプリケーションに対してはプログラム終了)。
`-i`	対話的 (interactive)。`-f` を無視し、獲得/再現ダイアログを使用。
`-I`	強制的にダイアログを表示。
`-j`	Java で使用可能な捕獲機能。
`-static-design`	アプリケーションシェルの初期状態のみを捕獲して終了。

UILソースの Sun WorkShop Visual 保存ファイルへの変換

uil2xd フィルタは UILソースコードを Sun WorkShop Visual 保存ファイルに変換します。このフィルタは、UILソースを標準入力から読み出して、Sun WorkShop Visual 保存ファイルに標準出力で書き込みます。

デフォルトにより、*uil2xd* は最新版 Sun WorkShop Visual 用の保存ファイルを生成します。以下のように入力してください。

```
uil2xd [-tlxywhpsaX] [-I include_dir]
```

次の表に、コマンド行オプションを示します。

表 24-5 uil2xd コマンド行オプション

オプション	意味
-t	テキストを含んでいるスクロールウィンドウをスクロールテキストに変換しない。 デフォルトでは、**uil2xd** は「テキスト」ウィジェットを含んでいる「スクロールウィンドウ」ウィジェットを「スクロールテキスト」ウィジェットに変換します。構造を保存しておくには、-t オプションを使用します。
-l	リストを含んでいるスクロールウィンドウをスクロールリストに変換しない。 デフォルトでは、**uil2xd** は「リスト」ウィジェットを含んでいる「スクロールウィンドウ」ウィジェットを「スクロールリスト」ウィジェットに変換します。構造を保存しておくには、-l オプションを使用します。
-x	XmNx リソースを変更せずに渡す。 デフォルトでは、**uil2xd** は保存ファイル内の絶対位置の出力は行いません。 XmNx リソースを出力ファイルに渡すには、-x オプションを使用します。
-y	XmNy リソースを変更せずに渡す。 デフォルトでは、**uil2xd** は保存ファイル内の絶対位置の出力は行いません。 XmNy リソースを出力ファイルに渡すには、-y オプションを使用します。
-w	XmNwidth リソースを変更せずに渡す。 デフォルトでは、**uil2xd** は保存ファイル内の絶対サイズの出力は行いません。 XmNwidth リソースを出力ファイルに渡すためには、-w オプションを使用します。
-h	XmNheight リソースを変更せずに渡す。 デフォルトでは、**uil2xd** は保存ファイル内の絶対サイズの出力は行いません。 XmNheight リソースを出力ファイルに渡すためには、-h オプションを使用します。
-p	出力ファイルに位置リソースを保存する。-x、-y と同じです。
-s	出力ファイルにサイズリソースを保存する。-w、-h と同じです。

表 24-5 uil2xd コマンド行オプション (続き)

オプション	意味
`-a`	出力ファイルに位置およびサイズリソースを保存する。`-p`、`-s` と同じです。
`-e`	構文エラーの解決方法を説明する。
`-A`	アタッチメントの設定されていないフォームの子に、仮のアタッチメントを生成する。
`-I` *include_dir*	*include_dir* を、インクルードファイルの検索を行うディレクトリの一覧に追加する。
-X	オプションの一覧を印刷する。

uil2xd は、以下の構造体の処理を行いません。

- コンパウンド文字列を含んでいる文字列表
- color_table
- アイコン
- 引数定義内の ASCII 表
- 引数定義内の整数表
- インポートされたキーワード (これは致命的エラーとなります)
- エクスポートされたキーワード
- プライベート・キーワード
- 作成関数
- デフォルト文字セット句
- 識別子セクション

インポートされたキーワードを除いて、*uil2xd* はこれらの構造体を無視します。

GIL ソースの Sun WorkShop Visual 保存ファイルへの変換

gil2xd フィルタは、サンの DevGuide ファイルを Sun WorkShop Visual 保存ファイルに変換します。コンバータによって OPEN LOOK オブジェクトが Motif オブジェクトに変換されます。このフィルタは GIL ソースを標準入力から読み出して、Sun WorkShop Visual 保存ファイルに標準出力で書き込みます。

このフィルタは最新バージョンの Sun WorkShop Visual 用の保存ファイルを生成します。以下のように入力してください。

```
gil2xd  [-xywhpsaX]
```

コマンド行オプションを次の表に示します。

表 24-6 gil2xd コマンド行オプション

オプション	意味
`-x`	XmNx リソースを変更せずに渡す。 デフォルトでは、**gil2xd** は保存ファイル内の絶対位置の出力は行いません。XmNx リソースを出力ファイルに渡すには、`-x` オプションを使用します。
`-y`	XmNy リソースを変更せずに渡す。 デフォルトでは、**gil2xd** は保存ファイル内の絶対位置の出力は行いません。XmNy リソースを出力ファイルに渡すには、`-y` フラグを使用します。
`-w`	XmNwidth リソースを変更せずに渡す。 デフォルトでは、**gil2xd** は保存ファイルにおける絶対サイズの出力は行いません。XmNwidth リソースを出力ファイルに渡すには、`-w` オプションを使用します。
`-h`	XmNheight リソースを変更せずに渡す。 デフォルトでは、**gil2xd** は保存ファイルにおける絶対サイズの出力は行いません。XmNheight リソースを出力ファイルに渡すには、`-h` オプションを使用します。
`-p`	出力ファイルに位置リソースを保存する。`-x`、`-y` と同じです。

表 24-6 gil2xd コマンド行オプション (続き)

オプション	意味
-s	出力ファイルに位置およびサイズリソースを保存する。-w、-h と同じです。
-a	出力ファイルに位置およびサイズリソースを保存する。-p、-s と同じです。
-X	オプションの一覧を印刷する。

gil2xd は、関数呼び出し以外の接続の処理は行わず、リンクに割り当てられたボタンに対するアクションを通知します。その他の接続は警告として報告されます。*gil2xd* は、これらの構造体を無視します。

変換

OPEN LOOK オブジェクトの Motif ウィジェットへの変換は、個々の内容に依存するため、単純には行われません。基本的な変換方法は以下のとおりです。

基本ウィンドウ

子にメインウィンドウおよびフォーム作業領域を持つダイアログシェルに変換されます。

ポップアップウィンドウ

子にフォームを持つダイアログシェルに変換されます。

キャンバス区画

水平スクロールバーまたは垂直スクロールバーが true である場合は、スクロールウィンドウの子になる描画領域に変換されます。描画領域の子として、関連するポップアップメニューが作成されます。

制御領域

フォームに変換されます。

メニュー

メニューウィジェットに変換されます。メニューにメニュータイトルの属性がある場合は、タイトルを表示するラベルウィジェットが最初の子になり、セパレータウィジェットが 2 番目の子になります。メニュー内の項目は、メニューウィジェットの子に割り当てられます。メニューの種類属性がコマンドである場合は、ウィジェットはトグルボタンになります。また、メニュー項目が関連付けられている場合は、カスケードボタンになります。その他の場合は、プッシュボタンになります。Sun WorkShop Visual には、メニューを共有する概念がないため、複数の場所から参照されるメニューは、メニューウィジェットの複製に割り当てられます。

メッセージ

ラベルに変換されます。

ボタン

メニューを持たない場合には、プッシュボタンに変換されます。その他の場合は、カスケードボタンに割り当てられます。このカスケードボタンは、メニューバー内に作成されます。同じ y 座標を持つカスケードボタンは、同一のメニューバー内に作成されます。可能であれば、メニューバーはメニューバーウィジェットを格納するメインウィンドウ内に作成されますが、その他の場合はケースバイケースで適切な位置に作成されます。

スライダとゲージ

両方ともスケールに変換されます。目盛の子として、セパレータが追加され、最小値文字列および最大値文字列を表示するラベルが追加される場合もあります。最小値および最大値はそれぞれ、スケールの最小フィールドおよび最大フィールドに変換されます。

設定

設定型がスタックである場合はオプションメニューに変換され、その他の場合はローカラムに変換されます。選択肢は、オプションメニューの場合にはプッシュボタンに、ローカラムの場合にはトグルボタンに変換されます。排他的または非排他的設定の場合には、トグルボタンはインジケータを使用しないように調整されます (シャドウの厚さ = 2、マージン左 = 0 、インジケータオン = false)。

テキストフィールド

ラベルを持つローカラムおよびテキストウィジェットに変換されます。テキストが複数行に設定されている場合は、スクロールテキストになります。

リスト

スクロールリストに変換されます。リストにラベルの属性が設定されている場合は、スクロールリストは、ラベルを表示するラベルウィジェットを子として持つローカラムの子として作成されます。リストにタイトルの属性がある場合は、スクロールリストは、そのタイトルを表示するラベルを持つフレームの子として作成されます。

ドロップターゲット

ラベルに割り当てられます。

スタック

子としてスタックメンバーウィジェットを持つローカラムに変換されます。

グループ

子として各メンバーウィジェットを持つローカラムに変換されます。

端末区画およびテキスト区画

いずれもスクロールテキストに変換されます。

属性

gil オブジェクトがウィジェットに割り当てられた場合は、属性は適切なウィジェットリソースに割り当てられなければなりません。以下のリソースが常に割り当てられます。

表 24-7 常に変換されるリソース

gil	xd	注釈
x	XmNx	
y	XmNy	
幅	XmNwidth	
高さ	XmNheight	
前景色	XmNforeground	
背景色	XmNbackground	
初期状態	mNsensitive	非活性、応答可能 = false 非表示、マネージ = false

幅および高さリソースは、`gil2xd` の実行時に `-w` または `-h` オプションが設定されている場合にのみ使用されます。x および y リソースは、`-x` または `-y` オプションが設定されている場合に出力されます。ただし、フォームの子であるウィジェットに対しては、x および y 座標を使用してデフォルトのフォームアタッチメントを算出し、適切な配置が保たれるように処理します。

Motif マネージャウィジェットの多くは、明示的な x、y、幅および高さのリソースを無視します。実行時オプションとは関係なく `gil2xd` フィルタを使用して、Sun WorkShop Visual で簡単に修正できる、適切な配置を作成することができます。

その他のリソースは、最も近いと思われるリソースに変換されます。

表 24-8 最も近いリソースに変換されるリソース

gil	xd	注釈
カラム	XmNcolumns	
固定幅	XmNrecomputeSize	
グループ型	XmNorientation	類似したグループ配置を再現するには、XmNnumcolumns, XmNorientation および XmNpackingを設定します。
アイコンファイル	XmNiconPixmap	
アイコンラベル	XmNiconName	
アイコンマスク	XmNiconMask	
初期状態	XmNinitialState	ダイアログシェルのみ
初期値	XmNvalue	
ラベル	XmNlabelString	該当するラベルが glyph 型の場合には、ラベルは labelPixmap に割り当てられます。
ラベル	XmNtitle	シェルの場合
ラベル	XmNtitleString	ゲージの場合
ラベル型	XmNlabelType	
配置型	XmNorientation	
最大値	XmNmaximum	
メニュー型	XmNradioBehavior	排他的の場合には、XmNradioBehavior = true
最小値	XmNminimum	
複数選択	XmNselectionPolicy	設定されている場合は、XmNselectionPolicy = MULTIPLE_SELECT
配置方向	XmNorientation	
ピン可能	XmNtearOffModel	ピン可能である場合は、XmNtearOffModel = TEAR_OFF_ENABLED

表 24-8 最も近いリソースに変換されるリソース (続き)

gil	xd	注釈
読み出し専用	XmNeditable	
サイズ変更可能	XmNallowResize	
行	XmNnumColumns	類似したグループ配置を再現するには、XmNnumColumns、XmNorientation、XmNpacking を設定します。
行	XmNrows	テキストウィジェットの場合
行	XmNvisibleItemCount	リストウィジェットの場合
要選択	XmNradioAlwaysOne	
ボーダー表示	XmNshadowThickness	設定されている場合は、シェルの子ではないフォームに対して、XmNshadowThickness を 1 に設定します。
値表示	XmNshowValue	
スライダ幅	XmNscaleWidth	配置方向によって、XmNscaleWidth または XmNscaleHeight を設定します。
保存長	XmNmaxLength	
テキスト初期値	XmNvalue	
テキスト型	XmNeditMode	複数行の場合は、XmNscrollVertical = false、行は XmNrows に変換されます。
タイトル	XmNlabelString	
値の長さ	XmNcolumns	

関数型 CallFunction を持つアクションは、適切なコールバックに変換されます。

表 24-9 コールバックに変換されるアクション

アクション	コールバック	ウィジェット
Create	XmNcreateCallback	任意
Destroy	XmNdestroyCallback	任意
Notify	XmNactivateCallback	プッシュボタン
select	XmNinputCallback	描画領域
adjust	XmNinputCallback	描画領域
DoubleClick	XmNinputCallback	描画領域
Repaint	XmNexposeCallback	描画領域
Resize	XmNresizeCallback	描画領域
Select	XmNvalueChangedCallback	ゲージ
Adjust	XmNdragCallback	ゲージ
Notify	XmNvalueChangedCallback	ゲージ
Popup	XmNmapCallback	メニュー
Popdown	XmNunmapCallback	メニュー
Notify	XmNentryCallback	メニュー
Notify	XmNvalueChangedCallback	トグルボタン
Unselect	XmNvalueChangedCallback	トグルボタン
Popup	XmNpopupCallback	シェル
Popdown	XmNpopdownCallback	シェル
Notify	XmNactivateCallback	テキスト
KeyPress	XmNvalueChangedCallback	テキスト
Notify	XmNentryCallback	ローカラム
Done	XmNunmapCallback	フォーム
Notify	XmNbrowseSelectionCallback	リスト

表 24-9 に記載されていない GIL アクションも多数存在します。ただし、適切な Motif コールバックが存在しないため、これらはフィルタでサポートされていません。

表示、非表示、有効化、無効化の接続を持つプッシュボタンに対する notify アクションは、適切な Sun WorkShop Visual リンクに変換されます。

第25章

構成

はじめに

リソースファイルあるいは visu_config を使用して Sun WorkShop Visual インタフェースをカスタマイズするには、いくつかの方法があります。本章では、以下に示すリソースファイルを使用してカスタマイズできる主要機能について説明します。

- コールバックとプレリュードの編集
- パレットアイコン
- パレット内容および配置
- ツールバー
- メークファイルテンプレート

Sun WorkShop Visual リソースの詳細は、993 ページの付録 D「アプリケーションのデフォルト」を参照してください。また、visu_config の使用については、719 ページの第 23 章「ユーザー定義ウィジェット」を参照してください。

リソースを使用して構成できる領域には、ダイナミックディスプレイウィンドウがあります。このウィンドウのアプリケーションクラス名は独自の名前で、リソースファイルも独自のリソースファイルになります。詳細は、818 ページの「ダイナミックディスプレイウィンドウ」を参照してください。

コールバックおよびプレリュード編集の設定

コールバックとプレリュードの編集には 2 通りの方法があります。1 つ目の方法では Sun Edit Services を使用し、2 つ目の方法では「xterm」ウィンドウで選択したエディタを呼び出すだけです。本節では、編集機能の構成に必要なアプリケーションリソース、アプリケーション、環境変数について説明します。Sun WorkShop Visual のアプリケーションリソースの使用については付録 D「アプリケーションのデフォルト」を参照してください。最初のリソースは、コールバック編集機能を有効にするかどうかを制御します。

callbackEditing

このリソースは、コールバック編集機能を有効にするかどうかを制御します。false に設定すると、この機能に関連するボタンは表示されません。

Sun WorkShop 編集サーバーを使用しないでコールバックとプレリュード編集機能を使用する場合は、使用するエディタを指定してください。

editor

このリソースは、エディタを起動するシェルラッパースクリプトの位置を指定します。デフォルトでは、次の場所に設定されています。

```
$VISUROOT/lib/scripts/xd_edit
```

`xd_edit` はテキストファイルで、さまざまなエディタに適用する起動コマンドを提供するシェルスクリプトを含んでいます。使用したいエディタがこのファイルにリストされていない場合は、デフォルトでエディタ「vi」に指定されている機能を使用します。エディタを呼び出すと、「xterm」が起動され、Sun WorkShop Visual はスタブファイル (コールバックを編集している場合) または基本ソースファイル (プレリュードを編集している場合) の適切な行に移動しようとします。これはすべてのエディタに適用されるわけではありません。たとえば、Motif ベースのエディタの場合は「xterm」は必要がなく、また一部のエディタは起動時に指定された行に移動することができません。

ファイル xd_edit には、各種エディタの例が説明のコメントとともに含まれています。既存の行をコピーして、選択したエディタに対する特別な情報を追加することができます。シェルスクリプト言語の詳細については、UNIX シェルのマニュアルを参照してください。

xd_edit シェルスクリプトは、EDITOR 環境変数で使用したいエディタの名前を調べます。この変数が設定されていないと、スクリプトは VISUAL 環境変数を調べます。この変数も設定されていない場合のデフォルトは「vi」です。これらの変数のいずれかを使用したい場合、そのエディタが PATH 環境変数にリストされているディレクトリにないときは絶対ディレクトリ名でエディタ名を指定します。エディタが決まると、スクリプトはそのエディタを環境変数 XD_TERM で定義される端末プログラムで実行します。この変数が設定されていない場合は、デフォルトで xterm プログラムを使用するよう設定されています。

環境変数の使用については、UNIX のマニュアルを参照してください。

編集機能の使用

編集機能を使用したコールバックの編集については 264 ページの「コールバック関数の追加」を、またプレリュード編集については 282 ページの「コードプレリュード」を参照してください。

パレットアイコン

Sun WorkShop Visual は、各ウィジェットクラスに対してアイコンを持っています。アイコンは、パレットボタンに描かれ、ツリー階層内に表示されます。アイコンは、フルカラー XPM フォーマットのピックスマップ、あるいはモノクロームビットマップにすることができます。起動時に、Sun WorkShop Visual はアプリケーションリソースを検索して各アイコンに対してのピックスマップまたはビットマップファイルを探し出します。ファイルが見つからない場合は、組み込まれているビットマップが使用されます。

アイコンファイルの指定

各 Motif ウィジェットには、アイコンファイルを指定するアプリケーションリソースがあります。アプリケーションリソースの例を次に示します。

```
$VISUROOT/lib/locale/<ロケール>/app-defaults/visu
```

$VISUROOT は Sun WorkShop Visual のインストールディレクトリです。<ロケール> には、使用する言語のロケールを指定します。デフォルトのロケールは C です。LANG 環境変数でロケール名を確認することができます。

アプリケーションリソースファイルには、Motif ウィジェットアイコンのリソース名をすべて示した完全なリストが含まれます。たとえば、矢印ボタンのリソースは次のようになります。

```
visu.arrowButtonPixmap: arrow.xpm
```

Sun WorkShop Visual は、*XmGetPixmap()* と同じ検索方法で、ビットマップファイルを探します。この検索パスは複雑であるため、詳細は Motif の資料を参照してください。実際問題として、Sun WorkShop Visual はデフォルトピックスマップファイルを $VISUROOT/bitmaps に配置し、XBMLANGPATH 環境変数に $VISUROOT/bitmaps/%B を追加します。ユーザーは、arrow.xpm 等の正しい名前のファイルを任意のディレクトリに作成し、以下のようにディレクトリとファイル一致文字列「/%B」を環境変数 SW_VISU_XBMLANGPATH に追加することにより、ユーザー独自のピックスマップを使用することができます。

```
setenv SW_VISU_XBMLANGPATH /home/me/pix/%B
```

デフォルトのピックスマップ

Sun WorkShop Visual には、2 組のアイコンピックスマップが提供されています。デフォルトのピックスマップは、$VISUROOT/bitmaps にあります。デフォルトアイコンは、最小限の色を使用して描画され、カラーまたはモノクロ画面のどちらでも動作します。

カラーのピックスマップは、$VISUROOT/color_icons にあります。デフォルトをカラーピックスマップに変更するには、環境変数 SW_VISU_ICONS を color_icons に設定するか、$VISUROOT/color_icons/%B を環境変数 SW_VISU_XBMLANGPATH に追加します。デフォルトのアイコンに戻す場合は、SW_VISU_ICONS の設定を解除、あるいは bitmap に設定します。

また、以下のようにカスタマイズされた Sun WorkShop Visual リソースファイルのリソースを設定して、別のファイル名を指定することができます。

```
visu.arrowButtonPixmap: my_arrow.xpm
```

または

```
visu.arrowButtonPixmap: /home/me/visu_bitmaps/my_arrow.xpm
```

ピックスマップの必要条件

アイコンに対する独自のカラーピックスマップは、XPM3 フォーマットを使用して作成することができます。このフォーマットは、Sun WorkShop Visual ピックスマップエディタを使用して作成することができます (詳細は 170 ページの「ピックスマップの編集」を参照してください)。アイコンピックスマップは任意のサイズにすることができます。パレットおよびツリーは、最も高さのあるアイコンを収めるため、垂直方向に間隔をとって表示されます。すべてのアイコンピックスマップを同じサイズにすると、Sun WorkShop Visual の表示が整って見えます。デフォルトのサイズは、大画面アイコンピックスマップの場合は 32 × 32、小画面バージョンの場合は 20 × 20 です。

Sun WorkShop Visual から印刷を行う場合、アイコンはグレースケール状態になります。

透明領域

アイコンには、透明の領域を持たせる必要があります。Sun WorkShop Visual はこの領域を使用して、強調の表示、ツリー内の色やパレットボタンの背景色を構成します。XPM は、カラー名 none (カラーオブジェクト none ではないことに注意) を使用して、透明色をサポートしています。透明色の設定方法については、183 ページの「透明色」を参照してください。

ユーザー定義ウィジェットのアイコン

visu_config のウィジェット編集ダイアログを使用すると、ユーザー定義ウィジェットのアイコンを指定することができます。詳細は、733 ページの「ウィジェットの属性」を参照してください。

パレット定義に対してのアイコン

「定義を編集」ダイアログを使用すると、各パレット定義のアイコンと、リソースが設定されていない場合に使用されるファイル名を指定することができます。デフォルトピックスマップと同じ方法で、ピックスマップファイルも検索されます。

Sun WorkShop Visual がピックスマップを探し出すことができない場合は、定義のルートにあるウィジェットのデフォルトピックスマップを使用します。詳細は第 23 章、312 ページの「定義ファイルの編集」のを参照してください。

パレットの内容

stopList リソースを使用して、特定の Motif ウィジェットクラスをウィジェットパレットに表示しないようにすることができます。また、visu_config を使用してユーザー定義ウィジェットパレットに表示しないようにすることもできます。停止されたウィジェットは、対話的に作成することはできません。これらのウィジェットは Sun WorkShop Visual が生成した保存ファイルから読み出すことができますが、選択することはできません。

ウィジェットクラスを停止するためには、*stopList* リソースでクラス名を指定します。たとえば、ウィジェットパレットから Motif 区画ウィンドウと矢印ボタンを削除するためには、以下のようにリソースを設定します。

```
visu.stopList: XmPanedWindow,XmArrowButton
```

ユーザー定義ウィジェットを停止するには、クラス名を指定します。

```
visu.stopList: boxWidgetClass,formWidgetClass
```

visu_config には、各 Motif ウィジェットに対するトグルのついた「停止された Motif ウィジェット」ダイアログがあります。詳細は、719 ページの第 23 章「ユーザー定義ウィジェット」を参照してください。visu_config でパレットから取り除かれたウィジェットは、Sun WorkShop Visual リソースを使用して元に戻すことはできません。

パレットの表示方法

デフォルト設定では、ウィジェットアイコンがメインウィンドウ上の垂直パレットに 3 列に表示されています。デフォルトの配置は、リソースファイルを使用して変更することができます。また、パレットメニューを使用して実行時にパレットの配置を変更することもできます。

別々のパレット

パレットは、別のウィンドウに表示することができます。実行時に別々のパレットを利用するためには、パレットメニューの「別々のパレット」オプションを使用します。デフォルトで独立したパレットを使用するためには、以下のリソースを設定します。

```
visu*pm_separate.set:true
```

リソースファイルで別々のパレットの設定をする場合は、Sun WorkShop Visual メインウィンドウのデフォルトの高さを明示的に設定する必要があります。

```
visu.main_window.height: 600
```

リソースファイルには、以下のような配置変更の例がいくつか含まれています。

```
! ユーザー定義ウィジェットを持っていない場合には、2 列がよいでしょう。
visu*widgetPalette.composite.buttonBox.numColumns: 2
visu*widgetPalette.basic.buttonBox.numColumns: 2
! ラベルとアイコンの両方をオンに設定します。
Visu*pm_both.set: true
! ツールの幅を少し広くする場合
Visu.main_window.width: 750
! 4 列で、ラベルを下に表示します。
XDesigner*widgetPalette.composite.buttonBox.XmRowColumn.\
orientation: VERTICAL
XDesigner*widgetPalette.composite.buttonBox.numColumns: 4
XDesigner*widgetPalette.basic.buttonBox.XmRowColumn.\
orientation: VERTICAL
XDesigner*widgetPalette.basic.buttonBox.numColumns: 4
XDesigner*widgetPalette*xwidget_icons.*orientation: VERTICAL
XDesigner*widgetPalette*xwidget_icons.numColumns: 4
XDesigner*widgetPalette*_defn_icons.*orientation: VERTICAL
XDesigner*widgetPalette*_defn_icons.numColumns: 4
```

ツールバー

Sun WorkShop Visual インタフェースには、ツールバーがあります。ツールバーのボタンは、メニューに対応しています。一般に、ツールバーボタンを選択すると対応するメニューボタンとまったく同じ動作が実行されます。

ツールバーの構成

ツールバーのボタンを構成するためには、*toolbarDescription* リソースを使用します。このリソースには、ボタンのウィジェット名をコンマで区切って指定します。また、項目の間隔を広くするための単語 *separator*、および Microsoft Windows 様式オプションメニューを挿入するための単語 flavor を指定することもできます。

ボタンの名前を確認するには、メニューバーのボタンの labelString リソースを設定するエントリを、 Sun WorkShop Visual のアプリケーションリソース内で検索します。アプリケーションリソースファイルの場所については。付録 D「アプリケーションのデフォルト」を参照してください。

たとえば、Sun WorkShop Visual のアプリケーションリソースには以下の行が含まれてます。

```
visu*em_cut.labelString: カット
```

em_cut は、編集メニュー内の「カット」ボタンのウィジェット名です。

以下の行は、「カット」、「コピー」、「ペースト」、「コアリソース」、「配置」(配置エディタ) および「C」(コード生成) ボタンを持つツールバーを作成します。

```
visu.toolbarDescription:separator,em_cut,\
em_copy,em_paste,separator,wm_prim,\
wm_layout,separator,gm_c
```

ツールバーボタンのラベル

ツールバーボタンは、対応するメニューボタンと同じウィジェット名を持っているため、デフォルト (ピックスマップが構成されていないことを前提とする) では、これらのボタンは同じ言葉で表示されます。たとえば、リソースファイルは以下の行を持ちます。

```
visu*gm_c.labelString: C...
```

デフォルトでは、これはメニューとツールバーの両方に適用されます。ツールバーのコード生成ボタンはダイアログを表示しないため、省略符号 (...) を取り除いてください。省略符号を取り除くには、以下の行を追加します。

```
visu*toolbar.gm_c.labelString: C
```

ツールバーボタンに対してのピックスマップ

ツールバーボタンは、パレットボタンとまったく同じ方法で、関連する XPM ピックスマップまたは X11 ビットマップを指定するための文字列リソースも持ちます。

```
visu*toolbar.em_copy_file.toolbarPixmap: em_copy_file
```

これらのピックスマップは、リソースの変更、あるいは検索パスであらかじめファイルを指定することによって書き換えることができます。

メークファイル機能

本節では、メークファイル生成を制御するリソースを説明します。この機能に対する説明は、639 ページの第 19 章「メークファイル生成」を参照してください。

ファイル接尾辞

オブジェクトおよび実行可能なファイル名は、接尾辞置換によってソースファイル名から取り出されます。接尾辞は、以下のアプリケーションリソースによって指定されます。

```
visu.objectFileSuffix: .o
visu.executableFileSuffix:
visu.uidFileSuffix: .uid
```

メークファイルテンプレート

新しいメークファイルの生成に使用するテンプレートは、ファイル名で、あるいは直接リソースファイルで定義されます。後者の方法は、ファイルが見つからない場合に、フォールバックとして使用されます。

テンプレートファイルは、以下のいずれかのリソースで指定されます。

```
visu.motifMakeTemplateFile:$VISUROOT/make_templates/motif[1]

visu.mmfcMakeTemplateFile:$VISUROOT/make_templates/mfc
```

2 個のリソースが存在するため、異なるテンプレートをカスタマイズして、適切なクラスライブラリを選択することができます。リソースの値には、/bin/sh によって拡張される環境変数を持たせることができます。

フォールバックテンプレートは、*makefileTemplate* リソースによって指定されます。

```
visu.makefileTemplate:\

# System configuration\n\

# ---------------------\n\

\n\

# everything is in /usr/include or /usr/lib\n\

XINCLUDES=\n\

XLIBS=\n\

LDLIBS=\n\

.

.
```

テンプレートの先頭部分でメークファイル変数を編集し、システム構成を反映させることができます。たとえば、変数 *XINCLUDES* を X インクルードファイルのパスに設定することができます。

テンプレートプロトコル

ここでは、 メークファイルテンプレートで使用される記号を説明します。一般に、先頭の変数を除いて、デフォルトテンプレートを編集することはお勧めできません。実際のテンプレート行を編集する場合は、まず 639 ページの第 19 章「メークファイル生成」を参照し、変数を設定して希望する結果を導き出すようにしてください。

#Sun WorkShop Visual: で始まるメークファイルテンプレート内の行を、テンプレート行と呼びます。Sun WorkShop Visual がメークファイルを生成または更新する場合、生成したファイルに基づいて各デザインファイルに適切なテンプレート行のインスタンスを作成し、 Sun WorkShop Visual が生成した接頭辞 *XDG_* で始まる特殊記号をファイル名に変換します。*XDG_* 記号は、記号名が接尾辞 *_LIST* で終わる場合にはファイルのリストに、その他の場合は単一のファイルに変換されます。

1. $VISUROOT は、Sun WorkShop Visual のインストールディレクトリです。

リスト記号は、以下に示すように、行において単独で使用され、通常の記号と組み合わされることはありません。

```
#Sun WorkShop Visual:XD_C_PROGRAMS=XDG_C_PROGRAM_LIST
```

XDG_C_PROGRAM_LIST 記号は、メークファイルが構築することのできるすべての実行可能ファイルのリストに変換されます。以下はこのテンプレート行の代表的なインスタンスの例です。

```
#DO NOT EDIT >>>

XD_C_PROGRAMS=\
myapp1\
myapp2\

#<<< DO NOT EDIT
```

接尾辞 *_LIST* を持たない通常のテンプレート記号は、単一のファイルを表わします。通常のテンプレート記号は、以下に示すように行中で複数の記号と組み合せることができます。

```
#Sun WorkShop Visual:XDG_C_PROGRAM: XDG_C_PROGRAM_OBJECT
$(XD_C_OBJECTS) $(XD_C_STUB_OBJECTS)

#Sun WorkShop Visual: $(CC) XDG_C_DEBUG_FLAGS $(CFLAGS) $(CPPFLAGS)
$(LDFLAGS) -o XDG_C_PROGRAM XDG_C_PROGRAM_OBJECT $(XD_C_OBJECTS)
$(XD_C_STUB_OBJECTS) $(MOTIFLIBS) $(LDLIBS)
```

Sun WorkShop Visual がメークファイルを生成する場合、「メインプログラム」トグルを設定してコードを生成したデザインファイルに対して、これらの行の別のインスタンスを追加します。アプリケーション内のその他のファイルは、*XD_C_OBJECTS* および *XD_C_STUB_OBJECTS* としてリンクされます。

```
#DO NOT EDIT >>>

myapp1: myapp1.o $(XD_C_OBJECTS) $(XD_C_STUB_OBJECTS) $(CC)
$(CFLAGS) $(XDG_C_DEBUG_FLAGS) $(CPPFLAGS) $(LDFLAGS) -o\myapp1
myapp1.o

$(XD_C_OBJECTS) $(XD_C_STUB_OBJECTS) $(MOTIFLIBS) $(LDLIBS)

#<<< DO NOT EDIT

#DO NOT EDIT >>>

myapp2: myapp2.o $(XD_C_OBJECTS) $(XD_C_STUB_OBJECTS) $(CC) $(CFLAGS)
$(XDG_C_DEBUG_FLAGS) $(CPPFLAGS) $(LDFLAGS) -o\myapp2 myapp2.o

$(XD_C_OBJECTS) $(XD_C_STUB_OBJECTS) $(MOTIFLIBS) $(LDLIBS)
```

```
#<<< DO NOT EDIT
```

以下の表に、一般的なテンプレート記号を示します。記号に接尾辞 *_LIST* を追加して、対応するリスト記号を生成することができます。*_LIST* は「FLAGS」記号には追加できません。( ) 内は、ファイル名の形式を示します。

表 25-1 メークファイルテンプレート記号

名前	用途
XDG_C_PROGRAM_SOURCE	メインプログラムを持つ C ソース (*main.c*)
XDG_C_PROGRAM_OBJECT	対応するオブジェクト (*main.o*)
XDG_C_PROGRAM	対応する実行可能ファイル (*main*)
XDG_C_SOURCE	C ソース (*foo.c*)
XDG_C_OBJECT	対応するオブジェクト (*foo.o*)
XDG_C_STUB_SOURCE	C スタブ (*stubs.c*)
XDG_C_STUB_OBJECT	対応するオブジェクト (*stubs.o*)
XDG_C_EXTERN	C ヘッダー (*externs.h*)
XDG_C_PIXMAP	C ピックスマップ (*pixmaps.h*)
XDG_CC_PROGRAM_SOURCE	メインプログラムを持つ C++ ソース (*main.cpp*)
XDG_CC_PROGRAM_OBJECT	対応するオブジェクト (*main.o*)
XDG_CC_PROGRAM	対応する実行可能ファイル (*main*)
XDG_CC_SOURCE	C++ ソース (*foo.cpp*)
XDG_CC_OBJECT	対応するオブジェクト (*foo.o*)
XDG_CC_STUB_SOURCE	C++ スタブ (stubs.cpp)
XDG_CC_STUB_OBJECT	対応するオブジェクト (*stubs.o*)
XDG_CC_EXTERN	C++ ヘッダー (*externs.h*)
XDG_CC_PIXMAP	C++ ピックスマップ (*pixmaps.h*)
XDG_JAVA_WIDGET_HDR	Java ウィジェット用のディレクトリのインクルード
XDG_JAVA_WIDGET_LIB	Java ウィジェット用のアーカイブディレクトリ
XDG_UIL_SOURCE	UIL ソース (*foo.uil*)
XDG_UIL_OBJECT	対応するコンパイル済み UIL (*foo.uid*)

表 25-1 メークファイルテンプレート記号 (続き)

名前	用途
XDG_C_FOR_UIL_PROGRAM_SOURCE	メインプログラムを持つ UIL ソース用の C (*main.c*)
XDG_C_FOR_UIL_PROGRAM_OBJECT	対応するオブジェクト (*main.o*)
XDG_C_FOR_UIL_PROGRAM	対応する実行可能ファイル (*main*)
XDG_C_FOR_UIL_SOURCE	UIL ソース用の C (*foo.c*)
XDG_C_FOR_UIL_OBJECT	対応するオブジェクト (*foo.o*)
XDG_C_FOR_UIL_EXTERN	UIL ヘッダー用の C (*externs.h*)
XDG_UIL_PIXMAP	UIL ピックスマップ (*pixmaps.uil*)
XDG_X_RESOURCE_FILE	X リソースファイル (*foo.res*)
XDG_C_DEBUG_FLAGS	C コンパイラのデバッガフラグ
XDG_CPP_DEBUG_FLAGS	C++ コンパイラのデバッガフラグ

さらに、スマートコード生成に使用するメークファイルテンプレート記号を以下の表に示します。ただし、これらの記号は Sun WorkShop Visual だけで使用できるものであって、変更することはできません。

表 25-2 スマートコード用の専用メークファイルテンプレート記号

名前	用途
XDG_CC_GROUP_OBJECT	スマートコード C++ オブジェクトファイル
XDG_CC_GROUP_SOURCE	スマートコード C++ ソースファイル
XDG_CC_SERVER_GROUP_OBJECT	サーバー側のスマートコード C++ オブジェクトファイル
XDG_CC_SERVER_GROUP_SOURCE	サーバー側のスマートコード C++ ソースファイル
XDG_CC_SERVER_PROGRAM	サーバー側のスマートコード C++ 実行可能ファイル名
XDG_CC_SERVER_PROGRAM_OBJECT	サーバー側のスマートコード C++ メインオブジェクトファイル
XDG_CC_SERVER_PROGRAM_SOURCE	サーバー側のスマートコード C++ メインソーストファイル

表 25-2 スマートコード用の専用メークファイルテンプレート記号 (続き)

名前	用途
XDG_C_GROUP_CFLAGS	C/C++ スマートコードに必要なコンパイラフラグ
XDG_C_GROUP_LFLAGS	C/C++ スマートコードに必要なリンカーフラグ
XDG_C_GROUP_OBJECT	スマートコード C オブジェクトファイル
XDG_C_GROUP_SOURCE	スマートコード C ソースファイル
XDG_C_SERVER_GROUP_CFLAGS	サーバー側のスマートコード C/C++ コンパイラフラグ
XDG_C_SERVER_GROUP_LFLAGS	サーバー側のスマートコード C/C++ リンカーフラグ
XDG_C_SERVER_GROUP_OBJECT	サーバー側のスマートコード C オブジェクトファイル
XDG_C_SERVER_GROUP_SOURCE	サーバー側のスマートコードC ソースファイル
XDG_C_SERVER_PROGRAM	サーバー側のスマートコード C 実行可能ファイル名
XDG_C_SERVER_PROGRAM_OBJECT	サーバー側のスマートコード C メインオブジェクトファイル
XDG_C_SERVER_PROGRAM_SOURCE	サーバー側のスマートコード C メインソースファイル

ダイナミックディスプレイウィンドウ

アプリケーションを独立して実行しているときと、Sun WorkShop Visual 内とでのアプリケーションの見た目と使い心地を同じにするために、ダイナミックディスプレイは個別の X リソースデータベースを実行して、独自のアプリケーションクラス名 Xdynamic を使用します。

つまり、ダイナミックディスプレイは Sun WorkShop Visual の特定のリソースを認識することはできません。Sun WorkShop Visual 内の表示内容は、コードをコンパイルして実行したときの表示内容と同じです。

ダイナミックディスプレイは独立して構成できます。これを行うには、XDdynamic ファイルを作成して、アプリケーション固有のリソースと一般的なリソースの両方をファイルに配置します。たとえば、フォームの挿入行を白黒で印刷するには、次を実行します。

```
XDdynamic*foreground: black

XDdynamic*background: white
```

アプリケーション固有のリソースの場合、次のようにして、プロダクトの Sun WorkShop Visual に表示されるアプリケーションクラス名を使用します。

```
XApplication*XmPushButton.background: #dedededededede
```

この型のアプリケーション固有のリソースは、現在のアプリケーションクラスがファイルのクラスと一致する場合に読み込むだけです。

XDdynamic ファイルは、Sun WorkShop Visual が通常のリソース検索機能を使用して認識できる場所に配置しなければなりません。詳細は、993 ページの「はじめに」を参照してください。

XDdynamic リソースはダイナミックディスプレイだけに影響し、生成されたコードには影響しません。

アプリケーションの緩い結合が XDdynamic リソースファイルの後に読み込まれて、同様に、個別のアプリケーションリソースデータベースに読み込まれます。この緩い結合は XDdynamic リソースに優先します。

リソースパネルで個々のウィジェットに適用した特定のリソースは何にも影響されることなく、常に有効です。

第26章

コマンドの要約

はじめに

本章は、Sun WorkShop Visual コマンドのクイックリファレンスガイドです。本章は以下の項目で構成されています。

- メインのメニューバー内のすべてのメニューコマンド
- その他の Sun WorkShop Visual コマンド
- オンラインヘルプの呼び出しの手順
- キーボード・アクセラレータの表

本章は、上級ユーザー向けのクイック・リファレンスとして使用されることを目的としています。コマンドについての詳細な説明は、本書の該当する章、およびオンラインヘルプを参照してください。

メニュー項目の中には、選択すると即座に実行されるものもありますが、実行される前にダイアログが表示され、ユーザーが情報を入力しなければならないものもあります。ダイアログを表示するコマンド名の後には、3 個のドット (...) が付いています。

多くのコマンドは、単一のキー入力で実行することができる、キーボード・アクセラレータを持っています。本章の終わりに、これらのアクセラレータ一覧を表に示します。

ウィジェット名と変数名

Sun WorkShop Visual メインウィンドウの上部にある 2 個のテキストボックスを使用すると、現在選択されているウィジェットに対する変数名およびウィジェット名を指定することができます。

変数名は、生成されたコードでそのウィジェットを識別するために使用される名前です。したがって、変数名は固有のものでなければなりません。変数名を指定しない場合は Sun WorkShop Visual が <ウィジェットクラス><*n*> の形式で固有の名前を割り当てます。数値 *n* は、デザインファイルを開くときに、以前と異なる値になっている場合があります。そのため、コード内ではデフォルト名は決して参照しないでください。

明示的に命名されたウィジェットの範囲は大域的です。また、明示的に命名されなかったウィジェットは、コアリソースパネル上でステータスの変更が行われない限り、その範囲は局所的です。

ウィジェット名は、変数名と同じであっても、異なっていてもかまいません。ウィジェット名はデザイン内において固有である必要はありません。ウィジェット名を共有するウィジェットのグループは、X リソースファイルに生成されているリソース設定も共有します。詳細は、241 ページの第 7 章「コードの生成」を参照してください。

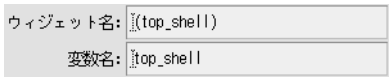


図 26-1 ウィジェット名と変数名のテキストフィールド

ファイルメニュー

Sun WorkShop Visual のファイルメニューのコマンドは、デザインファイルのみを制御します。デザインファイルは、Sun WorkShop Visual の基本ファイルであり、規約により接尾辞 .xd を持ちます。これらのファイルはデザイン階層 (複数のダイアログで構成されている場合もある)、リソースパネルで設定されているリソース値、コールバック、およびリンクを含んでいます。

新規

構成領域を消去し、新しいデザインを開始できるようにします。最後に変更を行なってから作業を保存していない場合は、構成領域を消去する前にその変更を保存するかどうかを尋ねるウィンドウが表示されます。

開く

既存のデザインファイルを開きます。ユーザーはファイル名の指定を要求されます。最後の変更を保存していない場合は、新しくファイルを読み出して置き換える前にその変更を保存するかどうかを尋ねるウィンドウが表示されます。

読む

ファイルの内容を現在のデザインに取り入れます。すべてのデザインファイルはシェルウィジェットから開始するため、このコマンドによりデザインに 1 つまたは複数のウィンドウが追加されます。

ウィジェットの変数名は、デザイン全体に対して固有にする必要があります。「読む」を使用して 2 個のデザインを組み合わせる場合、デザインにすでに存在する名前と重複する変数名は自動的に取り除かれ、デフォルト形式 <ウィジェットクラス><*n*> の局所的な名前に置き換えられます。

あるデザインの一部分のダイアログを、別のデザインと組み合わせる場合には、編集メニューの「ファイルにコピー」および「ファイルからペースト」コマンドを使用してください。

保存

すでに指定されているファイル名を使用して、現在のデザインを保存します。現在のデザインが新しく作成され、まだ保存したことがない場合は、「別名保存」の場合と同様にファイル名を指定するように要求するウィンドウが表示されます。

別名保存

現在のデザインを任意のファイル名で保存します。ファイル名を指定するための、Sun WorkShop Visual ファイルブラウザが表示されます。新しいデザインファイルを初めて保存する場合は、このコマンドを使用します。また、Java のビジュアルアプリケーションビルダーである Visaj に読み込むのに適したフォーマットでデザインを保存する場合にも、このコマンドを使用します。詳細については 402 ページの「Sun WorkShop Visual のデザインを Visaj に移行」を参照してください。

印刷

現在のデザインまたはその一部層を、プリンタまたはファイルに出力します。ファイルへの印刷を行うには、「ファイル」トグルをクリックし、「ファイル」ラベルの付いているテキストボックスにファイル名を入力します。プリンタに送る場合は、「コマンド」トグルをクリックして、「コマンド」ラベルの付いているテキストボックスに `lp` などのコマンドを入力します。出力は PostScript であるため、PostScript プリンタまたはビューア (表示プログラム) が必要です。

印刷ダイアログのオプションメニューを使用すると、用紙のサイズ、配置方向、ページおよびスケールを指定することができます。「スケール」オプションメニューで、縮小サイズを使用すると、階層が実際の 3 分の 2 の大きさで印刷されます。「適合」オプションを選択しない場合は、階層はそのままのサイズで必要なだけのページを使用して印刷されます。「ページ」オプションメニューを使用すると、デザインに複数のウィンドウが含まれている場合にデザインのすべての階層を印刷するか、現在構成領域にある階層だけを印刷するかを選ぶことができます。

「名前の表示」トグルを選択すると、変数名を印刷することができます。「印刷見出し」トグルを選択すると、階層の周辺に枠が描かれ、タイトルが印刷されます。タイトルは、「タイトル」テキストフィールドで指定することができます。また、タイトルに使用できるテキストは 1 行だけです。ここには半角の ASCII 文字のみが指定可能です。

終了

Sun WorkShop Visual を終了します。最後に変更を行なってから作業を保存していない場合は、Sun WorkShop Visual を終了する前にその変更を保存するかどうかを尋ねるウィンドウが表示されます。

編集メニュー

編集メニューには、デザイン階層を編集するためのコマンドがあります。すべての編集オプションは、現在選択されているウィジェット (階層内で選択されているすべての子を含む) 上で動作します。

カット

現在選択されているウィジェットをデザイン階層から取り除き、Sun WorkShop Visual クリップボードにコピーします。

コピー

現在選択されているウィジェットをクリップボードにコピーします。

ペースト

クリップボードの内容を、現在選択されているウィジェットの子として階層にコピーします。

クリップボードが空の場合、あるいはクリップボード内のウィジェットを現在選択されているウィジェットの子にすることができない場合は、「ペースト」は使用できません。

消去

現在選択されているウィジェットを削除します。ウィジェットが削除される前に確認ウィンドウが表示されます。「消去」されたウィジェットは、クリップボードにはコピーされないため、ペーストすることはできません。

ファイルにコピー

現在選択されているウィジェットをクリップボードファイルにコピーします。使用するファイルの名前を指定するよう要求するウィンドウが表示されます。

ファイルからペースト

クリップボードファイルの内容を、現在選択されているウィジェットの子として階層にコピーします。クリップボードファイルの名前を指定するよう要求するウィンドウが表示されます。クリップボードファイル階層のルートであるウィジェットが、現在選択されているウィジェットの子になれない場合にこの操作を行うと、「階層が無効です」というエラーメッセージが表示されます。

「ペースト」を使用する場合、ペーストされたウィジェットは常に選択されたウィジェットの最後の子になります。ウィジェットの子の順番を変更する場合は、後で説明するドラッグを使用します。

検索

「検索」ダイアログを表示します。詳細は、7-1 ページの「検索」を参照してください。

ドラッグして移動

ウィジェットアイコンのマウスボタン 1 を使用してドラッグすることにより、そのウィジェットを階層内の別の位置に移動することができます。この操作は、「カット」と「ペースト」を続けて使用した場合と同じ結果となります。

ドラッグしてコピー

ウィジェットアイコンをマウスボタン 2 を使用してドラッグすることにより、そのウィジェットをツリー内の別の位置にコピーすることができます。この操作は、「コピー」と「ペースト」を続けて使用した場合と同じ結果となります。

表示メニュー

「表示」メニューには、Sun WorkShop Visual のメインウィンドウの外観に影響するコマンドがあります。このメニューにある各コマンドはトグルであり、オンあるいはオフにすることができます。「表示」メニュー内のオプションは、Sun WorkShop Visual 画面の外観にのみ影響するものであり、デザインには影響しません。

変数名を表示

デザイン内の各ウィジェット変数名を、デザイン領域内のウィジェットアイコンの下に表示します。

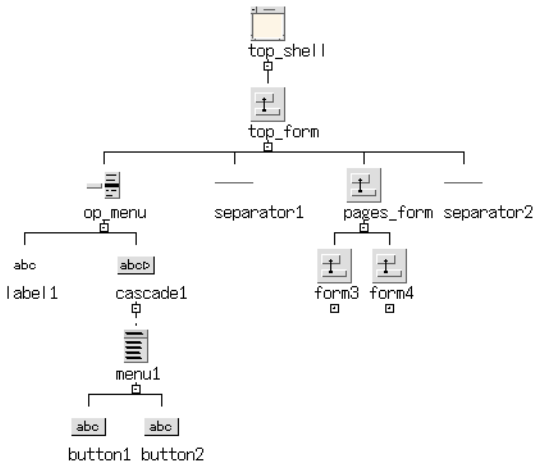


図 26-2 変数名を表示

ダイアログ変数名を表示

デザイン内の各シェルに割り当てられているウィジェットの変数名を、ウィンドウ保持領域の対応するアイコンの下に表示します。「ダイアログ変数名」は、複数のダイアログを持つデザインを操作する際に特に役に立ちます。シェルアイコンは、ダイアログ変数名を表示するスペースを空けるために縮小表示されます。

図 26-3 ウィンドウ保持領域のダイアログ変数名

ツリーの左寄せ

デザイン領域の階層の外観を、枝を左右に伸ばす中央揃えツリーから、右方向にのみ枝を伸ばす左寄せツリーに変更します。

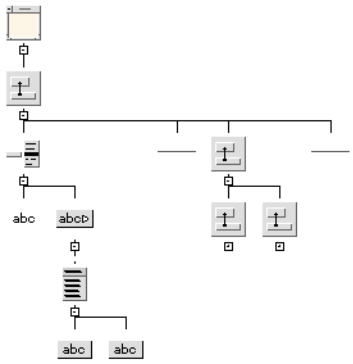


図 26-4 ツリーの左寄せ

ウィジェットの縮小

構成領域でのウィジェットのサイズを縮小します。このオプションは、構築中の階層が大きくなり、階層の全体あるいは広い範囲を表示する場合に使用します。ウィジェットは、縮小されて小さな正方形になります。その他の表示オプションと同様に、このオプションは実際のデザインには影響しません。

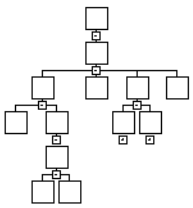


図 26-5 ウィジェットの縮小

ウィジェット注釈

指定された基準に従って、構成領域内でウィジェットに注釈を付けます。ウィジェットに次の項目が含まれている場合には、ツリーに注釈を付けることができます。

- コールバック
- メソッド
- リンク
- 作成の前プレリュード
- マネージの前プレリュード

また、ツリーに検索操作の結果を表示することもできます。注釈をオンにする場合は、注釈プルライトメニューから適切なトグルを選択します。図 26-6 に示されるように、プルライトメニューを切り離して、リファレンスとして使用することもできます。プルライトメニューを切り離すには、点線をクリックします。注釈の詳細は、51 ページの「ウィジェット注釈」を参照してください。

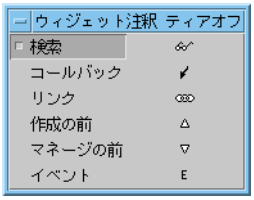


図 26-6 「注釈」ティアオフメニュー

構造の色

構成領域内のウィジェットを色で区別します。このオプションをオンにすると、階層内において、関数、データ構造体、C++ クラス、子のみのコード生成としてコアリソースパネルの「コード生成」ページを使用して指定されているウィジェットを区別するために、別々の色が指定されます。詳細は、53 ページの「構造の色」を参照してください。ウィジェットがすべてデフォルト構造の場合は、このオプションは効果を持ちません。

このオプションをオンにするには、プルライトメニューの「色を表示」トグルをオンにします。プルライトメニューは、図 26-7 のように切り離して、カラーリファレンスとして使用することができます。メニューを切り離すには、破線をクリックします。

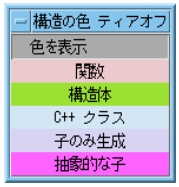


図 26-7 「構造の色」ティアオフメニュー

パレットメニュー

「パレット」メニューには、ウィジェットパレットの外観を変更するオプションが含まれています。

表示方法

パレットをアイコン、ラベル (ウィジェットのクラス名)、あるいはその両方を使用して表示するかどうかを指定します。プルライトメニューの適切なトグルボタンをオンにすると、変更が適用されます。

別々のパレット

ウィジェットパレットを別のダイアログウィンドウに表示して、デザイン領域のためのスペースを広くすることができます。分離されたウィジェットパレットからの選択および追加は通常の方法で実行することができます。

パレットを表示

分離されたウィジェットパレットを、重なり合ったウィンドウの一番上に再表示します。このオプションは、分離したウィジェットパレットを閉じてしまった場合、あるいは別のアプリケーションのウィンドウ中にパレットが紛れ込んでしまった場合に便利です。「別々のパレット」が選択されていない場合には、このオプションは使用できません。

定義する

現在選択されているウィジェットを、ウィジェット定義として指定します。このオプションを使用すると、ウィジェット定義を簡単に作成することができます。現在選択されているウィジェット (クラスとして指定されている場合) が、設定内容およびデフォルト構成を使用して、新しい定義を形成します。

定義を編集

「定義を編集」パネルを表示します。このパネルを使用すると、ウィジェット階層の一部を、ウィジェットパレットから選択できる再使用可能なオブジェクトにすることができます。

ウィジェットメニュー

ウィジェットメニューには、個々のウィジェットに適用するコマンドが含まれています。これらのすべてのコマンドは、階層内で現在選択されているウィジェットに適用されます。

リソース

現在選択されているウィジェットのリソースパネルを表示します。また、ほとんどのクラスのウィジェットのリソースパネルは、デザイン階層にあるウィジェットのアイコンをダブルクリックしても表示することができます。

リソースパネルの詳細は、59 ページの第 3 章「リソース」を参照してください。
また、851 ページの第 27 章「ウィジェット・リファレンス」でも、ウィジェットごとのリソース設定について説明しています。

コアリソース

コアリソースパネルを表示します。このパネルを使用すると、コア、プリミティブおよびマネージャ・スーパークラスから継承されるリソースを設定することができます。コアリソースは、前景と背景の色、およびウィジェットがイベントに対して応答可能であるかどうかが含まれています。

コアリソースパネルの「コード生成」ページでは、個々のウィジェットを、明示的に命名されているかに関係なく、局所的、大域的、または静的として指定することができます。C++ を使用している場合は、このページで公開、非公開、あるいは限定公開のアクセス指定子を選択されたウィジェットに指定することができます。

緩い結合

「緩い結合」ダイアログを表示します。このダイアログを使用すると、ウィジェットがリソースを共有できるように、リソースファイルでのリソースの生成方法を制御することができます。詳細は、97 ページの「緩い結合」を参照してください。

配置

配置エディタを表示します。このオプションは、フォーム、ブリテンボード、ローカラム、描画領域ウィジェットの 4 個のクラスに対して使用することができます。詳細は、109 ページの第 4 章「配置エディタ」を参照してください。

コンストレイント

区画ウィンドウまたはフォームのようなコンストレイントウィジェットの、子であるウィジェットに対してコンストレイントパネルを表示します。フォームの場合、通常、このパネルはコンストレイントリソースを再設定するためではなく、表示するためにのみ使用します。これは、配置エディタでフォームアタッチメントを設定した方が信頼性が高くなるためです。コンストレイントパネルの詳細は、59 ページの第 3 章「リソース」を参照してください。

コールバック

図 26-8 に示すような「コールバック」ダイアログを表示します。このダイアログでは、選択したウィジェットに対するコールバックを設定することができます。このダイアログについては、199 ページの「コールバックのデザイン」で詳細に説明しています。

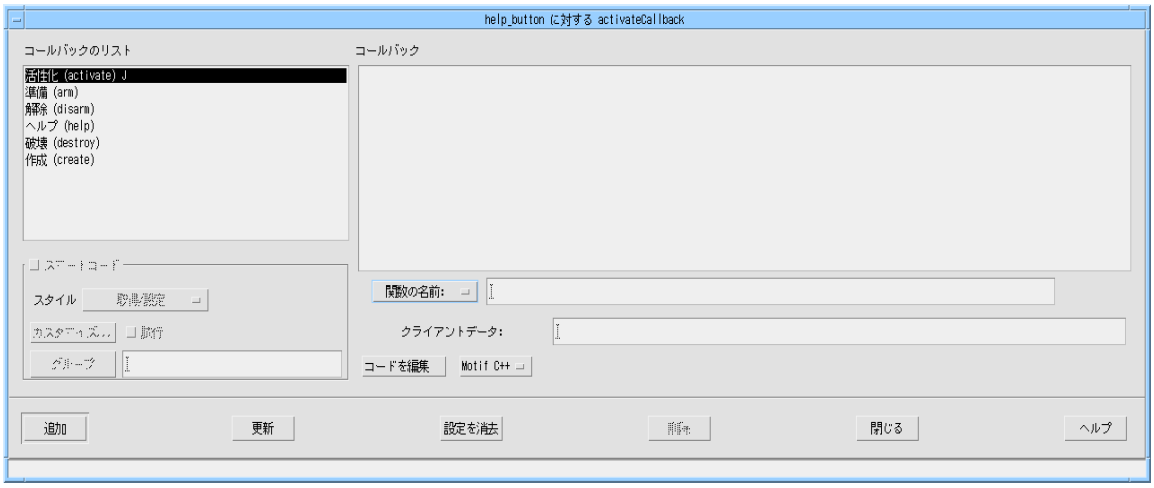


図 26-8 「コールバック」ダイアログ

イベントハンドラ

「イベントハンドラ」ダイアログを表示します。このダイアログで、ウィジェットに対して下位レベルの入力処理を追加できるため、ウィジェットのトランスレーションテーブルを使用する必要がなくなります。イベントハンドラは Sun WorkShop Visual によって一般的なウィジェット構成の一部としてコードファイルに生成され、スタブはスタブファイルに生成されます。詳細は、237 ページの「イベントハンドラ」を参照してください。

トランスレーション

現在選択されているウィジェットに対してトランスレーションを指定することができます。トランスレーションは、ウィジェット上の指定されたアクションに割り当てられたキー操作です。トランスレーションは、「上書き」または「追加」のどちらかです。「上書き」は、指定されたキー操作にすでに設定されているトランスレーションを上書きします。「追加」は、すでに設定されているトランスレーションのリストに追加を行います。

ダイアログの「置換」トグルを使用すると「上書き」が「置換」に切り替わり、ボックスに入力するトランスレーションによって、そのウィジェットにすでに設定されているいかなるトランスレーションをも置き換えられます。「置換」が設定してある場合は、「追加」テキストボックスは使用できません。

詳細は、221 ページの「トランスレーションとアクション」を参照してください。

グループに追加

「グループ・エディタ」ダイアログを表示します。このダイアログで、選択したウィジェットを既存のグループに追加できます。グループはスマートコードの主要部で、WWW に接続することのできるアプリケーションを作成する際の基盤となります。

「グループ・エディタ」ダイアログでは、549 ページの第 15 章「グループ」で説明しているように、グループを編集することもできます。568 ページの「取得と設定の学習」で、グループと基本的なスマートコードを理解するうえで役立つ簡単な演習を示しています。

新規グループに追加

選択したウィジェットを最初のメンバーとするグループを作成します。このオプションは「グループ・エディタ」ダイアログを表示します。このダイアログで、上記で説明したように、追加したばかりのグループも含めて任意の既存のグループを編集できます。

コードプレリュード

現在選択されているウィジェットに対して生成されたコードに数行のコードの追加を行うダイアログを表示します。最も一般的に使用されるプレリュードの種類は、「作成の前」、および「マネージの前」プレリュードです。「作成の前」プレリュードは、ウィジェットが作成される直前に挿入されます。「作成の前」プレリュードは、ウィジェット作成時にのみ設定可能なウィジェットリソースの設定により使用されます。

「マネージの前」プレリュードは、ウィジェットのコールバックが追加される直前に挿入されます。通常は、コールバックに対するクライアントデータ設定のために使用されます。

「コードプレリュード」を選択すると、コードを直接 (生成されたコード内で) 編集するか、ダイアログにコードを入力するかを選択して、Sun WorkShop Visual によって生成コードに追加させることができます。「コードを直接編集」を選択すると、選択したプレリュードの種類に適した行で生成コードが表示されます。プレリュード編集の詳細は、282 ページの「コードプレリュード」を参照してください。編集機能の使い方の詳細は、806 ページの「コールバックおよびプレリュード編集の設定」を参照してください。

非公開、限定公開、公開メソッドのコードプレリュードは、C++ クラスのウィジェットに対して使用することができます。詳細は、 338 ページの「クラスメンバーの追加」を参照してください。

メソッド宣言

ウィジェットの C++ クラスで、メソッドに対しての宣言を追加、削除、編集することができます。

リセット

現在選択されているウィジェットおよびそのすべての子を破壊して、再度作成します。このオプションは、リソースパネルまたは配置エディタで、特定のリソースを設定した後に役立ちます。ウィジェットを作成してからリソース値を変更すると、その値で最初からウィジェットを作成した場合とは異なる結果になる場合があります。ダイナミックディスプレイがリソース設定の変更を意図した通りに反映しない場合は、ウィジェットをリセットすると問題が解決する場合が多くあります。

リンクの編集

ウィジェットからのリンクを設定したり、取り除いたりすることができます。リンクは事前定義された活性化コールバックであり、ボタン型のウィジェットにのみ設定することができます。リンクは任意のウィジェットを表示/非表示、マネージ/アンマネージ、有効化/無効化することができます。1 つのボタンに複数のリンクを設定することができます。

フォールド / アンフォールド

選択されたウィジェットの子を非表示 (フォールド) あるいは表示 (アンフォールド) します。デザイン階層が大きくなり、構成領域内に収まらない場合には、このトグルを使用してスペースを節約します。選択されたウィジェットがフォールドされると、その子は表示されなくなりますが、デザインから取り除かれたわけではありません。フォールドされたウィジェットにはプラス記号 (+) の、アンフォールドされたウィジェットにはマイナス記号 (-) のアイコンが表示されます。このアイコンをクリックして、階層のフォールド/アンフォールドを行うこともできます。

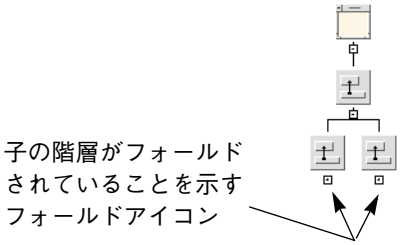


図 26-9 フォールドされたウィジェット

定義

現在選択されているウィジェットの階層を再使用可能なオブジェクト定義にします。定義は、ウィジェットパレットに挿入した後、パレット上の他のウィジェットアイコンと同様に選択することができます。

モジュールメニュー

このメニュー内のコマンドを使用すると、生成されたコードの基本モジュールが特定の場所に挿入されるコード行を指定することができます。個々のウィジェットに適用されるコードプレリュードとは異なり、モジュール・プレリュードと見出しはデザイン全体に適用されます。

緩い結合

「緩い結合」ダイアログを表示します。このダイアログを使用すると、一般的なアプリケーション全体に対してリソースの緩い結合を設定することができます。「緩い結合」ダイアログの詳細は、97 ページの「緩い結合」を参照してください。

モジュール・プレリュード

生成コードファイルの先頭部、またはその付近に挿入されるコード行を入力することができるダイアログボックスを表示します。

モジュール見出しは、基本コードモジュールおよびスタブファイルが生成される場合には、その先頭部に挿入されます。モジュール見出しは、通常、SCCS ID、バージョンまたはその他の識別情報を含めるために利用されます。

モジュール・プレリュードは、生成された Sun WorkShop Visual の *#include* 指令 (存在する場合) の後に挿入され、通常は *#include* または *#define* 指令、あるいはコードプレリュードで必要とする外部宣言に対して使用されます。

リソースプレリュードは、X リソースファイル内の Sun WorkShop Visual 生成コメントの後に挿入されます。大域的アプリケーションリソースの設定、あるいは別のリソースファイルを *#include* するために使用することができます。

ヘルプのデフォルト

ヘルプシステムに対するデフォルトを指定することができるダイアログを表示します。

デフォルトパス・フィールドは、ヘルプ文書名に使用されるパスを示します。これは、Sun WorkShop Visual で生成したコードの出力のデフォルト位置としても使用されます。

デフォルト文書フィールドは、ヘルプに対するマーカーが設定されていて、ウィジェットまたはウィジェット祖先に対しての文書が指定されていない場合に使用されます。

パスリソースおよびパス環境変数フィールドを使用すると、動的ヘルプ文書の環境設定を提供することができます。これを使用すると、指定された値を使用してデフォルトパスの設定を上書きすることができるように、コードが生成されます。実行時の環境変数は静的なリソース設定よりも優先されます。

デフォルト・トランスレーション・フィールドは、ヘルプマーカーを持つすべてのウィジェットにトランスレーションを追加します。このトランスレーションに対して関連付けられているアクションは常に「ヘルプ」です。たとえば、トランスレーション「Ctrl<key>A」は、指定されない限り、ヘルプマーカーを持つウィジェットに対して <Ctrl-A> キーによってヘルプメッセージウィンドウが表示されることを意味します。

「常に固有のウィンドウを持つ」トグルが設定されている場合、ヘルプシステムにアクセスするごとに新しいヘルプ文書ウィンドウを生成するようなコードが生成されます。

作業手続き

X ツールキットに他に処理するイベントがない場合に呼び出す手続きの名前を追加できるダイアログを表示します。そのため、X 作業手続きはバックグラウンドのバッチ処理を設定する場合に便利です。ウィジェットコールバックの場合と同様に、これらの手続きに対してスタブが生成されます。詳細は、233 ページの「Xt 作業手続き」を参照してください。

入力手続き

指定したファイルやパイプが読み取りや書き込み準備できたときに呼び出す手続きの名前を追加できるダイアログを表示します。このダイアログでは、イベントの代替ソースを使用できます。詳細は、234 ページの「入力手続き」を参照してください。

言語手続き

アプリケーションの起動時に呼び出される言語手続きの名前を指定できるダイアログを表示します。このダイアログで、ロケールの設定に必要な追加のコードを指定できます。詳細は、236 ページの「言語手続き」を参照してください。

タイムアウト手続き

指定した時間が経過した直後に呼び出される手続きの名前を追加できるダイアログを表示します。メインコードファイルに、これらの手続きに対するスタブが生成されます。詳細は、235 ページの「タイムアウト手続き」を参照してください。

アクション手続き

特定のトランスレーションに関連付けられているアクションを指定できるダイアログを表示します。ウィジェットのトランスレーションを設定する方法は 221 ページの「トランスレーションとアクション」を、アクション手続きを追加する方法は 229 ページの「追加のアクション」を参照してください。

グループ

「グループ・エディタ」ダイアログを表示します。このダイアログで、デザインのグループを編集できます。グループはスマートコードの主要部で、WWW に接続することのできるアプリケーションを作成する際の基盤となります。グループとグループエディタついての詳細は、549 ページの第 15 章「グループ」を参照してください。568 ページの「取得と設定の学習」で、グループと基本的なスマートコードを理解するうえで役立つ簡単な演習を示しています。

Java 準拠

このトグルボタンを選択すると、デザイン全体が検査され、そのデザインを Java コードとして生成できるかどうかが調べられます。この準拠検査に失敗した場合は、準拠不良を通知するダイアログが表示されます。準拠不良の一部は、希望すれば Sun WorkShop Visual で自動的に解決することができます。このダイアログについては、366 ページの「Java 準拠不良ダイアログ」のセクションを参照してください。

Microsoft Windows 準拠

このトグル (Sun WorkShop Visual が Microsoft Windows モードの場合にのみ表示される) は、デザインが Microsoft Windows 準拠であるか、つまり Microsoft Windows 対応のコード生成が可能であるかを示すために使用されます。トグルがオンに設定されている場合は、デザインが準拠していることを示します。デザインが Microsoft Windows に準拠していない場合は、このトグルがオフになり、Microsoft Windows 準拠不良ダイアログが表示されます。詳細については、425 ページの「準拠不良」を参照してください。

アプリケーションクラス

MFC および Motif XP 様式は、CWinApp クラスのインスタンスを使用して、アプリケーションを表します。アプリケーションクラスダイアログをポップアップすることにより、基底クラス名、クラス名、およびこのインスタンスのインスタンス名を変更することができます。この項目は、Sun WorkShop Visual が Microsoft Windows モードの場合にのみ表示されます。

生成メニュー

デザインが完成すると、「生成」メニューを使用してそのデザインに対してのコードを 3 種類の言語 (C、C++、UIL) で生成することができます。言語ごとにプルライトメニューがあり、生成したいファイルを選択することができます。このメニューを使用して、デザインで明示的に設定されたリソース値を含む X リソースファイル、Microsoft Windows のリソースファイル (Microsoft Windows モードで起動している場合)、および メークファイルを生成することもできます。また、「生成」オプションを選択して「コード生成」ダイアログを表示し、生成するファイルを指定することもできます。コードの生成、リンク、実行の手順については、241 ページの第 7 章「コードの生成」を参照してください。

メニューで選択した種類のファイル (C、C スタブ、C++ スタブなど) が以前に生成されていない場合にのみ、「コード生成」ダイアログが表示されます。すでに生成されている場合は、「コード生成」ダイアログは表示されずにコード生成が行われます。

「生成」メニュー内のすべてのコマンドは、「コード生成」ダイアログを使用します。「生成」メニューで、ある言語を選択すると、「コード生成」ダイアログ内の各項目が自動的にその言語用に設定されます。

「コード生成」ダイアログに入力するファイル名は、フルパス名で指定しても、ダイアログ内で指定したベースディレクトリからの相対パス名で指定してもかまいません。

「コード生成」ダイアログには、個々のファイルに関連した二次ダイアログが数多くあります。そのうちのひとつの「コードオプション」ダイアログを使用して、コード生成に関連のあるコードオプションを設定することができます。「コードオプション」ダイアログには、(コードファイルに対して) コードの各部分の生成をコントロールするオプションメニュー、 (コードおよびXリソースファイルに対して)リソース型

をコントロールするオプションメニュー、そして、リソース型と組み合せて動作するマスク方針のスイッチがあります。これらのスイッチ (図 26-10) については、241 ページの第 7 章「コードの生成」を参照してください。

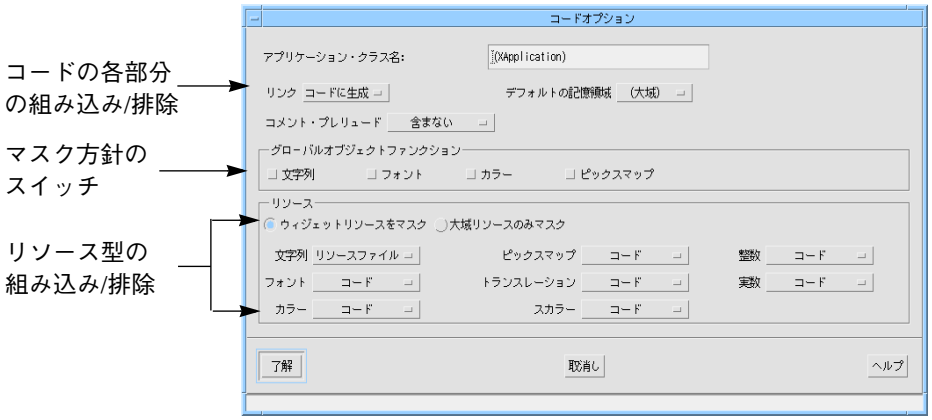


図 26-10 「コードオプション」ダイアログの設定

C

C ティアオフまたはプルライトメニューを表示します。

C++

C++ ティアオフまたはプルライトメニューを表示します。

UIL

UIL ティアオフまたはプルライトメニューを表示します。

X リソースファイル

デザインから X リソースファイルを生成します。規約により、生成された X リソースファイルは接尾辞 *.res* を持ちます。生成されたファイルを有効にするには、X で要求しているファイル名にファイルをコピーする必要があります。

X リソースファイルに対するアプリケーションクラス名は、基本モジュールを生成する場合に使用したアプリケーションクラス名と同じにする必要があります。同じクラス名を使用しないと、X はリソースと生成されたアプリケーションの関連付けを行うことができません。

Microsoft Windows リソース

Microsoft Windows リソースファイルおよび関連するビットマップ (*.bmp*) とアイコンファイル (*.ico*) を生成します。Microsoft Windows リソースファイルは、接尾辞 .rc を持ちます。このオプションは、 Sun WorkShop Visual が Microsoft Windows モードの場合にのみ表示されます。

メークファイル

アプリケーションを構築するためのメークファイルを生成します。「コード生成」ダイアログでコードファイルを指定すると、デザインファイルの中にそのファイルの名前が設定されます。コードファイルの生成を行わないと、メークファイル生成機能はファイルの名前を認識することができず、ファイルをメークファイルに追加することができません。

Java

Java コードを生成します。初めてこのオプションを選択した場合は、必要に応じてオプションを設定できるように、言語が「Java」に設定された状態で「生成」ダイアログが表示されます。Java コードを生成したことがある場合にこのオプションを選択すると、「生成」ダイアログの表示は省略され、以前に選択していたファイルが生成されます。代わりに、生成されたファイルを示すメッセージが表示されます。Java コードの生成の詳細については、389 ページの「生成ダイアログ」セクションを参照してください。

生成

図 26-11 に示すような「コード生成」ダイアログを表示します。このダイアログを使用して、ファイルを生成し、生成オプションの設定を保存すれば、生成中のファイルを一目で確認することができます。

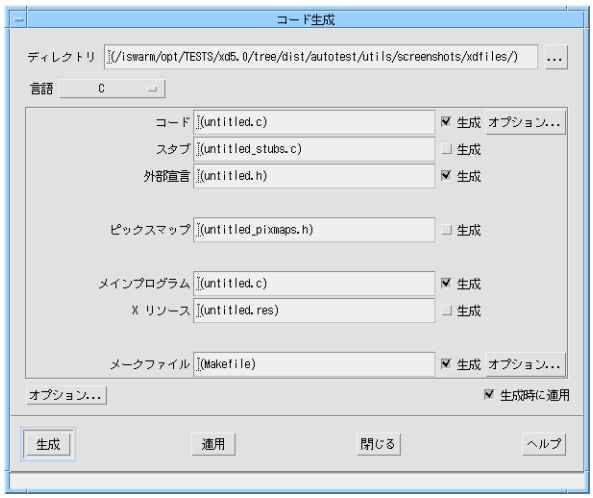


図 26-11 「コード生成」ダイアログ

ティアオフメニュー

コードの生成手順は、どの言語を選択しても基本的には同じです。C、C++、UIL のプルライトメニューは、メニュー上部にある破線をクリックして、別のウィンドウに切り離すことができます。また、メニューを切り離すことなく、プルライトメニューから通常の方法でコマンドを呼び出すこともできます。これらのメニューを表示、あるいは切り離した後、以下に示すコマンドの 1 つを選択することができます。

C、C++、UIL

基本コードモジュールを、選択した言語で生成します。

スタブ、C++ スタブ

デザインで指定されているコールバック関数に対しての関数宣言と、空の括弧を含むスタブファイルを生成します。スタブファイルを使用すると、コールバックは適切な構文を使用して確実に宣言されます。括弧の間にコードを書き込んで機能を追加することができます。

外部宣言、C++ 外部宣言、UIL 外部宣言

デザイン内のすべての大域的オブジェクトおよび関数を宣言するヘッダーファイルを構成します。大域的ウィジェットおよび定義オブジェクトに簡単にアクセスするには、このファイルを基本コードモジュールまたはスタブにインクルードします。

C、C++、UIL ピックスマップ

デザイン内にあるすべてのピックスマップの静的宣言を、別のファイルに生成します。このファイルは、ヘッダーファイルとして含まれるもので、規約により、接尾辞 *.h* を持ちます。

UIL のための C

UIL ファイルで取り扱っていない関数をアプリケーションで実行する C ファイルを生成します。このコマンドは、UIL アプリケーションにのみ適用します。UIL は Sun WorkShop Visual で提供している機能をすべて持っているわけではないため、このコマンドが存在します。最初に UIL ファイルを生成し、ファイル名を「Uid ファイル」テキストフィールドに指定します。

ツールメニュー

このメニューには、使用できる一連のツールが含まれています。ツールは、Sun WorkShop Visual からは独立していながら Sun WorkShop Visual と共同して動作するツールと、Sun WorkShop Visual に統合されているツールの 2 つの種類に分けられます。それらのツールを以降で説明します。

AppGuru デザイナー

「AppGuru デザイナー」ダイアログを表示します。このダイアログで、既存の AppGuru テンプレートを選択して編集し、新しいテンプレートを作成できます。このダイアログからテンプレートを選択すると、選択したテンプレートがデザインに追加され、再使用できる標準のインタフェースが生成されます。詳細は、482 ページの「AppGuru」を参照してください。

ピックスマップ・エディタ

ピックスマップを作成するための内部エディタです。詳細は、170 ページの「ピックスマップの編集」を参照してください。

フォント選択

フォントを選択し、フォントオブジェクトを作成するための内部ツールです。詳細は、159 ページの「フォントの設定」を参照してください。

カラー選択

色を選択し、カラーオブジェクトを作成するための内部ツールです。詳細は、154 ページの「色の設定」を参照してください。

Sun WorkShop Visual 捕獲

選択した Motif/Xt アプリケーションのデザインを捕獲する独立したツールです。デザインの捕獲の詳細は、489 ページの「Sun WorkShop Visual 捕獲機能」を参照してください。

Sun WorkShop Visual 再現

選択した Motif/Xt アプリケーションの使用内容を記録し、記録したスクリプトを再生することができる独立したツールです。記録と再生の詳細は、第 14 章「Sun WorkShop Visual 再現機能」を参照してください。

ヘルプメニュー

ヘルプメニューからは、一般的なヘルプメッセージが参照できます。

特定のヘルプを使用するには、そのダイアログボックス内の「ヘルプ」ボタンをクリックしてください。Sun WorkShop Visual のヘルプは、ハイパーテキスト・ネットワークとして組織されています。各ヘルプ画面は、図 26-12 に示すように、関連項目のリストを表示します。関連項目のヘルプを表示するには、その項目をダブルクリックしてください。

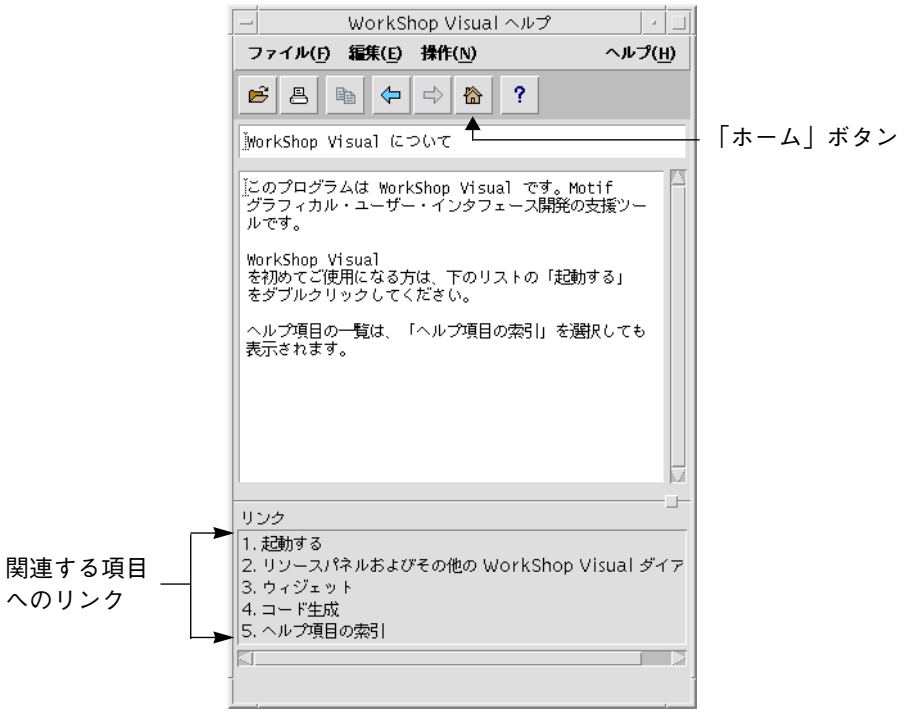


図 26-12 ヘルプビューア

「ホーム」ボタンをクリックすると、ヘルプ画面の先頭部に移動します。すべてのヘルプ項目のリストを表示するには、「ヘルプ項目の索引」をダブルクリックしてください。

Sun WorkShop Visual について

Sun WorkShop Visual の著作権画面およびソフトウェアのバージョンを表示します。

ウィジェットパレット

すべての Motif ウィジェットのアイコンの画面を表示します。アイコンをクリックすると、そのウィジェットクラスについての情報を得ることができます。ウィジェットパレットの画面は、Sun WorkShop Visual のメインウィンドウとは別にアイコン化することができます。

ヘルプ

ハイパーテキスト索引の最上位ヘルプ画面を表示します。この画面には、一般的なヘルプメッセージと、詳細な情報を得るためのリンクがあります。

ビューア

「ビューア」プルライトメニューで、Sun WorkShop Visual ヘルプの表示に使用するアプリケーションを選択することができます。メニューには、次の 2 つの項目があります。

- Netscape を使用
  一般的な HTML ブラウザ
- WorkShop Visual ヘルプを使用
  Sun WorkShop Visual 独自の HTML ブラウザ

キーボードによる短縮操作

以下の表に、Sun WorkShop Visual コマンドに対してのキーボード・アクセラレータを示します。

表 26-1 Sun WorkShop Visual コマンドのキーボード・アクセラレータ

メニュー	コマンド	アクセラレータ
ファイル	新規	Ctrl+N
	開く	Ctrl+O
	読む	Ctrl+R
	保存	Ctrl+S
	別名保存	Ctrl+A
	印刷	Ctrl+P
	終了	Ctrl+E
モジュール	モジュール・プレリュード	Ctrl+F9
編集	カット	<keypad>Cut[1]
	コピー	

表 26-1 Sun WorkShop Visual コマンドのキーボード・アクセラレータ (続き)

メニュー	コマンド	アクセラレータ
	ペースト	<keypad>Paste
	消去	Del
表示	ウィジェット名を表示	Ctrl+W
	ダイアログ名を表示	Ctrl+D
	ツリーの左寄せ	Ctrl+L
	ウィジェットの縮小	Ctrl+F3
ウィジェット	リソース	Ctrl+F4
	コアリソース	Ctrl+C
	コンストレイント	Ctrl+F5
	トランスレーション	Ctrl+F6
	コードプレリュード	Ctrl+F7
	リセット	Ctrl+T
	リンク編集	Ctrl+F8
	フォールド/アンフォールド	Ctrl+F
生成	C	Alt+C
	UIL	Alt+U
	UIL のための C	Alt+L
	X リソース	Alt+X
	メークファイル	Alt+K
	Java	Alt+J
ピックスマップエディタ	新規	Ctrl+N
	開く	Ctrl+O
	保存	Ctrl+S
	閉じる	Ctrl+Y
	元に戻す	Alt+BackSpace
	カット	<keypad>Cut
	コピー	<keypad>Copy
	ペースト	<keypad>Paste
	消去	Del

表 26-1 Sun WorkShop Visual コマンドのキーボード・アクセラレータ (続き)

メニュー	コマンド	アクセラレータ
	すべて選択	Ctrl+A
	サイズ変更	Ctrl+R
	パレットを編集	Ctrl+P
配置エディタ	閉じる	Ctrl+Y
	元に戻す	Ctrl+Z
	リセット	Ctrl+T
	ウィジェット名	Ctrl+W
	クラス名	Ctrl+N
	端の強調表示	Ctrl+E
ヘルプ	Sun WorkShop Visual について	Ctrl+F10
	ヘルプ	F1 または Help

1.<keypad> は、キーパッドのキーであることを示します。

第27章

ウィジェット・リファレンス

はじめに

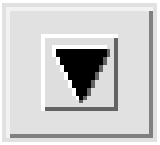
本章では、ウィジェットパレットにある各 Motif ウィジェットの概要と、効果的な使用方法について説明します。また、各ウィジェットの特徴についても説明します。ただし、ここでは基本的な情報のみを提供するものとします。各ウィジェットのすべてのリソースを含む詳細な記述については、『Motif プログラマーズ・リファレンス』を参照してください。

各ウィジェットについて、リソースパネルのページごとにリソースを分類して記述しています。また、そのウィジェットに対して使用できるコールバックも記述しています。ウィジェットに対するコールバックは、厳密に言えばリソースの一部ですが、コールバックダイアログを使用して個別に編集および表示することができます。

繰り返しを防ぐため、コアリソースは含まれていません。**太字**で示されているリソースは頻繁に設定されるものであり、経験のレベルとは関係なく、どのユーザーにも重要です。標準の字体で示されているリソースは、太字のリソースに比べると設定される頻度が少なく、効果的に使いこなすためにはある程度の知識を必要とするものです。「ラバーポジショニング」のように小さい文字サイズで示されているリソースは、そのクラスには使用できません。リソースパネルではグレー表示されていて、選択することはできません。

注 - `small_visu` コマンドを使用して Sun WorkShop Visual を呼び出す場合は、ウィジェットのアイコンがやや小さく、本章で示されているものとは少し異なります。

矢印ボタン (ArrowButton)



設定: 矢印の方向

トグル: ウィジェット
ガジェット

コールバック: 活性化 (activate)
準備 (arm)
解除 (disarm)

矢印ボタンウィジェットは、テキストラベルの代わりに矢印を持つボタンを提供します。この矢印は、上下左右を指すことができます。図 27-1 に示す矢印ボタンリソースパネルのオプションメニューから適切な方向を選択して、4 方向の中から 1 つを選択します。

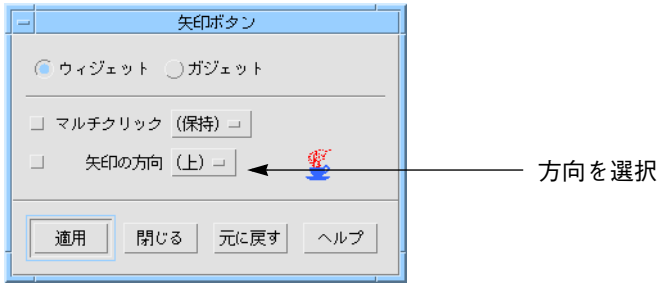
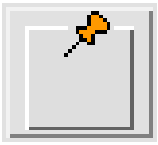


図 27-1 矢印ボタンリソースパネル

他のボタンウィジェットとは異なり、矢印ボタンはラベルから派生するものではありません。したがって、テキストまたはピックスマップラベルを表示することはできません。

ブリテンボード (BulletinBoard)



表示:	タイトル	設定:	ダイアログのスタイル
	マージンの高さ		サイズ変更方針
	マージンの幅		シャドウの種類
	横の間隔		オーバーラップ可
	縦の間隔		自動アンマネージ
	分数分母		デフォルトの位置
	取消しボタン		サイズ変更なし
	デフォルトボタン		ラバーポジショニング
フォント:	テキスト用フォント	コールバック:	フォーカス (focus)
	ボタン用フォント		マップ (map)
	ラベル用フォント		アンマップ (unmap)

ブリテンボードウィジェットは、最も基本的なコンテナウィジェットです。このウィジェットは、フォーム、選択ボックスおよびメッセージボックスなど、他のコンテナまたはコンポジットウィジェットを Motif で実装するために、最も一般的に内部で使用されます。多くの場合、これらの派生ウィジェットの方が、ブリテンボードそのものよりも便利です。

ブリテンボードは、コンテナウィジェットとして、子に特定の配置を強制しません。ブリテンボードは絶対位置とマージンの制約を提供し、その内部のウィジェットのオーバーラップを可能にするかどうかはユーザーが指定します。ブリテンボードのサイズ変更を行なっても、内部のウィジェットの移動あるいはサイズ変更は行われません。

ブリテンボードは、サイズ変更を行うことのない一時的ダイアログに対して大変役に立ちます。サイズ変更が可能なダイアログに対しては、フォームまたはダイアログテンプレートを使用してください。また、複雑な位置決めのために C コードの作成が必要である場合にもブリテンボードを使用することができます。

配置エディタでは、移動とサイズ変更オプションのみを使用することができます。ウィジェット間のアタッチメントおよび位置アタッチメントは使用できません。柔軟性の高い配置オプションが必要な場合には、フォームウィジェットを使用してください。

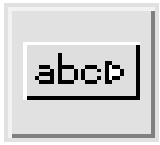
ブリテンボードがシェルの子である場合は、そのブリテンボードの「タイトル」リソースは、 シェルウィンドウのタイトルとして使用されます。

ブリテンボードがフォームの子である場合は、「タイトル」、「ダイアログのスタイル」、「デフォルトの位置」、「サイズ変更なし」は使用できなくなります。

「自動アンマネージ」リソースが「はい」に設定されていると、 ブリテンボードの子であるボタンをクリックするとダイアログが非表示になります。Motif でのデフォルト動作であるこの動作は、一時的ダイアログに対しては便利ですが、メインウィンドウにおいては混乱を招く場合があります。Sun WorkShop Visual は、ブリテンボードおよびその 2 個の派生ウィジェット (ダイアログテンプレートおよびフォーム) に対し、このリソースを明示的に「いいえ」に設定します。ブリテンボードから派生したその他のウィジェットに対しては、Sun WorkShop Visual はデフォルトの「はい」という設定の書き換えは行わないため、ユーザーが何らかのボタンをクリックするとダイアログは非表示になります。ダイナミックディスプレイを復元するためには、シェルをリセットするか、あるいはウィンドウ保持領域にあるシェルアイコンを選択します。

「デフォルトの位置」リソースは、画面上のウィンドウの位置がどのように決定されるかを制御します。シェルの子であるブリテンボード (または派生ウィジェット) 上でこのリソースを「いいえ」に設定すると、ウィンドウはシェルではなくブリテンボードの X および Y リソースが決定した位置に表示されます。この動作はウィンドウマネージャにより異なるため、一貫性はありません。

カスケードボタン (CascadeButton)



表示:	ラベル	マージン:	上
	フォント		下
	ピックスマップ		左
	応答不可能ピックスマップ		右
	カスケード・ピックスマップ		幅
	アームの色		高さ
	アームピックスマップ		間隔
	選択ピックスマップ		デフォルトのシャドウ
	選択応答不可能ピックスマップ		インジケータサイズ
トグル:	ウィジェット	キーボード:	アクセラレータ
	ガジェット		アクセラレータテキスト
			ニーモニック
			ニーモニック文字セット
			マッピング遅延
コールバック:	活性化 (activate)	設定:	揃え方
	カスケード (cascading)		種類
	準備 (arm)		サイズ変更
	解除 (disarm)		プッシュボタン
	露出 (expose)		シャドウの種類
	サイズ変更 (resize)		アーム時に塗りつぶし
	値変更 (value changed)		選択時に塗りつぶし
			インジケータ・オン
			インジケータタイプ
			マルチクリック
			オン
			オフの時に表示

カスケードボタンウィジェットは、メニューを表示するために使用されます。カスケードボタンは、メニューバーまたはメニューの子としてのみ使用することができます。メニューバーの子である場合にはメニューを表示し、メニューの子である場合にはプルライトメニューを表示します。メニューウィジェットの記述には、このようなカスケードボタンの使用法を示した簡単な階層例があります。

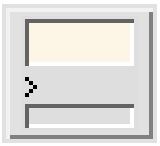
カスケードボタンでは、メニューだけを子として持つことができます。ユーザーがカスケードボタンをクリックすると、メニューが表示されます。

Motif においては、メニューはカスケードボタンの子ではなく、カスケードボタンの親の子です。これを示すため、カスケードボタンとそのメニューの間の接続は通常の実線ではなく、点線で描かれています。

カスケードボタンにキーボードニーモニックを設定し、ユーザーがマウスを使用することなくメニュー間を行き来できるようにすることが可能です。

マッピング遅延リソースは、ボタンがプルライトメニューの起動に使用される場合にのみ使用可能です。

コマンド (Command)



表示:
一致する文字列なし
パターン
最大履歴項目数
コマンド
履歴項目数
テキスト桁数
ディレクトリのマスク
ディレクトリ

ラベル:
適用ラベル
了解ラベル
取消しラベル
ヘルプのラベル
リスト用ラベル
プロンプト文字列
ディレクトリのラベル

設定:
ダイアログ型
ボタン最小化
既存項目と要一致
ファイルの種類
作業領域の場所

コールバック:
適用 (apply)
取消し (cancel)
了解 (ok)
一致なし (no match)
コマンド変更 (command changed)
コマンド入力 (command entered)

コマンドウィジェットは、コマンド履歴のスクロール可能リストからコマンドを選択するために使用するコンポジットウィジェットです。コマンドは、ウィジェットの下部にあるテキスト領域にタイプ入力することができます。コマンドが入力された場合、そのコマンドは履歴リストの末尾に追加されます。コマンドは、選択ボックスか

ら派生しています。また、一部のブリテンボードのリソースを継承します。ブリテンボードのリソースパネルを表示するためには、リソースパネル内の「ブリテンボードのリソース」をクリックします。

コマンドは、コマンド履歴領域に対するスクロールリストウィジェット、コマンド行プロンプトのためのラベル、コマンド入力領域のためのテキストウィジェットを含んでいます。これらの構成要素は、デザイン階層には表示されていないブリテンボードウィジェットに含まれています。構成要素ウィジェットに対するデフォルトリソース設定は変更することができますが、削除することはできません。プロンプトを変更するためには、その子であるラベルのリソースパネルではなく、コマンドのリソースパネルのラベルページにある「プロンプト文字列」リソースを修正します。

コマンドは、通常、シェルまたはメインウィンドウ内で使用されます。

コマンドには複数の子を追加することができます。最初の子は作業領域になります。これは、追加のウィジェットを含むコンテナウィジェットとすることができます。図 27-2 では、作業領域がコマンドウィジェットの階層の最後に表示されていますが、ダイアログに作業領域が表示される場所は「作業領域の場所」リソースが制御します。追加の子には、メニューバーおよび任意の数のプッシュボタンウィジェットを持たせることができます。

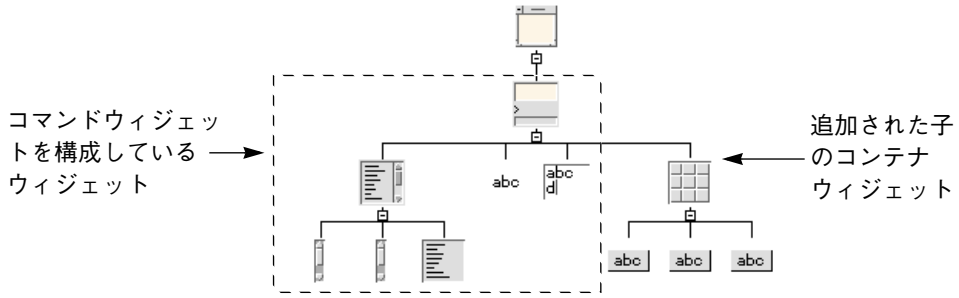
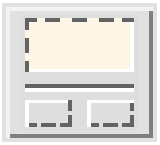


図 27-2 コマンドウィジェット階層

ダイアログテンプレート (DialogTemplate)



表示:
メッセージテキスト
了解ラベル
取り消しラベル
ヘルプラベル
記号用ピックスマップ

設定:
デフォルトボタン
ダイアログの種類
揃え方
ボタン最小化

コールバック:
取消し (cancel)
了解 (ok)

ダイアログテンプレートウィジェットは、通常シェルの子として多様な種類のダイアログに対して使用されます。このウィジェットは、上から、メニューバー、作業領域、セパレータ、ボタンボックスとする標準配置を提供します。

ダイアログテンプレートは、特別に構成されているメッセージボックスであり、 メッセージボックスのリソースパネルを共有します。また、一部のブリテンボードのリソースを継承しています。ブリテンボードのリソースパネルを表示するには、リソースパネルにある「ブリテンボードのリソース」をクリックします。

セパレータは、ダイアログテンプレートの構成要素部品です。セパレータを使用する場合には、図 27-3 のようにメニューバー、作業領域に対しての何らかの型のウィジェット、ボタンボックスのための任意の型のボタンなど、他の標準配置の要素を追加する必要があります。作業領域は、フォームなどのように、子を持つコンテナウィジェットとすることができます。ダイアログテンプレートは、階層に追加する順序とは関係なく、常に下から上への標準順序で子を配列します。

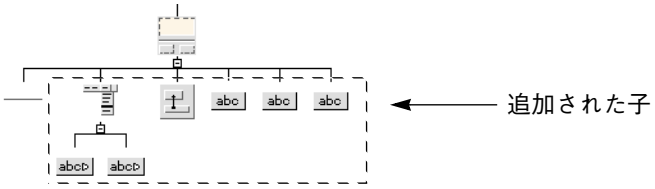
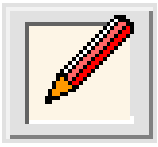


図 27-3 ダイアログテンプレートを使用した標準階層

標準配置の領域は、強制的に同じ幅になり、ボタンボックス内のボタンは 1 行または複数行に等間隔で配置されます。ウィンドウがサイズ変更されると、必要に応じてボタンは自動的に並べ替えられます。

描画領域 (DrawingArea)



マージン: 幅
高さ

サイズ変更の方針

コールバック: 露出 (expose)
入力 (input)
サイズ変更 (resize)

描画領域ウィジェットは、アプリケーションが出力グラフィックスを表示する領域を提供します。たとえば、Sun WorkShop Visual メインウィンドウにあるデザイン階層は、スクロールウィンドウに含まれている描画領域に描かれます。

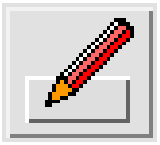
配置エディタを表示するには、ウィジェットメニューから「配置」を選択するか、またはツールバーの「配置」ボタンを押します。配置エディタでは移動およびサイズ変更オプションのみを使用することができます。ウィジェット間のアタッチメントおよび位置アタッチメントは使用できません。

リソースパネルを表示するには、ウィジェットメニューから「リソース」を選択するか、またはウィジェットをダブルクリックします。

描画領域は、任意の数の任意の型の子を持つことができますが、他のウィジェットの物理的な配置の管理にはあまり役に立ちません。この目的のためには、フォームなどの他のコンテナウィジェットを使用してください。

Sun WorkShop Visual では、描画領域内での描画は使用できません。描画領域内で描画を行うためには、X グラフィックス・コールを含んでいるコードを作成する必要があります。このコードは、通常「露出 (expose)」コールバックに入れられます。

描画ボタン (DrawnButton)



表示:
- ラベル
- フォント
- ピックスマップ
- 応答不能状態用ピックスマップ
- カスケード・ピックスマップ
- 準備 (arm) 時のカラー
- 準備 (arm) 時のピックスマップ
- 選択時の色
- 選択時のピックスマップ
- 選択時の応答不能状態用ピックスマップ

設定:
- 整列
- 種類
- サイズ変更
- プッシュボタン
- シャドウの種類
- 準備時に塗りつぶし
- 選択時に塗りつぶし
- インジケータオン
- インジケータタイプ
- マルチクリック
- オン
- オフのとき表示

マージン:
- 上
- 下
- 左
- 右
- 幅
- 高さ
- 間隔
- デフォルトのシャドウ
- インジケータサイズ

コールバック:
- 活性化 (activate)
- カスケード (cascading)
- 準備 (arm)
- 解除 (disarm)
- 露出 (expose)
- サイズ変更 (resize)
- 値変更 (value changed)

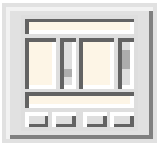
キーボード:
- アクセラレータ
- アクセラレータテキスト
- ニーモニック
- ニーモニック文字セット
- マッピング遅延

トグル:
- ウィジェット
- ガジェット

描画ボタンウィジェットは、プッシュボタンに似ていますが、ボタン面が自動的に描かれるのではなく、アプリケーションで描画しなければならないという点が異なります。このウィジェットを使用して、状況に応じた外観を持つボタンを提供することができます。

ボタン上に絵を表示する場合は、通常、プッシュボタンとイメージのピックスマップを一緒に使用して実行できます。描画ボタンに絵を描くためには、X グラフィックス・コールを含んでいるコードを作成する必要があります。このコードは、通常「露出 (expose)」コールバックに入れられます。

ファイル選択ボックス (FileSelectionBox)



表示:
一致する文字列なし
パターン
最大履歴項目数
ディレクトリ指定
可視項目数
テキスト文字数
ディレクトリ・マスク
ディレクトリ

ラベル:
適用ラベル
了解ラベル
取り消しラベル
ヘルプのラベル
ファイルリスト用ラベル
選択用ラベル
フィルタのラベル
ディレクトリのラベル

設定:
ダイアログ種類
ボタン最小化
既存項目と要一致
ファイル種類
作業領域の場所

コールバック:
適用 (apply)
取消し (cancel)
了解 (ok)
一致なし (no match)
コマンド変更 (command changed)
コマンド入力 (command entered)

ファイル選択ボックスウィジェットは、ユーザーがファイルシステム内でブラウズおよびファイルの選択を行うためのコンポジットウィジェットです。Sun WorkShop Visual のファイルブラウザは、ファイル選択ボックスの 1 つの例です。生成ダイアログは、子として作業領域を持っているファイル選択ボックスです。ファイル選択ボックスは、選択ボックスから派生され、そのリソースパネルを共有します。

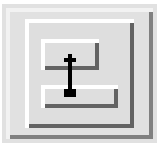
ファイル選択ボックスの組み合せは、2 個のスクロールリスト、2 個のテキストフィールド、4 個のラベル、1 個のセパレータ、4 個のプッシュボタンを含んでいます。これらはガジェットです。その構成要素は、デザイン階層には表示されていないブリテンボードウィジェットに含まれています。ブリテンボードから継承されたリソースを表示するためには、リソースパネル内の「ブリテンボードのリソース」をクリックします。

ファイル選択ボックスは、ブリテンボードが使用できる場所であれば、どの場合でも使用することができますが、通常はファイル選択のためにポップアップされるダイアログシェルに配置されます。

ボタンのラベルまたはラベルウィジェットのデフォルトを変更するためには、個々のウィジェットのリソースパネルではなく、ファイル選択ボックスのリソースパネルにあるリソースを修正します。

ファイル選択ボックスには、複数の子を追加することができます。最初の子は作業領域になります。この作業領域は、追加のウィジェットを含むコンテナウィジェットとすることができます。また、作業領域はファイル選択ボックスウィジェットの階層の最後に表示されますが、ダイアログに作業領域が表示される場所は「作業領域の場所」リソースによって制御されます。追加された子にはメニューバーおよび任意の数のプッシュボタンウィジェットを持たせることができます。

フォーム (Form)



表示:	タイトル	設定:	ダイアログスタイル
	マージンの高さ		サイズ変更方針
	マージンの幅		シャドウの種類
	横の間隔		オーバーラップ可
	縦の間隔		自動アンマネージ
	分数分母		デフォルトの位置
	取り消しボタン		サイズ変更なし
	デフォルトボタン		ラバーポジショニング
フォント:	テキスト用フォント	コールバック:	フォーカス (focus)
	ボタン用フォント		マップ (map)
	ラベル用フォント		アンマップ (unmap)

フォームウィジェットは、その子ウィジェットの絶対および相対位置を提供するコンテナウィジェットです。通常は、ダイアログ内にシェルの子として、あるいはダイアログテンプレートの類のウィジェットにある作業領域として、ウィジェットを配置するために使用されます。

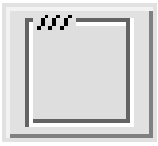
フォーム内でのウィジェットの配置は、フォームの子についてのアタッチメントを使用して指定されます。異なる種類のアタッチメントを使用すると、フォーム内での固定位置、フォーム内での相対位置、あるいはウィジェット間の固定間隔など、さまざまな位置関係を指定することができます。これらの機能により、ウィジェットやウィンドウがサイズ変更された場合に、柔軟で確実な対応をすることができます。Sun WorkShop Visual では、配置エディタを使用して、これらのアタッチメントを対話的に指定することができます。詳細は、第 4 章「配置エディタ」を参照してください。

コンストレイントパネルを使用すると、フォームの子に設定されているアタッチメントを表示することができます。フォームの任意の子を選択し、「ウィジェット」メニューをプルダウンして「コンストレイント」を選択します。このパネルの使用方法は、第 3 章「リソース」に説明されています。「コンストレイント」パネルは、あるウィジェットを別のウィジェットに付加したい場合に大変便利です。

「自動アンマネージ」リソースが「はい」に設定されていると、フォームの子であるボタンをクリックするごとにダイアログが消去されます。Motif でのデフォルト動作であるこの動作は、一時的に表示されるダイアログに対しては便利です。しかし、フォームはメインウィンドウに対して使用される場合が多いため、Sun WorkShop Visual はフォームに対しては、このリソースを意図的に「いいえ」に設定しています。フォームを自動アンマネージするためには、「はい」に設定してください。

詳細は、「ブリテンボード (BulletinBoard)」を参照してください。

フレーム (Frame)



表示:		
	マージンの幅	タイトルウィジェット
	マージンの高さ	タイトル間隔
	シャドウの種類	タイトルの揃え方 (水平)
		タイトルの揃え方 (垂直)

フレームウィジェットは、他には何もないウィジェットの周辺に境界枠 (タイトルが一緒の場合もある) を提供します。ウィジェットがすでに境界フレームを持っている場合にはそのフレームを強調するか、ウィジェットグループの周辺に境界フレームを作成します。フレームを使用すると、描画領域がへこんでいるように表示させるなどの 3 次元的な効果を提供することができます。

フレームには 2 個の子を持たせることができます。最初の子はフレームの内側に置かれ、次の子 (任意) はタイトルとして使用されます。2 番目の子は通常はラベルです。

ウィジェットグループの周辺に境界フレームを作成するには、図 27-4 に示すように、

それらのウィジェットがフレームの子であるローカラムまたはフォームなどのコンテナウィジェットに置かれている必要があります。

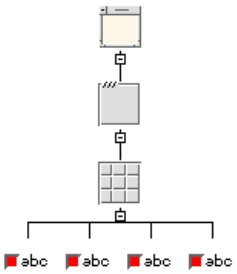


図 27-4 フレームウィジェットを境界フレームとして表示する階層

フレームは、その子のサイズに合わせて、自身のサイズを変更します。

ラベル (Label)

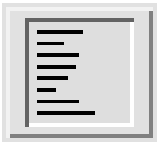
表示:	ラベル	設定:	整列
	フォント		種類
	ピックスマップ		サイズ変更
	応答不能状態用ピックスマップ		プッシュボタン
	カスケード・ピックスマップ		シャドウの種類
	準備 (arm) 時のカラー		準備時に塗りつぶし
	準備 (arm) 時のピックスマップ		選択時に塗りつぶし
	選択時の色		インジケータ・オン
	選択時のピックスマップ		インジケータタイプ
	選択時の応答不能状態用ピックスマップ		マルチクリック
			オン
			オフのとき表示
マージン:	上	コールバック:	活性化 (activate)
	下		カスケード (cascading)
	左		準備 (arm)
	右		解除 (disarm)
	幅		露出 (expose)
	高さ		サイズ変更 (resize)
	間隔		値変更 (value changed)
	デフォルトのシャドウ		
	インジケータサイズ		
キーボード:	アクセラレータ	トグル:	ウィジェット
	アクセラレータテキスト		ガジェット
	ニーモニック		
	ニーモニック文字セット		
	マッピング遅延		

ラベルウィジェットは、テキストまたはピックスマップイメージに対しての静的表示領域を提供します。ラベルは、一般に、記述的なテキスト文字列、アイコンあるいはロゴの表示に使用されます。ラベルをメニュー内に配置して、メニュー項目のグループを表すタイトル (選択不能) にすることができます。

ラベル内の文字列には、複数の行およびフォントを使用することができます。複数のフォントは、コンパウンド文字列エディタを使用してサポートされます。コンパウンド文字列エディタについては、186 ページの「コンパウンド文字列」を参照してください。

ラベルに対してピックスマップを設定した場合、ラベルは「種類」の設定を「ピックスマップ」に変更するまで、ピックスマップを表示しません。

リスト (List)



表示:	マージンの幅	**コールバック:**	ブラウズ (browse)
	マージンの高さ		デフォルト (default)
	項目間隔		拡張 (extended)
	可視項目数		複数 (multiple)
	最上部の項目		単一 (single)
	ダブルクリックの間隔		
	フォント		
設定:	自動選択	**項目:**	項目
	選択の方針		
	サイズ決定の方針		
	スクロールバー表示方針		

リストウィジェットは、テキスト項目のリストを表示するために使用されます。テキスト項目は、「選択の方針」リソースの設定にもとづき、単一の、あるいは複数の選択が可能となります。

テキスト項目のスクロールリストが必要な場合は、リストウィジェットを含むコンポジットウィジェットである、スクロールリストウィジェットを使用してください。

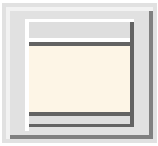
リストのリソースパネルの「項目」ページを使用すると、リストに項目を追加することができ、リストがどのように表示されるかを確認することができます。項目を追加するには、テキストを「項目」リソースボックスに入力し、「追加」を選択します。リストから項目を削除するには、テキストを「項目」リソースボックスに入力し、「削除」を選択します。項目の追加は、表示したい順序で行う必要がありますが、削除はどのような順序で行なってもかまいません。

追加の項目を確認するには、「可視項目数」リソースを変更します。

Motif ツールキットは、項目の追加、削除、置換など、リストを操作するための多数の関数を提供しています。詳細は、Motif のマニュアルを参照してください。

リスト内の各項目はコンパウンド文字列 (XmString) であることに注意してください。このため、理論的には項目ごとに、あるいは単一項目内の部分ごとに異なるフォントを使用することが可能です。しかし、実際にはMotif ツールキットの制限があるため、複数のフォントを使用することはお勧めできません。

メインウィンドウ (MainWindow)



スクロールウィンドウのマージン:	幅 高さ 間隔	メインウィンドウのマージン:	幅 高さ
コールバック:	隠れたものも走査 (traverse obscured)	設定:	スクロールバー表示方針 スクロールバーの位置 スクロールの方針 表示の方針 セパレータの表示 コマンドの位置 メッセージウィンドウ

メインウィンドウウィジェットを使用すると、アプリケーションのメインウィンドウ用の標準的な配置を行うことができます。標準配置には、上から下へ順番に、次のものが含まれています。

- メニューバー
- 履歴、プロンプトおよび入力領域を持つコマンド領域
- 作業領域
- メッセージ領域

メインウィンドウは、3 個のセパレータと 2 個のスクロールバーを持つコンポジットウィジェットです。標準配置の各要素に対して、ウィジェットを追加する必要があります。メニューバーにはメニューバーウィジェットを、コマンド領域にはコマンド

ウィジェットを使用します。メッセージ領域には、通常、テキストまたはテキストフィールドウィジェットが使用されます。メッセージ領域には、変数名を与え、その名前をメインウィンドウの「メッセージウィンドウ」リソースとして指定する必要があります。

その他、ほとんどの種類のウィジェットは作業領域にすることができます。子として他のウィジェットを持つコンテナウィジェットも、作業領域として使用することができます。メインウィンドウは通常、サイズが固定されている作業領域上にスクロールウィンドウを表示します。作業領域がフォームである場合は、「スクロールの方針」リソースを「アプリケーションにより設定」に変更する必要があるかもしれません。リソースの変更を行うと、スクロールバーが取り除かれ、ウィンドウとともにフォームのサイズ変更が可能となります。したがって、ユーザーが配置エディタの機能を使用して、サイズ変更動作を制御することができるようになります。

注 - 「スクロールの方針」は、ダイナミックディスプレイには影響を与えませんが、生成されたコードでは正確に動作します。

メインウィンドウに作業領域を追加しない場合は、生成されたコードは実行時に警告メッセージを発します。

リソースを効果的に使用することにより、フォームにメインウィンドウの動作をエミュレートさせることができます。多くの場合、この方法がより便利なようです。

メニュー (Menu)

表示:	エントリのフレーム	**設定:**	配置方向
	マージンの幅		間隔設定方法
	マージンの高さ		揃え方
	列数		最後を調整
	間隔		マージンの調整
	ヘルプ・ウィジェット		整列
	最後の選択項目		同種の子
			ポップアップ可 [1]
			常にラジオボタン 1 個選択
			ラジオボタン動作
			高さのサイズ変更
			幅のサイズ変更
			ティアオフ
キーボード:	アクセラレータ [1]	**コールバック:**	マップ (map)
	メニューポスト [1]		アンマップ (unmap)
	ニーモニック		進入 (entry)
	ニーモニック文字セット		

1. ポップアップメニューとして使用される場合のみに適用します。その他の場合には使用できません。

メニューウィジェットは、プルダウン、プルライトおよびポップアップメニューを持つ、特別に構成されたローカラムウィジェットです。

メニューの動作項目は、プッシュボタン、トグルボタン、またはカスケードボタンです。メニューは、表示の目的でセパレータおよびラベルを含むことができます。

メニューを作成するためには、メニューバーまたはオプションメニューの子であるカスケードボタンの子としてメニューを追加します。ユーザーがカスケードボタンをクリックすると、メニューが表示されます。

プルライトメニューを作成するためには、メニューの子であるカスケードボタンの子としてメニューを追加します。ユーザーがカスケードボタンをクリックすると、メニューが表示されます。プルライトメニューは、オプションメニューからではなく、メニューバーからプルダウンされるメニューでのみ使用することができます。

ポップアップメニューを作成するためには、描画領域の子としてメニューを追加します。ユーザーが右側のマウスボタンで描画領域をクリックすると、メニューが表示されます。描画領域には、子として複数のポップアップメニューを持たせることができます。この場合、ダイナミックディスプレイでポップアップするメニューは、デザイン階層でどのメニューが選択されているかによって異なります。

ティアオフメニューを作成するためには、ティアオフリソースを可に設定します。Motif バージョンではバグがあるため、このリソースがハードコードされておらず、しかもアプリケーションリソースファイルの一部である場合にこのリソースを有効にするためには、*XmRepTypeInstallTearOffModalConverter()* への呼び出しは、メインプログラムまたはメニューでの作成の前プレリュードから行われる必要があります。

図 27-5 に、3 種類のメニューを使用したデザイン階層を示します。

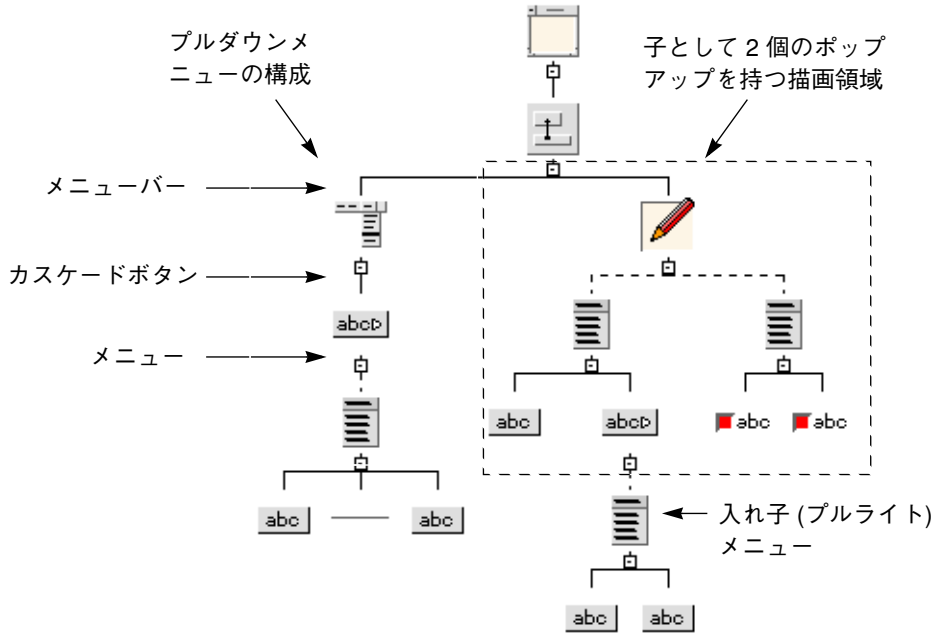


図 27-5 メニューを使用した階層の例

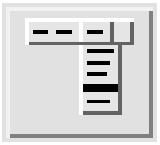
プルダウンおよびプルライトメニューとは異なり、ポップアップメニューは生成コードによって明示的にマネージ (管理) される必要があります。多くのアプリケーションにおいて、ポップアップメニューは状況に依存するため、Sun WorkShop Visual はこのマネージを自動的に行うことはありません。ユーザーは、描画領域の入力コールバックを使用して、メニューの位置を決めてマネージするか、あるいはトランスレー

ションメカニズムを使用するアクション関数を使用します。通常は、メニューは *XmMenuPosition()* を使用して位置が決定され、*XtManageChild()* を使用してマネージされます。

相互排他的なトグルボタンを持つメニューを作成するには、「ラジオボタン動作」リソースを「はい」に設定します。

Motif の場合、「メニュー」は厳密には表示が行われる描画領域またはカスケードボタンの子ではなく、兄弟です。しかし、描画領域またはカスケードボタンの動作は、親ウィジェットの動作と同じようにメニューに影響するため、 Sun WorkShop Visual ではメニューがこれらのウィジェットの子であるかのように階層を表示します。Sun WorkShop Visual は、カスケードボタンまたは描画領域へのメニューの接続に、実線ではなく点線を使用します。点線は、その接続が本当の Motif の親子関係ではないことを示します。

メニューバー (MenuBar)



表示:	エントリのフレーム	設定:	配置方向
	マージンの幅		**間隔設定方法**
	マージンの高さ		**揃え方**
	列数		最後を調整
	間隔		マージンの調整
	ヘルプ・ウィジェット		整列
	最後の選択項目		同種の子
			ポップアップ可
			常にラジオボタン 1 個選択
			ラジオボタン動作
			高さのサイズ変更
			幅のサイズ変更
			ティアオフ
キーボード:	アクセラレータ	コールバック:	マップ (map)
	メニューポスト		アンマップ (unmap)
	ニーモニック		進入 (entry)
	ニーモニック文字セット		

メニューバーウィジェットは、メニューをプルダウンすることができるカスケードボタンの集合を表示します。メニューバーは、特別に構成されたローカラムです。

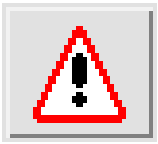
メインウィンドウ、ダイアログテンプレートおよび選択ボックスは、メニューバーを含むことができる標準配置を提供します。これらのウィジェットを使用せずにメニューバーを含ませる場合は、フォームを使用してその上部、左右にメニューバーをアタッチしなければなりません。上記ウィジェットの違いについては、メインウィンドウ、ダイアログテンプレート、選択ボックスおよびフォームの記述を参照してください。代表的な子の構成を持つメニューバーを含んでいるデザイン階層については、870 ページの図 27-5 を参照してください。

デフォルトのリソース設定により、『*Motif* スタイル・ガイド』に定義されている通りの標準メニューバーが提供されます。「間隔設定方法」リソースの設定は、「密」から「列」に変更することができます。「密」は、テキストを収めるために、すべてのボタンを最小サイズにします。「列」は、すべてのボタンを同じサイズにします。

「列」間隔設定方法を使用する場合は、「揃え方」リソースを設定してボタンをラベルの中央に設定することができます。他のリソースの変更は、お勧めできません。

メニューバーは、そのすべてのカスケードボタンを左側から狭い間隔で配置します。メニューバーが「ヘルプ」ボタンを持っている場合、『*Motif* スタイル・ガイド』ではそのボタンをメニューバーの右端に配置するように勧めています。カスケードボタンを「ヘルプ」ボタンに指定するためには、その変数名をメニューバーの「ヘルプウィジェット」リソースとして入力します。

メッセージボックス (MessageBox)



表示:	メッセージテキスト 了解ラベル 取消しラベル ヘルプラベル 記号用ピックスマップ	**設定:**	デフォルトボタン ダイアログの種類 揃え方 ボタン最小化
コールバック:	取消し (cancel) 了解 (ok)		

メッセージボックスウィジェットは、ユーザーにメッセージを表示します。

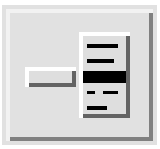
Sun WorkShop Visual のエラーメッセージはメッセージボックスの例です。メッセージボックスは、3 個のプッシュボタンガジェット、2 個のラベル、1 個のセパレータで構成されているコンポジットウィジェットです。これらの構成要素は、デザイン階層では表示されないブリテンボードに含まれています。継承されるブリテンボードのリソースを確認するためには、リソースパネルの「ブリテンボードのリソース」をクリックします。

メッセージボックスは、ブリテンボードが使用できる場所であれば、どの場所でも使用することができますが、通常はユーザーに警告を行うためにポップアップされるダイアログシェルに配置されます。

メッセージ領域にメッセージまたはピックスマップを表示する、あるいはボタンのラベルを変更するためには、構成要素ウィジェットのリソースパネルではなく、メッセージボックスのリソースパネルでリソースを変更します。

メニューバーおよび任意の数のボタンウィジェットをメッセージボックスの子として追加することができます。さらに、作業領域として、別の型の単一のウィジェットを追加できます。作業領域は、フォームのような、子を持つコンテナウィジェットにすることができます。

オプションメニュー (OptionMenu)



表示:	エントリのフレーム マージンの幅 マージンの高さ 列数 間隔 ヘルプ・ウィジェット 最後の選択項目	**設定:**	配置方向 間隔設定方法 揃え方 最後を調整 マージンの調整 整列 同種の子 ポップアップ可 常にラジオボタン 1 個選択 ラジオボタン動作 高さのサイズ変更 幅のサイズ変更 ティアオフ
キーボード:	アクセラレータ メニューポスト ニーモニック ニーモニック文字セット	**コールバック:**	マップ (map) アンマップ (unmap) 進入 (entry)

オプションメニューウィジェットは、複数のラジオボタンの場合に必要な画面スペースを使用せずに、多数の選択肢から 1 つの選択を表示するために使用されます。

Sun WorkShop Visual のリソースパネルにあるページセレクタは、オプションメニューの例です。オプションメニューは、特別に構成されたローカラムです。

オプションメニューは、ラベルとカスケードボタンを含んでいるコンポジットウィジェットです。 カスケードボタンには、メニューを子として追加し、選択ごとにプッシュボタンを追加します。セパレータを使用して、オプションのグループを分けることができます。

図 27-6 に階層の例を示します。カスケードするオプションメニューは持たせることができません。

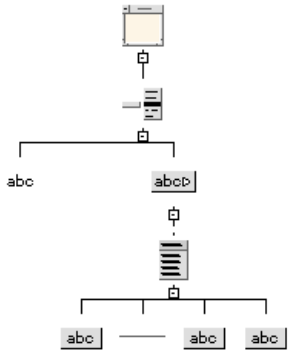
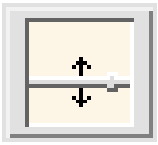


図 27-6 オプションメニューの階層例

ラベルのリソースパネルにある「ラベル」リソースを変更することにより、オプションメニューを識別するラベルを設定します。カスケードボタンのラベルは、オプションメニューの現在の設定を表示しているため、変更しないようにしてください。

区画ウィンドウ (PanedWindow)



マージン:	マージンの幅	マージンの高さ
	サッシの幅	サッシの高さ
	サッシのインデント	サッシのシャドウ
	間隔	
設定:	再配置	セパレータ

区画ウィンドウ・ウィジェットは、ウィジェットの集合を同じ幅の縦の列に配置するために使用されます。ウィンドウサッシのような可動セパレータにより、隣接する子と分離される区画内それぞれに、子ウィジェットが配置されます。ユーザーは、サッシを移動してそれぞれの子に割り当てる垂直方向のスペースを決定することができます。区画ウィンドウの高さは、子の高さを合計したものよりも少ないので、本来の機

能を損なうことなく垂直方向のスペースを節約することができます。区画ウィンドウの子は、他のウィジェットの配置を制御するコンテナウィジェットにすることができます。

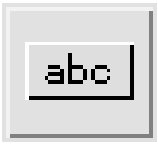
区画ウィンドウは、コンストレイントウィジェットです。コンストレイントパネルは、区画ウィンドウそのものではなく、区画ウィンドウの子に適用します。コンストレイントパネルの表示は、区画ウィンドウの子の 1 つが選択された場合にウィジェットから「コンストレイント」を選択することによって行います。このパネルの使用方法については、第 3 章「リソース」を参照してください。

コンストレイントパネルを使用すると、子に対しての「最小」および「最大」の高さのリソースを設定することができます。これらのリソースは、ウィジェットの区画およびサッシの位置の高さを制限します。

区画ウィンドウにある子は、強制的に最も幅の広い子と同じ幅に設定されます。

子の並べ替え、またはサイズ変更をするごとに、区画ウィンドウをリセットするようにしてください。

プッシュボタン (PushButton)



表示:	ラベル	マージン:	上
	フォント		下
	ピックスマップ		左
	応答不可能ピックスマップ		右
	カスケード・ピックスマップ		幅
	アームの色		高さ
	アームピックスマップ		間隔
	選択時の色		デフォルトのシャドウ
	選択時のピックスマップ		インジケータサイズ
	選択時の応答不能状態用のピックスマップ		
設定:	揃え方	コールバック:	活性化 (activate)
	種類		カスケード (cascading)
	サイズ変更		準備 (arm)
	プッシュボタン		解除 (disarm)
	シャドウの種類		露出 (expose)
	準備時に塗りつぶし		サイズ変更 (resize)
	選択時に塗りつぶし		値変更 (value changed)
	インジケータ		
	インジケータタイプ		
	マルチクリック		
	オン		
	オフの時表示		
トグル:	ウィジェット	キーボード:	アクセラレータ [1]
	ガジェット		アクセラレータテキスト [1]
			ニーモニック [1]
			ニーモニック文字セット [1]
			マッピング遅延

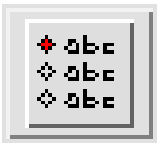
1. プッシュボタンがメニューの子である場合に応答可能です。

プッシュボタンウィジェットは、その上でマウスボタンをクリックして「押す」ことのできるボタンを表示します。ラベルと同様に、テキストまたはピックスマップのどちらかを表示することができます。

ボタンは、矢印ボタンや描画ボタンのように、用途によって種類が異なります。メニューをポップアップするボタンには、カスケードボタンを使用します。

「デフォルトとして表示」リソースの設定はお勧めできません。これは、親であるブリテンボードがこの設定を頻繁に変更するためです。デフォルトにするボタンの決定は、ブリテンボードが行います。

ラジオボックス (RadioBox)



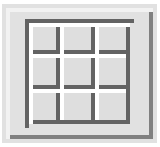
表示:	エントリのフレーム	設定:	**配置方向**
	マージンの幅		**間隔設定方法**
	マージンの高さ		揃え方
	列数		最後を調整
	間隔		マージンの調整
	ヘルプ・ウィジェット		整列
	最後の選択項目		同種の子
			ポップアップ可
			常にラジオボタン 1 個選択
			ラジオボタン動作
			高さのサイズ変更
			幅のサイズ変更
			ティアオフ
キーボード:	アクセラレータ	コールバック:	マップ (map)
	メニューポスト		アンマップ (unmap)
	ニーモニック		進入 (entry)
	ニーモニック文字セット		

ラジオボックスウィジェットは、相互排他的なラジオボタンとして動作する、トグルボタンのグループを収めるために使用されます。グループ内のトグルを選択すると、以前に選択されていたトグルが解除されます。「間隔設定方法」、「列数」、「配置方向」リソースを設定して、ローカラムに対して複数の列を作成することができます。

ラジオボックス内にあるトグルボタンはガジェットです。

ローカラムの「ラジオボタン動作」リソースを「はい」に設定すると、ローカラムをラジオボックスのように動作させることもできます。この構成のローカラムは、ラジオボックスよりも高い柔軟性を得ることができます。たとえば、トグルボタンを持つボックス内にラベル、セパレータ、その他のウィジェットを持たせたり、ガジェットの代わりにトグルボタンのウィジェット版を使用することができます。

ローカラム (RowColumn)



表示:	エントリのフレーム	**設定:**	**配置方向**
	マージンの幅		**間隔設定方法**
	マージンの高さ		揃え方
	列数		最後を調整
	間隔		マージンの調整
	ヘルプ・ウィジェット		整列
	最後の選択項目		同種の子
			ポップアップ可
			常にラジオボタン 1 個選択
			ラジオボタン動作
			高さのサイズ変更
			幅のサイズ変更
			ティアオフ
キーボード:	アクセラレータ	**コールバック:**	マップ (map)
	メニューポスト		アンマップ (unmap)
	ニーモニック		進入 (entry)
	ニーモニック文字セット		

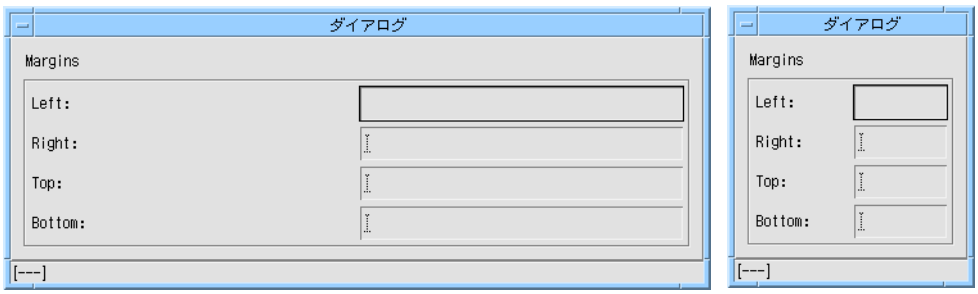
ローカラムウィジェットは、子ウィジェットを網目状に配列するために使用されます。多くの場合、ボタンあるいはトグルのグループなどの項目の配列に使用されます。たとえば、メニューは特別に構成されているローカラムウィジェットです。 ローカラムに基づいているその他のウィジェットには、オプションメニュー、メニューバー、メニューそしてラジオボックスがあります。

ローカラムには、任意の数の子を持たせることができます。ローカラム項目のデフォルト配列は、縦に 1 列です。複数の列を作成するためには、「間隔設定方法」リソースを「列」に設定し、その後「列数」リソースを設定します。

項目の順番は、「配置方向」リソースの設定が「垂直」の場合は、最初の列から下方向に、「方向」リソースの設定が「水平」の場合は、最初の行から横方向となります。「方向」リソース設定が「水平」の場合は、「列数」設定は垂直方向の行数を表わします。

ローカラムウィジェットは、配置が動的に変更されるようには設計されていないため、意図した変更を表示しない場合があります。ダイナミックディスプレイにおいて子の大きさが正しくない場合は、ローカラムをリセットしてください。

注 - 複数の列を使用する場合、ローカラムはすべての項目を強制的に同じ幅にします。しかし、これにより図 27-7 の「変更前」に表示されている (左の列が短いラベルであり、右の列が長いテキストフィールドである) ように、スペースを無駄にしてしまう場合があります。この場合、テキストフィールドをラベルの幅に合わせてサイズ変更することができます。ただし、新しい値がリソースパネルで受け入れられても、テキストフィールドのすべての値を変更するまでは、幅の変更は (図 27-7 の「変更後」のように) ダイナミックディスプレイには表示されません。



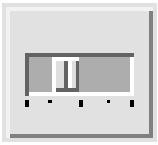
変更前　　変更後

図 27-7　ローカラムでのウィジェットのサイズ変更

幅が等しくない列を作成するには、ローカラムの代わりにフォームを使用します。また、ローカラムを入れ子にして行と列よりもさらに複雑な配置を作成することもできます。

ローカラム内にラジオボタンのグループを作成するには、トグルボタンを使用して、ローカラムの「ラジオボタン動作」リソースを「はい」に設定します。

スケール (Scale)



表示:
- 小数点
- 最小
- 最大
- 値
- タイトル
- スケールの幅
- スケールの高さ
- スケール倍率
- フォント

設定:
- **配置方向**
- **増加方向**
- **値を表示**

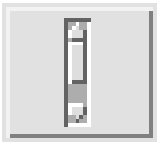
コールバック:
- ドラッグ (drag)
- **値変更** (value changed)

スケールウィジェットは、選択可能な値の範囲を提供し、現在の値を変更するために移動させるスライダを表示します。スライダは、クリックして連続的に移動、左マウスボタンで矩形の溝をクリックして単位ごとに移動、あるいは中央のマウスボタンで矩形の溝をクリックしてカーソル位置に移動させることができます。

スケールには、ほとんどのタイプの子を持たせることができます。通常は、ラベルが子となり、子はその長さに合わせて均等に配置されます。

スケールの配置方向を変更すると、動作が異常になる場合があります。問題が発生した場合は、スケールまたはその親をリセットしてみてください。

スクロールバー (ScrollBar)



表示:
スライダのサイズ
最小
最大
値
増加量
ページ増加量
初期の遅れ
反復時の遅れ
溝の色

設定:
配置方向
増加方向
矢印の表示

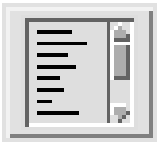
コールバック:
減少 (decrement)
ドラッグ (drag)
増加 (increment)
ページ減少 (page decrement)
ページ増加 (page increment)
下へ (to buttom)
上へ (to top)
値変更 (value changed)

スクロールバーウィジェットを使用すると、提供される表示領域よりも多くのスペースを必要とするデータを表示することができます。スクロールバーは、単独で使用されることはほとんどありません。スクロールウィンドウ、スクロールリスト、またはスクロールテキストなどのコンポジットウィジェットの一部として使用することが最も簡単です。

各スクロールバーは、両端に外側を向いた矢印を持ち、内側にスライダを持つ矩形として表わされます。表示領域は、スライダの移動あるいは矢印のクリックによってスクロールされます。スライダは、クリックして連続的に移動、左マウスボタンで矩形の溝をクリックして単位ごとに移動、あるいは中央のマウスボタンで矩形の溝をクリックしてカーソル位置に移動させることができます。リソースを編集して、1 回のスクロールアクションにスクロールする表示領域の量を制御することができます。

スクロールバーは、子を持つことができません。

スクロールリスト (ScrolledList)



マージン:	幅 高さ 間隔	設定:	スクロールバー表示方針 スクロールバーの位置 スクロールの方針 表示の方針 セパレータの表示 コマンドウィンドウ メッセージウィンドウ
コールバック:	隠れたものも走査 (traverse obscured)		

スクロールリストウィジェットは、スクロール可能な項目リストを表示するコンポジットウィジェットです。スクロールリストは、特別に構成されたスクロールウィンドウであり、リストウィジェットを含んでいます。子であるリストウィジェットのリソースは、通常の方法で設定することができます。

スクロールリストは、リストに項目を追加または削除するごとに自動的にサイズ変更を行なって、リスト内で最も幅の広い項目に幅を合わせます。Motif ツールキットのバージョンによっては、スクロールリストが正しい幅に調整されない場合があります。

不要なサイズ変更を避けるためには、何らかの方法でスクロールリストに制限を付ける必要があります。アタッチメントおよび位置アタッチメントを使用して、スクロールリストをフォームに制約することができます。しかし、フォームがまた他のウィジェットを含んでいる場合には、この制約によって異常な結果が生じる場合があります。これを避けるためには、図 27-8 に示すように、 スクロールリスト以外は何も含んでいないフォームでスクロールリストを使用してください。

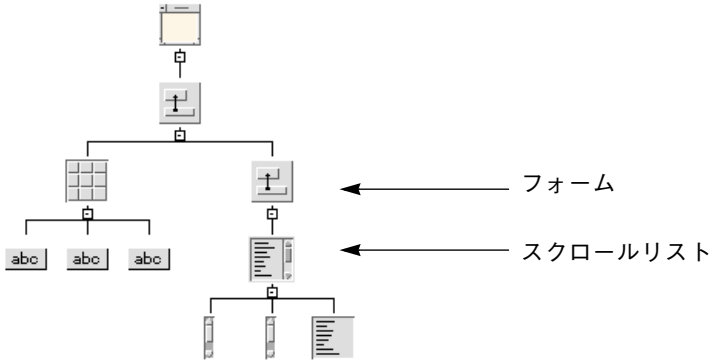
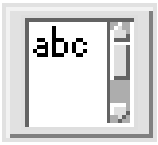


図 27-8 効果的なスクロールリストの配置

また、このフォームを他のウィジェットと一緒に別のフォームに入れることができます。スクロールリストを親フォームの 4 辺全部にアタッチし、 フォームの「サイズ変更方針」を「拡大のみ」あるいは「固定サイズ」に設定します。アタッチメントを使用することによってフォームの幅と高さを設定し、スクロールリストに適切なサイズを定義、あるいはフォームの初期サイズを固定することができます。その結果、アタッチメントを使用して中に含むスクロールリストのサイズを固定することができます。

フォーム用に設定されている制約は、スクロールリストの「可視項目数」リソース設定およびリスト内の個々の項目の幅より優先されます。

スクロールテキスト (ScrolledText)



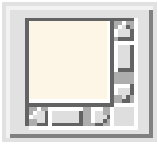
マージン: 幅
高さ
間隔

コールバック: 隠れたものも走査
(traverse obscured)

設定: スクロールバー表示方針
スクロールバーの位置
スクロール方針
表示の方針
セパレータの表示
コマンドの位置
メッセージウィンドウ

スクロールテキストウィジェットは、スクロール可能なテキスト領域を提供するコンポジットウィジェットです。スクロールテキストは、特別に構成されているスクロールウィンドウで、テキストウィジェットを含んでいます。子であるテキストウィジェットのリソースは、通常の方法で設定することができます。

スクロールウィンドウ (ScrolledWindow)



マージン: 幅
高さ
間隔

コールバック: 隠れたものも走査
(traverse obscured)

設定: スクロールバー表示方針
スクロールバーの位置
スクロールの方針
表示の方針
セパレータの表示
コマンドウィンドウ
メッセージウィンドウ

スクロールウィンドウ・ウィジェットは、使用可能な表示領域よりも多くのスペースを必要とするデータの表示に使用されます。 スクロールウィンドウウィジェットは、可視オブジェクト上に 2 個のスクロールバーと 1 個の表示領域を持つコンポジットウィジェットです。可視オブジェクトはスクロールウィンドウよりもサイズが大きい場合もあります。スクロールウィンドウには、どのような種類の子でも 1 個持たせることができます。

可視オブジェクトはどのような型のウィジェットでも構いませんが、通常は描画領域または他のウィジェットを含んでいるコンポジットウィジェットが使用されます。たとえば、フォームまたはローカラムウィジェットをスクロールウィンドウ内に配置することにより、ウィジェットの書式や表のスクロールに使用することができます。スクロール可能なリストまたはテキスト表示の場合は、スクロールリストまたはスクロールテキストウィジェットを使用します。

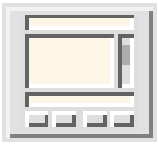
スクロールウィンドウに子を追加しない場合は、生成されたコードが実行時に警告メッセージを発します。

「スクロールの方針」リソースが「自動」に設定されている場合は、ツールキットがスクロールを処理して自動的にスクロールバーを作成します。

「スクロールの方針」リソースが「アプリケーションにより設定」に設定されている場合は、ユーザーがスクロールウィンドウの子に表示される情報を変更して、スクロールバーの動きに応答する必要があります。この場合、何らかのリソース、コールバックまたは名前が設定されていると、Sun WorkShop Visual はスクロールバーを作成するコードを生成します。

スクロールバーの動作を制御するリソース (「スクロールの方針」、「スクロールバー表示方針」) の作用は、ダイナミックディスプレイには反映されませんが、生成されたコードでは正常に動作します。

選択ボックス (SelectionBox)



表示:
一致する文字列なし
パターン
最大履歴項目
テキスト文字列
可視項目数
テキスト文字数
ディレクトリ・マスク
ディレクトリ

ラベル:
適用ラベル
了解ラベル
取消しラベル
ヘルプラベル
リスト用ラベル
選択用ラベル
フィルタのラベル
ディレクトリのラベル

設定:
ダイアログ型
ボタン最小化
既存項目と要一致
ファイル種類
作業領域の場所

コールバック:
適用 (apply)
取消し (cancel)
了解 (ok)
一致なし (no match)
コマンド変更 (command changed)
コマンド入力 (command entered)

選択ボックスウィジェットは、スクロール可能リストから 1 個あるいは複数の項目を選択するために使用されるコンポジットウィジェットです。選択ボックスの組み合せには、項目リストのためのスクロールリスト、2 個のラベル、1 個のセパレータ、4 個のプッシュボタンが含まれています。これらの構成要素はガジェットで、デザイン階層では表示されないブリテンボードに含まれています。継承されるブリテンボードのリソースを確認するためには、リソースパネルの「ブリテンボードのリソース」をクリックします。

選択ボックスはブリテンボードが使用できる場所であれば、どの場所でも使用することができますが、通常はユーザーからの選択を獲得するためにポップアップされるダイアログシェルに配置されます。

ボタンまたはラベルウィジェットのラベルを変更するためには、個々のウィジェットのリソースパネルではなく、選択ボックスのリソースパネルにあるリソースを変更します。

選択ボックスには複数の子を追加することができます。最初の子は作業領域になります。これには、複数の子を持つコンテナウィジェットを使用することもできます。作業領域は選択ボックスウィジェットの階層の最後に表示されていますが、ダイアログ

に作業領域が表示される場所は「作業領域の場所」リソースが制御します。追加された子にはメニューバーおよび任意の数のプッシュボタンウィジェットを持たせることができます。

提供されるプッシュボタンには、「了解」、「適用」、「取消し」、「ヘルプ」のラベルが付けられています。「適用」プッシュボタンは、プッシュボタンのコアリソースパネルにある「マネージ」トグルを設定すると表示されます。

選択プロンプト (SelectionPrompt)

表示:
一致する文字列なし
パターン
最大履歴項目
テキスト文字列
可視項目数
テキスト文字数
ディレクトリマスク
ディレクトリ

ラベル:
適用ラベル
了解ラベル
取消しラベル
ヘルプラベル
リスト用ラベル
選択ラベル
フィルタラベル
ディレクトリラベル

設定:
ダイアログ種類
ボタン最小化
既存項目と要一致
ファイルタイプ
作業領域の場所

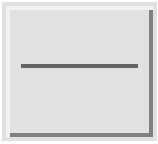
コールバック:
適用 (apply)
取消し (cancel)
了解 (ok)
一致なし (no match)
コマンド変更 (command changed)
コマンド入力 (command entered)

選択プロンプトウィジェットは、テキスト入力用のプロンプトを表示するために使用します。このウィジェットは、質問またはプロンプトに使用するラベル、答えを入力するためのテキストボックス、3 個のプッシュボタン (「了解」、「取消し」、「ヘルプ」) で構成されるコンポジットウィジェットです。「適用」プッシュボタンも用意されています。このボタンは、プッシュボタンのコアリソースパネルにある「マネージ」トグルを設定すると表示されます。これらの構成要素は、デザイン階層では表示されないブリテンボードに含まれています。継承されるブリテンボードのリソースを確認するためには、リソースパネルの「ブリテンボードのリソース」をクリックします。

選択プロンプトの内容は、選択ボックスとほとんど同じです。ただし、選択プロンプトにはリストが含まれていません。選択プロンプトは、 ブリテンボードが使用できる場所であれば、どこでも使用することができますが、通常はユーザーに入力を要求するためにポップアップされるダイアログシェルに配置されます。選択プロンプトまたはボタンのラベルを変更するためには、個々のウィジェットのリソースパネルではなく、選択プロンプトのリソースパネルにあるリソースを変更します。選択プロンプトには複数の行を設定することができます。

選択プロンプト内にあるプッシュボタンは、ガジェットです。選択プロンプトには複数の子を追加することができます。最初の子は作業領域になります。これには、コンテナウィジェットを利用することもできます。作業領域はプロンプトウィジェットの階層の最後に表示されていますが、ダイアログに作業領域が表示される場所は「作業領域の場所」リソースによって制御されます。追加された子にはメニューバーおよび任意の数のプッシュボタンウィジェットを持たせることができます。

セパレータ (Separator)



マージン:	種類	トグル:	ウィジェット
	配置方向		ガジェット

セパレータウィジェットは、オブジェクトを視覚的に分離するために使用される線です。セパレータには子を持たせることはできません。線の配置方向 (垂直または水平) を指定するためには、「配置方向」リソースを設定します。線の種類 (二重線または単破線など) を指定するためには、「種類」リソースを設定します。

セパレータを使用して、メニューまたはローカラムにある項目の分離、またはダイアログボックス内のウィジェットも分離することができます。フォームにあるウィジェットを分離するには、セパレータをその他のウィジェットとともにフォームの子にします。セパレータは、何らかの方法で制約を加えられるまでは、非常に小さなウィジェットとして表示されています。 セパレータの長さ、または幅をフォームにあわせて伸ばすには、フォームの両側あるいは他のウィジェットのそれぞれの端にセパレータをアタッチします。セパレータのサイズを明示的に設定することはお勧めできません。セパレータの「種類」を「線なし」に設定すると、目に見えないウィジェットとして使用することができます。

多くの場合、セパレータはメニュー内で項目をグループ化するために使用されます。この場合、図 27-9 に示されているように、隣接する兄弟間において表示されます。

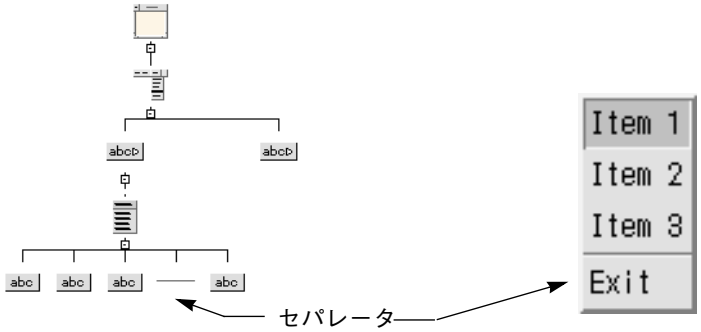


図 27-9 メニュー内のセパレータ

セパレータは、ローカラム内でも使用することができます。図 27-10 には階層例とその結果のダイナミックディスプレイを示します。セパレータをローカラム内で使用するには、セパレータの配置方向を明示的に「垂直」あるいは「水平」に設定します。ローカラム内にあるセパレータは、配列内のその他すべての要素のサイズにセルを伸ばします。しかし、この場合は望ましくない空白のスペースがセパレータ周辺に生じる可能性があります。割合を変更する場合は、列配置にフォームを使用します。

ローカラムの「間隔」リソースを 0 に設定すると、隣接するセパレータ間の隙間がなくなります。

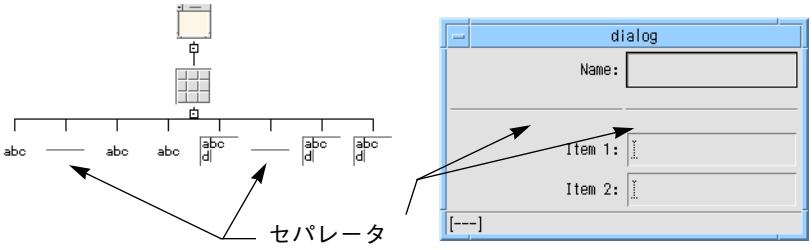
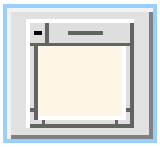
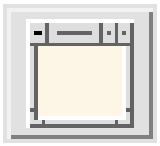
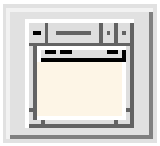


図 27-10 ローカラムでのセパレータの使用 (水平方向、4 行)

シェル (Shell) - ダイアログ、最上位、アプリケーション







表示:
- タイトル
- Mwm メニュー
- アイコン用マスク
- アイコン用ピックスマップ
- アイコン名 [1]
- ラベル用フォント
- ボタン用フォント
- テキスト用フォント
- 入力メソッド
- プリエディット種類

設定:
- 削除への応答
- キーボードフォーカス
- インプット
- 迂回
- サイズ変更可能
- オーバーライドリダイレクト: No
- アイコン状態 [1]
- 単位
- ウィンドウのグラビティ
- 初期状態
- 覆われた部分を保存
- 音による警告

寸法:
- ベースの幅
- ベースの高さ
- 幅の増分
- 高さの増分
- 最小の幅
- 最小の高さ
- 最大の幅
- 最大の高さ
- 最小縦横比 X
- 最小縦横比 Y
- 最大縦横比 X
- 最大縦横比 Y
- タイムアウト

コールバック:
- ポップダウン (pop down)
- ポップアップ (pop up)

トグル:
- アプリケーションシェル
- 最上位シェル
- ダイアログシェル

1. シェルがダイアログシェルに設定されている場合には、応答不能です。

シェルウィジェットは、デザインと Motif ウィンドウマネージャとのインタフェースとなります。Sun WorkShop Visual デザイン階層のルートウィジェットは、シェルでなければなりません。

Sun WorkShop Visual のパレットには、ダイアログシェル、最上位シェル、アプリケーションシェルの 3 種類のシェルウィジェットの型があります。シェルリソースパネルでいずれかのトグルをオンにして、任意のシェルに切り替えることができます。

アプリケーションシェルは、アプリケーションのメインウィンドウに使用されます。アプリケーションは、少なくとも 1 つ (通常は 1 つだけ) のアプリケーションシェルを持っていなければなりません。最上位シェルはその外観も動作もアプリケーションシェルに似ています。通常は、最初のウィンドウを除くアプリケーションのすべての一次的ウィンドウに使用されます。ダイアログシェルは、ポップアップダイアログなどの二次的ウィンドウに使用されます。アプリケーションまたは最上位シェルが閉じられたりアイコン化された場合は、関連するすべてのダイアログウィンドウも消滅します。

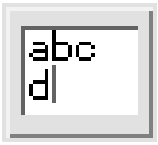
アプリケーションまたは最上位シェルは、ダイナミックディスプレイにはダイアログシェルとして現れますが、生成されたコードでは正確な種類のシェルが作成されます。アイコンピックスマップを確かめるには、「迂回」リソースを「いいえ」に設定し、その後シェルをリセットします。これにより、完全な装飾が作成されるため、ダイナミックディスプレイ・ウィンドウをアイコン化することができるようになります。

シェルは任意の種類の子を 1 つだけ持つことができます。しかし、シェルは、その子孫の物理的な配置を管理しないため、シェルの動作の多くは、子がブリテンボード、フォーム、あるいは類似したコンテナウィジェットであるという仮定に依存しています。シェルは、子を持ってはじめて可視状態になります。

シェルの幅と高さをコアリソースパネル上で設定しても、ウィンドウのサイズを制御することにはなりません。初期ウィンドウサイズを制御するためには、シェルの最小の幅と高さを設定する、あるいはシェルの子の幅と高さを設定します。

ウィンドウの初期位置を制御するには、シェルの子の「デフォルトの位置」リソースを「いいえ」に設定し、シェルではなく、子の X および Y リソースを設定します。

テキスト (Text)



表示:	値	設定:	**編集モード**
	カーソル位置		自動カーソル表示
	マージンの幅		編集可能
	マージンの高さ		保留の削除
	最大長		カーソル表示
	先頭文字位置		高さのサイズ変更
	セレクションしきい値		幅のサイズ変更
	点滅比率		ワードラップ
	桁数		ベルでの確認
	行数		横スクロール
	フォント		縦スクロール
			左側にスクロール
			上側にスクロール
コールバック:	活性化 (activate)		
	フォーカス (focus)		
	フォーカス消失中 (losing focus)		
	プライマリ取得 (gain primary)		
	プライマリ消失 (lose primary)		
	変更の確認 (modify verify)		
	動作の確認 (motion verify)		
	値変更 (value changed)		
トグル:	テキスト		
	テキストフィールド		

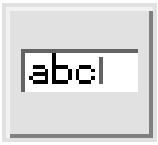
テキストウィジェットは、複数行のテキストを入力するための領域を提供します。入力の確認および検証を実行するために、広範囲のコールバックが用意されています。

複数行のテキストを使用するには、「編集モード」リソースを「複数行」に設定しなければなりません。「行数」リソース設定を 1 よりも大きな値に変更すると、テキストウィジェットの高さを変更して複数行のテキストを表示することができます。行数の変更の効果の有無は、 テキストウィジェットの親として使用されるウィジェットの型に依存します。

スクロール可能なテキスト編集領域を作成するには、テキストウィジェットを含むコンポジットウィジェットである、スクロールテキストを使用します。テキストウィジェットはスクロールウィンドウの子となることができますが、この構成は効果的ではありません。この構成を使用する場合は、「編集モード」リソースを「複数行」に変更し、行および列数をスクロールウィンドウの表示領域サイズよりも大きな値にします。

Motif ツールキットは、ウィジェット内のテキストを呼び出して変更するための関数を提供しています。詳細は、Motif のマニュアルを参照してください。

テキストフィールド (TextField)



表示:	値	設定:	編集モード
	カーソル位置		自動カーソル表示
	マージンの幅		編集可能
	マージンの高さ		保留の削除
	最大長		カーソル表示
	先頭文字位置		高さのサイズ変更
	セレクションしきい値		幅のサイズ変更
	点滅比率		ワードラップ
	桁数		ベルでの確認
	行数		横スクロール
	フォント		縦スクロール
			左側にスクロール
			上側にスクロール

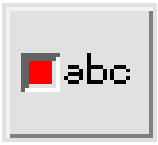
コールバック:	**活性化** (activate)
	フォーカス (focus)
	フォーカス消失中 (losing focus)
	プライマリ取得 (gain primary)
	プライマリ消失 (lose primary)
	変更の確認 (modify verify)
	動作の確認 (motion verify)
	値変更 (value changed)
トグル:	**テキスト**
	テキストフィールド

テキストフィールドウィジェットは、テキストウィジェットの変形で、単一行のテキストのみを入力するための領域を提供します。テキストフィールドウィジェットは、複数行機能を除くすべてのテキストの編集機能を持っています。

トグルを使用して、テキストフィールドからテキストに変更することができます。しかし、複数行機能を使用するためには、「編集モード」リソースを「複数行」に設定する必要があります。

Motif ツールキットは、ウィジェット内のテキストを呼び出して変更するための関数を提供しています。詳細は、Motif のマニュアルを参照してください。1012 ページの「X および Motif に関する資料」に参考文献についての記述があります。

トグルボタン (ToggleButton)



表示:	ラベル	設定:	整列
	フォント		種類
	ピックスマップ		サイズ変更
	応答不能状態用ピックスマップ		プッシュボタン
	カスケード・ピックスマップ		シャドウの種類
	準備 (arm) 時のカラー		準備時に塗りつぶし
	準備 (arm) 時のピックスマップ		選択時に塗りつぶし
	選択時の色		インジケータ・オン
	選択時のピックスマップ		インジケータタイプ
	選択時の応答不能状態用ピックスマップ		マルチクリック
			オン
			オフのときに表示
コールバック:	活性化 (activate)	マージン:	上
	カスケード (cascading)		下
	準備 (arm)		左
	解除 (disarm)		右
	露出 (expose)		幅
	サイズ変更 (resize)		高さ
	値変更 (value changed)		間隔
			デフォルトのシャドウ
			インジケータサイズ
キーボード:	アクセラレータ [1]	トグル:	ウィジェット
	アクセラレータテキスト [1]		ガジェット
	ニーモニック [1]		
	ニーモニック文字セット [1]		
	マッピング遅延		

1. トグルボタンがメニューの子である場合に応答可能です。

トグルボタンウィジェットは、「はい/いいえ」の選択を示すための簡単なオン/オフトグルを提供します。

トグルボタンは、ラジオボックス内、あるいは「ラジオボタン動作」リソースが「はい」に設定されているメニューまたはローカラム内に配置することにより、相互排他的なラジオボタンにすることができます。ラジオボタンは、図 27-11 に示すように、通常のトグルとは異なる形をしています。

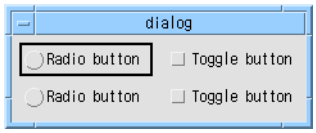


図 27-11 ラジオボタンと通常のトグルボタン

オンとオフの状態を表わすために、引っ込んだり、飛び出したりするプッシュボタンに似た外観のものを、トグルボタンを使用して実現することもできます。これを行うには、以下のようにします。

1. 「シャドウの厚さ」コアリソースを 2 に設定します。これにより、ボタンの周辺にフレームが描かれます。
2. 「インジケータ・オン」リソースを「いいえ」に設定します。これにより小さな正方形のインジケータが無効になります。
3. 左のマージンを 0 に設定します。これにより、インジケータが占有していたスペースが取り除かれます。

Motif ウィジェットの Microsoft Windows ウィジェットへの変換

Microsoft Windows モードの Sun WorkShop Visual で選択できる Motif ウィジェットと、それらがどの Microsoft Windows クラスに変換されるかを以下に示します。

アプリケーションシェル

CFrameWnd または Cdialog に変換されます。「コードオプション」ダイアログで「リソースとして生成」を選択します。

最上位シェル

CDialog に変換されます。

ダイアログシェル

CDialog に変換されます。

メインウィンドウとスクロールウィンドウ

シェルの子でない場合には、CWnd に対応します。シェルの子である場合は Microsoft Windows では無視されます。

フレーム、ラジオボックスおよびトグルボタン

CButton に変換されます。

ブリテンボード、フォーム、ローカラムおよびダイアログテンプレート

C++ クラスとして構成されている場合は、CWnd に変換されます。

描画領域

親がスクロールウィンドウ、メインウィンドウまたはシェルである場合を除き (Microsoft Windows では無視されるため)、CWnd に変換されます。その他の場合は、C++ クラスとして構成されます。

メニューバー、ポップアップメニューおよびカスケードボタン

CMenu に変換されます。C++ クラスとして構成することはできません。

オプションメニュー

CComboBox に変換されます。C++ クラスとして構成することはできません。

ファイル選択ボックス

CFileDialog クラスに変換されます。

区画ウィンドウ

CSplitterWnd に変換されます。

ラベル

CStatic に変換されます。

プッシュボタン

XmNlabelType が XmLABEL の場合は CButton に、XmNlabelType が XmPIXMAP である場合は CBitmapButton に変換されます。

セパレータ

Microsoft Windows 上のオブジェクトには変換されません。メニューの一部である場合には、メニューの属性として追加されます。メニュー以外に存在する場合には、無視されます。

スケールおよびスクロールバー

スクロールウィンドウの一部である場合を除いて、CScrollbar に変換されます。スクロールウィンドウの一部である場合には、包含しているクラスに適切なスタイルが追加され、ウィジェットとしては無視されます。スケールは正しく変換してください。「コードオプション」ダイアログで「リソースとして生成」を選択します。

テキストフィールドおよびテキスト

CEdit に変換されます。

リスト

CListBox に変換されます。

スクロールテキスト

適切なスクロールスタイルの CEdit に変換され、スクロールウィンドウ部分は無視されます。

スクロールリスト

適切なスクロールスタイルの CListBox に変換され、スクロールウィンドウ部分は無視されます。

Motif リソースの Microsoft Windows への変換

Microsoft Windows はリソースを使用しますが、その使用方法は X/Motif とは異なります。Microsoft Windows で使用されるリソースは、コンパイル時にアプリケーションに組み込まれます。また、Motif よりもその設定は厳密になっています。

Sun WorkShop Visual は、ビットマップ、アイコンおよびアクセラレータのみを Microsoft Windows リソースとして生成します。その他の Motif リソースは、表示されるウィンドウの属性に変換されるか、ソースコードに書き込まれます。

ウィンドウスタイル

Microsoft Windows オブジェクトの作成時には、複数のウィンドウスタイルを指定することができます。これらは、ビットフラグで or されます。以下に、トグルボタンの作成方法を示します。

```
Create ( "Classical", WS_CHILD | WS_VISIBLE | WS_TABSTOP |
BS_AUTORADIOBUTTON, rect, this, IDC_shell_classical);
```

基底 MFC クラスから継承されるメソッドへの 2 番目の引数がウィンドウスタイルです。Sun WorkShop Visual でリソースを設定すると、自動的に適切なウィンドウスタイルが選択されます。Microsoft Windows オブジェクトに変換できる各ウィジェットに対して使用可能なウィンドウスタイルのリストを、以下に示します。また、使用される状況および対応する Motif リソースも示しています。

シェル

すべてのシェルは、以下のウィンドウスタイルを持ちます。

- WS_POPUP
- WS_CAPTION

- WS_SYSMENU
- WS_MINIMIZE (*XmNinitialState* がアイコン表示に設定されている場合)
- WS_VSCROLL および WS_HSCROLL (子がメインウィンドウまたはスクロールウィンドウであり、適切なスクロールバーが指定されているか、リソースが設定されているか、コールバックまたはメソッドが設定されている場合)

アプリケーションシェル

シェルスタイルに加えて、以下のスタイルを持ちます。

- WS_THICKFRAME
- WS_MINIMIZEBOX
- WS_MAXIMIZEBOX

最上位シェル

アプリケーションシェルと、まったく同じスタイルを持ちます。

注 - アプリケーションシェルと最上位シェルは Microsoft Windows 上ではまったく同じであるという意味ではありません。これらは異なるクラスです。

ダイアログシェル

シェルスタイルに加え、以下のスタイルを持ちます。

- WS_THICKFRAME (ブリテンボードから派生した子の *XmNoResize* が True に設定されている場合を除く)

メインウィンドウおよびスクロールウィンドウ

XmNscrollingPolicy が XmAPPLICATION_DEFINED に設定されている場合にのみサポートされます。

- WS_CHILD
- WS_VISIBLE (ウィジェットがマネージされている場合)
- WS_DISABLED (*XmNsensitive* が False である場合)
- WS_TABSTOP (*XmNtraversalOn* が True である場合)

- WS_VSCROLL および WS_HSCROLL (子がメインウィンドウまたはスクロールウィンドウであり、適切なスクロールバーが指定されているか、リソースが設定されているか、コールバックまたはメソッドが設定されている場合)

フレーム

- WS_CHILD
- WS_VISIBLE (ウィジェットがマネージされている場合)
- WS_DISABLED (*XmNsensitive* が False である場合)
- WS_TABSTOP (*XmNtraversalOn* が True である場合)
- WS_GROUP
- BS_GROUPBOX

ブリテンボード、フォーム、ローカラム、描画領域、ダイアログテンプレート

- WS_CHILD
- WS_VISIBLE (ウィジェットがマネージされている場合)
- WS_DISABLED (*XmNsensitive* が False である場合)
- WS_TABSTOP (*XmNtraversalOn* が True である場合)

ラジオボックス

- WS_CHILD
- WS_VISIBLE (ウィジェットがマネージされている場合)
- WS_DISABLED (*XmNsensitive* が False である場合)
- WS_TABSTOP (*XmNtraversalOn* が True である場合)
- WS_GROUP
- BS_GROUPBOX (親がフレームではない場合)

メニューバー、ポップアップメニュー、カスケードボタン

他のすべてのオブジェクト同様、これらのウィジェットは Microsoft Windows 上ではウィンドウではなく、CMenu オブジェクトです。Cmenu は、Cwnd から派生していません。つまり、これらには関連するウィンドウスタイルはありません。その子 (カ

スケードボタンの場合は、メニューの子) は、それぞれの子の AppendMenu への呼び出しによって生成されます。子の種類により、AppendMenu には以下のフラグが渡されます。

- MF_POPUP
  メニューを持つカスケードボタンに対して
- MF_STRING
  XmNlabelPixmap 用の有効なピックスマップを持たないカスケードボタン (メニューを持たないもの) 、プッシュボタン、ラベルおよびトグルボタンに対して
- MF_GRAYED
  XmNsensitive が False である場合 (メニューバーの場合はカスケードボタンに対して)
- MF_MENUBREAK
  項目 (メニューバーの場合はカスケードボタン) が新しい列を開始する場合

以下のフラグは、ポップアップメニューまたはカスケードボタンからの AppendMenu への呼び出しにのみ適用します。

- MF_DISABLED (*XmNsensitive* が True の場合はラベルに対して)
- MF_SEPARATOR (セパレータに対して)
- MF_CHECKED (*XmNset* が True であるトグルボタンに対して)

オプションメニュー

- WS_CHILD
- CBS_DROPDOWNLIST
- WS_VISIBLE (ウィジェットがマネージされている場合)
- WS_DISABLED (*XmNsensitive* が False である場合)
- WS_TABSTOP (*XmNtraversalOn* が True である場合)
- WS_GROUP (*XmNnavigationType* が XmNONE ではない場合)
- ウィジェットが *XmNbuttonfontList* リソースに対して設定されているフォントオブジェクトリソースを持っている場合、あるいは格納しているブリテンボードまたはシェルに対して設定されているフォントオブジェクトを継承している場合は、SetFont メソッドが呼び出されます。

ファイル選択ボックス

これは、CFileDialog クラスに変換されます。Createメソッドは、CFileDialog に対しては明示的に呼び出されないため (InitDialog と DoModal が呼び出される) 、スタイルは一切存在しません。ただし、リソースは New メソッドに渡される引数に変換されます。

- OpenFileDialog (常に TRUE)
- ●□▲■✥❄❁✥▼→常に NULL)
- lpszFileName (指定されている場合は *XmNdirSpec* の値に設定、その他の場合は NULL)
- dwFlags (常にOFN_HIDEREADONLY|OFN_OVERWRITEPROMPT)
- ●□▲■✦❄●▼❄□→*XmNpattern* が指定されている場合は、この値は以下のように設定される。

```
"<XmNfilterLabelString>(<XmNpattern>)|XmNpattern|All
files(*.*)|*.*||"
```

XmNpattern が設定されていない場合は、この引数は NULL になる)

- pParentWnd (メインウィンドウ)

区画ウィンドウ

- WS_CHILD
- WS_VISIBLE (ウィジェットがマネージされている場合)
- WS_DISABLED (*XmNsensitive* が False である場合)
- WS_TABSTOP (*XmNtraversalOn* が True である場合)
- それぞれの子区画が作成される CreateView メソッドが呼び出される。これには、子区画クラスが動的作成をサポート可能であること (つまり、IMPLEMENT_DYNCREATE マクロを持っていること) が必要です。Sun WorkShop Visual は、適切なマクロ呼び出しを生成して、子区画クラスの動的作成をサポートします。

ラベル

- WS_CHILD
- WS_VISIBLE (ウィジェットがマネージされている場合)
- WS_DISABLED (*XmNsensitive* が False である場合)
- WS_TABSTOP (*XmNtraversalOn* が True である場合)

- ラベルの位置揃えに基づく配列 (SS_LEFT, SS_CENTER, SS_RIGHT) *XmNalignment* または親がローカラムである場合は親の *XmNentryAlignment* によって決定される
- SS_ICON (*XmNlabelType* が *XmPIXMAP* に設定されており、*XmNlabelPixmap* がピックスマップ・オブジェクトに設定されている場合)
- 設定されている場合には、Create メソッドへの表題引数は *XmNlabelString* の値となり、その他の場合にはウィジェット名となる
- ウィジェットが *XmNfontList* リソースに対して設定されているフォントオブジェクトリソースを持っている場合には、SetFont メソッドが呼び出される。ウィジェットが作成されている場合 (つまり、構成要素ではない場合) に、祖先のブリテンボードまたはシェルの *XmNlabelFontList* が設定されているのであれば、SetFont が呼び出される
- ウィジェットが *XmNlabelPixmap* に対して設定されている有効なピックスマップ・オブジェクトを持っている場合は、SetIcon メソッドが呼び出される

プッシュボタン

- WS_CHILD
- WS_VISIBLE (ウィジェットがマネージされている場合)
- WS_DISABLED (*XmNsensitive* が False である場合)
- WS_TABSTOP (*XmNtraversalOn* が True である場合)
- BS_OWNERDRAW (*XmNlabelType* が XmPIXMAP に設定されている場合)
- BS_DEFPUSHBUTTON (ウィジェットが、その祖先のブリテンボード (ダイアログシェルまたは最上位シェルの子孫) に対してのデフォルトボタンとして設定されている場合で、ボタンと Cdialog との間に CWnd オブジェクトが存在しない場合)
- 設定されている場合には、Create メソッドの表題引数は *XmNlabelString* の値となり、その他の場合はウィジェット名となる
- ウィジェットが *XmNfontList* リソースに対して設定されているフォントオブジェクトリソースを持っている場合には、SetFont メソッドが呼び出される。ウィジェットが作成されている場合 (たとえば、構成要素ではない場合) に、祖先のブリテンボードまたはシェルの *XmNlabelFontList* が設定されているのであれば、SetFont が呼び出される

トグルボタン

- WS_CHILD
- WS_VISIBLE (ウィジェットがマネージされている場合)

- WS_DISABLED (*XmNsensitive* が False である場合)
- WS_TABSTOP (*XmNtraversalOn* が True である場合)
- BS_AUTORADIOBUTTON (*XmNindicatorType* が XmONE_OF_MANY に設定されている場合。その他の場合は BS_AUTOCHECKBOX)
- *XmNset* が設定されている場合には、SetCheck メソッドが呼び出される
- 設定されている場合には、Create メソッドへの表題引数は *XmNlabelString* の値となり、その他の場合にはウィジェット名となる
- ウィジェットが *XmNfontList* リソースに対して設定されているフォントオブジェクトリソースを持っている場合には、SetFont メソッドが呼び出される

スケールおよびスクロールバー

- WS_CHILD
- WS_VISIBLE (ウィジェットがマネージされている場合)
- WS_DISABLED (*XmNsensitive* が False である場合)
- WS_TABSTOP (*XmNtraversalOn* が True である場合)
- SBS_HORIZ or SBS_VERT (*XmNorientation* の設定に依存)
- ウィジェットがスクロールウィンドウ構成要素である場合、あるいは *XmNmaximum* または *XmNminimum* が設定されている場合には、SetScrollRange が呼び出される
- XmNvalue が設定されている場合には、*SetScrollPos* が呼び出される

テキストフィールドおよびテキスト

- WS_CHILD
- WS_VISIBLE (ウィジェットがマネージされている場合)
- WS_DISABLED (*XmNsensitive* が False である場合)
- WS_TABSTOP (*XmNtraversalOn* が True である場合)
- WS_BORDER (*XmNshadowThickness* が 0 よりも大きい場合)
- WS_GROUP (*XmNnavigationType* が XmNONE ではない場合)
- ES_MULTILINE および ES_WANTRETURN (*XmNeditMode* が*XmMULTI_LINE_EDIT* に設定されている場合)
- ES_READONLY (*XmNeditable* が False に設定されている場合)
- WS_VSCROLL (親がスクロールテキストであり、*XmNscrollVertical* がデフォルトまたは True に設定されている場合)
- WS_HSCROLL (親がスクロールテキストであり、*XmNscrollHorizontal* がデフォルトまたは True に設定されている場合)

- *XmNvalue*リソースが設定されている場合は、SetWindowText メソッドが呼び出される
- ウィジェットが *XmNfontList* リソースに対して設定されているフォントオブジェクトのリソースを持っている場合、あるいは格納しているブリテンボードまたはシェルに対して設定されているフォントオブジェクトを継承している場合には、SetFont メソッドが呼び出される
- *XmNmaxLength* リソースが設定されている場合は、LimitText メソッドが呼び出される

リスト

- WS_CHILD
- WS_VISIBLE (ウィジェットがマネージされている場合)
- WS_GROUP (*XmNnavigationType* が XmNONE ではない場合)
- WS_TABSTOP (*XmNtraversalOn* が True の場合)
- WS_BORDER (*XmNshadowThickness* がゼロよりも大きい場合)
- WS_VSCROLL および WS_HSCROLL (親がスクロールリストである場合)
- LBS_EXTENDEDSEL (*XmNselectionPolicy* が *XmEXTENDED_SELECT* である場合)
- LBS_MULTIPLESEL (*XmNselectionPolicy* が *XmMULTIPLESELECT* である場合)
- LBS_DISABLENOSCROLL (親がスクロールリストであり、*XmNscrollbarDisplayPolicy* が *XmAS_NEEDED* ではない場合)
- ウィジェットが *XmNfontList* リソースに対して設定されているフォントオブジェクトのリソースを持っている場合、あるいは格納しているブリテンボードまたはシェルに対して設定されているフォントオブジェクトを継承している場合には、SetFont メソッドが呼び出される

第28章

障害発生時の対処

はじめに

本章は、Sun WorkShop Visual を初めて使用するユーザーのための一般的な質問および問題に対してのクイックリファレンスです。機能ごとに次の各節で構成されています。

- 「Sun WorkShop Visual インタフェース」
- 「定義とインスタンス」
- 「サポートされていないロケール」
- 「リソースパネル」
- 「配置エディタ」
- 「リンク」
- 「コード生成」
- 「Sun WorkShop Visual 再現、捕獲機能」

本章での小見出しは、問題の状況を説明します。問題の状況は、太字で表示してある小見出しの下に簡単に記述してあります。

Sun WorkShop Visual インタフェース

本節では、Sun WorkShop Visual が正しいリソースファイルを探し出すことができない場合に生じる問題を説明します。Sun WorkShop Visual は、X が Sun WorkShop Visual リソースファイルを参照するようにインストールされていなければなりません。以下に示す問題は、ダイナミックディスプレイではなく、Sun WorkShop Visual のインタフェースに関連しています。

ラベルが正しく表示されない

状況: Sun WorkShop Visual のボタン、プロンプト、およびメニューのラベルが正しく表示されない。

原因と対応:
これらのラベルは、正しいリソースが読み出された場合にのみ使用することができます。ラベルが使用できない場合、X は変数名を代用します。Sun WorkShop Visual が Sun WorkShop Visual リソースファイルを読み出すように、システムを再構成します。X でシステムを再構成するには、数種類の方法があります。システム管理者にお問い合わせください。

一部のラベルが正しくない

状況: ほとんどの Sun WorkShop Visual の表示は正確に行われているが、数個のボタンラベルまたは他のリソースが正しくない。

原因と対応:
構成が Sun WorkShop Visual リソースファイルの旧バージョン (このバージョンでは使用されていないもの) を読み取っている可能性があります。ソフトウェアのバージョンを確認し、Sun WorkShop Visual がそのバージョンで提供されているリソースファイルを読み出すようにしてください。

ヘルプ画面に何も表示されない

状況: ヘルプ画面が空白。

原因と対応:
X 環境が適切なリソースファイルを探し出していない、あるいはリソースファイルが適切な検索パスを呼び出していません。helpDir リソースが Sun WorkShop Visual へ

ルプデータベースに対しての検索パスを含んでいることを確認してください。環境変数 LANG が 適切なロケール (ja、あるいは C 等) に設定されていることも確認してください。

定義とインスタンス

本節では、定義とインスタンスの作成と使用に関連した事項について説明します。詳細は、310 ページの「定義」を参照してください。

Sun WorkShop Visual でファイルを開くと、「クラス・オブジェクトが認識不能」というエラーメッセージが表示される

開こうとしているデザインファイルに、Sun WorkShop Visual の現在のセッションでは認識できない定義のインスタンスが含まれています。定義を作成すると、ホームディレクトリに `.xddefinitionsrc` という特殊なファイルが作成されます。Sun WorkShop Visual は、このファイルを使用して、インスタンスが参照する定義を特定することができます。定義の内容を理解して、以下を実行してください。

1. 定義を含むデザインファイルを開いて、定義のルートを選択します。「ウィジェット」メニューの「定義」トグルをオフにして、「定義を編集」メニューから「定義」を選択します。すると、`.xddefinitionsrc` ファイルが作成されます。インスタンスを含むファイルを開く前に、定義デザインを保存する必要はありません。
2. Sun WorkShop Visual を終了します。定義を作成済みのユーザーのディレクトリから `.xddifinitionsrc` ファイルを自分のホームディレクトリにコピーします。最初に、既存のファイルが上書きされないように確認してください。
3. 他のユーザーと定義を共有できるように `definitionFileName`リソースを設定します。詳細は、311 ページの「定義ファイル」を参照してください。

インスタンスを含むコードのコンパイル

以下は、定義のインスタンスを含むデザインから Sun WorkShop Visual が生成したコードをコンパイルするときに発生する可能性のある問題です。

コンパイラが「記号が未定義」を示すエラーを返す

定義とインスタンスが同じデザイン内になく、別々の生成されたコードファイルにあることを確認してください。定義は別のデザインで保持します。定義コードをコンパイルして、インスタンスを含むコードで使用できるようにします。

コンパイラがインクルードファイルを見つけることができず、インスタンスが「未定義かまたは指定の型でない」ことを示すエラーを返す

定義に対するコードを別のディレクトリに生成した場合、その定義のヘッダーファイルを見つけることができるように、メークファイルを変更する必要があります。このヘッダーファイルの名前は、「定義を編集」ダイアログに表示される名前と同じでなければなりません。

リンカーが「未定義のシンボルが検出された」ことを示すエラーを返す

定義コードは、インスタンスを含むコードとコンパイルする必要があります。次の 2 通りの方法があります。

1. 定義を含むコードをライブラリにコンパイルして、ライブラリをそのインスタンスを含むアプリケーションにリンクします。リンカーがライブラリを見つけることができるようにメークファイルを変更する必要があります。
2. 定義のコードを、インスタンスのコードと同じディレクトリに生成します。最初に「新規」トグルと「テンプレート」トグルをオンにして、次に「テンプレート」トグルをオンにして、メークファイルを生成します。すると、定義とインスタンスコードを 1 つのアプリケーションにコンパイルするメークファイルが作成されます。

サポートされていないロケール

Sun WorkShop Visual では認識できない LANG 環境変数を使用してロケールを指定した場合、起動時に、指定したロケールはサポートされていないので「C」に強制的に指定されるというメッセージが表示されます。これによって生じる影響の詳細については、717 ページの「サポートされていないロケール」を参照してください。

リソースパネル

本節では、リソースパネルを使用する場合に起こる可能性がある問題について説明します。実行時にリソース値に関係する問題が発生した場合は、本章の 921 ページの「コード生成」を参照してください。

特定のウィジェットまたはウィジェットの組み合わせに対してリソースを設定する場合は、851 ページの第 27 章「ウィジェット・リファレンス」、および Motif マニュアルを参照してください。リソース設定の外観に関する問題の多くは、
関連のウィジェットまたはシェルをリセットすることによって解決できます。

Sun WorkShop Visual のリソースパネルに値を設定する場合、Sun WorkShop Visual は実際に、ダイナミックディスプレイにおいてウィジェットインスタンスのリソースをリセットします。結果が意図したものと異なる場合もあるので注意してください。

リソース設定が拒否される

状況:

- リソースパネルのフィールドに入力した値が受け入れられない
- リソースパネルの値が以前の値に戻る
- リソースパネルの値が他の値に変更される
- ダイナミックディスプレイにリソースパネルの値が反映されるが、値はユーザーが入力したものではない

原因: Motif はユーザーが指定した新しい値を設定することができません。最も一般的な原因は、選択されたウィジェットが他のウィジェット (通常はその親) によって制限されていることです。これは、頻繁に書き換えられるコアリソースパネルの幅や高さのリソースなど、サイズや位置に関係するリソースに多く該当します。

たとえば、ラジオボックスがトグルボタンのグループを持っている場合、個々のトグルボタンの幅がそのテキストとマージンリソースによって決定されます。ラジオボックスの幅は、その子の中で最も幅の広いものによって計算されます。そして、すべてのトグルボタンは最も幅の広い子と同じ幅に強制されます。コアリソースパネル上の幅の設定は、計算された値で書き換えられます。反対に、ラジオボックスがフォームの子である場合、配置エディタで設定されたアタッチメントは、ラジオボックスの規則を書き換えることができます。

対応: デザインに、設定を書き換える可能性のある制約があるかどうかを調べます。意図した効果を実現するためには、他のウィジェットによって指定される制約を使用します。

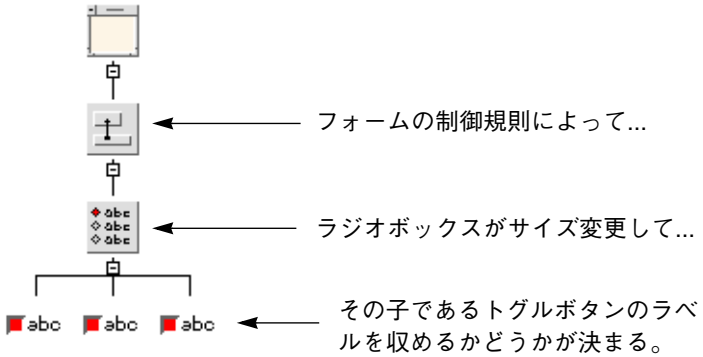


図 28-1 リソースの関係

リソース設定が有効にならない

状況: 新しい値はリソースパネル上に受け入れられたが、ダイナミックディスプレイには変更が即座に反映されない。

対応: ウィジェットをリセットしてください。ウィジェットのリセットを行なってもリソース値が有効にならない場合は、シェルをリセットしてください。シェルのリセットを行なっても意図通りに動作しない場合には、第 27 章「ウィジェット・リファレンス」を参照してください。単一の効果を得るために、2 個以上のリソースの設定が必要な場合もあります。たとえば、ラベルまたはボタンにピックスマップを表示するためには、「ピックスマップ」リソースを設定して使用するピックスマップを指定し、「種類」リソースを「ピックスマップ」に設定する必要があります。

以下に説明するように、ダイナミックディスプレイには決して反映されないリソースもいくつか存在します。

物理的条件のリソースが書き換えられる

状況: ウィジェットのコアリソースパネルの幅と高さのリソースが書き換えられる。

原因と対応:
ウィジェットの幅と高さのコアリソースは、ウィジェットクラスに固有のリソースによって、あるいはウィジェットの親の物理的条件の規則によって書き換えられる場合

があります。ウィジェットおよびそのサイズを制御している可能性がある親については、851 ページの第 27 章「ウィジェット・リファレンス」、あるいは Motif のマニュアルを参照してください。

原因と対応:
シェルの幅と高さのコアリソースは、シェルの子の対応しているリソースによって書き換えられる場合があります。ダイアログのサイズを制御する 1 つの方法は、シェルの子の幅と高さのリソースを設定することです。もう 1 つの方法は、シェルの最小幅と最小高さのリソースを設定することです。これは子によって書き換えられません。

原因と対応:
シェルの X と Y 位置リソースは、シェルの子がブリテンボード、あるいはフォーム、ダイアログテンプレート、またはメッセージボックスなどのブリテンボード派生物である場合は、シェルの子のリソースによって書き換えられる場合もあります。シェルの *X* と *Y* 位置リソースを使用してダイアログの位置を制御するためには、ブリテンボードの「デフォルトの位置」リソースを「いいえ」に設定します。

リセットの後にリソースが反映されない

状況: リソース設定はリソースパネル上で受け入れられたが、ウィジェットをリセットしてもリソース設定がダイナミックディスプレイに反映されない。

原因: 一部のリソースは、ウィジェットが最初に作成された際にのみ設定され、ウィジェットがダイナミックディスプレイに追加されてしまうと変更することはできません。これらのリソースには、「スクロール方針」などのスクロールバーに関連するさまざまなリソースが含まれます。Sun WorkShop Visual ではリソースパネル上の値の変更を認めていますし、また、値がダイナミックディスプレイに反映されない場合であっても、実行時には新しい設定が有効になります。個々のリソースについては、『*Motif* プログラマーズ・ガイド』を参照してください。

原因: ブリテンボードおよびブリテンボードから派生したウィジェットクラスの「ダイアログのスタイル」リソースは、「モードなし」、「システムモード付き」または「アプリケーションモード付き」に設定することができます。モード付きダイアログは、ユーザーからの答えを受け取るまでは他のすべてのダイアログを使用不能にします。したがって、「システムモード付き」または「アプリケーションモード付き」の設定が、ダイナミックディスプレイで有効である場合は、ダイナミックディスプレイを閉じるまでは、ユーザーは何もできなくなります。これらの設定のひとつを選択することは可能ですが、Sun WorkShop Visual は、ユーザーが階層を構築している間は設定を使用しません。しかし、生成されたコードは設定を正しく使用します。

対応: デザインの中にこの種類のリソースがある場合は、デザインが完了してコードを生成した後にのみ最終結果を確認することができます。以下に示す手順を使用してコードを生成し、プロトタイプを実行することにより、途中の段階で結果を確認することができます。

1. 基本コードモジュールを生成します。すべての型のリソースと *main()* プログラムを組み込みます。
2. スタブファイルを生成します (デザインにコールバックが含まれている場合のみ必要)。関数の括弧は空のままにします。
3. 生成したファイルをコンパイルしてリンクします。
4. 結果として生じたプログラムを実行します。

意図したフォントが Sun WorkShop Visual で表示されない

状況: デフォルトのフォント設定を持つウィジェットが、ダイナミックディスプレイにおいて意図したフォントを表示しない。

原因: ウィジェットが明示的なフォント設定を持たない場合、Motif はデザイン階層をさかのぼって検索を行い、ウィジェットに最も近いブリテンボード、ブリテンボード派生物あるいはシェルの祖先を探し出します。そして現在のウィジェットにその祖先のフォント設定を適用します。したがって、いくつかのボタンがすべてデフォルトのフォント設定を持っていて、それらが異なるブリテンボードの子である場合は、ボタンによって異なるフォントが表示されることもあります。

対応: 個々のウィジェットにではなく、ブリテンボードまたはシェルにフォントを設定すると、すべてのフォントを一度に簡単に設定することができます。統一性を得るためには、明示的なフォント設定により、すべての親に同じフォントを使用する必要があります。この場合、フォントオブジェクトを使用すると、簡単に実行することができます。

親でのフォント変更が子に影響しない

状況: ブリテンボード、ブリテンボード派生物、あるいはシェルでのフォントの変更が、子に影響しない。

原因と対応:
親ウィジェットの新しいフォント設定は、親ウィジェットをリセットするまで子には影響しません。親ウィジェットをリセットしてください。

原因と対応:
子が明示的に行われたフォント設定を独自に持っている場合は、親ウィジェットのフォント設定は子に影響しません。子のフォント設定をデフォルトにしてください。

ダイアログテンプレートのカスケードボタンが表示されない

状況: ダイアログテンプレートが、メニューバーにカスケードボタンを収めるようにサイズ変更しない。

原因: これは、Motif の一部のバージョンに見られるバグです。ダイアログテンプレートはボタンボックスおよび作業領域が収まるように、適切にサイズ変更を行いますが、メニューバーがボタンボックスと作業領域の幅よりも広い場合は、切り捨てられます。

対応: カスケードボタンをメニューバーに追加する前に、子であるボタンボックスおよび作業領域をダイアログテンプレートに追加します。たいていの場合、作業領域またはボタンボックスの幅はメニューバーを収めるために十分な幅を作成します。

対応: メニューバーの幅が広すぎて表示できない場合は、作業領域としてフォームを使用する、あるいはフォーム内に作業領域を配置します。その後、配置エディタを使用して作業領域の周辺に余分のスペースを追加します。

対応: コアリソースパネルの「幅」リソースを設定することにより、ダイアログテンプレートを強制的に幅広くすることができます。

配置エディタ

本節では、配置エディタ使用時の問題を説明します。配置エディタでの問題の多くは、フォームをリセットすることで解決できます。

ウィジェットの大きさが極端に変化する

状況: ウィジェットが極端に小さくなったり、大きくなったりする。

原因と対応:
Motif の一部のバージョンには、フォームがダイアログテンプレートの子である場合に、フォームにバグが現れるものがあります。この場合、フォームをリセットすると、フォーム内のコンテナウィジェットが極端に小さくなります。この問題が発生し

た場合は、フォームではなくダイアログテンプレートをリセットすると、適切な表示が行われます。ダイアログテンプレートのリセットを行うには、まず、配置エディタ画面から Sun WorkShop Visual メイン画面に移動し、構成領域からダイアログテンプレートを選択した後、「リセット」コマンドを指定します。配置エディタでの作業を再開するには、最初に構成領域でフォームをもう一度選択します。

原因と対応:
アタッチメントがウィジェットのサイズを変更する場合があります。たとえば、ウィジェットの端をフォームの対応する端にアタッチすると、そのウィジェットは強制的にフォームの幅または高さいっぱいに広げられます。次に、フォームをリセットしてください。フォームのリセットが効果を持たない場合は、ウィジェットからいくつかのアタッチメントを取り除き、再度リセットを行なってください。

アタッチメントを取り除く方法を以下に示します。詳細は、109 ページの第 4 章「配置エディタ」を参照してください。

- 「元に戻す」を使用して、最新のアタッチメントを取り除く
- ウィジェットを移動する
- 「アタッチ」モードの場合、ウィジェットのすぐに内側をクリックする、あるいはウィジェットの端のすぐ内側からウィジェットの中央に向かってドラッグする
- 古いアタッチメントと新しいアタッチメントを置き換える
- デザイン階層内でウィジェットを選択してコンストレイントパネルを起動し、その端のアタッチメントの「種類」を「なし」にリセットする

「循環依存」エラーメッセージ

状況: アタッチメントを作成した後に「循環依存」メッセージが表示される。

原因と対応:
2 個以上のウィジェットに影響する循環を引き起こすようなアタッチメントを追加する場合、Motif は循環を検出してエラーメッセージを返します。その場合は、「元に戻す」をクリックしてください。それでもエラーメッセージが表示される場合は、配置にアタッチメントループがあるかどうかを注意して調べて、アタッチメントの 1 つを取除きます。

「脱出 (Bailed Out)」エラーメッセージ

状況: アタッチメントを作成した後に「脱出」メッセージが表示される。

原因と対応:
この Motif メッセージは、循環していない、互いに矛盾するアタッチメントが配置に含まれていることを示します。通常は「自己」または「位置」アタッチメントに発生します。「元に戻す」を選択するかウィジェットを移動して矛盾するアタッチメントを取り除き、フォームをリセットしてください。

ウィジェットがフォームの境界と重なり合う

状況: フォームの端にあるウィジェットが、フォームの境界線に入り込む。

原因: その端がフォームの端と一致するウィジェットは、フォーム周辺に描かれた線と重なり、線が分断されてしまいます。これは、フォームがシェルの直接の子である場合にのみ発生します。以下の 3 通りの場合に、フォームは重なり合いの問題を引き起こします。

- オフセット 0 あるいは 1 ピクセルで、ウィジェットがフォームの端にアタッチされている
- デフォルトオフセットで、垂直あるいは水平間隔の値 0 でウィジェットがフォームの端にアタッチされている
- フォームの下あるいは右端と、その端の最も近くにあるウィジェットの間にアタッチメントがない

対応: オフセットを 2 ピクセル以上にして、ウィジェットをフォームにアタッチメントで接続します。明示的オフセットを使用することもできますが、フォームの垂直および水平間隔リソースを、オフセット値に設定してデフォルトオフセットを使用することもできます。配置の下および右側のウィジェットが、2 ピクセル以上のオフセットまたは間隔で接続されるようにします。

対応: フォームを別のフォームあるいはダイアログテンプレートなどの他のマネージャウィジェットの中に配置します。この方法が最も簡単で柔軟性のある対処法です。

リンク

本節は、ウィジェットメニューの「リンク編集」コマンドを使用する場合に生じる可能性のある問題を説明します。リンクのその他の情報については、次の節の「コード生成」を参照してください。

「追加」が使用できない

状況:「追加」オプションがグレー表示されている。

原因と対応:
リンク機能を使用するには、ターゲットウィジェットに明示的な変数名が必要です。ターゲットウィジェットがシェルである場合、直接の子もまた明示的な名前を持っている必要があります。Sun WorkShop Visual では、明示的な名前が見つからない場合に「追加」オプションをグレー表示します。適切なウィジェットに名前を付けてください。

リンクの動作が停止する

状況: 動作していたリンクが停止する。「リンク編集」ダイアログには、アイコンの代わりに空白スペースが表示される。

原因と対応:
ウィジェットの名前を変更した場合、Sun WorkShop Visual はそのウィジェットを参照するリンクを自動的に更新しないため、リンクの機能が停止します。使用しなくなったリンクを取り除き、新しいリンクで置き換えます。

別のボタンを選択した時にリンクが更新されない

状況: リンクパネルが、リソースパネルのように動作しない。あるボタンのリンク編集をしてから別のボタンを選択した際に、「リンク」パネルは以前に選択されたボタンからのリンクをそのまま表示している。

原因: Sun WorkShop Visual は、新しいウィジェットが選択されると、そのウィジェットを以前に選択されたボタンの新しいリンクになる可能性があるターゲットウィジェットであると解釈します。

対応: ウィジェットメニューをプルダウンして再度「リンク編集」を選択し、2 番目のボタンのリンクを表示して編集します。最初にリンクパネルを閉じる必要はありません。

コード生成

本節では、コード生成時に発生する可能性のある問題を説明します。これらの問題は、Sun WorkShop Visual が柔軟性のあるファイル配置を提供しているために生じるものもあります。たとえば、1 個のファイルにおいてのみリンク関数を生成する、あるいは必要とされるファイルにおいてインクルードを生成することなどを徹底してください。278 ページの「ファイルの整理」を参照してください。

「アプリケーションシェルがありません」という警告

状況: *main()* プログラム内で基本モジュールを生成しようとする場合に、Sun WorkShop Visual が「アプリケーションシェルがありません」という警告メッセージを表示します。

原因と対応:
デザインに必要なアプリケーションシェルが含まれていません。デザインのメインウィンドウのシェルに対するリソースパネルを起動し、「アプリケーションシェル」トグルをクリックして、「適用」をクリックしてください。

リンクが未定義

状況: リンク時に、リンク関数がリンクされない。

原因と対応:
基本モジュールを使用してリンクを生成する場合は、リンクに対しての実際のコード (リンク関数) を基本モジュールまたはスタブファイルのどちらかのコードファイルの1つに生成する必要があります。リンク関数は、確実に適切なファイルに生成してください。

大域ウィジェットが未定義

状況: スタブファイルのコンパイル時に、大域的ウィジェットが定義されない。

原因と対応:
大域ウィジェットおよびオブジェクトの宣言は、基本モジュールに生成されますが、スタブファイルには行われません。これらを宣言するヘッダーファイルを生成するためには、「外部宣言」オプションを使用して、結果として生じたヘッダーファイルをコールバックと一緒にインクルードします。この方法は、エラーが少なくなり、成功する可能性が高いため、独自の外部宣言の作成や、基本モジュールに生成された外部宣言のコピーを行う方法よりも適切であると言えます。外部宣言ファイルは、デザイン内での変更を反映するために必要である場合は、再生成することができます。

アプリケーションが X リソースファイルからのリソースを使用しない

状況: 生成されたアプリケーションが、生成された X リソースファイルからリソース値を使用しない。たとえば、ラベルやボタン上にラベル文字列の代わりに変数名が表示される、色やフォントが正しくない、など。状況は、X リソースファイルに生成されたリソースファイル、およびハードワイヤされたリソースファイルに依存します。

原因と対応:
コードを再生成した際に、X リソースファイルの再生成が行われませんでした。デザインでウィジェットを追加あるいは削除した場合は、デフォルトウィジェット名が変更され、生成された X リソースファイルでのウィジェット名と対応していない可能性があります。X リソースファイルを再生成してください。

原因と対応:
アプリケーションの実行時に、X は X リソースファイルを見つけることができません。X リソースファイルの名前変更が必要な場合があります。通常は、アプリケーションクラスと同じ名前で、接尾辞を付けません。詳細は、X のマニュアルを参照してください。

原因と対応:
異なるアプリケーションクラス名が生成されたコードファイルおよび X リソースファイルに使用されているため、X はファイル内でウィジェット名を識別することができません。両方に同じアプリケーションクラスが使用されるように、アプリケーションおよび X リソースファイルを再生成します。システムでアクセスする可能性のあるその他のリソースとの混乱を避けるため、固有のアプリケーション名を使用してください。

実行時にデフォルトリソースが変更される

状況: 実行時の色、フォント、あるいは他のリソースが、ダイナミックディスプレイでのリソースと異なる。

原因: ダイナミックディスプレイに表示されるリソースには、Sun WorkShop Visual から継承されるものがあります。リソースパネルで明示的に設定されていない場合は、プログラムを実行するプラットフォームによって、これらのリソースは実行時に他のリソースからの値を継承する可能性があります。

対応: 確実に正確な色やフォントを使用するためには、リソースパネル上でそれらの値を明示的に設定します。前景および背景の色は、デザインの各シェルに設定することができ、シェルのすべての子が設定を継承します。フォントはブリテンボード、ブリテンボードの派生物、あるいはシェルに設定することができ、そのすべての子に適用されます。

複数のウィジェットがウィジェット名を共有した場合に予期しない結果が生じる

リソース値は、共通の名前を持つウィジェット間で共有することができますが、共有できるのはそれらのリソース値が X リソースファイルからリソースデータベースに読み出される場合に限られています。以下の規則が適用されます。

- リソースの共有は、実行時にのみ可能
  ダイナミックディスプレイにおいては、ウィジェット間でリソース値は共有されません。
- ハードワイヤされたリソースは共有されない
- オブジェクトバインドは共有されない

状況: 実行時のリソース値がダイナミックディスプレイの値とは異なる。リソースが X リソースファイルに生成される場合、結果はハードワイヤされた場合とは異なる。

原因: これらは、ウィジェットがウィジェット名を共有する場合に予想される結果です。ダイナミックディスプレイとハードワイヤされたリソース設定は、共通ウィジェット名を無視します。しかし、X リソースファイルに生成されたリソースは、共通ウィジェット名を持つすべてのウィジェットに影響します。

状況: リソースが X リソースファイルに生成されているにも関わらず、共通ウィジェット名を持つウィジェットが色またはフォントを共有しない。

原因と対応:
色またはフォントは、色またはフォント設定の代わりにオブジェクトバインディングを使用しました。単純な色またはフォント設定を使用するか、値を共有するウィジェットの共通の親にリソースを設定してください。

状況: 明示的に設定したリソース値が、実行時に書き換えられる。

原因: ウィジェットがウィジェット名を共有し、リソースが X リソースファイルに生成される場合、複数の値が設定されていても 1 個だけの値が使用される。

対応: ウィジェットにすべてのリソース値を共有させない限り、ウィジェット名を共有させないでください。対応策としては、手作業でリソース値を強制的に単一のウィジェットに制限することができます。その場合は、リソースパネルのマスクトグルを使用します。

Sun WorkShop Visual 再現、捕獲機能

Sun WorkShop Visual 再現、捕獲機能に関して、頻繁に問い合わせのある内容について説明します。

なぜ Netscape などの特定のアプリケーションは記録および再現することができないのですか。

典型的な原因としては、次のいずれかの場合が考えられます。

- アプリケーションが、Xt ライブラリと動的にでなく、静的にリンクされている
- アプリケーションが、Xt 動作手続きを許可しない独自のマルチスレッド方式を採用している
- その製品が複数のアプリケーションシェルを使用している
  次の節を参照してください。

ご自身で作成したソフトウェアを記録および再現できない場合は、そのソフトウェアが *libXt.so* を使用するように再度リンクを行なってください。

なぜテキストウィジェット上でクリックした位置は、記録されないのですか。

すべての「位置に敏感な」Motif ウィジェットの記録および再現には、特別な Sun WorkShop Visual 再現 ルーチンが使用されます。このようなルーチンのソースコードは、`src/examples/replay/libcvtXm` ディレクトリにあります。

XmText および *XmTextField* に対する変換ルーチンは、デフォルトでは構築されません。テストという目的に関しては、ほとんどの場合、テキストフィールドを単なるデータ入力フィールドと見なし、内容を自由に置き換えるようにするだけで十分です。

たとえば、例 1 の方が例 2 よりも明確です。

例 1

```
doubleclick mytextwidget
type halloworld
```

例 2

```
doubleclick mytextwidget( position, 25)
type halloworld
```

また、特定の位置の文字をダブルクリックすることで、すべてのテキストが選択されるかどうかを確認することは、困難でもあります。

テストによっては、テキストフィールドの内容を変更する場合もあります。重要なものはテキストウィジェットの名前であって、その中に入力する値ではありません。

アプリケーション内で *XmText* ウィジェットの編集機能をテストしたい場合は、`cc` コマンドに `-DHANDLE_TEXT` オプションを追加して `libcvtXm` ディレクトリを再構築する必要があります。次に、共有オブジェクト *libcvtXm.so* を `lib/xds` にコピーして標準のライブラリを上書きしてください。

Sun WorkShop Visual 再現アプリケーションの著作権に関するメッセージは表示されますが、その後 Sun WorkShop Visual 再現アプリケーションが終了してしまいます。

Sun WorkShop Visual 再現アプリケーションは、再現しようとしているアプリケーションが Motif ではないと判断した場合には終了します。

-o オプションを使用すると、Motif 以外のアプリケーションを再現しようとした場合にも Sun WorkShop Visual 再現アプリケーションは終了せず、強制的にそのアプリケーションの機能を再現することができます。

注 - 再現しようとするアプリケーションが Motif であっても、アプリケーションシェルが複数存在する場合は、Sun WorkShop Visual 再現アプリケーションが終了する場合があります (次の節を参照してください)。

Sun WorkShop Visual 再現アプリケーションは起動しましたが、動作しないようです。

典型的な原因としては、次のいずれかの場合が考えられます。

- アプリケーションに、起動直後に瞬間的に表示されるアプリケーションシェルが存在する。または、 1 つ以上のアプリケーションシェルが起動直後に一時的に作成される

デフォルトでは、Sun WorkShop Visual 再現機能は 1 番目のアプリケーションシェルを記憶し、参照対象として使用します。通常は、最初に表示されたアプリケーションシェルが1番目のシェルとして認識されますが、次のようにオプションを指定して、4 番目のシェルを記憶するように変更することができます。

コマンド行で次のように入力します。

```
-use 4
```

または、環境変数 *XDSUSESHELL* を次にように設定します。

```
setenv XDSUSESHELL 4
```

注 - 「本当の」アプリケーションシェルが参照対象として使用されているかどうかを確認するには、そのアプリケーションを記録してください。

正しいアプリケーションシェルが使用されている場合は、記録されたログに次にように記述されています。

```
in ApplicationShell

    ......
```

次のように記述されている場合は、正しいアプリケーションシェルが参照対象として使用されていない可能性があります。

```
in myapp
.....
```

ただし、記録したアプリケーションに、マップされていないアプリケーションシェルがあり、かつ最上位シェルが複数存在する場合は、上述の例でも問題はありません。

現在の設定が正しくない場合は、-use オプションで現在よりも大きな値を指定します。本当のアプリケーションシェルが何番目のシェルか確認できない場合は、すべてのアプリケーションシェルを無視するように設定します。このように設定すると、いずれのアプリケーションシェルも記録、再現、捕獲されません。

- 記録しようとしたアプリケーションが、 Xt イベント処理を再実行または破壊した

具体的には、次のような例が挙げられます。

- ✍ オプションで Sun WorkShop Visual 再現アプリケーションを起動した。再現しようとしたアプリケーションは表示されたが、Sun WorkShop Visual 再現 のダイアログは表示されない
- -I オプションで Sun WorkShop Visual 再現アプリケーションを起動した。Sun WorkShop Visual 再現のダイアログの方が、再現するアプリケーションよりも先に表示された

このような場合にはデザインを捕獲することはできますが、再現することはできません。

- 捕獲/再現しようとするアプリケーションが Motif にリンクしていない。

この場合、デフォルトでは、Sun WorkShop Visual 再現アプリケーションは異常終了します。-O オプションを指定すると、この設定は変更することができます。ただし、Motif に依存する機能は、記録または再現することができません。

他社製のウィジェットを使用したアプリケーションを捕獲したいのですが、どのようにすればよいでしょうか。

捕獲機構では、捕獲したデザインのファイルを Sun WorkShop Visual の *.xd* 形式で作成します。つまり、標準の Sun WorkShop Visual が使用されていて、Motif 以外のウィジェットが追加されていないと仮定しています。デフォルトでは、Motif 以外のウィジェットは、捕獲ファイルの中では Motif の描画領域ウィジェットに置き換えられています。

他社製のウィジェットも捕獲できるような Sun WorkShop Visual を使用するには、リソースを次のように設定する必要があります。

```
*xdsCaptureUserWidgets:true
```

Sun WorkShop Visual 再現機能で日本語やその他の言語および入力メソッドを使用することはできますか。

使用できます。テキストフィールドに入力されたテキストの内容が記録され、テキストの入力がそのまま再現されます。この機能は、Motif のテキストウィジェットおよびテキストフィールドウィジェットに対してのみ有効であるように設定されています。設定ソフトウェアは、以下の場所にあります。

```
src/examples/replay/motif/motif4.c
```

また、このソフトウェアを登録するための機構は、以下のとおりです。

```
src/examples/replay/motif/register.c
```

入力メソッドによっては、このルーチンにアクセスできる場合があります。デフォルトの動作は「ウィジェットの中にある文字列にアクセスする」です。

Control + Space キーは、入力メソッドに対する作成要求にハードワイヤされています。また、代替手段の “compose” キーシムのリソースが、デフォルトで変換キーに設定されています。

たとえば、作成キーが F3 キーである入力メソッドを持つソフトウェアを記録する場合は、次のように設定してソフトウェアを実行してください。

```
-xrm *xdsImComposeKeySym:F3
```

または、このリソースを defaults ファイルに設定するか、xrdb を使用して設定してください。

自分が開発したアプリケーションは、一般ユーザーによって記録/再現が可能でしょうか。

一般ユーザーが Sun WorkShop Visual 再現機能を使用して、あなたの作成したアプリケーションを記録および再現できるようにするには、コードに次の行を挿入し、さらにアプリケーションを *libxdsclient.a* ライブラリとリンクする必要があります。

```
xdsAllowUserAccess()
```

付録 A

Sun WorkShop Visual 再現機能のコマンド構文

はじめに

本付録では、Sun WorkShop Visual 再現のスクリプトで使用されるキーワードについて説明します。

Sun WorkShop Visual 再現スクリプトのキーワードは、機能に応じて以下に示すように区分することができます。

- 記録および再生のコマンド
  - アクションのコンテキスト指定
  - ボタンアクション (簡単な制御)
  - プルダウンメニュー操作
  - オプションメニュー操作
  - キーボード操作
  - テキスト入力
  - ボタンアクション (位置に依存した制御)
- 追加コマンド
  - リソース評価
  - ウィジェット階層分析
  - 非アプリケーション操作

- 条件節
- 表示式
- ウィジェット状態式
- ユーザー定義コマンドの取り込み

アクションのコンテキスト指定

キーワード

in - スクリプトにおいて以降のアクションのコンテキストを指定する

ApplicationShell - アプリケーションの最上位シェル

形式

```
in shell_widget
        commands
in ApplicationShell
        commands
```

入力内容

shell_widget メインアプリケーションシェル以外のシェルウィジェットの名前

機能説明

Sun WorkShop Visual 再現のスクリプトは、ウィジェットに対するアクションで構成されています。これらのアクションは、該当するウィジェットを含むシェル (すなわちダイアログ) のコンテキスト内で行われる必要があります。シェルを認識できないと、その時点でスクリプトは失敗します。***in*** コマンドを入れ子にすることはできません。一度シェルから (別のシェルに移動するために) 離れると、そのコンテキスト内でアクションを行う前に ***in*** キーワードを使用してそのシェルに戻る必要があります。

使用例

```
in ApplicationShell
                  push this_button
                  push that_button
  push help_dialog_button
in help_dialog_popup
 push cancel_button
in ApplicationShell
 push another_button
```

ボタンアクション (簡単な制御)

キーワード

push **-** マウスボタンを押して離す

doubleclick **-** マウスボタンをダブルクリックする

形式

push *widget* [with [*modifier-*]*button[1-5]*]

doubleclick *widget*

入力内容

widget	ウィジェットの名前
modifier	キーボード修飾子
button[1-5]	マウスボタンの番号 (デフォルトは修飾子なしでマウスボタン 1)

機能説明

push は指定されたウィジェット上でのマウスボタンを使用したクリック (マウスボタンを押して離すという動作) をシミュレートします。***with*** キーワードを使用すると、特定のマウスボタンを指定することができます。このキーワードを使用しない場合は、ボタン 1 (マウスの左ボタン) が使用されます。キーボード修飾子 (Shift キーなど) を使用して、マウスボタンイベントの置換を拡張することができます。使用できる修飾子は Alt、Ctrl、Shift です。

doubleclick は、マウスの左ボタンを使用したダブルクリックをシミュレートします。このキーワードはどのウィジェットにも使用できますが、テキストウィジェットから選択する場合に特に便利です (938 ページの「テキスト入力」を参照)。

用法

ウィジェットの中には、マウスをクリックする位置は重要ではないものもあります。たとえば、ボタンウィジェット上のどの部分をクリックしても、そのボタンは活性化されます。しかし、位置が重要となるウィジェットもあります。たとえば、スケールウィジェット上では押す位置によって結果が異なります。

以下の表に、位置に依存するウィジェットと位置に依存しないウィジェットを示します。

位置非依存ウィジェット	位置依存ウィジェット
ボタン	スライダ
トグル	スケール
	リスト

位置非依存ウィジェット	位置依存ウィジェット
	描画領域
	テキストウィジェット [1]
	非 Motif ウィジェット

1.ユーザーとテキストウィジェットとの対話の記録および再生については、938 ページの「テキスト入力」で説明します。

位置依存リストの残りのウィジェットの記録および再生については、940 ページの「ボタンアクション (位置に依存した制御)」を参照してください。

使用例

```
in ApplicationShell
  push this_button
in ApplicationShell
  push that_button with shift-button2
in my_dialog_popup
                  if color_toggle->set:true
                      push color_toggle
                  endif
```

メニュー操作

キーワード

cascade **-** プルダウンメニューを表示する

pullright **-** プルダウンメニューからプルライトメニューを表示する

形式

```
cascade cascadebutton
                      select widget
cascade cascadebutton
```

```
pullright cascadebutton
```

入力内容

cascadebutton カスケードボタンの名前

widget カスケードボタンのプルダウンメニュー内にあるウィジェットの名前

機能説明

cascade は、メニュー操作の記述を簡略化した方法です。これ以外にも、対応するカスケードボタンを押すか、キーボードアクセラレータを使用して、メニューを表示することもできます。また、メニューオプションも同じように、アクセラレータまたはキーボードニーモニックを使用して選択することができます。

cascade で表示するプルダウンメニューを使用して、選択を行うことができます。選択項目は、ウィジェット (すなわちそのメニューのオプション) またはプルライトメニューを表示するカスケードボタンです。

使用例

```
in ApplicationShell
                  cascade file_m
                     select open_file
in ApplicationShell
                  cascade format_menu
                     pullright character_menu
```

注意事項

Sun WorkShop Visual 再現が Motif スタイルガイドに準拠してサポートするプルライトメニューは、1 階層だけです。ただし、スクリプトで push コマンドを使用すると、それ以降の階層のプルライトメニューも選択することができます。

オプションメニュー操作

キーワード

option **-** オプションメニューから選択する

形式

option *opmenu-widget*::*member_widget*

入力内容

opmenu-widget	オプションメニューウィジェットの名前
member-widget	オプションメニューのプルダウンメニュー内にあるウィジェットの名前

機能説明

option は、オプションメニューからオプションを選択します。

使用例

```
in ApplicationShell
                    cascade format_menu
                       pullright character_menu
                       option character_menu::bold
```

次の例では、オプションメニュー自体がユーザー入力に応答性がある場合に限りオプションの選択を行います。

```
if IsSensitive(myoptionmenu->OptionButton)
                   option myoptionmenu::thisoption
endif
```

オプションメニューの現在の設定 (すなわち、直前に選択された設定) を確認したい場合は、次の例に示すようにオプションメニューの *menuHistory* リソースを調べるだけでわかります。

```
if myoptionMenu->menuHistory: select_yes
    message he said yes
endif
```

注意事項

上記以外でオプションメニューのメンバーを選択するには、オプションボタンを押し、次に目的のメンバーウィジェットを押してください。ただし、***option*** 構文の方がユーザーアクションにより近い動作をするので、この構文を使用することをお勧めします。

キーボード操作

キーワード

alt - 現在の単語を選択する

ctrl - 現在の行を選択する

key - キーボードからキーシムを入力する

形式

alt *char*

ctrl *char*

key *keysym*

入力内容

char 1 文字

keysym 任意の X キーシム (キーシム一覧については ***X11/keysymdef.h*** を参照)

機能説明

キーボード入力は、フォーカスを持つウィジェットに送られます。Sun WorkShop Visual 再現では、キーボードから入力するための追加プログラミングは必要ありません。

ユーザーもテストスクリプトも、テキストを入力する際はウィンドウマネージャを使用する必要があります。明示的なフォーカスの置かれている位置 (ウィンドウをクリックしてフォーカスを得る必要があります) で、このウィンドウマネージャをテストスクリプト内にプログラミングする必要があります。

使用例

```
in ApplicationShell
                    alt f
                    type o
in open_file_popup
                    multiclick selection_field
                    type foo.xd
                    push ok_button
                    doubleclick my_text_field
                    type hallo world
                    key Return
```

注意事項

テキストフィールドでの push または doubleclick には、フォーカスを得るという二次作用があります。Sun WorkShop Visual 再現では、この場所でのみフォーカスを直接処理することができます。

テキストフィールドへのデータ入力は以前の内容を上書きすることがよくありますが、doubleclick または multiclick によって優先されます。

テキスト入力

キーワード

type - キーボードからテキストを入力する

key - キーボードからキーシムを入力する

doubleclick - 現在の単語を選択する

multiclick - 現在の行を選択する

形式

```
type text
key keysym
doubleclick textwidget
multiclick textwidget
```

入力内容

keysym XK_ 接頭辞のない任意の X キーシム (リストについては *X11/keysymdef.h* を参照)

textwidget テキストウィジェットの名前

text テキスト文字列

機能説明

アプリケーションにおける大部分のテキストウィジェットは、1 行データ入力 (ファイル選択ボックスの選択フィールドなど) に使用されます。Sun WorkShop Visual 再現を使用すると、フィールドのデフォルトの内容を既知の値に置換して結果を調べることができます。

type は、テキストをテキストウィジェットに入力します。***doubleclick*** は単語の選択を、***multiclick*** は行の選択をそれぞれプログラミングします。テストスクリプトでは、行中の単語数に関係なくテキストフィールドの内容を置換したい場合に、***multiclick*** が最もよく使用されます。

使用例

```
in form_attr_dialog_popup
                    doubleclick formHorizSpacingField
                    type 100
in coreDialog
                    multiclick title_t
                    type My Dialog Title
```

注意事項

Sun WorkShop Visual 再現が処理できるのは、1 行当たり 512 文字までです。この制限を超えたテキスト文字列を入力する場合は、テキストを分割し、それぞれの部分に ***type*** キーワードを使用します。

Motif の一部のバージョンでは XmTextField ウィジェットでトリプルクリックが正しく処理されないという問題が、Sun WorkShop Visual 再現では回避されています。このような場合は、スクリプトに ***multiclick*** が含まれていると、***doubleclick*** に変換されます。

ボタンアクション (位置に依存した制御)

キーワード

push **-** マウスボタンを押して離す

drag **-** 同一ウィジェット内で押す操作と離す操作を組み合わせる

形式

```
push widget(mame,qual)
drag widget(name1,qual1)-widget(name2,qual2)
```

入力内容

widget	ウィジェット名
name、*name1*、*name2*	アプリケーションまたはウィジェットに依存した記述

機能説明

一部のウィジェット (描画領域など) では、クリックする場所が重要になります。描画領域の場合は、描画領域内の位置が必要です。リストの場合は、選択した項目の指定が必要です。上記バージョンの ***push*** は、そのような位置に依存したウィジェットを対象としたコマンドです。

このようなウィジェットでは大部分の場合、クリック以外の操作が必要です。ある位置でボタンを押し、別の場所で離す必要があります。Sun WorkShop Visual では、フォームに対する配置エディタ上でウィジェット間にアタッチメントを設定する場合がその一例です。この設定はサーバーグラブに関係する場合があるので、最初の部分はボタンを押す位置を記述し、2 番目の部分はボタンを離す位置を記述するような単独の ***drag*** 操作として記述されます。

この機能は、アプリケーション内での描画領域など、ユーザー定義ウィジェットの単一インスタンスに使用することができます。また、ウィジェットクラス全体に (XmList、XmScale、XmScrollBar、他社の各種ウィジェットセットの場合のように) 使用することもできます。

使用例

最初の例は、Motif の描画領域ウィジェットが Sun WorkShop Visual テスト用に実装される様子を示しています。

```
in ApplicationShell

  push tree_da(mybutton,centre)
```

次の例は、Sun WorkShop Visual のフォームレイアウトエディタでインタフェースウィジェットと *button_box* ウィジェットの間でアタッチメントが作成される様子を示しています。

```
in form_layout

                  drag layout(frame1,right)-layout(button_box,left)
```

これらの結果は Sun WorkShop Visual で試してみることができます。

注意事項

独自に作成した位置に依存するウィジェットまたは他社のウィジェットを処理する方法については、528 ページの「Sun WorkShop Visual 再現ウィジェットセットの 拡張」を参照してください。

リソース評価

キーワード

printres **-** ウィジェットリソースの値を出力する

形式

printres *widget->resource*

入力内容

widget　ウィジェットの名前

resource　ウィジェットリソースの名前

機能説明

printres は、選択したウィジェット内の指定リソースの現在の値を出力します。既知のリソース値の出力が予想されるテストスクリプトでは、特にこのキーワードが便利です。リソースの名前は、「XmN」接頭辞を付けないで指定する必要があります (「labelString」など)。

スクリプトの条件式内には、リソース評価が含まれている場合がよくあります。

使用例

```
in my_shell
                if !my_option_menu->menuHistory:default_option
                    message FAIL: bad setting for my_option_menu
                    message Setting should be:
                    printres my_option_menu-
>menuHistory:default_option
                endif
```

ウィジェット階層分析

キーワード

tree **-** 現在のウィジェット階層の回帰的リストを表示する

dump **-** ウィジェットに割り当てられているリソースを表示する

snapshot **-** 現在のウィジェット階層の回帰的リストおよび各ウィジェット に割り当てられているリソースを表示する

形式

`tree` *`widget`*

`dump` *`widget`*

`snapshot` *`widget`*

入力内容

widget　　ウィジェットの名前

機能説明

tree、***dump***、***snapshot*** の各コマンドを使用すると、アプリケーションインタフェース内のウィジェットの構造、およびそれらのウィジェットに割り当てられているリソースの値を分析することができます。分析結果は、標準エラーに表示されます。

tree は、指定したウィジェットのウィジェット階層内にあるウィジェット名の回帰的リストを表示します。

dump は、指定したウィジェットのリソース設定を表示します。

snapshot は、指定したウィジェットのリソース設定、および指定したウィジェットのウィジェット階層内にある残りのウィジェットすべてのリソース設定を表示します。

使用例

次のコマンドは、button1 ウィジェットに割り当てられているリソースを表示します。

```
in ApplicationShell
                dump button1
```

上記コマンド例の出力の一部を以下に示します。

```
button1():
                Boolean ancestorSensitive:true
                HorizontalDimension width:58
                VerticalDimension height:22
                Pixel background:color('black')
                Pixel foreground:color('#72729F9FFFFF')
                HorizontalDimension highlightThickness:1
                Pixel highlightColor:color('black')
                XmString labelString:'Button A'
  Pixel armColor:color('red')
```

次のコマンドは、form1 ウィジェットのウィジェット階層を表示します。

```
in ApplicationShell
                tree form1
```

上記コマンド例の出力の一部を以下に示します。

```
                rowcol1():
                    buttonA():
                    button2():
                address_area():
                    label1():
                    text1():
```

注意事項

Sun WorkShop Visual 再現は、シェル内でウィジェット名を共有するウィジェットに対して一意の名前 (HorScrollBar#1、HorScrollBar#2、Apply#3、Apply#5、など) を割り当てます。再現機能用の名前が実際のウィジェット名と異なる場合は、角括弧に囲んで表示されます。

非アプリケーション操作

キーワード

delay - ユーザーアクションの再生を一時停止する

message - メッセージを出力する

sequence - スクリプトの一部にラベルを付ける

shell - シェルコマンドを実行する

setenv - シェル環境変数を設定する

breakpoint - スクリプト内にブレークポイントを指定する

exit - スクリプトを終了する
message - メッセージの印刷
sequence - スクリプトのラベル部分
shell - シェルコマンドの実行

形式

```
delay duration

message text

sequence text

shell command

setenv env-var env-value

breakpoint widget

exit status
```

入力内容

duration	秒単位の時間
text	テキスト文字列
widget	ウィジェットの名前
status	1 または 0
command	実行可能コマンド
env-var	環境変数
env-value	環境変数に割り当てられた値

機能説明

delay を使用すると、スクリプト内に一時停止を挿入することができます。これは、実行中に特定の場所でアプリケーションを視覚的に確認したいときに便利です。スクリプト内の次のアクションは、一時停止後に続行されます。

message は、標準エラーにメッセージを表示します。このコマンドを使用すると、スクリプトの異なる部分にラベルを付けて、予想される結果およびエラーをユーザーに通知します。メッセージテキストは、引用符で囲む必要はありません。

sequence を使用すると、スクリプトの異なる部分にラベルを付けることができます。そしてエラーが生じると、ラベル付けされている次のシーケンスまでスキップし、そこから続行することができます。

sequence を使用するには、*-skip-on-error* フラグを設定した状態で `visu_replay` を呼び出す必要があります。`visu_replay` はデフォルトでは *-user-on-error* フラグを設定した状態で実行し、エラーが生じるとテストを中止してそのままアプリケーション内に残ります。残りのエラーフラグ (*-exit-on-error*) は、エラーが生じるとアプリケーションを終了します。

shell は、スクリプトからシェルコマンドを実行します。シェルコマンドが終了すると、スクリプトは処理を続行します。この機能を使用すると、ユーザーアクションをただ再実行するよりもはるかに幅広い操作を行うスクリプトを作成することができます。

setenv は ***shell*** コマンドとともに使用して、環境変数を介してシェルに情報を引き渡します。***setenv*** は 2 つの引数をとります。最初の引数は変数の名前で、2 番目の引数はウィジェットのリソース値と以下の簡易関数のいずれかを結合できる式です。

- `WindowId(widget)`

- WindowFrame(widget)
- Parent(widget)
- Shell(widget)

breakpoint はデバッガとともに使用して、指定したウィジェットが活性化された際にスクリプト内にブレークポイントを設定します。こうすることによって、個々のウィジェットの内部を調べることができます。

breakpoint キーワードを含むスクリプトは、次のように呼び出す必要があります。

```
visu_replay -f script debugger app
```

上記の *script* はスクリプトの名前、*debugger* はデバッガの名前、*app* はスクリプトがテスト実行するアプリケーションの名前です。デバッガは、Sun WorkShop Visual 再現が実行します。***breakpoint*** キーワードで、ブレークポイントを直接設定しているのと同じようにアプリケーションが停止します。これで、アプリケーションが最適化されていても、ウィジェット内部を調べることができます。

exit は、指定した終了ステータスでスクリプトを終了します。

使用例

ウィジェットを押した後で 5 秒間遅延させるには、次のように記述します。

```
in ApplicationShell
    push mywidget
    delay 5
    push yourwidget
```

ウィンドウマネージャ装飾を付けないで、シェルの画面ダンプを撮影するには、次のように記述します。

```
in ApplicationShell
    setenv ID WindowId(ApplicationShell)
    shell xwd -id $ID -out /tmp/shell.xwd
```

ウィンドウマネージャ装飾を付けて、画面ダンプを撮影するには、次のように記述します。

```
in ApplicationShell
    setenv ID WindowFrame(ApplicationShell)
    shell xwd -id $ID -out /tmp/shell.xwd
```

カスケードボタンの名前しか分からないプルダウンメニューの画面ダンプを撮影するには、次のように記述します。

```
in ApplicationShell
    push cascade_button
    setenv ID WindowId(cascade_button->subMenuId)
    shell xwd -id $ID -out /tmp/shell.xwd
```

注 - ボタンを最初に押さないとメニューが表示されないため、xwd はそのメニューの画面ダンプを撮影することができません。

オプションメニューを使用して同じ動作をするには、次のように記述します。

```
in ApplicationShell
    push option_menu.OptionButton
    setenv ID WindowId(option_menu->subMenuId)
    shell xwd -id $ID -out /tmp/shell.xwd
```

カスケードボタンの親の背景色を表示するには、次のように記述します。

```
in ApplicationShell
    setenv ID Parent(cascade_button)->background
    shell echo The Color $ID
```

条件節

キーワード

```
if
else
elif
endif
```

形式

```
if expression
                actions
[elif expression
                actions]
[else
                actions]
endif
```

入力内容

expression	true か false かを評価する式
actions	複数のユーザーアクション

機能説明

if 文を使用すると、スクリプト内の制御フローをアプリケーションの実行中にアプリケーション内の条件に対応させることができます。各 ***if*** には、対応する ***endif*** が必ず必要です。必要に応じて、別の選択肢の ***elif*** (省略可能) およびデフォルトですべてに適用される ***else*** 条件を含めることができます。

使用例

```
in my_shell
                if !my_option_menu->menuHistory:default_option
                        message FAIL: bad setting for my_option_menu
                        message Setting should be:
                        printres my_option_menu-
>menuHistory:default_option
                else
                    message setting ok for my_option_menu
endif
```

表示式

キーワード

IsPseudoColor

IsDirectColor

IsTrueColor

IsStaticColor

IsStaticGrey

IsGreyScale

形式

if *expression*

actions

endif

入力内容

expression　前述のキーワードのいずれか 1 つ

機能説明

あるディスプレイ上で記録されたスクリプトは、異なる種類のディスプレイ上では動作しないことがあります。あるアプリケーションでは非常に多くの色を使うため、カラーマップに制限があるディスプレイ上でアプリケーションを実行する際に、色制限に関する警告メッセージが表示される場合があります。スクリプトはこのような状況に対応する必要があります。

使用例

```
if !IsPseudoColor
                message Non PseudoColor display
                in warning_popup
                    push warning.OK
```

```
endif
```

ウィジェット状態式

キーワード

```
IsVisible
IsManaged
IsRealized
IsHere
```

形式

```
if expression
                    actions
endif
```

入力内容

expression　前述のキーワードのいずれか 1 つ

機能説明

ダイアログの一部が選択表示されている場合は、***IsManaged*** 式および ***IsRealized*** 式を使用して管理したり認識している部分を調べることができます。

IsVisible は、画面上ではダイアログ全体を見ることができない小さい (VGA) ディスプレイ向けのコマンドです。Motif TAB ナビゲーショントラバーサル型のディスプレイは、画面外のコントロールを無視するので、このコマンドは重要です。

IsHere は、ウィジェットが現在のシェルに存在するかどうかだけを調べます。

使用例

```
in ApplicationShell
                    cascade file_menu
```

```
    select fm_menu.fm_exit
if IsVisible(save_dialog)
    in save_dialog
        push save.ok
else
    message Save Dialog cannot be seen
endif
```

ユーザー定義コマンドのインポート

キーワード

import **-** 追加コマンドのモジュールを読み込む

user **-** 読み込んだモジュールからコマンドを呼び出す

形式

`import` *module*

`user` *command text*

入力内容

module モジュールの名前

command コマンドの名前

text コマンドに渡す引数

機能説明

Sun WorkShop Visual 再現のコマンドセットの使用目的は、ユーザーアクションを再生したり、アプリケーションのウィジェット階層やリソース設定に関するアプリケーションの状態を確認することです。以下に示すように、必要に応じて独自のコマンドを追加しても、まったく問題はありません。

- 再生中のさまざまな位置で、画面ダンプを作成する場合。
- ウィジェット階層に対して別の種類の一貫性検査を行う場合。Doug Young の *widgedlint* ライブラリとのインタフェースをとることなどは、その一例です。
- 特定のデバッグ対象の問題に対してプローブまたはパッチを挿入する場合。これは、デバッガの全機能を利用することが不可能な分解された最適化バイナリに最も役立ちます。

import を使用すると、独自のコマンドのモジュールをスクリプトに読み込むことができます。一度モジュールを読み込むと、モジュール内のコマンドは user コマンドを使用して呼び出すことができます。呼び出すモジュールの数には制限はありません。

使用例

```
import mymodule
in ApplicationShell
                cascade file_menu
                    select fm_print
                    in print_dialog
                        user myscreendumper print_dialog
```

注意事項

実行する必要のあるアクションがウィジェット階層への拡張アクセス、またはプログラム内部の検査とは関係ない場合は、shell および setenv のインタフェースが優先経路です。後者の場合は、543 ページの「独自の Sun WorkShop Visual 再現機能コマンドの追加」を参照してください。

Sun WorkShop Visual 再現機能でのウィジェット命名規約

Sun WorkShop Visual 再現では、ウィジェット名はウィジェットを参照するときに使用します。ウィジェットベースのテストツールの主な処理の 1 つが、正しいウィジェットを識別することです。命名規約は、複雑すぎずかつ明確である必要があります。

Sun WorkShop Visual 再現が使用している規則は、以下のとおりです。

1. コントロールがウィジェット (ガジェットではない) で、かつ現在のダイアログでその名前を持つ唯一のウィジェットの場合は、次のようにウィジェット名を使用します。

```
in ApplicationShell
    push mywidget
```

2. コントロールがガジェットの場合は、次のように <親の名前>.<ガジェット> 名を使用します。

```
in ApplicationShell
    push myradiobox.mytogglebuttongadget
```

3. ウィジェット名がヌル (つまり “”) の場合は、次のように unnamed を使用します。

```
in ApplicationShell
    push myradiobox.unnamed
```

4. 現在のシェルに当該ウィジェット名 (またはガジェット名) のインスタンスが複数ある場合は、次のようにインスタンスを番号で参照します。

```
in ApplicationShell
    push mywidget#17
    push myradiobox.unnamed#3
    push myradiobox#2.unnamed#2
```

5. スクリプトを独自に作成する場合は、次のように `tree` コマンドを使用してウィジェット階層を調べます。

```
in ApplicationShell
    tree ApplicationShell
```

これで、ウィジェット階層の回帰的リストが出力されます。リストには、実際のウィジェット名が含まれます。また、Sun WorkShop Visual 再現で使用する名前と実際の名前が異なる場合は、Sun WorkShop Visual 再現で使用する名前も括弧で囲んで表示されます。

6. シェル名があいまいな場合は、次のようにインスタンスを使用します。

```
in myshell#2
    push button1
```

```
in myshell#3
    push button2
```

注 - インスタンス番号は、スクリプトを記録する際に自動的に計算されます。`Instance #3` は、シェルのウィジェット階層の最下位から始まり、左から右への検索でその名前が 3 個目に出現したことを意味しています。

付録 B

Motif XP リファレンス

はじめに

Motif XP ライブラリは、ほとんどの MFC クラスおよびメソッドを Motif ウィジェットと X または UNIX 呼び出しに変換することで、Motif と Microsoft Windows との間でコード共有を可能にします。本章では、ライブラリについて説明します。

Motif XP の使用

以下の情報を有効に活用するために、まず、取り扱う MFC クラスを確認してください。898 ページの「Motif ウィジェットの Microsoft Windows ウィジェットへの変換」で、この情報が記述されています。次に、本章で説明するクラスの記述を参照し、使用可能なメソッドを確認してください。

注 - `xd_` で始まるすべての変数およびメソッドは、Motif XP に固有のものです。これらは Motif 上では使用することができますが、Microsoft Windows 上で使用することはできません。

MFC およびメソッドについての詳細は、Microsoft Windows 上で使用している環境に対応した MFC のマニュアルを参照してください。

Motif XP の強化

Sun WorkShop Visual では、Motif XP に対してのソースコードが用意されています。ユーザーは、希望に応じてこのソースコードに内容を追加することができます。

Motif XP 用のソースコードは、`$VISUROOT/src/motifxp/lib` にあります (`$VISUROOT` は、Sun WorkShop Visual のインストールディレクトリです)。各クラスはそれぞれ独立したソースファイルを持っており、場所を示すコメントが付けられています。公開ヘッダー (インクルードファイル) は、 `$VISUROOT/src/motifxp/h` ディレクトリ内の `xdclass.h` にあります。

Motif XP ライブラリ

class CObject

CObject クラスは、MFC ライブラリの主要な基底クラスです。他のすべてのクラスは、このクラスから派生しています。

virtual ~CObject();

CObject オブジェクトを破壊します。

protected CObject();

CObject オブジェクトを構築します。

virtual Widget xd_rootwidget();
virtual void xd_rootwidget(Widget xd_rootwidget);

最初の *xd_rootwidget()* は、*CObject* オブジェクトが表わす階層のルートにあるウィジェットのウィジェットポインタを返します。2 番目の xd_rootwidget は、ルートウィジェットを設定します。

class CFrameWnd : public CWnd

*CFrameWnd*クラスは、Microsoft Windowsの単一文書インタフェースのオーバーラップした機能、あるいはポップアップフレームウィンドウの機能を提供します。
この機能は、アプリケーションシェル・ウィジェットをサポートするために、Sun WorkShop Visual が使用します。

protected virtual int xd_get_window_text (LPSTR lpszStringBuf, int nMaxCount) const;

シェルウィジェットに対しての *XmNtitle* の値を獲得します。ウィジェットがまだ作成されていない場合には 0 を返し、その他の場合にはテキストの長さを返します。

protected virtual int xd_get_window_text_length() const;

ウィジェットの *XmNtitle* リソースの長さを返します。ウィジェットがまだ作成されていない場合には 0 を返します。

protected virtual void xd_set_window_text(LPCSTR lpszString);

シェルウィジェットの *XmNtitle* および *XmNiconName* を *lpszString* に設定します。

protected virtual BOOL xd_show_window(int nCmdShow);

アプリケーションシェルに対して `ShowWindow` を実装するために使用されます。`SW_SHOWMINIMIZED`、`SW_HIDE` そして `SW_RESTORE` のみをサポートします。

class CCmdTarget : public CObject

CCmdTarget クラスは、XP ライブラリのメッセージマップ・アーキテクチャの基底クラスです。メッセージマップは、コマンドまたはメッセージをユーザー定義のメンバー関数へ送ります。この Motif バージョンには、機能が組み込まれていません。クラスは、Microsoft Windows コードとの互換性のためだけに組み込まれます。

class CWnd : public CCmdTarget

CWnd クラスは、XP ライブラリ内にあるすべてのウィンドウクラスの基本機能を提供します。以下の MFC のメソッドが実装されています。

CWnd();

virtual ~CWnd();

int GetWindowText(LPSTR lpszStringBuf, int nMaxCount) const;

ウィジェットのウィンドウテキストを獲得します。これは、仮想メンバー関数 *xd_get_window_text()* を呼び出すことにより実装されます。

int GetWindowTextLength() const;

ウィジェットのウィンドウテキストの長さを獲得します。これは、仮想メンバー関数 *xd_get_window_text_length()* を呼び出すことにより実装されます。

BOOL EnableWindow(BOOL bEnable=TRUE);

ウィンドウを使用可能、あるいは使用不可能にします。ウィジェットがまだ作成されていない場合には 0 を、ウィジェットが以前に使用可能とされた場合には 0 を、そしてウィジェットが以前に使用不可能にされた場合には 0 以外を返します。

void SetWindowText(LPCSTR lpszString);

ウィジェットのウィンドウテキストを設定します。これは、仮想メンバー関数 *xd_set_window_text()* を呼び出すことにより実装されます。

BOOL ShowWindow(int nCmdShow);

ウィンドウを表示、アイコン化 (アプリケーションシェルまたは最上位シェルのみ) 、あるいは非表示にします。ウィジェットがまだ作成されていない場合には 0 を、ウィジェットが以前に非表示にされた場合には 0 を、そしてウィジェットが以前に可視状態にされた場合には 0 以外を返します。これは、仮想メンバー関数 *xd_show_window()* を呼び出すことにより実装されます。

void xd_call_data (XmAnyCallbackStruct *call_data); XmAnyCallbackStruct *xd_call_data () { return _xd_call_data; }

最初の *xd_call_data()* は、そのクラスにコールバックの call_data を格納するために、Sun WorkShop Visual の生成したコードで使用されます。call_data は、2 番目の xd_call_data() を使用して、コールバック・メソッドに取り出すことが可能です。

protected virtual int xd_get_window_text (LPSTR lpszStringBuf, int nMaxCount) const;

GetWindowText() を実装するために、サブクラスによって使用されます。

protected virtual int xd_get_window_text_length() const;

GetWindowTextLength() を実装するために、サブクラスによって使用されます。

protected virtual void xd_set_window_text(LPCSTR lpszString);

SetWindowText() を実装するために、サブクラスによって使用されます。

protected virtual BOOL xd_show_window(int nCmdShow);

ShowWindow に対してのデフォルトの表示および非表示動作を実装します。ガジェットに対しては、そのガジェットをマネージおよびアンマネージし、ウィジェットに対しては適切に *mappedWhenManaged* を設定します。

class CDialog : public CWnd

CDialog クラスは、画面上にダイアログボックスを表示するために使用される基底クラスです。有効なクラスを作成するためには、通常は *Cdialog* から別のクラスを派生させます。

protected virtual int xd_get_window_text(LPSTR lpszStringBuf, int nMaxCount) const;

シェルウィジェットの *XmNtitle* の値を獲得します。ウィジェットがまだ作成されていない場合には 0 を返します。その他の場合にはテキストの長さを返して、そのテキストを *lpszStringBuf* に配置します。

protected virtual int xd_get_window_text_length() const;

ウィジェットの *XmNtitle* リソースの長さを返します。ウィジェットがまだ作成されていない場合には 0 を返します。

protected virtual void xd_set_window_text(LPCSTR lpszString);

シェルウィジェットの *XmNtitle* および *XmNiconName* を *lpszString* に設定します。

protected virtual BOOL xd_show_window(int nCmdShow);

最上位シェルまたはダイアログシェルに対して `ShowWindow` を実装します。`SW_SHOWMINIMIZED` (最上位シェルのみ) 、`SW_HIDE` および `SW_RESTORE` をサポートします。

class CScrollBar : public CWnd

CScrollBar クラスは、Microsoft Windows のスクロールバーコントロールの機能を提供します。

int GetScrollPos() const;

ウィジェットがまだ作成されていない場合には 0 を返し、その他の場合には *XmNvalue* を返します。

void GetScrollRange(LPINT lpMinPos, LPINT lpMaxPos) const;

ウィジェットが作成されている場合には、*lpMinPos* および *lpMaxPos* を *XmNminimum* および *XmNmaximum* にそれぞれ設定します。

int SetScrollPos(int nPos, BOOL bRedraw = TRUE);

ウィジェットが設定されている場合には、*XmNvalue* を *nPos* に設定し、以前の *XmNvalue* を返します。その他の場合には 0 を返します。

void SetScrollRange(int nMinPos, int nMaxPos, BOOL bRedraw = TRUE);

ウィジェットが作成されている場合には、*XmNminimum* および *XmNmaximum* を *nMinPos* および *nMaxPos* にそれぞれ設定します。

void ShowScrollBar(BOOL bShow = TRUE);

ウィジェットが作成されている場合には、*bShow* の値に従ってマネージあるいはアンマネージします。

class CFileDialog : public CDialog

CFileDialog クラスは、Microsoft Windows の共通ファイルダイアログボックスをカプセル化します。それによって、(その他のファイル選択ダイアログボックスのように)「開くファイル」および「別名保存」ダイアログボックスが、Microsoft Windows 準拠の形式で容易に実装されます。

```
CFileDialog (BOOL bOpenFileDialog,
    LPCSTR lpszDefExt = NULL,
    LPCSTR lpszFileName = NULL,
    DWORD dwFlags =
    OFN_HIDEREADONLY |OFN_OVERWRITEPROMPT,
    LPCSTR lpszFilter = NULL,
    CWnd* pParentWnd = NULL);
```

コンストラクタは、単純に *CFileDialogObject* を構築します。*lpszFileName* および *lpszFilter* 引数は、 *DoModal()* メソッドでファイル選択ボックスの *XmNdirSpec* と *XmNpattern* リソースを設定するために使用されます。*pParentWnd* リソースは、*CFrameWnd* オブジェクトを指します。

virtual ~CFileDialog();

非公開クラス変数を解放して、*CFileDialog* オブジェクトを破壊します。

virtual int DoModal();

XmNdirSpec および *XmNpattern* リソースをコンストラクタで指定されている通りに設定し、了解、取消し、ポップダウンコールバックが処理されるまで非公開イベントループを実行します。

CString GetPathName() const;

ファイル選択ボックスの *XmNdirSpec* リソースの値を返します。

protected virtual void OnCancel();

ユーザーが取消しボタンを押した場合、あるいはウィンドウメニューからダイアログをポップダウンした場合に呼び出されます。ファイル選択を完了する必要がある場合、サブクラスは *OnCancel()* を上書きする際にこのメソッドを呼び出します。

protected virtual void OnOK();

ユーザーが了解ボタンを押した場合に呼び出されます。ファイル選択を完了する必要がある場合、サブクラスは *OnOk()* を上書きする際にこのメソッドを呼び出します。

virtual BOOL OnInitDialog();

デフォルトで True を返します。これは、ウィジェットを作成する作成メソッドを呼び出すために、Sun WorkShop Visual 生成コード内で上書きされます。

class CSplitterWnd : public CWnd

区画ウィンドウを実装するために使用されます。

class CMenu : public CObject

クラス *CMenu* は、Microsoft Windows メニューコントロールを処理するためのクラスです。

CMenu();

CMenu オブジェクトを作成します。

~CMenu();

CMenu オブジェクトを破壊します。

UINT CheckMenuItem(UINT nIDCheckItem, UINT nCheck);

トグルボタンに対応するメニュー項目に対してのチェック状態を設定します。*nCheck* 引数は、項目の要求されている状態 (`MF_CHECKED` または `MF_UNCHECKED`) と *nIDCheckItem* の解釈 (`MF_BYCOMMAND` および `MF_BYPOSITION`) の両方を指定します。これらの 2 つの値は、ビットごとの OR、たとえば *menu->CheckMenuItem*

(*ID_toggle_b,* MF_BYCOMMAND | MF_CHECKED) を使用して指定される必要があります。関数は、メニュー項目が見つからない、あるいはトグルボタンではない場合 (MFC Windows では、セパレータを含むどのようなメニュー項目もチェックすることが可能です) には、-1 を返します。その他の場合には、以前の状態 (MF_CHECK または MF_UNCHECKED) が返されます。*nCheck* が MF_BYCOMMAND を含む場合には、すべてのサブメニューが検索されます。

UINT EnableMenuItem(UINT nIDEnableItem, UINT nEnable);

メニュー項目を使用可能または不可能にします。*nEnable* 引数は、項目の要求されている状態 (MF_ENABLED または MF_GRAYED) および *nIDEnableItem* の解釈 (*MF_BYCOMMAND* および *MF_BYPOSITION*) の両方を指定します。これらの 2 つの値は、ビットごとの OR、たとえば *menu->EnableMenuItem* (*ID_toggle_b,* MF_BYCOMMAND | MF_GRAYED) を使用して指定される必要があります。メニュー項目が見つからない場合、あるいは MF_BYCOMMAND が指定されていて、メニューバー、メニュー、セパレータまたはカスケードボタンである場合は、-1 を返します。その他の場合には、以前の状態 (MF_ENABLED または MF_GRAYED) が返されます。*nEnable* が MF_BYCOMMAND を含む場合には、すべてのサブメニューが検索されます。MF_DISABLED 状態 (応答不可能であるが、グレー表示されていない) はサポートされていないことに注意してください。

UINT GetMenuState(UINT nID, UINT nFlags)const;

メニュー項目の状態を獲得します。*nFlags* 引数は、*nID* の解釈 (MF_BYCOMMAND または MF_BYPOSITION) を指定します。関数は、メニュー項目が見つからない場合、あるいは セパレータである場合に -1 を返します。その他の場合は、状態のビットごとの OR (MF_CHECKED、MF_UNCHECKED、 MF_SEPARATOR、MF_ENABLED または MF_GRAYED) が返されます。*nFlags* が MF_BYCOMMAND である場合には、すべてのサブメニューも検索されます。MFC の場合は、ポップアップメニューに対する *GetMenuState* では高順位バイトに項目数をも返します。これは、Motif XP ではサポートされていません。

BOOL TrackPopupMenu (UINT nFlags,int x, int y,CWnd *pWnd,LPCRECT lpRect = 0);

この関数は、MFC Windows の TrackPopupMenu 関数と同様の動作をします。関数は、 *pWnd* によって指定されているウィンドウから call_data を取り出します (これは、コールバック関数によって保存されています) 。call_data にあるイベントが

ButtonPress イベントである場合には、ポップアップメニューは call_data のイベントを使用して (関数引数ではない) 配置され、メニューがマネージされて、TRUE が返されます。その他の場合には、FALSE が返されます。

void xd_register_menu(CMenu *menu);

IDをメニュー項目に割り当てるために、ツールキットによって使用されます。

void xd_register_menu_item(UINT nIDItem, Widget item);

IDをメニュー項目に割り当てるために、ツールキットによって使用されます。

protected Widget xd_get_menu_item_by_position(UINT nPos);

IDをメニュー項目に割り当てるために、ツールキットによって使用されます。

protected Widget xd_get_menu_item_by_id(UINT nIDItem);

IDをメニュー項目に割り当てるために、ツールキットによって使用されます。

class CComboBox : public CWnd

CComboBox クラスは、*ComboBox* に等価のインタフェースを提供するためにオプションメニューをラップするために使用されます。

int GetCurSel() const;

現在選択されている項目の索引 (最初の項目は 0) を返します。ウィジェットがまだ作成されていない場合には 0 を返します。

int GetLBText(int nIndex, LPSTR lpszText) const;

nIndex によって識別される項目のテキストのコピーを *lpszText* 内に獲得し、その長さを返します。ウィジェットがまだ作成されていない場合には 0 を返し、索引が範囲外である場合には LB_ERR を返します。

int GetLBTextLen(int nIndex) const;

nIndex によって識別される項目のテキストの長さを返します。ウィジェットがまだ作成されていない場合には 0 を返し、索引が範囲外である場合には LB_ERR を返します。

int SetCurSel(int nSelect);

現在の選択を、*nSelect* によって識別される項目に設定します。ウィジェットがまだ作成されていない場合には 0 を返し、索引が範囲外である場合には LB_ERR を返します。その他の場合には、選択された項目の索引を返します。MFC Windows とは異なり、選択を消去するために *nSelect* を -1 として渡す動作処理はサポートされていません。

protected virtual int xd_get_window_text(LPSTR lpszStringBuf, int nMaxCount) const;

lpszStringBuf で選択された項目のテキストを返します。ウィジェットがまだ作成されていない場合には 0 を返し、選択された項目がない場合には LB_ERR を返します。その他の場合には、テキストの長さを返します。

protected virtual int xd_get_window_text_length() const;

ウィジェットがまだ作成されていない場合には -1 を返し、作成されている場合には 0 を返します。これは MFC 動作に対応しています。

protected virtual void xd_set_window_text(LPCSTR);

これは、*CComboBox* に対する空の命令です。

class CStatic : public CWnd

CStatic クラスは、ラベルウィジェットと一緒に実装される単純なテキストフィールドである、Microsoft Windows の静的コントロールを実装します。

protected virtual void xd_set_window_text(LPCSTR lpszString);

`XmStringCreateLocalized()` で作成されたXmString に、ウィジェットの *XmNlabelString* リソースを *lpszString*を使用して設定します

protected virtual int xd_get_window_text(LPSTR lpszStringBuf, int nMaxCount) const;

ウィジェットの *XmNlabelString* の値を *lpszStringBuf* 内に獲得します。ウィジェットがまだ作成されていない場合には 0 が返され、その他の場合には文字列の長さが返されます。

protected virtual int xd_get_window_text_length() const;

ウィジェットに対しての *XmNlabelString* リソースの長さを返します。ウィジェットがまだ作成されていない場合には 0 が返されます。

class CButton : public CWnd

Cbutton クラスは、Microsoft Windows のボタンコントロールの機能を提供し、プッシュボタンまたはトグルボタンのいずれかとともに実装されます。

int GetCheck() const;

ボタンのチェック状態を獲得します。ウィジェットがまだ作成されていない、ウィジェットがトグルボタンではない、あるいはトグルボタンであっても設定されていない場合には 0 が返されます。ウィジェットがトグルボタンであり、設定されている場合は、 1 が返されます。MFC では、Motif XP でサポートされていない値 2 (未確定状態) を返すことができます。

void SetCheck(int nCheck);

nCheck に従って、トグルボタンの状態を設定します。これは、プッシュボタンに対しては空の命令です。

protected virtual void xd_set_window_text(LPCSTR lpszString);

`XmStringCreateLocalized()` で作成された XmString に、ウィジェットの *XmNlabelString* リソースを、 *lpszString* を使用して設定します。

protected virtual int xd_get_window_text(LPSTR lpszStringBuf, int nMaxCount) const;

ウィジェットの *XmNlabelString* の値を *lpszStringBuf* 内に獲得します。ウィジェットがまだ作成されていない場合には 0 が返され、その他の場合には文字列の長さが返されます。

protected virtual int xd_get_window_text_length() const;

ウィジェットの *XmNlabelString* リソースの長さを返します。ウィジェットがまだ作成されていない場合には 0 が返されます。

class CBitmapButton : public CButton

CBitmapButton クラスは、テキストの代わりにビットマップを使用してボタンを実装します。

class CListBox : public CWnd

CListBox クラスは、ユーザーが表示して選択することができる項目を表示するリストボックスの機能を提供します。

CListBox();

非公開データを初期化します。

virtual Widget xd_rootwidget();
virtual void xd_rootwidget(Widget xd_rootwidget);

クラスがリストとスクロールリストとを区別することができるように、*Cobject* 内のメソッドを書き換えます。

virtual Widget xd_listwidget();

オブジェクトのリストウィジェットを返します。通常のリストの場合には、これはルートウィジェットと同じですが、スクロールリストの場合には、ルートウィジェットとは異なります。

int DeleteString(UINT nIndex);

nIndex (最初の項目は 0) によって識別されるリスト項目を削除します。ウィジェットがまだ作成されていない場合には 0 を返し、索引が範囲外、あるいはリスト内の残りの項目数以外の場合には LB_ERR を返します。

int GetCount() const;

ウィジェットがまだ作成されていない場合には 0 を、その他の場合にはリスト内の項目数 (*XmNitemCount*) を返します。

int GetCurSel() const;

単一選択リスト内で現在選択されている項目の索引を獲得します (*XmNselectionPolicy* が *XmSINGLE_SELECT* または *XmBROWSE_SELECT* である場合)。ウィジェットがまだ作成されていない場合は 0 を、リストが複数選択リストである場合または項目が選択されていない場合には LB_ERR を返します。その他の場合は、選択されている項目の索引が返されます。リストが複数選択リストである場合には、Motif XP は常に LB_ERR を返しますが、MFC は任意の正の値を返します。

int GetSel(int nIndex) const;

nIndex によって示される項目の選択状態を返します。ウィジェットがまだ作成されていない、あるいは項目が選択されていない場合には 0 を返します。索引が範囲から外れている場合には LB_ERR を、また、項目が選択されている場合には正の値を返します。

int GetSelCount() const;

複数選択リスト内で選択されている項目の数を返します。ウィジェットがまだ作成されていない場合には 0 を返します。リストが単一選択リストである場合には LB_ERR を、また、その他の場合には選択されている項目の数を返します。

int GetSelItems(int nMaxItems, LPINT rgIndex) const;

複数選択リスト内で選択されている項目の索引を獲得し、それを配列 *rgIndex* にコピーします。ウィジェットがまだ作成されていない場合には 0 を、リストが単一選択リストである場合には LB_ERR を、また、その他の場合にはコピーされた索引の数を返します。

int GetText(int nIndex, LPSTR lpszBuffer) const;

nIndex によって識別される項目のテキストを、*lpszBuffer* に獲得します。ウィジェットがまだ作成されていない場合には 0 を、索引が範囲から外れている場合には LB_ERR を、また、その他の場合にはテキストの長さを返します。

int GetTextLen(int nIndex) const;

nIndex によって識別されるテキストの長さを獲得します。ウィジェットがまだ作成されていない場合には 0 を、索引が範囲から外れている場合には LB_ERR を、また、その他の場合にはテキストの長さを返します。

int GetTopIndex() const;

リスト上部で可視状態である項目の索引を返します。ウィジェットがまだ作成されていない場合には 0 を返します。

int InsertString(int nIndex, LPCSTR lpszItem);

nIndex により指定されたリスト内の位置に項目を挿入します。nIndex が -1 である場合には、項目はリストの末尾に追加されます。ウィジェットがまだ作成されていない場合には 0 を、索引が範囲から外れている場合には LB_ERR を、また、その他の場合には項目が挿入された位置が返されます。

void ResetContent();

リストからすべての項目を削除します。

int SelItemRange(BOOL bSelect, int nFirstItem, int nLastItem);

bSelect に従って、複数選択リスト内の項目の範囲を選択あるいは選択解除します。ウィジェットがまだ作成されていない場合には 0 を、リストが単一選択リストである場合には LB_ERR を、また、その他の場合には LB_ERR 以外の値を返します。

int SetCurSel(int nSelect);

単一選択リスト内で、*nSelect* により識別される項目を選択し、スクロールしてその項目を表示します。*nSelect* が -1 である場合には、選択は消去されます。ウィジェットがまだ作成されていない場合には 0 を、リストが複数選択リストである、あるいは索引が範囲から外れている場合には LB_ERR を、また、その他の場合には LB_ERR 以外の値を返します。

int SetSel(int nIndex, BOOL bSelect = TRUE);

bSelect に従って、複数選択リスト内の項目を選択または選択解除します。nIndex が -1 である場合には、すべての項目が選択または選択解除されます。ウィジェットがまだ作成されていない場合には 0 を、リストが単一選択リストである、あるいは索引が範囲から外れている場合には LB_ERR を、また、その他の場合には LB_ERR 以外の値を返します。

int SetTopIndex(int nIndex);

nIndex により識別される項目を可視状態にするために、リストをスクロールします。ウィジェットがまだ作成されていない場合には 0 を、索引が範囲から外れている場合には LB_ERR を、また、その他の場合には LB_ERR 以外の値を返します。

class CEdit : public CWnd

クラス *CEdit* は、ユーザーがテキストを入力することが可能な矩形ウィンドウである、Microsoft Windows 編集コントロールの機能を提供します。テキストまたはテキストフィールドウィジェットと一緒に実装されます。

CEdit();

非公開データを初期化します。

virtual Widget xd_rootwidget();
virtual void xd_rootwidget(Widget xd_rootwidget);

クラスがテキストとスクロールテキストとを区別することができるように、*Cobject* 内のメソッドを書き換えます。

virtual Widget xd_textwidget();

オブジェクトのテキストウィジェットを返します。通常のテキストに対しては、これはルートウィジェットと同じですが、スクロールテキストの場合には異なっています。

void Clear();

現在選択されているテキストを削除します (`XmTextRemove()`)。

void Copy();

現在選択されているテキストをクリップボードにコピーします (`XmTextCopy()`)。

void Cut();

現在選択されているテキストを削除して、それをクリップボードにコピーします (`XmTextCut()`)。

void GetSel(int & nStartChar, int & nEndChar) const;

選択されているテキストの先頭および末尾を獲得します。テキストが選択されていない場合には、先頭および末尾を 0 として返します。

void LimitText(int nChars = 0);

入力可能な文字の数を制限します。*nChars* が 0 である場合には、制限が最大値に設定されます。

void Paste();

クリップボードからのデータをテキストウィジェットに挿入します (`XmTextPaste()`)。

void ReplaceSel(LPCSTR lpszNewText);

現在の選択を、*lpszNewText* 内に供給されるテキストで置き換えます。選択が存在しない場合には、挿入カーソル位置にテキストが挿入されます。

BOOL SetReadOnly(BOOL bReadOnly = TRUE);

ウィジェットの *XmNeditable* リソースを、!*bReadOnly* になるように設定します。ウィジェットがまだ作成されていない場合には 0 を、また、その他の場合には 1 を返します。

void SetSel(int nStartChar, int nEndChar, BOOL bNoScroll = FALSE);

現在の選択を *nStartChar* および *nEndChar* によって指定されているテキストに設定します。また、*XmNautoShowCursorPosition* を !*bNoScroll* に設定します。

protected virtual void xd_set_window_text(LPCSTR lpszString);

ウィジェットの値を *lpszString* に設定します (`XmTextSetString()`)。

protected virtual int xd_get_window_text(LPSTR lpszStringBuf, int nMaxCount) const;

ウィジェットから *XmTextGetString()* でテキストを獲得し、*lpszStringBuf* に設定します。ウィジェットがまだ作成されていない場合には 0 が返され、その他の場合には文字列の長さが返されます。

protected virtual int xd_get_window_text_length() const;

ウィジェットのテキストの長さが返されます。ウィジェットがまだ作成されていない場合には 0 が返されます。

class CWinApp : public CCmdTarget

クラス *CWinApp* は、アプリケーションを初期化して実行するための Microsoft Windows アプリケーションオブジェクトを派生させる基底クラスです。

CWinApp(const char* pszAppName = NULL);

CWinApp オブジェクトを構築します。

const char* m_pszAppName;

アプリケーションの名前です。これは、 *CWinApp* コンストラクタに渡される引数から取り込まれます。

int m_nCmdShow;

デフォルトを SW_RESTOR にします。

CWnd *m_pMainWnd;

アプリケーションのメインウィンドウ (ApplicationShell) です。

Display *xd_display();
void xd_display(Display *display);

これらの 2 個の関数は、アプリケーションの表示接続の保存および取り出しを行います。

char **xd_argv() const;
void xd_argv(char **argv);

これらの 2 個の関数は、 main() に渡される argv 引数の保存および取り出しを行います。

int xd_argc() const;
void xd_argc(int argc);

これらの 2 個の関数は、 main() に渡される argc 引数の保存および取り出しを行います。

char *xd_app_class() const;
void xd_app_class(char *app_class);

これらの 2 個の関数は、XtOpenDisplay() で使用されるアプリケーションクラス名の保存および取り出しを行います。

CWinApp* AfxGetApp()

CWinApp オブジェクトのインスタンスを 1 つだけ返します。

コンパイラでのリンクエラー

C++ コンパイラによっては、リンクに失敗すると、以下のようなエラーが発生します。

```
Undefined symbol
CWnd::xd_get_window_text_length(void) const
CFrameWnd::xd_get_window_text_length(void) const
CButton::xd_get_window_text_length(void) const
CButton::__vtbl
CMenu::xd_register_menu_item(unsigned int,
_WidgetRec*)
CDialog::xd_show_window(int)
CDialog::xd_get_window_text_length(void) const
CDialog::xd_set_window_text(const char*)
CEdit::__vtbl
```

問題

コンパイラが、コピー・コンストラクタの実装を要求しています。

解決策

エラーが生じた場合には、以下の操作を行います。

1. ヘッダーファイル `xdclass.h` を `$VISUROOT/src/motifxp/h` 内で探します。`$VISUROOT` は、Sun WorkShop Visual のルートディレクトリです。
2. 次に示す行を検索します。

   ```
   private:
   ```

```
                // Certain C++ compilers (e.g. gcc 2.5) require
there to be an
                // implementation of the copy constructor. If your
application
                // fails to link try using the second version of the
constructor
                CObject(const CObject& objectSrc);              //
no default copy
                //CObject(const CObject& objectSrc) { abort();} //
no default
                copy
```

3. 指示に従って Cobject で始まる最初の行をコメントし、2 番目の行のコメントマーカーを削除します。

```
                //CObject(const CObject& objectSrc);
                // no default copy
                CObject(const CObject& objectSrc) { abort();}
                // no default copy
```

4. 次のインクルード行を、abort への呼び出しを含む行の上のどこかに追加します。

```
                    #include <stdlib.h>
```

5. 生成されたコードを再コンパイルします。

   今度は問題なくリンクが行われます。

付録C

getter と setter

はじめに

値または状態を保持できるウィジェットには getter と setter のルーチンが用意されています。この付録では、ウィジェットごとに使用できる getter と setter を一覧します。これらのルーチンはツールキットに依存しません。getter と setter を使用するには、最初にウィジェットを含んでいるグループを定義し、次にスマートコードのコールバックを設定する必要があります。この処理の詳細については、以下の章を参照してください。

1. 549 ページの第 15 章「グループ」
2. 559 ページの第 16 章「取得と設定用のスマートコード」
3. 577 ページの第 17 章「thin クライアント用スマートコード」
4. 615 ページの第 18 章「インターネット用スマートコード」

この付録で取り上げるウィジェットは次のとおりです。

- ラベルとボタン
- トグル
- テキストフィールド、テキスト、スクロールテキスト
- スケール
- リストとスクロールリスト
- オプションメニュー
- ラジオボックス

この情報の使い方

getter と setter でアクセスできる X リソースの名前が、ウィジェットごとに表形式で示されています。表中のリソースのうち、1 つはデフォルトです。これは、サーバーからアクセスされるリソースです。サーバーは、ウィジェットの「値」の取得や設定を行うだけですが、デフォルトとはこの「値」のことです。

C、C++、および Java コードに対して getter と setter を使用する例も、ウィジェットごとに示されています。

詳細について

広範囲に渡るオンラインドキュメントが用意されています。内容の一覧を見るには、HTML ブラウザで次のファイルを開いてください。

```
$VISUROOT/lib/locale/<YourLocale>/sc/index.html
```

ここで、VISUROOT は Sun WorkShop Visual のインストールディレクトリであり、<YourLocale> はユーザーが使用しているロケールです。自分のロケールがわからない場合には、端末ウィンドウに locale と入力してみてください。これによってロケール情報が出力されます。出力された内容のうち、LANG に割り当てられた文字列を使用します。ロケールの例を次に示します。

- C (英語用)
- ja (日本語用)

また、いったんコードを生成すると、コードが生成されたディレクトリに `index.html` というファイルが作られます。このファイルにはハイパーテキストリンクが含まれており、このリンクからオンライン資料にアクセスできます。

ラベルとボタン

表 C-1　ラベルとボタンに使用できる getter と setter

リソース名	型	
Value	char *	デフォルト
Sensitive	int	

C のコード例

C のコードでは、SC_GET と SC_SET のマクロを使用します。これらのマクロは、最初の引数としてリソース名 (つまり、取得または設定しているもの) を受け取り、2 番目の引数としてグループ構成要素を受け取ります。SC_SET の場合には、3 番目の引数は新しく設定する値になります。詳細については、559 ページの第 16 章「取得と設定用のスマートコード」を参照してください。

次の例では、ラベルの値を取得および設定します。「label1」はグループ mygroup のメンバーです。

```
char * val = SC_GET(Value,mygroup->label1);
SC_SET(Value,mygroup->label1,"my label");
```

C++ のコード例

C++ では、グループはクラスです。グループメンバーは、このクラスの変数であり、クラスでもあります。getter と setter は、グループメンバーのクラスのメソッドです。

次の例は、上述の C のコード例と同様にラベルの値を取得および設定します。「label1」はグループ mygroup のメンバーです。

```
mygroup_c * g = (mygroup_c *) getGroup();
char * val = g->label1->getValue();
g->label1->setValue("my label");
```

Java のコード例

Java では、グループはクラスです。グループメンバーは、このクラスの変数であり、クラスでもあります。getter と setter は、グループメンバーのクラスのメソッドです。

次の例は、上述の C のコード例と同様にラベルの値を取得および設定します。「label1」はグループ mygroup のメンバーです。

```
mygroup_c g = (mygroup_c ) getGroup();
String val = g.label1.getValue();
g.label1.setValue("my label");
```

トグル

表 C-2　トグルに使用できる getter と setter

リソース名	型	
State	int[1]	デフォルト
Sensitive	int	

1.Java の場合、State はブール値です。

C のコード例

C のコードでは、SC_GET と SC_SET のマクロを使用します。これらのマクロは、最初の引数としてリソース名 (つまり、取得/設定しているもの) を受け取り、2 番目の引数としてグループ構成要素を受け取ります。SC_SET の場合には、3 番目の引数は新しく設定する値になります。詳細については、559 ページの第 16 章「取得と設定用のスマートコード」を参照してください。

次の例では、トグルの状態 (オンまたはオフ) を取得および設定します。「toggle1」と「toggle2」は、グループ mygroup のメンバーです。

```
int state = SC_GET(State,mygroup->toggle1);
SC_SET(State,mygroup->toggle2, state);
```

C++ のコード例

C++ では、グループはクラスです。グループメンバーは、このクラスの変数であり、クラスでもあります。getter と setter は、グループメンバーのクラスのメソッドです。

次の例は、上述の C のコード例と同様に一方のトグルの状態を取得し、他方のトグルの状態を設定します。「toggle1」と「toggle2」は、グループ mygroup のメンバーです。

```
mygroup_c * g = (mygroup_c *) getGroup();

int state = g->toggle1->getState();

g->toggle2->setState(state);
```

Java のコード例

Java では、グループはクラスです。グループメンバーは、このクラスの変数であり、クラスでもあります。getter と setter は、グループメンバーのクラスのメソッドです。

次の例は、上述の C のコード例と同様にトグルの状態を取得および設定します。「toggle1」と「toggle2」は、グループ mygroup のメンバーです。

```
mygroup_c g = (mygroup_c ) getGroup();

boolean state = g.toggle1.getState();

g.toggle2.setState(state);
```

ここでは「state」はブール値です。C と C++ では int になります。

テキストフィールド、テキスト、スクロールテキスト

テキスト制御用の getter と setter は、groups_c/sc_types.h で定義された SC_GET() マクロと SC_SET() マクロを介して使用できます。

表 C-3　テキストに使用できる getter と setter

リソース名	型	
Value	char *	デフォルト
Sensitive	int	

C のコード例

C のコードでは、SC_GET と SC_SET のマクロを使用します。これらのマクロは、最初の引数としてリソース名 (つまり、取得/設定しているもの) を受け取り、2 番目の引数としてグループ構成要素を受け取ります。SC_SET の場合には、3 番目の引数は新しく設定する値になります。詳細については、559 ページの第 16 章「取得と設定用のスマートコード」を参照してください。

次の例では、テキストの値 (つまり、内容) を取得および設定します。「text1」はグループ mygroup のメンバーです。

```
char * contents = SC_GET(Value,mygroup->text1);

SC_SET(Value,mygroup->text1,”a new string”);
```

C++ のコード例

C++ では、グループはクラスです。グループメンバーは、このクラスの変数であり、クラスでもあります。getter と setter は、グループメンバーのクラスのメソッドです。

次の例は、上述の C のコード例と同様にテキストの内容を取得および設定します。「text1」はグループ mygroup のメンバーです。

```
mygroup_c * g = (mygroup_c *) getGroup();

char * contents = g->text1->getValue();
```

```
g->text1->setValue("a new string");
```

Java のコード例

Java では、グループはクラスです。グループメンバーは、このクラスの変数であり、クラス自身でもあります。getter と setter は、グループメンバーのクラスのメソッドです。

次の例は、上述の C のコード例と同様にテキストの内容を取得および設定します。「text1」はグループ mygroup のメンバーです。

```
mygroup_c g = (mygroup_c ) getGroup();
String contents = g.text1.getValue();
g.text1.setValue("a new string");
```

スケール

表 C-4　スケールに使用できる getter と setter

リソース名	型	
Value	int	デフォルト
Sensitive	int	

C のコード例

C のコードでは、SC_GET と SC_SET のマクロを使用します。これらのマクロは、最初の引数としてリソース名 (つまり、取得/設定しているもの) を受け取り、2 番目の引数としてグループ構成要素を受け取ります。SC_SET の場合には、3 番目の引数は新しく設定する値になります。詳細については、559 ページの第 16 章「取得と設定用のスマートコード」を参照してください。

次の例では、スケールウィジェットの値 (つまり、スケール上の数値) を取得および設定します。「scale1」はグループ mygroup のメンバーです。

```
int val = SC_GET(Value,mygroup->scale1);
SC_SET(Value,mygroup->scale1, 1);
```

C++ のコード例

C++ では、グループはクラスです。グループメンバーは、このクラスの変数であり、クラス自身でもあります。getter と setter は、グループメンバーのクラスのメソッドです。

次の例は、上述の C のコード例と同様にスケールウィジェットのスケール値を取得および設定します。「scale1」はグループ mygroup のメンバーです。

```
mygroup_c * g = (mygroup_c *) getGroup();

int scaleValue = g->scale1->getValue();

g->scale1->setValue(1);
```

Java のコード例

Java では、グループはクラスです。グループメンバーは、このクラスの変数であり、クラスでもあります。getter と setter は、グループメンバーのクラスのメソッドです。

次の例は、上述の C のコード例と同様にスケールウィジェットのスケール値を取得および設定します。「scale1」はグループ mygroup のメンバーです。

```
mygroup_c g = (mygroup_c ) getGroup();

int scaleValue = g.scale1.getValue();

g.scale1.setValue(1);
```

リストとスクロールリスト

表 C-5　リストに使用できる getter と setter

リソース名	型	
Items	char **	
Sensitive	int	
SelectedItems	char **	デフォルト

C のコード例

C のコードでは、SC_GET と SC_SET のマクロを使用します。これらのマクロは、最初の引数としてリソース名 (つまり、取得/設定しているもの) を受け取り、2 番目の引数としてグループ構成要素を受け取ります。SC_SET の場合には、3 番目の引数は新しく設定する値になります。詳細については、559 ページの第 16 章「取得と設定用のスマートコード」を参照してください。

次の例では、リストウィジェットの選択した項目のリストを取得および設定します。「list1」はグループ mygroup のメンバーです。選択した項目のリストは、ヌルで終わっている文字列配列です。

```
char ** my_stringlist = SC_GET(SelectedItems,mygroup->list1);

SC_SET(SelectedItems,mygroup->list1,a_new_stringlist);
```

C++ のコード例

C++ では、グループはクラスです。グループメンバーは、このクラスの変数であり、クラスでもあります。getter と setter は、グループメンバーのクラスのメソッドです。

次の例は、上述の C のコード例と同様にリストウィジェットの選択項目を取得および設定します。「list1」はグループ mygroup のメンバーです。選択した項目のリストは、ヌルで終わっている文字列配列です。

```
mygroup_c * g = (mygroup_c *) getGroup();

char ** my_stringlist = g->list1->getSelectedItems();

g->list1->setSelectedItems(a_new_stringlist);
```

Java のコード例

Java では、グループはクラスです。グループメンバーは、このクラスの変数であり、クラスでもあります。getter と setter は、グループメンバーのクラスのメソッドです。

次の例は、上述の C のコード例と同様にリストウィジェットの選択項目を取得および設定します。「list1」はグループ mygroup のメンバーです。配列内に存在する文字列の数を調べるには、Java に内蔵されている「<配列>.length」を使用してください。

```
mygroup_c g = (mygroup_c ) getGroup();

String [] my_stringlist = g.list1.getSelectedItems();

int how_many = my_stringlist.length;

g.list1.setSelectedItems(a_new_stringlist);
```

オプションメニュー

表 C-6　オプションメニューに使用できる getter と setter

リソース名	型	
Label	char *	
Sensitive	int	
SelectionByName[1]	char *	
SelectionByIndex	int	デフォルト

1. これは表示される文字列であり、ウィジェット名ではありません。

C のコード例

C のコードでは、SC_GET と SC_SET のマクロを使用します。これらのマクロは、最初の引数としてリソース名 (つまり、取得 / 設定しているもの) を受け取り、2 番目の引数としてグループ構成要素を受け取ります。SC_SET の場合には、3 番目の引数は新しく設定する値になります。詳細については、559 ページの第 16 章「取得と設定用のスマートコード」を参照してください。

次の例では、オプションメニューの選択項目を取得および設定します。「optionMenu1」はグループ mygroup のメンバーです。

```
char * val = SC_GET(SelectionByName, mygroup->optionMenu1);

SC_SET(SelectionByName,mygroup->optionMenu1, "Option 1");
```

C++ のコード例

C++ では、グループはクラスです。グループメンバーは、このクラスの変数であり、クラスでもあります。getter と setter は、グループメンバーのクラスのメソッドです。

次の例は、上述の C のコード例と同様にオプションメニューの選択項目を取得および設定します。「optionMenu1」はグループ mygroup のメンバーです。

```
mygroup_c * g = (mygroup_c *) getGroup();

char * val = g->optionMenu1->getSelectionByName();

g->optionMenu1->setSelectionByName("Option 2");
```

Java のコード例

Java では、グループはクラスです。グループメンバーは、このクラスの変数であり、クラスでもあります。getter と setter は、グループメンバーのクラスのメソッドです。

次の例は、上述の C のコード例と同様にオプションメニューの選択項目を取得および設定します。「optionMenu1」はグループ mygroup のメンバーです。

```
mygroup_c g = (mygroup_c ) getGroup();

String val = g.optionMenu1.getSelectionByName();

g.optionMenu1.setSelectionByName("Option 3");
```

ラジオボックス

表 C-7　ラジオボックスに使用できる getter と setter

リソース名	型	
Label	char *	
Sensitive	int	
SelectionByName[1]	char *	
SelectionByIndex	int	デフォルト

1. これは表示される文字列であり、ウィジェット名ではありません。

C のコード例

C のコードでは、SC_GET と SC_SET のマクロを使用します。これらのマクロは、最初の引数としてリソース名 (つまり、取得/設定しているもの) を受け取り、2 番目の引数としてグループ構成要素を受け取ります。SC_SET の場合には、3 番目の引数は新しく設定する値になります。詳細については、559 ページの第 16 章「取得と設定用のスマートコード」を参照してください。

次の例では、ラジオボックスウィジェットの選択項目のラベルを取得および設定します。「radiobox1」はグループ mygroup のメンバーです。

```
char * str = SC_GET(SelectionByName, mygroup->radiobox1);

SC_SET(SelectionByName,mygroup->radiobox1, "The text to show");
```

C++ のコード例

C++ では、グループはクラスです。グループメンバーは、このクラスの変数であり、クラスでもあります。getter と setter は、グループメンバーのクラスのメソッドです。

次の例は、上述の C のコード例と同様にラジオボックスウィジェットの選択項目のラベルを取得および設定します。「radiobox1」はグループ mygroup のメンバーです。

```
mygroup_c * g = (mygroup_c *) getGroup();

char * str = g->radiobox1->getSelectionByName();
```

```
g->radiobox1->setSelectionByName("The text to show");
```

Java のコード例

Java では、グループはクラスです。グループメンバーは、このクラスの変数であり、クラスでもあります。getter と setter は、グループメンバーのクラスのメソッドです。

次の例は、上述の C のコード例と同様にラジオボックスウィジェットの選択項目のラベルを取得および設定します。「radiobox1」はグループ mygroup のメンバーです。

```
mygroup_c g = (mygroup_c ) getGroup();
String str = g.radiobox1.getSelectionByName();
g.radiobox1.setSelectionByName("The text to show");
```

付録D

アプリケーションのデフォルト

はじめに

Sun WorkShop Visual には、独自のアプリケーションリソース設定のセットがあります。この付録では、ユーザーが状況に合わせて変更する可能性の高いリソースについて説明します。これらのリソースは、システムの構成に合わせて任意のアプリケーションリソースで変更、あるいは追加することができます。たとえば、 次のような Sun WorkShop Visual のアプリケーションファイルのみの変更が望ましい場合もあります。

`$VISUROOT/lib/locale/<ロケール>/app-defaults/visu`

CDE ウィンドウマネージャの場合

`$VISUROOT/lib/locale/<ロケール>/app-defaults/CDE/visu`

`$VISUROOT` は、Sun WorkShop Visual のインストールディレクトリです。上述の <ロケール> ディレクトリは、デフォルトで `ja` に設定されます。ただし、`ja` 以外のロケールを使用している場合には、現在使用しているロケールの名前になります。ロケールの詳細は、706 ページの「ロケール」を参照してください。X ウィンドウシステムの参考文献については、1011 ページの付録 E「参考資料」を参照してください。

また、Sun WorkShop Visual のリソース設定を、ホームディレクトリの `.Xdefaults` という名前のファイルに追加することができます。リソース設定をしていないものがある場合は、新たに作成することもできます。任意の X ウィンドウのアプリケーションに対するリソース情報を、本章で説明する形式でリソース設定に含めることができます。詳細は、X ウィンドウシステムのマニュアルを参照してください。

本節では、リソース名は太字で印刷してあります。これらは、適切なリソースファイルに 1 行を追加するだけで設定することができます。次の例を参考にしてください。

```
visu.autoSave: true
```

上の行の **autoSave** はリソース名であり、*true* は設定値です。

Sun WorkShop Visual リソースファイルには、ここで紹介する数よりも多くのリソースが含まれています。ほとんどのリソースは、他国語で作業を行う場合、または特別な条件がある場合に限り、変更を必要とします。本付録で説明されていないリソースについては、Sun WorkShop Visual リソースファイルのコメントを参照してください。

リソースには、Sun WorkShop Visual の大画面および小画面に対して異なる設定を持たせることができます。小画面の場合は `small_visu`、大画面に対してはアプリケーションクラス名Sun WorkShop Visualを使用します。Sun WorkShop Visual のリソース設定は、特に`small_visu`で設定がない場合、`small_visu` にも適用します。

本章に含まれていないリソースを使用しての Sun WorkShop Visual のカスタマイズについては、805 ページの第 25 章「構成」を参照してください。

汎用

nameFont

ウィジェット変数名を表示するフォントです。`visu` のデフォルトは以下のとおりです。

--helvetica-medium-r-normal--12-*-*-*-*-*-*-**

`small_visu` のデフォルトは以下のとおりです。

--helvetica-medium-r-normal--10-*-*-*-*-*-*-**

labelFontList

ダイアログやメニュー上のラベルを表示するフォントです。visu のデフォルトは以下のとおりです。

--helvetica-medium-r-normal--12-*-*-*-*-*-*-**

`small_visu` のデフォルトは以下のとおりです。

--helvetica-medium-r-normal--10-*-*-*-*-*-*-*

buttonFontList

ダイアログやメニュー上のボタンのラベルを表示するフォントです。visu のデフォルトは以下のとおりです。

--helvetica-medium-r-normal--12-*-*-*-*-*-*-*

small_visu のデフォルトは以下のとおりです。

--helvetica-medium-r-normal--10-*-*-*-*-*-*-*

textFontList

テキストボックスやリスト上のテキストを表示するフォントです。visu のデフォルトは以下のとおりです。

--helvetica-medium-r-normal--12-*-*-*-*-*-*-*

small_visu のデフォルトは以下のとおりです。

--helvetica-medium-r-normal--10-*-*-*-*-*-*-*

warnOnClose

保存されていない変更のあるダイアログを閉じようとする場合に、Sun WorkShop Visual に警告を発生させます。デフォルト: *true*

warnOnSelect

現在選択されているウィジェットに保存されていない変更がある時に、別のウィジェットを選択しようとする場合に、Sun WorkShop Visual に警告を発生させます。デフォルト: *true*

dialogsTopLevel

Sun WorkShop Visual に、ダイナミックディスプレイに対してダイアログシェルにではなく、最上位置シェルを作成させます。これにより、ダイアログのアイコン化およびスタック方法のプロパティが変更されます。ただし、この設定は最終的なアプリケーションに影響します。デフォルト: *false*

visu**dialogs.transient:false* で、同様の効果を得ることができます。

smallScreen

Sun WorkShop Visual に小アイコンを使用させます。デフォルトの `visu` に対しては *false*、 `small_visu` の場合は *true*

intrinsicsHeadersPrefix

X イントリンシクス・ヘッダーファイルの接頭辞です。X のインストール場所に依存する、重要なリソースです。デフォルト: *X11*

definitionsFileName

パレットに構成した定義に対しての指定を含んでいるファイルです。このリソースの値は、*/bin/sh* によって拡張されるため、環境変数を含むことができます。
デフォルト: *$HOME/.xddefinitionsrc*

コールバックとプレリュード編集

callbackEditing

このリソースは、コールバック編集が使用可能かどうかを制御します。false に設定されている場合には、この機能に関連するボタンは表示されません。デフォルト: *true*

editor

このリソースは、ファイルの編集に使用される実行可能プログラムの位置を指定します。デフォルト: `$BINARYROOT/lib/scripts/xd_edit`

sunEditService

Sun WorkShop Visual で Sun Edit Server を使用するかどうかを決めるブール型のリソースです。使用しない場合は、上記の「**editor**」リソースを使用します。デフォルト: *true*

sunEditServerPath

バイナリ名を含む eserve バイナリのパスを指定します。リソースファイルに定義されていない場合は、デフォルトで NULL に設定されます。これは、sunEditService を false に設定するのと同じ効果になります。

Microsoft Windows

Windows

このリソースが設定されている場合は、Sun WorkShop Visual は Microsoft Windows モードで実行します。詳細は、第 11 章「Microsoft Windows 用のデザイン」を参照してください。デフォルト: *false*

mfcTextWarningBackground

リソースが Microsoft Windows 様式コードでは使用されないことを示すために使用する色です。

mfcCarriageReturn

このリソースが設定されている場合は、Microsoft Window に対して生成されたファイルにおいて、改行文字に加えてリターン文字を生成します。デフォルト: *false*

フィルタ

objectFileSuffix

生成されたメークファイルで使用されるオブジェクトファイル接尾辞です。
デフォルト: *.o*

executableFileSuffix

生成されたメークファイルで使用される実行可能ファイルの接尾辞です。
デフォルト: 空文字列

uidFileSuffix

生成されたメークファイルで使用される *uid* ファイルの接尾辞です。
デフォルト: *.uid*

コード生成ダイアログ

「コード生成」ダイアログでは、デフォルトで設定された名前が生成されるファイル名として使用されます。このファイル名のデフォルト設定は、変更することができます。詳細を以下に示します。

cCodeSuffix

C コードを生成する際のコードファイルに使用するデフォルトの接尾辞です。
デフォルト: .c

cppCodeSuffix

C++ コードを生成する際のコードファイルに使用するデフォルトの接尾辞です。
デフォルト: .cpp

cStubsCodeSuffix

C コードを生成する際のスタブファイルに使用するデフォルトの接尾辞です。
デフォルト: .c

cppStubsCodeSuffix

C++ コードを生成する際のスタブファイルに使用するデフォルトの接尾辞です。
デフォルト: .cpp

cExternsCodeSuffix

C コードを生成する際の外部宣言ファイルに使用するデフォルトの接尾辞です。
デフォルト: .h

cppExternsCodeSuffix

C++ コードを生成する際の外部宣言ファイルに使用するデフォルトの接尾辞です。
デフォルト: .h

uilExternsCodeSuffix

UIL を生成する際の外部宣言ファイルに使用するデフォルトの接尾辞です。
デフォルト: .h

uilCodeSuffix

UIL を生成する際のコードファイルに使用するデフォルトの接尾辞です。
デフォルト: .uil

cUilCodeSuffix

UIL 用の C コードを生成する際のコードファイルに使用するデフォルトの接尾辞です。デフォルト: .c

cPixmapsCodeSuffix

C コードを生成する際のピックスマップ・ファイルに使用するデフォルトの接尾辞です。デフォルト: .h

cppPixmapsCodeSuffix

C++ コードを生成する際のピックスマップ・ファイルに使用するデフォルトの接尾辞です。デフォルト: .h

uilPixmapsCodeSuffix

UIL を生成する際のピックスマップ・ファイルに使用するデフォルトの接尾辞です。
デフォルト: .h

xResourcesCodeSuffix

X リソースファイルのデフォルトの接尾辞です。デフォルト: .res

windowsResourcesCodeSuffix

Microsoft Windows のリソースファイルのデフォルトの接尾辞です。デフォルト: .rc

backupCodeSuffix

追加のスタブ生成に使用する、バックアップファイルのデフォルトの接尾辞です。デフォルト: .bak

defaultCodeFileName

メークファイルを除くすべてのファイルの基本となる名前です。デザインが保存されると、保存ファイル名が使用されます。デフォルト: untitled

makefileCodeFileName

メークファイルのデフォルトの名前です。デフォルト: Makefile

注 - CodeFileName という文字列と基本となる言語を示す語で、独自のリソース名を作成することもできます。たとえば、cCodeFileName は、C コードを生成する際のコードファイルの名前を示しています。

---

cUilCodeExtension

UIL 用の C コードを生成する際に、コードファイル名に追加されるデフォルトの拡張子です。デフォルト: _uil

cStubsCodeExtension

C コードを生成する際に、スタブファイル名に追加されるデフォルトの拡張子です。デフォルト: _stubs

cppStubsCodeExtension

C++ コードを生成する際に、スタブファイル名に追加されるデフォルトの拡張子です。デフォルト: _stubs

cPixmapsCodeExtension

C コードを生成する際に、スタブファイル名に追加されるデフォルトの拡張子です。デフォルト: _pixmaps

cppPixmapsCodeExtension

C++ コードを生成する際に、スタブファイル名に追加されるデフォルトの拡張子です。デフォルト: _pixmaps

uilPixmapsCodeExtension

UIL を生成する際に、スタブファイル名に追加されるデフォルトの拡張子です。デフォルト: _pixmaps

生成

makefileTemplate

デフォルトのメークファイルを定義します。詳細は、第 25 章「構成」を参照してください。

motifMakeTemplateFile

Motif 様式のデフォルトメークファイル・テンプレートを指します。
デフォルト: `$VISUROOT/make_templates/motif`

mmfcMakeTemplateFile

Motif XP 様式のデフォルトメークファイル・テンプレートを指します。
デフォルト: `$VISUROOT/make_templates/mfc`

javaWidgetLib と javaWidgetsInclude

生成されたメークファイルのリンクとコンパイルの行に、`libjavawidgets.a` ライブラリと関連するインクルードファイルを指すために置かれます。デフォルト値はそれぞれ、`$VISUROOT/src/java_widgets/lib/libjavawidgets.a` と *-I*`$VISUROOT/src/java_widgets/h` です。

c++BaseClassHeader

基底クラス定義を含んでいる C++ 用ヘッダーファイルです。必要に応じて引用符 “” または山括弧 <> を含みます (デフォルトは括弧)。これを無効にするためには、空文字列に設定します。デフォルト: *xdclass.h*

xpmHeader

XPM ライブラリが必要とする定義用のヘッダーファイルです。必要に応じて引用符 (“”) または山括弧 <> を含みます (デフォルトは括弧)。デフォルト: *<xpm.h>*

generateXFuncCLinkage

Sun WorkShop Visual に *_XFUNCPROTOBEGIN* および *_XFUNCPROTOEND* マクロをヘルプリンク外部宣言の付近に作成させます。デフォルト: *true*

generateRedefineDefaultWidgetName

C++ 基底クラス宣言において、Sun WorkShop Visual はメンバー関数 *create()* に対するウィジェット名引数のデフォルト値を *widget_name = NULL* に設定します。派生クラスに対しては、基底クラスからこれらのデフォルトの引数の値を継承するため、この動作は必要ではありません。しかし、*widget_name* 引数なしで *create()* 関数が呼び出される場合には、多くのコンパイラがこの動作を必要とします。Sun WorkShop Visual は、このようなコードは決して生成しませんが、この動作を必要とするソースをすでにユーザーが持っている場合もあります。デフォルト: *true*

生成されたコード内のコメント

javaBeginGuard と javaEndGuard

生成された Java コード内のガードコメントの一部として記述されるテキストです。このテキストには、コードを読みやすくするという目的しかありません。デフォルトでは、それぞれ次のようになります。

```
Sun WorkShop Visual-generated code - do not edit >>>
```

および

```
<<< Sun WorkShop Visual-generated code ends.
```

ヘルプ

helpKey

ヘルプコールバックを呼び出すキーです。デフォルト: *<Key>F1*
キーボードに Help キーがある場合には、*<Key>Help* を試してください。

helpDir

ヘルプファイルを探す際に使用する検索パスです。検索パスは、コロンで区切って指定します。通常のヘルプの場合は、Sun WorkShop Visual はこのディレクトリ内で `entry.html` という名前のファイルを探し出します。このファイルから他のヘルプファイルへのリンクが開始されます。ユーザー定義ウィジェットまたは定義に対してテキストヘルプを使用している場合は、Sun WorkShop Visual はこのディレクトリでサブディレクトリ *local* を探し出します。
デフォルト: *$*VISUROOT*/lib/locale/<* 使用するロケール *>/help*
$VISUROOT は Sun WorkShop Visual インストールディレクトリのルートのパスです。

userHelpCatString

ユーザー定義ウィジェットと、定義に対してのヘルプファイル名の構築に使用される区切り文字列です。デフォルト: “.”

ユーザー定義ウィジェットに対するテキストファイルヘルプは、ファイルとタグの組み合わせによって定義されます。通常のヘルプの場合は、ファイルとタグは連結されて *helpDir/local* に相対的なファイル名を生成します。*userHelpCatString* の値は、文字列の作成時には文書 (document) とタグ (tag) の間のセパレータとして使用されます。たとえば、*userHelpCatString* が "." である場合、ヘルプファイルは *document.tag* となります。別の設定としては、"/" を使用して *document/tag* を生成することができます。ヘルプファイルは、HTML 形式で作成されることを前提としています。

自動保存

autoSave

自動保存機能を有効にします。デフォルト: *true* (有効)

autoSaveThreshold

自動保存の前に行われる変更の数です。デフォルト: 20

autoSaveExt

自動保存により追加されるファイル名の拡張子です。デフォルト: *.sav*
たとえば、*fred.xd* は *fred.xd.sav* になります。

配置エディタ

以下のリソースは、配置エディタの色を制御します。

formFillColor

配置エディタの背景色です。デフォルト: *#dedededededede*

formStrokeColor

配置エディタがフォームの輪郭に使用する色です。デフォルト: *Black*

widgetFillColor

配置エディタでウィジェットを表わすボックスの塗りつぶしに使用する色です。
デフォルト: *Blue*

widgetStrokeColor

ウィジェットの輪郭に使用する色です。デフォルト: *Black*

widgetDestinationColor

整列あるいは均等配置を行う場合に、最後に選択されたウィジェットを示すために使用する色です。このウィジェットに対して操作が行われます。
デフォルト: *#9a9ae1163232*

widgetSelectColor

位置揃えあるいは均等配置を行う場合に、選択されたウィジェットを示すために使用する色です。これらのウィジェットは、操作によって移動させることができます。
デフォルト: *#ecc9c9eacdda*

attachmentColor

配置エディタでアタッチメントを表わす線に使用される色です。デフォルト: *Black*

階層の色

以下のリソースは、デザイン階層にウィジェットを描画する場合に使用されます。

widgetForeground

ビットマップ型のウィジェットアイコンの前景色です。デフォルト: *Black*
これは、ピックスマップまたはアイコンが描画される色を表わします。ビットマップアイコンの場合は、*widgetBackground* リソースと対照的な色でなければなりません。カラーピックスマップアイコンには使用されません。

widgetBackground

ウィジェットアイコンの背景色です。この色は、カラーが *none* (なし) と設定されているカラーピックスマップから透けて表示されます。デフォルト: *#fdfdf5f5ebeb*

highlightForeground

ウィジェットが強調表示されている場合に、ビットマップ型アイコンを描画するために使用される前景色です。ビットマップアイコンのために、*highlightBackground* リソースと対照的な色で表示を行う必要があります。カラーピックスマップアイコンに対しては使用されません。デフォルト: *#fdfdf5f5ebeb*

モノクロディスプレイを使用する場合は、デフォルト設定にアイコンが黒く表示される可能性があります。この場合は、*widgetBackground* と同じ値に設定します。

highlightBackground

ウィジェットが強調表示されている場合に使用される背景色です。ビットマップアイコンのために、*highlightForeground* リソースと対照的な色で表示を行う必要があります。デフォルト: *Red* この色は、カラーが *none* (なし) と設定されているカラーピックスマップの部分から透けて表示されます。

モノクロディスプレイを使用する場合は、デフォルト設定にアイコンが黒く表示される可能性があります。この場合は、*widgetForeground* と同じ値に設定します。

構造の色

以下のリソースは、構造あるいは C++ クラスとして指定されているウィジェットを示すために使用される色を制御します。これらのリソースは、「構造の色」トグルが設定されている場合に限って有効です。

widgetFunctionBackground

関数構造として指定されているウィジェットに対しての背景色です。
デフォルト: *#ecc9c9eacdda*

widgetStructBackground

データ構造として指定されているウィジェットに対しての背景色です。
デフォルト: *#9a9ae1163232*

widgetClassBackground

C++ クラスとして指定されているウィジェットに対しての背景色です。
デフォルト: *#d2dfe785f3ce*

widgetChildrenBackground

「子のみ生成」として指定されているウィジェットに対しての背景色です。
デフォルト: *#d8c0d2d1f3f2*

定義およびインスタンスに対しての背景

以下のリソースは、定義およびインスタンスの表示を制御します。

widgetInstanceBackground

インスタンスのツリーにおける背景色です。デフォルト: *#ecc0ecc086db*

widgetDefinitionBackground

定義のツリーにおける背景色です。デフォルト: *#86dbecc0ecc0*

widgetInstanceBitmap

モノクロディスプレイの場合のみ。インスタンスである階層のツリーにおける背景ビットマップです。デフォルト: *25_foreground*

widgetDefinitionBitmap

モノクロディスプレイの場合のみ。定義である階層のツリーにおける背景ビットマップです。デフォルト: *25_foreground*

問題回避

tileOriginBug

ツリーアイコンの表示が困難なサーバーにおける問題を回避します。
デフォルト: *false*

alternateFolding

フォールドしたウィジェットを点描表示できないというサーバーにおける問題を回避します。フォールドされたアイコンを、×印で描画します。デフォルト: *false*

xorByInvert

カラーマップセルが不必要に使用されないように、Sun WorkShop Visual は、カラーマップに存在する色を使用して XOR 表示を行います。これにより、対話的な描画ができなくなる場合があります。この場合は、描画方法を XOR から INVERT (反転) に変更し、異なる結果を生じさせることもできます。デフォルト: *false*

freeStaticColors

サーバーによっては、アプリケーションによる静的カラーマップ内のカラーセルの解放を許可しない (または、単純に要求を無視する) ものもあります。静的なデフォルト表示タイプを持つディスプレイ (代表的なものは、VGA タイプの画面) を使用していて、カラー解放を行おうとすると X エラーによって Sun WorkShop Visual がクラッシュする場合には、このリソースを設定します。デフォルト: *true*

closeColourMatching

サーバーの許容範囲を超える時間を使用して Sun WorkShop Visual のメインウィンドウを読み込む場合、このリソースを false に設定すると、クローズカラー一致の検索でカラーが見つからなかったときに冗長な検索を行わずに済みます。そのため、Sun WorkShop Visual のピックスマップの外観は変になります。デフォルト: *true*

FrameMaker

これらのリソースは、FrameMaker でアプリケーションのヘルプを開発する場合に使用される引数の指定に使用します。

frameMakerBinary

FrameMaker ヘルプファイルを表示するために実行されるバイナリです。
デフォルト：*viewer*

frameMakerTimeout

Sun WorkShop Visual が FrameMaker への対話を始める前に FrameMaker の起動を待つ時間 (ミリ秒) です。デフォルト：*20*

構成

パレットおよびツールバーの構成については、805 ページの第 25 章「構成」も参照してください。

stopList

パレットに表示させないウィジェットのリストです。ウィジェットはコンマで区切られます。以下にすべての Motif ウィジェットを示します。

```
XmDialogShell, XmMainWindow, XmMenuBar, XmPulldownMenu, XmRadioBox,
XmRowColumn, XmFrame, XmDrawingArea, XmBulletinBoard, XmForm,
XmPanedWindow, XmScrolledWindow, XmMessageBox, XmMessageTemplate,
```

```
XmCommand, XmSelectionPrompt, XmSelectionBox, XmFileSelectionBox,
XmLabel, XmPushButton, XmToggleButton, XmDrawnButton, XmArrowButton,
XmCascadeButton, XmOptionMenu, XmSeparator, XmScale, XmScrollBar,
XmTextField, XmText, XmScrolledText, XmList, XmScrolledList
```

ユーザー定義ウィジェットなどに対しては、以下のようにクラス名を使用してください。

```
boxWidgetClass, formWidgetClass
```

デフォルトは空文字列です。

pm_icons、pm_labels、pm_both

これらはパレットメニューの「表示方法」メニューにある 3 個のトグルボタンです。その中の 1 個がデフォルトの表示方法として設定されます。
デフォルト: visu**pm_icons.set:true*

付録 E

参考資料

はじめに

本付録では、このマニュアルで取り上げている資料の詳細、およびその他の役立つ参考資料を紹介します。

ISBN 番号が示してありますので、最新版については書店にお問い合わせください。

本マニュアルで取り上げている資料

Kernighan, B.W. and Ritchie, D.M., *The C Programming Language*.
Prentice Hall, 1978

First edition 1978 ISBN 0-13-110163-3
Second edition 1988 ISBN 0-13-110362-8
(プログラミング言語 C ～ANSI 規格準拠～ 第 2 版 (訳書訂正)、ブライアン・W・カーニハン、デニス・M・リッチ、共立出版)

Open Software Foundation, *OSF/Motif* (5 vols). Prentice Hall, 1990, 1991, 1992.

OSF/Motif Style Guide1993 ISBN 0-13-643123-2
(Motif スタイル・ガイド)

OSF/Motif Programmer's Guide 1993 ISBN 0-13-643107-0
(Motif プログラマーズ・ガイド)

OSF/Motif Programmer's Reference 1993 ISBN 0-13-643115-1
(Motif プログラマーズ・リファレンス)

OSF/Motif User's Guide 1993 ISBN 0-13-643131-3
(Motif ユーザーズガイド)

Application Environment/Specification (AES) User Environment, Revision C 1992 ISBN 0-13-043621-6

O'Reilly and Associates, *The X Window System Series* (8 vols). O'Reilly and Associates, Inc., 1988, 1989, 1990, 1991, 1992,1993
(Motif プログラミング・マニュアル、ソフトバンク)

Volume 0: 1992 ISBN 1-56592-008-2

Volume 1: 1992 ISBN 1-56592-002-3

Volume 2: 1992 ISBN 1-56592-006-6

Volume 3M: 1993 ISBN 1-56592-015-5

Volume 4M: 1992 ISBN 1-56592-013-9

Volume 5: 1992 ISBN 1-56592-007-4

Volume 6A: 1994 ISBN 1-56592-016-3

Volume 6B: 1993 ISBN 1-56592-038-4

Volume 7: 1993 ISBN 0-937175-87-0

Volume 8: 1992 ISBN 0-937175-83-8

X および Motif に関する資料

以下の書籍は、X Window System および OSF/Motif についての参考文献です。

初級 / 中級

Berlage, Thomas, *OSF/Motif: Concepts and Programming*.
Addison-Wesley, 1991. ISBN 0-201-55792-4

Jones, Oliver, *Introduction to the X Window System*.
Prentice Hall, 1989. ISBN 0-13-499997-5
(X Window ハンドブック、オリバー・ジョーンズ、三浦明美、アスキー)

Rost, Randi J., *X and Motif Quick Reference Guide*.
Digital Press, 1993. ISBN 13-972746-9
(X/Motif クイックリファレンスガイド、ランド・J・ロスト、秋元信彦、トッパン)

Young, Douglas A., *X Window System: Programming and Applications with Xt, 2nd OSF/Motif Edition*. Prentice Hall, 1994. ISBN 0-13-123803-5
(X Toolkit プログラミング OSF/MOTIF 版 (第 2 版)、D. A. ヤング、川手恭輔、トッパン)

Young, Douglas A., *OSF/Motif Reference Manual*.
Prentice Hall, 1990. ISBN 0-13-642786-3

中級 / 上級

Asente, Paul J. and Swick Ralph R., *X Window System Toolkit*.
Digital Press, 1990. ISBN 1-55558-051-3

Scheifler, R.W. and Gettys, J., *X Window System, 3rd edition*.
Digital Press, 1992. ISBN 13-971201-1

George, Alistair and Riches, Mark, *Advanced Motif Programming Techniques*.
Prentice Hall, 1993. ISBN 0-13-219965-3

ユースネットの X ニュースグループに、より多くの参考資料に関する情報がわかりやすく掲示されています。内容は、DEC 社の Ken Lee 氏によって毎月更新されています。

C++ およびオブジェクト指向プログラミングに関する資料

Stroustrup, Bjarne, *The C++ Programming Language, 2nd edition*.
Addison-Wesley Publishing Company, 1991. ISBN 0-201-53992-6
(プログラミング言語 C++ (第 2 版)、アジソンウェスレイ・トッパン情報科学シリーズ、ビョーン・ストラウストラップ、斎藤信男、トッパン)

Young, Douglas, *Object-Oriented Programming with C++ and OSF/Motif*.
Prentice-Hall, 1992. ISBN 0-13-630252-1
(オブジェクト指向プログラミングと C++: OSF/Motif 版、ダグラス・A・ヤング、磯谷正孝、トッパン)

Microsoft Foundation Class に関する資料

Blaszczak, Mike, *The Revolutionary Guide to MFC 4 Programming with Visual C++*. Wrox Press Ltd. ISBN 1-874416-92-3

Java に関する資料

Flanagan, David, *Java in a Nutshell*.
O'Reilly & Associates, Inc., 1996. ISBN 1-56592-183-6

Arnold, Ken and Gosling, James, *The Java Programming Language*.
Prentice Hall, 1996. ISBN 0-201-63455-4

ネットワーキングとWWWに関する資料

Harold, Elliotte Rusty, *JAVA Network Programming*.
O'Relly, 1997. ISBN 1-56592-227-1

World Wide Web Consortium, *World Wide Web Journal: Key Specifications of the World Wide Web*.
O'Reilly 季刊誌 ISBN 1-56592-190-9 (Volume 1 の Issue 2)

Ed Tittel, Mark Gather, Sebastian Hassinger & Mike Erwin, *World Wide Web Programming With HTML & CGI*.
IDG Books Worldwide, Inc., 1995. ISBN 1-56884-703-3

国際化に関する資料

O'Donnell, Snadra Martin, *Programming for the World: a guide to internationalization*.
Prentice Hall, 1994. ISBN 0-13-722190-8

Lunde, Ken, *Understanding Japanese Information Processing*.
O'Reilly & Associates Inc., 1993. ISBN 1-56592-043-0

CDE に関する資料

Common Desktop Environment User's Guide.
Addison-Wesley. ISBN 0-201-48951-1
(共通デスクトップ環境 ユーザーズ・ガイド)

HTML に関する資料

Graham, Ian S., *HTML Source book*.
John Wiley & Sons Inc., 1995. ISBN 0-471-11849-4

用語集

C++ クラス ウィジェット (C++ class widget)

C++ クラスに指定されているデザイン階層内のウィジェット。Sun WorkShop Visual は、そのウィジェットに対し C++ のクラス定義のコードを生成し、子孫ウィジェットをそのクラスのメンバーとします。

CGI

Common Gateway Interface の省略名。CGI 標準規格では、Web HTTP サーバーで外部プログラムを実行するための規則が規定されています。外部プログラムはサーバーに対して外部情報を提供することから、ゲートウェイと呼ばれます。

CGI スクリプト (CGI Script)

データ通信の CGI 標準規格に準拠した言語で書かれたプログラムまたはシェルスクリプト。

config ユーティリティ (config utility)

visu_config ユーティリティの一般名。

DTD

Document Type Difinition の省略名。DTD とは、SGML 構文にある宣言 (エンティティ、要素、属性、リンク、マップなど) の集まりのことです。この宣言によって、ドキュメントのクラス (型) に使用できる構成要素や構造体が定義されます。

FontList

1 つの文字列の中で複数の異なるフォントを表示できるように、グループ化されたフォントの集まり。FontList は Motif の用語です。

GIF

Graphic Interchange Format の省略名。イメージを格納するときのファイル形式。GIF はピクセルごとに 8 ビットのカラー情報しか格納しないので、他の形式のほうが有用です。

HTML

HyperText Markup Language の省略名。HTML とは、印刷可能な文字のみを使用するパブリックドメイン形式のことで、どのテキストエディタでも作成することができます。多くのアプリケーションで標準として使用されています。

HTTP

HyperText Transfer Protocol の省略名。URL の冒頭に付ける「http」という 4 文字は、Web サーバーに対してブラウザとの通信方法を示しています。WWW サーバーと接続すると、両方のシステムでこのプロトコルが使用され、ドキュメントがサーバーからシステムに転送されます。詳細は、次の URL を参照してください。

`http://info.cern.ch/hypertext/WWW/Protocols/HTTP/HTTP2.html`

IDE

Integrated Development Environment の省略名。IDE とは、プログラム開発に役立つ統合ツールのセットで構成されているアプリケーションのことです。IDE には、通常、コンパイラ、デバッガ、コードエディタが含まれています。

Java

インターネット環境に対応したライブラリを備えるプログラミング言語。Java は、オブジェクト指向でインタプリタ型の言語であり、移植性に優れています。また、スレッド化されており、自動で記憶領域の管理や例外割り込みを行います。

MFC

Microsoft Foundation Classes の省略名。Microsoft Windows 上で実行可能なユーザーインタフェースを構築するための基底クラスのセット。

MIME

Multimedia Internet Message Extension の省略名。MIME を使用すると、グラフィックス、音声、マルチメディアなど、テキスト以外のものを Web ブラウザで処理する機能を拡張できます。これに対して HTML が処理するのはテキストだけであり、その他の情報を処理するには拡張機能を利用しなくてはなりません。MIME は電子メールにバイナリファイルをアタッチする場合にも使用します。ブラウザは、MIME の型をカテゴリごと、ファイルタイプごとにスラッシュで区切って認識します (例 : image/gif)。MIME の型が登録済みの場合、ブラウザはファイルを復号して開くためのアプリケーションを起動します。

SGML

Standard Generalized Markup Language の省略名。SGML は、ドキュメントの情報を他のドキュメントのパブリッシングシステムや電子配信、構成管理、データベース管理、在庫管理などのアプリケーションで共有させることのできるデータの符号化方式です。SGML の定義については、ISO 8879:1986 Information Processing Text and Office Systems; Standard Generalized Markup Language (SGML) を参照してください。

thin クライアント (Thin Client)

クライアントとは、クライアント/サーバーシステムにおいて「サービスを受ける側」のことです。たとえば、ftp サイトからファイルをダウンロードするには、ftp クライアントソフトウェアを使用します。*thin* クライアントとは、ユーザーインタフェースの thin 階層だけを提供するクライアントのことです。クライアントに必要なすべての処理は、サーバーが実行します。

XmString

Motif のコンパウンド文字列構造体。

X リソースファイル (X resource file)

Sun WorkShop Visual が生成する編集可能なファイル。デザインのリソースファイルの一部、あるいは全部を明示的に含みます。

アクション (action)

ウィジェットのイベントに対する動作を規定する名前 (「準備 (arm)」、「活性化 (activate)」、「ヘルプ (help)」など)。ウィジェットのトランスレーション・テーブルに定義されています。アクションは、ウィジェットのアクションテーブルの関数に対応づけられています。

アクションテーブル (action table)

アクションを実行するアクション関数にアクションを対応づける、ウィジェットに関連したテーブルです。

アクションルーチン (action routine)

アクションを実行する関数。アクション関数の多くは、Motif によって提供されます。Sun WorkShop Visual のユーザーは、独自のアクション関数を作成することもできます。

アクセラレータ (accelerator)

マウスボタンを使用しないでメニュー内のコマンドを実行するためのキー、あるいはキーの組み合せ。

アクセラレータテキスト (accelerator text)

アクセラレータをユーザーに示すために、メニューのボタン上に表示されるテキスト。

アタッチメント (attachment)

ウィジェットの 1 辺を兄弟ウィジェットあるいはその親配置ウィジェットの 1 辺に固定する制約。アタッチメントは、固定幅、または配置ウィジェットの寸法のパーセント (%) で作成することができます。

アプリケーションクラス名 (application class name)

生成されたアプリケーションのアプリケーションシェルに与えられる名前。この名前はシェルのウィンドウのタイトル、およびそのアプリケーションに属しているリソースを識別するために使用されます。Sun WorkShop Visual では、アプリケーションクラス名はコード生成時に割り当てられます。

アプリケーションシェル (Application Shell)

アプリケーションの一次ウィンドウに使用されるシェルウィジェットの一種。

アプレット (Applet)

Java でかかれた小規模のアプリケーション。Web ページに組み込まれており、ページをブラウズすると実行されます。

位置アタッチメント (position attachment)

ウィジェットの 1 辺を、親フォームの幅または高さに指定されたパーセント位置に接続すること。この型のアタッチメントは、現在のフォームの寸法に合わせて調整を行います。

一次セレクション (primary selection)

配置エディタに関する記述で使用する用語で、最後に選択されたウィジェットを指します。ウィジェットの整列を行う場合、他のすべての選択されたウィジェットは、一次セレクションのウィジェットを基準に整列します。

イベント (event)

キーを押す、ボタンを押すなど、ユーザー入力の要素の 1 つ。

インスタンス (instance)

個々のウィジェットデータ構造体。ウィジェットのインスタンスとは、ウィジェットクラスの個々の実体のことです。ウィジェットのインスタンス化とは、ウィジェットクラスのインスタンスを作成することを意味します。

イントラネット (Intranet)

企業内のネットワーク。通常、同じイントラネット上のコンピュータ間にはファイアウォールを設定しませんが、イントラネットとインターネットの間には、ファイアウォールを設定することがあります。

ウィジェット (widget)

Motif ツールキット、あるいはグラフィカル・ユーザーインタフェースの構築ブロックとして使用され、他のツールキットで事前定義されているデータ構造体。

ウィジェットアタッチメント (widget attachment)

フォーム内で、ウィジェットから別のウィジェットへ設定されるアタッチメント。

ウィジェットクラス (widget class)

特定の型のウィジェット。

ウィジェットパレット (widget palette)

Sun WorkShop Visual メイン画面の左側にある領域で、使用可能なウィジェットクラスのアイコンを表示します。

ウィジェット名 (widget name)

(1) X リソースファイル内でウィジェットのインスタンスを区別するために使用される名前。この名前は、変数名と同じである必要はなく、また、固有である必要はありません。
(2) Sun WorkShop Visual 再現機能で使用するウィジェット名。複数のインスタンスに同じウィジェット名を使用する場合は、番号が付けられます。

ウィンドウ保持領域 (window holding area)

Sun WorkShop Visual メイン画面の右上にある領域。デザイン内の各ウィンドウに対して 1 個ずつシェルアイコンが表示されます。

オフセット (offset)

アタッチメントで接続された 2 個のウィジェット間、あるいは、ウィジェットとそのウィジェットが接続されている配置ウィジェットとの間の固定距離 (ピクセル単位)。

親 (parent)

子の配置を管理し、決定するウィジェット。Sun WorkShop Visual の場合、デザイン階層内では親ウィジェットを子の上に表示します。

ガジェット (gadget)

Primitive クラスから派生する特定のウィジェットの変形。ウィジェットとは異なり、ガジェットは各インスタンスに対してウィンドウを内部で作成する必要がありません。ウィジェットの代わりにガジェットを使用した場合の利点は、システムによって異なります。

カラーオブジェクト (color object)

名前と色を関連付けたもの。

関数構造体 (function structure)

Sun WorkShop Visual が、生成されたコードに独自の作成関数を生成する対象となるウィジェット。

起因する (originate)

アタッチメントに関して使用される用語。アタッチメントの作成元であるウィジェットを、アタッチメントが「起因する」ウィジェットと呼びます。

キーシム (keysym)

トランスレーションの詳細フィールドにおいて、キーを識別するために使用される文字列。

基本モジュール (primary module)

Sun WorkShop Visual により生成されるコードモジュール。インタフェースの作成関数を含んでいます。

クライアントデータ (client data)

コールバック関数に渡すことができる単一の引数。

クラス階層 (class hierarchy)

Motif におけるウィジェットクラスの抽象的な階層。クラス階層は、デザイン階層とは区別されます。

グレー表示 (graying out)

アイコン、プッシュボタン、メニューオプションなどがぼやけて表示されること。Sun WorkShop Visual では、使用できないコマンドはグレー表示されます。

継承 (inherit)

スーパークラス (親) の属性を所有すること。派生したウィジェットクラスは「そのスーパークラスを継承する」というように使用されます。

子 (children)

親ウィジェットにより管理され、可視状態の場合はその範囲内に含まれるウィジェット。Sun WorkShop Visual では、子ウィジェットはデザイン階層で親の下に表示されます。

候補語リスト (candidate list)

入力方式に従って入力した読みがなに対する、候補となる語 (表意文字) のリスト。通常、表意文字を使用する場合にのみ使用されます。

コアウィジェット (Core widget)

すべてのウィジェットクラスを派生する広義のスーパークラス。

コアリソースパネル (Core resource panel)

コア、プリミティブ、マネージャの各スーパークラスのリソースを設定することができる特殊なリソースパネル。

コードプレリュード (code prelude)

Sun WorkShop Visual のダイアログを使用して作成され、生成されたコードの特定の場所に挿入されるコード行。

コールバック (callback)

コールバック関数およびユーザーのアクションのリストを指定するウィジェット構造体のフィールド。ウィジェットでユーザーアクションが発生すると、コールバックリストにある関数が実行されます。

コールバック関数 (callback function)

コールバックリストにある関数の 1 つ。

コールバックリスト (callback list)

コールバックに関連する関数 (コールバック関数) のグループ。

構成領域 (construction area)

Sun WorkShop Visual メイン画面上の描画領域。デザイン階層が表示されます。

子のみ生成ウィジェット (Children Only widget)

コード生成時に Sun WorkShop Visual が場所の確保のために使用するウィジェット。「子のみ生成」ウィジェット自体に対しては、コードは生成されませんが、データ構造体、関数構造体、C++ クラスとして指定されるその子孫には、コードが生成されます。

コンストレイントウィジェット (constraint widget)

子がコンストレイントパネルを持っている 2 種類のウィジェット (フォームと区画ウィンドウ) のどちらか一方。

コンストレイントリソース(constraint resources)

コンストレイントパネルに表示されるリソース。これらのリソースはコンストレイントウィジェットの任意の子に対して表示できます。一般に、子のサイズまたは位置を制御するウィジェットのリソース。

コンテナウィジェット (container widget)

子を含んだり、編成したりすることを主目的とするウィジェット。

コントロール (Control)

Windows や Java などのシステムでインタフェースオブジェクトに幅広く使用されている用語。ボタンやテキストフィールドなど、基本的なユーザーインタフェースオブジェクトを指しています。

コンバータ (converter)

リソースパネル上の文字表現を数値的なリソース値に変換する関数または関数の集合。特定の型のユーザー定義ウィジェットに必要です。

コンパウンド文字列 (compound string)

テキスト文字列とフォント情報、方向情報を組み合わせた、Motif のデータ構造体。

サーバー (Server)

クライアント/ サーバーシステムで、処理を担当する側。サーバーとは、通常、情報、ファイル、Web ページなどのサービスをログインしているクライアントに提供するコンピュータのことです。サーバーという用語は、サーバーハードウェアを起動するソフトウェアやオペレーティングシステムを示す場合にも使用します。クライアント / サーバーシステムは、レストランのウェイターと客の関係によく似ています。

最上位シェル (Top level Shell)

アプリケーションのメインウィンドウ以外で、デザイン内の一次ウィンドウとして使用されるシェルウィジェットの種類。

作業領域 (work area)

任意の型の子ウィジェットを 1 個持つことができるコンポジットウィジェット (メッセージボックス、ダイアログテンプレート、ファイル選択ボックスなど) の中央にある領域。

作成関数 (creation procedure)

Sun WorkShop Visual により生成される関数。シェルウィジェットとその子を作成します。

作成の前プレリュード (pre-create prelude)

指定されたウィジェットが作成される前に挿入されるコードプレリュード。

サブクラス (subclass)

他のクラスから派生したウィジェットクラス。

修飾キーリスト (modifier list)

トランスレーションのイベント指定にあるフィールド。イベントを発生させるために修飾キー (<Ctrl> および <Shift> など) を押す必要があるかどうかを指定します。

循環アタッチメント (circular attachment)

フォーム配置において、ウィジェット A からウィジェット B にアタッチされている場合のウィジェット B からウィジェット A へのアタッチメント。ウィジェット A から B、B から C、そして C から A へのアタッチメント、あるいはさらに大きなループ状のアタッチメントも、循環とみなされます。循環アタッチメントは Motif では認められていません。

照会文字列 (Query String)

疑問符 (?) の後に表示される URL の一部を指します。照会文字列には標準の形式があり、検索機能などのアプリケーションで使用できます。

詳細 (detail)

トランスレーションのイベント指定にある最後のフィールド。通常は押すべきキーを指定するために使用します。

縮小表示 (thumbnail sketch)

捕獲したダイアログのアイコンを縮小表示したもの。Sun WorkShop Visual 捕獲ダイアログで使用されます。

シングルステップ (single step)

スクリプトを再現する際に、次のコマンドまでの内容のみを実行すること。

スーパークラス (superclass)

他のクラスの派生元となるウィジェットクラス。

スクリプト (script)

記録モードで、Sun WorkShop Visual 再現ツールが作成するファイル。アプリケーション上で行われたアクションが記録されています。再現モードでは、Sun WorkShop Visual 再現ツールがこのスクリプトを読み取り、そこに記述されているアクションを実行します。スクリプトは単純なテキストで、キーワードと変数で構成されています。

スタブファイル (stubs file)

コールバックに対しての空の括弧、関数宣言、 文を含む生成ファイル。

ステータス行 (status line)

(1) Sun WorkShop Visual のメインウィンドウおよび配置エディタの左下部領域のこと。この領域には、現在マウスポインタが置かれているメニューオプション、ツールバー上のボタン、ウィジェットに関する情報が表示されます。

(2) 入力方式が有効である場合に、アクティブウィンドウ (通常はメインウィンドウの最下部) に文字列が表示されます。この文字列はステータス行と呼ばれ、現在の入力モードおよび入力方式で供給されるその他の情報をユーザーに報告します。

選択されたウィジェット (selected widget)

デザイン階層でそのアイコンが現在強調表示されているウィジェット。Sun WorkShop Visual では、何らかの動作を行う前にはウィジェットを選択する必要があります。

ソースファイル (source file)

Sun WorkShop Visual の生成するコードファイルの 1 つ。「リソースファイル」と比較されている場合は、基本モジュールのことを示します。

ダイアログシェル (Dialog Shell)

2 次ウィンドウに使用されるシェルウィジェットの型。ダイアログシェルは親シェルから独立させてアイコン化することはできません。

ダイナミックディスプレイ (dynamic display)

ユーザーが構築し、編集するとおりにデザイン階層を表示する動画。Sun WorkShop Visual 上で作成されます。これによって、デザインの結果を目で確認することができます。

代用サーバー (Proxy)

代用サーバーとは、他のサーバーにアクセスする際に使用する情報をキャッシュに書き込んで、アクセス速度を向上させるシステムです。Web では、まず、局所的なデータ検索が試行され、該当するデータが見つからなかった場合は、データが常駐しているリモートサーバーからデータをフェッチします。

タグ (tag)

ドキュメント名と、ドキュメント内のハイパーテキストマーカーで構成されるハイパーテキストヘルプ情報。

単純フォントオブジェクト (simple font object)

名前と単一フォントを関連付けたもの。

停止 (pause)

スクリプトを再生する際に、再生を停止してスクリプト中の同じ場所に一定の時間とどまっていること。

ディザ (dither)

白と黒のピクセルをさまざまなパターンで使用して、グレーを表現します。

データ構造体 (data structure)

Sun WorkShop Visual がデータ構造の `typedef` と、構造体の型を設定する作成関数を生成し、それに対してポインタを返す対象となるウィジェット。

デザイン階層 (design hierarchy)

インタフェースに対してのデザインを構成する個々のウィジェットインスタンスの階層。クラス階層とは区別されます。

テンプレート (template)

AppGuru デザイナーで使用するユーザーインタフェースの例。この例をデフォルト設定として、デザイン作成が開始されます。

トランスレーション (translation)

キーやボタンを押す操作などのイベントや、イベントのシーケンスをアクションに変換すること。

トランスレーションテーブル (translations table)

個々のウィジェットに対するトランスレーションのリスト。

ニーモニック (mnemonic)

メニューまたはメニュー項目の文字のひとつ (頭文字である場合が多い)。ニーモニックの文字のキーを押すと、該当するメニューが選択されます。

二次セレクション (secondary selection)

配置エディタに関する記述で使用する用語。最後に選択されたウィジェット (一次セレクション) 以外のすべての選択されたウィジェットを指します。ウィジェットの整列を行う場合、二次セレクションのウィジェットは、一次セレクションであるウィジェットを基準に整列します。

入力フォーカス (input focus)

ウィンドウ、またはウィンドウ内の構成要素がキーボード入力を受けつけることを示します。キーボードフォーカスと呼ばれる場合もあります。

ハードワイヤされたリソース (hard-wired resource)

X ソースファイルにではなく、ソースコードに生成されたリソース値。ハードワイヤされたリソース値は、アプリケーションを作成し直す以外の方法で変更することはできません。

配置 (layout)

インタフェースにおけるウィジェットの物理的配列。

配置ウィジェット (layout widget)

配置エディタで使用可能なウィジェット。フォーム、ブリテンボード、ローカラム、描画領域など。

配置エディタ (Layout Editor)

フォーム、ブリテンボード、描画領域などのウィジェットにコンストレイントリソースを設定するために使用する、対話型の画面エディタ。

ハイパーテキスト (Hypertext)

リンクを介して接続可能なテキスト、グラフィック、その他のメディア。

派生ウィジェットクラス (derived widget class)

クラス階層で、別のクラスの下位にあるウィジェットクラス。派生クラスは、その上位のクラスのすべての属性のほか、独自の特殊な属性も備えています。

パッケージ (package)

Java コードとの接続に使用する用語。パッケージは C のライブラリのようなものです。パッケージによって、互いに関連しているオブジェクトファイルをグループにまとめることができます。各ソースファイルのラベルには、ソースファイルが属しているパッケージの名前を使用します。パッケージ名は、ソースファイルを含んでいるディレクトリに基づきます。ソースファイルは、使用するパッケージを「取り込む」ことができます。

ピックスマップ・オブジェクト (pixmap object)

名前とピックスマップを関連付けたもの。

フォームアタッチメント (Form attachment)

ウィジェットの 1 辺を、その親フォームの 1 辺へ固定オフセット (水平または垂直) で接続すること。フォームのサイズ変更時に、オフセットの変更は行われません。

フォールド (fold)

Sun WorkShop Visual の表示コマンドのひとつ。ウィジェットの子がフォールドされているときは画面上に表示されないため、表示スペースを節約できます。

フォントオブジェクト (font object)

名前とフォントまたはフォントリストを関連付けたもの。

複合フォントオブジェクト (multiple font object)

Motif によって FontList リソースにグループ化されたフォント群を、ある名前に結び付けたもの。

複数選択 (multiple selection)

作業領域上のウィジェット階層で、一度に複数のウィジェットを選択すること。

プリミティブウィジェット (Primitive widget)

Motif クラス階層における広義のスーパークラス。すべてのボタン型ウィジェットおよびその他のクラスのいくつかが派生されます。プリミティブウィジェットは、コアウィジェットから派生しています。

プレリュード (prelude)

生成されたファイルに挿入されるコードの集まり。「作成の前プレリュード (pre-create prelude)」、「マネージの前プレリュード (pre-manage prelude)」、「モジュールプレリュード (module prelude)」も参照してください。

ページセレクタ (page selector)

リソースパネルの上部にあるオプションメニュー。リソースのあるページから別のリソースページに移動するときに使用します。

編集モード (editing modes)

配置エディタにおいて、画面左側のラジオボタンで制御されるように指定されたマウスボタン 1 の動作。「移動」、「サイズ変更」、「自己」、「位置」のモードがあります。

編集領域 (editing area)

配置エディタ上の描画領域。編集可能なウィジェットの配置図が表示されます。

変数名 (variable name)

生成コードでウィジェットのデータ構造体の識別に使用する名前。C 変数名であるため、その他のウィジェット、アプリケーション内での他の変数名や関数名、他の C の予約語と同じ名前は使用できません。

捕獲 (capture)

Motif アプリケーションを実行しているデザインを分析し、理想的なデザインを含む .xd ファイルを作成すること。この処理は、Sun WorkShop Visual 捕獲機能で行われます。

ボタンボックス (button box)

メッセージボックス、ダイアログテンプレート、ファイル選択ボックスなどのコンポジットウィジェットの下部にある、ボタンのための領域。

マスクトグル (masking toggle)

リソースパネル内で各リソースの隣にあるラベルのついていないトグル。X リソースファイルに生成されるリソースと、コードにハードワイヤされるリソースを指定するために使用されます。

マネージの前プレリュード (pre-manage prelude)

指定されたウィジェットが作成された後、マネージされる前に挿入されるコードプレリュード。通常は、コールバックに対してのクライアントデータの設定に使用されます。

マネージャウィジェット (Manager widget)

Motif クラス階層における広義のスーパークラス。ほとんどのコンテナウィジェットはこのクラスから派生されます。マネージャウィジェットは、コアウィジェットから派生します。

密な結合 (tight binding)

ウィジェットが必ず含まれるように設定されたリソース結合。

モジュール
プレリュード
(module prelude)

(1) Sun WorkShop Visual の生成ヘッダーおよび *#include* 指令の直後に挿入されるコード行。

(2) モジュールプレリュードまたはモジュール見出しのどちらかを意味する一般的な用語。

モジュール見出し
(module heading)

生成された基本モジュールおよびスタブファイルの先頭に挿入される数行のコード。

モニター (monitor)

スクリプトの記録または再現動作の実行中に、再現コマンドをウィンドウ内に表示すること。

緩い結合
(loose binding)

ウィジェット群またはアプリケーション全体に渡って、共通リソースとして使用することのできるリソース結合。

ラジオボタン
(radio buttons)

ラジオボックス内、あるいは「ラジオボタン動作」リソースが「はい」に設定されているメニューやローカラム内でグループ化されているトグルボタン。グループ内では、一度に 1 個のボタンしか選択することができません。

リソース (resource)

ウィジェットデータ構造内で設定可能なフィールド。リソースは、ウィジェットの外観や動作に関する多くの性質を制御します。リソースの設定は、設計者でもユーザーでも行うことができます。

リソースパネル
(resource panel)

Sun WorkShop Visual の対話型画面。ウィジェットに対してのリソース値を指定することができます。

リソースプレリュード (resource prelude)

生成された X リソースファイルの先頭に挿入されるプレリュード。通常は、個々のウィジェットにではなく、アプリケーション全体に大まかなリソースの結合を追加するために使用されます。

リンク (link)

Sun WorkShop Visual によって提供される、事前定義されたコールバック。

リンク関数 (link function)

リンクによって実行される関数コード。

列挙型リソース (enumeration resource)

使用可能な値の集合が限定されているリソース。Sun WorkShop Visual では、列挙型リソースは、リソースパネルの「設定」ページに表示されます。

ユーザーアクション (user action)

キーやボタンの操作など、コールバックを起動する事前定義されたイベントの集合。

索引

記号

数字

A

C

L

M

N

T

U

V

W

X

あ

い

う

え

お

か

き

く

け

こ

さ

し

す

せ

そ

た

ち

つ

て

と

な

に

ね

は

ひ

ふ

へ

ほ

れ

ろ

わ