iSeries™

AS/400e VisualAge® RPG プログラミング バージョン4.5.1 (Windows 版)

Windows® 版 バージョン 4

SC88-5607-05
(英文原典：SC09-2449-05)

iSeries™

AS/400e VisualAge® RPG プログラミング バージョン4.5.1 (Windows 版)

Windows® 版 バージョン 4

SC88-5607-05
(英文原典：SC09-2449-05)

ご注意

本書の情報および本書で紹介する製品をご使用になる前に、 503 ページの『特記事項』に記載されている一般情報をお読みください。

本書は、IBM WebSphere Development Studio Client for iSeries のバージョン 4、および新しい版で明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。

本書は、SC88-5607-04 の改訂版です。

本文および図表の変更や追加は、その変更や追加の箇所の左側に縦線で示してあります。

本マニュアルに関するご意見やご感想は、次の URL からお送りください。今後の参考にさせていただきます。

http://www.ibm.com/jp/manuals/main/mail.html

なお、日本 IBM 発行のマニュアルはインターネット経由でもご購入いただけます。詳しくは

http://www.ibm.com/jp/manuals/ の「ご注文について」をご覧ください。

(URL は、変更になる場合があります)

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

原　典： SC09–2449–05
iSeries
Programming with VisualAge® RPG
Version 4 for Windows

発　行： 日本アイ・ビー・エム株式会社

担　当： ナショナル・ランゲージ・サポート

第1刷 2004.4

この文書では、平成明朝体™W3、平成明朝体™W9、平成角ゴシック体™W3、平成角ゴシック体™W5、および平成角ゴシック体™W7を使用しています。この(書体*)は、(財）日本規格協会と使用契約を締結し使用しているものです。フォントとして無断複製することは禁止されています。

注* 平成明朝体™W3、平成明朝体™W9、平成角ゴシック体™W3、
平成角ゴシック体™W5、平成角ゴシック体™W7

目次

本書について

本書は、VisualAge® RPG を使用してクライアント/サーバー・アプリケーションを開発するための手引きです。ここでは、設計から梱包および配布までのアプリケーション開発サイクルのすべての段階でのそれぞれのステップを説明します。概念およびプロセスを明確にするためにプログラミングの例を含めてあります。

本書の対象読者

本書は VisualAge RPG を使用してクライアント/サーバー・アプリケーションを開発するプログラマーのために書かれています。読者は iSeries™ 400™ システムでの RPG アプリケーションの開発に精通していると見なされます。

前提条件および関連情報

iSeries および AS/400e の技術情報を調べるための開始点として iSeries Information Center を使用してください。Information Center をアクセスする方法には、以下の 2 つがあります。

- 次の Web サイトから:

  `http://www.ibm.com/eserver/iseries/infocenter`
- OS/400 オーダーで配送された CD-ROM から:

  iSeries Information Center, SK88-8055-03.

iSeries Information Center には、CL コマンド、システム・アプリケーション・プログラミング・インターフェース (API)、論理区画、クラスター、Java ™、TCP/IP、Web サービス、およびセキュア・ネットワークなどのアドバイザーおよび重要なトピックが入っています。また、関連した IBM® Redbooks へのリンクおよび Technical Studio や IBM ホーム・ページなどのその他の Web サイトへのインターネット・リンクも含まれています。

本書の使用法

注: この製品については、*WebSphere Development Studio Client for iSeries ご使用に際して*, SD88-5054-03を参照してください。

*VisualAge for RPG プログラミングブック*は次のパーツが構成されています。

クライアント/サーバー・アプリケーションを初めて見る
この部では、クライアント/サーバー・アプリケーションを VisualAge RPG で作成する際に必要となるステップについて説明します。サンプル・アプリケーションの設計および開発に触れて、設計上の問題を説明します。

パーツの処理
この部には、VisualAge RPG パーツとのグラフィカル・ユーザー・インターフェースの作成およびそれらのパーツを動かすためのプログラム・ロジックの作成についてのヒントが入っています。ここでは、すべての命令コード

の使用法は説明しておらず、すべての属性やイベントの詳細についても説明していません。そのような情報については、*VisualAge for RPG WINDOWS 版 言語解説書* および *ADTS/CS VisualAge for RPG パーツ解説書* を参照してください。

iSeries 400 データの処理
この部では、iSeries 400 サーバー上のデータにアクセスするためのアプリケーションの設定方法、および既存のサーバー・アプリケーションをプログラマブル・ワークステーション (PWS) で実行される VisualAge RPG アプリケーションに変換することによって、そのアプリケーションを再使用する方法について説明します。

拡張トピック
この部では、VisualAge RPG アプリケーションに追加できる多くの機能についてそのハイライトを示します。ここには、ユーザー・アプリケーションからの印刷、出力の編集、デバッガーの使用法、ピクチャーおよびサウンド・ファイルの使用法、オンライン・ヘルプの作成、メッセージの追加、および DBCS システムでのアプリケーションの実行などのトピックがあります。また、VisualAge RPG アプリケーションがデータを共用して通信できる別のさまざまな方法についても説明しています。

アプリケーションの配布
この部では、VisualAge RPG 実行時コードとアプリケーションのパッケージ方法について説明します。また、ユーザーの PWS への実行時コードおよびアプリケーションのインストール方法についても説明しています。

VisualAge RPG ライブラリー

VisualAge RPG ライブラリーには、以下の資料が含まれています。

VisualAge for RPG プログラミング

本書には、VisualAge RPG を使用してアプリケーションを作成する場合の具体的な情報が記載されています。これには、アプリケーション開発サイクルのすべての段階で実行する必要があるステップが、設計からパッケージおよび分配まで説明されています。VisualAge RPG アプリケーション開発の概念と工程を明らかにするために、プログラミング例が収録されています。

ADTS/CS VisualAge for RPG パーツ解説書

本書には、VisualAge RPG の**パーツ**、**パーツ属性**、**パーツ・イベント**、および**イベント属性**に関する情報が記載されています。本書は、VisualAge RPG を使用してアプリケーションを開発しているすべての方々の参照資料です。

VisualAge for RPG WINDOWS 版 言語解説書

本書には、VisualAge RPG コンパイラーを使用して実装される RPG IV 言語に関する情報が記載されています。 次の事項が含まれています。
- 文字セット、記号名および予約語、コンパイラー指示、ならびに標識などの言語基本エレメント
- データ・タイプおよびデータ形式
- エラーおよび例外処理

- 仕様
- 組み込み関数、式、および命令コード

製品全体の概要については、*WebSphere Development Studio Client for iSeries ご使用に際してを参照してください。*

関連資料のリストについては、本書の終わりにある参考文献を参照してください。

また以下のオンライン・ソースでも IBM WebSphere Development Studio Client for iSeries についての最新の情報を見つけることができます。

「Development Studio Client」ホーム・ページ
`ibm.com/software/ad/wdsc/`

オンライン情報へのアクセス

VisualAge RPG には、種々のオンライン・ブックおよびオンライン・ヘルプが含まれています。プロダクトの使用中にヘルプにアクセスしたり、またプロダクトの使用中または別途に本書を表示することができます。

オンライン・ブックの使用法

オンライン・ブックを表示するためには、次のいずれかを実行してください。
- VisualAge RPG GUI Designer またはエディター・ウィンドウの**ヘルプ** プルダウン・メニューから本書の名前を選択してください。
- **「スタート」**メニューから本書にアクセスしてください。**「プログラム」→「IBM WebSphere Development Studio Client for iSeries」**を選択します。次に**「文書」**を選択します。

PDF 形式の資料

VisualAge RPG 資料は、URL `http://www.ibm.com/eserver/iseries/infocenter` の iSeries Information Center から PDF の形式で使用可能です。

注: ワークステーションで資料を PDF 形式で表示するには、Adobe Acrobat Reader (Windows 版 3.01 以降) が必要です。システム設置場所にこのプログラムがない場合には、Adobe Systems Web サイト (http://www.adobe.com) からコピーをダウンロードできます。

以下の VisualAge RPG 資料は、PDF 形式で使用可能です。
- *VisualAge for RPG プログラミング*
- *ADTS/CS VisualAge for RPG パーツ解説書*
- *VisualAge for RPG WINDOWS 版 言語解説書*

この製品については、*WebSphere Development Studio Client for iSeries ご使用に際して*, SD88-5054-03を参照してください。

オンライン・ヘルプの使用法

オンライン・ヘルプは VisualAge RPG のすべての領域から使用することができます。特定のウィンドウ、ダイアログ・ボックス、またはプロパティー・ノートブックのヘルプを表示するには、**「ヘルプ」**プッシュボタン (使用可能な場合) を選択します。

注: HTML 形式のヘルプを表示するには、ワークステーションには Netscape Navigator 4.04 以降、Microsoft® Internet Explorer 4.01 以降などのフレーム使用可能 Web ブラウザーが必要です。(推奨ブラウザーは、Netscape Navigator 4.6 または Internet Explorer 5.0 です。)

ヘルプ情報の使用法

任意の時点でヘルプ情報を表示するためには、 F1 キーを押してください。表示されるヘルプは、入力フォーカスを持つインターフェースの領域に固有のものです。入力フォーカスはメニュー項目、ウィンドウ、ダイアログ・ボックス、およびプロパティー・ノートブック上にあるか、あるいはこれらの特定の部分にあります。

ダイアログ・ボックスのヘルプ情報については、ウィンドウの右上隅の疑問符 (使用可能な場合) をクリックしてください。 マウスの矢印の隣に疑問符 (?) が現れます。ワードまたはフィールドをクリックすると、その特定のフィールドについてのヘルプ情報が表示されます。

言語依存ヘルプの使用法

言語依存ヘルプを表示するためには、編集ウィンドウで F1 キーを押してください。カーソルが命令コードに位置付けられている場合には、その命令コードのヘルプが表示されます。その他の場合は、現在の指定のヘルプが表示されます。

このリリースの新機能

新規のパーツ属性およびイベントには以下のものがあります。

- **キャンバス**・パーツには現在 **VKeyPress** イベントがあります。このイベントに応答できるすべてのパーツに 1 つのアクション・サブルーチンをコーディングするだけで済みます。
- **ActiveX**: **ReturnVal** は、メソッド呼び出しの値を戻します。**OCXPropIdx** は、配列プロパティーの索引を設定または検索します。この配列は、ストリングまたは数値とすることができます。
- **コンテナー**: **DeleteRcd** は、1 つまたはすべてのレコードを削除するために設定できます。
- ***component**: **HostName** は、ワークステーションのホスト名を host_name IP_address の形式で戻します。
- **メニュー項目**: **Visible** 属性が追加されました。さらにショート・キーが追加されています。ショートカット・キーのテキストは、実行時にメニュー項目ラベルの横に自動的に表示されるようになります。

本書には、前のリリースの README からの情報および他の技術的訂正が含まれています。前の README の内容には、以下の新規 VisualAge RPG パーツ属性の説明が含まれています。

- サブファイル・パーツ属性

  EditColumn、**EditIndex**、および **EditText** は、列番号、列の行、または編集中のテキストを戻します。

  MapViewCol および **ViewColumn** は、サブファイルのデータ列が表示される順序を変更するのに使用できます。

  DColFRVCol および **VColFRDCol** は、それぞれデータ列と表示列を戻します。

  SortAsc および **SortDesc** は、列のデータをソートするのに使用できます。
- ウィンドウ・パーツおよびイメージ・パーツ用の **PrintAsIs** は、イメージを印刷し、その縦横比を維持します。
- **DropValue** は、ドロップ操作の後でラベルを変更または保存するために使用されます。
- **OnTop** は、指定されたノートブック・ページを現行ページに変更します。
- **グラフ**・パーツには現在 **Popup** イベントがあります。

変更箇所は、縦線 (|) で示されています。

第 1 部 クライアント/サーバー・アプリケーションを初めて見る

第 1 章 クライアント/サーバー・アプリケーションの作成

この項では、VisualAge® RPG を使用してクライアント/サーバー・アプリケーションを作成する方法について説明します。サンプル・アプリケーションは、次の開発過程の段階を説明するために使用されます。

1. ユーザーへの表示内容とアプリケーションによる処理内容の設計。
2. グラフィカル・ユーザー・インターフェース (GUI) の作成。
3. GUI パーツの属性の設定。
4. GUI を駆動するプログラム・ロジックの作成。
5. アプリケーション用のメッセージとオンライン・ヘルプの作成。

サンプル・アプリケーションについて

ビデオ・ストア・カタログと呼ばれるサンプル・アプリケーションは VisualAge RPG を使用して作成されていて、VisualAge RPG サンプル・フォルダーに入っています。これは、お客様がビデオを購入するために使用することができて、ビデオを発注する前にビデオ・クリップを見るために使用できるプレビュー・コンポーネントのあるオンライン・カタログを提供します。

ビデオに関する情報は iSeries 400 サーバーのデータベースに保管されます。ビデオ・ストア・カタログを使用するお客様は実際には、iSeries サーバー上に保管されたデータを見ることになります。カスタマーが提供する名前や電話番号などの情報もホストに保管されます。

注: 別の例であるビデオ・ストア・キャッシャーは、ホスト上の同一のデータベースにアクセスすることができます。これは、カスタマーの注文からの情報を使用してビデオ・ストアの在庫を更新して、カスタマーに請求します。この項ではビデオ・ストア・キャッシャーについては説明しません。そのビルドと実行に関する情報については、ビデオ・ストア・キャッシャー・プロジェクト・フォルダー内の CASHIER.VPG ファイルのコメントを参照してください。

サンプル・アプリケーションのビルド

ビデオ・ストア・カタログ・アプリケーションでは、F 仕様で定義されたファイルが iSeries サーバー上にあることが必要です。*WDSC\samples\vidcust* サブディレクトリーには、必要なファイルをもつライブラリーの保管ファイル *VIDEOSTORE.sav* が入っています。

必要なファイルをアップロードして、アプリケーションとその関連プレビュー・コンポーネントをビルドするためには、次のステップに従ってください:

1. 必ず、VisualAge RPG と VisualAge RPG サンプルの両方ともインストールする。
2. 次のように、VIDEOSTORE.sav ファイルを iSeries サーバーにアップロードして復元する:
   a. iSeries サーバーで、任意のライブラリー (たとえば USER) に VIDEOSTORE という名前の保管ファイルを作成します。

b. ワークステーションで、現行ディレクトリーを *vidcust* に変更して、次のように ftp コマンドを出します:

```
x:\...\WDSC\samples\vidcust>ftp HOSTNAME
```

c: は製品をインストールしたドライブであり、**HOSTNAME** は保管ファイルを作成した iSeries サーバーの名前です。(代わりにサーバーの TCP/IP アドレスを使用することができます。)

c. プロンプトが出されたらユーザー ID を入力します。
d. プロンプトが出されたら自分のパスワードを入力します。
e. サインオンした後で、USER ライブラリーを現行ライブラリーにします。次のように入力します:

```
ftp>cd user
```

f. ファイル転送を 2 進数として指定します。次のように入力します:

```
ftp>binary
```

200 表記タイプは 2 進数 IMAGE です というメッセージが表示されます。

g. 保管ファイルを転送します。次のように入力します:

```
ftp>put VIDEOSTORE.sav
```

ファイル転送が正常に完了しました。 というメッセージは、VIDEOSTORE.sav がアップロードされたことを示します。

h. 次のように入力します: `ftp>quit`

これで VIDEOSTORE 保管ファイルが iSeries サーバーのライブラリー USER の中にあるはずです。

i. RSTLIB を使用してライブラリーを復元し、同じ名前 (VIDEOSTORE) を iSeries サーバーに保存します。あるいは、Catalog.rst ファイルの中の *REMOTE_FILE_NAME* パラメーターを、保管ファイルを復元した名前に変更します。

j. Catalog.rst ファイルで、*REMOTE_LOCATION_NAME* パラメーターを変更して iSeries サーバーのリモート・ロケーション名をポイントするようにします。

3. ビデオ・ストア・カタログのサンプル・アプリケーションを作成する。VisualAge RPG サンプル・アプリケーション・フォルダーの中のビデオ・ストア・カタログ・プロジェクト・フォルダーのポップアップ・メニューから **ビルド > Windows** または**ビルド > Java** を選択してください。プロジェクトが正常に作成されたら、Windows の実行可能プログラム CATALOG.EXE または Java の CATLAOG.CLASS がビデオ・ストア・カタログ・プロジェクト・フォルダーに作成されます。
4. 関連のプレビュー・コンポーネントを作成する。プレビュー・プロジェクト・フォルダーのポップアップ・メニューから **ビルド > Windows** または**ビルド > Java** を選択してください。プロジェクトが正常に作成されたら、Windows のダイナミック・リンク・ライブラリー (COMMON.DLL) または Java の COMMON.CLASS ファイルが作成されます。

ビデオ・ストア・カタログ・アプリケーションを実行するためには、次の方法の 1 つを使用します。

- ビデオ・ストア・カタログ・プロジェクト・フォルダーのポップアップ・メニューから **実行 > Windows** または **実行 > Java** を選択する。

- ビデオ・ストア・カタログ・プロジェクト・フォルダーをオープンして、CATALOG.EXE アイコンをダブルクリックする。
- コマンド行に `catalog` と入力する。

注:

1. このアプリケーションのマルチメディアの局面には追加のハードウェアとソフトウェアが必要です。プレビューでオーディオを実行するためには、システムにサウンド・カードが必要です。ビデオ・クリップをプレビューで実行するには、メディア・プレイヤーがインストールされていなければなりません。Java アプリケーションには、Java メディア・フレームワーク (JMF) API が必要です。
2. ビデオ・クリップは、プレビュー・フォルダーに保管されている .AVI ファイル (Windows の場合) または .MOV ファイル (Java の場合) です。
3. Java:アプリケーションを作成して実行するには、Sun の Java 2 Software Development Kit (J2SDK) Version 1.2、またはそれ以降がワークステーションにインストールされていなければなりません。 J2SDK がない場合には、次の URL で Sun Microsystems からダウンロードできます。

   `http://java.sun.com/products/`

   J2SDK をダウンロードした後で、Java コンパイラーおよび Java Runtime Environment (JRE) の両方の場所をポイントするように、PATH 環境変数を設定してください。たとえば、J2SDK のホーム・ディレクトリーが *c:\jdk1.2* の場合には、次の path ステートメントを追加してください: `c:\jdk1.2\bin`

ユーザーへの表示内容の決定

ユーザー・アプリケーションを作成する場合に重要なステップは、ユーザーがそのアプリケーションによって何をできるようにしたいかを決定することと、ユーザーがそれを実行できるように何を提供する必要があるかを決定することです。

ここでは、ビデオ・ストア・アプリケーションの計画段階で、カスタマーが特定のカテゴリー（アクション/アドベンチャーまたはコメディーなど）のビデオをリストすることができるようにすることに決めています。また、好きなディレクターが制作しているか、好きな俳優が出演しているか、あるいはストアで上位 10 位までの売れ筋に入っているビデオもリストすることができるようにします。特定のビデオを購入するかどうかの決定に役立つように、そのビデオをプレビューすることができるようにします。購入したいビデオが見つかった後で、それを注文してから、キャッシャー・カウンターでその購入代金を支払うことができます。

これで、カスタマーがこのアプリケーションによって何を実行できるかが項目分けされているので、ビデオ・カタログを表示する時に何が表示されることになるかを設計することができます。これは、アプリケーション内のウィンドウの内容、数、および順序を設計する際の手始めです。

ビデオ・ストア・カタログへようこそ!

メイン・ウィンドウまたはアプリケーションへの入り口点は、「ビデオ・カタログ — ウェルカム」ウィンドウです。これは、カスタマーがカタログを用いて何を実行

できるかの段階をセットします。カタログを使用するためには、カスタマーはグラフィック・プッシュボタンを押して、次の選択項目から選択しなければなりません。

カテゴリー別の参照...

新規リリース...

上位 10 ベスト・セラー...

特定のタイトルの検索...

ヘルプ・カタログ

ヘルプ・カタログを選択すると、*カタログ使用時のヘルプの表示* ウィンドウが表示されます。その他のプッシュボタンの 1 つが押された場合には、別のウィンドウが表示され、そこからその他のアクション（リストの表示、クリップのプレビュー、または購入注文の実行依頼など）を実行することができます。

カテゴリー別の参照

カテゴリー別の参照を選択すると、「ビデオ・カタログ — カテゴリー」ウィンドウが表示されます。このウィンドウには、次のような選択用のビデオ・カテゴリーのリストが表示されます:

アクション/アドベンチャー	ホラー
子供向け	ウエスタン
サイエンス・フィクション	ロマンス
コメディー	クラシック

カテゴリーを選択するためには、カスタマーはそれと対応しているプッシュボタンを押します。これで、そのカテゴリーに関係した項目がリストされているビデオ・タイトル・ウィンドウが表示されます。カスタマーは、タイトルのいくつかをプレビューし、あるタイトルを自分の注文に追加し、気が変わった場合はそれを注文から削除し、さらに注文をキャッシャーに実行依頼することができます。

特定のタイトルの検索

「ビデオ・カタログ — 検索」ウィンドウによって、お客様はビデオをカテゴリー、タイトル、ディレクター、または俳優別に検索することができます。検索基準を指定して、データベースの検索を開始するための**「検索」**プッシュボタンを押した後で、その結果が「ビデオ・タイトル」ウィンドウに表示されます。

タイトルのプレビュー

カスタマーは、リスト上のビデオの概要を読むか、あるいは適切なハードウェアおよびおよびソフトウェアがある場合はそのクリップをオーディオ付きで表示することによって、そのビデオをプレビューすることができます。

注文の変更および実行依頼

「ビデオ・カタログ — 選択内容の検討/注文」ウィンドウによって、カスタマーは自分の注文を変更することができます。カスタマーは、ビデオをリストから削除し、購入したいコピー数を変更し、さらにビデオのメディア・タイプ (テープまたはレーザー・ディスク) を変更することができます。このウィンドウは、カスタマ

ーがビデオ・タイトル・ウィンドウ内のリストから ビデオを選択してから、**「注文の検討/実行依頼」**プッシュボタンを押した時に表示されます。

自分の注文に関する情報をすべて入力した時に、カスタマーは自分の注文をこのウィンドウから実行依頼します。

注文の実行依頼

カスタマーが自分の注文を実行依頼する場合には、「ビデオ・カタログ — 注文参照」ウィンドウに名前、住所、および電話番号を入力しなければなりません。この情報は iSeries 400 サーバーのデータベースに保管されます。

高水準ウィンドウ設計

ビデオ・ストア・カタログのウィンドウのすべてを作成する方法について説明することは、この項の範囲を超えています。次の項では、GUI Designer を使用して、コメディーおよびプレビュー・ウィンドウを例にして 2 つのウィンドウを作成し、そのパーツ属性の一部を変更し、さらに関連のプログラム・ロジックをいくらか作成する方法について説明します。これらのウィンドウは、ビデオ・ストア・カタログ・アプリケーションで次の経路をたどることによって見つかります。

ビデオ・カタログ — ウェルカム 	「ビデオ・カタログ — ようこそ!」ウィンドウの**「カテゴリー別の検索」**プッシュボタンを押すと、「ビデオ・カタログ — カテゴリー」ウィンドウが表示されます。
ビデオ・カタログ — カテゴリー 	「ビデオ・カタログ — カテゴリー」 ウィンドウの**「コメディー」**プッシュボタンを押すと、「ビデオ・カタログ — コメディー」ウィンドウが表示されます。
ビデオ・カタログ — コメディー 	「ビデオ・カタログ — コメディー」ウィンドウでタイトルを選択した後で**プレビュー**・ボタンを押すと、「プレビュー」ウィンドウが表示されます。
ビデオ・カタログ — プレビュー 	プレビューはこのウィンドウで実行されます。

サンプル・アプリケーションの設計を見たい場合は、VisualAge RPG サンプル・アプリケーション・フォルダーの中のビデオ・ストア・カタログ・プロジェクト・フォルダーのポップアップ・メニューから**編集**を選択してください。これによって、アプリケーションのプロジェクト・ウィンドウおよびパーツ・パレットが表示されます。プロジェクト・ウィンドウは、アプリケーション用に定義されたすべてのウ

ィンドウを表示します。項目をダブルクリックして、関連パーツとともにその設計ウィンドウを表示します。プロジェクトの VARPG ソース・コードを表示するには、プロジェクト・ウィンドウから**プロジェクト > ソース・コードの編集**を選択してください。

コメディー・ウィンドウの作成

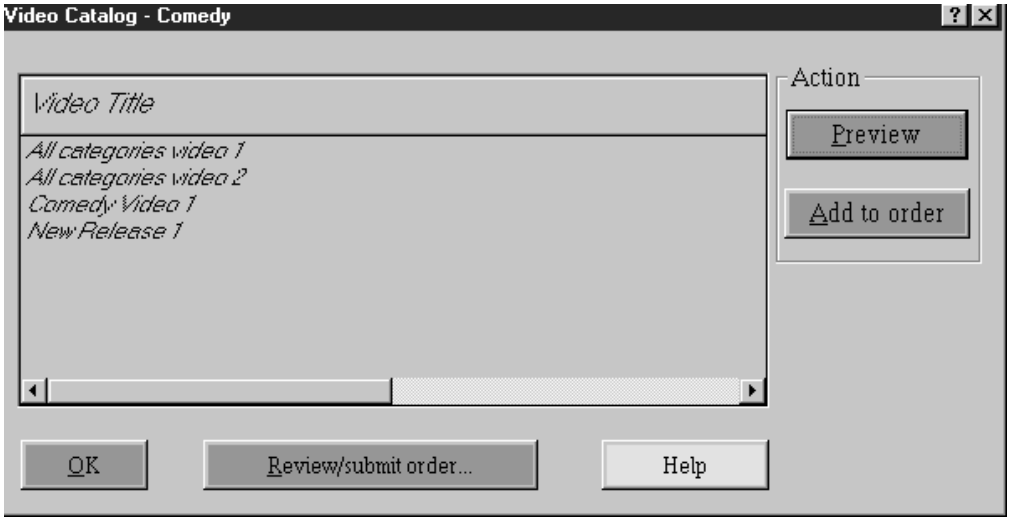


図 1. コメディー・ウィンドウ

コメディー・ウィンドウには、カスタマーが購入できるコメディー・ビデオのリストが表示されます。この項では、これに類似しているウィンドウを作成する方法について説明します。

GUI の作成

右マウス・ボタンを使用して、パーツ・パレットからキャンバス付きウィンドウを選択して、ポインター・アイコンを GUI Designer のプロジェクト・ビューに移動してから、再び右クリックします。これが設計ウィンドウになり、パレットから次のパーツを入れます:グループ・ボックス、プッシュボタン、静的テキスト、およびサブファイル。

パーツの位置合せ

GUI Designer で位置合せツールを使用して、パーツが図 1に示されているようになるようにサイズ変更、位置合せ、およびスペーシングを行なうことができます。これらのツールの使用法については、オンライン・ヘルプまたは HTML チュートリアルを参照してください。

属性の設定

パーツをウィンドウに入れて位置決めした後で、そのプロパティー・ノートブックを使用してパーツ属性のデフォルト設定を変更することができます。これを実行するためには、パーツを右クリックしてから、パーツのポップアップ・メニューから**プロパティー**を選択してください。

ユーザーが変更できるパーツ属性のいくつかについては以下で説明します。

ウィンドウ属性

ウィンドウに表示したい項目 (システム・メニュー、タイトル・バー、および最小化ボタンと最大化ボタンなど) を選択して、ウィンドウの枠を構成することができます。デフォルトによって、このウィンドウはシステム・フォントを使用し、背景は白になります。フォントとカラーは変更することができます。

キャンバス属性

デフォルトによって、キャンバス・パーツはシステム・フォントを使用し、フォルダーの背景と同じカラーになります。キャンバス・パーツのフォントと背景カラーは変更することができます。また、キャンバス・パーツにグラフィックを配置することもできます。

サブファイル属性

デフォルトによって、サブファイル・パーツは欄なしで作成されます。データベース・フィールド名がわかっている場合には、GUI Designer を使用してサブファイル入力フィールドを作成することができます。そうでない場合には、次のステップに従うことによって、データベース中の既存のフィールドを参照することができます。

1. **サーバー**・メニューから**参照フィールドの定義**を選択します。参照フィールドの定義ウィンドウが表示されます。
2. iSeries 400 サーバーおよびライブラリー情報を指定してデータベース・フィールド情報を表示します。
3. 右マウス・ボタンで**フィールド**・リスト・ボックスから該当のフィールドを選択し、ポインター・アイコンを設計ウィンドウのサブファイル・パーツに移動してから、再び右クリックします。

   新規サブファイル入力フィールドは、元のフィールドから次の属性を継承します: **長さ**はカラム幅にセットされ、**タイプ** はデータ・タイプにセットされます。

該当のプロパティー・ノートブックを使用してサブファイル入力フィールドのスタイルおよびデータ・タイプをセットします。たとえば、長さまたはデータのタイプをセットすることができます。

プッシュボタン属性

プッシュボタンのそれぞれにラベルを付けてその目的をユーザーに指示します。各プッシュボタンの簡略記号を作成するためには、ラベルの文字の前に簡略記号の ID を入れます。 Windows では、アンパーサンド (&) を使用してください。ユーザーが自分の注文を入れるボタンを押した後でさらに情報を入力しなければならないことがわかるように、**「注文の検討/実行依頼」** プッシュボタンのラベルに省略記号 (...) が入れられている点に注意してください。

それぞれのプッシュボタンごとに、ユーザーがそのプッシュボタンを押した時に行なわれるアクションの内容を指定します。たとえば、**「プレビュー」**プッシュボタンの場合には、アクション・ サブルーチンが実行されることになります。**「ヘルプ」**プッシュボタン の場合には、そのウィンドウのヘルプが表示されることになります。この情報は、プッシュボタンのプロパティー・ノートブックのアクション・

タブで設定することができます。 (関連情報については、 17 ページの『オンライン・ヘルプの作成』を参照してください。)

プログラム・ロジックの追加

ある種の GUI 機能を駆動するためにはプログラム・ロジックが必要です。この項では、ビデオ・ストア・カタログのプログラム・ロジックのいくつかについて説明します。 (ソース・ファイル CATALOG.VPG はビデオ・ストア・カタログ・フォルダーに入っています。)

注: 特定のイベント用のプログラム・ロジックは、GUI Designer から編集セッションを起動することによって入力することができます。たとえば、プログラム・ロジックを特定のプッシュボタンの **押す** イベントに追加するためには、そのプッシュボタンのポップアップ・メニューから**イベント > 押す**を選択します。

コメディー・ウィンドウの表示

ユーザーが「ビデオ・タイトル — カテゴリー」ウィンドウで **「コメディー」**プッシュボタンを押した時に、コメディー・ウィンドウが表示されるようにするためには、次を実行します。

1. **「コメディー」**プッシュボタンの **Press** イベントを処理するアクション・サブルーチンを作成します。

   ここでは、COMEDYGPB アクション・サブルーチン (図 2 を参照) を作成して、このイベントを処理します。ユーザーがこのプッシュボタンを押した時に、COMEDYGPB サブルーチンが **brComedy** ユーザー・サブルーチンを呼び出します。このサブルーチンはデータベースを読み取り、もう 1 つのユーザー・サブルーチン **dspbrowse** を呼び出して、データベースが空になっているかどうかを調べます。空になっている場合には、メッセージが表示されます。空になっていない場合には、制御を brComedy ユーザー・サブルーチンに戻し、ウィンドウのタイトルが変更され、データベース検索の結果が表示されます。

```
**********************************************************************
**                                                                  **
**  Categories window action-link subroutines                       **
**                                                                  **
**                                                                  **
**********************************************************************
*
* This routine is executed when the Comedy graphic push button in the
* Categories window is pressed.
*
C     COMEDYGPB     BEGACT    PRESS         CATW
C                   z-add     0             srchdir
C                   z-add     0             srchact
C                   exsr      brComedy
C                   ENDACT
```

図 2. PRESS イベントの処理

2. データベースからコメディー・ビデオ・タイトルを読み取り、タイトルのリストとともにサブファイル・パーツを移植するプログラム・ロジックを作成します。dspbrowse サブルーチンを呼び出して、データベースが空かどうかを調べます。データベースが空でない場合には、ブラウズ・ウィンドウのタイトルをセットして見つかったコメディー・タイトルを表示します。そうでない場合には、メッセ

ージ番号 MSG0001 を表示して、データベースに一致するタイトルが見つからなかったことをユーザーに通知します。 図 3を参照してください。

```
     ****************************************************************
     *                                                              *
     * User Subroutine: brComedy                                    *
     * Description    : Show browse window with comedy videos       *
     *                                                              *
     ****************************************************************

    C     brComedy      BEGSR
    C                   clear                   browsesf
     * Get records from vil0004, the logical file on the AS/400
     * for comedy type videos.
    C     *start        setll     vil0004
    C                   read      vil0004                           61
    C     *IN61         doweq     '0'
    C                   exsr      ckcriteria
    C                   read      vil0004                           61
    C                   end
    C                   exsr      dspbrowse
     * The next three lines set the browse window's title bar text.
    C                   movel     *blanks       vdocatstl
    C                   movel     stlcmdy       vdocatstl
    C                   eval      %setatr('browsew':'browsew':'Label') =
    C                               vdocatttl
    C                   ENDSR

⋮

     ****************************************************************
     *                                                              *
     * User Subroutine: dspbrowse                                   *
     * Description    : Check if the browse subfile is empty.  If so,    *
     *           display message MSG0001 saying match not found.    *
     *                                                              *
     ****************************************************************

    C     dspbrowse     BEGSR
    C                   eval      items=%getatr('BROWSEW':'BROWSESF':'Count')
    C     items         ifeq      0
    C     *MSG0001      dsply                   msgrsp            9 0
    C                   else
    C                   eval      %setatr('BROWSEW': 'BROWSEW': 'VISIBLE')=1
    C                   eval      %setatr('BROWSEW': 'BROWSEW': 'FOCUS')=1
    C                   endif
    C                   ENDSR
```

図 3. iSeries データベースの読み取りおよび結果ウィンドウの表示

プレビュー・ウィンドウの表示

ここでは、アクション・サブルーチン (12 ページの図 4を参照) を作成して、コメディー・ウィンドウの**「プレビュー」**プッシュボタン の PRESS イベントを処理します。ユーザーがこのボタンを押した時に、PREVIEWPB アクション・サブルーチンが呼び出されて、プレビュー・ウィンドウを表示する共通のコンポーネントが開始されます。

```
*****************************************************************
* When the preview button in the browse window is pressed, the common
* component is started.  The common component displays the preview
* window of a video.
*
*
C     PREVIEWPB     BEGACT    PRESS         BROWSEW
C                   READS     BROWSESF                              55
C     *IN55         ifeq      '0'
C                   start     'common'
C                   parm                    brsfpart
C                   endif
C                   ENDACT
```

図4. プレビュー・ウィンドウを表示するためのアクション・サブルーチン

プレビュー・ウィンドウの作成

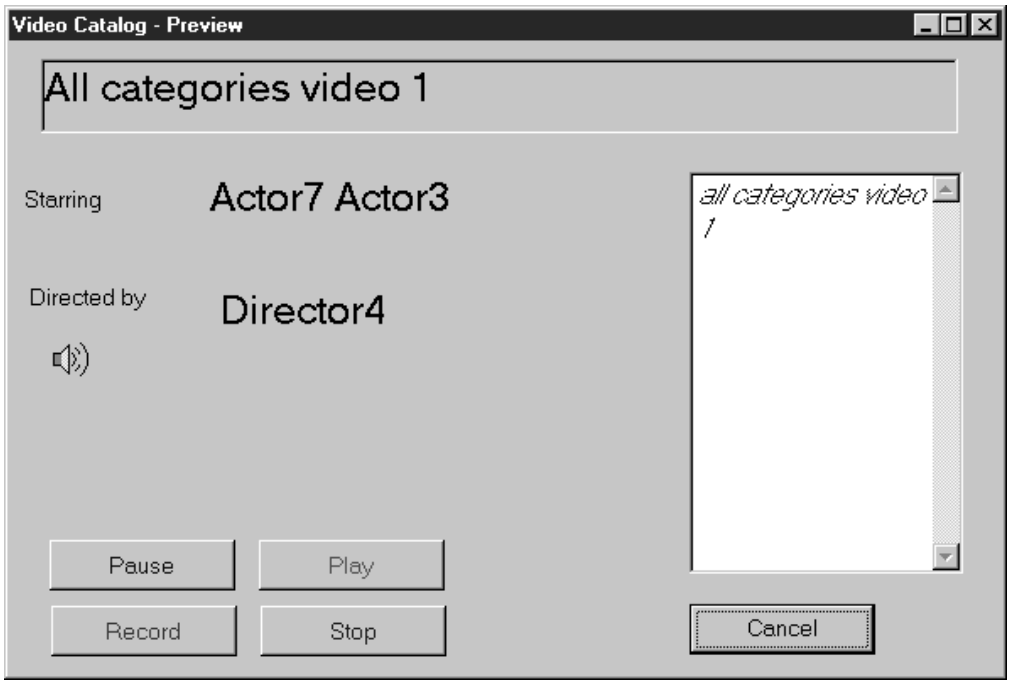


図5. プレビュー・ウィンドウ

オペレーティング・システムのマルチメディア機能を使用したプレビュー・ウィンドウで、ユーザーはビデオ・クリップを一目だけ見ることができます。この項では、上記のようなウィンドウを作成する方法について説明します。

注: プレビューでオーディオを実行するためには、システムにサウンド・カードが必要です。ビデオ・クリップを実行するには、メディア・プレイヤーがインストールされていなければなりません。Java アプリケーションには、Java メディア・フレームワーク (JMF) API が必要です。

GUI の作成

次のパーツをキャンバス・パーツのあるウィンドウにポイント・アンド・クリックして、プレビュー・ウィンドウに類似したウィンドウを作成します:

- メディア・パーツ
- 複数行編集パーツ
- プッシュボタン・パーツ
- 静的テキスト・パーツ

設計時の属性の設定

パーツをウィンドウに入れて位置決めした後で、それと対応するプロパティー・ノートブックを使用してそのパーツの属性をセットすることができます。ユーザーがセットできるパーツ属性のいくつかについては以下で説明します。

メディア・パーツ属性

プッシュボタン・パーツは、ビデオ・クリップの再生を制御するために使用されます: **再生**、**一時停止**、**録画**、および**停止**。**AudioMode** 属性はメディア・パーツの操作モードを設定します。

静的テキスト属性

静的テキスト・パーツのフォント属性を変更して、画面上のその他のテキストより目立つようにすることができます。静的テキスト・パーツが最長のテキストを保持するのに十分な長さになるようにサイズ変更します。 (将来、ユーザーのアプリケーションが変換される場合には、余分なスペースを残しておくのも一案です。)

複数行編集パーツ

コード中で、テキストを受け入れるために、ABSTMLE と呼ばれる複数行編集 (MLE) パーツをセットしています。パーツのプロパティー・ノートブックで、パーツが読み取り専用であることを指示してあります。

実行時の属性の設定

実行時に表示されるウィンドウのタイトルを変更するために、SETATR 命令コードを使用して、**Label** 属性 (15 ページの図 6を参照)をセットします。

プログラム・ロジックの追加

プレビュー・ウィンドウである種の GUI 機能を駆動するためには、プログラム・ロジックを提供しなければなりません。この項では、プレビュー・コンポーネント用のプログラム・ロジックをいくつか (15 ページの図 6を参照) 説明します。

プレビューするビデオの指定

コメディー・ウィンドウで選択されるビデオによって、どのビデオ・プレビューを再生するかが決まります。 **previeww** アクション・サブルーチンが、どのビデオを使用するかを読み取ってから、実際のビデオ・ファイルのファイル名を判別します。

ビデオの制御

メディア・パーツを使用して、ビデオを制御するコードを作成することができます。例では、メディア・パーツは、選択されたビデオと関連するディジタル・ビデオ・ファイルを再生するために使用されています。

関連の AudioMode 属性をもつプッシュボタンは、ビデオ・ファイルの再生を制御します:

1 休止

2 再生

3 レコード

4 停止

プレビュー・コンポーネントのコードは次の通りです:

```
 ****************************************************************************
 *
Fvideo     if   e           k disk    remote BLOCK(*YES)
 *
DFlg             s              1    inz(*OFF)
DFldx            s             12
 *
 ****************************************************************************
 *
C     *entry        PLIST
C                   parm                    partno            5 0
 *
 ****************************************************************************
 * Action link subroutines for PREVIEWW                                     *
 ****************************************************************************
 *
C     PREVIEWW      BEGACT    CREATE        PREVIEWW
C     partno        setll     video                                       50
C  N50*msg0001      DSPLY                   msgrsp            9 0
C                   read      video                                       51
C     *IN51         IFEQ      '0'
C     'TITLEST'     SETATR    vititle       'label'
C     'DIRST'       SETATR    vidirect      'label'
 *
C     viactr1       CAT       viactr2:1     actors           41
C     'ACTST'       SETATR    actors        'label'
 *
C     'ABSTMLE'     SETATR    vireview      'text'
 *
 *  If its for Java, use .mov file
 /If defined(COMPILE_JAVA)
 *
C     vibitmap      CAT       '.mov':0      videofil         13
 *  If its not for Java, then use .avi file
 /else
C     vibitmap      CAT       '.avi':0      videofil         13
 /EndIf
C                   endif
 *signify videofil is not yet loaded to Audio part
C                   move      'N'           loaded
 *
C                   ENDACT
 *                                                                         *
```

図 6. プレビュー・コンポーネント (1/3)

```
 ********************************************************************
 *
C     PBPLAY        BEGACT    PRESS         PREVIEWW
 *
C                   if        loaded='N'
C                   eval      %setatr('previeww':'audo':'FileName')
C                              =videofil
C                   move      'Y'           loaded            1
C                   endif
 *
C                   eval      %setatr('previeww':'audo':'audioMode')=2
 *
C                   ENDACT
 *                                                                  *
 ********************************************************************
 *
C     PBPAUSE       BEGACT    PRESS         PREVIEWW
 *
C                   eval      %setatr('previeww':'audo':'audioMode')=1
C                   ENDACT
 *                                                                  *
 ********************************************************************
 *
C     PBRECORD      BEGACT    PRESS         PREVIEWW
 *
C                   eval      %setatr('previeww':'audo':'audioMode')=3
C                   ENDACT

 *                                                                  *
 ********************************************************************
 *
C     PBSTOP        BEGACT    PRESS         PREVIEWW
 *
C                   eval      %setatr('previeww':'audo':'audioMode')=4
C                   ENDACT
 *                                                                  *
```

図 6. プレビュー・コンポーネント (2/3)

```
 ********************************************************************
 *
C     CANCELPB      BEGACT    PRESS         PREVIEWW
 *
C                   move      *on           Flg
C                   STOP
C                   ENDACT
 *                                                                  *
 ********************************************************************
 *
C     PREVIEWW      BEGACT    CLOSE         PREVIEWW
 *
C                   if        Flg=*ON
C                   eval      Fldx='*DEFAULT'
C                   else
C                   eval      Fldx='*NODEFAULT'
C                   endif
C                   ENDACT    Fldx
 *
```

図 6. プレビュー・コンポーネント (3/3)

メッセージの作成

メッセージを追加するためには、GUI Designer から**プロジェクト > メッセージの定義**を選択します。メッセージの定義ウィンドウが現れます。**作成**を選択してから、作成したいメッセージのタイプ (たとえば、通知または警告) を選択します。メッセージの実際のテキストを入力して、追加情報または 2 次レベル・ヘルプがあればこれを入力します。

VisualAge RPG はユーザーが作成するメッセージのメッセージ ID を自動的に生成します。ユーザーのコードではそのメッセージ ID を参照します。たとえば、 11 ページの図 3では、DSPLY 命令コードによって MSG0001 が使用されます。

オンライン・ヘルプの作成

ここでは、ビデオ・ストア・カタログ・アプリケーションに異なるタイプのヘルプを追加しています。次の項では、このヘルプの一部を複製する方法について説明します。

ヘルプ情報

ヘルプ情報は、パーツのポップアップ・メニューから**「ヘルプ・テキスト」**を選択することによって、**「カテゴリーによるブラウズ」**グラフィック・プッシュボタン・パーツに追加します。これで、すでに図 7に示されているような情報が入っている編集セッションが開始されます。

```
:h1 res=01.PSB0000C:p.Help
```

図 7. オンライン・ヘルプを追加するための編集セッション

`:h1 res=01.` はリソース ID を含む見出しタグです。リソース ID は自動的に生成されます — このテキストは編集**しないで**ください。見出しはこのタグの直後に現れます。これはヘルプ・パネルで使用され、実行時にヘルプ索引にリストされます。デフォルトによって、テキストを追加するパーツの名前が見出しとして使用されます。それを、ヘルプ・パネルの目的を識別し、ユーザーに分かりやすい見出しによって置き換える必要があります。実際のヘルプ・テキストは `:p.` タグの後に入力します。デフォルトによって、編集セッションに **Help** という語が表示されます。

ビデオ・ストア・カタログ・アプリケーションからのヘルプ・テキストの例は 図 8 に示されています。そのソースから生成されて、実行時に表示されるヘルプ・パネルは 18 ページの図 9に示されています。

```
:h1 res=12.カテゴリー別の参照
:p.ビデオをカテゴリーによってブラウズするためにはこれを押してください。
```

図 8. カテゴリー・グラフィック・プッシュボタンによるブラウズのヘルプ

図 9. コンテキストに依存したオンライン・ヘルプ・パネルの例

「ヘルプ」プッシュボタンの作成

「プレビュー」ウィンドウまたは「コメディー」ウィンドウの下部にヘルプ・グラフィック・プッシュボタンを作成するためには、右マウス・ボタンでパーツ・パレットからグラフィック・プッシュボタンを選択し、ポインター・アイコンを設計ウィンドウに移動してから、再び右クリックしてください。プロパティー・ノートブックで、グラフィック・プッシュボタンに表示するイメージを指定して、 **Press** イベントの発生時にヘルプを表示したいことを指定します。

19 ページの図 10は、ウェルカム・ウィンドウにヘルプを提供するプッシュボタンのソースを示しています。 `:link.` タグは、ユーザーが適切な情報を迅速かつ容易に見つけることができるように、ヘルプ情報の関連のある断片をリンクするために使用されます。このタグは別のパネル内のヘルプ・テキストと関連しているテキストの周囲に配置します。 `:link.` タグと `:elink.` タグの間のテキストはアプリケーションの実行時に強調表示されます (19 ページの図 11を参照)。強調表示されたテキストを選択することによって、ユーザーは関連したターゲット・ヘルプ・パネルにジャンプします。ターゲット・パネルのリソース ID (resid) は link タグのパラメーターです。

```
:h1 res=22.カタログ使用法のヘルプの表示
:p.グラフィック・プッシュボタンの 1 つを選択してください。
:p.
:link reftype=hd res=12.カテゴリーによるブラウズ
:elink.カテゴリーによってブラウズするにはこのボタンを押してください。
:p.
:link reftype=hd res=19.新規リリース
:elink.新規ビデオ・リリースを表示するにはこのボタンを押してください。
:p.
:link reftype=hd res=20.上位 10 位売れ筋
:elink.上位 10 位を表示するにはこのボタンを押してください。
:p.
:link reftype=hd res=21.特定タイトルの検索
:elink.特定のタイトルを検索するにはこのボタンを押してください。
```

図 10. ウェルカム・ウィンドウのヘルプ

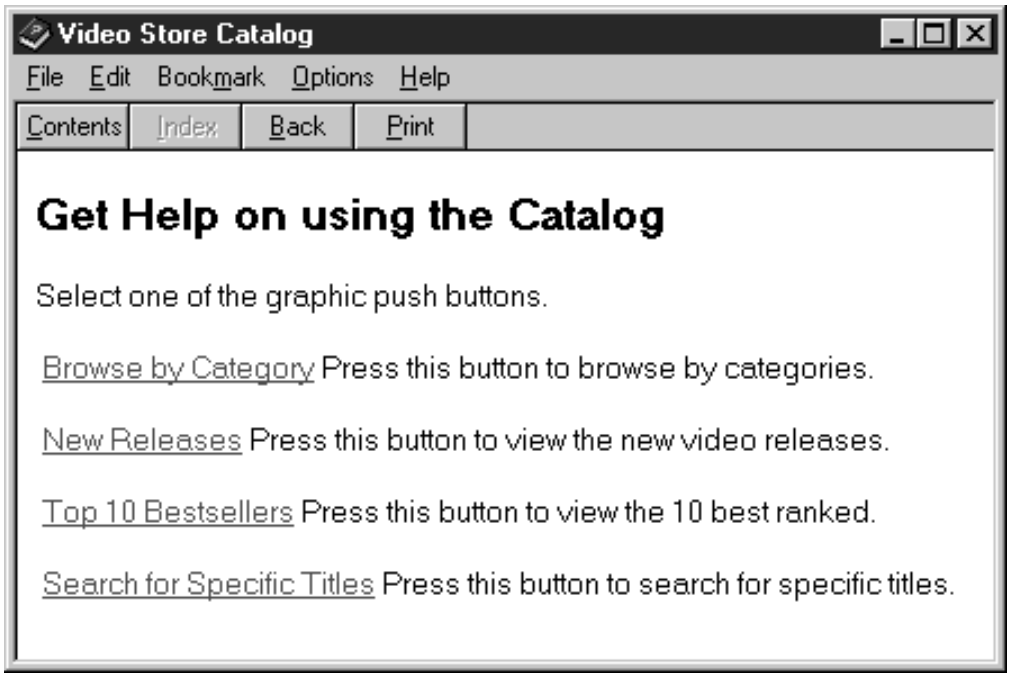


図 11. ハイパーテキスト・リンクを含むウィンドウのヘルプの例

アプリケーションのオンライン・ヘルプを作成する場合の詳細については、以下のトピックを参照してください:

- 257 ページの『第 13 章 IPF を用いたオンライン・ヘルプ作成のヒント』
- 261 ページの『第 14 章 Windows ヘルプの作成および使用のヒント』
- 267 ページの『第 15 章 JavaHelp 作成のヒント』

ビジュアル・プログラミングの検討

前項で説明したステップは、VisualAge RPG を使用してユーザー独自のアプリケーションを作成する場合に行なうステップに類似しています。これらのステップは次の通りです。

1. ユーザーに何を表示するかの決定

新規アプリケーションの作成を開始する前に、ユーザー・アプリケーションの目的、ユーザーへの表示方法、およびその他アプリケーションとの連絡方法を選択する必要があります。

2. GUI Designer を使用した GUI の作成

   アプリケーションを設計した後で、GUI Designer を使用してグラフィカル・ユーザー・インターフェースを作成することができます。 VisualAge RPG は、GUI パーツを選択するためのカタログをユーザーに提供し、ユーザーのニーズに合うユーザー定義のパーツを作成する機能を提供します。インターフェースに表示したいパー値を選択することができ、設計ウィンドウ上のそれらの位置を選択することができます。パーツは必要に応じてカスタマイズします。

   ウィンドウの作成、ウィンドウへのパーツの追加、およびパーツの位置合せとカスタマイズについては、オンライン・ヘルプを参照してください。

3. 属性の取り出しおよび設定

   一部のパーツ属性は、設計時にパーツのプロパティー・ノートブックを使用してセットすることができます。また、命令コード GETATR および SETATR、または組み込み関数 %getatr および %setatr を使用して、実行時にパーツの属性を取り出したりセットしたりすることもできます。パーツ属性の取り出し時または設定時に、パーツは GUI Designer でそれに定義された名前を使用して参照します。

   パーツの詳細およびパーツ属性の取り出しおよび設定の方法に関する詳細については、*ADTS/CS VisualAge for RPG パーツ解説書* を参照してください。

4. プログラム・ロジックの作成

   各パーツは一組の定義済みイベントに応答します。一般に、イベントは、あるユーザーが GUI と対話した結果として生成されます。たとえば、プッシュボタンを選択すると、**Press** イベントのシグナルが送られます。また、イベントはユーザー・プログラムによって生成することもできます。たとえば、DDE クライアント・パーツは、事前に決められた時間以内にサーバー・プログラムとの会話を開始することができない場合に、 **Timeout** イベントを生成します。

   イベントには、ユーザー・プログラムで命令コード BEGACT (begin action) と ENDACT (end action) をコーディングすることによって応答します。これらの命令コードの間のコードは、アクション・サブルーチンと呼ばれ、特定のイベントに対して実行されます。アクション・サブルーチンをコーディングしない場合には、イベントの発生時にアクションは実行されません。

5. メッセージおよびオンライン・ヘルプの追加

   GUI の作成およびそれを実行させるプログラム・ロジックの作成の他に、ユーザー・アプリケーションにメッセージとオンライン・ヘルプを追加することができます。

第 2 章 アプリケーションの計画

この項では、新しいアプリケーションのコーディングを開始したり、既存の OS/400* アプリケーションを VisualAge RPG アプリケーションに一時変更する前に行なう必要がある事項について説明します。

新しいアプリケーションを作成している場合は、その目的、ユーザーへの提供方法、および他のアプリケーションとの通信方法を決定する時点です。

既存の OS/400 アプリケーションの再利用を計画している場合は、古い文字画面表示を評価し、グラフィカル・ユーザー・インターフェース・パーツを使用してそれらを改善する方法を決定する時点です。(既存のアプリケーションの再利用の詳細については、 203 ページの『第 3 部 iSeries データの処理』を参照してください。)

この項の情報は、ユーザーのニーズに合った、実現が可能なアプリケーションの設計に役立ちます。

セキュア Java アプリケーションの使用可能化

Java アプリケーションを WWW で使用するように開発する場合には、OS/400、バージョン4、リリース 4、またはそれ以降で実行されるシステムだけが Secure Sockets Layer (SSL) 仕様をサポートすることに注意してください。古い OS/400 バージョンで実行されるワークステーション・アプリケーションとサーバーの間のデータ・フローは保護されません。

VisualAge RPG の SSL サポートのセットアップについては、 481 ページの『付録 D. Secure Sockets Layer (SSL) セットアップ』を参照してください。 VARPG アプリケーションが、クライアント・セキュリティー・ファイルを使用するアプレットを実行する場合は、 219 ページの『アプレット用のセキュリティー・ファイルの使用』 を参照してください。

提供する機能の決定

最初に、アプリケーションの主目的が何であるかを決定し、そのためにどのような機能を提供する必要があるかを決定します。中心となる機能を決定した後で、動的データ交換 (DDE) や印刷などの拡張機能に取り組みます。

ユーザーのヘルプ

ユーザーはさまざまな度合いの GUI 経験を持っています。通常のユーザーの知識レベルに合わせて調整したオンライン・ヘルプの提供を考慮してください。VisualAge RPG では、GUI にオンライン・ヘルプを追加するのは容易です。次の 4 つの種類のヘルプを追加することができます:

ヘルプ情報

現在の選択項目、オブジェクト、あるいは選択項目またはオブジェクトのグループに適応したヘルプ情報。

タスク・ヘルプ
ユーザーがアプリケーションで実行できるタスクについての情報。

ツール・ヒント・ヘルプ
ユーザーが使用できるツールについての吹き出しタイプのヘルプ。

ウィンドウ・レベルのヘルプ
ウィンドウの内容についての情報。

ユーザーを助けるもう 1 つの方法は、タスクの完了に必要な すべての情報を与え、意味のあるプロンプトや GUI パーツのラベルを提供する方法です。省略記号 (...) を使用して、特定のアクションを実行する前に、さらに情報が必要なことを示すことができます。(たとえば、**Display...** を使用して、表示アクションを実行する前にさらに情報を求めるプロンプトが出されることをユーザーに伝えます)。ボタンを押した直後にアクションが実行される場合には、ラベルに省略記号を使用してはいけません。たとえば、ヘルプ情報はボタンを押すとすぐに表示されるので、**ヘルプ**・ボタンに省略記号は不要です。ヘルプは、マウス・ポインターがインターフェースの別のパーツの上に移動すると更新される静的テキストの形で提供することもできます。

ユーザーがデフォルト値を設定して提供する情報の量を最小にすることができます。たとえば、組み合わせボックス・パーツを使用して、ユーザーに通常使用される選択項目のリストから選択するオプションを提供することができます。これで、実行時のキー入力エラーが防止されます。

ウィンドウの設計を簡単にする

効果的なウィンドウの設計には次の 2 つの基本的な観点があります。
- アプリケーション内のウィンドウの数と構造
- それぞれのウィンドウの内容

ウィンドウの数

ユーザーがすべてのメインタスクを開始できる 1 つのメイン・ウィンドウを持つのが理想的です。ユーザーがタスクを完了するために使用しなければならない追加の情報用にセカンダリー・ウィンドウを提供します。

レイヤーが多過ぎると簡単なタスクも複雑に見えるので、多数のネストされるウィンドウは避けます。また、ウィンドウが多過ぎると、特にユーザーが複数のアプリケーションを実行している場合には、画面が煩雑になることも忘れないでください。画面上のウィンドウが多過ぎるとユーザーが迷子になる可能性があります。

それぞれのウィンドウのパーツの数が最小になるようにしてください。ウィンドウが表示される時のパフォーマンスが向上することになります。多くのウィンドウを持ち、ウィンドウ当りのパーツ数が少ないアプリケーションは、ウィンドウの数が少なくてウィンドウ当りのパーツ数が多い同じアプリケーションよりもパフォーマンスが優れています。

それぞれのウィンドウの内容

関連したすべての情報を一か所にまとめます。グループ・ボックス・パーツと外枠ボックス・パーツを使用して、どのラジオ・ボタンが関連しているかを目で見て分かるように示します。

グラフィック・イメージとアイコンを使用して、タスクを識別したりウィンドウの語句を補足します。すべてのタスクのつづりが正しいことを確認してください。

ウィンドウ内のパーツは、きちんとした、論理的な方法で配置します。パーツの位置は事前に定義されるので、一部のパーツの位置は計画する必要がありません。たとえば、メニュー・バー・パーツは常にウィンドウのタイトル・バーのすぐ下に置かれます。

ウィンドウに共通パーツがある場合には、それらのパーツは一貫した場所に表示してください。これで、ユーザーが共通情報を容易に見つけられるようになります。

コードを効果的に計画する

GUI を設計した後で、ユーザーが実行するアクションをサポートするためにどのコードが必要かを決定しなければなりません。VisualAge RPG は、たくさんのコードを書くことなしに GUI の作成を助けます — ユーザーに代わって日常的なタスクを実行します。すべてのパーツにはデフォルトの属性があり、これは GUI Designer またはユーザー・プログラムで変更することができます。ユーザーは他の属性を明示的に設定する必要があります。たとえば、ユーザーが **Display...** のラベルが付いたプッシュボタンを押してグラフィックスを表示できるようにしたい場合には、プッシュボタン・パーツをパレットから設計ウィンドウにポイントしてクリックする以上のことをしなければなりません: 少なくとも **Display...** を読み取るためのラベル属性を設定し、データを見つけて表示するロジックを書かなければなりません。

ユーザーに知らせておく

特に重要な情報や緊急の情報を提供するためにはメッセージを使用します。問題を説明する詳細で無駄のないメッセージを渡し、可能ならばその訂正方法を説明します。VisualAge RPG は 3 種類のメッセージの表示方法を提供するので、表示したい情報のタイプに最も適した方法を選択しなければなりません。

ウィンドウ
: ユーザーが知っている必要がある緊急情報（たとえば、正常に完了しなかったプロセス）を提供します。

メッセージ・サブファイル
: 選択項目についての情報を提供したり、アクションまたはプロセスの完了についてのメッセージを入れます。

2 次レベル・メッセージのヘルプ
: すべての時点ですべてのユーザーに必要ないかもしれない詳細レベルを提供します。たとえば、初心者ユーザーが知らないようなアクションの経過を説明します。

長いプロセスの場合には、ユーザーに知らせておくために進行状況表示を追加することができます。

例外を提示するためにユーザーにテキスト、視覚または聴覚による待ち行列を提供することを計画します。たとえば、プッシュボタンのテキストをグレー表示して、そのプッシュボタンが使用できないことを示すことができます。可能なすべてのユーザー・アクションを計画することは誰にもできないので、アプリケーションが解釈できないアクションについてユーザーに通知する方法も計画しなければなりません。たとえば、編集セッション中に行なった変更を保管しないでファイルを終了しようとした場合に、メッセージを表示することができます。

一貫性のあるスタイルの使用

混同を最小にするために一貫性のある用語を使用してください。たとえば、1 つのウィンドウでログインを使用する場合には、同じ概念を参照するためにどこかでユーザー ID を使用してはいけません。

アプリケーション・ウィンドウを通して一貫性のある簡略記号を使用してください。簡略記号は、選択項目を選択しあるいはアクションを実行するために押すことができる文字キーです。この文字キーは、プッシュボタンまたはメニューで下線づけされた選択項目の文字に対応しています。たとえば、1 つのウィンドウのプッシュボタンで保管機能を表すために**保管(S)** を使用する場合には、そのプッシュボタンを持つすべてのウィンドウでそれを使用してください。

変換の問題を予測する

現在の計画にアプリケーションを別の言語に変換することが含まれていない場合でも、将来容易に変換できるようにアプリケーションを設計し作成する必要があります。そうすることによって、変換の必要性が生じた場合に、手直しが少なくて済みます。

テキストとは別にコードを実行可能にすることを考慮してください。そのようにすれば、標準の実行可能なコードで適切な言語のテキストを使用することができます。

注: テキストをコードと別にすることを考慮しなければならない理由は他にもあります。テキストのエラーを訂正して、将来のリリースで用語をより容易に変更することができます。

それぞれの言語用に別のメッセージ・ファイルを作成して、それぞれに別のファイル拡張子を割り当てることができます。それぞれのメッセージ・ファイルには同一のメッセージ番号がなければなりませんが、テキストは別の言語で書くことができます。適切なメッセージ・ファイルを使用するだけで、すべての言語に対して 1 つの .EXE を作成することができます。たとえば、コンパイル済みのメッセージ・ファイルの英語バージョンは SAMPLE.ENG という名前にして、ドイツ語バージョンは SAMPLE.GER という名前にすることができます。アプリケーションを実行する前に、該当するメッセージ・ファイルを SAMPLE.MSG に名前変更するようにユーザーに指示することができます。詳細については、 273 ページの『第 16 章 メッセージの処理』 を参照してください。

また、変換によってテキスト (ラベルなど) 入力フィールド、バッファー、およびウィンドウのサイズ要件が変わる場合があることにも留意してください。置き換えラ

ベルを持つ GUI Designer のパーツをサイズ決めする場合には、変換されたテキストが元のテキストより長くなる場合があることに留意してください。

簡略記号を使用する場合には、言語が異なれば簡略記号文字が異なる場合があることに留意してください。

第 2 部 パーツの処理

第 3 章 パーツを用いたプログラミング

この項では、パーツを用いてプログラミングするためのヒントをいくつか提供します。これらのトピックには、パーツ属性の取り出しと設定、ユーザー・プログラム内でのパーツの参照、イベントおよびシステム属性への応答、イベントおよびシステム属性の処理、および静的テキストおよび入力フィールド・パーツのコーディングの方法が含まれています。

パーツ属性の取り出しおよび設定

一部のパーツ属性は、設計時にパーツのプロパティー・ノートブックを使用してセットすることができます。また、命令コード GETATR および SETATR、または組み込み関数 %getatr および %setatr を使用して、実行時に一部のパーツの属性を変更したり検索したりすることもできます。属性および属性の設定場所の詳細については、*ADTS/CS VisualAge for RPG パーツ解説書* を参照してください。

GETATR および **SETATR** は、イベントを生成したパーツと同じウィンドウ上のパーツを参照するために使用することができる固定命令コードです。たとえば、イベントを生成したパーツが WINDOW1 上にある場合には、固定命令コードは WINDOW1 上のパーツしか参照することができません。命令コード GETATR または SETATR が別のウィンドウ上のパーツを参照している場合には、コンパイラーが、参照されたパーツがそのウィンドウ上にあるかを確認するために、単一リンク・アクション・サブルーチンの場合は、コンパイル時エラーが発生します。多重リンク・アクション・サブルーチンの場合は、実行時エラーが発生します。

異なるウィンドウ上のパーツを参照するためには、組み込み関数 **%getatr** および **%setatr** を使用しなければなりません。これらの組み込み関数を使用して、ウィンドウ名とパーツ名を指定することができます。

これらの命令コードと組み込み関数の使用については、*VisualAge for RPG WINDOWS 版 言語解説書* を参照してください。

ユーザー・プログラム内でのパーツの参照

パーツ属性の取り出し時または設定時に、パーツは GUI Designer でそれに定義された名前を使用して参照します。この名前は AS/400 システム命名規則に従っていなければなりません。特に、名前は次の通りです。

- 長さが 10 文字を超えてはいけません。SBCS 文字しか使用することができません。文字は、英字 A-Z、数字 0-9、@、#、$、または _ (下線) としなければなりません。
- 英字 A-Z、@、#、または $ で始まっていなければなりません。
- 大文字または小文字で入力することができます。
- 組み込みブランクが**あってはいけません**。
- 拡張名としては**いけません** (すなわち、二重引用符で囲んではいけません)。

注: ユーザー・プログラムの実行中に、**作成** 済みのパーツの名前のみ参照することができます。パーツは、現行のウィンドウも作成される時に作成されます。ウ

ィンドウまたはパーツを作成すると、それがメモリーの中にロードされます。まだ作成されていないパーツを参照しようとすると、*パーツが見つからない* というメッセージが表示されることになります。

イベントへの応答

各パーツは一組の定義済みイベントに応答します。次の方法の 1 つを使用して、定義済みイベントのリストを獲得することができます。

1. 完全なリストについては、*ADTS/CS VisualAge for RPG パーツ解説書* を参照してください。
2. パレットまたはカタログ内のパーツにフォーカスがある時に F1 キーを押して、パーツの一般説明と、属性およびそれと関連したイベントのリストを表示します。
3. GUI Designer で、パーツのポップアップ・メニューを起動して、**イベント**項目を選択を選択します。

一般に、イベントは、ユーザー・インターフェースと対話した結果として生成されます。たとえば、プッシュボタンを押すと、**Press** イベントのシグナルが送られます。また、イベントはユーザー・プログラムによって生成することもできます。たとえば、DDE クライアント・パーツは、事前に決められた時間以内にサーバー・プログラムとの会話を開始することができない場合に、 **Timeout** イベントを生成します。ユーザー・プログラムが入力フィールド・パーツのテキスト値を変更した場合には、その入力フィールドによって **Change** イベントのシグナルが送られます。

イベントには、ユーザー・プログラムで命令コード BEGACT (begin action) と ENDACT (end action) をコーディングすることによって応答します。これらの命令コードの間のコードは、**アクション・サブルーチン** と呼ばれ、特定のイベントに対して実行されます。特定のイベントのアクション・サブルーチンを作成する時に、**アクション・リンク**が定義されます。特定のイベントのアクション・サブルーチンをコーディングしない場合には、そのイベントの発生時にアクションは実行されません。アクション・サブルーチンのコードは、ENDACT 命令コードに達するまで実行されます。したがって、アクション・サブルーチン内に EXSR 命令コードをコーディングした場合には、これらのサブルーチン (**ユーザー・サブルーチン** と呼ばれる) も実行されます。

アクション・サブルーチンは EXSR 命令コードを使用して呼び出すことができません。しかし、特定のアクション・サブルーチンを複数のアクションで呼び出すことはできます。たとえば、プッシュボタンが押された時またはメニュー項目が選択された時に実行されるコードにすることができます。アクション・サブルーチン・ウィンドウで、どのイベントにアクション・サブルーチンがあるかを検討して、リンク・イベントをアクション・サブルーチンに変更することができます。アクション・サブルーチン・ウィンドウを表示するためには、次を実行します。

1. GUI Designer の**プロジェクト**・メニューから**ソース・コードの編集**を選択します。これで、編集セッションが開始されます。
2. 編集セッションで、**編集 > アクション・サブルーチン**を選択します。アクション・サブルーチン・ウィンドウが表示されます。

イベント属性には、イベントに関係のあるデータが入っています。たとえば、**MouseMove** イベントは X 座標と Y 座標を保管して、イベントの発生時のマウス

の位置を指示します。ユーザー・プログラムでイベント属性を使用する前に、定義仕様でそれが定義されていなければなりません。イベント属性の名前は、定義仕様のエンティティーの名前です。コンパイラーは変数の長さを検査しない上に、一部の属性は可変長になっているために、必ず、必要な値を入れるのに十分に大きい長さを指定してください。

注: イベント属性はユーザー・プログラムで変更することはできません。したがって、結果フィールドに表示されること、または EVAL 演算命令のターゲット・フィールドとして表示されることはありません。

ADTS/CS VisualAge for RPG パーツ解説書 には、すべてのイベント属性について記述されています。

次の例は、イベント属性 **%MouseX** と **%MouseY** をプログラムで定義して使用する方法の説明です。

```
 *
 * Define mouse x and y coordinate event attributes
 *
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++
D%MouseX          S               4P 0
D%MouseY          S               4P 0
 *
 * Check if Mouse coordinates in range:
 *
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq...
CSRN01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++++
C     %MouseX       ifgt      100
C     %MouseY       andgt     100
C                    ..
C                   endif
 *
 ***** End of Source
```

システム属性

システム属性は、特定のパーツではなくユーザー・アプリケーションに属しています。

イベント属性の場合と同様に、システム属性は定義仕様で定義されていなければなりません。また、それらをユーザー・プログラムで変更することはできません。

VisualAge RPG は次のシステム属性をサポートしています。

表 1. システム属性

属性	説明	タイプ	長さ
%DspHeight	実行時に画面の高さ（ピクセル数）を戻します。	数値	4
%DspWidth	実行時に画面の幅（ピクセル数）を戻します。	数値	4

イベントおよびシステム属性の処理

各イベント属性は特定のイベントに有効であり、そのイベントにリンクしているアクション・サブルーチン内でしか使用することができません。たとえば、**ReSize** イベントにリンクされているアクション・サブルーチン内で **MouseMove** イベントのイベント属性を使用した場合には、実行時エラーが出されます。タイプ検査が実行されるのは実行時のイベント属性に対してだけです。数値イベント属性に対して文字フィールドを定義した場合に、このエラーが検出されるのは実行時だけです。

システム属性は特定イベントにリンクしているわけではないので、ユーザー・プログラム内のどの場所でも使用することができます。システム属性に対するタイプ検査はコンパイル時に実行されます。

イベント属性とシステム属性は、VisualAge RPG コンポーネントから使用する前に、定義仕様に定義されていなければなりません。これらは、コンパイラーによって自動ストレージ内の読み取り専用フィールドとして取り扱われます。ネストされたアクション・サブルーチンおよびアクティブ・アクション・サブルーチンは、イベント属性の独自のコピーをもっています。

たとえば、ENT0000A+CHANGE+WIN1 アクション・サブルーチンは、ウィンドウ WIN1、入力フィールド・パーツ ENT0000A、およびイベント CHANGE にリンクしているものとします。

```
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
CSRN01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++
C     ENT0000A      BEGACT    CHANGE        WIN1
C                     .
C                     .
C     %PART         dsply     boxid         reply
 * 'ENT0000A' is displayed.
C                     .
C                     .
C                   endact
```

さらに、PSB0000A+PRESS+WIN1 サブルーチンは、ウィンドウ WIN1、プッシュボタン・パーツ PSB0000A、およびイベント PRESS にリンクしているものとします。

```
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
CSRN01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++
C     PSB0000A      BEGACT    PRESS         WIN1
C                     .
C     %PART         dsply     boxid         reply
 * 'PSB0000A' is displayed.
C                     .
C     ENT0000A      SETATR    10            TEXT
 * This triggers the CHANGE event for entry field ENT0000A
 * which causes action subroutine ENT0000A+CHANGE+WIN1 to be
 * invoked.
C                     .
C                     .
C     %PART         dsply     boxid         reply
 * 'PSB0000A' is displayed.
C                     .
C                     .
C                   endact
```

プッシュボタン PSB0000A が押された時は、アクション・サブルーチン PSB0000A+PRESS+WIN1 が呼び出されます。 SETATR 演算命令が実行された時に、CHANGE イベントが入力フィールド・パーツ ENT0000A 用に起動されます。これで、ENT0000A+CHANGE+WIN1 アクション・サブルーチンが呼び出されます。

イベント属性属性は自動ストレージ内にあるために、各アクション・サブルーチンは次のように、%PART 用の独自のストレージをもっています。

- アクション・サブルーチン PSB0000A+PRESS+WIN1 では、 %PART に 'PSB0000A' が入っている。
- アクション・サブルーチン ENT0000A+CHANGE+WIN1 では、 %PART に 'ENT0000A' が入っている。
- アクション・サブルーチン ENT0000A+CHANGE+WIN1 が完了して、アクション・サブルーチン PSB0000A+PRESS+WIN1 が実行を続行している場合には、%PART には 'ENT0000A' ではなく 'PSB0000A' が入っている。

静的テキストおよび入力フィールド・パーツのコーディング

次の項には、静的テキストおよび入力フィールド・パーツをコーディングするためのヒントがいくつか記載されています。

入力フィールド・パーツの作成および検索

注: この項は、静的テキスト・パーツにも適用されます。単純にするために、本分では入力フィールド・パーツだけに言及します。

READ が実行された時に、VisualAge RPG は検索した値をどこに保管するか?WRITE が実行された時に、VisualAge RPG が値をセットするために使用する値はどれか?

それぞれの入力フィールド・パーツごとに、VisualAge RPG はパーツと同じ名前をもつフィールドを作成します。このフィールドは、**Text** 属性 (または静的テキスト・パーツの場合は **Label** 属性) の定義と一致するように定義されます。たとえば、ENT00012 と呼ばれる入力フィールド・パーツがあり、 **Text** 属性が 20 文字として定義されている場合には、 VRPG は自動的に *ENT00012* と呼ばれる 20 文字のフィールドを定義します。このフィールドはユーザー・プログラム内で使用することができます。

定義仕様のフィールドの定義は、同じ名前のフィールドを定義することによってオーバーライドすることができます。しかし、フィールドの定義は、タイプおよび長さの互換性に関して VisualAge RPG の規則に従っていなければなりません。たとえば、フィールドは属性定義と同じ長さでなければなりません。数値フィールドの場合は、フィールドは属性定義と同じタイプにする必要はありません。

ユーザー・アプリケーションを実行する時に、入力フィールドは GUI Designer で指定した値によって初期設定されます。しかし、この値は定義仕様に INZ キーワードを設定するか、あるいは値をプログラム・フィールドに転送することによって上書きすることができます。これらの場合には、これらのフィールドのそれぞれに保管される値は、必ずしも画面上に表示される対応するパーツの値と一致していません。

ユーザー・サブルーチンまたはアクション・サブルーチンでフィールドに異なる値を保管した場合には、VisualAge RPG はその新しい値を画面上に反映させません。したがって、フィールドに保管される値は、画面上に表示される値とは異なっています。画面上に保管された値を反映するためには、WRITE 演算命令または SETATR 演算命令を使用しなければなりません。

SHOWWIN の場合にも同じことが言えます。ウィンドウが最初にオープンされる時に、画面上に表示される値は、GUI Designer でパーツに指定された値と対応しています。ウィンドウを表示する前に、対応する VisualAge RPG フィールド用に保管された値を変更した場合には、そのフィールドの値は画面上に表示される値と一致しません。 2 つの値を等しくするためには、ウィンドウの **Create** イベントにリンクされているアクション・サブルーチンで、WRITE 演算命令または属性設定演算命令を実行する必要があります。これで、フィールドに保管するされた値は画面上の値と同期するようになり、ウィンドウが表示される時に、ユーザーには新しい値しか表示されません。

一般に、**Create** イベントにリンクしているアクション・サブルーチンを使用して、ウィンドウがオープンされた時に画面上に表示される値を設定することは良い考えです。

ウィンドウ・パーツの命令コード

VisualAge RPG では、ウィンドウおよびそのパーツに対して操作する幾つかの命令コード (READ、WRITE、CLEAR、および RESET) が拡張されています。これらの命令コードはウィンドウで使用することができ、静的テキストと入力フィールド・パーツに影響を与えます。

READ 影響を与えたすべての静的テキストおよび入力フィールド・パーツに対して属性取り出し操作を実行します。

WRITE
影響を与えたすべての静的テキストおよび入力フィールド・パーツに対して属性設定操作を実行します。

CLEAR
すべての数値入力フィールド・パーツをゼロに、すべての文字入力フィールド・パーツをブランクにセットします。(これは、静的テキスト・パーツに対しては操作されません。)

RESET
静的テキストおよび入力フィールド・パーツを初期値に戻します。

ウィンドウ命令コードは次の属性を使用します。

Text READ、WRITE、CLEAR、および RESET 命令を実行するために使用される入力フィールド・パーツの属性。

Label READ、WRITE、および RESET 命令を実行するために使用される静的テキスト・パーツの属性。

同じ名前をもつパーツに対するウィンドウ命令コードの使用

パーツが異なるウィンドウに属している限り、同じ名前をもつ 2 つの入力フィールド、同じ名前をもつ 2 つの静的テキスト・パーツ、または静的テキスト・パーツと

同じ名前をもつ入力フィールド・パーツすらもつことができます。この項では、不注意にこれらのパーツの 1 つの値を別のパーツの値に設定するのを避ける方法について説明します。

指定されたパーツ名に対してプログラム・フィールドは 1 つしか作成されません。MYPART という名前のウィンドウ W1 に入力フィールド・パーツがあり、MYPART という名前のウィンドウ W2 に入力フィールド・パーツがある場合には、MYPART と呼ばれる VisualAge RPG フィールドが 1 つ作成されます。コンパイラーは、パーツ定義の 1 つと一致するように定義を作成します。

同じ名前をもつパーツが複数個ある場合には、そのパーツの定義に互換性がないと、コンパイラーはエラー・メッセージを出すことになります。パーツに互換性があるというのは、それらのパーツが同じ長さになっている同じタイプのデータ (数値または文字) を受け入れ、 (数値の場合は) 小数部の桁数が同じになっている場合のことです。

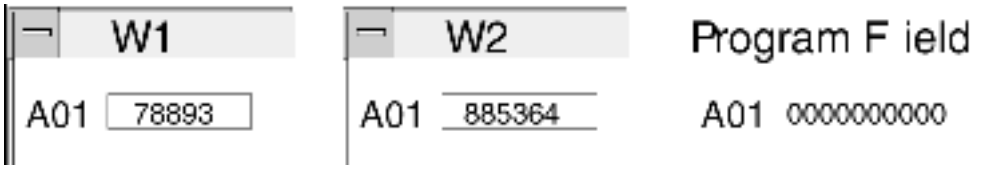
フィールドを共用しているパーツが異なる初期値をもつ場合には、フィールドの初期値は、入力フィールド・パーツ用の内部フィールドの作成時に、コンパイラーが最初に見つけたパーツに従って設定されます。これは、作成のたびに異なる場合があるので、複数パーツが同一フィールドを共用している場合には、すべての初期値を同じ値になるように設定しない限り、特定の初期値をもっているフィールドに依存してはいけません。

これらのパーツそのものが入っているウィンドウの 1 つ、またはそのパーツの 1 つに対して操作を実行すると、入力フィールドに、操作の影響を受けたパーツの画面の値と一致する値が入ることになります。しかし、このフィールドには、このフィールドを共用する他方のウィンドウ上の他方のパーツの画面の値と一致していないと考えられる値が入ります。複数パーツが同一フィールドを共用している場合でも、これらのパーツのいずれに対する操作も、操作時に指定されたパーツまたは操作時に指定されたウィンドウに含まれているパーツにしか影響を与えません。フィールドを共用しているその他のパーツは影響を受けません。

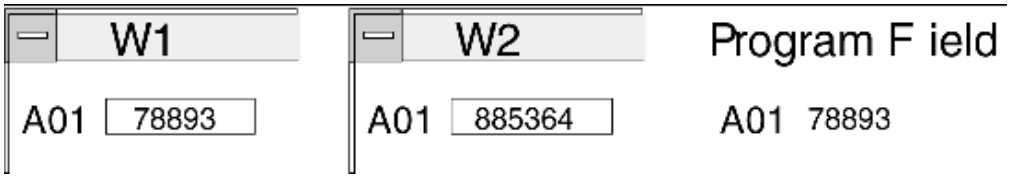
例

次の例は、パーツがフィールドを共用している時に、一方のパーツの値をもう一方のパーツの値に設定した時に何が起こると考えられるかを示しています。

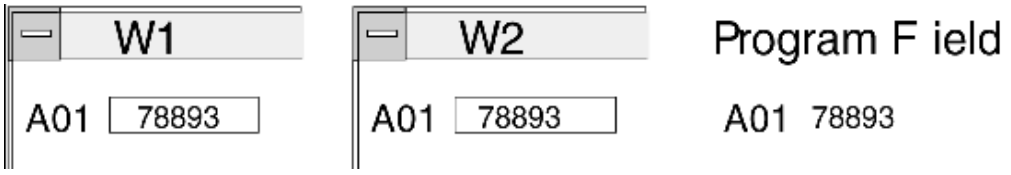
1. フィールドの定義: ウィンドウ W1 の入力フィールド A01 は 10 文字として定義されていて、ウィンドウ W2 の入力フィールド A01 は 10 文字として定義されています。 W1 の画面上の値は *78893* であり、W2 の画面上の値は *885364* です。フィールド A01 には値 *0000000000* が入っています。これらが初期値です。



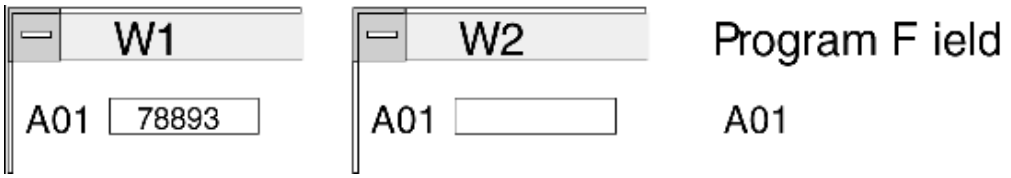
2. W1 に対する READ の実行: 現在、フィールド A01 には *78893* が入っています。これは W1 の入力フィールド A01 と一致します。



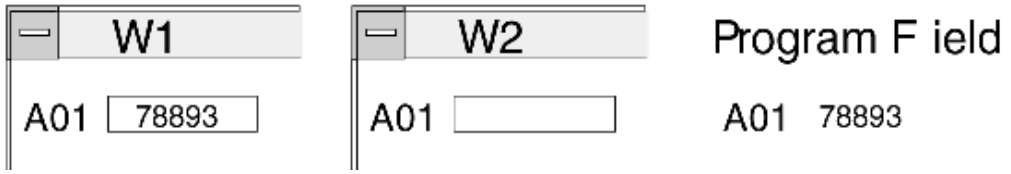
3. W2 に対する WRITE の実行: 現在、W2 の入力フィールド A01 の画面の値は *78893* です。



4. W2 に対する CLEAR の実行: 現在、フィールド A01 にはブランクが入っています。これは W2 の入力フィールド A01 と一致します。 W1 の入力フィールド A01 - 画面上の値は *78893* です。 W2 の入力フィールド A01 - 画面上の値はブランクです。



5. ターゲット・フィールド A01 をもつ W1 の入力フィールド A01 に対する GETATR の実行: 現在、フィールド A01 には *78893* が入っています。これは W1 の入力フィールド A01 と一致します。



あるフィールドを共用しているすべてのパーツが同じ値を表示するようにしたい場合には、ソース値としてそのフィールドを使用して、すべてのパーツに対して SETATR 命令を実行するか、あるいはこれらのパーツの 1 つを含むすべてのウィンドウに対して WRITE 命令を実行しなければなりません。

偶然にもパーツの一方の値をもう一方の値に設定するのを避けるためには、コンポーネント内のすべての入力フィールド・パーツに固有の名前を付けるようにお奨めします。

第 4 章 VisualAge RPG のサンプル・プログラム

サンプル・フォルダー (VisualAge RPG プロジェクト・フォルダーの) には、本書のこの部で説明しているソース・コードとサンプル・アプリケーションが入っています。表 2 サンプル・プログラムをリストします。

表 2. VisualAge RPG のサンプル・プログラム

プログラム	説明
アニメーション	アニメーション・コントロール・パーツの例
ActiveX	ActiveX コントロールの例
bean	Java bean パーツの例
カレンダー	カレンダー・パーツの例
コンポーネント参照パーツ	コンポーネント参照の例
コンテナー	コンテナー・パーツの例
カスタマー・メインテナンス*	カスタマー・メインテナンスの例
DDE クライアント	DDE クライアント・パーツの例
DDE ホット・リンク	DDE ホット・リンクの例
ドラッグ・アンド・ドロップ	データ転送の例
グラフ	グラフ・パーツの例
イメージ*	イメージ・パーツの例
リスト・ボックス	リスト・ボックス・パーツの例
メッセージ・サブファイル	メッセージ・サブファイル・パーツの例
複数行編集	複数行編集パーツの例
ノートブック	ノートブック・パーツの例
Odbcceld	ODBC/JDBC インターフェース・パーツの例
ポップアップ・メニュー	ポップアップ・メニュー・パーツの例
進行状況	進行状況バー・パーツの例
サイズ変更	サイズ変更の例
Runtime_control_of_server_connections	API へのサインオン例を使用したサーバー接続の制御
スクロール	スクロール・バー・バー・パーツの例
スライダー**	スライダー・パーツの例
スピン・ボタン	スピン・ボタン・パーツの例
サブファイル*	サブファイル・パーツの例
タイマー	タイマー・パーツの例
VARPG プラグイン	ベンダー・プラグインの例
ビデオ記憶キャッシャー*	ビデオ記憶キャッシャーの例
ビデオ記憶カタログ*	ビデオ記憶カタログの例
ようこそ!	ウェルカムの例

注:

1. * この例では iSeries 400 サーバー上のデータが必要です。
2. ** **BackMix** および **ForeMix** 属性の使用法も示します。

開始する前に

サンプル・アプリケーションを実行する前に、VisualAge RPG コンポーネントをインストールしなければなりません。関連サンプルはサンプル・フォルダー (VRPG プロジェクト・フォルダーの中) にあります。

サンプル・プログラムのコメントをお読みください。コメントには、制約事項とともにヒントおよび要件が含まれています。

Java アプリケーションをビルドして実行する前に、Sun Microsystems の Java 2 ソフトウェア開発キット (J2SDK) バージョン 1.2 以上をワークステーションにインストールしておかなければなりません。J2SDK がない場合には、次の URL で Sun Microsystems からダウンロードできます。

```
http://java.sun.com/products/
```

J2SDK をダウンロードした後で、Java コンパイラーおよび Java Runtime Environment (JRE) の両方の場所をポイントするように、PATH 環境変数を設定してください。たとえば、J2SDK のホーム・ディレクトリーが *c:\jdk1.2* の場合には、次の path ステートメントを追加してください: `c:\jdk1.2\bin`

ブラウザー内で VisualAge RPG アプレットを実行する予定の場合は、クライアント・ワークステーションに国際バージョンの JRE をインストールしなければなりません。

サンプルのビルド

ほとんどのサンプルを実行したい場合は、最初にアプリケーションをビルドしなければなりません。

サンプル・プログラムの 1 つをビルドするには、そのサンプル・フォルダーのポップアップ・メニューを表示して、**ビルド > Windows** または**ビルド > Java** を選択してください。

サンプルの実行

サンプル・プログラムを実行するには、そのプログラムのポップアップ。メニューを表示して、**実行 > Windows** または**実行 > Java** を選択してください。

iSeries 400 サーバーへのアクセス

一部のサンプル・プログラム（サブファイルの例など）は、iSeries 400 サーバー上のデータにアクセスします。これらのプログラムによって使用されるデータ・ファイルは VisualAge RPG には同梱されていません。しかし、ソース・ファイルのコメントのセクションにその例のファイルのレイアウトについての説明があります。ユーザーがサーバー上にデータ・ファイルを作成して、データを供給しなければなりません。

これらのサンプルを実行するためには、サンプル・プログラムで GUI Designer を開始し、**サーバー**・プルダウン・メニューの「iSeries 情報の定義」ノートブックを使用して次のことを行なってください。

1. リモート・ロケーション・パラメーターを変更して、アクセスしたいサーバーをポイントするようにします。
2. リモート・ファイル名パラメーターを、このサンプルに適切なデータ・ファイルをアクセスできるように変更します。

iSeries 400 情報を定義する場合の詳細については、205 ページの『第 8 章 iSeries の接続性』を参照してください。

第 5 章 共通属性

この項では、ほとんどのパーツに共通の属性をリストして、それらの使用法について説明します。

PartName 属性

すべてのパーツは名前を持ちます。VisualAge RPG は、パーツの作成時に自動的にこの名前を生成します。パーツの名前は、そのプロパティー・ノートブックで変更するか、GUI Designer のプロジェクト・ウィンドウのツリー・ビューで直接編集して変更することができます。*コンポーネント・パーツ名は編集できません。

注: 実行時にパーツ名を変更することはできません。

それぞれのウィンドウには固有の名前があり、所定のウィンドウのすべてのパーツは固有名を持たなければなりません。別のウィンドウのパーツは同じ名前を持つことができます。ただし、サブファイル・パーツ名はそのコンポーネントを通して固有でなければなりません。

コンパイラーは、**PartName** 属性を使用して暗黙に入力フィールドのフィールド名と静的テキスト・パーツを定義します。これらのパーツの値を参照したい場合には、ユーザー・プログラムにこの名前を使用することができます。詳細については、29 ページの『第 3 章 パーツを用いたプログラミング』 を参照してください。

パーツの名前を変更する場合には、プログラム・ソースのそのパーツに対するすべての参照を変更しなければなりません。名前変更されたパーツの古い名前を参照すると、パーツが見つからないことを示すコンパイル・エラーまたは実行時エラーが表示されます。

ParentName 属性

ParentName 属性は、親パーツの名前を返します。親は、パーツが入っているウィンドウです。ウィンドウ・パーツの場合には、親はウィンドウ自体です。

PartType 属性

PartType 属性を使用して、プログラム内のパーツのタイプを決定することができます。**PartType** は、VisualAge RPG によって定義されたパーツのタイプを返します。VisualAge RPG パーツに対して返される値は、ストリング *FVDES* とそれに続くパーツ・タイプから構成されます。たとえば、入力フィールド・パーツの場合には、パーツ・タイプは FVDESEntryField となります。この規則の 1 つの例外はコンポーネント参照パーツであり、これは接頭部 FVDESV をもっています。

42 ページの表 3 には、各 VisualAge RPG パーツごとの **PartType** 属性の値が要約されています。

表 3. VisualAge RPG パーツの PartType 属性

PartType 属性	VisualAge RPG パーツ
FVDESOCX	ActiveX
FVDESAnimationControl	アニメーション制御
FVDESCalendar	カレンダー
FVDESCanvas	キャンバス
FVDESCheckBox	チェック・ボックス
FVDESComboBox	組み合わせボックス
FVDESContainerControl	コンテナー
FVDESVComponentReference	コンポーネント参照
FVDESDDEClient	DDE クライアント
FVDESEntryField	入力フィールド
FVDESGraph	グラフ
FVDESGraphicPushButton	グラフィック・プッシュボタン
FVDESGroupBox	グループ・ボックス
FVDESHScrollBar	水平方向のスクロール・バー
FVDESImage	イメージ
FVDESJavaBean	Java Bean
FVDESListBox	リスト・ボックス
FVDESAudio	メディア
FVDESMediaPanel	メディア・パネル
FVDESMenuItem	メニュー項目
FVDESMessageSubfile	メッセージ・サブファイル
FVDESMultiLineEdit	複数行編集
FVDESNotebook	ノートブック
FVDESNotebookPage	ノートブック・ページ
FVDESODBCInterface	ODBC/JDBC インターフェース
FVDESOutlineBox	外枠
FVDESPopUpMenu	ポップアップ・メニュー
FVDESProgressBar	進行状況表示バー
FVDESPushButton	プッシュボタン
FVDESRadioButton	ラジオ・ボタン
FVDESSlider	スライダー
FVDESSpinButton	スピン・ボタン
FVDESStaticText	静的テキスト
FVDESStatusBar	ステータス・バー
FVDESSubfile	サブファイル
FVDESSubmenu	サブメニュー
FVDESTimer	タイマー
FVDESVScrollBar	縦方向のスクロール・バー
FVDESFrameWindow	ウィンドウ

カラー属性

ほとんどのパーツのカラーは、**BackColor** および **ForeColor** 属性を使用して変更することができます。属性値は、特定のカラーを表す数字です。コンパイラーは、カラーの設定に使用できる *RED や *GREEN などの一連の表意定数です。これらの名前については、*VisualAge for RPG WINDOWS 版 言語解説書* を参照してください。

パーツのカラー・ミックスは、 **ForeMix** および **BackMix** 属性を使用して指定することができます。これらの属性の値は、赤、緑、および黒の基本カラーの混合を表します。これは、**RGB** カラー値と呼ばれることもあります。 この RGB 値は、コロン (:) で区切られた 3 つの値から構成されます。それぞれの値は、赤、緑、および青の輝度をこの順序で表します。それぞれのカラーの値は、0 ～ 255 の間です。

次のコードの例では、静的テキスト・パーツの背景カラー・ミックスは青の中間色調に設定されます。

```
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
 *
C     'ST1'         setatr    '000:000:128' 'BackMix'
```

ForeMix および **BackMix** 属性の指定方法の詳細な例については、 155 ページの『スライダー』 を参照してください。

Enabled 属性

パーツが使用可能になると、ユーザーの対話に応答することができ、イベントを生成することができます。たとえば、使用可能になったプッシュボタンを押すと、ユーザー・プログラムで処理できる**Press** イベントが生成されます。

パーツは、特定の条件が存在しなければ使用可能にすることはできません。プッシュボタンの場合には、ユーザーが押すことができるのは、リスト・ボックスで品目が選択された時だけです。

パーツが使用可能になっていない場合には、ユーザーの対話に応答しないで、そのラベルがグレー表示されています。

パーツを使用可能にするためには、**Enabled** 属性値を *1* に設定してください。使用可能にする必要がない場合には、この値を *0* に設定してください。

サイズおよび位置属性

Height および **Width** 属性を使用して、ほとんどのパーツのサイズ（ピクセル数）を指示することができます。

また、**Left**、**Bottom**、および **Top** 属性を使用して、それが入っているパーツ (通常はウィンドウ) 内のパーツの位置を指定することができます。位置の値もピクセルで表されます。任意のパーツを位置指定すると、その値は左上隅に呼応します。

これらの属性は、実行時にパーツのサイズと位置を動的に変更するので、非常に有用です。たとえば:

- ユーザーがウィンドウのサイズを変更できる場合には、 **ReSize** イベントを処理するアクション・サブルーチンをコーディングして、そのウィンドウのパーツの位置がウィンドウ内の中心に留まるように変更することができます。これを行わない場合には、パーツはウィンドウの左上隅に呼応して留まります。
- アプリケーションが異なる解像度のモニターを使用するシステムで実行される場合には、**%DspWidth** および **%DspHeight** システム属性を使用して、画面の解像度に関係なくウィンドウが見えるようにします。ウィンドウは画面の中心またはその他の座標で位置決めすることができます。

ウィンドウの **Create** イベントで実行できる計算を示します。この例では、*Window1* と呼ばれるウィンドウを中心に置くための適切な座標を計算し、次にこのウィンドウを新しい座標に移動してから画面上に表示します。

```
 *
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
 *
 * 画面サイズ・システム属性を宣言します。
D%DspHeight        S             4P0
D%DspWidth         S             4P0
 *
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq
 *
 * Window1 の作成イベントを処理します。
 * 画面サイズを取り出して、ピクセル座標を計算して、
 * ウィンドウを画面の中心に位置づけます。
 *
C     WINDOW1       BEGACT    CREATE        WINDOW1
 *
 * ウィンドウの寸法を取り出します。
C     'Window1'     GETATR    'Width'       winWidth          4 0
C     'Window1'     GETATR    'Height'      winHeight         4 0
 *
 * ウィンドウを中心に置く新しい座標を計算します。
C     %DspWidth     SUB       winWidth      newLeft           4 0
C     %DspHeight    SUB       winHeight     newBottom         4 0
C     newLeft       DIV       2             newLeft
C     newBottom     DIV       2             newBottom
 *
 * ウィンドウを中心に置いて、見えるようにします。
C     'Window1'     SETATR    newLeft       'Left'
C     'Window1'     SETATR    newBottom     'Bottom'
C     'Window1'     SETATR    1             'Visible'
C                   ENDACT
 *
```

Visible 属性

Visible 属性を使用して、パーツまたはウィンドウを表示する時点を指定することができます。たとえば、プッシュボタンは実行時のみに画面上に表示することができます。このためには、GUI Designer でプッシュボタンを作成する時に、そのパーツのプロパティー・ノートブックに進んで visible フラグをオフに設定してください。その後で、実行時に、プッシュボタンを表示したい時に **Visible** 属性値を *1* に設定します。

Focus 属性

ウィンドウのユーザーがインターフェースと対話できる部分は入力フォーカスを持ちます。入力フォーカスを持つパーツは、キーやボタンを押すなどのユーザー・アクションに応答できなければなりません。

ユーザーが直ちに使用できるように、プログラムでパーツにフォーカスしたい場合があります。たとえば、複数の入力フィールドを検査していて、特定の入力フィールドに情報を再入力する必要がある場合には、その入力フィールドの **Focus** 属性値を *1* に設定します。この属性を設定すると、ユーザーが入力フィールドに データをタイプできる場所にカーソルが現れます。

特定のパーツにフォーカスを付与することに加えて、パーツがフォーカスを獲得したかどうかを確認することができます。パーツは、ユーザーがタブ移動するかまたはマウスで選択して、そのパーツを選択した時にフォーカスを獲得します。これが起こると、そのパーツに **GotFocus** イベントが生成されます。逆に、パーツがフォーカスを失った時には、 **LostFocus** イベントが生成されます。

UserData 属性

すべてのパーツは **UserData** 属性をサポートします。この属性は、パーツにテキスト・ストリングを割り当てるために使用します。このストリングは、パーツの他の属性の値には影響を与えず、表示されません。 **UserData** 属性には、最大 65,535 文字を入れることができます。

Label 属性

いくつかのパーツは **Label** 属性を持ちます。これは、パーツの目的を説明する記述テキストです。プッシュボタンに表示されるテキストはこの属性の例です。

次のパーツは **Label** 属性を持ちます。
- チェック・ボックス
- コンテナー
- グループ・ボックス
- メニュー項目
- プッシュボタン
- ラジオ・ボタン
- 静的テキスト
- ウィンドウ
- キャンバスのあるウィンドウ

ラベルの置き換え

ラベルのテキストは、**Label**、 **TabLabel**、または **InfoLabel** 属性を設定する時に、記号を使用することによって置き換えることができます。これは、GUI Designer で定義されたラベルおよびメッセージを変換できるので、他の言語で使用するアプリケーションを開発する場合に特に有用です。

パーツのラベルの置き換えを指定する場合には、コンポーネントのメッセージ・ファイルに入力を行ないます。複数のパーツが同じラベルの置き換え (たとえば、^OK.) を使用することができます。すべての参照に同じメッセージ・ファイル項目が使用されます。

パーツの置き換えラベルは、組み込みブランクのない、脱字 (^) 記号が先行するストリングを指定して、そのプロパティー・ノートブックで定義します。これは、メッセージ・ファイルにメッセージを追加します。アプリケーションの実行時に、メッセージ・エディターを呼び出して **Label** 属性と置き換えたいメッセージ・テキストを追加するためには、**プロジェクト > メッセージの定義...** を選択します。

変換のヒント

置き換えラベルを持つ GUI Designer のパーツをサイズ決めする場合には、変換されたテキストが元のテキストより長くなる場合があることに留意してください。

簡略記号を使用する場合には、他の言語では簡略記号文字が異なる場合があることに留意してください。

それぞれに別のファイル拡張子を割り当てることによって、実行時サブディレクトリーの中に複数の変換済みメッセージ・ファイルを持つことができます。たとえば、コンパイル済みメッセージ・ファイルの英語バージョンは SAMPLE.ENG という名前にして、ドイツ語バージョンは SAMPLE.GER という名前にすることができます。アプリケーションを実行する前に、該当するメッセージ・ファイルを SAMPLE.MSG にコピーするようにユーザーに指示することができます。

第 6 章 データ転送の使用

この項では、データ転送を使用して一部のパーツの値を操作する方法について説明します。

注: データ転送は Java アプリケーションではサポートされていません。

代表的なデータ転送のシナリオ

代表的なデータ転送のシナリオでは、ユーザーがマウスでパーツ (**ソース・パーツ** と呼ばれる) を選択して別のパーツ (**ターゲット・パーツ** と呼ばれる) までドラッグし、ボタンを放してその値をターゲット・パーツにドロップします。これで情報がソース・パーツからターゲット・パーツに転送され、ターゲット・パーツはその情報に影響を与えることができます。

注: データ転送では、パーツ自体は移動しません。これは、転送されるパーツの値です。

データ転送をサポートするパーツ

次の表に、データ転送をサポートするパーツおよびそのそれぞれが影響を与えるデータをリストします。

表 4. VisualAge RPG データ転送をサポートするパーツ

パーツ	転送されたデータ
組み合わせボックス	組み合わせボックスの入力フィールド部分の値または組み合わせボックスのドロップダウン・リスト・タイプの選択済み項目。
入力フィールド	**Text** 属性値
リスト・ボックス	選択済み項目
メッセージ・サブファイル	選択済みメッセージ
複数行編集	**Text** 属性値
静的テキスト	**Label** 属性値

データ転送用のパーツの使用可能化

パーツをソース・パーツにしたい場合には、その **DragEnable** 属性を値 1 に設定し、パーツをターゲット・パーツにしたい場合には、その **DropEnable** 属性を値 1 に設定します。これらの属性は、パーツのプロパティー・ノートブックまたはプログラム中で設定できます。これらの属性を実行中にリセットすることは**できません**。パーツをデータ転送に使用できるようにすると、そのパーツは設定されたままになります。

DragEnable および **DropEnable** 属性の設定後には、ソース・パーツをドラッグしてターゲット・パーツの上にドロップすることができます。これでターゲット・パーツに **Drop** イベントが行なわれます。

注: DragEnable および **DropEnable** 属性は Java アプリケーションではサポートされません。

データ転送の例

次の例では、ウィンドウに **EF1** と **EF2** という名前の 2 つの入力フィールドが あります。 EF1 には **DragEnable** 属性が設定され、 EF2 には **DropEnable** 属性が設定されています。 EF1 のテキスト値をドラッグして EF2″ にドロップすることができます。

入力フィールド **EF2** には特定の値しか使用できません。このパーツの **Drop** イベントでは、アクション・サブルーチンがドロップされる値が有効であるかどうかを検査します。値が有効でない場合には、メッセージ・サブファイル・パーツにメッセージが追加されます。

```
 ****************************************************************************
 *                                                                          *
 * プログラム ID. : DragDrop                                                *
 *                                                                          *
 * 説明 . . . . . : DROP イベントに応答してドロップされたデータ            *
 *                  にアクセスする方法を示すサンプル・プログラム。          *
 *                                                                          *
 *                  注:  ドラッグ・アンド・ドロップは Java では             *
 *                       サポートされません                                 *
 *                                                                          *
 ****************************************************************************
 *
 * DROP データ・イベント属性の定義
D%Data            S              5A
 ****************************************************************************
 *                                                                          *
 * ウィンドウ . . : MAIN                                                    *
 *                                                                          *
 * パーツ . . : PB_EXIT                                                     *
 *                                                                          *
 * イベント . . : PRESS                                                     *
 *                                                                          *
 * 説明: プログラムを終了します                                             *
 *                                                                          *
 ****************************************************************************
 *
C     PB_EXIT       BEGACT    PRESS         MAIN
 *
C                   move      *on           *inlr
 *
C                   ENDACT
```

図 12. ドラッグ・アンド・ドロップ・コードのサンプル (1/2)

```
*************************************************************************
*                                                                       *
* ウィンドウ . . : MAIN                                                 *
*                                                                       *
* パーツ . . : EF2                                                      *
*                                                                       *
* イベント . : DROP                                                     *
*                                                                       *
* 説明: このアクション・サブルーチンは、値が入力フィールド EF2          *
*             に「ドロップ」された時に制御を獲得します。                *
*             %Data イベント属性を検査することによって、ドロップ        *
*             された値が入力フィールドに使用できるかどうかが            *
*             確認され、それに応じてメッセージ・サブファイル・          *
*             パーツにメッセージが追加されます。                        *
*                                                                       *
*************************************************************************
*
C     EF2           BEGACT    DROP          MAIN
*
* メッセージ・サブファイルの消去
C     'Msg1'        setatr    0             'RemoveMsg'
*
* ドロップされた値が使用可能かどうかの検査
*
C                   if        %Data <> 'Yes  ' and
C                             %Data <> 'No   ' and
C                             %Data <> 'Maybe'
*
C     'Msg1'        setatr    1             'AddMsgID'
C                   endif
*
C                   ENDACT
*************************************************************************
*                                                                       *
* ウィンドウ . . : MAIN                                                 *
*                                                                       *
* パーツ . . : EF2                                                      *
*                                                                       *
* イベント . : CHANGE                                                   *
*                                                                       *
* 説明:                                                                 *
*                                                                       *
*************************************************************************
*
C     EF2           BEGACT    CHANGE        MAIN
*
C                   ENDACT
```

図 12. ドラッグ・アンド・ドロップ・コードのサンプル (2/2)

第 7 章 パーツの使用

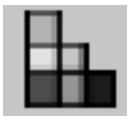
この項には、VisualAge RPG パーツの使用についての有益なヒントを収めてあります。それぞれのパーツの説明には、そのパーツに適用される属性およびイベントのリストを入れてあります。次のパーツについて詳細に説明しています。

- 192 ページの『ウィンドウ』
- 193 ページの『キャンバス付きウィンドウ』
- 201 ページの『*Component』

注: パーツは、使用する順序ではなく、アルファベット順に示してあります。アプリケーションのビルドを開始するには、最初にウィンドウ・パーツまたはキャンバス・パーツ付きのウィンドウを使用し、必要に応じて他のパーツを追加します。

パーツ属性、イベント、およびイベント属性については、*ADTS/CS VisualAge for RPG パーツ解説書*を参照してください。追加のプログラミング上のヒントについては、 29 ページの『第 3 章 パーツを用いたプログラミング』 を参照してください。

ActiveX



*** 制約事項:** このパーツは Java アプリケーションではサポートされていません。

ActiveX パーツを使用して、ActiveX コントロール・オブジェクトをプロジェクトに追加してください。そうすると、アプリケーションは、その属性にアクセスでき、イベントをモニターすることができます。(ActiveX 制御は、第三者ベンダーによって開発および提供されます。)

追加しようとしている ActiveX コントロールに精通している必要があります。VARPG GUI Designer は、これらのパーツによって提供される機能を制御できません。

注: VARPG は、C++ で書かれたインターフェースを持つ ActiveX 制御でしか機能しません。使用したい ActiveX 制御で VARPG が機能するかどうかを ActiveX 制御のプロバイダーに確認してください。

パーツ属性

Activate	AddEvent	Bottom	DeActivate
HasPrpPage	Height	Left	Method
OCXProp	OCXPropIdx	OCXValue	ParentName
PartName	PartType	Refresh	ReturnVal
RmvEvent	ShowProp	Top	UserData
Visible	Width		

適用可能なイベント

Create	Destroy	OCXEvent

ActiveX コントロールの追加

ActiveX コントロールをプロジェクトに追加するには、パーツ・パレットで ActiveX パーツをクリックします。マウス・ポインターで、設計ウィンドウの ActiveX パーツを配置したい場所をクリックします。「オブジェクトの挿入」ダイアログが表示されます。処理したい ActiveX コントロールを選択します。

ActiveX コントロールには、プログラマーが操作し、呼び出し、応答するプロパティー、メソッド、およびイベントを含めることができます。**ActiveX コントロール** という用語は、使用する実際の ActiveX コンポーネントを指します。例には、グラフ、カレンダー、スプレッドシート・コントロールなどが含まれます。*ActiveX パーツ* という用語は、VARPG パーツ・パレット上に見られる ActiveX パーツを指します。

プロパティーの設定

ActiveX コントロールのプロパティーは、ビルド時または実行時に設定することができます。ビルド時にプロパティーを設定するには、設計ウィンドウから「ActiveX パーツのプロパティー」ノートブックをオープンします。設計ウィンドウで ActiveX アイコンを右クリックして、**「プロパティー」**を選択します。 ActiveX コントロールにプロパティー・エディターを使用できる場合には、それも表示されます。これで、直接プロパティーの値を編集することができます。

ActiveX コントロールのプロパティー、メソッド、およびイベントは、「ActiveX パーツのプロパティー」ノートブックの情報ページに表示されます。使用できるものを調べるには、該当するラジオ・ボタンを選択してください。

実行時にプロパティーを設定しあるいは表示するには、2 つのステップが必要です。最初に、ActiveX パーツの **OCXProp** 属性を、使用している ActiveX コントロールのプロパティー名に設定します。プロパティーを設定または表示するには、**OCXValue** 属性を使用します。

次の例では、ActiveX 円グラフ・コントロールの Depth プロパティーを設定します。

```
C     'PieChart'    Setatr    'Depth'       'OCXProp'
C     'PieChart'    Setatr    DepthValue    'OCXValue'
```

注: OCXValue 属性はストリング値を取ります。VisualAge RPG ランタイムは、ActiveX コントロールに値を渡す前に、該当する変換を処理します。

OCXPropIdx 属性は、ActiveX ストリング配列プロパティー・タイプの索引を設定または検索します。この属性を **OCXProp** および **OCXValue** 属性と一緒に使用して、ストリング配列プロパティー・タイプを取り扱うことができます。

たとえば、次のコードはプロパティー ***DataFiles*** の最初のエレメントを c:\temp\Sample.mdb に設定します。すなわち、DataFiles[0]='c:\temp\Sample.mdb' となります。

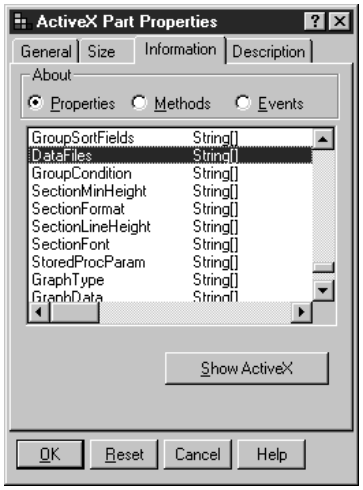
```
C                   EVAL      %SETATR('WINNEWJOB':ocxname:'OCXPROP')=
C                              'DataFiles'
C                   EVAL      %SETATR('WINNEWJOB':ocxname:'OCXPROPIDX')='0'
C                   EVAL      %SETATR('WINNEWJOB':ocxname:'OCXVALUE')
C                              ='C:\temp\Sample.mdb'
```

多次元配列プロパティーのエレメントを設定するには、各エレメントの索引値を 'n1 n2 n3' (n1 は次元 1 の索引など) としてストリングに入れて渡します。たとえば、次のコードは 3DImageData[2][4][7] を 200 に設定します:

```
C                   EVAL      %SETATR('WINNEWJOB':ocxname:'OCXPROP')=
C                              '3DImageData'
C                   EVAL      %SETATR('WINNEWJOB':ocxname:'OCXPROPIDX')
C                              = '2 4 7'
C                   EVAL      %SETATR('WINNEWJOB':ocxname:'OCXVALUE')
C                              = 200
```

OCXProp 属性が設定されるたびに、ランタイムの配列の内部索引値が NULL にリセットされます。したがって、配列プロパティーを設定または取り出すには、最初に **OCXProp** をプロパティー名に設定します。その後で、**OCXPropIdx** を配列の索引値に設定します。最後に、**OCXValue** 属性を設定または取り出します。

ActiveX パーツ・プロパティー・ノートブックでは、「情報」タブにプロパティー・タイプが示されています。プロパティーのタイプがストリング配列である場合には、「情報」タブには ストリング[][]...[] が表示されます。ここで、[] のペアの数は配列の次元を示します。数値配列の場合は、数値[][]...[] が表示されます。非配列プロパティーの場合は、「数値」または「ストリング」が表示されます。次の例では、プロパティーはすべてストリング配列です:



メソッドの呼び出し

VisualAge RPG IDE は、ActiveX コントロールに使用可能なメソッドのリストをコンパイルしようとします。このリストは、「ActiveX パーツのプロパティー」ノートブックの情報ページで使用可能です。それぞれのメソッドに対して、入力パラメーターの場合には IN、出力パラメーターの場合には OUT を入れ、情報が使用不可の場合には何も入れない大括弧付きのパラメーターが示されます。

注: ビルダーが ActiveX コントロールに使用可能なすべてのメソッドを見つけることはできないかもしれません。完全なリストについては、コントロールの資料を調べてください。

ActiveX コントロールのメソッドの呼び出しは、ActiveX パーツの **Method** 属性を設定して、実行時に行なわれます。**Method** 属性は、メソッド名の後にコンマで区切った複数のパラメーター値 (ゼロの場合もある) を続けたストリング値を取ります。構文は次のとおりです。

```
method_name, value1, value2, ...
```

次の例では、ActiveX 円グラフ・コントロールの *AboutBox* メソッドを呼び出します。このメソッドはパラメーターを取りません。

```
C     'PieChart'    Setatr    'AboutBox'    'Method'
```

次の例では、ActiveX 円グラフ・コントロールの *AddData* メソッドを呼び出します。このメソッドは、ストリング・ラベルと浮動値の 2 つのパラメーターを取ります。

```
D datax           C                   const('AddData, ItemX, 3.0')

C     'PieChart'    Setatr    datax         'Method'
```

イベントへの応答

VisualAge RPG は、使用している ActiveX コントロールによって生成されたイベントに応答できます。ActiveX コントロール・イベントは、VisualAge RPG によって **OCXEvent** として受け入れられます。**%RealName** イベントを使用して、イベントの実際の名前を検索することができます。

ActiveX コントロールからイベントを受け取るには、最初に VisualAge RPG ランタイムでイベントを登録する必要があります。ActiveX パーツの **AddEvent** 属性を、受け取るイベントのストリング名に設定します。また、**AddEvent** 属性を *ALL に設定して、ActiveX コントロールによって生成されたすべてのイベントを受け取ることもできます。イベントを抹消するには、**RmvEvent** 属性をそのイベントのストリング名に設定します。

```
 * Declare the event attribute
D %RealName         s         20a

 * Register the event with the VisualAge RPG run time
C     'DataQ'       Setatr    'Click'       'AddEvent'

 * Respond to the OCXEvent
C     DATAQ         BEGACT     OCXEVENT      FRA000000B
C                   if         %RealName = 'Click'
 * Do something here
C                   endif
```

ActiveX パーツがイベントを実行するとき、イベントにパラメーターが含まれている可能性があります。 WDSC V5.1 の前は、これらのパラメーターは無視されました。これらは、アプリケーションで使用可能ではありませんでした。現在、V5.1 にこれらの 2 つの属性が導入されたことにより、イベント・パラメーターがアプリケーションで使用可能になりました。

パラメーターにアクセスするためのステップは、以下の通りです。

1. OCXEVENT が実行されるとき、そのイベント・アクション・サブルーチン内で、最初にパラメーター索引を次のように設定します。

```
C     'ocx'         setatr    2             'PARAMINDEX'
```

2. パラメーターを取得し、文字変数 ″param″ に入れます。

```
C     'ocx'         getatr    'Parameter'   param
```

3. パラメーターが数値の場合、″param″ を数値形式に変換します。

上記のステップは、非ポインター・パラメーターにアクセスするためのものです。ポインター・パラメーターの場合、以下を行う必要があります。

ActiveX コントロール DieRoll を例として挙げます。これは、ダイスの回転をシミュレートし、イベントを持ちます。

```
BeforeRollTo(PTR to Variant number, PTR to Boolean useNumber )
```

上記の情報は、イベントに 2 つのパラメーターがあることを示しています。最初のパラメーター number は、bstrVal メンバーを含む Variant 構造へのポインターで、この bstrVal には、ダイス上の点の数 (「one」から「six」) の値が含まれる場合があります。 vstrVal の値は、アクション・サブルーチンで検索して、別の数値に変

更することができます。 2 つ目のパラメーター useNumber は Boolean 値へのポインターです。この値を「true」に設定した場合、ダイス・コントロールは、ここに設定されている数値が表示されるように回転します (例えば、number=「Four」とすると、これを「five」に設定した場合、ダイスには 5 つの点が表示されます。変更がない場合、ダイスには 4 つの点が表示されます)。

コントロールにパラメーターを渡すには、コントロールがパラメーターを (希望のタイプへの) ポインターとして定義し、ポインターが渡し戻されるようにする必要があります。(ポインター・パラメーターの中には「読み取り専用」のものがあり、アプリケーションはその値を変更できません。これは ActiveX コントロール開発者によって制御されます。)

イベントの BOOL パラメーターへのアクセスおよびその変更方法:

```
     d BoolVal         s              1b 0 based(pBool)
     d Bool            DS
     d pBool                  1      4*            Pointer
     d PB_Value               1      4b 0
     d*'param' is used to receive the pointer parameter. Then
     d*move it into 'PB_Value' to get the numeric value.
     d param           s              8A            pointer, char string

     C*Get the second parameter, a Bool pointer ( BOOL *).
     C     'ocx2'        setatr    2             'PARAMINDEX'
     C     'ocx2'        getatr    'Parameter'   paramBool         7
     C     'msb'         setatr    paramBool     'addmsgtext'
     C                   EVAL      PB_Value = 0
     C                   MOVE      PARAMBool     PB_Value
     C*Change the value to be TRUE, so that the DieRoll control
     C*will use the new parameter1 value set here in this subroutine.
     C                   eval      Boolval = x'01'
```

イベントの最初のパラメーターへのアクセスおよびそれを変更するには、まず VARIANT 構造を調べると役に立ちます。

```
struct VARIANT {
       VARTYPE vt;  //Indicate data type of this structure. VT_BSTR(8) is a string.
       unsigned short wReserved1;
       unsigned short wReserved2;
       unsigned short wReserved3;
       union {
         unsigned char bVal;        // VT_UI1
         short    iVal;             // VT_I2
         long     lVal;             // VT_I4
         float    fltVal;           // VT_R4
         double       dblVal;           // VT_R8
         VARIANT_BOOL xbool;             // VT_BOOL
         DATE     date;             // VT_DATE
         BSTR     bstrVal;          // VT_BSTR
         short    FAR* piVal;       // VT_BYREF|VT_I2
         long     FAR* plVal;       // VT_BYREF|VT_I4
         ... ...
}
```

注: 詳しくは、MicroSoft の ActiveX コントロールの資料を参照してください。

この場合、「number」はストリングとして定義され、タイプ vt は VT_BSTR (これは 8 です) になり、 union 内の ″bstrVal″ メンバーはストリング (「five」など) を指します。

イベントが実行されると、最初のパラメーターは以下のような構造を指します。このポインターを使用して、構造内の値を変更できます。これは、次のように行います。

```
d*'param' is used to receive the pointer parameter. Then
d*move it into 'PTR_Value' to get the numeric value.
d param           s              8A                pointer, char string
dTEST             DS
d P1                       1      4*                Pointer
d PTR_Value                1      4b 0
d
d*This structure is for receiving a VARIANT * pointer from runtime
d*when an event is fired.
d*The value 'bstrValParam' can then be changed.
d VARIANTStrPar   DS                  based(p1)     8=VT_BSTR
d typeStrParam             1      2b 0
d reserved1Param           3      4b 0
d reserved2Param                  6b 0
d reserved3Param           7      8b 0
d bstrValParam                     *                String
d B0                       9     12B 0
d padding2Param           13     16b 0
d
d*Change the 'bstrValParam' member of the structure.
d NewBSTRVal      s             20A   based(bstrValParam)
d

C     OCX2          BEGACT    OCXEVENT      MAIN
C                   IF        %REALNAME='BeforeRollToVariant'
C*Get the pointer (char string), and transform it into a pointer.
C*The first parameter is a pointer to a VARIANT structure.
C     'ocx2'        setatr    1             'PARAMINDEX'
C*Please note that the length of the returned "PARAMETER" string
C*varies. E.g., the length of the first parameter here is 8, while
C*the second one is 7. So, a long-enough variable need to be
C*defined to receive the string, and it's need to be processed so
C*that there's no SPACE characters in it before it is converted
C*to a numeric value.
C     'ocx2'        getatr    'Parameter'   param
C                   EVAL      PTR_Value = 0
C                   MOVE      PARAM         PTR_Value
C*Change the value of 'bstrValParam' member of the structure.
C                   eval      strFour='four'+x'00'
C                   eval      NewBSTRVal=strFour
```

この方法によって、最初のパラメーターで指し示される構造の内容を変更できます。上記のコードによって、ダイスには 4 つの点が表示されます。

アニメーション制御

Windows アプリケーションでは、アニメーション・コントロール・パーツが AVI 拡張子のビデオ・ファイルを再生します。このパーツは、ビデオが実際にはプログラム・ロジックとは独立に再生されるという点で、メディア・パーツとは異なります。このパーツの典型的な使用法の 1 つは、ファイルがあるフォルダーから別のフォルダーに移動中であるといった何らかの進行を示す AVI ファイルを表示することです。

アニメーション制御パーツは、音声を伴わずにビデオ・ファイルを再生します。AVI ファイルは、RLE (ラン・レングス・エンコード) 方式で圧縮されていない限り、圧縮形式にすることはできません。

Java アプリケーションでは、アニメーション・コントロール・パーツは、**NbrOfImage** 属性を使用して、アニメーション GIF ファイル・シーケンスを再生するために使用されます。

パーツ属性

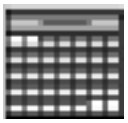
FileName	FrameRate	Handle*	Left
Mode	NbrOfImage	ParentName	PartName
PartType	Top	UserData	Visible

***注:** 制約事項については、属性の説明を参照してください。

適用可能なイベント

Create	Destroy

カレンダー



カレンダー・パーツは、月間カレンダーを表します。ユーザーは、月矢印の 1 つをクリックし、次の月または直前の月に行くことによって、カレンダーをナビゲートすることができます。

また、特定の日付に行ったり、ユーザーが選択した日付を決定したり、短いテキスト・コメントをカレンダーの個々の日に追加したりするなど、プログラムによってカレンダーを完全に制御することもできます。

パーツ属性

Border	Bottom	ClearAll	ClearDate
ClearMonth	ClearYear	Color	ColorArea
ColorMix	Date	DateIdx	DateText
DateUnder	Day	DayIdx	DayLen
DayNumPos	DayNumRect	DayStart	Enabled
FontArea	FontBold	FontItalic	FontName
FontSize	FontStrike*	FontUnder*	FrmtString
Handle*	Height	HRule	Left
Month	MonthArrow	MonthIdx	MonthLen
OutlineRcl	ParentName	PartName	PartType
Refresh	ShowRects	ShowText	TipText
Top	UserData	Visible	VRule
WeekDay	WeekDayIdx	Width	Year
YearIdx	YearLen		

*** 注:** 制約事項については、属性の説明を参照してください。

適用可能なイベント

Click	Create	Destroy	DblClick
MouseDown	MouseEnter	MouseExit	MouseMove
MouseUp	MthChange	YearChange	

ユーザーが選択した日付の決定

DateUnder 属性を使用して、マウス・ポインターが指す日付を決定することができます。次の例では、ユーザーがカレンダーをクリックした時に、日付が検索されます。**DateUnder** 値がブランクであるかどうかを検査していることに、注意してください。これは、マウスがどこにあってもクリック・イベントが発生しないためです。マウスが特定の日付を指していない場合には、**DateUnder** 属性はブランクになります。

また、日付は YYYYMMDD の形式の文字ストリングとして戻されることにも注意してください。

```
C     'Cal1'        Getatr    'DateUnder'   YYYYMMDDA         8
*
 * マウスが日を指している場合...
C                   If        YYYYMMDDA <> *Blanks
 *
 * 日付を数字にする
C                   Move      YYYYMMDDA     YYYYMMDD          8 0
 *
 * 日付を処理する
 *                  ...
 *                  ...
C                   EndIf
 *
```

図 13. カレンダー・パーツの例

日付索引属性の使用

現在表示されているものに影響することなくカレンダーにアクセスできるようにする属性がいくつか用意されています。この例では、カレンダー内の特定の日にコメントが追加されます。

```
 *
 * 索引を参照する日付に設定
C     'Cal1'        Setatr    '19971210'    'DateIdx'
 *
 * その日についてのコメントを設定
C     'Cal1'        Setatr    'Vacation'    'DateText'
 *
```

図 14. コメントの追加

キャンバス

ウィンドウまたはノートブック・ページ上に複数のパーツを入れたい場合には、ウィンドウまたはノートブック・ページでキャンバス・パーツを使用してください。キャンバス上の各種のパーツを指示してクリックし、それらを配置してから編成して、グラフィカル・ユーザー・インターフェースを作成することができます。

キャンバス・パーツは、ウィンドウまたはノートブック・ページのどちらかのクライアント域を占めます。ウィンドウまたはノートブック・ページにキャンバスがない場合には、パーツがメニュー・バーやメッセージ・サブファイルなどのウィンドウの拡張機能でない限り、クライアント域には 1 つのパーツしか入れることができません。

たいていの場合、複数のパーツを持つウィンドウおよびノートブック・ページを作成します。その場合、ユーザーはキャンバス・パーツを持つノートブック・ページおよびキャンバス・パーツを持つウィンドウを使用する必要があります。これらはすでにキャンバス・パーツを含んでいるため、ユーザーのステップを省くことができます。

ビルド時に、**FileName** 属性を指定して、ビットマップ・イメージをキャンバスの背景として組み込むこともできます。このイメージは、**Tile** 属性を設定することによってタイル表示することができます。Java アプリケーションの場合には、GIF または JPG イメージを背景として組み込むことができます。

注:

1. キャンバス・パーツ、ウィンドウ (キャンバスなし) パーツ、およびノートブック・ページ (キャンバスなし) パーツは、パーツ・パレットではなくパーツ・カタログに入っています。頻繁に使用したい場合には、それらをパーツ・パレットに追加することができます。
2. キャンバス・パーツ上のパーツにデフォルトのフォントの設定 (デフォルトのシステム・フォント) がある場合には、それらはキャンバス・パーツに指定されたフォントの定義を継承します。キャンバス上の大多数のパーツにフォントを適用するには、個々のパーツごとのフォントではなく、キャンバス・パーツのフォントを指定してください。

関連情報については、以下を参照してください。

- 192 ページの『ウィンドウ』
- 193 ページの『キャンバス付きウィンドウ』
- 125 ページの『ノートブック・ページ』
- 127 ページの『キャンバス付きノートブック・ページ』.

パーツ属性

BackColor	BackMix	Bottom*	Enabled
FileName	FontBold	FontItalic	FontName
FontSize	FontStrike*	FontUnder*	Handle*
Height	Left	ParentName	PartName
PartType	ReadOnly	Refresh	Tile
Top*	UserData	Width	

***注:** 制約事項については、属性の説明を参照してください。

適用可能なイベント

Click	Create	DblClick	Destroy
MouseDown	MouseMove	MouseUp	Popup
VKeyPress			

チェック・ボックス

たとえばオンとオフのように、明確に区別できる 2 つの状態の間でユーザーに選択させたい場合には、チェック・ボックス・パーツを使用してください。

チェック・ボックスに関連したラベルには、選択された時の設定が記述されます。

一般的に言えば、オプションのリストを提供するため、チェック・ボックスのグループを使用することができます。ユーザーは、1 つまたは複数のチェック・ボックスを選択することもできるし、あるいは何も選択しないことも可能です。オプションは互いに排他的でなく、したがって、1 つのチェック・ボックスの選択が、ウィンドウ上のその他のチェック・ボックスに影響しません。ユーザーが 2 つ以上のオプションから 1 つだけ選択できるようにしたい場合には、チェック・ボックスの代わりにラジオ・ボタンを使用してください。詳細については、151 ページの『ラジオ・ボタン』を参照してください。

チェック・ボックスの状態を設定するには、ユーザーはマウスでチェック・ボックスをクリックするか、チェック・ボックスがフォーカスされている時にキーボードのスペース・キーを押すか、または (割り当てられている場合には) 簡略記号キーを押すことができます。

パーツ属性

AddLink*	AllowLink*	BackColor	BackMix
Bottom	Checked	Enabled	Focus
FontBold	FontItalic	FontName	FontSize
FontStrike*	FontUnder*	ForeColor	ForeMix
Handle*	Height	Highlight*	Label
Left	ParentName	PartName	PartType
Refresh	RemoveLink*	ShowTips	TipText
Top	UserData	Visible	Width

***注:** 制約事項については、属性の説明を参照してください。

適用可能なイベント

Create	Destroy	MouseEnter	MouseExit
MouseMove	Popup	Select	

チェック・ボックス・パーツの状態の設定

チェック・ボックスの状態を記述するためには、**Checked** 属性を次の値の 1 つに設定してください。

1　　チェック・ボックスがセットされて状態がオンになります。

0　　チェック・ボックスはセットされず状態はオフです。

チェック・ボックスには、その状態がオンの時にはチェック・マークが入ります。

簡略記号の設定

注: 簡略記号は Java アプリケーションではサポートされていません。

チェック・ボックスの簡略記号キーを指定するためには、チェック・ボックスのラベルの文字の前に簡略記号 ID を入れてください。Windows では、アンパーサンド (&) を使用してください。この文字は簡略記号キーで、インターフェース上に下線と一緒に表示されます (たとえば、**V**isible)。下線はキーボードで下線の付いた文字を押してチェック・ボックスを選択できることを知らせます。

イベントの通知

ユーザーがチェック・ボックスを選択して状態をオンまたはオフにすると、**Select** イベントが通知されます。

組み合わせボックス

組み合わせボックスは、特定の情報を入力するオプションまたは通常使用される選択項目のリストから選択するオプションをユーザーに提供します。

これは、ユーザーがスクロールできる値のリストを示すリスト・ボックスのついた入力フィールドで構成されます。ユーザーがこれらの値の 1 つを選択すると、その値が入力フィールドに表示され、既存のテキストと置き換えられます。これに代わる方法として、ユーザーは入力フィールドに値を直接入力することができますが、この値はリストのどの値とも一致している必要はありません。

組み合わせボックスには、いろいろなスタイルがあります。パーツのプロパティー・ノートブックから必要なスタイルを選択することができます。

関連情報については、以下を参照してください。
- 99 ページの『リスト・ボックス』
- 81 ページの『入力フィールド』

パーツ属性

AddItemEnd	AutoScroll*	BackColor	BackMix
Bottom	Case*	Count	DelimChar
DeSelect*	DragEnable*	DropEnable*	Enabled
FieldExit	FirstSel	Focus	FontBold
FontItalic	FontName	FontSize	FontStrike*
FontUnder*	ForeColor	ForeMix	GetItem
Handle*	Height	Index	InsertItem*
ItemKey	Left	ParentName	PartName
PartType	ReadOnly	Refresh	RemoveItem
Search*	SearchType*	Selected	SelectItem
Sequence*	SetItem	SetTop*	Showtips
SizeToFit*	Text	TipText	Top
UseDelim	UserData	Visible	Width

***注:** 制約事項については、属性の説明を参照してください。

適用可能なイベント

Change	Create	Destroy	Drop*
DropDown*	Enter	GotFocus	KeyPress
LostFocus	MouseEnter	MouseExit	MouseMove
Popup	Select	VKeyPress	

***注:** 制約事項については、イベント記述を参照してください。

組み合わせボックスのタイプの選択

組み合わせボックス・プロパティー・ノートブックで、作成したい組み合わせボックスのタイプを選択してください。

組み合わせボックス
組み合わせボックスの入力フィールド部分とリスト・ボックス部分の両方を表示します。

ドロップダウン組み合わせボックス
入力フィールド部分を表示します。リスト・ボックス部分は、下矢印 (↓) が押されるまで隠されています。

ドロップダウン・リスト組み合わせボックス
入力フィールド部分を表示します。リスト・ボックス部分は、下矢印 (↓) が押されるまで隠されています。入力フィールドはテキストを受け入れません。リスト・ボックス部分からの選択項目を表示するために使用されるだけです。

注: すべてのタイプの組み合わせボックスは、入力フィールド部分からのドラッグをサポートします。簡単な組み合わせボックスは、リスト・ボックス部分からのドラッグをサポートするだけです。すべてのタイプがリスト・ボックス部分でなく入力フィールド部分へのドロップをサポートします。

項目の初期順序の追加および設定

組み合わせボックス・プロパティー・ノートブックを使用して、設計時に組み合わせボックスに項目の初期リストを入れることができます。

デフォルトでは、項目はそれらを追加した順序で組み合わせボックスに表示されます。それらをより正確な順序で表示したい場合には、追加を開始する前に **Sequence** 属性を昇順、降順、または索引のいずれかに設定してください。これで、項目は追加された時の ASCII 照合順序でソートされます。

Sequence 属性を使用して、すでに組み合わせボックスにある項目の順序を変更することはできません。

実行時の項目の追加

実行時に、**InsertItem** 属性を使用して、組み合わせボックスに項目を挿入することができます。項目が表示される順序は、 **Sequence** 属性によって決定されます。

リストの項目の更新

すでにリストにある項目を更新することができます。**Index** 属性を使用して変更したい項目を指示し、**SetItem** 属性を使用して変更済みデータを設定します。

注: **SetItem** 属性は、**Sequence** 属性の設定値に関係なく、項目をリストの元の場所に置き換えます。たとえば、組み合わせボックスに入れる前に **Sequence** 属性を昇順に設定している場合には、項目は組み合わせボックスに昇順で表示されますが、項目を検索してその値を変更し、**SetItem** 属性を使用して組み合わせボックスで項目を置き換えると、その項目は前にあった位置と同じ位置に挿入されます。したがって、変更後にはリストが昇順になることもあれば、ならないこともあります。

リストの最上部の設定

SetTop 属性は、組み合わせボックスの最上部にどのリスト項目を表示するかを指定するために使用します。この項目を設定しても、リストの項目は再順序付け**されません。**選択した項目が最上部に表示され、続いてその後の項目が表示されるようにリストがスクロールされます。

項目の除去

除去したい項目の **RemoveItem** 属性および **Index** 値を使用してください。**Index** 値は *1* から始まります。リストから項目が除去されると、除去された項目の後のすべての項目がリスト内で 1 位置だけ上に移動します。

リストのすべての項目を除去するためには、**Index** 値の *0* を指定してください。

項目の選択および選択解除

ユーザーはマウスまたはキーボードを使用して項目を選択したり選択解除することができます。プログラムの **Selected** および **DeSelect** 属性を使用して項目を選択したり選択解除することができます。これらの属性を使用する前に、**Index** 属性を設定してください。

ユーザー選択項目の検索

組み合わせボックスのリストから項目を選択すると、その項目が入力フィールドに入れられます。 **Text** 属性を使用してこの項目を取り出すことができます。また、**FirstSel** 属性を使用して、選択した項目の索引を決定することもできます。

ユーザーは組み合わせボックスの入力フィールド部分に値を入力することもできます。この値はリストの値の 1 つである必要はありません。ユーザーがリストにある項目しか選択できないようにしたい場合には、**ReadOnly** 属性を *0* に設定してください。

Count 属性を使用して、検索する項目があるかどうかを確認することができます。

キーの使用法

リスト・ボックスおよび組み合わせボックスの両方で、「キー」部分と「データ」部分から成る項目をリストに追加することができます。項目をリストに追加すると、その項目のデータ部分だけが表示されます。ユーザーが項目を選択すると、その項目のキー部分をプログラム式に検索することができます。

リスト内のキーを使用可能にするためには、パーツ設定ノートブックの ’区切り文字’ ページの ’区切り文字の使用’ チェック・ボックスをチェックし、区切り文字を指定する必要があります。デフォルトの区切り文字はセミコロン (;) です。リスト内の項目は、キー部分、区切り文字、データ部分の順になっています。たとえば、以下のようになります。

```
01;Shipping
```

1 例として、ユーザーが選択できるリストに部門のリストを表示させたいとします。データベースには、部門が 2 文字のフィールドとして格納されていますが、ユーザーには記述名が表示されるようにしたいとします。次のデータをリストに追加することになります。

- 01;Shipping
- 02;Manufacturing
- 03;Payroll
- 04;Distribution

注: 組み合わせボックスを使い、設定ノートブックのデータ・ページでデフォルト・リストを追加することができます。

ユーザーがそのリストから選択を行なうと、次のコードを使用して、当該項目のキー部分を取得することができます。

```
C     'Combo1'      Getatr    'FirstSel'    X                2 0
C     'Combo1'      Setatr    X             'Index'
C     'Combo1'      Getatr    'ItemKey'     Key
```

入力フィールドのテキストの設定

組み合わせボックスが最初に表示された時には、その入力フィールドはブランクです。入力フィールドにリスト項目の 1 つを入れたい場合には、**SelectItem** 属性の値を、使用する項目の索引で設定してください。

イベントの通知

Select イベントは次の場合に通知されます。

- ユーザーが組み合わせボックスにある項目を選択した。
- ユーザーがプログラムのリストにある項目を選択する。
- ユーザーがすでに選択されている項目を選択する。

Enter イベントは次の場合に通知されます。

- ユーザーが組み合わせボックスにある項目をダブルクリックした。
- リスト・ボックスにフォーカスされて項目が選択されている時に Enter キーが押された。

これらのイベントのアクション・サブルーチンでは、**Selected** 属性を使用してどの項目が選択されたかを確認することができます。

コンポーネント参照

コンポーネント参照パーツは、1 つの VARPG コンポーネントを他のコンポーネントと通信できるようにします。コンポーネント参照パーツを使用して、他のコンポーネント上のパーツに作用します。参照されるコンポーネントは、コンポーネント参照パーツとして同じアプリケーションで実行中でなければなりません。

コンポーネント参照パーツは、また他のコンポーネントの指定されたイベントをモニターします。モニター・イベントが現れると、コンポーネント参照パーツによって **Notify** イベントが通知されます。

パーツ属性

AddSrcEvt	AttrValue	Bottom	CompName
Left	NotSrcEvt	NotSrcPart	NotSrcWin
ParentName	PartName	PartType	RefAttr
RefParent	RefPart	RmvSrcEvt	UserData
Visible			

適用可能なイベント

Create	Destroy	Notify

他のコンポーネントのパーツ属性の参照

別のコンポーネントのパーツの属性を参照するために使用できる方法は次の 2 つです。

1. コンポーネント参照パーツのプロパティー・ノートブックで属性を定義します。
2. 実行時に適切なコンポーネント参照パーツ属性を設定します。

別のコンポーネントのパーツ属性を参照する前に、他のコンポーネントが実行中であることを確認する必要があります。START 命令コードを使用して別のコンポーネントを始動してください。

次のコード部分は、1 つのコンポーネントのコンポーネント参照パーツがどのようにして別のコンポーネントのパーツ属性の値を変更するかを示しています。この例では、コンポーネント COMPB のウィンドウ WIN01 のイメージ・パーツ (IMG1) の **FileName** 属性が新しい値に更新されます。

```
 *
 * Change the bitmap for image part IMG1 on
 * window WIN01 in Component COMPB
C     'CR1'         Setatr    'COMPB'       'CompName'
C     'CR1'         Setatr    'WIN01'       'RefParent'
C     'CR1'         Setatr    'IMG1'        'RefPart'
C     'CR1'         Setatr    'FILENAME'    'RefAttr'
C     'CR1'         Setatr    'D:\PIC.BMP'  'AttrValue'
 *
```

別のコンポーネントのイベントのモニター

1 つのコンポーネントのコンポーネント参照パーツを使用して、同じアプリケーションで実行中の別のコンポーネントに現れるイベントをモニターすることができます。コンポーネント参照パーツのプロパティー・ノートブックでモニターするイベントを定義するか、または実行時に適切な属性を設定して行います。他のコンポーネントにモニターするイベントが現れると、コンポーネント参照パーツによって**通知**イベントが通知されます。

次のコード部分は、別のコンポーネントのイベントをモニターするために実行時にコンポーネント参照パーツをどのように設定することができるかを示しています。この例では、モニターするイベントは、コンポーネント COMPB のウィンドウ WIN01 の PB1 と呼ばれるプッシュボタンを**押す**イベントです。

```
 *
 * Monitor for the PRESS event of push button
 * PB1 on window WIN01 in Component COMPB
C     'CR1'         Setatr    'COMPB'       'CompName'
C     'CR1'         Setatr    'PRESS'       'NotSrcEvt'
C     'CR1'         Setatr    'PB1'         'NotSrcPart'
C     'CR1'         Setatr    'WIN01'       'NotSrcWin'
 *
```

コンテナー

コンテナー・パーツを使用して、関連したレコードを保管します。レコードは、アイコン・ビュー、ツリー・ビュー、テキスト・ツリー・ビュー、または詳細ビューに表示することができます。

パーツ属性

AddRcd	Arrange	BackColor	BackMix
BlankChar	Bottom	ChildCount	ChildList
Collapsed	ColNumber	Count	DeleteRcd
DeSelect	EditItem	Enabled	ExtSelect*
FirstSel	Focus	FontBold	FontItalic
FontName	FontSize	FontStrike*	FontUnder*
ForeColor	ForeMix	GetNewID	GetRcdFld
GetRcdIcon	GetRcdText	Handle*	Height
InUse*	Label	Left	MiniIcon
ParentId	ParentList	ParentName	PartName
PartType	ReadOnly	RecordID	Refresh
RemoveRcd	Selected	SelectRcd	SetRcdFld
SetRcdIcon	SetRcdText	SetTop*	ShowTips
SortAsc	SortDesc	TipText	Top
UserData	View*	Visible	VisTitle
VisTitlSep	Width		

***注:** 制約事項については、属性の説明を参照してください。

適用可能なイベント

Click	Collapsed	ColSelect	Create
DblClick	Destroy	Enter	Expanded
GotFocus	KeyPress	LostFocus	MouseDown
MouseEnter	MouseExit	MouseMove	MouseUp
Popup	Select	VKeyPress	

コンテナーへの列の追加

詳細ビューでは、レコードはコンテナー・パーツの行に対応し、その行のそれぞれの列はそのレコードのフィールドに対応します。コンテナーにレコードを追加する前に、コンテナー・パーツのプロパティー・ノートブックを使用して、フィールドを表示するために必要な列を追加する必要があります。

次の 4 つのタイプの列のどれを作成したいかを指定する必要があります。

オブジェクト・テキスト

オブジェクト・テキスト列には、レコードの作成時に指定された記述テキストが表示されます。このテキストは、実行時に **SetRcdText** 属性で変更することができます。 ユーザーは実行時に Alt キーを押し、マウスでフィー

ルドを選択してこのテキストを変更することができます。この列の値を取り出すためには、**GetRcdText** 属性を使用してください。

オブジェクト・アイコン
オブジェクト・アイコン列には、レコードの作成時に指定されたアイコン・ファイルが表示されます。アイコン・ファイル名は、**SetRcdIcon** 属性を使用して、実行時に変更することができます。

Text テキストは、追加の情報が入っているストリングです。テキスト列には任意のストリング値を入れることができます。テキスト中にブランクを入れることはできません。実行時にブランクを表示したい場合には、ストリング中に下線 (_) 文字を使用してください。実行時にこのテキストを変更するためには、**SetRcdFld** 属性を使用してください。ユーザーは実行時に Alt キーを押し、マウスでフィールドを選択してテキストを変更することができます。この列の値を取り出すためには、**GetRcdFld** 属性を使用してください。

アイコン
アイコンは、追加のグラフィカル・レコード情報を表示します。アイコン列には、アイコン・ファイルが表示されます。実行時にアイコン・ファイル名を変更するためには、**SetRcdIcon** を使用してください。

コンテナー・パーツには最大 20 列まで追加することができます。これらのうち最大 15 までをオブジェクト・テキストとテキスト列の組み合わせにし、最大 5 までをオブジェクト・アイコン列およびアイコン列とすることができます。Windows のもとでは、最初の列にしかアイコンを入れられないことに注意してください。

実行時に列の数や列タイプを変更することはできません。コンテナーの列の数より多いフィールドを持つレコードを追加すると、余分のフィールドは無視されます。

コンテナーへのレコードの追加

コード中で、**AddRcd** 属性を使用してコンテナーにレコードを追加します。この属性は次のストリングから構成されます。

```
ID Text FileName ParentID {field_data field_data ...}
```

ブランクは区切り文字として使用されます。パラメーターは次の通りです。

ID レコードに指定する固有の数値。この数字はゼロより大きくなければなりません。作成するそれぞれのレコードに固有の ID を割り当てるためには、**GetNewID** 属性を使用してください。

Text オブジェクト・アイコンに表示されるテキスト。このテキストは実行時に **SetRcdText** 属性を使用して変更することができます。

FileName
オブジェクト・アイコン列のアイコン・イメージが入っているファイルの名前。このアイコンはコンテナーのアイコンおよびツリー・ビューに表示されます。アイコン・ファイル名は、**SetRcdIcon** 属性を使用して、実行時に変更することができます。

ParentID
このレコードが現れるレコードの固有の ID。このレコードに親レコードがない場合には、このフィールドに *0* を入れてください。

Field_data

テキストまたはアイコン列に表示されるレコードの追加の情報。それぞれの *field_data* 値は、コンテナー・パーツの対応する列を更新します。空のテキスト列またはアイコン・コンテナー列が必要な場合には、下線 (_) を、対応する *field_data* パラメーターに指定する必要があります。

次のコード部分は、サンプル・レコードをコンテナーに追加するために指定するパラメーターを示します。この例では列データは追加されません。

```
 *
 * This is not a child record
C                   Eval      Parent = '0'
 *
 * Use the icon text specified in the GUI designer
C                   Eval      IconText = '_'
 *
 * Set the icon file name to be used for this record
C                   Eval      IconFile = '.\\TOM.ICO'
 *
 * Get a new container record ID and make it character
C     'CT1'         Getatr    'GetNewId'    NextIDN           6 0
C                   Move      NextIDN       NextID            6
 *
 * Create the container record structure
C                   Eval      NextRcd = NextID + ' ' +
C                                       IconText + ' ' +
C                                       IconFile + ' ' +
C                                       Parent
 *
 * Add the record to the container
C     'CT1'         Setatr    NextRcd       'AddRcd'
 *
```

Count 属性は、コンテナーにどれだけ多くのレコードがあるかを調べるために使用します。

コンテナー列の更新

コンテナーにレコードを追加すると、レコード・フィールドのデータがコンテナーの対応する列に表示されます。個々のコンテナーのテキスト列のデータは、レコード・フィールドを更新することによって更新することができます。

注: 更新できるのは列のデータだけです。コンテナーの列の数を変更することはできません。列の数は、GUI Designer でコンテナーを作成する時に設定されます。

列を更新するためには、**RecordID** 属性をその列に対応するレコードに設定して、**ColNumber** 属性を更新済みデータが入るそのレコードのフィールドに設定します。次のコード部分は、コンテナーの 3 番目の目列の更新方法を示しています。

```
 *
 * Set the record id to be referenced
C     'CT1'         Setatr    NextIDN       'RecordID'
 *
 * Reference the third column in the record
C     'CT1'         Setatr    3             'ColNumber'
 *
 * Update the column with the new data
C     'CT1'         Setatr    'Larry'       'SetRcdFld'
 *
```

新しい列の値に組み込みブランクを入れたい場合には、それぞれのブランクを表すために下線文字を使用してください。下線文字は、列の更新時にブランクと置き換えられます。たとえば、次のようになります。

```
 *
 * 'New data' is set in the column
C     'CT1'         Setatr    'New_data'    'SetRcdFld'
 *
```

GetRcdFld 属性は、レコード・フィールドの内容を検索するために使用します。**RecordID** 属性をレコードの固有の ID に設定して、**ColNumber** 属性を必要な列の数に設定してください。

コンテナーからのレコードの除去

RemoveRcd 属性は、コンテナー・パーツからレコードを除去して、そのレコードを固有に識別するレコード ID を除去するために使用します。コンテナーのすべてのレコードを除去するためには、レコード ID 値をゼロに設定してください。

次のコード部分は、コンテナーからレコードを除去する方法を示しています。

```
 *
 * Get ID of first selected record
C     'CT1'         Getatr    'FirstSel'    TmpID             6 0
 *
 * If a record was selected, remove it from the container
C                   If        TmpID <> 0
C     'CT1'         Setatr    TmpID         'RemoveRcd'
C                   EndIf
 *
```

コンテナー・ビューの変更

ビューを変更するためには、**View** 属性を使用してください。コンテナー・パーツは、アイコン、ツリー・アイコン、ツリー・テキスト、および詳細のデータ・ビューを表示できます。

アイコン・ビュー

アイコン・ビューには、アイコンとその下のテキストで表されるそれぞれのレコードがあります。アイコン・ビューには子レコードは表示されません。コンテナーにレコードを追加する時には、レコード構造にアイコン・ファイル名と記述テキストを指定します。

実行時に **SetRcdIcon** および **SetRcdText** 属性を使用して、アイコンおよびアイコン・テキストを変更することができます。

コンテナーの各行にアイコンを表示するためには、**Arrange** 属性を *1* に設定してください。

ミニアイコンを使用するためには、**MiniIcon** 属性を *1* に設定するか、あるいはプロパティー・ノートブックの 'スタイル' ページのミニアイコン・ボックスをチェックしてください。

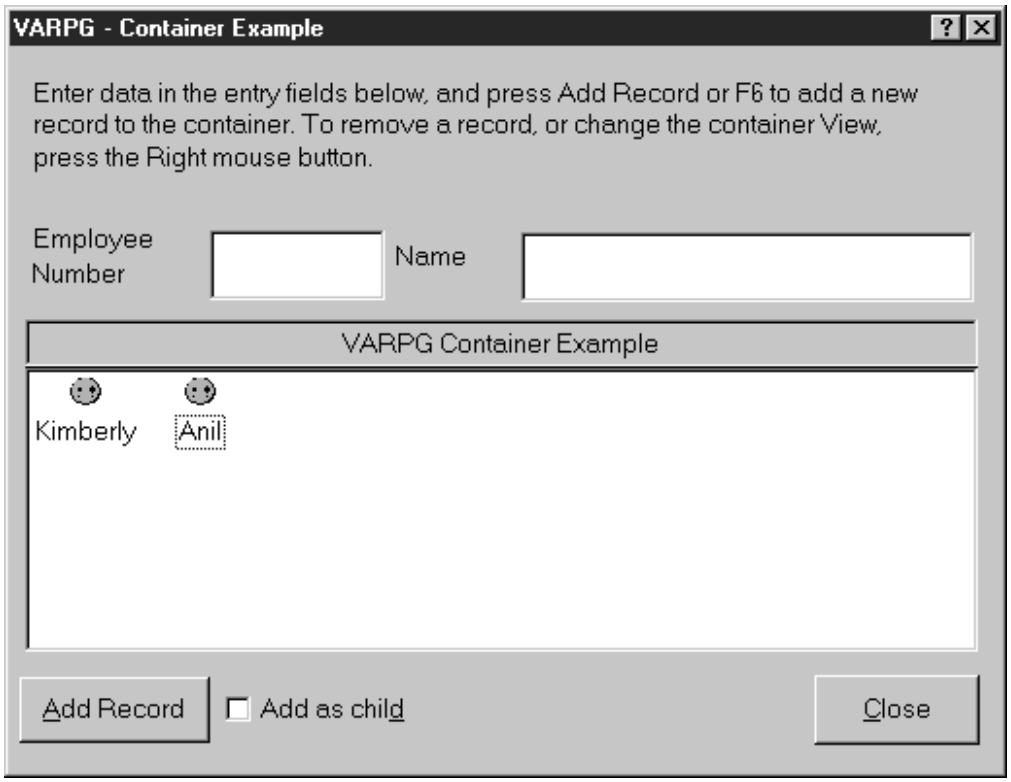


図 15. サンプル・アイコン・ビュー

ツリー・ビュー

ツリー・ビューでは、レコードは階層で表示されます。**ツリー・アイコン**/ビューには、そのアイコンとその横のアイコン・テキストでそれぞれのレコードが表示されます。レコードに子レコードがある場合には、そのアイコンの横にプラス符号が表示されます。プラス符号を選択すると、このレコードに関連したすべてのレコードが表示されます。 **ツリー・テキスト**・ビューには、テキスト専用モードの場合を除き、ツリー・アイコン・ビューと同様な方法で、アイコンなしでレコードが表示されます。

関連レコードの間には、それらの関係を示す接続線が引かれます。

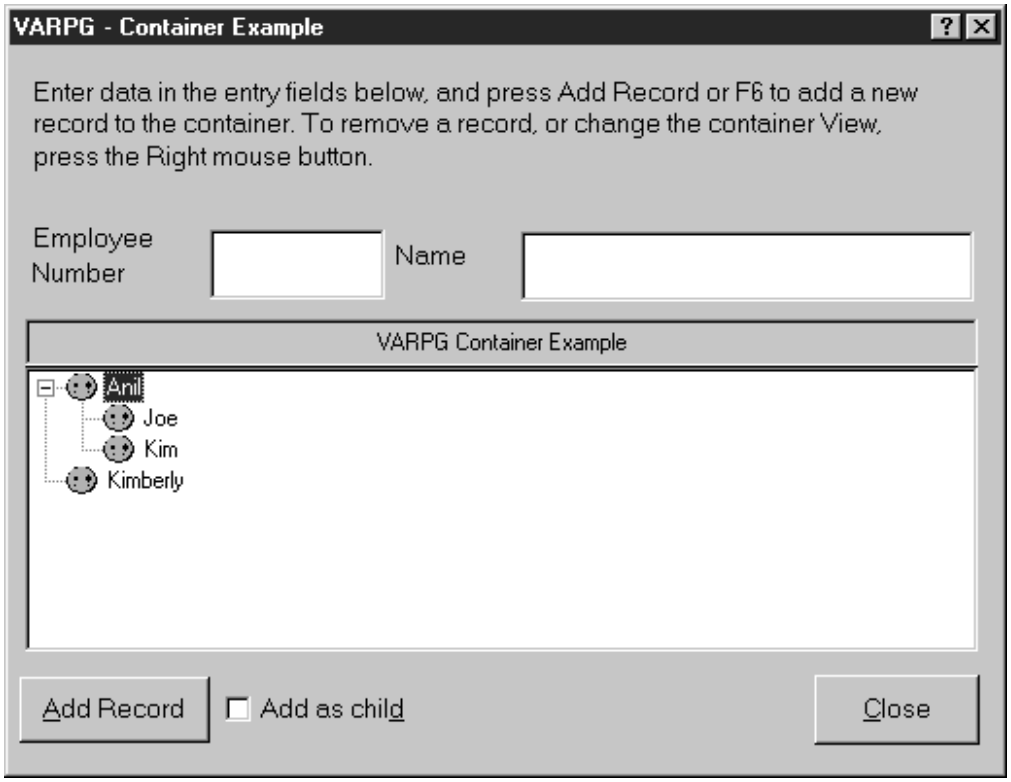


図 16. サンプル・ツリー・ビュー

詳細ビュー

詳細ビューでは、レコードはそれぞれの列を 1 つずつ表示して (サブファイルと同様に) 示されます。このタイプのビューには、子レコードは表示されません。

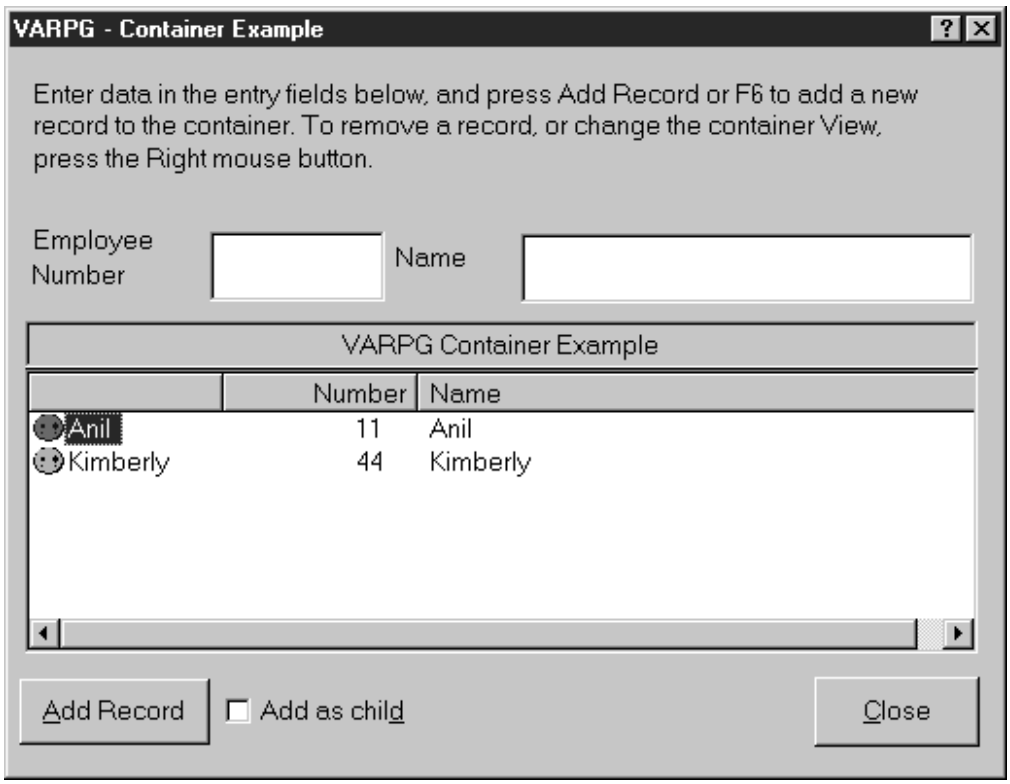


図 17. サンプル詳細ビュー

コンテナーがすべてのレコードを一度に表示できるほど大きくない場合には、スクロール・バーが自動的に追加されます。

ビューを変更するためには、**View** 属性を使用してください。

ミニアイコン

このオプションによって、プログラマーはコンテナー・パーツに入っているアイコンを通常のアイコンとして表示するかミニアイコンとして表示するかを指定することができます。これはアイコン・ビューに影響を与えるだけで、ツリー・ビューや詳細ビューは変更されません。

DDE クライアント

*** 制約事項:** このパーツは Java アプリケーションではサポートされていません。

DDE クライアントは、動的データ交換 (DDE) プロトコルをサポートする他のアプリケーション (スプレッドシート・アプリケーションなど) とデータを交換するために使用します。

交換は **DDE 会話**と呼ばれます。会話を開始するアプリケーションが **クライアント**で、これに応答するアプリケーションが**サーバー** です。あるアプリケーションが DDE をサポートしているかどうかを調べるには、そのアプリケーションに付属の資料を参照してください。

DDE クライアント・パーツは**コールド・リンク**と**ホット・リンク会話**の両方をサポートします。コールド・リンク会話は、サーバー・プログラムに対する明示的要求を作成するクライアント・プログラムからなっています。ホット・リンク会話は、データが変更される時に自動的にクライアント・プログラムを更新するサーバー・プログラムからなっています。

コールド・リンクまたはホット・リンク会話は、DDE クライアント・パーツのプロパティー・ノートブックからユーザーのプログラム・ロジックで構成することができます。

パーツ属性

AppName	Bottom	DDEAddLink	DDEMode
DDERmvLink	Execute	Format	Item
Left	ParentName	PartName	PartType
Poke	Request	TimeOut	Top
Topic	UserData	Visible	

適用可能なイベント

Create	Data	Destroy	ExecuteAck
PokeAck	Terminate	TimeOut	

入力フィールド

予測できないなにかの値をユーザーに入力させたい場合には、入力フィールド・パーツを使用します。入力フィールドとは、ユーザーがテキストを入力するか、あるいは入れることができる領域のことです。通常、その境界は指示されます。現在表示されているよりも多くの情報がある場合には、ユーザーは入力フィールドのテキストをスクロールすることができます。

ユーザーは、文字、数字、あるいは 2 バイト文字セット (DBCS) データを受け入れるように、入力フィールド・パーツを構成することができます。

この入力フィールドを読み取り専用にして、ユーザーによって直接変更できない情報を含めることができます。

パーツ・パレットの入力フィールド・パーツを指示してクリックしてから、サブファイル・パーツ上のそれをクリックして、サブファイル入力フィールドを作成することができます。

パーツ属性

AddLink*	Alignment	AllowLink*	AutoScroll*
AutoSelect	BackColor	BackMix	Bottom
CapsLock	Copy	CsrAtEnd	Cut
DataType	Delete	DragEnable*	DropEnable*
Enabled	FieldExit	Focus	FontBold
FontItalic	FontName	FontSize	FontStrike*
FontUnder*	ForeColor	ForeMix	Handle*
Height	InsertMode*	Left	Masked
ParentName	PartName	PartType	Paste
ReadOnly	Refresh	RemoveLink*	ShowTips
Text	TextEnd	TextLength	TextSelect
TextStart	TipText	Top	UserData
Visible	Width		

***注:** 制約事項については、属性の説明を参照してください。

適用可能なイベント

Change	Click	Create	DblClick
Destroy	Drop	GotFocus	KeyPress
Link*	LostFocus	MouseDown	MouseEnter
MouseExit	MouseMove	MouseUp	Popup
VKeyPress			

***注:** 制約事項については、イベント記述を参照してください。

InsertMode 属性の使用法

Windows では、挿入モードは常にオンです。オフにすることはできません。

Text 属性の使用法

Text 属性は、入力フィールドの値を取得または設定するために使用します。このために使用するフィールドは、入力フィールドと同じタイプで定義しなければなりません。たとえば、数値入力フィールドの値を取り出す場合には、その値を受け取るフィールドも数値として定義しなければなりません。

ウィンドウの情報の取得および設定

コンパイル中に、コンパイラーはユーザー・プログラムのフィールドを入力フィールド・パーツと同じ名前で、同じデータ・タイプと長さで明示的に定義します。演算項目 2 にウィンドウ名を指定して READ および WRITE 命令コードを使用することにより、**Text** 属性の値がこれらのフィールドからあるいはこれらのフィールドに自動的にコピーされます。READ および WRITE 命令コードは、ユーザー・インターフェースに多数の入力フィールドがある場合、一連の get および set 属性を実行しなくても済むので非常に効果的です。

詳しくは、 29 ページの『第 3 章 パーツを用いたプログラミング』を参照してください。

妥当性検査

プロパティー・ノートブックを使用して、入力フィールドにユーザーが指定した基準を満たすデータだけを受け入れるように指定することができます。データが特定の値と一致するようにするためには、**比較**値を設定してください。データが事前定義値の範囲内に入るようにするためには、**範囲**の値を設定してください。

注: VisualAge RPG は妥当性検査に ASCII の照合順序を使用します。これは EBCDIC の照合順序を使用するサーバーとは異なります。したがって、システム間で結果が異なる場合があります。

妥当性検査を実行するためには、ウィンドウに **Validate** 属性を設定した少なくとも 1 つのプッシュボタンまたはグラフィック・プッシュボタンがなければなりません。このプッシュボタンを押すと、妥当性検査基準が定義されているそれぞれの入力フィールドに対して妥当性検査が実行されます。入力フィールドのいずれかがこの妥当性検査に失敗すると、メッセージ・ウィンドウが表示されて、press イベントは通知されません。

プロパティー・ノートブックの最上部にあるフィールドを使用して、妥当性検査が失敗した場合に表示されるメッセージを設定することもできます。表示するメッセージのテキストを入力するか、または対応するメッセージが表示される名前の付いたメッセージ・ファイル (*MSG0001 など) を入力します。このフィールドをブランクのままにした場合には、妥当性検査に失敗するとデフォルトのシステム・メッセージが表示されます。

注: このフィールドには 15 文字の制限があります。

ユーザー入力の阻止

ユーザーが入力フィールドにデータを入力するのを阻止するためには、次の 1 つを行なってください。

- **ReadOnly** 属性を *1* に設定します。この属性を設定しても、まだユーザー・プログラムの入力フィールドの値は変更することができます。
- **Enabled** 属性を *0* に設定します。この属性を設定すると、入力フィールドはイベントに応答しないので、ユーザーは入力フィールドにタブ移動することができません。

重要データのマスキング

入力フィールドにパスワードやアカウント番号などの重要データが入っている場合には、 **Masked** 属性を *1* に設定します。この属性を設定すると、入力フィールドの入力したそれぞれの文字に対してアスタリスク文字 (*) が表示されます。これは入力フィールドから読み取られる実際のデータには影響を与えません。

グラフ

グラフ・パーツにより、プロジェクト内でグラフを作成および設計することができます。実行時に、データをグラフに送り、グラフ属性およびグラフ・タイプを変更することができます。グラフ・パーツは、「円」、「線」、「棒」、および「線と棒」グラフ・タイプをサポートします。

棒グラフおよび折れ線グラフ・タイプは、**ToolTip** テキスト制御をサポートします。使用可能な場合には、データ・ポイント上にマウスを移動すると、ToolTip テキスト制御が表示されます。プログラム・ロジック内でこのサポートを使用するには、そのポイントの値を **TipText** 属性に設定して、そのグラフ・パーツが入っているウィンドウに **ShowTips** 属性を設定してください。

パーツ属性

AutoInc	BarLabel	Bottom	Color
ColorArea	ColorMix	DataGroup	DataPoint
DataValue	Enabled	FillStyle*	FontArea
FontBold	FontItalic	FontName	FontSize
FontStrike*	FontUnder*	GnEqGrpCol	GnEqPntCol
GraphType	GroupLabel	GrphHiLite	Handle*
Height	HitItem*	HlitPoints*	LabelPlace
Left	LegendType	ParentName	PartName
PartType	Refresh	StartNew	TipText
Title	TitlePlace	Top	UnderPoint*
UseData	UserData	Visible	Width
XAxisLabel	YAxisLabel	YInc	

***注:** 制約事項については、属性の説明を参照してください。

適用可能なイベント

Click	Create	DblClick	Destroy
MouseDown	MouseEnter	MouseExit	MouseMove
MouseUp	Popup		

データをグラフに送る

データをグラフに送る前に、値を受け取る **DataGroup** および **DataPoint** を指示する必要があります。 **DataPoint** 属性は、その値を表すグラフの位置エレメントを表します。棒グラフの場合には、これは棒になります。折れ線グラフの場合には点で、円グラフの場合にはスライスになります。 **DataGroup** は任意選択です。この属性は、グラフで表すデータのグループがあることを示します。デフォルトの **DataGroup** は 1 です。**DataValue** 属性を設定すると、選択されたエレメントの値が設定されます。

DataGroup の例として、与えられた年の各月の最高気温と最低気温をプロットしたいとします。このグラフは 2 つのデータ・グループを持つことになります。最初のグループはある月の最低気温を表し、2 番目のグループはその月の最高気温を表します。このグラフの場合には、まず **DataGroup** 属性を 1 に選択します。次に、ループ内で、各 **DataPoint** の **DataValue** を最低気温値に設定します。グラフを完成するために、**DataGroup** を 2 に設定することによって最高気温についても同じステップを繰り返します。

データをグラフに送っても、グラフは更新されません。データを表示するためには、**UseData** 属性を設定する必要があります。

次のコード・フラグメントは、どのようにすればこれを行なえるかを示しています。

```
 *
C                   Do        2             Group             2 0
C     'Graph1'      Setatr    Group         'DataGroup'
 *
C                   Do        12            Point             2 0
C     'Graph1'      Setatr    Point         'DataPoint'
 *
C                   If        Group = 1
C     'Graph1'      Setatr    Low(Point)    'DataValue'
 *
C                   Else
C     'Graph1'      Setatr    High(Point)   'DataValue'
C                   Endif
 *
C                   EndDo
 *
C                   EndDo
 *
C     'Graph1'      Setatr    1             'UseData'
```

図 18. データをグラフに送る

グラフが円グラフの場合には、各グループは別々の円として表されます。

グラフィック・プッシュボタン

頻繁に使用するアクションへの便利なアクセスを提供するためには、グラフィック・プッシュボタンを使用してください。

グラフィック・プッシュボタンはプッシュボタンと同じ機能を提供します。これは、ユーザーが選択すると開始されるアクションを示しますが、その機能を記述するラベルを表示する代わりにイメージを表示します。**FileName** 属性は、使用するイメージの名前を指定します。

有効な **Windows** イメージ形式には、次が含まれます。
- Windows および OS/2® ビットマップ (BMP、VGA、BGA、RLE、DIB、RL4、RL8)
- アイコン (ICO)
- Microsoft/Aldus タグ・イメージ・ファイル・フォーマット (TIF、TIFF)
- CompuServe グラフィック交換形式 (GIF)
- ZSoft PC Paintbrush イメージ・ファイル形式 (PCX)
- Truevision Targa/Vista ビットマップ (TGA、VST、AFI)
- Amiga インターリーブ・ビットマップ形式 (IFF、ILBM)
- X Windows ビットマップ (XBM)
- IBM プリンター・ページ・セグメント (PSE、PSEG、PSEG38PP、PSEG3820)
- Joint Photographic Experts Group 形式 (JPG, JPEG)

注: この製品の JPG/JPEG 形式に対するサポートは、部分的に Independent JPEG Group の働きに基づいています。

有効な **Java** イメージ形式には、次が含まれます。
- CompuServe グラフィック交換形式 (GIF)
- Joint Photographic Experts Group 形式 (JPG, JPEG)

関連情報については、 149 ページの『プッシュボタン』を参照してください。

パーツ属性

Bottom	Enabled	FileName	Focus
Handle*	Height	HelpEnable	HighLight
Left	ParentName	PartName	PartType
Refresh	ShowTips	TipText	Top
UserData	Validate	Visible	Width

*** 注:** 制約事項については、属性の説明を参照してください。

適用可能なイベント

Create	Destroy	GotFocus	LostFocus
MouseEnter	MouseExit	MouseMove	Popup
Press			

イメージの設定

グラフィック・プッシュボタンで表示されるイメージを設定するためには、**FileName** 属性を使用して、有効なビットマップ (.BMP) またはアイコン (.ICO) ファイル名を指定してください。適切な実行時ディレクトリーにシステム特有のビットマップおよびアイコン・ファイルを保管しなければなりません。詳細については、255 ページの『第 12 章 ピクチャー、サウンド、およびビデオ・ファイルの使用』を参照してください。

コマンド・キーの割り当て

グラフィック・プッシュボタンにコマンド・キーを割り当てることができます。そのためには、プロパティー・ノートブックをオープンして、使用可能なリストからコマンド・キーの 1 つを選択します。ユーザーが実行時にコマンド・キーを押すと、マウス・ボタンまたはキーボードのキーを押した時と同じ効果があります。プログラムには **Press** イベントが通知されます。

イベントの通知

プッシュボタンを押すと、**Press** イベントがプログラムに通知されます。

グループ・ボックス

グループ・ボックスは、ウィンドウ内のパーツのグループを視覚的に区別するために使用します。

グループ・ボックスは、パーツが関連していることを示すために、パーツのグループの周りに描かれた四角のボックスです。通常グループ・ボックスにラベルを付ける場合に有用です。ラベルが必要でない場合には、外枠を使用することができます。

パーツ属性

Bottom	Enabled	FontBold	FontItalic
FontName	FontSize	FontStrike*	FontUnder*
ForeColor	ForeMix	Handle*	Height
Label	Left	ParentName	PartName
PartType	Refresh	Top	UserData
Visible	Width		

注: 制約事項については、属性の説明を参照してください。

適用可能なイベント

Create	Destroy

グループ・ボックスのラベル付け

Label 属性は、グループ・ボックスのラベルに使用するストリングを指定するために使用します。

ラジオ・ボタンのグループ化

関連情報については、 151 ページの『ラジオ・ボタンのグループ化』 を参照してください。

水平方向のスクロール・バー

ユーザーが情報のペインを左から右または右から左にスクロールできるようにするには、水平方向のスクロール・バー・パーツを使用します。この情報は、ファイルのリストであったり、データベース中のレコードであったり、また文書中の列などである場合があります。 **Range** 属性を使用してスクロールするオブジェクトの合計数を表し、 **PageSize** 属性を使用して 1 ページに表示できるオブジェクトの数を決定することができます。

パーツ属性

Bottom	Enabled	Focus	Handle*
Height	Left	NextLine	NextPage
PageSize	ParentName	PartName	PartType
Position	PrevLine	PrevPage	Range
Top	UserData	Visible	Width

***注:** 制約事項については、属性の説明を参照してください。

適用可能なイベント

Create	Destroy	Scroll

イメージ

ピクチャーを表示するには、イメージ・パーツを使用します。 **FileName** 属性は、使用するイメージの名前を指定します。

有効な **Windows** イメージ形式には、次が含まれます。
- Windows および OS/2 ビットマップ (BMP、VGA、BGA、RLE、DIB、RL4、RL8)
- アイコン (ICO)
- Microsoft/Aldus タグ・イメージ・ファイル・フォーマット (TIF、TIFF)
- CompuServe グラフィック交換形式 (GIF)
- ZSoft PC Paintbrush イメージ・ファイル形式 (PCX)
- Truevision Targa/Vista ビットマップ (TGA、VST、AFI)
- Amiga インターリーブ・ビットマップ形式 (IFF、ILBM)
- X Windows ビットマップ (XBM)
- IBM プリンター・ページ・セグメント (PSE、PSEG、PSEG38PP、PSEG3820)
- Joint Photographic Experts Group 形式 (JPG, JPEG)

注: この製品の JPG/JPEG 形式に対するサポートは、部分的に Independent JPEG Group の働きに基づいています。

有効な **Java** イメージ形式には、次が含まれます。
- CompuServe グラフィック交換形式 (GIF)
- Joint Photographic Experts Group 形式 (JPG, JPEG)

これらのファイルは、ホスト上ではなく、プログラマブル・ワークステーション (PWS) 上にあります。システム特有のビットマップおよびアイコン・ファイルは、適切な実行時ディレクトリー (RT_JAVA または RT_WIN32) に保管して、アプリケーションをパッケージ化する時にパッケージ・ユーティリティーがそれらを組み込むようにしてください。

注: イメージ・パーツは、キャンバス付きのノートブック・ページかまたはキャンバス付きウィンドウにしかドロップできません。

パーツ属性

AddLink*	AllowLink*	BackColor	BackMix
Border	Bottom	Enabled	FileName
Handle*	Height	Left	Magnify
Panel	ParentName	PartName	PartType
Print	PrintAsIs	Refresh	RemoveLink*
ShowTips	TipText	Top	UserData
Visible	Width		

*** 注:** 制約事項については、属性の説明を参照してください。

適用可能なイベント

Click	Create	DblClick	Destroy
Link*	MouseEnter	MouseExit	MouseMove

*** 注:** 制約事項については、イベント記述を参照してください。

イメージ・パーツの作成

イメージ・パーツは、キャンバス・パーツ上でのみ作成することができます。

ファイル名の設定

イメージ・パーツ中のピクチャーを表示するには、イメージを含むファイルの名前で **FileName** 属性を設定してください。Windows アプリケーションでは、このファイルは、有効なビットマップまたはアイコン・イメージを含んでいなければなりません。Java アプリケーションでは、このファイルは、有効な GIF または JPG イメージを含んでいなければなりません。表示装置ドライバーによっては、個々のワークステーションでイメージが異なって現れることがあります。有効でないファイル名を指定した場合には、ピクチャーは表示されません。 **FileName** 属性をブランクに設定することによって、イメージ・パーツを消去することができます。 SETATR 命令コードを使用してファイル名をセットする時に、エラー(*ファイルが見つからない* など)を受け取った場合には、エラー標識がオンになります。詳細については、255 ページの『第 12 章 ピクチャー、サウンド、およびビデオ・ファイルの使用』を参照してください。

拡大パネルの制御

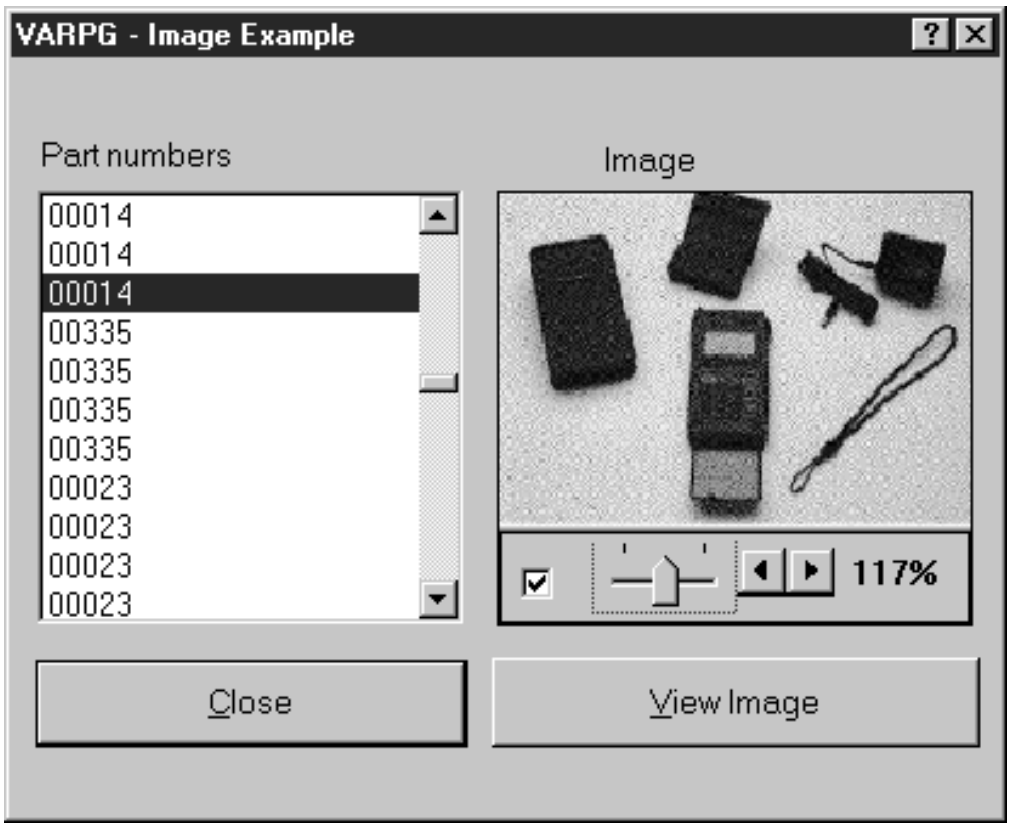
デフォルトでは、イメージ・パーツは、拡大パネルで作成されます。イメージ・パーツのプロパティー・ノートブックで拡大パネルを使用不能にすることによって、その拡大パネルを除去することができます。

プログラム中で **Panel** 属性を使用することによっても、拡大パネルを使用可能にするか、使用不能にすることができます。 **Panel** 属性を *0* にセットした場合には、拡大パネルは表示されず、さらに多くのピクチャーを表示することができます。*1* に設定した場合には、拡大パネルは表示されますが、それ以上ピクチャーを表示するスペースはなくなります。

プログラム中で拡大の程度を設定することができます。拡大値は、25-200 のパーセントで表されます。0 の値を指定すると、最適になります。この場合、イメージはイメージ・ウィンドウにちょうど収まり、横と縦の比率が一定に保たれます。

イメージの例

この例では、ファイルは iSeries 400 サーバーから読み取られます。ファイル中の各レコードはパーツ番号フィールドを含み、各レコードが読み取られると、このフィールドがリスト・ボックスに挿入されます。ユーザーがリスト・ボックス中のパーツ番号を選択して、**「ビュー」**プッシュボタンを押すと、そのパーツ番号がストリング *.BMP* と連結され、ファイル名を作ります。ここでこのファイル名が使用され、ピクチャーを表示するためのイメージ・パーツの **FileName** 属性をセットします。PINFO 静的テキスト・パーツの **Label** 属性が更新され、ファイル名属性をセットした結果を示します。プログラムを終了するためには、**「クローズ」**プッシュボタンを押してください。



```
*************************************************************************
*                                                                       *
* プログラム ID .: IMAGE                                                *
*                                                                       *
* 説明 . . . . . : イメージ・パーツの例                                 *
*                                                                       *
*                  このサンプル・プログラムは VARPG クライアントで      *
*                  イメージ・パーツを設定する方法を示したものです。     *
*                                                                       *
*                  この例では、ホスト AS/400 システムに PARTS と        *
*                  呼ばれるファイルがあることを想定しています。この     *
*                  ファイル形式は PARTNO と呼ばれるフィールドから       *
*                  なります。                                           *
*                                                                       *
*                  アプリケーションが開始されると、ウィンドウ WIN1      *
*                  の Create イベントが呼び出され、これがファイル       *
*                  からすべてのレコードを読み取り、リスト・ボックス     *
*                  LB1 に PARTNO フィールドの値を挿入します。           *
*                                                                       *
*                  ユーザーが「ビュー」プッシュボタンを押すと、         *
*                  イメージ・ファイル名が連結され、これが使用されて     *
*                  イメージ・パーツ IMG1 の FILENAME 属性がセット       *
*                  されます。                                           *
*                                                                       *
*************************************************************************
 *
H
 * PARTS ファイルの定義
 *
FPARTS     IF   E            DISK    REMOTE
 *
DPath              C                   ''
dnopic             C                   'Picture not available'
 *
```

図 19. イメージ・パーツの使用例 (1/3)

```
 ********************************************************************
 *                                                                  *
 * ウィンドウ : WIN1                                                *
 *                                                                  *
 * パーツ . . : Close                                               *
 *                                                                  *
 * イベント . . : Press                                               *
 *                                                                  *
 * 説明 . . . . . : プログラムの終了                                *
 *                                                                  *
 ********************************************************************
 *
C     CLOSE         BEGACT    PRESS         WIN1
 *
C                   move      *on           *inlr
 *
C                   ENDACT
 ********************************************************************
 *                                                                  *
 * ウィンドウ : WIN1                                                *
 *                                                                  *
 * パーツ . . : WIN1                                                *
 *                                                                  *
 * イベント . . : Create                                            *
 *                                                                  *
 * 説明 . . . : このアクション・サブルーチンは、ウィンドウ WIN1 の   *
 *               作成時に実行されます。                             *
 *               PARTS ファイルからのパーツ番号の値でリスト・       *
 *               ボックスが埋められます。                           *
 *                                                                  *
 ********************************************************************
 *
C     WIN1          BEGACT    CREATE        WIN1
 *
 * データベースからの項目で listbox パーツを埋めます
C                   read      produc1                             9999
 *
C     *in99         doweq     *off
C     'LB1'         setatr    partno        'InsertItem'
C                   read      produc1                             9999
C                   enddo
 *
C                   ENDACT
 *
```

図 19. イメージ・パーツの使用例 (2/3)

```
 ************************************************************************
 *                                                                      *
 * ウィンドウ : WIN1                                                    *
 *                                                                      *
 * パーツ . . : VIEW                                                    *
 *                                                                      *
 * イベント . . : PRESS                                                 *
 *                                                                      *
 * 説明 . . . : 選択されたパーツのイメージを表示します                 *
 *                                                                      *
 ************************************************************************
 *
C     VIEW          BEGACT    PRESS         WIN1
 *
 * 選択された項目の索引を取り出します
C     'LB1'         getatr    'FirstSel'    x                4 0
 *
 * 項目が選択されると、ビットマップ・ファイル名をビルドします
C     x             ifgt      *zero
C     'LB1'         setatr    x             'Index'
C     'LB1'         getatr    'GETITEM'     tmp20           20
C                   movel     tmp20         part             5
C                   endif
 *
C                   move      *blanks       fullpath        64
C                   move      *blanks       tmp64           64
C     path          cat       part:0        tmp64
C     tmp64         cat       '.gif':0      fullpath
 *
 * イメージの FILENAME 属性中にファイル名をセットして、
 * イメージを表示します
C     'IMG1'        setatr    fullpath      'FILENAME'            80
 *
 * 標識 80 がオンの場合には、イメージ・ファイル名の設定は失敗しています。
 * すなわち、ファイルが見つかりません。
 * 状況を指示するために、静的テキスト・パーツ PINFO に Label 属性を
 * 設定します
C     *in80         ifeq      *on
C     'PINFO'       setatr    nopic         'Label'
 *
C                   else
C     'PINFO'       setatr    *BLANKS       'Label'
C                   endif
 *
C                   ENDACT
 *
```

図 19. イメージ・パーツの使用例 (3/3)

Java Bean

*** 制約事項:** このパーツは、Windows アプリケーションではサポートされません。

プロジェクトに JavaBeans® を追加するには、Java Bean パーツを使用します。Java メソッドを呼び出すことによって、直接 JavaBeans を使用できます。Java メソッドの呼び出しの詳細については、295 ページの『第 18 章 VisualAge RPG プログラムからの Java メソッドの呼び出し』 を参照してください。

Java Bean パーツを使用するアプリケーションを開発するには、Sun Microsystems の Java 2 ソフトウェア開発キット (J2SDK)、標準版、バージョン 1.2 以上がワークステーションにインストールされている必要があります。 J2SDK がない場合には、Sun Microsystems から次の URL でダウンロードすることができます。

```
http://java.sun.com/products/
```

J2SDK をインストールしたら、PATH 環境変数を Java コンパイラーと **jvm.dll**(J2SDK および JRE (Java Runtime Environment) のパーツ) の両方が入っている場所を示すように設定してください。たとえば、J2SDK のホーム・ディレクトリーが *x:\jdk1.2* の場合には、次の path ステートメントを追加してください。

```
x:\jdk1.2\bin

x:\jdk1.2\jre\bin\classic
```

パーツ属性

AddEvent	Bottom	Enabled	Height
Left	ParentName	PartName	PartType
RmvEvent	ShowProp	Top	UserData
Visible	Width		

適用可能なイベント

Create	Destroy	BeanEvent

プロジェクトへの Beans の追加

プロジェクトに bean を追加するには、パーツ・パレットで Java Bean パーツをクリックします。マウス・ポインターで、設計ウィンドウの bean を配置したい場所をクリックします。ファイル「オープン」ダイアログが表示されます。処理したい 1 つまたは複数の bean が入っている JAR ファイルを選択します。JAR ファイルで使用可能なすべての bean をリストしたウィンドウが表示されます。リストから bean を選択すると、その bean がインスタンス化されます。これは別のウィンドウに、関連するプロパティー・シート・ダイアログおよび bean カスタマイザー (使用可能な場合) と一緒に表示されます。 (Bean のプロパティーは、プロパティー・シート・ダイアログおよび bean カスタマイザーで変更できます。)

Bean のプロパティー、メソッド、およびイベントを表示するには、設計ウィンドウから「Java Bean パーツのプロパティー」ノートブックをオープンします。設計ウィンドウで Java Bean アイコンを右クリックして、**「プロパティー」**を選択します。 Bean のプロパティー、メソッド、およびイベントは「情報」ページにあります。使用できるものを調べるには、該当するラジオ・ボタンを選択してください。

Bean のすべてのイベントがサポートされるわけではありません。VARPG でサポートされるイベントには、イベント・リストの前にアスタリスク (*) が付けられています。

Bean JAR ファイルの位置

プロジェクトの bean と関連したすべての JAR ファイルは、内部 bean ディレクトリー `x:\...\WDSC\beans` の中になければなりません。この場合の *x:\..\WDSC* は、VARPG がインストールされているホーム・ディレクトリーです。このディレクトリーには、bean 従属 JAR ファイルも含まれていなければなりません。たとえば、BeanA が BeanA.jar にあって beanclass.jar にあるクラス・ファイルを必要とする場合には、BeanA.jar と beanclass.jar の両方が内部 bean ディレクトリーにコピーされていなければなりません。

プロジェクトの編集時には、内部 bean ディレクトリーにない bean を JAR ファイルから選択することができます。この JAR ファイルは、プロジェクトのビルド後に内部 bean ディレクトリーにコピーされます。しかし、bean 従属 JAR ファイルをこのディレクトリーにコピーする必要があります。

使用されない JAR ファイルは内部 bean ディレクトリーから除去してください。これを行うことによって、不必要なプロジェクト以外の JAR をロードしなくて済みます。

JAR クラスパスの設定

VARPG パッケージ・ユーティリティーでは、JAR ファイルのクラスパスの設定は処理されません。クラスパス変数を設定して、そのプロジェクトで bean によって使用されるすべての JAR ファイルを含める必要があります。たとえば、BeanA が BeanA.jar を使用して、beanclasses.jar に依存している場合には、次のようにクラスパスを設定してください。

```
SET CLASSPATH=x:\beandir\BeanA.jar;x:\beandir\beanclasses.jar;%CLASSPATH%;
```

この場合の *x:\beandir* は、bean の JAR ファイルへのパスです。

JavaBean のプロパティーの設定 / 取得およびメソッドの呼び出し

VARPG は、java メソッドの直接の呼び出しをサポートします。(詳細については、295 ページの『第 18 章 VisualAge RPG プログラムからの Java メソッドの呼び出し』 を参照してください。)VARPG ランタイムは、プロジェクトの JavaBeans オブジェクトにアクセスするために、Java アクセス機能クラスを提供します。アクセス機能クラス *com.ibm.varpg.parts.VarpgBeanPart* は varpg.jar ファイルに入っています。このクラスによって、VARPG ランタイムでインスタンス化された bean オブジェクトを検索することができます。これらの bean オブジェクトは VARPG プロジェクトのパーツです。Bean オブジェクトを取得するためのメソッドは次の通りです。

```
public static Object getBeanObject(String strComponentName,
                                    String strParentName,
                                    String strPartName );
```

この場合に、*strComponentName* は bean パーツが入っているコンポーネントの名前、*strParentName* は bean パーツが入っているウィンドウの名前、*strPartName* は bean パーツの名前です。呼び出し側は、メソッドによって戻されたヌル参照を調べて、呼び出しが正常に行われたことを確認する必要があります。

プロパティーを設定または取得し、メソッドを呼び出すための実際のメソッドの仕様を調べるには、ベンダーが提供している bean の資料を参照するか、あるいは「Java Bean パーツのプロパティー」ノートブックの「情報」ページを調べてください。

リスト・ボックス

リスト・ボックス・パーツを使用して、1 つまたは複数の項目を選択できる項目のリストをユーザーに提供します。リスト・ボックスは読み取り専用の項目から構成されています。リスト・ボックス中の項目は文字のストリングです。

ユーザーは水平方向および縦方向のスクロール・バーを使って、現在表示されていないリストのセクションを見ることができます。ユーザーは、1 項目だけを選択するように、あるいは複数項目を選択できるようにリスト・ボックスを構成することができます。**Search**、**SearchType**、および **Case** 属性を使用して、リストの特定の項目を容易に検索することができます。

パーツ属性

AddItemEnd	AddLink*	AllowLink*	BackColor
BackMix	Bottom	Case*	Count
DelimChar	DeSelect	DragEnable*	DropEnable*
Enabled	ExtSelect*	FirstSel	Focus
FontBold	FontItalic	FontName	FontSize
FontStrike*	FontUnder*	ForeColor	ForeMix
GetItem	Handle*	Height	Index
InsertItem*	ItemKey	Left	MultSelect
NbrOfSel	ParentName	PartName	PartType
Refresh	RemoveItem	RemoveLink*	Search*
SearchType*	Selected	SelectItem	SelectList
Sequence*	SetItem	SetTop	ShowTips
SizeToFit	TipText	Top	TopItem
UseDelim	UserData	Visible	Width

*** 注:** 制約事項については、属性の説明を参照してください。

適用可能なイベント

Create	Destroy	Drop*	Enter*
GotFocus	KeyPress	LostFocus	MouseEnter
MouseExit	MouseMove	Popup	Select
VKeyPress			

*** 注:** 制約事項については、イベント記述を参照してください。

項目の追加および項目順序の設定

デフォルトでは、項目は、追加された順序でリスト・ボックス中に表示されます。それらをより正確な順序で表示したい場合には、追加を開始する前に **Sequence** 属性を昇順、降順、または索引のいずれかに設定してください。これで、項目は追加された時の ASCII 照合順序でソートされます。

リスト・ボックスにすでにある項目の順序を変更するのに、**Sequence** 属性を使用することはできません。

実行時の項目の追加

InsertItem 属性を使用することによって、実行時にリスト・ボックスに項目を挿入することができます。項目が表示される順序は、 **Sequence** 属性によって決定されます。

リストの項目の更新

リストにすでにある項目を変更することができます。**Index** 属性を使用して、変更したい項目を示し、**SetItem** 属性を使用して変更されるデータを指定してください。

注: **SetItem** 属性を使用して項目を変更した時には、**Sequence** 属性の値に関係なく、項目は元の位置に留まります。たとえば、リストを作成した時に **Sequence** 属性を昇順にセットした場合には、項目はリスト・ボックス中に昇順で現れます。しかしその後で項目を検索し、その値を変更し、**SetItem** 属性を使用してリスト・ボックス中のその項目を置き換えた場合には、その項目は前にあったのと同じ位置に挿入されます。 したがって、変更後にはリストが昇順になることもあれば、ならないこともあります。

リストの最上部の設定

リスト・ボックスの最上部に現れることになるリスト項目を指定するには、**SetTop** 属性を使用してください。この項目をセットすると、リストがスクロールされます。これにより表示は変更されますが、リスト中の項目の順序は変更**されません**。

項目の除去

リストから項目を除去するには、**RemoveItem** 属性を使用してください。索引値を使用して、除去される項目を指定してください。索引値は *1* から始まります。リストから項目を除去すると、除去された項目の後のすべての項目がリスト中で 1 つ上に移動します。

リストからすべての項目を除去するには、 *0* という索引値を指定してください。

項目の選択および選択解除

ユーザーは、マウスまたはキーボードを使って項目を選択または選択解除することができます。プログラム中で **Selected** および **DeSelect** 属性をセットして項目を選択および選択解除することができます。これらの属性を使用するには、最初に **Index** 属性をセットしてください。

選択のタイプ

属性を使って、リスト・ボックス中での項目の選択方法を指定することができます。単一選択、複数選択、および拡張選択が使用可能です。

単一選択

単一選択 (デフォルト) では、リスト中の項目は一度に 1 つしか選択することができません。項目が現在選択されている場合には、別の項目が選択されると、その項目が選択解除されます。

複数選択

ユーザーはオブジェクトをいくつでも選択することもできますし、または何も選択しないでおくこともできます。

拡張選択

このタイプの選択は単一オブジェクトの選択のために最適化されていますが、ユーザーは必要であれば選択を複数のオブジェクトに拡張することができます。

リストからの項目の検索

リスト・ボックスから項目を検索するには、**GetItem** 属性を使用してください。最初に、検索したい項目を示すために、**Index** 属性をセットしてください。

一般的には、ユーザーによって選択されている項目を検索します。リスト・ボックス中のどの項目が選択されているかを判別するには、**FirstSel** または **Selected** 属性を使用してください。**FirstSel** 属性では、リスト中の最初に選択された項目の索引が戻されます。選択されている後続の項目を調べる必要がある場合には、かならず **DeSelect** 属性を使ってこの項目を選択解除してください。さもないと、**FirstSel** 属性で同じ項目が戻されます。

特定の項目が選択されているかどうかを判別するには、 **Selected** 属性を使用してください。**Selected** 属性では **Index** 属性の値が使用され、その項目が選択されているかどうかが判別されます。

Count 属性を使用して検索する項目があるかどうかを判別することができます。

キーの使用法

リスト・ボックスおよび組み合わせボックスの両方で、「キー」部分と「データ」部分から成る項目をリストに追加することができます。項目をリストに追加すると、その項目のデータ部分だけが表示されます。ユーザーが項目を選択すると、その項目のキー部分をプログラム式に検索することができます。

詳細については、組み合わせボックス・パーツの記述の中の「キーの使用法」のセクションを参照してください。

イベントの通知

Select イベントは次の場合に通知されます。

- ユーザーがリスト・ボックスにある項目を選択する。
- ユーザーがプログラムのリストにある項目を選択する。
- ユーザーがすでに選択されている項目を選択する。

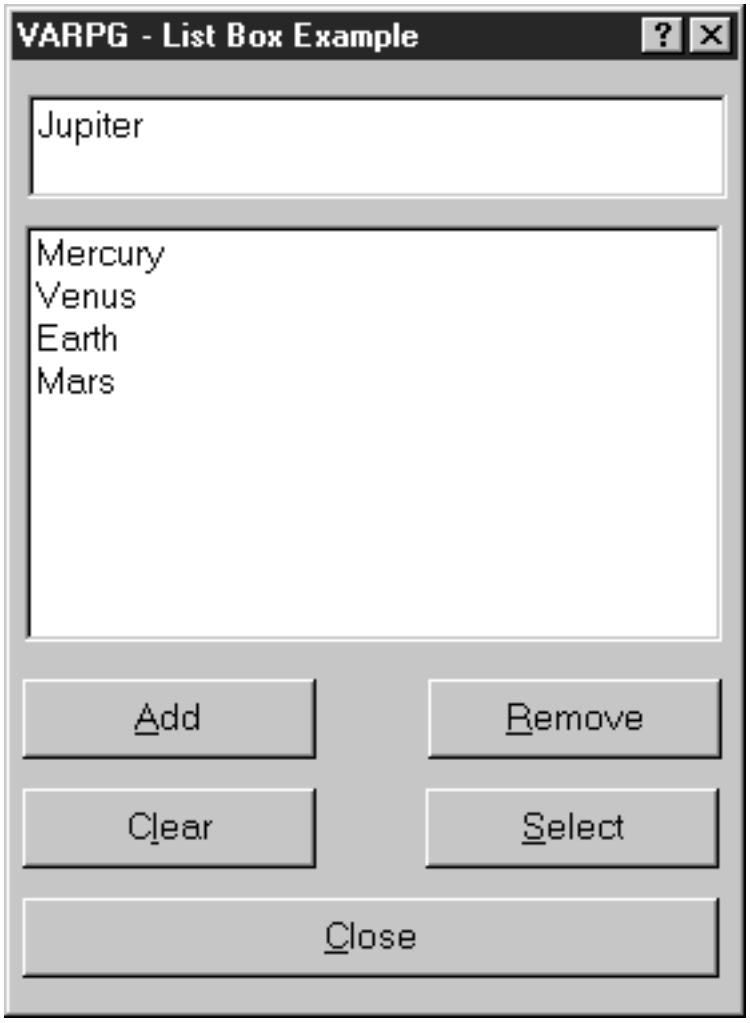
Enter イベントは次の場合に通知されます。

- ユーザーがリスト・ボックス中の項目をダブルクリックする。
- リスト・ボックスがフォーカスされ、項目が選択されている時にユーザーが改行キーを押す。

これらのイベントに対するアクション・サブルーチン中で、 **Selected** または **FirstSel** 属性を使用して、選択されている項目を判別することができます。

リスト・ボックスの例

入力フィールド・パーツからリストにテキスト値を挿入するには、**「追加」**プッシュボタンを押してください。リストを消去するには、**「消去」**プッシュボタン、リストから項目を選択するには、**「選択」**プッシュボタン、リストから選択した項目を除去するには、**「除去」**プッシュボタンをそれぞれ押してください。プログラムを終了するには**「クローズ」**を押します。



```
*****************************************************************
*                                                               *
* プログラム ID : LISTBOX                                       *
*                                                               *
* 説明  . . . . : リスト・ボックス・パーツを説明するサンプル・  *
*                 プログラム                                    *
*                                                               *
*****************************************************************
*
*****************************************************************
*                                                               *
* ウィンドウ . . : MAIN                                         *
*                                                               *
* パーツ . . : CLEAR                                            *
*                                                               *
* イベント . . : PRESS                                          *
*                                                               *
* 説明 . . . . : リスト・ボックスおよび入力フィールドを消去します。*
*                 入力フィールド・パーツをフォーカスします。    *
*                                                               *
*****************************************************************
*
C     CLEAR         BEGACT    PRESS         MAIN
*
C     'LB1'         setatr    0             'RemoveItem'
C     'EF1'         setatr    *blanks       'Text'
C     'EF1'         setatr    1             'Focus'
*
C                   ENDACT
*****************************************************************
*                                                               *
* ウィンドウ : FRA0000B                                         *
*                                                               *
* パーツ . . : CLOSE                                            *
*                                                               *
* イベント . . : PRESS                                          *
*                                                               *
* 説明 . . . . . : プログラムの終了                             *
*                                                               *
*****************************************************************
*
C     CLOSE         BEGACT    PRESS         MAIN
*
C                   move      *on           *INLR
*
C                   ENDACT
```

図 20. リスト・ボックス・パーツを使用するコーディングの例 (1/3)

```
*********************************************************************
*                                                                   *
* ウィンドウ . . : MAIN                                             *
*                                                                   *
* パーツ . . : REMOVE                                               *
*                                                                   *
* イベント . . : PRESS                                              *
*                                                                   *
* 説明 . . . : リスト・ボックスから選択した項目を除去します。       *
*              最初に選択された項目の索引を判別するために 'FirstSel'*
*              属性が使用されます。                                 *
*                                                                   *
*********************************************************************
 *
C     REMOVE        BEGACT    PRESS         MAIN
 *
C     'LB1'         getatr    'FirstSel'    Index             3 0
 *
C     Index         ifgt      *zero
C     'LB1'         setatr    Index         'RemoveItem'
C                   endif
 *
C                   ENDACT
*********************************************************************
*                                                                   *
* ウィンドウ : FRA0000B                                             *
*                                                                   *
* パーツ . . : ADD                                                  *
*                                                                   *
* イベント . . : PRESS                                              *
*                                                                   *
* 説明 . . . : 入力フィールドの値を新しい項目として                 *
*              リスト・ボックス・パーツに追加します。               *
*                                                                   *
*********************************************************************
 *
C     ADD           BEGACT    PRESS         MAIN
 *
C     'EF1'         getatr    'TEXT'        tmp              30
 *
C     tmp           ifne      *blanks
C     'LB1'         setatr    tmp           'InsertItem'
C     'EF1'         setatr    *blanks       'Text'
C     'EF1'         setatr    1             'Focus'
C                   endif
 *
C                   ENDACT
```

図 20. リスト・ボックス・パーツを使用するコーディングの例 (2/3)

```
*************************************************************************
*                                                                       *
* ウィンドウ . . : MAIN                                                 *
*                                                                       *
* パーツ . . : SELECT                                                   *
*                                                                       *
* イベント . . : PRESS                                                  *
*                                                                       *
* 説明 . . . . : 選択された項目をリスト・ボックスから検索して、それを *
*                 入力フィールドにコピーします。                        *
*                                                                       *
*************************************************************************
 *
C     SELECT        BEGACT    PRESS         MAIN
 *
C     'LB1'         getatr    'FirstSel'    x                3 0
 *
C     x             ifgt      *zero
C     'LB1'         setatr    x             'Index'
C     'LB1'         getatr    'GetItem'     temp            20
C     'EF1'         setatr    temp          'Text'
C                   endif
 *
C                   ENDACT
*************************************************************************
*                                                                       *
* ウィンドウ . . : MAIN                                                 *
*                                                                       *
* パーツ . . : EF1                                                      *
*                                                                       *
* イベント . : CHANGE                                                   *
*                                                                       *
* 説明 . . . . : CRP サンプルの通知イベント用                           *
*                                                                       *
*************************************************************************
 *
C     EF1           BEGACT    CHANGE        MAIN
 *
C                   ENDACT
 *
```

図 20. リスト・ボックス・パーツを使用するコーディングの例 (3/3)

検索の例

リスト中の特定の項目を検索するには、Search、SearchType、および Case 属性を使用することができます。これらを使用する方が、プログラム中で各項目を読み取って特定の値と比較するより速いです。

次の例では、ユーザーが入力フィールドに名前を入力した時に、リスト・ボックス中の得意先名を見つける方法を示します。ウィンドウの名前は MAIN で、リスト・ボックスは LB1 で、入力フィールドは EF1 とします。ユーザー・インターフェースは次の通りです :

ウィンドウの Create イベントで、この検索では大文字・小文字を区別しないことを指示するために、リスト・ボックスの Case 属性を 0 に設定します。検索ストリング中の文字数をリスト項目の最初の文字数と比較するだけなので、SearchType 属性は 1 に設定します。コードの残りの部分は、iSeries データベースからのレコードでリスト・ボックスを埋め込みます。

```
     C     MAIN          BEGACT    CREATE        MAIN
 *
C     'LB1'         Setatr    0             'Case'
C     'LB1'         Setatr    1             'SearchType'
 *
C                   Read      Custom01                                99
 *
C                   DoW       NOT *in99
C     'LB1'         Setatr    CustNa        'AddItemEnd'
C                   Read      Custom01                                99
C                   EndDo
 *
C     'LB1'         Setatr    1             'SelectItem'
 *
C                   ENDACT
```

次のコードは入力フィールド EF1 の Change イベントのアクション・サブルーチンです。入力フィールドに文字が入力されるたびに、このアクション・サブルーチンが呼び出されます。

入力フィールドの Text 属性の値が検索され、それがブランクでない場合には、その値がリスト・ボックスの Search 属性の検索ストリングとして使用されます。一致

が見つかると (Index 属性が 0 より大きい)、見つかった項目が選択され、 Settop 属性により、リスト・ボックスの最上部に移動します。

```
C     EF1           BEGACT    CHANGE        MAIN
 *
C     'EF1'         Getatr    'Text'        Search           40
 *
C                   If        Search <> *Blanks
C     'LB1'         Setatr    Search        'Search'
 *
C                   If        %Getatr('Main':'LB1':'Index')<>0
C                   Eval      %Setatr('Main':'LB1':'SelectItem')=
C                               %Getatr('Main':'LB1':'Index')
C                   Eval      %Setatr('Main':'LB1':'SetTop')=
C                               %Getatr('Main':'LB1':'Index')
C                   EndIf
 *
C                   Else
C     'LB1'         Setatr    1             'SetTop'
C     'LB1'         Setatr    1             'SelectItem'
C     'LB1'         Setatr    1             'Index'
 *
C                   EndIf
 *
C                   ENDACT
```

入力フィールドがブランクの場合には、リスト・ボックスの最初の項目が最上部に移動し、選択されます。後続の検索がリストの最上部から開始されるように、INDEX 属性が 1 に設定されます。

メディア

メディア・パーツは、オーディオ情報を再生または記録し、ビデオ・ファイルを再生するために使用します。

メディア・パーツによって、ユーザー・プログラムは wave (.WAV)、MIDI (.MID)、および QuickTime ムービー (.MOV) ファイルを処理することができます。オーディオ・ファイルを使用するには、これらのファイルの処理に使用可能なサウンド・カードがコンピューターに装備されていなければなりません。サウンド・ファイルを記録するためには、マイクロホンまたはサウンド・カード用にサポートされているその他の入力装置が必要となります。メディア・パーツでは、MIDI ファイルは記録することはできません。

Java アプリケーションには、Java メディア・フレームワーク (JMF) API が必要です。メディア・パーツは、Java 環境でのオーディオおよびビデオ・ファイルの再生だけをサポートしています。

処理できるビデオ・ファイルの形式は次の通りです。 MPEG (*.mpg) ファイル、QUICKTIME ムービー (*.mov) ファイル、*.dat ファイル、 Microsoft® Video for Windows *.avi ファイルが Windows 用にサポートされています。これらのビデオ・ファイルを再生するためには、コンピューターに適切なドライバーがなければなりません。

パーツ属性

AddLink*	AllowLink*	AudioMode	Bass*
Bottom	FileName	Handle*	InPlace
Left	Length	Panel	ParentName
PartName	PartType	Position	RemoveLink*
Top	Treble*	UserData	Visible
Volume			

***注:** 制約事項については、属性の説明を参照してください。

適用可能なイベント

Complete	Create	Destroy	Link*

***注:** 制約事項については、イベント記述を参照してください。

ファイル名の指定

FileName 属性の値は、処理したいファイルの名前を指定するために使用します。詳細については、 255 ページの『第 12 章 ピクチャー、サウンド、およびビデオ・ファイルの使用』 を参照してください。

注: 一部のウェーブ・ファイルは圧縮形式で出荷されます。メディア・パーツが処理するのは、圧縮されていないウェーブ・ファイルだけです。

AudioMode の設定

ファイルを処理するためには、**AudioMode** 属性を次の値の 1 つに設定してください。

値	説明
1	一時停止 — ファイルの処理を保留
2	再生 — ファイルを再生
3	記録 — ファイルを記録
4	停止 — ファイルの処理を停止

ボリュームの設定

Volume 属性を使用して、メディア・パーツおよびシステムの出力波およびシンセサイザーのボリュームを設定します。

位置の設定

Position 属性は、処理するファイルの開始位置を決定するために使用します。この属性値はミリ秒で表します。

メディア・パネル・パーツの使用

メディア・パネル・パーツはメディア・パーツを制御するために使用することができます。メディア・パネル・パーツのプロパティー・ノートブックで、 **AddLink** 属性にメディア・パーツ名を設定して、**AllowLink** 属性を使用可能にします。これによって、ユーザーはメディア・パネルの該当するボタンを押すだけでメディア・パーツを制御することができます。詳細については、 110 ページの『メディア・パネル』 を参照してください。

イベントの通知

メディア・パーツがファイルを完全に処理すると、 **Complete** イベントが通知されます。

メディア・パネル

注: このパーツは Java アプリケーションではサポートされていません。

メディア・パネル・パーツを使用して、頻繁に使用するアクションを容易にアクセスできるようにします。

また、メディア・パネル・パーツを使用して、プログラム・ロジックを作成することなしに、他のパーツをユーザー制御することもできます。たとえば、それを使用してプッシュボタンを作成したり、メディア・パーツのボリュームまたはモードを制御するスライダー・コントロールを作成することができます。

メディア・パネル・パーツのプロパティー・ノートブックでは、次のことを決定することができます。
- 定義された一連のボタンからどのボタンをメディア・パネルに入れるか
- 位置およびボリュームのスライダー・コントロールを表示するかどうか

注: メディア・パネル・パーツはキャンバス付きのノートブックまたはキャンバス付きウィンドウにだけ入れることができます。

パーツ属性

AddLink	AllowLink	BackColor	BackMix
Bottom	Enabled	Handle	Height
Left	PanelItem	PanelMode	ParentName
PartName	PartType	Position	RemoveLink
Top	UserData	Visible	Volume
Width			

適用可能なイベント

Change	Create	Destroy	Link
MouseEnter	MouseExit	MouseMove	Popup
Press			

メディア・パネル・パーツの作成

メディア・パネル・パーツは、キャンバス・パーツでのみ作成することができます。

他のパーツのリンク

メディア・パネル・パーツを別のパーツにリンクするには 2 つの方法があります。1 つはプロパティー・ノートブックを使用する方法で、もう 1 つはプログラム・ロジックを書く方法です。最初の方法が 1 番簡単です。プログラム・ロジックを書く必要があるのは、リンクを実行時にセットしたい場合だけで、**AddLink** および **AllowLink** 属性をセットすることになります。代表的な例では、メディア・パネル

はメディア・パーツにリンクされます。メディア・パネルで制御が変更されると、リンクのメカニズムにより、自動的にメディア・パーツが影響されます。

メディア・パネル・パーツから別のパーツへのリンクを作成すると、メディア・パネル・パーツでは一部のボタンだけが使用可能になります。すべてのボタンを使用可能にするには、そのパーツからメディア・パネル・パーツへのリンクも作成しなければなりません。

メディア・パネル・パーツにリンクできるパーツの詳細については、*ADTS/CS VisualAge for RPG パーツ解説書*, SD88-5040-03 の中の **AddLink** の説明を参照してください。

イベントの通知

ボリューム・スライダーまたは位置スライダーが移動されると、 **Change** イベントのシグナルが出されます。変更されたスライダーを判別するには、**PanelItem** 属性を使用してください。ボリューム・スライダーの値を判別するには、**Volume** 属性、位置スライダーの値を判別するには、 **Position** 属性をそれぞれ使用してください。

メディア・パネルでプッシュボタンが押されると、**Press** イベントのシグナルが出されます。 **PanelItem** 属性で戻された数値を使用して、**Press** イベントの原因となったボタンを判別してください。使用可能な値のリストについては、 *ADTS/CS VisualAge for RPG パーツ解説書*, SD88-5040-03を参照してください。

メニュー・バー

メニュー・バー・パーツは、ユーザーがプルダウン・メニューにアクセスするために使用します。メニュー・バーには、サブメニュー・パーツおよびメニュー項目パーツを追加することができます。

メニュー・バーは、ウィンドウ・フレームの上の方、タイトル・バーのすぐ下に表示されます。ユーザーがメニュー・バーからメニュー項目を選択すると、プルダウン・メニューが現れ、メニュー上に項目を表示します。メニュー項目を選択すると、記述されたアクションが即時に開始されます。

注: このパーツのプロパティー、イベントなどを操作できるのは、プロジェクト・ツリー・ビューのそのポップアップ・メニューからだけです。

関連情報については、以下を参照してください。
- 113 ページの『メニュー項目』
- 183 ページの『サブメニュー』
- 147 ページの『ポップアップ・メニュー』

パーツ属性

PartType	PartName	ParentName	UserData

適用可能なイベント

Create	Destroy

プルダウン・メニューの作成

メニュー・バー、サブメニュー、またはメニュー項目のプロパティー・ノートブックは、ダブルクリックやポップアップ・メニューを介した操作ではオープンできません。ツリー・ビューでそれを操作する必要があります。

メニュー項目

プルダウンあるいはポップアップ・メニューを構成するには、メニュー項目を使用します。

メニュー項目は、ユーザーがその項目を選択した時に開始されるアクションを記述します。

メニューを構成するには、次のようにしてください。
1. サブメニュー・パーツをメニュー・バーまたはポップアップ・メニューにドロップする。
2. メニュー項目をサブメニューにドロップする。

注: このパーツのプロパティー、イベントなどを操作できるのは、プロジェクト・ツリー・ビューのそのポップアップ・メニューからだけです。

関連情報については、以下を参照してください。
- 112 ページの『メニュー・バー』
- 147 ページの『ポップアップ・メニュー』
- 183 ページの『サブメニュー』

パーツ属性

Checked	Enabled	FileName	Label
ParentName	PartName	PartType	UserData
Visible			

適用可能なイベント

Create	Destroy	MenuSelect

メニュー項目の横にチェック・マーク

メニュー項目の横のチェック・マークは、そのメニュー項目によって表されるアクションが選択されていることをユーザーに指示します。たとえば、**グリッド表示**のメニュー項目の横にチェック・マークが現れている場合には、グリッドが表示されます。

メニュー項目の横にチェック・マークを表示するには、 **Checked** 属性を *1* にセットしてください。チェック・マークを除去するには、この属性を *0* にセットしてください。

メニュー・テキストの設定

メニュー項目のテキストをセットするには、**Label** 属性を使用してください。

簡略記号の設定

注: 簡略記号は Java アプリケーションではサポートされていません。

メニュー項目の簡略記号キーを指定するには、**Label** 属性のテキストの文字の前に簡略記号 ID を入れます。 Windows では、アンパーサンド (&) を使用してください。指定された文字が下線付きでインターフェース上に表示されます。たとえば **Display**。この下線は、キーボードの下線のついたその文字を押してメニュー項目を選択できるということをユーザーに指示します。

メニュー項目の使用可能化

ユーザーがメニュー項目を選択した時に、**MenuSelect** イベントが出されるかどうかを制御することができます。

デフォルトでは、メニュー項目は、作成した時に使用可能になります。使用可能になったメニュー項目は、選択されると、**MenuSelect** イベントを生成します。

メニュー項目を使用可能にしたくない場合には、**Enabled** 属性を *0* にセットしてください。メニュー項目が使用不能になると、そのメニュー項目は画面上でぼかし表示になり、選択された時に **MenuSelect** イベントを生成しません。

イベントの通知

ユーザーがメニュー項目を選択すると、**MenuSelect** イベントのシグナルが出されます。

注: メニュー項目は、**MenuSelect** イベントのシグナルを出すだけです。他のメニュー項目につながっているサブメニュー (カスケード・メニューなど) は、これを行いません。

メッセージ・サブファイル

定義済みメッセージを表示するか、エラーや状況情報など、プログラム・ロジックで提供したテキストを表示するには、メッセージ・サブファイル・パーツを使用してください。

このパーツは常にウィンドウ・フレームの最下部に位置付けされ、ウィンドウの幅を実行します。ウィンドウの幅のサイズ変更を行うことはできませんが、さらにメッセージを表示するようにその高さのサイズ変更を行うことはできます。実行時点で、ユーザーはすべてのメッセージを見るために、スクロール・バーを使用することができます。

パーツ属性

AddMsgID	AddMsgText	Count	DragEnable*
DropEnable*	Enabled	FirstSel	FontBold
FontItalic	FontName	FontSize	FontStrike*
FontUnder*	ForeColor	ForeMix	GetItem
Handle*	Height	Index	MsgSubText
NbrOfSel	ParentName	PartName	PartType
RemoveMsg	Selected	ShowTips	TipText
UserData	Visible		

*** 注:** 制約事項については、属性の説明を参照してください。

適用可能なイベント

Create	Destroy	Drop	Enter
MouseEnter	MouseExit	MouseMove	Popup
Select			

定義済みメッセージの表示

GUI Designer ですでに定義されているメッセージを表示するには、**AddMsgId** 属性を表示したいメッセージの ID 番号にセットします。メッセージ ID の数字部分を使用してください。

プログラム中で提供されたテキストの表示

定義済みメッセージの一部でないテキストを表示するには、プログラム中に **AddMsgText** 属性を使用して、テキスト・ストリングまたはリテラルをメッセージ値として提供してください。

置換変数の使用

メッセージ・サブファイル・パーツは、置換変数をサポートします。置換変数は、前にパーセント (%) 文字を付けた数値 (たとえば、*%123*) を入力してメッセージ

を作成する時に定義されます。この置換変数は、メッセージの追加前にプログラムによって提供されたデータで置き換えられます。メッセージ置換データは **AddMsgID** および **AddMsgText** 属性に適用します。

メッセージ置換データは、ブランクで区切られた一連の語からなります。メッセージがメッセージ・サブファイル・パーツに追加される前に、各置換語が対応する置換変数に置き換わります。メッセージ置換データをセットするには、**AddMsgID** 属性をセットする**前に** **MsgSubText** 属性を使用してください。

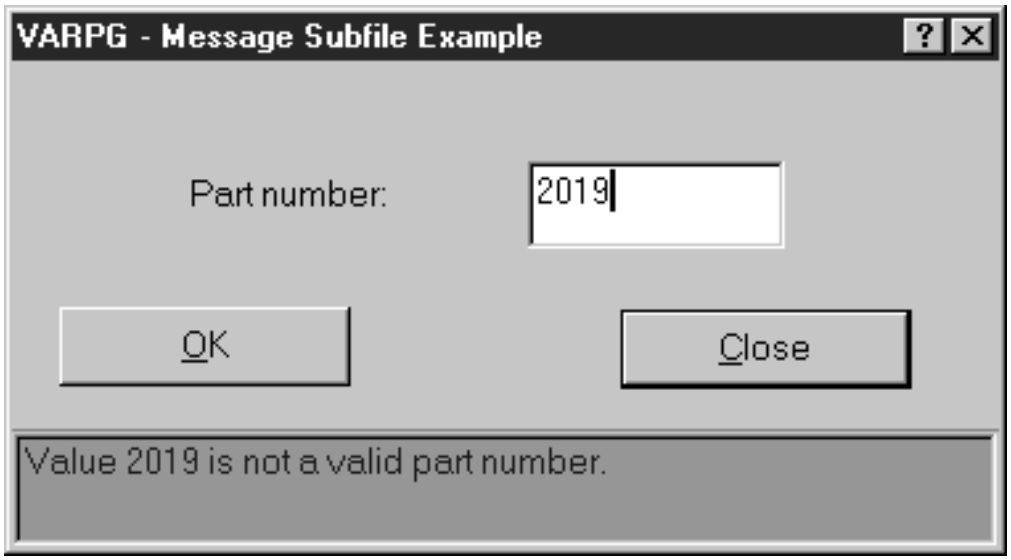
注: 置換データは、別の**MsgSubText** 属性が使用されるまで有効となっています。

メッセージの除去

メッセージ・サブファイル・パーツからメッセージを除去するには、**RemoveMsg** 属性を使用してください。除去されるメッセージの索引番号を指定してください。すべてのメッセージを除去するには、*0* の索引値を使用します。

メッセージ・サブファイルの例

この例では、処理するパーツ番号を入力するためのプロンプトがユーザーに表示されます。このパーツ番号は、ゼロより大きく、2000 より小さくなければなりません。**「OK」**プッシュボタンが押されると、値が必要な範囲になっているかどうかをプログラムがチェックします。値がこの範囲にない場合には、メッセージ・サブファイル・パーツにメッセージが追加されます。




```
*****************************************************************
*                                                               *
* プログラム ID  : MESSAGE                                      *
*                                                               *
* 説明 . . . . . : メッセージ・サブファイル・パーツを説明する   *
*                  サンプル・プログラム                         *
*                                                               *
*****************************************************************
*
*****************************************************************
*                                                               *
* ウィンドウ . . : MAIN                                         *
*                                                               *
* パーツ . . : PB_CLOSE                                         *
*                                                               *
* イベント . . : PRESS                                          *
*                                                               *
* 説明 . . . . . : プログラムの終了                             *
*                                                               *
*****************************************************************
*
C     PB_CLOSE      BEGACT    PRESS         MAIN
*
C                   move      *on           *inlr
*
C                   ENDACT
```

図 21. メッセージ・サブファイル・パーツを使用するコーディング例 (1/2)

```
*********************************************************************
*                                                                   *
* ウィンドウ . . : MAIN                                             *
*                                                                   *
* パーツ . . : PB_OK                                                *
*                                                                   *
* イベント . . : PRESS                                              *
*                                                                   *
* 説明 . . . : 入力された値が使用可能かどうかをチェックします。使用 *
*              可能でない場合には、メッセージ・サブファイル・パーツ *
*              にメッセージを追加します。                           *
*                                                                   *
*              入力された値は、メッセージ中の置換テキストとして     *
*              使用されます。                                       *
*                                                                   *
*********************************************************************
 *
C     PB_OK         BEGACT    PRESS         MAIN
 *
 * メッセージ・サブファイルの消去
 *
C     'Msg1'        setatr    0             'RemoveMsg'
 *
 * パーツ番号の取り出し
 *
C     'PartNum'     getatr    'Text'        tmp4              4 0
 *
 * パーツ番号が正しくない場合、メッセージ・パーツにメッセージ
 * を追加します。ユーザーによって入力されたパーツ番号は、
 * 置換テキストとして使用されます。
 * 置換テキストはストリングでなければならないので、数値の
 * パーツ番号を文字フィールドに移動し、それを置換テキスト
 * として使用します。
C     tmp4          ifle      *zero
C     tmp4          orgt      1999
C                   move      tmp4          char4             4
C     'Msg1'        setatr    char4         'MsgSubText'
C     'Msg1'        setatr    1             'AddMsgID'
 *
 * PartNum 入力フィールド FOCUS を与えてください。すると、
 * カーソルはそこに戻ります。
C     'PartNum'     setatr    1             'Focus'
 *
 * パーツ番号が有効です。処理は続行されます。
C                   else
 *                   ...
 *                   ...
 *                   ...
```

図 21. メッセージ・サブファイル・パーツを使用するコーディング例 (2/2)

複数行編集

ユーザーが複数行のテキストを入力できるようにしたい場合には、複数行編集パーツを使用します。

複数行編集パーツは境界を定義しています。テキストのすべてが表示されているとは限りません。ユーザーは、上下左右にスクロールして、現在表示されていないテキストを表示することができます。

パーツ属性

AddLineEnd	AddOffset	BackColor	BackMix
Bottom	CanUndo	CharOffset	Copy
CsrLine	CsrPos	Cut	Delete
DragEnable*	DropEnable*	Enabled	Focus
FontBold	FontItalic	FontName	FontSize
FontStrike*	FontUnder*	ForeColor	ForeMix
Handle*	Height	InsertLine	InsertText
Left	LineNumber	LineText	NbrOfLines
ParentName	PartName	PartType	Paste
ReadOnly	Refresh	ShowTips	Text
TextEnd	TextLength	TextSelect	TextStart
TextString	TipText	Top	TopLine
Undo	UserData	Visible	Width
WordWrap			

*** 注:** 制約事項については、その属性の説明を参照してください。

適用可能なイベント

Change	Click	Create	DblClick
Destroy	Drop	GotFocus	KeyPress
LostFocus	MouseDown	MouseEnter	MouseExit
MouseMove	MouseUp	Popup	VKeyPress

テキストの取り出しおよび設定

複数行編集パーツのテキストを取り出すか、またはセットするには、**Text** 属性を使用してください。

注: 複数行編集パーツのプロパティー・ノートブックに入力されたデフォルトのテキストは、保管されません。複数行編集パーツのテキストは、実行時にのみセットすることができます。

複数行編集パーツ中のテキスト行の処理

複数行編集パーツに新しい行を挿入するには、次のようにしてください。

1. テキストを挿入したい直前の行番号に **LineNumber** 属性をセットします。
2. **InsertLine** 属性を使用します。

   テキストは、指定した行の後に挿入されます。指定した行の下にあった行は、すべて挿入されたテキストのスペースを作るために下に移動します。

複数行編集パーツ中の文字の処理

複数行編集パーツに文字ストリングを挿入するには、次のようにしてください。

1. 新しいテキストを挿入したい場所を指定するように、**CharOffset** 属性をセットします。

   CharOffset 値に続くテキストは、新しいテキストで置き換えられます。
2. **AddOffset** 属性を使用して、テキストを **CharOffset** に追加します。

複数行編集パーツ中のテキストの選択部分の処理

いくつかの属性を使用して、複数行編集パーツ中のテキストの選択部分を処理することができます。

選択されたテキストだけを戻すには、**TextSelect** 属性を使用してください。テキストが選択されていない場合には、**TextSelect** 属性はヌル・ストリングを戻し、テキストを受け取るはずの結果フィールドは未変更のままです。

TextStart および **TextEnd** 属性を使用して、選択されたテキストの開始および終了文字位置を戻します。

カラーの変更

背景カラーがシステム・デフォルトにセットされているキャンバス・パーツに複数行編集パーツが存在する場合には、キャンバスの背景カラーに対する変更は複数行編集パーツに継承されます。キャンバスに追加された後続の複数行編集パーツは、このカラーを継承しません。これを訂正するには、すべての複数行編集パーツを配置するまで、キャンバスの背景カラーの設定を据え置いてください。代替方法として、キャンバスのカラーをシステム・デフォルトに設定してから、事前定義の RGB カラー設定に戻すことによって、複数行編集パーツにカラーを継承させることができます。

カラーを複数行編集パーツのスクロール・バーにドラッグ・アンド・ドロップした場合には、そのカラーは保管されません。複数行編集パーツは新規カラーに変更されますが、ウィンドウをクローズしてから再オープンした時には、カラーは元に戻ります。

フォントの選択

必ずしもすべてのフォントが複数行編集パーツによってサポートされるとは限りませ。このパーツのフォントを選択すると、パーツはその選択されたフォントに一番近いものを表示するように調整されます。

ユーザー入力の阻止

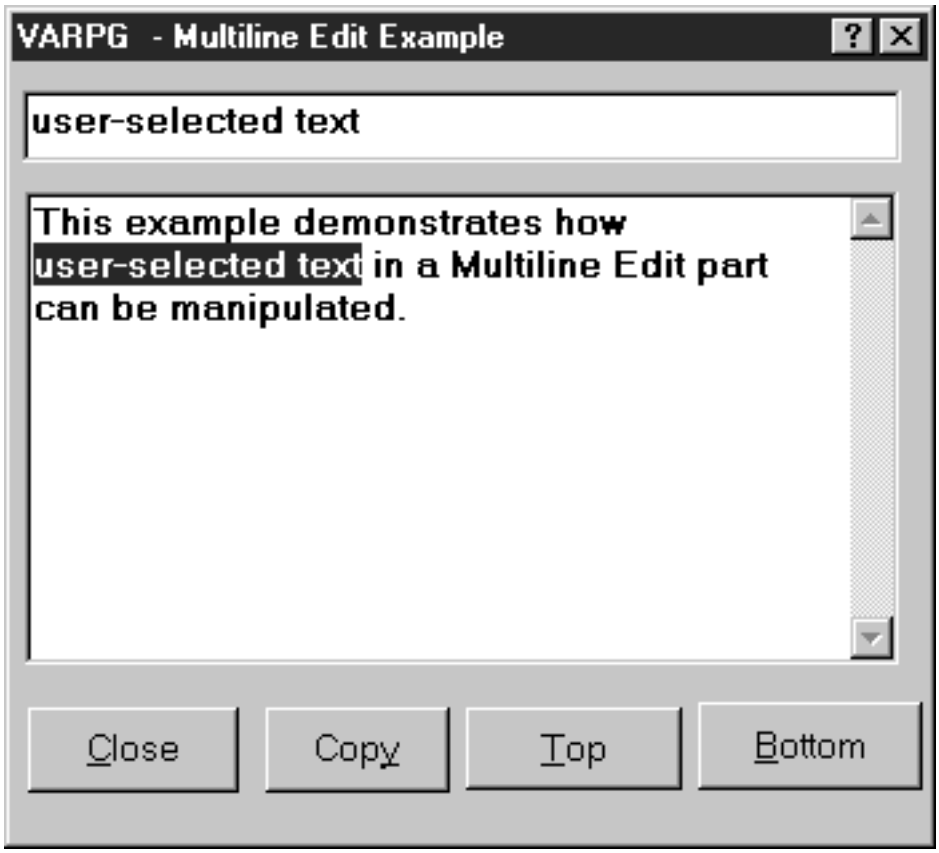
次の1 つ行うことによって、ユーザーが複数行編集パーツにテキストを入力できないようにすることができます。

- **ReadOnly** 属性を *0* にセットする。
- **Enabled** 属性を *0* にセットする。 (またこの場合、複数行編集パーツが **Change** や **GotFocus** などのイベントに応答できなくなります。)

プログラム中で複数行編集パーツの値を変更することもできます。

複数行編集の例

この例では、**「コピー」**プッシュボタンを押して、選択したテキストを複数行編集から入力フィールドにコピーします。**「クローズ」**プッシュボタンを押すと、プログラムが終了します。



```
*****************************************************************
*                                                               *
* プログラム ID  : MLE                                          *
*                                                               *
* 説明 . . . . . : 複数行編集パーツを例示するサンプル・         *
*                   プログラム。                                *
*                                                               *
*****************************************************************
*****************************************************************
*                                                               *
* ウィンドウ . . : MAIN                                         *
*                                                               *
* パーツ . . : PB_CLOSE                                         *
*                                                               *
* イベント . . : PRESS                                          *
*                                                               *
* 説明 . . . . . : プログラムの終了                             *
*                                                               *
*****************************************************************
 *
C     PB_CLOSE      BEGACT    PRESS         MAIN
 *
C                   move      *on           *inlr
 *
C                   ENDACT
*****************************************************************
*                                                               *
* ウィンドウ . . : MAIN                                         *
*                                                               *
* パーツ . . . : PB_COPY                                        *
*                                                               *
* イベント . . : PRESS                                          *
*                                                               *
* 説明 . . . . . : MLE 中の選択されたテキストを入力フィールド・ *
*                   パーツにコピーします。                      *
*                                                               *
*****************************************************************
 *
C     PB_COPY       BEGACT    PRESS         MAIN
 *
C     'EF1'         setatr    *blanks       'Text'
C     'MLE1'        getatr    'TextStart'   start             5 0
C     'MLE1'        getatr    'TextSelect'  selected        128
 *
C     start         ifgt      *zero
C     'ef1'         setatr    selected      'Text'
C                   endif
 *
C                   ENDACT
```

図 22. 複数行編集パーツを使用するコーディング例 (1/2)

```
*********************************************************************
*                                                                   *
* ウィンドウ . . : Main                                             *
*                                                                   *
* パーツ . . . . : Top                                              *
*                                                                   *
* イベント . . : Press                                              *
*                                                                   *
* 説明 . . . . . : MLE の 5 行目を最上行として設定します。*
*                                                                   *
* 変更活動 . . . :                                                 *
*                                                                   *
*********************************************************************
*
C     TOP           BEGACT    PRESS         MAIN
*
C                   eval      %setatr('MAIN':'MLE1':'TOPLINE') = 5
C                   ENDACT
*********************************************************************
*                                                                   *
* ウィンドウ . . : Main                                             *
*                                                                   *
* パーツ . . . : Bottom                                             *
*                                                                   *
* イベント . . : Press                                              *
*                                                                   *
* 説明 . . . . : 最下部の設定                                       *
*                                                                   *
*********************************************************************
*
C     BOTTOM        BEGACT    PRESS         MAIN
*
C                   eval      %setatr('MAIN':'MLE1':'TOPLINE') = 0
C                   ENDACT
```

図 22. 複数行編集パーツを使用するコーディング例 (2/2)

ノートブック

名前、出荷先アドレス、注文、信用などのカテゴリーにソートされたカスタマー情報といったトピックで論理的に類別できるデータを提示するには、ノートブック・パーツを使用してください。

ノートブック・パーツは、バインド・ノートブックのグラフィカル表現です。(Windows アプリケーションでは、これはタブ制御として知られています。)ユーザーはノートブックにページを追加することができ、またそれらのページを、タブ付きの区切り記号によって分離されたセクションにグループ化することができます。ノートブック・ページにキャンバスがある場合には、ページに複数のパーツを追加することができます。ノートブック・ページにキャンバスがない場合には、ページに 1 つのパーツしか追加することができません。

ユーザーは、ノートブックのページをめくって、あるページから次のページに移動することもできるし、区切り記号タブをクリックして、あるセクションに直接進むこともできます。

次の方法でノートブック・ページを追加することができます。
- ノートブック・パーツ用のプロパティー・ノートブックを使用する。
- プロパティー・タブ またはキャンバス付きノートブック・ページを、ノートブック・パーツで指示してクリック (ドラッグ・アンド・ドロップ) する。

関連情報については、以下を参照してください。
- 125 ページの『ノートブック・ページ』
- 127 ページの『キャンバス付きノートブック・ページ』

パーツ属性

BackColor	BackMix	Bottom	Count
Enabled	Focus	FontBold	FontItalic
FontName	FontSize	FontStrike*	FontUnder*
ForeColor	ForeMix	Handle*	Height
Left	PageNumber	ParentName	PartName
PartType	Refresh	ShowTabs*	Top
UserData	Visible	Width	

***注:** 制約事項については、属性の説明を参照してください。

適用可能なイベント

Create	Destroy

フォント強調の変更

ノートブック・パーツのフォント強調を下線または取り消し線に変更すると、タブ・テキストではなく状況テキストに新規強調が取り込まれます。

ノートブック・ページ

ノートブックにページを追加するためにはノートブック・ページ・パーツを使用します。

ユーザーはノートブック・ページに 1 つのパーツしか追加できません。そのパーツは、ページ全体に適合するように自動的にサイズ設定されます。ページに複数のパーツを追加したい場合には、ノートブック・ページでキャンバス・パーツを指示してクリックしなければなりません。代替方法として、ユーザーはキャンバス・パーツ付きノートブック・ページを使用して、ステップを保管することができます。

注: このパーツのプロパティー、イベントなどを操作できるのは、プロジェクト・ツリー・ビューのそのポップアップ・メニューからだけです。

ユーザーは左および右矢印キーを押して 1 つのページから次のページに移動することができます。

関連情報については、次を参照してください。
- 124 ページの『ノートブック』
- 127 ページの『キャンバス付きノートブック・ページ』

パーツ属性

Enabled	OnTop	ParentName	PartName
PartType	Refresh	TabImage	TabLabel
UserData	Visible		

適用可能なイベント

Create	Destroy	PageSelect	SelPending*

*** 注:** 制約事項については、イベント記述を参照してください。

タブ・テキストの表示

DBCS マシンでは、MINCHO プロポーショナル・フォントが使用された時に、ノートブック・ページのタブにそのテキスト全体が表示されない場合があります。MINCHO ノーマルや MINCHO システムなどの別のスタイルにフォントを変更すると、この問題が修正されます。

簡略記号の設定

ノートブック・ページの簡略記号キーを指定するには、**Label** 属性のテキストの文字の前に簡略記号 ID を入れます。この指定された文字が下線付きでインターフェース上に表示されます (たとえば、**D**isplay)。 Windows の場合、簡略記号は表示されますが、ノートブック・ページでは機能しないということに注意してください。

注: 略号は Java アプリケーションではサポートされていません。

キャンバス付きノートブック・ページ

ノートブック・パーツにページを追加するためにはキャンバス付きのノートブックを使用します。

キャンバス・パーツは、ノートブック・ページ・パーツのクライアント域を占めます。キャンバス・パーツにパーツを追加することによって、ユーザーはグラフィカル・ユーザー・インターフェースを作成することができます。

ページに 1 つのパーツのみを追加したい場合には、キャンバス・パーツ付きのノートブック・ページの代わりに、ノートブック・ページ・パーツを使用することができます。ノートブック・ページ・パーツは、その中にキャンバスを持っていないので、追加したパーツは自動的にサイズが決められます。

関連情報については、以下を参照してください。
- 124 ページの『ノートブック』
- 125 ページの『ノートブック・ページ』

パーツ属性

Enabled	OnTop	ParentName	PartName
PartType	Refresh	TabImage	TabLabel
UserData	Visible		

適用可能なイベント

Create	Destroy	PageSelect	SelPending

ODBC/JDBC インターフェース

ODBC/JDBC インターフェース・パーツは、Windows ODBC API または Sun Microsystems の JDBC API をサポートするデータベース・ファイルを処理できる機能を提供します。これらのデータベース・ファイル・タイプの例としては、Foxpro、Access、および Paradox があります。

ODBC/JDBC インターフェース・パーツを使用できるアプリケーションを開発するには、SQL に精通している必要があり、Windows ODBC SDK または Sun Microsystems の Java 2 ソフトウェア開発キット (J2SDK) 標準版のいずれかがワークステーションにインストールされていなければなりません。

ODBC SDK がない場合には、次の URL の Microsoft からダウンロードすることができます。

```
http://www.microsoft.com/odbc/download.htm
```

JDBC サポートは、Java™ 2 ソフトウェア開発キット (J2SDK) バージョン 1.2 (Windows 用) のパーツです。J2SDK がない場合には、Sun Microsystems から次の URL でダウンロードすることができます。

```
http://java.sun.com/products/
```

JDBC データベースのデータにアクセスして操作するアプリケーションには、適切な JDBC 2.0 承諾ドライバーが必要です。 JDBC ドライバーおよびその他の情報は次の URL で検索することができます。

```
http://java.sun.com/products/jdbc/
```

注: JDBC はアプレットではサポートされません。

ODBC または JDBC データベースは 1 つまたは複数のテーブルから構成されています。データは一連の行としてテーブルに保管されます。各行またはレコードには、データの入った多数の列が含まれています。行を操作するために、あるいはプログラム・フィールドとテーブルの列の間でデータを移動するために、ユーザーのプログラムは ODBC/JDBC インターフェース・パーツ属性とともに SQL ステートメントを実行依頼することができます。

既存のデータベースを処理するためには、その前にユーザーの VARPG プログラムをデータベースに接続し、どのテーブルを参照するかを指示しなければなりません。テーブル中の行を操作するためには、ユーザーのプログラムで、 ODBC/JDBC インターフェース・パーツによって戻され保守されるレコードを識別するレコード・セットを作成しなければなりません。行の中のデータをアクセスするためには、テーブル行で使用される各列を、プログラム中のプログラム・フィールドとバインドしなければなりません。Java アプリケーションでは、ポインターはサポートされません。列はパーツにバインドされます。バインドに使用できるパーツは、静的テキストおよび入力フィールドのパーツだけです。

ODBC/JDBC インターフェース・パーツを使用する Java アプリケーションを作成する場合には、そのアプリケーションを実行するエンド・ユーザーはワークステーションに *varpgjdb.jar* ファイルをインストールし、そのクラスパス・ステートメントにその場所を追加しなければなりません。 パッケージ・ユーティリティーにはこの JAR ファイルが含まれています。この JAR ファイルは *WDSC\java* サブディレクトリーにあります。

パーツ属性

AllowChg*	BindPart	Bottom	BufferDec*
BufferLen*	BufferPtr*	BufferType*	CharData
Column	ColumnDec	ColumnLen	ColumnName
Columns	ColumnType	Connect	Connected
ConnectStr	CurrentRow	DeleteRow	ExecuteSQL
Fetch	FetchNext	FetchPrior	GetTables
Handle*	Height	InsertRow	IsData
Left	ParentName	PartName	PartType
Refresh	Rows*	SQLError	SQLMsgBox
SQLQuery	Top	UnBind	UpdateRow
UserData	Visible	Width	

***注:** 制約事項については、属性の説明を参照してください。

適用可能なイベント

Create	Destroy

ODBC データベースへの接続

既存のデータベースを処理するためには、その前にユーザーの VARPG プログラムをデータベースに接続し、どのテーブルを参照するかを指示しなければなりません。

注: VARPG ODBC/JDBC インターフェース・パーツは一度に 1 つのテーブルにしか接続できません。そのテーブルが他のテーブルに依存しているか、関連がある場合には、ユーザーのプログラムはデータベース中のレコードを更新または削除することができません。

データベースに接続するためには、最初に ConnectStr 属性をデータベースの必須接続ストリングに設定してください。その後で、Connect 属性を使用して接続してください。Windows では、ConnectStr 属性を *BLANKS に設定した場合には、ODBC マネージャーは、接続するテーブルを選択できる「データ・ソースの選択」ダイアログを表示します。接続が行われたら、Connected 属性は 1 に設定されます。接続が失敗したら、Connected 属性は 0 に設定されます。

次のコード・フラグメントが実行される時に、「データ・ソースの選択」ダイアログが表示されます。テーブルが選択されている場合には、ConnectStr 属性は接続ストリングを含めます。Connect 属性は接続を行うように設定されます。

```
C     'ODBC'        Setatr    *Blanks       'ConnectStr'
C     'ODBC'        Setatr    1             'Connect'
C                   If        %Getatr('Main':'ODBC':'Connected')=1
```

```
C                   ...
C                   Else
C                   ...
C                   EndIf
```

レコード・セットの作成

データベースに接続したら、そのデータベース中のデータにアクセスする前に、レコード・セットを作成しなければなりません。レコード・セットを作成するためには、SQLQuery および ExecuteSQL 属性を使用して SELECT ステートメントを ODBC/JDBC インターフェース・パーツに発信してください。SELECT ステートメントは、データベース中のどのテーブルがアクセスされ、テーブル中のどのレコード・グループが処理されるのかを識別します。

次のコード・セグメントは、テーブル CUSTOMERS の中のすべてのレコードのレコード・セットを作成する例です。

```
D SelAll          C                   'Select * From "Customers"'
 *
C     'ODBC'        Setatr    SelAll        'SQLQuery'
C     'ODBC'        Setatr    1             'ExecuteSQL'
```

テーブル・データのアクセス

データは一連の行としてテーブルに保管されます。各行には、データの入った多数の列が含まれています。ユーザーは FetchNext、FetchPrior、UpdateRow などの ODBC/JDBC インターフェース・パーツを使用して、テーブル中の行を操作することができます。ただし、行の中のデータをアクセスするためには、テーブル行の中の各列をプログラム中プログラム・フィールドとバインドしなければなりません。このバインディングが設定されたら、ODBC/JDBC インターフェース・パーツは、プログラム・フィールドとテーブル列の間でデータを移動することができます。

プログラム・フィールドをバインドするためには、次の ODBC/JDBC インターフェース・パーツ属性を使用します。

列　テーブル中のどの列がバインドされるかを明確にします。

BufferPtr
列にバインドするプログラム・フィールドのアドレスを含めます。

BufferDec
バッファー列の小数点以下の桁数を指定します。

BufferLen
プログラム・フィールドの長さを指定します。

BufferType
プログラム・フィールドのデータ・タイプを指示します。

次の例では、D 仕様に定義されている 2 つのフィールドがテーブル中の列 1 および 2 にバインドされます。

```
D first           S             20
D last            S             30
 *
D fptr            S               *   INZ(%Addr(first))
D lptr            S               *   INZ(%Addr(last))
 *
```

```
C     'ODBC1'       Setatr    1             'Column'
C     'ODBC1'       Setatr    20            'BufferLen'
C     'ODBC1'       Setatr    fptr          'BufferPtr'
C     'ODBC1'       Setatr    1             'BufferType' *
C     'ODBC1'       Setatr    2             'Column'
C     'ODBC1'       Setatr    30            'BufferLen'
C     'ODBC1'       Setatr    lptr          'BufferPtr'
C     'ODBC1'       Setatr    1             'BufferType'
```

また、C 仕様で %ADDR 組み込みディレクトリーを使用して、ポインターを定義する D 仕様のコーディングを避けることもできます。

```
C                   Eval      %Setatr(`Main':'ODBC1':'BufferPtr')=%Addr(first)
```

データ・タイプ

BufferType 属性は、BufferPtr 属性で参照されたプログラム・フィールドのデータ・タイプを示すために使用します。 ODBC/JDBC インターフェース・パーツは、BufferType 属性を使用して、プログラム・フィールドとテーブル列間でデータを移動する時に、正しいデータ変換を実行します。フィールド・タイプが適切かどうかの検査は行われないので、この属性を正しく設定することが重要です。

BufferType 属性を使用する前に、Column 属性を設定してください。プログラム・フィールドがインターフェース上のパーツと関連付けられている場合には、DataType 属性を使用してバッファー・タイプを取得することができます。

対応する VARPG データ・タイプをサポートされている SQL データ・タイプに設定するためには、次の図表を使用してください。図表にリストされている BufferLen および BufferDec 属性だけを指定してください。

文字、10 進数、整数、または短整数データ・タイプの場合には、BufferLen 属性のみを指定してください。

倍精度、浮動、および実データ・タイプは、VARPG では浮動 (F) または Zoned として定義できることに注意してください。これらを Zoned として定義した場合には、VARPG ランタイムは、列からのデータの移動時に BufferDec で指定された小数点以下の桁数のみを使用します。この方式では、データ・ソースが BufferDec 属性で指定された小数点以下の桁数より大きい場合に、精度が失われることがあります。これらのフィールドを浮動 (F) として定義する場合には、BufferLen または BufferDec 属性を指定**しない**ようにしてください。

SQL データ・タイプ	VARPG データ・タイプ	プログラム・フィールドの長さを指定 (BufferLen を使用) します	バッファー列の小数点以下の桁数を指定 (BufferDec を使用) します
文字	CHAR	X	
小数部分	Zoned	X	
整数	Zoned	X	
短整数	Zoned	X	
倍精度	8F		
倍精度	Zoned	X	X
浮動	4F		
浮動	Zoned	X	X
実数	4F		

SQL データ・タイプ	VARPG データ・タイプ	プログラム・フィールドの長さを指定 (BufferLen を使用) します	バッファー列の小数点以下の桁数を指定 (BufferDec を使用) します
実数	Zoned	X	X

ODBC/JDBC インターフェース・パーツでサポートされないデータ・タイプが列に入っている場合には、その列の AllowChg 属性を 0 に設定してください。ODBC/JDBC インターフェース・パーツは、プログラム・フィールドとその列の間のデータ移動を行わなくなります。データはもとのままです。

テーブル行の検索

テーブル中の行を処理するためには、最初にレコード・セットを作成しなければなりません。レコード・セットは、ODBC/JDBC インターフェース・パーツによって戻され保守されるレコードのグループです。ユーザーのプログラムは、SQLQuery および ExecuteSQL 属性を使用することにより、SQL ステートメントを ODBC/JDBC インターフェース・パーツに発信します。最初に SQLQuery 属性が、実行する SQL ステートメントに設定されます。次に、ExecuteSQL 属性が 1 に設定されて照会を実行します。

次の例では、すべてのレコードがテーブル Customers から選択されます。

```
D SelAll          C                   'Select * From "Customers"'
 *
C     'ODBC1'       Setatr    SelAll        'SQLQuery'
C     'ODBC1'       Setatr    1             'ExecuteSQL'
```

SQLQuery の結果として戻された行数を判別するためには、Rows 属性の値を調べることができます。

レコード・セットが戻されたら、FetchNext および FetchPrior 属性を使用して各行を処理することができます。レコード・セット中の次の行を戻すためには、FetchNext 属性を 1 に設定してください。レコード・セット中の前の行を戻すためには、FetchPrior 属性を 1 に設定してください。FetchNext または FetchPrior が正常に行を 戻したかどうかを判別するためには、IsData 属性の値を調べてください。値 1 は、データが戻されたことを示します。そうでない場合には、IsData の値は 0 に設定されます。

次の例では、レコード・セット中のすべてのレコードが読み取られ、列 1 の値 (フィールド first) がリスト・ボックス LB1 に追加されます。

```
C     'ODBC1'       Setatr    1             'FetchNext'
C     'ODBC1'       Getatr    'IsData'      Temp              1 0
 *
C                   DoW       Temp = 1
C     'LB1'         Setatr    first         'AddItemEnd'
C     'ODBC1'       Getatr    'IsData'      Temp
C                   EndDo
```

行データの更新

行の中のデータを更新するためには、UpdateRow 属性を使用して、更新する行を指定します。 UpdateRow によってすべての行が更新されてしまうことに注意してくだ

さい。最初に行を取り出す必要はありません。しかし、通常は取り出したばかりの行を更新します。この場合には、CurrentRow 属性を使用します。この属性には、取り出したばかりの行の行番号が入っています。

次のコード・セグメントでは、1 つの行が読み取られて、情報がウィンドウに表示されていると見なします。ユーザーは変更を行った後で、更新ボタンを押します。

```
C     PB_Update     BEGACT    PRESS         Main
 *
C                   Read      'Main'
C     'ODBC1'       Getatr    'CurrentRow'  Row              1 0
C     'ODBC1'       Setatr    Row           'UpdateRow'
 *
C                   ENDACT
```

行の削除

行の削除は行の更新と類似しています。(132 ページの『行データの更新』 を参照してください。) DeleteRow 属性を使用して、削除する行を指定します。UpdateRow 属性と同様に、DeleteRow によって、行セット中のすべての行が削除されることになります。最初に行を取り出す必要はありません。

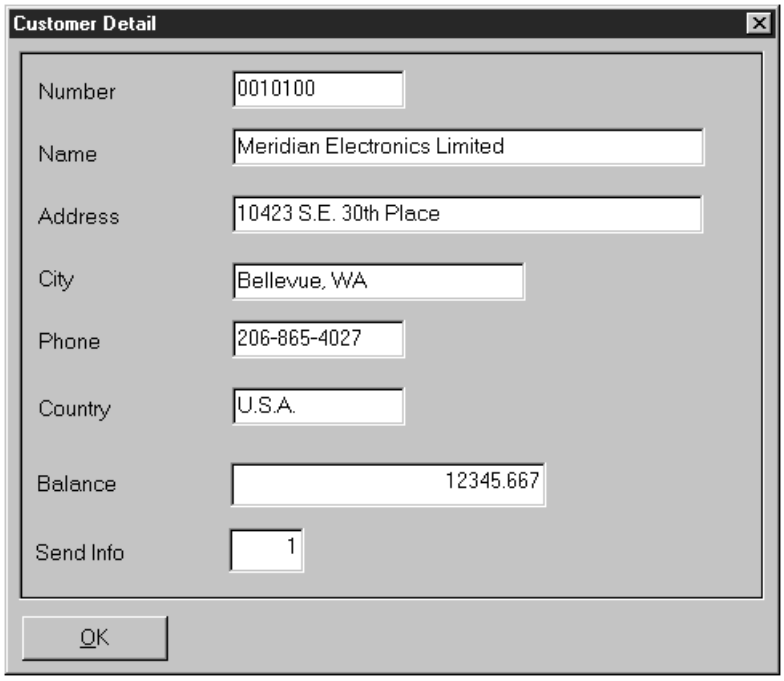
次の例では、取り出されたばかりで現在ウィンドウに表示されているレコードを削除するためにユーザーが「削除」プッシュボタンを押しています。

```
C     PB_Delete     BEGACT    PRESS         Main
 *
C     'ODBC1'       Getatr    'CurrentRow'  Row              1 0
C     'ODBC1'       Setatr    Row           'DeleteRow'
 *
C                   ENDACT
```

ODBC/JDBC インターフェース・パーツの例

次の例では Microsoft Access で作成されたデータベースを使用します。このデータベースには *CUSTOMERS* という名前のテーブルが 1 つあります。この簡単な照会プログラムはカスタマーに関する詳細が入っているウィンドウを表示します。プッシュボタンは、次のレコードおよび直前のレコードに進むためのボタンと、現行レコードを更新および削除するボタンが提供されます。

次の図は照会ウィンドウを表示したものです。



この例のコードは次の通りです。

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

 * 接続ストリングを定義します
 *
D ConnectStr      C                   'DSN=MS Access 97 Database;DBQ=-
D                                     CelDial.mdb;-
D                                     DriverId=25;FIL=-
D                                     MS Access;MaxBufferSiz'
 * 作動中の変数
DClr              S               2  0
DCol              S              10
DSQL              S             255A
D%ColNumber       S               2  0
D%Part            S              10
D%Character       S               2
D AppStart        C                   'HourGlas.ANI'
 *
DDel              M                   MsgNbr(*MSG0003)
D                                     MsgData(CustNo:CustNa)
D J               S               4  0
D I               S               4  0
 *
D CSENDINFO       S               1S 0
 *
 * フィールド・バッファーのポインターを定義します
 *
DCBalance         S              18S 3                                    収支 (二重数値)
DP_001            S                *                                      番号
DP_002            S                *                                      名前
DP_003            S                *                                      代表番号
DP_004            S                *                                      連絡先
DP_005            S                *                                      電話番号
DP_006            S                *                                      ファクシミリ
DP_007            S                *                                      住所
DP_008            S                *                                      市町村
DP_009            S                *                                      国
DP_010            S                *                                      郵便番号
DP_011            S                *                                      郵便番号の地域
DP_012            S                *                                      収支
DP_013            S                *                                      市場情報の送信先?
 *
 * すべてのレコードを選択します
 *
DSelAll           C                   'Select * From "Customers"'
```

図 23. ODBC/JDBC 照会プログラム例のコード (1/11)

```
 ********************************************************************
 *                                                                  *
 * ウィンドウ/パーツ/イベント:                                      *
 *                                                                  *
 * 説明: プログラム・フィールドを列にバインドし、データベース・     *
 *       テーブル CUSTOMERS に接続します                            *
 *                                                                  *
 ********************************************************************
 *
C     Main          BEGACT    CREATE        Main
 *
C     'Main'        Setatr    appstart      'MouseIcon'
C     'ODBC'        Setatr    0             'Visible'
C                   MoveL     'ASC '        Seq               4
C                   Eval      CLR = *White
C                   Move      '255:255:255' Mix
C     'SFL1'        Setatr    1             'SizeToFit'
C     'SFL1'        Getatr    'BackColor'   RowClr            2 0
 * フィールドを列にバインドします
 *
 * バインド列: 番号
C     'ODBC'        SetAtr    1             'Column'
C     'ODBC'        SetAtr    7             'BufferLen'
C     'ODBC'        SetAtr    1             'BufferType'
C                   Eval      P_001=%Addr(CUSTNO)
C     'ODBC'        SetAtr    P_001         'BufferPtr'
 *
 * バインド列: 名前
C     'ODBC'        SetAtr    2             'Column'
C     'ODBC'        SetAtr    40            'BufferLen'
C     'ODBC'        SetAtr    1             'BufferType'
C                   Eval      P_002=%Addr(CUSTNA)
C     'ODBC'        SetAtr    P_002         'BufferPtr'
 *
 * バインド列: 代表番号
C     'ODBC'        SetAtr    3             'Column'
C     'ODBC'        SetAtr    5             'BufferLen'
C     'ODBC'        SetAtr    1             'BufferType'
C                   Eval      P_003=%Addr(Repno)
C     'ODBC'        SetAtr    P_003         'BufferPtr'
 *
 * バインド列: 連絡先
C     'ODBC'        SetAtr    4             'Column'
C     'ODBC'        SetAtr    30            'BufferLen'
C     'ODBC'        SetAtr    1             'BufferType'
C                   Eval      P_004=%Addr(Contac)
C     'ODBC'        SetAtr    P_004         'BufferPtr'
 *
```

図 23. ODBC/JDBC 照会プログラム例のコード (2/11)

```
 * バインド列: 電話番号
C     'ODBC'        SetAtr    5             'Column'
C     'ODBC'        SetAtr    17            'BufferLen'
C     'ODBC'        SetAtr    1             'BufferType'
C                   Eval      P_005=%Addr(CPhone)
C     'ODBC'        SetAtr    P_005         'BufferPtr'
 *
 * バインド列: ファクシミリ
C     'ODBC'        SetAtr    6             'Column'
C     'ODBC'        SetAtr    17            'BufferLen'
C     'ODBC'        SetAtr    1             'BufferType'
C                   Eval      P_006=%Addr(CFax)
C     'ODBC'        SetAtr    P_006         'BufferPtr'
 *
 * バインド列: 住所
C     'ODBC'        SetAtr    7             'Column'
C     'ODBC'        SetAtr    40            'BufferLen'
C     'ODBC'        SetAtr    1             'BufferType'
C                   Eval      P_007=%Addr(CAddr)
C     'ODBC'        SetAtr    P_007         'BufferPtr'
 *
 * バインド列: 市町村
C     'ODBC'        SetAtr    8             'Column'
C     'ODBC'        SetAtr    30            'BufferLen'
C     'ODBC'        SetAtr    1             'BufferType'
C                   Eval      P_008=%Addr(CCity)
C     'ODBC'        SetAtr    P_008         'BufferPtr'
 *
 * バインド列: 国
C     'ODBC'        SetAtr    9             'Column'
C     'ODBC'        SetAtr    20            'BufferLen'
C     'ODBC'        SetAtr    1             'BufferType'
C                   Eval      P_009=%Addr(CCount)
C     'ODBC'        SetAtr    P_009         'BufferPtr'
 *
 * バインド列: 郵便番号
C     'ODBC'        SetAtr    10            'Column'
C     'ODBC'        SetAtr    10            'BufferLen'
C     'ODBC'        SetAtr    1             'BufferType'
C                   Eval      P_010=%Addr(CZip)
C     'ODBC'        SetAtr    P_010         'BufferPtr'
 *
 * バインド列: 郵便番号の地域
C     'ODBC'        SetAtr    11            'Column'
C     'ODBC'        SetAtr    1             'BufferLen'
C     'ODBC'        SetAtr    1             'BufferType'
C                   Eval      P_011=%Addr(CZiplo)
C     'ODBC'        SetAtr    P_011         'BufferPtr'
 *
```

図 23. ODBC/JDBC 照会プログラム例のコード (3/11)

```
 * バインド列: 収支
C     'ODBC'        SetAtr    12            'Column'
C     'ODBC'        SetAtr    18            'BufferLen'
C     'ODBC'        SetAtr    0             'BufferType'
C     'ODBC'        SetAtr    3             'BufferDec'
C                   Eval      P_012=%Addr(CBalance)
C     'ODBC'        SetAtr    P_012         'BufferPtr'
C     'ODBC'        SetAtr    ConnectStr    'ConnectStr'
 *
 * バインド列: 情報の送信
C     'ODBC'        SetAtr    13            'Column'
C     'ODBC'        SetAtr    1             'BufferLen'
C     'ODBC'        SetAtr    0             'BufferType'
C     'ODBC'        SetAtr    0             'BufferDec'
C                   Eval      P_013=%Addr(CSendInfo)
C     'ODBC'        SetAtr    P_013         'BufferPtr'
 *
 * データベースに接続し、すべてのレコードを選択します
C     'ODBC'        SetAtr    1             'Connect'
C     'ODBC'        SetAtr    SelAll        'SQLQuery'
C     'ODBC'        SetAtr    1             'ExecuteSQL'
 *
C     'Main'        Setatr    1             'ProgresBar'
C     'ODBC'        Getatr    'Rows'        Rows              4 0
C     'Main'        Setatr    Rows          'PBRange'
C                   Eval      %Setatr('Main':'Main':'PBStepSize')=110
 *
C                   Z-Add     22            MaxRows           2 0
C                   Exsr      Fill
 *
C                   ENDACT
 ************************************************************************
 *                                                                      *
 * ウィンドウ/パーツ/イベント: Main/PB_Close/Press                      *
 *                                                                      *
 * 説明: プログラムを終了します                                         *
 *                                                                      *
 ************************************************************************
 *
C     PB_CLOSE      BEGACT    PRESS         Main
 *
C                   Move      *ON           *INLR
 *
C                   ENDACT
```

図 23. ODBC/JDBC 照会プログラム例のコード (4/11)

```
 ********************************************************************
 *                                                                  *
 * ウィンドウ/パーツ/イベント: Main/PB_OK/Press                       *
 *                                                                  *
 * 説明: 詳細ウィンドウをクローズします                               *
 *                                                                  *
 ********************************************************************
 *
C     PB_OK         BEGACT    PRESS         DETAIL
 *
C                   ClsWin    'Detail'
 *
C                   ENDACT
 ********************************************************************
 *                                                                  *
 * ウィンドウ/パーツ/イベント: Main/SFL1/Enter                        *
 *                                                                  *
 * 説明: 選択されたカスタマーの詳細を表示します                       *
 *                                                                  *
 ********************************************************************
 *
C     SFL1          BEGACT    ENTER         MAIN
 *
C                   ReadS     SFL1
C                   Eval      SQL='SELECT * FROM CUSTOMER WHERE CUSTNO='+
C                               Custno
C                   ShowWin   'Detail'                            80
C                   Write     'Detail'
C                   Eval      %Setatr('Detail':'Detail':'Focus')=1
 *
C                   ENDACT
 ********************************************************************
 *                                                                  *
 * ウィンドウ/パーツ/イベント: Main/PB_Columns/Press                  *
 *                                                                  *
 * 説明: オプション・ウィンドウを表示します                          *
 *                                                                  *
 ********************************************************************
 *
C     PB_COLUMNS    BEGACT    PRESS         MAIN
 *
C                   ShowWin   'Columns'                           88
C                   Eval      %Setatr('Columns':'Columns':'Focus')=1
 *
C                   ENDACT
```

図 23. ODBC/JDBC 照会プログラム例のコード (5/11)

```
 ********************************************************************
 *                                                                  *
 * ウィンドウ/パーツ/イベント: Columns/PB_Cancel/Press              *
 *                                                                  *
 * 説明: オプション・ウィンドウをクローズします                     *
 *                                                                  *
 ********************************************************************
 *
C     PB_CANCEL     BEGACT    PRESS         COLUMNS
 *
C                   ClsWin    'Columns'
 *
C                   ENDACT
 ********************************************************************
 *                                                                  *
 * ウィンドウ/パーツ/イベント: Main/SFL1/ColSelect                   *
 *                                                                  *
 * 説明: 選択された列によって列をソートします                       *
 *                                                                  *
 ********************************************************************
 *
C     SFL1          BEGACT    COLSELECT     MAIN
 *
C                   Eval      SQL='SELECT * FROM CUSTOMERS ORDER BY ' +
c                             %EditC(%ColNumber:'1') + ' ' + Seq
C     'ODBC'        Setatr    SQL           'SQLQuery'
C     'ODBC'        Setatr    1             'ExecuteSQL'
C                   Exsr      Fill
 *
C                   ENDACT
 ********************************************************************
 *                                                                  *
 * ウィンドウ/パーツ/イベント: Main/PB_Update/Press                  *
 *                                                                  *
 * 説明: 更新済みの変更されたレコード                               *
 *                                                                  *
 ********************************************************************
 *
C     PB_UPDATE     BEGACT    PRESS         MAIN
 *
C                   ReadC     SFL1                                  99
 *
C                   If        NOT *IN99
C     'SFL1'        Getatr    'FirstSel'    Sel               4 0
C     'ODBC'        Setatr    Sel           'UpdateRow'
 *
C                   Else
C     *MSG0002      Dsply                   mRC               9 0
C                   EndIf
 *
C                   ENDACT
```

図 23. ODBC/JDBC 照会プログラム例のコード (6/11)

```
 ********************************************************************
 *                                                                  *
 * ウィンドウ/パーツ/イベント: Main/PB_Delete/Press                 *
 *                                                                  *
 * 説明: 選択されたレコードを削除します                             *
 *                                                                  *
 ********************************************************************
 *
C     PB_DELETE     BEGACT    PRESS         MAIN
 *
C                   ReadS     SFL1                                  99
 *
C                   If        NOT *in99
C     Del           Dsply                   mRC
 *
C                   If        mRC=*YESButton
C     'SFL1'        Getatr    'FirstSel'    Sel              4 0
C     'ODBC'        Setatr    Sel           'DeleteRow'
C                   EndIf
 *
C                   EndIf
 *
C                   ENDACT
 ********************************************************************
 *                                                                  *
 * ウィンドウ/パーツ/イベント: Main/PB_Opt/Press                    *
 *                                                                  *
 * 説明: オプション・ウィンドウを表示します                         *
 *                                                                  *
 ********************************************************************
 *
C     PB_Opt        BEGACT    PRESS         MAIN
 *
C                   ShowWin   'Columns'                          88
 *
C                   ENDACT
 ********************************************************************
 *                                                                  *
 * ウィンドウ/パーツ/イベント: Columns/CB_Hrule/Select              *
 *                                                                  *
 * 説明: 水平方向の罫線を切り替えます                               *
 *                                                                  *
 ********************************************************************
 *
C     CB_HRULE      BEGACT    SELECT        COLUMNS
 *
C                   Eval      %Setatr('Main':'SFL1':'HRule')=
C                               %Getatr('Columns':'CB_HRule':'Checked')
 *
C                   ENDACT
```

図 23. ODBC/JDBC 照会プログラム例のコード (7/11)

```
 ********************************************************************
 *                                                                  *
 * ウィンドウ/パーツ/イベント: Columns/CB_VRule/Select              *
 *                                                                  *
 * 説明: 縦方向の罫線を切り替えます                                 *
 *                                                                  *
 ********************************************************************
 *
C     CB_VRULE      BEGACT    SELECT        COLUMNS
 *
C                   Eval      %Setatr('Main':'SFL1':'VRule')=
C                               %Getatr('Columns':'CB_VRule':'Checked')
 *
C                   ENDACT
 ********************************************************************
 *                                                                  *
 * サブルーチン: Fill                                               *
 *                                                                  *
 * 説明: データベースからサブファイルを埋めます                     *
 *                                                                  *
 ********************************************************************
C     Fill          Begsr
 *
C     'Main'        Setatr    99            'MouseShape'
C                   Z-Add     0             Count
C                   Eval      *IN01 = *OFF
C                   Clear                   SFL1
C     'SQL'         Setatr    SQL           'Text'
C     'ODBC'        Setatr    1             'FetchNext'
C     'ODBC'        Getatr    'IsData'      Temp              1 0
 *
 * データがある間に実行します
C                   DoW       Temp = 1
C                   Write     SFL1
 *
C                   Eval      *IN01 = NOT *IN01
 *
C                   If        *IN01
C                   Eval      %Setatr('Main':'SFL1':'RowBGMix')=Mix
C                   EndIf
 *
 * 進行状況バーを移動させます
C                   Add       1             Count
C     'Main'        Setatr    1             'PBStep'
 * Check if there is another row
C     'ODBC'        Setatr    1             'FetchNext'
C     'ODBC'        Getatr    'IsData'      Temp              1 0
C                   EndDo
 *
```

図 23. ODBC/JDBC 照会プログラム例のコード (8/11)

```
C     'Main'        Setatr    0             'PBSetPos'
C     'Count'       Setatr    Count         'Label'
C     'SFL1'        Setatr    1             'SelectItem'
C                   Eval      %Setatr('Main':'SQL':'Text')=
C                              %Getatr('Main':'ODBC':'SQLQuery')
C     'Main'        Setatr    1             'MouseShape'
 *
C                   EndSr
 ********************************************************************
 *                                                                  *
 * ウィンドウ/パーツ/イベント: Columns/ST06/Click                   *
 *                                                                  *
 * 説明: リストのカラーを変更します                                 *
 *                                                                  *
 ********************************************************************
 *
C     ST06          BEGACT    CLICK         COLUMNS
 *
C                   Eval      *IN01 = *OFF
C                   Eval      I=%Getatr('Main':'SFL1':'Count')
C     %Part         Getatr    'BackMix'     Mix              11
 *
C                   Do        I             J
C                   Eval      *IN01 = NOT *IN01
 *
C                   If        *IN01
C                   Eval      %Setatr('Main':'SFL1':'Index')=J
C                   Eval      %Setatr('Main':'SFL1':'RowBGMix')=Mix
C                   EndIf
 *
C                   EndDo
 *
C                   ENDACT
 ********************************************************************
 *                                                                  *
 * ウィンドウ/パーツ/イベント: Main/PB_Sql/Press                    *
 *                                                                  *
 * 説明: SQL ステートメントを処理します                             *
 *                                                                  *
 ********************************************************************
 *
C     PB_SQL        BEGACT    Press         MAIN
 *
C     'SQL'         Getatr    'Text'        SQL
C                   Eval      %Setatr('Main':'ODBC':'SQLQuery')=SQL
C                   Eval      %Setatr('Main':'ODBC':'ExecuteSQL')=1
C                   Exsr      Fill
 *
C                   ENDACT
```

図 23. ODBC/JDBC 照会プログラム例のコード (9/11)

```
 ***************************************************************************
 *                                                                         *
 * ウィンドウ/パーツ/イベント: Columns/CB01/Select                          *
 *                                                                         *
 * 説明: 列を隠す/表示します                                               *
 *                                                                         *
 ***************************************************************************
 *
C     CB01          BEGACT    SELECT        COLUMNS
 *
C                   MoveL     %Part         TempName          4
C                   Move      TempName      Num2              2 0
C                   Eval      %Setatr('Main':'SFL1':'ColNumber')=Num2
C     %Part         Getatr    'Checked'     State             1 0
 *
C                   If        State = 1
C                   Eval      State = 0
 *
C                   Else
C                   Eval      State=1
C                   EndIf
 *
C                   Eval      %Setatr('Main':'SFL1':'Hidden')=State
 *
C                   ENDACT
 ***************************************************************************
 *                                                                         *
 * ウィンドウ/パーツ/イベント: Columns/CB_Sort/Select                       *
 *                                                                         *
 * 説明: ソート順序を設定します                                            *
 *                                                                         *
 ***************************************************************************
 *
C     CB_SORT       BEGACT    SELECT        COLUMNS
 *
C                   If        %Getatr('Columns':'CB_Sort':'Checked')=1
C                   Eval      SEQ = 'ASC '
 *
C                   Else
C                   Eval      SEQ = 'DESC'
C                   EndIf
 *
C                   ENDACT
```

図 23. ODBC/JDBC 照会プログラム例のコード (10/11)

```
 ********************************************************************
 *                                                                  *
 * ウィンドウ/パーツ/イベント: Detail/Detail/Create                 *
 *                                                                  *
 * 説明: すべてのフィールドを読み取り専用に設定します               *
 *                                                                  *
 ********************************************************************
 *
C     DETAIL        BEGACT    CREATE        DETAIL
 *
C     'CAN0000037'  Setatr    1             'ReadOnly'
 *
C                   ENDACT
 ********************************************************************
 *                                                                  *
 * ウィンドウ/パーツ/イベント: Main/MI_Tips/MenuSelect              *
 *                                                                  *
 * 説明: ヒント・テキストの表示を切り替えます                       *
 *                                                                  *
 ********************************************************************
 *
C     MI_TIPS       BEGACT    MENUSELECT    MAIN
 *
C                   If        %Getatr('Main':'MI_Tips':'Checked')=0
C                   Eval       %Setatr('Main':'MI_Tips':'Checked')=1
 *
C                   Else
C                   Eval       %Setatr('Main':'MI_Tips':'Checked')=0
C                   EndIf
C                   Eval      %Setatr('Main':'Main':'ShowTips')=
C                              %Getatr('Main':'MI_Tips':'Checked')
 *
C                   ENDACT
```

図 23. ODBC/JDBC 照会プログラム例のコード (11/11)

外枠

パーツのグループが関連していることを指示するためには、その回りに外枠を使用してください。

外枠とは、長方形の、ラベルのないボックスです。ボックスにラベルが必要な場合には、代わりにグループ・ボックス・パーツを使用してください。

関連情報については、 88 ページの『グループ・ボックス』を参照してください

パーツ属性

Bottom	Handle*	Height	Left
ParentName	PartName	PartType	Refresh
Top	UserData	Visible	Width

注: 制約事項については、属性の説明を参照してください。

適用可能なイベント

Create	Destroy

特殊な高さおよび幅の設定

2 つの外枠属性を使用して線を作成することができます。縦線を作成するには、**Width** 属性を *1* にセットし、横線を作成するには、**Height** 属性を *1* にセットしてください。

ポップアップ・メニュー

ユーザー・インターフェース上の特定のパーツに関連する多くの選択項目を表示するためには、ポップアップ・メニュー・パーツを使用します。ユーザーは、ポップアップ・メニュー・パーツにメニュー項目パーツおよびサブメニュー・パーツを追加することができます。

このメニューは、ユーザーが該当するキーまたはマウス・ボタンを押した時に現れるので、“ポップアップ”と呼ばれています。

注: このパーツのプロパティー、イベントなどを操作できるのは、プロジェクト・ツリー・ビューのそのポップアップ・メニューからだけです。

関連情報については、以下を参照してください。
- 112 ページの『メニュー・バー』
- 113 ページの『メニュー項目』
- 183 ページの『サブメニュー』

パーツ属性

Handle*	InvName	InvPName	ParentName
PartName	PartType	UserData	Visible*
X	Y		

***注:** 制約事項については、属性の説明を参照してください。

適用可能なイベント

このパーツにはイベントはありません。

進行状況バー

ファイルのコピー、データベースのロード、などの処理の進行をグラフィカルに示すためには、進行状況バー・パーツを使用してください。

たとえば、100 個のファイルのコピーの進行状況を示すために **PBRange** 属性を 100 に設定し、 **PBStepSize** 属性を 10 に設定することができます。そうすると、ユーザーのコードは、コピー・ファイル処理をモニターし、10 個のファイルがコピーされるごとに、進行状況バーの標識を前方に移動することができます。

Java アプリケーションでは、進行状況バーの幅がその高さよりも小さい場合には、進行状況バーは縦方向となります。

パーツ属性

Bottom	Handle*	Height	Left
ParentName	PartName	PartType	PBRange
PBSetPos	PBStep	PBStepSize	Top
UserData	Visible	Width	

***注:** 制約事項については、属性の説明を参照してください。

適用可能なイベント

Create	Destroy

進行状況バーの例

次の例では、読み取り操作が行われるたびに、進行状況バー標識が更新されます。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq----
 *
C                   EVAL      %setatr('win01':'WIN1':'PBRange')=10
C                   EVAL      %setatr('win01':'WIN1':'PBStepSize')=1
C                   DO        10
C                   Read      Input
C                   EVAL      %setatr('win01':'WIN1':'PBStep')=1
C                   EndDo
 *
```

プッシュボタン

頻繁に使用するアクションへの便利なアクセスを提供するためには、プッシュボタンを使用してください。

各プッシュボタン・パーツは特定のアクションをコントロールします。ユーザーがプッシュボタンをクリックすると、ただちにアクションが開始されます。プッシュボタン上のテキスト・ラベルは、そのアクションを説明するものです。

86 ページの『グラフィック・プッシュボタン』と比較してください

パーツ属性

BackColor	BackMix	Border*	Bottom
Enabled	Focus	FontBold	FontItalic
FontName	FontSize	FontStrike*	FontUnder*
ForeColor	ForeMix	Handle*	Height
HelpEnable	HighLight	Label	Left
ParentName	PartName	PartType	Refresh
ShowTips	TipText	Top	UserData
Validate	Visible	Width	

***注:** 制約事項については、属性の説明を参照してください。

適用可能なイベント

Create	Destroy	GotFocus	LostFocus
MouseEnter	MouseExit	MouseMove	Popup
Press			

デフォルトのプッシュボタンの設定

プッシュボタン・パーツのプロパティー・ノートブックでは、そのプッシュボタンを設計中のウィンドウのデフォルトのプッシュボタンとしたいことを指定することができます。デフォルトのプッシュボタンは黒い太枠付きで表示され、改行キーを押した時に関連のアクションが実行されます。

注: ウィンドウごとに 1 つだけデフォルトのプッシュボタンを定義することができます。複数を定義した場合には、VisualAge RPG が 1 つを選択します。

簡略記号の設定

注: 簡略記号は Java アプリケーションではサポートされていません。

プッシュボタンごとに、**Label** 属性を使用してテキストを特定のプッシュボタンに関連付けてください。そのテキストはボタン上に現れます。

プッシュボタンの簡略記号キーを指定するには、**Label** 属性のテキストの文字の前に簡略記号 ID を入れます。Windows では、アンパーサンド (&) を使用してください。この指定された文字が下線付きでインターフェース上に表示されます (たとえば、**C**ancel)。この下線は、キーボード上の Alt キーと下線付きの文字を押すことによってこのプッシュボタンを選択できることをユーザーに示します。

コマンド・キーの割り当て

プッシュボタンにコマンド・キーを割り当てることができます。これを行うには、パーツのプロパティー・ノートブックをオープンして、使用可能リストからコマンド・キーの 1 つを選択してください。

ユーザーが実行時にコマンド・キーを押すと、マウス・ボタンまたはキーボードのキーを押した時と同じ効果があります。プログラムには **Press** イベントが通知されます。

イベントの通知

次の時点でプログラムに対して **Press** イベントのシグナルが出されます。

- ユーザーがプッシュボタンを選択する。
- デフォルトのプッシュボタンが定義されている場合にユーザーが改行キーを押す。
- ユーザーがプッシュボタンに割り当てられているコマンド・キーを押す。

ラジオ・ボタン

関連してはいるが、互いに排他的な選択項目のグループの中から 1 つだけユーザーが選択できるようにしたい場合には、ラジオ・ボタンを使用します。ユーザーが選択を行った時に、グループ内の前に選択した選択項目は選択解除されます。

ラジオ・ボタンは、そばにテキストのあるラベル付きの立体的な円形のボタンとして現れます。選択すると、円形のボタンはドットとして表示されます。

ユーザーが一度に複数の選択項目を選択できるようにしたい場合には、ラジオ・ボタンを使用しないでください。その場合には、64 ページの『チェック・ボックス』を参照してください。

パーツ属性

BackColor	BackMix	Bottom	Checked
Enabled	Focus	FontBold	FontItalic
FontName	FontSize	FontStrike*	FontUnder*
ForeColor	ForeMix	Handle*	Height
HighLight*	Label	Left	ParentName
PartName	PartType	Refresh	SelectIdx
ShowTips	TipText	Top	UserData
Visible	Width		

***注:** 制約事項については、属性の説明を参照してください。

適用可能なイベント

Create	Destroy	Enter	MouseEnter
MouseExit	MouseMove	Popup	Select

簡略記号の設定

ラジオ・ボタンの簡略記号キーを指定するには、**Label** 属性のテキストの文字の前に簡略記号 ID を入れます。Windows では、アンパーサンド (&) を使用してください。この指定された文字は、インターフェース上に下線付きで表示されます (たとえば、**Blue**)。この下線は、キーボードの下線のついたその文字を押してラジオ・ボタンを選択できるということをユーザーに指示します。

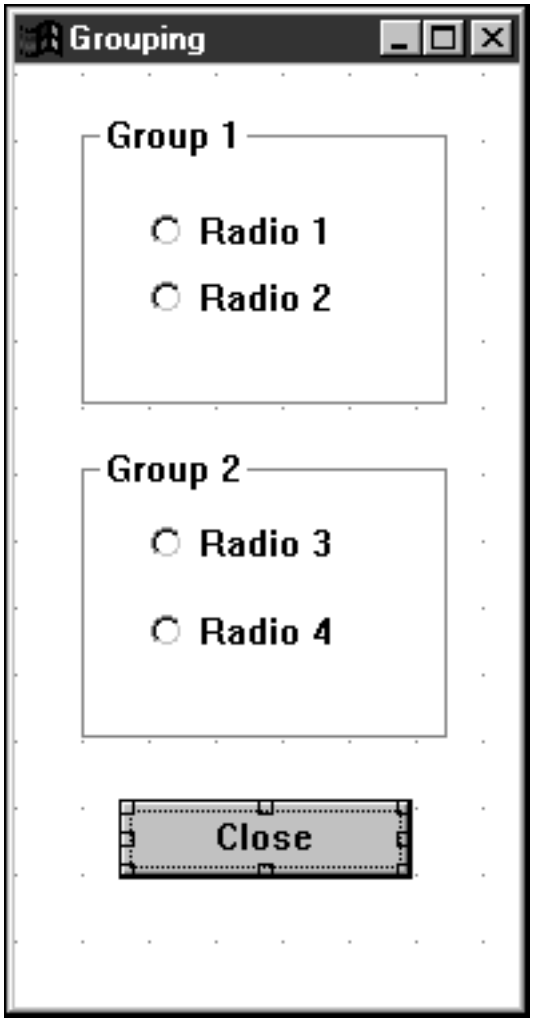
注: 簡略記号は Java アプリケーションではサポートされていません。

ラジオ・ボタンのグループ化

ラジオ・ボタンを作成する時には、それを論理的にグループ化して、ボタンの選択がその独自グループにあるボタンの状態だけに影響するようにしてください。

たとえば、設計ウィンドウに 4 つのラジオ・ボタンがあるとします。RB1 と RB2 は相互に排他的で、また RB3 と RB4 も相互に排他的です。これらのボタンは、2

つの論理グループにグループ化しなければなりません。次の図は、設計ウィンドウでのこれらのラジオ・ボタンのグループ化方法を示したものです。



ラジオ・ボタンを 2 つの論理グループに配置するには、次のようにしてください。

1. ラジオ・ボタンを希望通りに配置し、オプションで各グループの回りにグループ・ボックスをおきます。(88 ページの『グループ・ボックス』を参照してください。)
2. 設計ウィンドウでキャンバス・パーツを選択し、右マウス・ボタンを押します。
3. ポップアップ・メニューから、**タブおよびグループ...**を選択します。

   タブおよびグループのカスタマイズ・ウィンドウが現れ、そこに設計ウィンドウ上のすべてのパーツがリストされます。必要な場合には、このウィンドウをサイズ変更して、すべてのパーツを表示してください。
4. 左マウス・ボタンをクリックして、最初のグループの最初のボタンとなるラジオ・ボタンを選択します。この例では、RB1 がグループ 1 の最初のラジオ・ボタンです。
5. 右マウス・ボタンをクリックして、このラジオ・ボタン・パーツのポップアップ・メニューを表示し、**グループ・マーク**を選択します。

X マーク記号は、**グループ・マーク**列の下にあるラジオ・ボタンの横に表示されます。

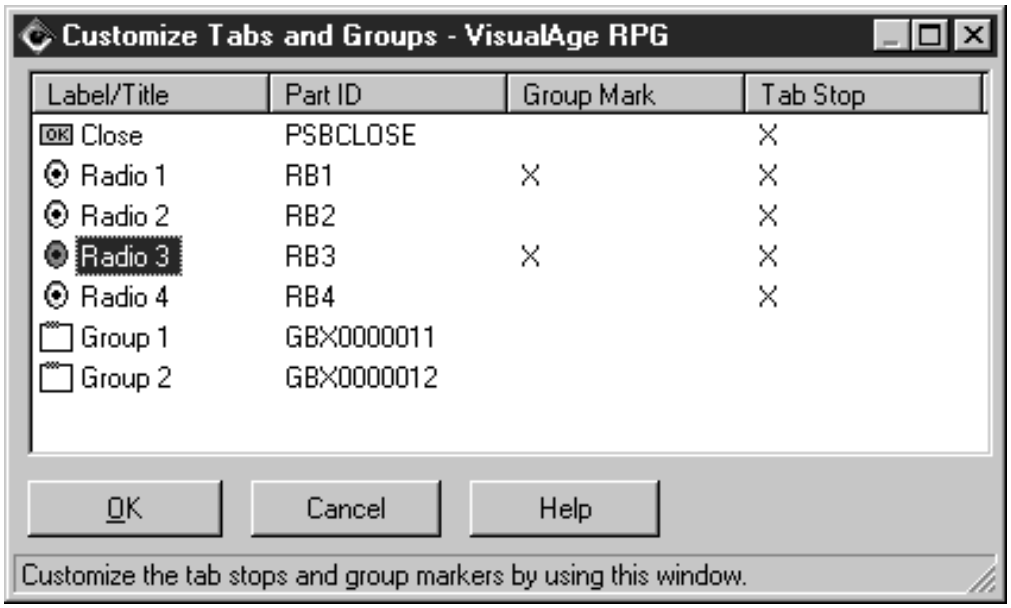
注: パーツのプロパティー・ノートブックでグループ・マークをセットすることもできます。

6. Ctrl + 上矢印および Ctrl +下矢印を使用して、RB2 を RB1 の後に位置指定します。この位置指定はツリー・ビュー内で行うことも、**上に移動**および**下に移動**メニュー項目を使用して行うこともできることに注意してください。RB2 に**グループ・マーク**属性が設定されていないことを確認してください。

   これは、RB2 がグループ 1 の 2 番目のラジオ・ボタンであることを指定します。
7. **OK** を押してこのウィンドウをクローズします。

   注: システム・メニューを使用してウィンドウをクローズ**しないでください。**さもないと、変更は保管されません。

ここで RB1 と RB2 は 1 つのグループの部分と見なされるので、一方を選択すると、もう一方に影響するだけです。同じ処理を RB3 と RB4 に繰り返します。次の図は、パーツがグループ化された後のタブおよびグループのカスタマイズ・ウィンドウを示したものです。



注: パーツのプロパティー・ノートブック内から個々のパーツにタブ・ストップおよびグループ・マークを設定することもできます。

ラジオ・ボタンの状態の設定

ラジオ・ボタンのプロパティー・ノートブックでは、始めにラジオ・ボタンが選択されるか否かを指示することができます。一度にグループ内の 1 つのラジオ・ボタンしか選択することはできません。複数を選択した場合には、グループ内の最後のものだけが選択されます。

デフォルトでは、ラジオ・ボタンを作成すると、**Checked** 属性が *0* にセットされます。このことは、ラジオ・ボタンが選択されず、その状態がオフであることを意味します。このラジオ・ボタンは、空の円で表示されます。

セットされ、状態がオンであるラジオ・ボタンを作成したい場合には、**Checked** 属性を *1* にセットしなければなりません。この場合には、ラジオ・ボタンは、部分的に塗りつぶされた円で表示されます。

プロパティー・ノートブックまたはプログラム中で **Checked** 属性をセットすることができます。

イベントの通知

ユーザーがラジオ・ボタンを選択すると、**Select** イベントのシグナルが出されます。

スライダー

スライダー軸に沿ったスライダー標識を動かすことによって、ユーザーが値を表示、設定、あるいは変更できるようにしたい場合には、スライダー・パーツを使用します。

スライダーは通常、秒数または度数などの度を超えた増分を持つ値に使用されるか、あるいは完了したタスクのパーセントを表示するために使用されます。

デフォルトによって、スライダーは左サイドにスライダー・シャフトを持って、ボックスの中央に水平方向に置かれます。スケールは、シャフトについての計測単位を表示するために表示されます。

以下を行うためには、スライダー・パーツ用のプロパティー・ノートブックを使用してください:
- スライダーが戻ることのできる値の範囲を設定
- ウィンドウ内でスライダーを縦方向または水平方向位置に設定
- スライダーが表す計測単位を示す目盛りを提供

パーツ属性

AddLink*	AllowLink*	BackColor	BackMix
Bottom	Enabled	Focus	FontBold
FontItalic	FontName	FontSize	FontStrike*
FontUnder*	ForeColor	ForeMix	Handle*
Height	Left	Maximum	Minimum
ParentName	PartName	PartType	Refresh
RemoveLink*	ShowTips	TickLabel	TickNumber
TipText	Top	UserData	Value
Visible	Width		

***注:** 制約事項については、属性の説明を参照してください。

適用可能なイベント

Change	Create	Destroy	GotFocus
Link*	LostFocus	MouseEnter	MouseExit
MouseMove	Popup		

***注:** 制約事項については、イベント記述を参照してください。

スライダー値の取り出しおよび設定

Value 属性の値を使用することによって、スライダーの値を取り出しまたは設定することができます。

Value 属性を取り出す場合には、戻り値を十分に収容できる大きさの結果フィールドを定義したことを確認してください。

イベントの通知

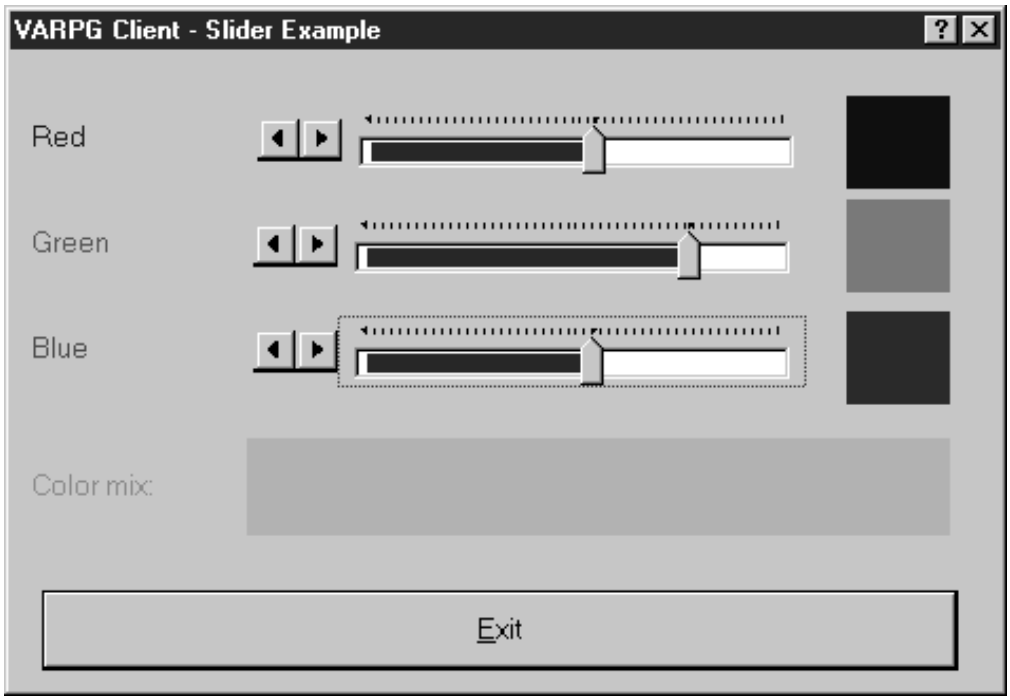
スライダー・アームの位置が変更された場合には、**Change** イベントのシグナルが送られます。

増分ボタンを使用してスライダー・アームを移動させた場合には、そのボタンを押している間は継続的に **Change** イベントのシグナルが送られます。

マウスを使用してスライダー・アームを移動させた場合には、マウス・ボタンを離した時に **Change** イベントが起こります。

スライダーの例

この例は、**BackMix** 属性を使用して他のパーツのカラーを制御するためにスライダー・パーツを使用することができる方法を示しています。各スライダーを動かすたびに、その値を使用して対応する静的テキスト・パーツの背景カラーの混合が判別され、そのカラーの輝度が表示されます。**サンプル**というラベルのついた静的テキストの背景カラーは、3 つのすべてのカラーが結合されたカラーの混合を表示するように更新されます。



```
*****************************************************************
*                                                               *
* プログラム ID : Slider                                        *
*                                                               *
* 説明 . . . . : スライダー・パーツを例示するためのサンプル・   *
*                 プログラム。                                  *
*                                                               *
*                 スライダー・アームを動かすたびに、そのスライダー *
*                 について CHANGE イベントのシグナルが送られます。 *
*                 CHANGE アクション・サブルーチンは、スライダーの *
*                 値を検索し、対応する静的テキスト・パーツの背景 *
*                 カラーの混合を更新して、そのカラーの輝度を表示 *
*                 します。                                      *
*                                                               *
*                 すべてのカラーの値を混合した結果を表示するため *
*                 に、静的テキスト・パーツ 'SAMPLE' の背景カラー *
*                 の混合も更新されます。                        *
*                                                               *
*****************************************************************
 *
H
 *
*****************************************************************
*                                                               *
* ウィンドウ . . : MAIN                                         *
*                                                               *
* パーツ . . : PB_EXIT                                          *
*                                                               *
* イベント . . : PRESS                                          *
*                                                               *
* 説明 . . . . : プログラムの終了。                             *
*                                                               *
*****************************************************************
 *
C     PB_EXIT       BEGACT    PRESS         MAIN
 *
C                   move      *on           *inlr
 *
C                   ENDACT
```

図 24. スライダー・パーツを使用するコーディング例 (1/4)

```
 ********************************************************************
 *                                                                  *
 * ウィンドウ . . : MAIN                                            *
 *                                                                  *
 * パーツ . . : GREEN                                               *
 *                                                                  *
 * イベント . : CHANGE                                              *
 *                                                                  *
 * 説明 . . . : 緑のカラー値の更新。                                *
 *                                                                  *
 ********************************************************************
 *
C     GREEN         BEGACT    CHANGE        MAIN
 *
C     'green'       getatr    'Value'       val               3 0
C                   move      val           grnmix            3
C                   move      *blanks       mix              11
C                   movel     '000:'        mix
C     mix           cat       grnmix:0      mix
C     mix           cat       ':000':0      mix
C     'STGreen'     setatr    mix           'BackMix'
C                   exsr      update
 *
C                   ENDACT
 ********************************************************************
 *                                                                  *
 * ウィンドウ . . : MAIN                                            *
 *                                                                  *
 * パーツ . . : BLUE                                                *
 *                                                                  *
 * イベント . : CHANGE                                              *
 *                                                                  *
 * 説明 . . . : 青のカラー値の更新。                                *
 *                                                                  *
 ********************************************************************
 *
C     BLUE          BEGACT    CHANGE        MAIN
 *
C     'blue'        getatr    'Value'       val
C                   move      val           blumix            3
C                   move      *blanks       mix
C                   movel     '000:000:'    mix
C     mix           cat       blumix:0      mix
C     'STBlue'      setatr    mix           'BackMix'
C                   exsr      update
 *
C                   ENDACT
```

図 24. スライダー・パーツを使用するコーディング例 (2/4)

```
*****************************************************************
*                                                               *
* サブルーチン : UPDATE                                         *
*                                                               *
* 説明 . . . . : 静的テキスト・パーツ 'Sample' の背景カラーの混合  *
*                 を更新して、3 つのスライダーからのカラーの値を  *
*                 結合した結果を表示します。                    *
*                                                               *
*****************************************************************
 *
C     UPDATE        BEGSR
 *
C                   move      *blanks       smpmix          11
C                   movel     redmix        smpmix
C     smpmix        cat       ':':0         smpmix
C     smpmix        cat       grnmix:0      smpmix
C     smpmix        cat       ':':0         smpmix
C     smpmix        cat       blumix:0      smpmix
C     'Sample'      setatr    smpmix        'BackMix'
 *
C                   ENDSR
*****************************************************************
*                                                               *
* ウィンドウ . . : MAIN                                         *
*                                                               *
* パーツ . . : RED                                              *
*                                                               *
* イベント . : CHANGE                                           *
*                                                               *
* 説明 . . . . : 赤のカラー値の更新。                           *
*                                                               *
*****************************************************************
 *
C     RED           BEGACT    CHANGE        FRA0000B
 *
C     'red'         getatr    'Value'       val
C                   move      val           redmix           3
C                   move      *blanks       mix
C                   movel     redmix        mix
C     mix           cat       ':000:000':0  mix
C     'STRed'       setatr    mix           'BackMix'
C                   exsr      update
 *
C                   ENDACT
```

図 24. スライダー・パーツを使用するコーディング例 (3/4)

```
*****************************************************************
*                                                               *
* ウィンドウ . . : MAIN                                         *
*                                                               *
* パーツ . . : MAIN                                             *
*                                                               *
* イベント . : CREATE                                           *
*                                                               *
* 説明 . . . : カラーの混合を初期化します                       *
*                                                               *
*****************************************************************
 *
C     MAIN          BEGACT    CREATE        MAIN
 *
C                   move      '000'         grnmix
C                   move      '000'         blumix
C                   move      '000'         redmix
 *
C                   ENDACT
```

図 24. スライダー・パーツを使用するコーディング例 (4/4)

スピン・ボタン

論理的な連続順序を持つ、関連しているが互いに排他的な選択項目のグループ (たとえば年月など) を順序通りに表示するために、スピン・ボタンを使用します。選択項目は、リング状に配置されているように表示されます。ユーザーは、上矢印キーを押して次に高い値に移ったり、下矢印キーを押して次に低い値に移ったりすることによって、選択項目全体で移動 (すなわち“スピン”) することができます。あるいは、選択項目の 1 つをスピン・ボタンの入力フィールドに直接入力することもできます。

パーツ属性

AddItemEnd	Alignment*	BackColor	BackMix
Bottom	Enabled	Focus	FontBold
FontItalic	FontName	FontSize	FontStrike*
FontUnder*	ForeColor	ForeMix	Handle*
Height	Left	Maximum	Minimum
ParentName	PartName	PartType	ReadOnly
Refresh	RemoveItem	ShowTips	Text
TipText	Top	UserData	Value
Visible	Width		

***注:** 制約事項については、属性の説明を参照してください。

適用可能なイベント

Change	Create	Destroy	GotFocus
Link*	LostFocus	MouseEnter	MouseExit
MouseMove	Popup	SpinDown	SpinEnd
SpinUp			

***注:** 制約事項については、イベント記述を参照してください。

スピン・ボタンの値の設定

スピン・ボタンのデータ・タイプによって、スピン・リストの値を設定するために使用される方式が決まります。

数字スピン・ボタンに使用可能な値を指定するためには、 **Maximum** および **Minimum** 属性を設定してください。

文字スピン・ボタンの初期スピン・リストの値を指定するためには、追加したい各項目ごとに **AddItemEnd** 属性を設定してください。文字スピン・ボタンの項目は自動的にソートされないので、それらの項目を表示したい順に追加してください。

スピン・ボタンの値の取り出し

スピン・ボタンで選択した値を検索するために使用する属性は、スピン・ボタンのタイプによって異なります。

- 文字スピン・ボタンの場合には、**Text** 属性を使用してください。
- 数字スピン・ボタンの場合には、**Value** 属性を使用してください。この属性は、スピン・ボタンに指定された最小値から最大値までの範囲の値を戻します。

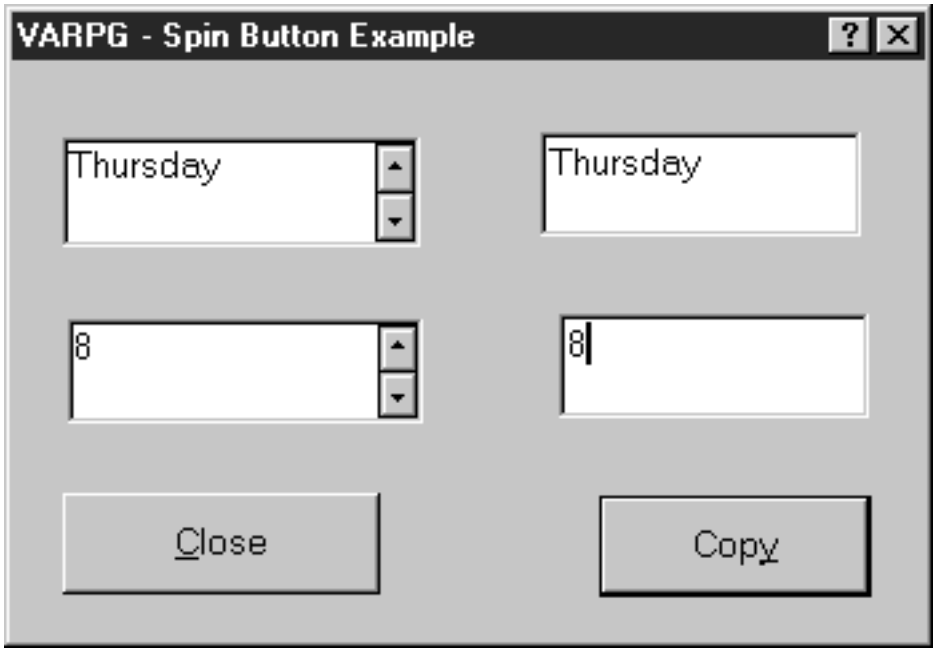
ユーザー入力の阻止

スピン・ボタンのプロパティー・ノートブックに **ReadOnly** 属性を設定するか、あるいはプログラムの中で **ReadOnly** 属性を *1* に設定することによって、ユーザーがスピン・ボタンと関連したフィールドに値を直接入力するのを防止することができます。

スピン・ボタンの例

この例は、数字および文字スピン・ボタンの値を設定して取り出す方法を示しています。プログラムの開始時には、各スピン・ボタンに初期リストが挿入されます。**「コピー」**プッシュボタンを選択した場合には、各スピン・ボタンの値が 関連した入力フィールド・パーツにコピーされます。

プログラムを終了するためには、**「クローズ」**プッシュボタンを押してください。



```
*********************************************************************
*                                                                   *
* プログラム ID : SPIN                                              *
*                                                                   *
* 説明. . . . . : スピン・ボタン・パーツを例示するサンプル・        *
*                  プログラム。                                     *
*                                                                   *
*                 文字および数字スピン・ボタンは、その初期化方法    *
*                 およびその値の検索方法を表示するために使用され    *
*                 ます。                                            *
*                                                                   *
*********************************************************************
 *
H
 *
DDAY             S             10A   DIM(7) PERRCD(1) CTDATA
 *
*********************************************************************
*                                                                   *
* ウィンドウ . . : MAIN                                             *
*                                                                   *
* パーツ . . . : PB_COPY                                            *
*                                                                   *
* イベント . . : PRESS                                              *
*                                                                   *
* 説明 . . . . : 各スピン・ボタンの値をその対応する入力フィールド・  *
*                 パーツにコピーします。                            *
*                                                                   *
*********************************************************************
 *
C     PB_COPY       BEGACT    PRESS         MAIN
 *
C     'SPB1'        Getatr    'Value'       tmp2N             2 0
C     'EF1'         Setatr    tmp2N         'Text'
 *
C     'SPB2'        Getatr    'Text'        tmp              10
C     'EF2'         Setatr    tmp           'Text'
 *
C                   ENDACT
```

図 25. スピン・ボタン・パーツを使用するコーディング例 (1/2)

```
     ****************************************************************
     *                                                              *
     * ウィンドウ . . : MAIN                                        *
     *                                                              *
     * パーツ . . : MAIN                                            *
     *                                                              *
     * イベント . : CREATE                                          *
     *                                                              *
     * 説明 . . . : 画面上のウィンドウを中央にそろえ、スピン・ボタン     *
     *              を初期化します。                                *
     *                                                              *
     ****************************************************************
     *
    C     MAIN          BEGACT    CREATE        MAIN
     *
     * 文字スピン・ボタンを配列 DAY からの週の曜日によって
     * 初期化します。
    C                   Do        7             I               2 0
    C     'SPB2'        Setatr    day(i)        'AddItemEnd'
    C                   EndDo
     *
     * 数字スピン・ボタンを初期化します。
    C     'SPB1'        Setatr    1             'Minimum'
    C     'SPB1'        Setatr    10            'Maximum'
     *
    C                   ENDACT
     ****************************************************************
     *                                                              *
     * ウィンドウ . . : MAIN                                        *
     *                                                              *
     * パーツ . . : PB_EXIT                                         *
     *                                                              *
     * イベント . . : PRESS                                         *
     *                                                              *
     * 説明 . . . : プログラムの終了。                              *
     *                                                              *
     ****************************************************************
     *
    C     PB_EXIT       BEGACT    PRESS         MAIN
     *
    C                   Move      *On           *INLR
     *
    C                   ENDACT
**CTDATA DAY
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
```

図 25. スピン・ボタン・パーツを使用するコーディング例 (2/2)

静的テキスト

入力フィールド・パーツのプロンプトなどのその他のパーツに対しては、ラベルとして静的テキスト・パーツを使用します。静的テキスト・パーツは、エンド・ユーザー入力を受け入れません。 Java アプリケーションでは、静的テキストを表示できるのは単一の行上だけです。

パーツ属性

Alignment	BackColor	BackMix	Bottom
DataType	DragEnable*	DropEnable*	DropValue*
Enabled	FontBold	FontItalic	FontName
FontSize	FontStrike*	FontUnder*	ForeColor
ForeMix	Handle*	Height	Label
Left	ParentName	PartName	PartType
Refresh	ShowTips	TipText	Top
UserData	Visible	Width	

***注:** 制約事項については、属性の説明を参照してください。

適用可能なイベント

Click	Create	DblClick	Destroy
Drop	Link*	MouseDown	MouseEnter
MouseExit	MouseMove	MouseUp	Popup

***注:** 制約事項については、イベント記述を参照してください。

静的テキスト・パーツのテキストの変更

静的テキスト・パーツは、長方形の区域で、そこにテキストが入れられます。静的テキスト・パーツのテキストを変更するためには、**Label** 属性を使用してください。

テキストが元のテキストより長くなるように変更した場合には、新規テキストはそれを囲む長方形の枠に合わせて切り取られます。**FontName** および **FontSize** 属性をより大きなフォントまたはサイズに変更した場合も、テキストが切り取られます。

プログラム内のテキストを変更する場合には、GUI Designer の静的テキスト・パーツが新規テキストを十分に表示できる大きさであることを確認してください。

静的テキストの値の取り出し

静的テキスト・パーツの値を取り出すためには、**Label** 属性を指定しなければなりません。数字静的テキスト・パーツの値を取り出している場合には、その値を受け取るフィールドも数字フィールドとして定義されていなければなりません。

ウィンドウの情報の取得および設定

コンパイル時に、コンパイラーは、プログラム内のフィールドを静的テキスト・パーツと同じ名前、同じデータ・タイプ、および長さで暗黙的に定義します。演算項目 2 にウィンドウ名を指定した READ および WRITE 命令コードを使用することによって、 **Label** 属性の値がこれらのフィールドの間で自動的にコピーされます。ユーザー・インターフェースの中に多くの静的テキスト・パーツがある場合には、一連の取り出しおよび設定属性を実行する必要がなくなるので、READ および WRITE 命令コードはとくに有用です。

詳しくは、 29 ページの『第 3 章 パーツを用いたプログラミング』を参照してください。

出力の編集

データ・タイプが数字である場合には、静的テキスト・パーツの内容を編集することができます。編集の説明については、 251 ページの『第 11 章 出力の編集』 を参照してください。

状況バー

ウィンドウのプロセスまたはアクションについての追加情報を指定するためには、状況バー・パーツを使用してください。状況バーに対して最大 5 つのペインを作成することができます。状況バー・パーツには、ウィンドウ・パーツの **StatusBar** 属性よりも融通性があります。

デフォルトでは、状況バーはウィンドウの最下部に作成されます。ただし、プロパティー・ノートブックを使用して、その位置を最上部に変更することができます。また、枠のスタイル、ペインの数、およびテキストの配置を設定することもできます。

パーツ属性

Handle*	ParentName	PartName	PartType
SBIndex	SBLabel	SBPanes	UserData
Visible			

***注:** 制約事項については、属性の説明を参照してください。

適用可能なイベント

Create	Destroy

状況バーの例

次の例では、何らかの初期処理が行われる間に状況バー・ラベルが更新されます。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq----
 *
C     STBAR         BEGACT    CREATE        MAIN
C     'STBAR'       SETATR    'Wait...'     'SBLABEL'
 *
 * 何か処理を実行します。
 *
C                   DO
C                   ...
C                   ENDDO
 *
 * 状況バー・ラベルを消去します。
 *
C     'STBAR'       SETATR    *BLANKS       'SBLABEL'
C                   ENDACT
 *
```

サブファイル

それぞれが 1 つまたは複数のフィールドからなるレコードのリストを表示するには、サブファイル・パーツを使用してください。

サブファイル・パーツは iSeries™ サブファイルと類似した機能を持っています ユーザーは、サブファイルのスクロール・バーを使用して、リストを水平方向または垂直方向にスクロールできます。

サブファイル入力フィールドを作成するには、「参照フィールドの定義」ウィンドウまたはパーツ・パレットからフィールドを指示してクリックして、サブファイル・パーツの上にそれをクリックしてください。 プロパティー・ノートブックを使用してフィールドを追加することもできます。

注: サブファイル・パーツはキャンバス付きのノートブックまたはキャンバス付きウィンドウ上にだけ指示してクリックすることができます。

パーツ属性

AddItemEnd	AllowEdit	AutoSelect	BackColor
BackMix	Bottom	ButtonIdx	Buttons
ButtonTip	CapsLock	CellBGClr	CellBGMix
CellFGClr	CellFGMix	ColBGClr	ColBGMix
ColFGClr	ColFGMix	ColNumber	ColWidth
Count	DColFRVCol	DeSelect	EditColumn
EditIndex	EditText	EnableBtn	Enabled
ExtSelect*	FirstSel	Focus	FontArea
FontBold	FontItalic	FontName	FontSize
FontStrike*	FontUnder*	ForeColor	ForeMix
Handle*	HdgBGClr	HdgBGMix	HdgFGClr
HdgFGMix	HdgIdx	HdgText	Height
Hidden	HRule	Index	ItemCount
Left	MapViewCol	MultSelect	NbrOfSel
OpenEdit	PageSize	ParentName	PartName
PartType	RemoveItem	RowBGClr	RowBGMix
RowFGClr	RowFGMix	Scale	Selected
SelectItem	SelectList	SetTop	SflNxtChg
ShowTips	SizeToFit	SortAsc	SortDesc
StartAt	TipText	Top	TopRecord
UserData	VColFRDCol	ViewColumn	Visible
VRule	Width		

*** 注:** 制約事項については、属性の説明を参照してください。

適用可能なイベント

Change	ColSelect	Create	Destroy
Enter	FirstRec	GotFocus	KeyPress

LastRec	LostFocus	MouseEnter	MouseExit
MouseMove	NextRec	PageDown	PageEnd
PageTop	PageUp	Popup	PrevRec
Select	VKeyPress		

サブファイル・パーツの作成

サブファイル・パーツは、キャンバス・パーツ上でのみ作成することができます。

サブファイル当りのフィールドの最大数

サブファイルには最大 99 フィールドを定義することができます。

サブファイル・パーツを処理するための命令コード

属性を使用してサブファイルを制御する他に、いくつかの命令コードを使用してサブファイル・パーツに影響を及ぼすことができます。演算項目 2 にサブファイルの名前を指定してください。名前を引用符で囲ま**ないでください。**

次の命令コードがサポートされています。それぞれの詳しい説明については、*VisualAge for RPG WINDOWS 版 言語解説書*, SC88-5608-04 または言語依存ヘルプを参照してください。

コード 演算

CHAIN
索引を指定することによって、サブファイルからレコードを読み取ります。

CLEAR
サブファイルからすべてのレコードを消去します。

DELETE
サブファイルからレコードを削除します。削除済みレコードに続くすべてのレコードは 1 つ上に移動します。

READC
レコードの入力フィールドの値が変更された場合にそのレコードを読み取ります。

READS
サブファイルから選択されたレコードを削除します。ユーザーは、マウスまたはキーボードでレコードを選択することができます。レコードが読み取られると、選択解除されます。

UPDATE
既存のサブファイル・レコードを更新します。この命令コードを使用するには、その前にレコードを読み取る必要があります。

WRITE
サブファイルに新規レコードを追加します。

サブファイルのロード

サブファイル・パーツ中の情報を表示するために、その情報が一度に 1 レコードずつサブファイル・パーツに書き込まれます。GUI Designer でサブファイル・パーツ

用に定義されているサブファイル・フィールドが所要の値にセットされ、サブファイル・レコード様式に対して WRITE 演算命令が実行されます。

サブファイル・サイズの決定

iSeries 400 サブファイルと異なり、サブファイル・パーツにはサブファイルまたはサブファイル・ページのサイズがありません。サブファイルに保持できるレコードの数は、ワークステーションのメモリーの容量によって制限されます。サブファイル・ページ・サイズ (すなわち一度に表示できるレコード数) は、GUI Designer でサブファイルを作成する時に決定されます。

レコード・カウントの取り出し

サブファイル中に現在あるレコードの数を判別するには、 **Count** 属性を使用します。

レコードの読み取りおよび更新

サブファイル・パーツ中のレコードは、更新または削除することができます。レコードを更新するには、まず更新したいレコードにサブファイルを位置付けてください。サブファイルの位置付けは、CHAIN、READC、または READS 演算命令によって行うことができます。これらの演算命令では、検索されたレコードのフィールド値がサブファイル・レコード様式の対応するプログラム・フィールドに割り当てられます。ここで、プログラムはこのフィールド値を変更することができます。

次にサブファイル・パーツで実行される UPDATE 演算命令が関連するフィールドの現在の値をサブファイルに戻します。サブファイル内の相対的位置付けでレコードを選択するには CHAIN、ユーザーがサブファイル表示で変更したレコードを選択するには READC、ユーザーが選択したレコードを選択するには READS をそれぞれ使用してください。

次の例は、READS 演算命令がループを実行してサブファイル中の選択されたすべてのレコードを入手し、それらを処理して、一度に 1 レコードずつ更新することを示しています。これは、**レポート**と呼ばれるプッシュボタンの **Press** イベント用のアクション・サブルーチン中にコーディングします。

```
:
     C     REPORT        BEGACT    PRESS         WIN1
      *
     C                   READS     SUBF1                                  99
      *
     C     *IN99         DOWEQ     *OFF
      *
      *
      *         選択されたレコードの場合、それを処理して、サブファイル表示
      *         中で 'Reported' としてマークします。
      *
     C                   MOVEL     '(Reported)'  SF1NAME
      *
     C                   UPDATE    SUBF1
      *
     C                   READS     SUBF1                                  99
     C                   END
      *
     C                   ENDACT
```

図 26. レコードの読み取りおよび変更のコーディング例

サブファイル・フィールドの変更

注: サブファイル・フィールドが読み取り専用としてセットされている場合には、ユーザーはそれを変更することができません。

ユーザーがサブファイル中のフィールドを変更できるようになるには、アプリケーションのユーザーまたはプログラムがそのフィールドを編集用にオープンしなければなりません。

1. ユーザーはマウス・ポインターでフィールドを選択してから、Alt キーを押さえたままで左マウス・ボタンをクリックします。ここでユーザーは、タブ・キーと後退タブ・キーを使用して、同じレコードの別のフィールドに移動し、また上下の矢印キーを使用して異なるレコードに移動することができます。
2. プログラム中でフィールドを編集用にオープンするには、次のようにしてください。
   a. **Index** 属性を使用して、編集されるフィールドが含まれるレコードを示します。
   b. **ColNumber** 属性をセットして、編集されるフィールドの列番号を示します。
   c. **OpenEdit** 属性値を *1* にセットしてフィールドを編集用にオープンします。(この属性値を *0* にセットして、現在編集用にオープンされているどのフィールドもクローズすることができます。)

ユーザーがサブファイル中のフィールドを変更したかどうかを判別するには、READC 演算命令を使用します。

隠しフィールド

サブファイル・パーツのプロパティー・ノートブックでは、サブファイル・フィールドを隠す に設定して、フィールドが表示されないようにすることができます。た

とえば、サブファイル・レコードは、隠しフィールドにレコード・キー情報を含むことができます。隠しフィールドは見ることができませんが、サブファイル・レコードとともにプログラムに戻されます。

サブファイル・フィールドの形式設定

サブファイル中のフィールドは、いくつかの方法で強調表示することができます。前景カラーおよび背景カラーは、サブファイル見出しと個々のサブファイル・フィールドの両方にセットすることができます。サブファイル内に横線または縦線を入れることができます。

詳細については、 251 ページの『第 11 章 出力の編集』を参照してください。

タブを使用可能にする

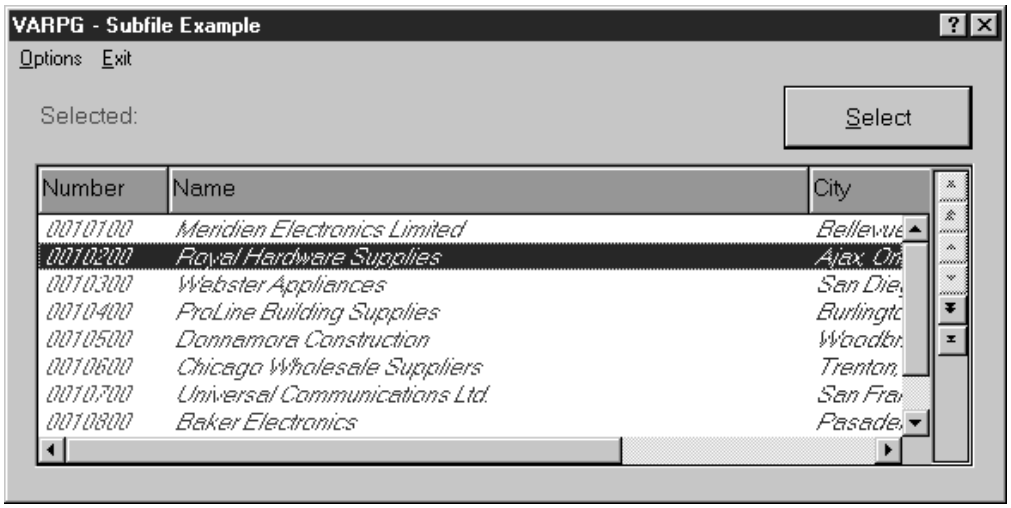
「タブおよびグループのカスタマイズ」ダイアログは、サブファイル・パーツのタブを使用可能にするために使用します。サブファイル・パーツが入っているウィンドウのキャンバス・パーツを右クリックします。ポップアップ・メニューから**「タブおよびグループ」**を選択します。「タブおよびグループのカスタマイズ」ダイアログが表示されます。タブ・ストップの設定値を設定または消去するには、パーツ名を右クリックして、**「タブ・ストップ」**メニュー項目を選択します。

サブファイルの例

次の例では、iSeries 400 サーバー上のデータベース・ファイルからレコードを表示するためにサブファイル・パーツが使用されています。データベースからのレコードでサブファイルをすべて埋める代わりに、サブファイル中のレコードのスクロールを制御するために、ナビゲーション・プッシュボタン
(FirstRec、LastRec、PageTop、PageUp、PageDown、PrevPage、NextPage) が提供されています。

「選択」プッシュボタンを選択すると、READS 命令コードが使用されて選択されたレコードが判別され、静的テキスト・パーツに CUSTNO フィールドの値が表示されます。また、レコードの最初のフィールドが編集用にオープンされます。

終了 メニュー項目を選択してプログラムを終了します。




```
     ****************************************************************
     *                                                              *
     * プログラム ID  : SUBFILE                                     *
     *                                                              *
     * 説明 . . . . . : サブファイル・パーツを説明するサンプル・    *
     *                   プログラム                                 *
     *                                                              *
     *                   このサンプル・プログラムには、CUSTOMER と呼ばれる*
     *                   物理データベース・ファイルが AS/400 で必要です。 *
     *                                                              *
     ****************************************************************
     *
    H
    FCUSTOMER  IF   E             DISK    REMOTE INFDS(INFDS) BLOCK(*Yes)
     *
     * データベース・ファイル用の INFDS。ファイルがオープンされると、FileSize に
     * ファイル中のレコード数が入ります。
    DINFDS            DS
    DFileSize               156    159B 0
     *
     ****************************************************************
     *                                                              *
     * サブルーチン . : *INZSR                                      *
     *                                                              *
     * 説明 . . . . . : 作業変数を初期化します。                    *
     *                                                              *
     ****************************************************************
     *
    C     *INZSR        BEGSR
     *
    C                   Z-Add     10            PageSize          2 0
    C                   Z-Add     1             CurRec            6 0
    C     FileSize      Sub       PageSize      LastPage          6 0
    C                   Add       1             LastPage
     *
    C                   ENDSR
     *                                                              *
```

図 27. サブファイル・パーツを使用するコーディング例 (1/10)

```
*****************************************************************
*                                                               *
* サブルーチン . : NEXTPAGE                                     *
*                                                               *
* 説明 . . . . . : データベースからレコードの次ページを取り出します *
*                                                               *
*****************************************************************
 *
C     NEXTPAGE      BEGSR
 *
C                   add       PageSize      CurRec
 *
C     CurRec        IfGt      FileSize
C                   Sub       PageSize      CurRec
 *
C                   Else
C                   Exsr      FillPage
C     'SFl1'        setatr    2             'BUTTONIDX'
C     'SFL1'        setatr    1             'ENABLEBTN'
C                   EndIf
 *
C                   ENDSR
```

図 27. サブファイル・パーツを使用するコーディング例 (2/10)

```
 ****************************************************************
 *                                                              *
 * サブルーチン . : PREVPAGE                                    *
 *                                                              *
 * 説明 . . . . . : データベースからのレコードの直前のページに  *
 *                  戻します。                                  *
 *                                                              *
 ****************************************************************
 *
C     PREVPAGE      BEGSR
 *
C                   Sub       PageSize      CurRec
 *
C     CurRec        IfLe      *zero
C                   Add       PageSize      CurRec
 *
C                   Else
C                   Exsr      FillPage
C     'SFL1'        setatr    5             'BUTTONIDX'
C     'SFL1'        setatr    1             'ENABLEBTN'
C                   EndIf
 *
C                   ENDSR
 ****************************************************************
 *                                                              *
 * サブルーチン . : FILLPAGE                                    *
 *                                                              *
 * 説明 . . . . . : データベースからのレコードのページでサブファイル *
 *                  埋めます。                                  *
 *                                                              *
 ****************************************************************
 *
C     FILLPAGE      BEGSR
 *
C                   Clear                   Sfl1
C     CurRec        Setll     customer
C                   Z-Add     1             count             2 0
C                   Read      customer                              9999
 *
C     *in99         DoWeq     *off
C     count         AndLE     PageSize
C                   Write     Sfl1
 *
C                   If        %Getatr('Main':'HILITE':'Checked')=1
C     'SFL1'        Setatr    Count         'Index'
C     'SFL1'        Setatr    1             'ColNumber'
C     'SFL1'        Setatr    *DarkGreen    'CellFGClr'
C     'SFL1'        Setatr    2             'ColNumber'
C     'SFL1'        Setatr    *DarkPink     'CellFGClr'
C     'SFL1'        Setatr    3             'ColNumber'
C     'SFL1'        Setatr    *DarkBlue     'CellFGClr'
C                   EndIf
```

図 27. サブファイル・パーツを使用するコーディング例 (3/10)

```
 *
C                   Add       1             count
C                   Read      customer                              9999
C                   EndDo
 *
C                   Read      customer                              9999
 *
C     CurRec        ifeq      1
C     'SFl1'        setatr    1             'BUTTONIDX'
C     'SFL1'        setatr    0             'ENABLEBTN'
C     'SFl1'        setatr    2             'BUTTONIDX'
C     'SFL1'        setatr    0             'ENABLEBTN'
C     'SFl1'        setatr    5             'BUTTONIDX'
C     'SFL1'        setatr    1             'ENABLEBTN'
C     'SFl1'        setatr    6             'BUTTONIDX'
C     'SFL1'        setatr    1             'ENABLEBTN'
C                   endif
 *
C     *in99         ifeq      *on
C     CurRec        oreq      LastPage
C     'SFl1'        setatr    1             'BUTTONIDX'
C     'SFL1'        setatr    1             'ENABLEBTN'
C     'SFl1'        setatr    2             'BUTTONIDX'
C     'SFL1'        setatr    1             'ENABLEBTN'
C     'SFl1'        setatr    5             'BUTTONIDX'
C     'SFL1'        setatr    0             'ENABLEBTN'
C     'SFl1'        setatr    6             'BUTTONIDX'
C     'SFL1'        setatr    0             'ENABLEBTN'
C                   endif
C                   ENDSR
 *
 ****************************************************************************
 *                                                                          *
 * ウィンドウ . . : MAIN                                                    *
 *                                                                          *
 * パーツ . . : MAIN                                                        *
 *                                                                          *
 * イベント . : CREATE                                                      *
 *                                                                          *
 * 説明 . . . : レコードの最初のページを取り出します。                      *
 *                                                                          *
 ****************************************************************************
 *
C     MAIN          BEGACT    CREATE        MAIN
 *
C                   Exsr      FillPage
 *
C     'SFL1'        Setatr    *Green        'HdgBGClr'
C     'SFL1'        Setatr    *Black        'HdgFGClr'
C     'SFL1'        Setatr    1             'ColNumber'
```

図 27. サブファイル・パーツを使用するコーディング例 (4/10)

```
 *
C     'MAIN'        Setatr    1             'Visible'
C     'SFL1'        Setatr    1             'BUTTONIDX'
C     'SFL1'        Setatr    '*MSG0001'    'BUTTONTIP'
C     'SFL1'        Setatr    0             'ENABLEBTN'
C     'SFL1'        Setatr    2             'BUTTONIDX'
C     'SFL1'        Setatr    '*MSG0002'    'BUTTONTIP'
C     'SFL1'        Setatr    0             'ENABLEBTN'
C     'SFL1'        Setatr    3             'BUTTONIDX'
C     'SFL1'        Setatr    0             'ENABLEBTN'
C     'SFL1'        Setatr    4             'BUTTONIDX'
C     'SFL1'        Setatr    0             'ENABLEBTN'
C     'SFL1'        Setatr    5             'BUTTONIDX'
C     'SFL1'        Setatr    '*MSG0004'    'BUTTONTIP'
C     'SFL1'        Setatr    6             'BUTTONIDX'
C     'SFL1'        Setatr    '*MSG0005'    'BUTTONTIP'
 *
C                   ENDACT
 *                                                                     *
 **********************************************************************
 *                                                                     *
 * ウィンドウ . . : MAIN                                               *
 *                                                                     *
 * パーツ . . : PB_SELECT                                              *
 *                                                                     *
 * イベント . . : PRESS                                                *
 *                                                                     *
 * 説明 . . . . : 選択されたサブファイル・レコードを読み取ります。     *
 *                 静的テキスト・パーツ 'Selected' が更新されて、      *
 *                 選択されたカスタマー番号を表示するようにします。    *
 *                 サブファイル中の最初のフィールドが編集用にオープン  *
 *                 されます。                                          *
 *                                                                     *
 **********************************************************************
 *
C     PB_SELECT     BEGACT    PRESS         MAIN
 *
C                   Reads     sfl1                                    27
 *
C     *in27         IfEq      *off
C     'Selected'    Setatr    custno        'Label'
C     'SFL1'        Setatr    1             'ColNumber'
C     'SFL1'        Setatr    1             'OpenEdit'
C                   EndIf
 *
C                   ENDACT
```

図 27. サブファイル・パーツを使用するコーディング例 (5/10)

```
*********************************************************************
*                                                                   *
* ウィンドウ . . : MAIN                                             *
*                                                                   *
* パーツ . . : HRULE                                                *
*                                                                   *
* イベント . :  MENUSELECT                                          *
*                                                                   *
* 説明:                                                             *
*                                                                   *
*********************************************************************
 *
C     HRULE         BEGACT    MENUSELECT    MAIN
 *
C                   If        %Getatr('Main':'HRULE':'Checked')=1
C     'SFL1'        Setatr    0             'HRule'
C     'HRULE'       Setatr    0             'Checked'
 *
C                   Else
C     'SFL1'        Setatr    1             'HRule'
C     'HRULE'       Setatr    1             'Checked'
C                   EndIf
 *
C                   ENDACT
*********************************************************************
*                                                                   *
* ウィンドウ . . : MAIN                                             *
*                                                                   *
* パーツ . . : VRULE                                                *
*                                                                   *
* イベント . :  MENUSELECT                                          *
*                                                                   *
* 説明:                                                             *
*                                                                   *
*********************************************************************
 *
C     VRULE         BEGACT    MENUSELECT    MAIN
 *
C                   If        %Getatr('Main':'VRULE':'Checked')=1
C     'SFL1'        Setatr    0             'VRule'
C     'VRULE'       Setatr    0             'Checked'
 *
C                   Else
C     'SFL1'        Setatr    1             'VRule'
C     'VRULE'       Setatr    1             'Checked'
C                   EndIf
 *
C                   Exsr      FillPage
 *
C                   ENDACT
```

図 27. サブファイル・パーツを使用するコーディング例 (6/10)

```
*****************************************************************
*                                                               *
* ウィンドウ . . : MAIN                                         *
*                                                               *
* パーツ . . : HILITE                                           *
*                                                               *
* イベント . :  MENUSELECT                                      *
*                                                               *
* 説明:                                                         *
*                                                               *
*****************************************************************
*
C     HILITE        BEGACT    MENUSELECT    MAIN
*
C                   If        %Getatr('Main':'HILITE':'Checked')=1
C                   Eval      %Setatr('Main':'HILITE':'Checked')=0
*
C                   Else
C                   Eval      %Setatr('Main':'HILITE':'Checked')=1
C                   EndIf
*
C                   Exsr      FillPage
*
C                   ENDACT
*****************************************************************
*                                                               *
* ウィンドウ . . : MAIN                                         *
*                                                               *
* パーツ . . :  SFL1                                            *
*                                                               *
* イベント . :  PAGETOP                                         *
*                                                               *
* 説明:                                                         *
*                                                               *
*****************************************************************
*
C     SFL1          BEGACT    PAGETOP       MAIN
*
C                   Z-Add     1             CurRec
C                   Exsr      FillPage
*
C                   ENDACT
```

図 27. サブファイル・パーツを使用するコーディング例 (7/10)

```
 ****************************************************************************
 *                                                                          *
 * ウィンドウ . . : MAIN                                                    *
 *                                                                          *
 * パーツ . . :  SFL1                                                       *
 *                                                                          *
 * イベント . :  PAGEUP                                                     *
 *                                                                          *
 * 説明:                                                                    *
 *                                                                          *
 ****************************************************************************
 *
C     SFL1          BEGACT    PAGEUP        MAIN
 *
C                   exsr      PrevPage
 *
C                   ENDACT
 *
 ****************************************************************************
 *                                                                          *
 * ウィンドウ . . : MAIN                                                    *
 *                                                                          *
 * パーツ . . :  SFL1                                                       *
 *                                                                          *
 * イベント . :  LASTREC                                                    *
 *                                                                          *
 * 説明:                                                                    *
 *                                                                          *
 ****************************************************************************
 *
C     SFL1          BEGACT    LASTREC       MAIN
 *
C     FileSize      Sub       PageSize      CurRec
C                   Add       1             CurRec
 *
C     CurRec        IfLt      1
C                   Z-Add     1             CurRec
C                   EndIf
 *
C                   Exsr      FillPage
 *
C     'SFL1'        setatr    1             'BUTTONIDX'
C     'SFL1'        setatr    1             'ENABLEBTN'
C     'SFL1'        setatr    2             'BUTTONIDX'
C     'SFL1'        setatr    1             'ENABLEBTN'
C     'SFL1'        setatr    5             'BUTTONIDX'
C     'SFL1'        setatr    0             'ENABLEBTN'
C     'SFL1'        setatr    6             'BUTTONIDX'
C     'SFL1'        setatr    0             'ENABLEBTN'
C                   ENDACT
```

図 27. サブファイル・パーツを使用するコーディング例 (8/10)

```
 ****************************************************************************
 *                                                                          *
 * ウィンドウ . . : MAIN                                                    *
 *                                                                          *
 * パーツ . . :  SFL1                                                       *
 *                                                                          *
 * イベント . :  PAGEDOWN                                                   *
 *                                                                          *
 * 説明:                                                                    *
 *                                                                          *
 ****************************************************************************
 *
C     SFL1          BEGACT    PAGEDOWN      MAIN
 *
C                   Exsr      NextPage
 *
C                   ENDACT
 ****************************************************************************
 *                                                                          *
 * ウィンドウ . . : MAIN                                                    *
 *                                                                          *
 * パーツ . . :  SFL1                                                       *
 *                                                                          *
 * イベント . :  FIRSTREC                                                   *
 *                                                                          *
 * 説明:                                                                    *
 *                                                                          *
 ****************************************************************************
 *
C     SFL1          BEGACT    FIRSTREC      MAIN
 *
C                   Z-Add     1             CurRec
C                   Exsr      FillPage
 *
C     'SFL1'        setatr    1             'BUTTONIDX'
C     'SFL1'        setatr    0             'ENABLEBTN'
C     'SFL1'        setatr    2             'BUTTONIDX'
C     'SFL1'        setatr    0             'ENABLEBTN'
C     'SFL1'        setatr    5             'BUTTONIDX'
C     'SFL1'        setatr    1             'ENABLEBTN'
C     'SFL1'        setatr    6             'BUTTONIDX'
C     'SFL1'        setatr    1             'ENABLEBTN'
C                   ENDACT
```

図 27. サブファイル・パーツを使用するコーディング例 (9/10)

```
*********************************************************************
*                                                                   *
* ウィンドウ . . : MAIN                                             *
*                                                                   *
* パーツ . . :  EXIT                                                *
*                                                                   *
* イベント . :  MENUSELECT                                          *
*                                                                   *
* 説明:                                                             *
*                                                                   *
*********************************************************************
 *
C     EXIT          BEGACT    MENUSELECT    MAIN
 *
C                   Move      *on           *inlr
 *
C                   ENDACT
```

図 27. サブファイル・パーツを使用するコーディング例 (10/10)

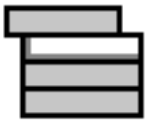
イベントの通知

Select イベントは次の場合に通知されます。
- ユーザーがサブファイルにある項目を選択する。
- ユーザーがプログラム中でリスト中の項目を選択する。
- ユーザーがすでに選択されている項目を選択する。

Enter イベントは次の場合に通知されます。
- ユーザーがサブファイル中の項目をダブルクリックする。
- サブファイルがフォーカスされ、項目が選択されている時にユーザーが改行キーを押す。

これらのイベントに対するアクション・サブルーチン中で、READS 命令コードを使用して、選択されている項目を判別することができます。

サブメニュー



サブメニューは次のために使用します:
- 既存のメニューにあるメニュー項目から新しいカスケード・メニューを開始する。
- メニュー・バー上のメニュー項目からプルダウン・メニューを開始する。

サブメニューを作成した後に、ツリー・ビューでのみ、メニュー項目パーツをサブメニュー・パーツ上で指示してクリック (またはドラッグ・アンド・ドロップ) することによってサブメニューにメニュー項目を追加することができます。

注: このパーツのプロパティー、イベントなどを操作できるのは、プロジェクト・ツリー・ビューのそのポップアップ・メニューからだけです。

関連情報については、 113 ページの『メニュー項目』 を参照してください。

パーツ属性

ParentName PartName PartType UserData

適用可能なイベント

Create Destroy

タイマー

ユーザー・プログラムが事前設定時間間隔で操作を必ず実行しなければならない場合には、タイマー・パーツを使用します。たとえば、ウィンドウをクローズするか、または非活動化されている一定期間の後にアプリケーションを終了するためなどに、タイマー・パーツを使用することができます。

タイマー・パーツは時間の単位をカウントし、2 つのイベントの間の事前設定時間間隔を計測して、インターバルが経過すると 2 番目のイベントに切り替えます。

GUI ビルダーにタイマー・パーツを作成すると、そのパーツがアイコンとして設計ウィンドウ上に表示されます。しかし、タイマー・パーツのためのプロパティー・ノートブックでは、プログラムの実行中はアイコンを表示したくない旨を指定することができます。

注: 精密なタイミングが必要な場合には、タイマー・パーツは使用しないようにしてください。システムでは他のプログラムが実行されているために、**Tick** イベントは必ずしも指定された正確な間隔で起こらないことがあります。

パーツ属性

AddLink*	AllowLink*	Bottom	Interval
Left	Multiplier	ParentName	PartName
PartType	RemoveLink*	TimerMode	TimerTicks
Top	UserData	Visible	

*** 注:** 制約事項については、属性の説明を参照してください。

適用可能なイベント

Create	Destroy	Link*	Tick

*** 注:** 制約事項については、イベント記述を参照してください。

タイマー・アイコンの表示

デフォルトにより、プログラムの実行中はタイマー・アイコンが表示されるように、 **Visible** 属性は *1* に設定されます。このアイコンを表示したく**ない**場合には、この属性を *0* に設定してください。

間隔の設定

タイマー間隔はミリ秒で表されます。この間隔が経過すると、タイマーの **Tick** イベントのシグナルが送られます。この間隔は、タイマー・パーツのプロパティー・ノートブックの中で設定することができます。これは、また、 **Interval** 属性を使用してプログラム内で設定することができます。

注: 最小タイマー間隔は 100 ミリ秒です。

タイマー・パーツには、**Multiplier** 属性があります。この属性を設定することによって、タイマーの **Tick** イベントが生成される前に間隔値が何度経過するかを決定することができます。デフォルトの乗数値は、タイマーが各間隔の終わりで **Tick** イベントを生成するように *1* に設定されます。

Tick イベントの生成

タイマーが開始された場合には、その間隔値がゼロにリセットされます。間隔値に達した場合には、タイマーが **Tick** イベントを生成し、間隔値を更新します。

タイマー値の取り出し

タイマーが **Tick** イベントを生成するたびに、その値が 1 ずつ増やされます。タイマーの現在の値を取り出すためには、 **Value** 属性を使用してください。タイマーの値は、プロパティー・ノートブックまたはユーザー・プログラムの中で設定することができます。

タイマー・モードを使用したタイマーの制御

タイマーを制御するためには、**TimerMode** 属性を使用してください。

タイマーを開始するためには、**TimerMode** を *1* に設定してください。タイマーを開始することによって **Tick** イベントの生成が始まり、間隔値に達した時にその **Value** 属性が増やされます。

タイマーを停止するためには、**TimerMode** を *2* に設定してください。タイマーが停止すると、 **Tick** イベントが生成され、その値は更新されません。

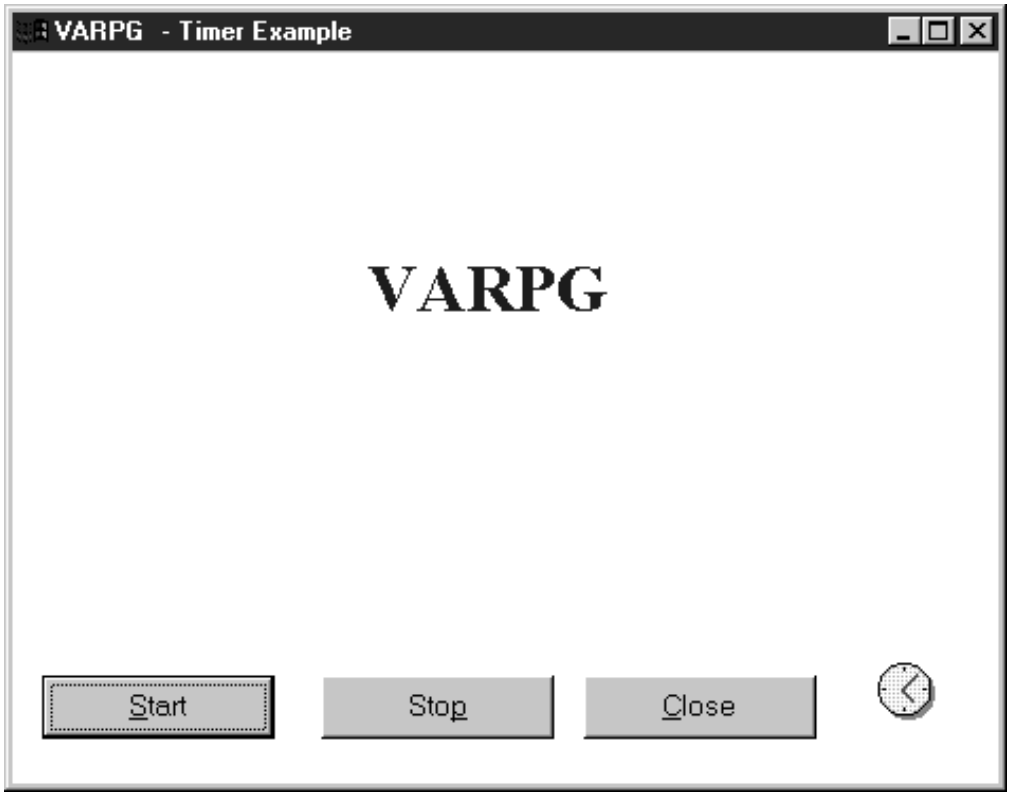
タイマーの例

この例では、タイマーの **Tick** イベントごとに静的テキスト・パーツがウィンドウの中で移動します。

「開始」プッシュボタンを押した場合には、タイマーのモードが *1* に設定されます。これによってタイマーが開始され、**Tick** イベントが生成されます。 **Tick** イベントの処理時には、静的テキスト・パーツのための新規の座標が計算され、そのパーツが新規の位置に設定されます。

「停止」プッシュボタンを押した場合には、 **TimerMode** が *2* に設定されます。これによってタイマーは停止します。

プログラムを終了するためには、**「クローズ」**プッシュボタンを押してください。




```
 ****************************************************************************
 *                                                                          *
 * プログラム ID : TIMER                                                    *
 *                                                                          *
 * 説明. . .  . : タイマーの 'Tick' のたびに静的テキスト・パーツ            *
 *                 をウィンドウの中で移動させることによって                 *
 *                 タイマー・パーツを例示するサンプル・プログラム。         *
 *                                                                          *
 ****************************************************************************
 *
H
 *
 * 画面サイズのシステム属性の宣言
D%DspHeight       S              4  0
D%DspWidth        S              4  0
 *
 * 新規のサイズのイベント属性の宣言
D%NewHeight       S              4  0
D%NewWidth        S              4  0
 *
 * 作業変数の定義
DminX             S              4  0 INZ(0)
DmaxX             S              4  0
DminY             S              4  0
DmaxY             S              4  0
DxChange          S              4  0 INZ(5)
DyChange          S              4  0 INZ(5)
 *
```

図 28. タイマー・パーツを使用するコーディング例 (1/6)

```
*********************************************************************
*                                                                   *
* ウィンドウ : FRA0000B                                             *
*                                                                   *
* パーツ . . : FRA0000B                                             *
*                                                                   *
* イベント . : CREATE                                               *
*                                                                   *
* 説明       : ウィンドウを画面の中央にそろえます。                 *
*                                                                   *
*              開始値の計算。                                       *
*              ウィンドウ・パーツの高さ (Height) 属性にはタイトル・ *
*              バーが含まれているので、静的テキスト・パーツが       *
*              ウィンドウ・フレーム内に留まるように、タイトル・     *
*              バーの高さが減算されます。                           *
*                                                                   *
*              SVGA の場合には、この値はおよそ 20 ピクセルです。    *
*              他の解像度の場合には、これを調整することができます。 *
*                                                                   *
*********************************************************************
 *
C     FRA0000B      BEGACT    CREATE        FRA0000B
 *
 * 開始ウィンドウの高さ (Height) と幅 (Width) の取り出し
C     'FRA0000B'    getatr    'Height'      winHeight         4 0
C     'FRA0000B'    getatr    'Width'       winWidth          4 0
 *
 * 画面上のウィンドウの中央そろえ
C                   eval      %setatr('FRA0000B':
C                                     'FRA0000B':
C                                     'Left')=(%DspWidth-winWidth)/2
 *
C                   eval      %setatr('FRA0000B':
C                                     'FRA0000B':
C                                     'Bottom')=(%DspHeight-winHeight)/2
 *
 * 静的テキスト・パーツの開始座標の取り出し
C     'ST1'         getatr    'Left'        picX              4 0
C     'ST1'         getatr    'Bottom'      picY              4 0
 *
 * 静的テキスト・パーツのディメンションの取り出し
C     'ST1'         getatr    'Height'      picHeight         4 0
C     'ST1'         getatr    'Width'       picWidth          4 0
 *      * 最小および最大 Y 座標の計算
C     'Start'       getatr    'Height'      startH            4 0
C     'Start'       getatr    'Bottom'      startB            4 0
C                   eval      minY = startB + startH
C                   eval      maxY = winHeight - picHeight - 20
 *
```

図 28. タイマー・パーツを使用するコーディング例 (2/6)

```
      * 最大 X 座標の取り出し
C                   eval      maxX = winWidth - picWidth
 *
C                   ENDACT
 ****************************************************************************
 *                                                                          *
 * ウィンドウ : FRA0000B                                                    *
 *                                                                          *
 * パーツ . . : START                                                       *
 *                                                                          *
 * イベント . . : PRESS                                                     *
 *                                                                          *
 * 説明 . . . : タイマーの開始。                                            *
 *                                                                          *
 ****************************************************************************
 *
C     START         BEGACT    PRESS         FRA0000B
 *
C     'Timer1'      setatr    1             'TimerMode'
 *
C                   ENDACT
 ****************************************************************************
 *                                                                          *
 * ウィンドウ : FRA0000B                                                    *
 *                                                                          *
 * パーツ . . : STOP                                                        *
 *                                                                          *
 * イベント . . : PRESS                                                     *
 *                                                                          *
 * 説明 . . . : タイマーの停止。                                            *
 *                                                                          *
 ****************************************************************************
 *
C     STOP          BEGACT    PRESS         FRA0000B
 *
C     'Timer1'      setatr    2             'TimerMode'
 *
C                   ENDACT
```

図 28. タイマー・パーツを使用するコーディング例 (3/6)

```
*****************************************************************
*                                                               *
* ウィンドウ : FRA0000B                                         *
*                                                               *
* パーツ . . : CLOSE                                            *
*                                                               *
* イベント . . : PRESS                                          *
*                                                               *
* 説明 . . . . : プログラムの終了。                             *
*                                                               *
*****************************************************************
*
C     CLOSE         BEGACT    PRESS         FRA0000B
*
C                   eval      *inlr = *on
*
C                   ENDACT
*****************************************************************
*                                                               *
* ウィンドウ : FRA0000B                                         *
*                                                               *
* パーツ . . : TIMER1                                           *
*                                                               *
* イベント . : TICK                                             *
*                                                               *
* 説明 . . . . : 静的テキスト・パーツをウィンドウの中で移動させる     *
*              ことによってタイマーの Tick イベントに応答します。   *
*                                                               *
*              静的テキスト・パーツがウィンドウ・フレームの外側     *
*              に移動した場合には、方向を反転させるためにその      *
*              xChange または yChange の値に -1 が乗算されます。    *
*                                                               *
*****************************************************************
```

図 28. タイマー・パーツを使用するコーディング例 (4/6)

```
 *
C     TIMER1        BEGACT    TICK          FRA0000B
 *
 * 新規の静的テキスト座標の計算
C                   eval      picX = picX + xChange
C                   eval      picY = picY + yChange
 *
 * 静的テキストがウィンドウの境界内に留まっているかどうかのチェック
C                   select
 *
C     picX          whenlt    0
C                   eval      xChange = xChange * -1
C                   eval      picX = minX + xChange
 *
C     picX          whengt    maxX
C                   eval      xChange = xChange * -1
C                   eval      picX = maxX + xChange
 *
C     picY          whenlt    minY
C                   eval      yChange = yChange * -1
C                   eval      picY = minY + yChange
 *
C     picY          whengt    maxY
C                   eval      yChange = yChange * -1
C                   eval      picY = maxY + yChange
 *
C                   endsl
 *
 * 新規の座標への静的テキストの移動
C     'ST1'         setatr    picX          'Left'
C     'ST1'         setatr    picY          'Bottom'
 *
 *
C                   ENDACT
```

図 28. タイマー・パーツを使用するコーディング例 (5/6)

```
 ****************************************************************
 *                                                              *
 * ウィンドウ : FRA0000B                                        *
 *                                                              *
 * パーツ . . : FRA0000B                                        *
 *                                                              *
 * イベント . : RESIZE                                          *
 *                                                              *
 * 説明 . . . : 静的パーツがウィンドウ全体を使用できるように、  *
 *              ウィンドウが表示された後にそのサイズを取り出します。 *
 *                                                              *
 ****************************************************************
 *
C     FRA0000B      BEGACT    RESIZE        FRA0000B
 *
C                   eval      maxY = %NewHeight - picHeight - 20
C                   eval      maxX = %NewWidth - picWidth
 *
C                   ENDACT
```

図 28. タイマー・パーツを使用するコーディング例 (6/6)

縦方向のスクロール・バー

情報ペインを縦方向にスクロールできるようにするためには、縦方向のスクロール・バー・パーツを使用してください。この情報は、ファイルのリストであったり、データベース中のレコードであったり、また文書中の列などである場合があります。 **Range** 属性を使用してスクロールするオブジェクトの合計数を表し、**PageSize** 属性を使用して 1 ページに表示できるオブジェクトの数を判別することができます。

パーツ属性

Bottom	Enabled	Focus	Handle*
Height	Left	NextLine	NextPage
PageSize	ParentName	PartName	PartType
Position	PrevLine	PrevPage	Range
Top	UserData	Visible	Width

***注:** 制約事項については、属性の説明を参照してください。

適用可能なイベント

Create	Destroy	Scroll

ウィンドウ

ウィンドウは、プログラムと相互に作用するユーザーの基本的な手段です。 ユーザーのアプリケーションには、少なくとも 1 つのウィンドウが含まれていなければなりません。

たとえば、メニュー・バー、ポップアップ・メニュー、およびメッセージ・サブファイルのようなウィンドウ・フレームについて拡張するパーツを除いて、ウィンドウのクライアント域にはただ 1 つのパーツしか追加できません。追加できるパーツは自動的にクライアント域のサイズに合わせられます。

ウィンドウに複数のパーツを入れたい場合には、キャンバス・パーツを追加しなければなりません。あるいは、キャンバス・パーツ付きのウィンドウを使用して、ステップを節約します。

注: ウィンドウ・パーツは、パーツ・パレットでなく、パーツ・カタログの**フレーム**・セクションにあります。

関連情報については、以下を参照してください。
- 62 ページの『キャンバス』
- 193 ページの『キャンバス付きウィンドウ』

パーツ属性

Bottom	Center	Enabled	FileName*
Focus*	FontBold*	FontItalic*	FontName*
FontSize*	FontStrike*	FontUnder*	Handle*
Height	IconHandle*	Label	Left
MouseIcon*	MouseShape*	ParentName	PartName
PartType	PBRange	PBSetPos	PBStep
PBStepSize	Print	PrintAsIs	ProgresBar
Refresh	SBLabel	SBPosition	SBStyle
ShowTips	StatusBar	Top	UserData
Visible	Width	WindowMode*	

***注:** 制約事項については、属性の説明を参照してください。

適用可能なイベント

Activate	Close	Create	DeActivate
Destroy	LClickTray	Moved	RClickTray
ReSize	ShutDown		

キャンバス付きウィンドウ

ウィンドウは、エンド・ユーザーがプログラムと対話を交わすのに用いる最も重要な手段です。 キャンバス付きウィンドウ・パーツのキャンバスによって、ウィンドウに多くのパーツを追加することができます。

キャンバス部分の各種のパーツを指示してクリックし、それらを配置してから編成して、グラフィカル・ユーザー・インターフェースを作成することができます。メニュー・バー、ポップアップ・メニュー、およびメッセージ・サブファイルなどの、ウィンドウ・フレームの拡張となるパーツを追加することもできます。

ウィンドウのクライアント域に 1 つのパーツだけを入れる必要がある場合には、キャンバス・パーツ付きのウィンドウは必要ありません。代わりにウィンドウ・パーツ (パーツ・カタログの**フレーム**・セクションにある) を使用してください。キャンバスがなければ、追加したパーツは、クライアント域に合わせて自動的にサイズ変更されます。

関連情報については、以下を参照してください。
- 62 ページの『キャンバス』
- 192 ページの『ウィンドウ』

パーツ属性

Bottom	Center	Enabled	FileName*
Focus*	FontBold*	FontItalic*	FontName*
FontSize*	FontStrike*	FontUnder*	Handle*
Height	IconHandle*	Label	Left
MouseIcon*	MouseShape*	ParentName	PartName
PartType	PBRange	PBSetPos	PBStep
PBStepSize	Print	PrintAsIs	ProgresBar
Refresh	SBLabel	SBPosition	SBStyle
ShowTips	StatusBar	Top	UserData
Visible	Width	WindowMode*	

***注:** 制約事項については、属性の説明を参照してください。

適用可能なイベント

Activate	Close	Create	DeActivate
Destroy	LClickTray	Moved	RClickTray
ReSize	ShutDown		

ウィンドウの表示

デフォルトでは、すべてのウィンドウが、GUI Designer で作成された時に **Visible** および **Open Immediately** としてマークされています。

最初に表示したいウィンドウを決定します。このウィンドウはメイン・ウィンドウまたはプライマリー・ウィンドウと呼ばれ、それに応じて **Visible** および **Open Immediately** 属性を設定しなければなりません。デフォルトの設定を変更しないと、アプリケーションの始動時にすべてのウィンドウが表示されます。

Open Immediately 属性の設定

アプリケーションの始動時にウィンドウを作成したい場合には、設計時にこの属性を設定します。ウィンドウが作成されるとそれがメモリーにロードされます。これに関連したオーバーヘッドがあるので、アプリケーションの始動時にどのウィンドウをロードする必要があるかを決定してください。(後で他のウィンドウをロードすることができます。) SHOWWIN 命令コードを使用して、即時オープンするようにそれらを設定する代わりに、それほど頻繁に表示されないウィンドウ (製品の著作権などを表示するウィンドウ) を表示することができます。

注: **Open Immediately** 属性は、ウィンドウが画面上に実際に表示されたかどうかは制御しません。ウィンドウを表示するためには、ユーザー・プログラムでその **Visible** 属性を *1* に設定するか、あるいはそのプロパティー・ノートブックで **Visible** としてマークする必要があります。

SHOWWIN 命令コードの使用法

SHOWWIN 命令コードの演算項目 2 にウィンドウ名を指定して、ユーザー・プログラムにウィンドウをロードすることができます。この命令コードはメモリーにウィンドウをロードします。

注: SHOWWIN 命令は、ウィンドウが画面上に実際に表示されたかどうかは制御しません。ウィンドウを表示するためには、ユーザー・プログラムでその **Visible** 属性を *1* に設定するか、あるいはそのプロパティー・ノートブックで **Visible** にマークする必要があります。

ウィンドウの属性を設定できるのはそれがロードされてからです。ウィンドウをロードするためには、パーツのノートブックの始動ページで**即時オープン**・チェック・ボックスを選択するか、あるいはユーザー・プログラムで SHOWWIN 命令コードを使用します。

ウィンドウが**即時オープン**として定義されていてユーザー・プログラムでそのウィンドウに SHOWWIN 命令コードを出した場合には、ウィンドウがすでにロードされていることを示す実行時エラーが出されます。このエラーは、SHOWWIN 命令コードでエラー標識をコーディングし、ユーザー・プログラムでエラー標識をチェックすることによって避けることができます。この標識がオンの場合には、ウィンドウはすでに表示されているので、 **Visible** 属性をオンに設定してください。これでウィンドウが表示されて、エラーは出されません。

参照

ウィンドウのパーツはウィンドウの作成時に作成されます。したがって、まだロードされていないウィンドウのパーツを参照しようとしたり、ウィンドウが作成される前にウィンドウの属性を参照しようとすると、*パーツが見つかりません* のメッセージが出されます。

ヒント

ウィンドウが表示されてそのタイトル・バーをクリックできない場合には、以下の方法でウィンドウを移動してください。
1. マウス・カーソルをウィンドウを表示部分に位置付けます。
2. 左マウス・ボタンをクリックして放します。
3. Alt-スペース・キーの組み合わせを押します。次に M を押します。
4. 矢印キーを使用してウィンドウの位置を変更します。
5. ウィンドウが希望する位置にきたら Enter を押します。

ウィンドウのサイズ変更

アプリケーションを作成するために 1 つまたは複数の方法を実行でき、ウィンドウのサイズを変更するためには 1 つまたは複数の方法があります。

- GUI Designer では、ウィンドウの枠を **Sizeable** として設定します。この設定によって、ユーザーはマウス・ボタンでウィンドウの枠を選択し、マウス・ボタンを押したままで枠のサイズを変更することができます。マウス・ボタンを放すと、**ReSize** イベントが通知されます。
- ウィンドウに**最大化**および**最小化**ボタンを追加します。ユーザーはこれらのボタンの 1 つを選択してウィンドウのサイズを変更することができます。

ウィンドウのサイズが変更された後でパーツがウィンドウの枠内でその相対位置とサイズを保つように、ウィンドウのパーツを位置付けることができます。このためには、**ReSize** イベントを **%NewHeight** および **%NewWidth** イベント属性と一緒に使用してください。

次のコーディング例では、PB1 とラベル付けされたプッシュボタン・パーツがウィンドウの右上隅にあります。このウィンドウがサイズ変更されると、 **ReSize** アクション・サブルーチンが新しい **Left** および **Bottom** 属性値を計算して、プッシュボタンがウィンドウの枠内に留まるようにします。

```
*************************************************************************
*                                                                       *
* プログラム ID . . : ReSize                                            *
*                                                                       *
* 説明  . : ウィンドウのサイズが変更された場合に、パーツが確実に        *
*           ウィンドウ内に残されるようにする方法を例示するサンプル      *
*           プログラム。                                                *
*                                                                       *
*           プッシュボタンは、ウィンドウの右上隅にあります。            *
*           ウィンドウがより小さいサイズに変更された場合には、          *
*           すべてのパーツがそのウィンドウの左下隅との相対関係で        *
*           維持されるので、プッシュボタンは可視でなくなります。        *
*           RESIZE イベントは、プッシュボタンの位置もウィンドウの       *
*           右上隅との相対関係で確実に維持されるようにするために        *
*           使用されます。                                              *
*                                                                       *
*************************************************************************
 *
H
 *
 * Declare display size System attributes
D%DspHeight       S              4  0
D%DspWidth        S              4  0
 *
```

図 29. ウィンドウのサイズ変更後に、パーツが正しく表示されるようにします。 (1/3)

```
 * %NewHeight および %NewWidth イベント属性の宣言。これらには、
 * ウィンドウのサイズが変更された後のそのウィンドウの幅および
 * 高さが入れられます。
D%NewHeight       S              4  0
D%NewWidth        S              4  0
 ******************************************************************************
 *                                                                            *
 * ウィンドウ : FRA0000B                                                      *
 *                                                                            *
 * パーツ . . : FRA0000B                                                      *
 *                                                                            *
 * イベント . : RESIZE                                                        *
 *                                                                            *
 * 説明 . . . . . : プッシュボタン・パーツ 'PB1' が確実に可視のまま  *
 *                  残されるようにします。                                    *
 *                                                                            *
 ******************************************************************************
 *
C     FRA0000B      BEGACT    RESIZE        FRA0000B
 *
C     %NewWidth     sub       HOffset       NewLeft           4 0
C     %NewHeight    sub       VOffset       NewBottom         4 0
C     'PB1'         setatr    NewLeft       'Left'
C     'PB1'         setatr    NewBottom     'Bottom'
 *
C                   ENDACT
 ******************************************************************************
 *                                                                            *
 * ウィンドウ : FRA0000B                                                      *
 *                                                                            *
 * パーツ . . : PSB0000D                                                      *
 *                                                                            *
 * イベント . . : PRESS                                                       *
 *                                                                            *
 * 説明 . . . . : プログラムの終了。                                          *
 *                                                                            *
 ******************************************************************************
 *
C     PSB0000D      BEGACT    PRESS         FRA0000B
 *
C                   move      *on           *inlr
 *
C                   ENDACT
```

図 29. ウィンドウのサイズ変更後に、パーツが正しく表示されるようにします。 (2/3)

```
*********************************************************************
*                                                                   *
* ウィンドウ : FRA0000B                                             *
*                                                                   *
* パーツ . . : FRA0000B                                             *
*                                                                   *
* イベント . : CREATE                                               *
*                                                                   *
* 説明       : ウィンドウを画面の中央にそろえます。                 *
*              プッシュボタン PB1 の座標とそのオフセットを          *
*              ウィンドウの右上隅から取り出します。                 *
*                                                                   *
*********************************************************************
 *
C     FRA0000B      BEGACT    CREATE        FRA0000B
 *
C     'FRA0000B'    getatr    'Height'      winHeight         4 0
C     'FRA0000B'    getatr    'Width'       winWidth          4 0
C     %DspWidth     sub       winWidth      diffWidth         4 0
C     %DspHeight    sub       winHeight     diffHeight        4 0
 *
C                   eval      %setatr('FRA0000B':
C                                     'FRA0000B':
C                                     'Left') = diffWidth / 2
 *
C                   eval      %setatr('FRA0000B':
C                                     'FRA0000B':
C                                     'Bottom') = diffHeight / 2
 *
 * プッシュボタン・パーツ 'PB1' のウィンドウの右上隅からのオフセット
 * を計算します。これらの値は、ウィンドウのサイズが変更された場合
 * に、このオフセットを維持するために使用されます。
C     'PB1'         getatr    'Left'        PBLeft            4 0
C     'PB1'         getatr    'Bottom'      PBBottom          4 0
C     'FRA0000B'    getatr    'Width'       WinWidth          4 0
C     'FRA0000B'    getatr    'Height'      WinHeight         4 0
C     WinWidth      sub       PBLeft        HOffset           4 0
C     WinHeight     sub       PBBottom      VOffset           4 0
 *
C                   ENDACT
```

図 29. ウィンドウのサイズ変更後に、パーツが正しく表示されるようにします。 (3/3)

フォーカスの設定

ユーザーに最初に使用させたいウィンドウを決定し、**Focus** 属性を使用してそのウィンドウにフォーカスします。これを行わないと、VisualAge RPG が、アプリケーションのロード時にフォーカスされるウィンドウを決定します。デフォルトでは、これは、**Visible** 属性が設定された、最後に作成されたウィンドウです。

ウィンドウ・リスト

ウィンドウ・パーツのプロパティー・ノートブックで、ウィンドウがウィンドウ・リストに現れるかどうかを指示することができます。このリストは、Windows で Ctrl+Alt+Delete を押した時に現れます。デフォルトでは、ウィンドウ・パーツはウィンドウ・リストに現れません。 ウィンドウ・リストに表示するには、少なくともメイン・ウィンドウを設定しなければなりません。タスク・リストを使用してウィンドウを再表示することができます。

プログラムの終了

ユーザーがウィンドウのシステム・メニューから**「クローズ」** オプションを選択すると、オペレーティング・システムはウィンドウをクローズしますが必ずしもプログラムを終了するとは限りません。これを避けるために、次の 1 つを行うことができます。

- ウィンドウのプロパティー・ノートブックの 2 番目のスタイル・ページで、**クローズ時に終了**チェック・ボックスを選択します。これで、ユーザーがウィンドウをクローズするとプログラムが終了します。
- ウィンドウのプロパティー・ノートブックの最初のスタイル・ページで、システム・メニューなしにウィンドウが作成されるように**システム・メニュー**・チェック・ボックスを選択解除します。(デフォルトでは、すべてのウィンドウがシステム・メニューで作成されます。)
- **Close** イベントを使用します。このイベントは、ユーザーがシステム・メニューから**クローズ**を選択すると通知されます。**Close** イベント・アクション・サブルーチンでは、**LR** 標識をオンに設定するか、またはこのウィンドウをクローズしてもよいかどうかの確認のプロンプトを出して、それに応じて **ENDACT** 戻り点を設定します。たとえば、戻り値を **NODEFAULT* に設定すると、クローズ要求は無視されてウィンドウはクローズしません。

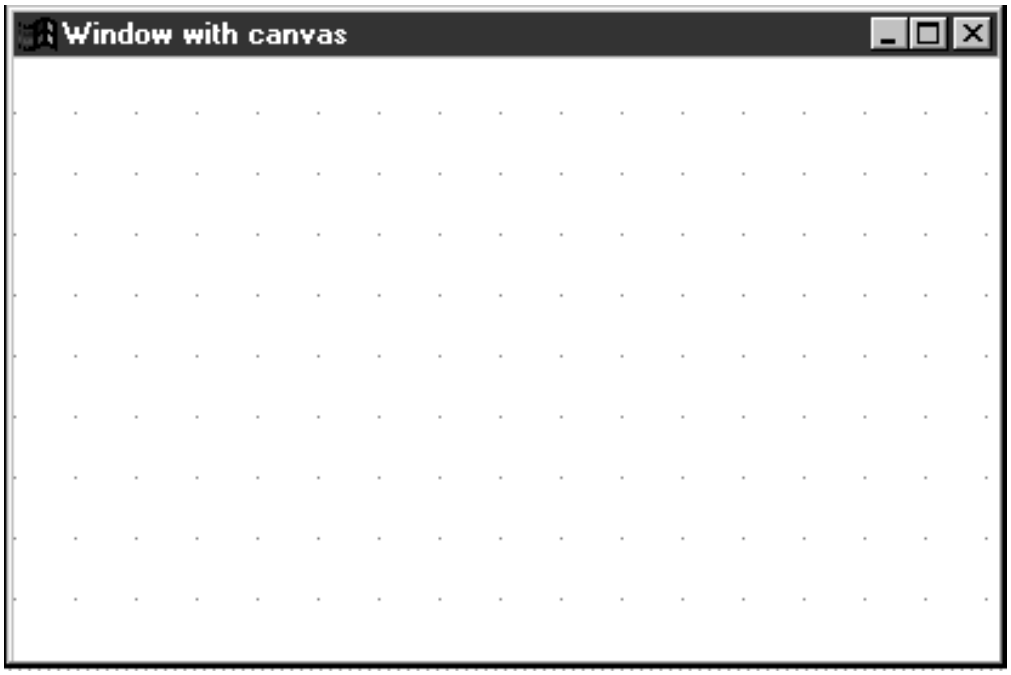
```
 *
 * メッセージ・ボックス変数の定義
Dstyle           M                  button(*yesbutton: *nobutton)
D                                   style(*WARN)
Dmsg             M                  msgtext('Are sure you want to exit?')
 *
 ****************************************************************************
 *                                                                          *
 * ウィンドウ : FRA0000B                                                    *
 *                                                                          *
 * パーツ . . : FRA0000B                                                    *
 *                                                                          *
 * イベント . . . : CLOSE                                                   *
 *                                                                          *
 * 説明 . . . . . : システム・メニューからの Close イベントを処理    *
 *                   して、ユーザーがこのウィンドウをクローズしたい   *
 *                   を確認します。                                         *
 *                                                                          *
 ****************************************************************************
 *
C     FRA0000B      BEGACT    CLOSE         FRA0000B
 *
 * クローズのプロンプト
C     msg           dsply     style         rc               9 0
 *
 * Yes であれば、プログラムを終了してクローズ
C     rc            ifeq      *YESBUTTON
C                   move      *on           *inlr
C                   movel     '*DEFAULT    'return          12
 *
 * そうでなければ、このウィンドウをクローズしない
C                   else
C                   movel     '*NODEFAULT  'return
C                   endif
 *
C                   ENDACT    return
```

ウィンドウ上のフィールドの消去

ウィンドウ上にいくつかの入力フィールドがある場合には、 CLEAR 命令コードを使用することができます。これは、すべての入力フィールドの値を消去してそれらのデフォルト値にします。数字フィールドはゼロで消去され、文字フィールドはブランクで消去されます。

ウィンドウ・パーツの例

下に示すウィンドウ・パーツは、**システム・**メニュー、**最小化**ボタン、および**最大化**ボタンを持っています。



*Component

*Component パーツによって、プログラマーはコンポーネントおよびシステム・ワイドの属性にアクセスし使用することができます。

*Component パーツはコンポーネントの「パーツ表示」です。それぞれのコンポーネントに 1 つの *Component パーツが自動的に作成されます。これは非表示でパレット上にはありません。

パーツ属性

Active*	Alarm	AppData	Button
ClipBoard	CurrentDir	Dialog	DIRName*
DlgOwner	DoEvents*	DspHeight	DspWidth
FileName	HelpWindow	HostName*	LookNFeel*
MsgData	MsgFile*	MsgID	MsgText
Name	OS	Parent	PartCount
PartList	Platform	PlugCmd*	PlugDLL*
PlugID*	PlugRC*	PlugResult*	Printer*
SelPrinter*	ShData	ShDataLen	ShDataName
ShDataPos	ShowMsgID	SwitchTo*	WrkStnName*

***注:** 制約事項については、属性の説明を参照してください。

適用可能なイベント

このパーツに関連したイベントはありません。

*Component パーツの使用

*Component パーツによって、プログラマーはコンポーネントおよびシステム・ワイドの属性にアクセスし使用することができます。*Component パーツは、コンポーネントの 'パーツ表現' です。それぞれのコンポーネントに 1 つの *Component パーツが自動的に作成されます。これは非表示でパレット上にはありません。

ファイル・オープン/別名保管ダイアログの表示。

*Component パーツの **Button**、**FileName**、**Dialog**、および **DlgOwner** 属性は、Windows に共通の「ファイルを開く」または「名前を付けて保存」ダイアログを表示するために使用されます。表示するダイアログのタイプは、ダイアログ属性が決定します。「ファイルを開く」ダイアログを表示するには 1 に、「名前を付けて保存」ダイアログを表示するには 2 に設定します。 **DlgOwner** 属性は、どのパーツがダイアログの '所有者' であるかを決定します。この属性が設定されると、所有者はダイアログにとって 'モーダル' になります。すなわち、ダイアログが隠されるまで、イベントに応答できません。**FileName** 属性を設定すると、「ファイルを開く」ダイアログが表示されます。ユーザーがダイアログを隠すために使用したボタンを判別するには、**Button** 属性の値を検索します。

次の例では、「ファイルを開く」ダイアログが表示されます。一定の拡張子を持つファイルのみを表示するように **FileName** 属性を設定できることに、注意してください。

```
 *
 * Display File Open dialog
C     '*Component'  Setatr    1             'Dialog'
 *
 * This window is the owner
C     '*Component'  Setatr    'Main Main'   'DlgOwner'
 *
 * Show only .DAT files
C     '*Component'  Setatr    '*.DAT'       'Filename'
 *
 * Get the button pressed
C     '*Component'  Getatr    'Button'      Button            1 0
 *
 * Handle the OK button
C                   If        Button = 1
 *
 * User canceled
C                   Else
 *
C                   EndIf
```

図 30. 「ファイルを開く」ダイアログの表示

プリンターの選択

アプリケーションがワークステーションに接続されているプリンターに印刷する場合には、**SelPrinter** および **Printer** 属性を使用して、ユーザーが出力の宛先プリンターを選択できるようにすることができます。**SelPrinter** 属性を 1 に選択すると、Windows 印刷ダイアログが表示されます。ユーザーがそのダイアログからプリンターを選択すると、アプリケーションからの印刷出力がそのプリンターに送られます。

プラグインの使用

PlugDLL、**PlugID**、**PlugCmd**、**PlugRC**、および **PlugResult** 属性により、GUI Designer の機能性を拡張することができます。開発したプログラムにさらに機能性を追加します。ベンダー・メニューを使用することによってアプリケーションが GUI ビルダーに登録されると、アプリケーションは GUI Designer と対話することができます。プラグイン作成の詳細については、第 20 章を参照してください。

コンポーネント内のパーツの照会

Parent、**PartCount**、および **PartList** 属性を実行時に使用して、コンポーネント内のパーツ名を照会することができます。たとえば、これらの属性を使用して、ウィンドウがサイズ変更された場合にウィンドウ内のパーツのサイズ変更と位置変更を行うことができます。

第 3 部 iSeries データの処理

第 8 章 iSeries の接続性

アプリケーションの開発時 (たとえば、表示装置ファイルのインポート時)、またはその実行時 (たとえば、入出力用の iSeries データベース・ファイルへのアクセス時) に iSeries サーバーを使用する場合には、アプリケーションで使用される iSeries 情報を定義しなければなりません。この情報はアプリケーションとは別に保管されるので、アプリケーション自体を変更しないで更新することができます。

この項では、次のトピックについて説明します。
- iSeries 情報の定義
- 設計時およびランタイムの iSeries サーバーのセットアップ
- データ域の使用
- iSeries データベース・ファイルの使用
- データベース入出力上の考慮事項
- 実行時のサーバー接続の制御
- アプレット用のセキュリティー・ファイルの使用

iSeries 情報の定義

アプリケーションの開発中に、**iSeries 情報の定義** プロパティー・ノートブックを使用して、次の iSeries 情報の別名 (一時変更名) を定義することができます。
- サーバー
- ファイル
- プログラム
- データ域
- ロック・レベル

アプリケーションを開発し、それをワークステーションにインストールする準備ができたら、次のことを確認する必要があります。
- SNA 通信の場合は、次のように構成されていること:

  クライアント・アクセスを使用してルーターを定義しなければなりません。このルーター名はリモート・ロケーション名としても使用されます。
- TCP/IP 通信の場合は、次のように構成されていること:

  iSeries サーバー用に定義されたホスト名をリモート・ロケーション名として使用します。

さらに、iSeries 情報を定義するためのステップについてはオンライン・ヘルプを参照してください。

ノートブックについての考慮事項

iSeries 情報の定義プロパティー・ノートブック・ページにプログラム、データ域、またはデータベース・ファイルの一時変更名が入っていない場合には、次のことが行なわれます。
1. プログラムの名前、データ域、またはプログラムのデータベース・ファイルが使用されます。

2. プログラム名、データ域、またはデータベース・ファイルがプログラムのライブラリー修飾である場合には、このライブラリーが使用されます。
3. プログラム名またはデータベース・ファイルがプログラムのライブラリー修飾でない場合には、 iSeries サーバーのライブラリー・リスト (*LIBL) が検索されます。
4. サーバー・ページに最初にリストされたサーバーが使用されます。

注: 「iSeries 情報の定義」ノートブックのサーバー・ページに最初にリストされたサーバーが、デフォルトのサーバーとして認識されます。iSeries サーバーを使用するすべてのプログラムに少なくとも 1 つのサーバーが必要です。

サーバーの設定

アプリケーションを開発する場合には、そのアプリケーションの編集、コンパイル、およびデバッグを行なう時にアクセスできるように、サーバーを設定しなければなりません。アプリケーションをパッケージして他のワークステーションに配布する場合には、実行中のアプリケーションが設計時に使用していたものと違うサーバーにアクセスする場合にもサーバーを設定する必要があります。

サーバーを設定する場合には、サービス・ジョブのライブラリー・リストに処理したいリモート・リソースが入っていることを確認してください。

設計時のサーバーの設定

アプリケーションの開発時にサーバーを使用する必要がある場合には、「サーバー・ログオンの定義」 ウィンドウと「iSeries 情報の定義」ノートブックを使用して、サーバー情報を定義しなければなりません。詳細については、オンライン・ヘルプを参照してください。

また、ライブラリー・リストを設定するために iSeries ジョブ記述も定義しなければなりません。 ライブラリー・リストは iSeries サーバーのジョブ記述と関連付けることができます。次に、このジョブ記述はユーザー・プロファイルと関連付けることができます。サーバーにログオンするように VisualAge RPG のプロンプトがあった場合には、このユーザー・プロファイルからのユーザー ID を使用してください。iSeries サービス・ジョブには正しいライブラリー・リストが入っています。

実行時のサーバーの設定

アプリケーションの実行時にサーバーにアクセスする必要がある場合には、 iSeries 情報が正しいサーバーを指していることを確認しなければなりません。「iSeries情報の定義」ノートブックを呼び出すには、iSeries 情報定義ユーティリティーを使用します。

また、ジョブ記述を変更するか、CL コマンド QCMDDDM または QCMDEXC を使用して、ライブラリー・リストも設定しなければなりません。

ライブラリー・リストを設定するためのジョブ記述の定義

ライブラリー・リストは iSeries サーバーのジョブ記述と関連付けることができます。次に、このジョブ記述はユーザー・プロファイルと関連付けることができます。サーバーにログオンするように VisualAge RPG のプロンプトがあった場合に

は、このユーザー・プロファイルからのユーザー ID を使用してください。iSeries サービス・ジョブには正しいライブラリー・リストが入っています。

ライブラリー・リストの変更

VisualAge RPG プログラムが CL コマンドを呼び出す場合:

- CL コマンドが iSeries ファイル用のコマンドを出す場合には、QCMDDDM に対して CALL を指定します。
- CL コマンドがサーバー・プログラムまたはデータ域にコマンドを出す場合には、QCMDEXC に対して CALL を指定します。

CL コマンドは、CALL 命令コードを使用して DDM サービス・ジョブ内で実行するように発行することができます。CL コマンドを DDM サービス・ジョブ内で実行するためには、特別のプログラムを呼び出さなければなりません。この特別のプログラムが QCMDDDM です。このインターフェースは、QCMDEXC を呼び出す場合のインターフェースと同じです。 QCMDEXC と QCMDDDM の相違点は、QCMDEXC がリモート呼び出し要求およびデータ域要求のサービスに使用される別のジョブで実行されることです。

QCMDDDM は、DDM サービス・ジョブのライブラリー・リストを変更して、データベース・ファイルが入っているライブラリーが DDM ジョブのライブラリー・リスト中にあるようにするために使用することができます。

データ域の使用

アプリケーションがデータ域を使用する前に、サーバーを設定する必要があります。

アプリケーションがデータ域にアクセスする場合には、このデータ域の名前はデータ域の名前か一時変更名のいずれかにすることができます。GUI Designer の一時変更名は、「iSeries 情報の定義」ノートブックのデータ域ページを使用して定義することができます。

このノートブック・ページにデータ域の一時変更名が入っていない場合には、 205 ページの『ノートブックについての考慮事項』 を参照してください。

表 5 および 208 ページの図 31 に、一時変更名を使用したデータ域のアクセス方法を示します。

表 5. この情報は、「iSeries 情報の定義」ノートブックのデータ域ページに入力します。

データ域一時変更名:	DTAARA (これは大文字で入力しなければなりません)
リモート・データ域名:	REMDTAARA
サーバーの別名:	SERVER01

データ域を使用する前に、データ域が初期化されていることを確認してください。VisualAge RPG プログラムのパック 10 進数サブフィールドを持つデータ域データ構造で検索しようとしている時に、サーバーのデータ域に有効なパック 10 進数値数が入っていないと、ランタイム例外が出されます。

```
*********************************************************************
*                                                                   *
* プログラム ID. : dtaaraex.vpg                                     *
*                                                                   *
* 説明 . . . . . : AS/400 データ域の目次を入手するための           *
*                  コード・セグメント。                             *
*********************************************************************
*
D dtaara          S              6P 0 DTAARA
*********************************************************************
*                                                                   *
* ウィンドウ : WIN1                                                 *
*                                                                   *
* パーツ . . : PSB0000C                                             *
*                                                                   *
* イベント . . : PRESS                                              *
*                                                                   *
* 説明: AS/400 データ域の内容を取得します。                         *
*********************************************************************
*
C     PSB0000C      BEGACT    PRESS         WIN1
C                   IN        dtaara
C                   ENDACT
```

図 31. データ域へのアクセス

iSeries 400 データベース・ファイルの使用

アプリケーションが iSeries 400 データベース・ファイルにアクセスする前に、サーバーを設定する必要があります。

VisualAge RPG プログラムで使用するリモート DISK ファイル名は、iSeries 400 ファイル名またはファイル別名とすることができます。ファイル別名は、「iSeries 情報の定義」ノートブックの「ファイル」ページを使用して定義することができます。このノートブック・ページに iSeries 400 ファイルのファイル別名が入っていない場合にどうなるかについては、205 ページの『ノートブックについての考慮事項』 を参照してください。

リモート・サーバー DDM ジョブで出されたデータベース・ファイル一時変更は、VisualAge RPG アプリケーションによって出されたオープン要求では無視されます。DDM サービス・ジョブで実行されるサーバー・プログラムが出したオープン要求では、ファイル一時変更の無視または適用のいずれかを選択することができます。

VisualAge RPG は「iSeries 情報の定義」ノートブックの「ファイル」ページを使用したサーバーのライブラリー名、ファイル名、およびメンバー名の一時変更をサポートします。

「iSeries 情報の定義」ノートブックは、アプリケーションのビルド時およびアプリケーションの実行時に使用されます。ビルド時には、ファイル・ページは、ファイルの外部記述を見つけるファイルの取り出し中に使用されます。これは、定義仕様で指定されている場合には、外部記述データ構造にも使用されます。

アプリケーションの実行時には、「ファイル」ページは使用する実際のリモート iSeries 400 データベース・ファイルを見つけるために使用されます。 VisualAge RPG プログラムで使用されるファイル別名は、ファイル・ページで該当する項目を見つけるために使用されます。

ファイル・ページに項目が存在しない場合には、VisualAge RPG プログラムのファイルと同じ名前のファイルを見つけるために、サーバー・ページで定義された最初のサーバーのライブラリー・リストが使用されます。

実際のファイル名を VisualAge RPG プログラムで使用されるファイル名とは別にすることによって、実際のファイルを再ターゲット指定することができます。 VisualAge RPG プログラムを変更しないで、同じ iSeries サーバーまたは別の iSeries サーバー上の別のファイルに送ることができます。

210 ページの図 32 これを示す例が入っています。
- ファイル名と「iSeries 情報の定義」ノートブックの「ファイル」ページの中のファイル項目との関連付け。
- フィールドとパーツ名の突き合わせ。

注: NAME および ADDRESS 情報は、アプリケーションのウィンドウで入力しなければなりません。この情報は、**「OK」** プッシュボタンを押すと、データベースに入力されます。

```
 ****************************************************************************
 *                                                                          *
 * プログラム ID. : ioex.vpg                                                *
 *                                                                          *
 * 説明 . . . . . : ウィンドウのデータでデータベース・レコードを作成 *
 *                                                                          *
 * ファイル . . . : FILE1                                                   *
 *                                                                          *
 ****************************************************************************
 *
FFILE1     UF A E             DISK    REMOTE USROPN
 ****************************************************************************
 *                                                                          *
 * ウィンドウ : WIN1                                                        *
 *                                                                          *
 * パーツ . . : *INZSR                                                      *
 *                                                                          *
 * イベント . : 初期化ルーチン                                              *
 *                                                                          *
 * 説明: オープン・データベース・ファイル (FILE1)。                         *
 *                                                                          *
 ****************************************************************************
 *
C     *INZSR        BEGSR
C                   OPEN      FILE1
C                   ENDSR
 ****************************************************************************
 *                                                                          *
 * ウィンドウ : WIN1                                                        *
 *                                                                          *
 * パーツ . . . : PSB0000D                                                  *
 *                                                                          *
 * イベント . . : PRESS                                                     *
 *                                                                          *
 * 説明: レコードの作成を終了しました。アプリケーションを終了します。*
 *                                                                          *
 ****************************************************************************
 *
C     PSB0000D      BEGACT    PRESS         WIN1
C                   SETON                                         LR
C                   ENDACT
```

図 32. データベース・ファイルの例 (1/2)

```
*************************************************************************
*                                                                       *
* ウィンドウ : WIN1                                                     *
*                                                                       *
* パーツ . . : PSB0000C                                                 *
*                                                                       *
* イベント . . : PRESS                                                  *
*                                                                       *
* 説明: 画面からフィールド情報を読み取って AS/400 データベース・        *
*               ファイルにレコードを追加します。                        *
*                                                                       *
*                                                                       *
*************************************************************************
*
C     PSB0000C      BEGACT    PRESS         WIN1
C                   READ      'WIN1'
C                   WRITE     FORMAT1
C                   ENDACT
```

図 32. データベース・ファイルの例 (2/2)

ファイル仕様の FILE1 は、関連の「iSeries 情報の定義」ノートブックの「ファイル」ページに項目が存在するので、ファイル別名として使用されます。

サーバー TORAS180 のライブラリー LIB1 の FILE1 の最初のメンバーは、ファイルのオープン中に使用されます。リモート名の FILE1 は、ファイル項目の一時変更名と突き合わせる必要はありません。一時変更名は、VisualAge RPG プログラムで使用されるファイル・ページのファイル項目とファイル名の間のリンクを表します。

この 2 つの入力フィールドのパーツ名は NAME と ADDRESS です。VisualAge RPG は、同じ名前で同じ属性を持つフィールドを作成します。この例では、NAME および ADDRESS は 20 文字のフィールドです。データベース・ファイルにも、NAME と ADDRESS という名前 (両方とも 20 文字) の 2 つのフィールドが入っています。これらのフィールドの DDS を下に示します。

```
            R RECORD100
A             NAME             20A
A             ADDRESS          20A
```

フィールド名とその属性が一致すると、1 つのフィールドだけが作成されます。この例ではウィンドウからデータを読み取ります。

```
C* N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..

C                   READ      'WIN1'
```

この READ が実行されると、データは自動的に画面から NAME と ADDRESS の 2 つのフィールドに移動されます。データは現在は適切なフィールドにあるので、これ以上のフィールドの移動なしで直接データベースに書き込むことができます。

```
C* N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..

C                   WRITE     FORMAT1
```

この例では、NAME と ADDRESS の 2 つのフィールドのデータは、iSeries 400 データベースへの書き込みコマンドが出される前に、出力バッファーに自動的に移動されます。

レベル検査

VisualAge RPG は、使用する VisualAge RPG プログラムと iSeries 400 データベース・ファイルの間のレベル検査をサポートします。

コンパイラーは常にレベル検査に必要な情報を提供します。レベル検査は、データベース・ファイルの作成または変更時に LVLCHK(*NO) を指定しない限り、ファイルのオープン時にレコード様式基準で行なわれます。

注: レベル検査が行われると、これは入出力エラーとして処理されます。詳細については、 *VisualAge for RPG WINDOWS 版 言語解説書*を参照してください。

データベース・ファイルのロック

OS/400 システムでは、ジョブの実行中に使用されるファイルをロック状態（排他、排他読み取り許可、共用更新、共用非更新、または共用読み取り）にすることができます。ファイル・ロック状態ではジョブ内のプログラムは影響を受けません。ファイル・ロック状態が適用されるのは、別のジョブのプログラムが同時にファイルを使用しようとした時だけです。 ファイル・ロック状態は、CL コマンド ALCOBJ (オブジェクト割り振り) で割り振ることができます。リソースおよびロック状態を割り振る場合の詳細については、Web サイト **http://www.ibm.com/eserver/iseries/infocenter** の **Information Center** の中の *「プログラミング」* カテゴリーの *「CL および API」* セクションを参照してください。

OS/400システムは、ファイルのオープン時にデータベース・ファイルを次のロック状態にします。

- INPUT 用にオープン: 共用読み取りロック状態
- UPDATE 用にオープン: 共用更新ロック状態
- ADD 用にオープン: 共用更新ロック状態
- OUTPUT 用にオープン: 共用更新ロック状態

データベース・ファイルの指定変更

データベース・ファイルのライブラリー名またはファイル名を指定変更するためには、次の例に示されているように QCMDDDM コマンドを使用してください。

```
     D QCMDDDM         C                   'QCMDDDM' Linkage(*Server)
     C     OvrMenufl     BEGSR

     C                   Eval      QCMDDDM_Parm1 = 'OVRDBF FILE(MENUFL)' +
     C
' TOFILE(SYSLIBT/MENUFL)' +
     C
' MBR(' + MemberName + ')' +
     C
' OVRSCOPE(*JOB)' +
     C
' OPNSCOPE(*JOB)'
     C                   Exsr      CallExecDDM
```

```
C                   ENDSR


C     CallExecDDM   BEGSR

C                   EVAL      QCMDDDM_Parm2 =  %LEN(QCMDDDM_Parm1)
C                   Call          QCMDDDM
C                   Parm                    QCMDDDM_Parm1
C                   Parm                    QCMDDDM_Parm2

C                   ENDSR
```

iSeries 400 データベース入出力についての考慮事項

一般に、ILE RPG/400 言語で使用可能なすべての VisualAge RPG データベース入出力操作は、VisualAge RPG 言語でも使用することができ、意味体系上で同等です。詳細については (どの命令コードがローカル・アクセス、リモート・アクセス、あるいはその両方をサポートするかを含めて)、*VisualAge for RPG WINDOWS 版 言語解説書* を参照してください。

パフォーマンスを改善するためのレコード・ブロックの使用

アプリケーションが iSeries 400 サーバーからデータを読み取る場合には、レコード・ブロックを使用してアプリケーションのパフォーマンスを改善することができます。レコード・ブロックとは、ファイルの入出力操作が、一度に 1 レコードでなく、複数の順次レコード (レコード・ブロック) で行なわれることを意味します。

VisualAge RPG は、次のいずれかが進の場合にはデフォルトのレコード・ブロックを提供します。

- ファイルが出力専用で 1 つのレコード様式しか含まれていない。
- ファイルが入出力共用ファイルである。
- ファイルが入力専用で、1 つのレコード様式しか含まれておらず、OPEN、CLOSE、FEOD、および READ 命令コードしか使用しない。
- ファイル記述仕様に RECNO キーワードが指定されていない。

さらに、次の基準に適合するファイルでは明示のレコード・ブロックを実行することができます。

- ファイル追加項目 (20 桁目) がブランクである。
- ファイルに RECNO キーワードが使用されていない。
- ファイルに 1 つのレコード様式しかない。
- ファイルに CHAIN、SETLL、または SETGT 命令コードが使用されている。
- ファイルに READE、READP、または READPE 命令コードが使用されて**いない**。

ファイルが上の基準に適合する場合には、ファイル記述で BLOCK(*YES) を指定してプログラムを更新し、プログラムを再コンパイルして、レコード・ブロックを使用可能にすることができます。 214 ページの図 33 BLOCK キーワード・オプションを使用する例を示します。

```
     FFILE1     IF   E           K DISK    BLOCK(*YES)
     F                                     REMOTE

⋮
     C     FLD2          SETLL     REC1

⋮
     C                   READ      REC1                                   10
```

図 33. BLOCK キーワード・オプションの使用例

ファイル記述で BLOCK(*NO) を指定してプログラムを再コンパイルすると、デフォルトのレコード・ブロックを VisualAge RPG がサポートしていても、レコード・ブロックは行なわれません。

使用する iSeries 400 サーバー

TCP/IP を使用する場合は、最適化されたセントラル・サーバーおよびリモート・コマンド・サーバーがアクティブになっていなければなりません。STRHOSTSVR コマンドを使用してこれらのサーバーを始動します。このコマンドの詳細については、URL http://www.ibm.com/eserver/iseries/infocenter にある iSeries Information Center を参照してください。

これらのサーバーは、アプリケーションの開発時に必要となります。さらに、アプリケーションの実行時には、TCP/IP DDM サーバーもアクティブにしなければなりません。このサーバーを始動するには、STRTCPSRV コマンドを使用します。

実行時のサーバー接続の制御

VARPG は、実行時に始動される接続を制御するために 2 つのAPI を提供します。これらの API は VARPG プログラムで直接使用することができます。「API へのサインオン」によって、ユーザーは自分自身のサインオン情報を提供して iSeries サーバーにサインオンすることができます。「パスワード API の変更」によって、プログラマーはサインオン・パスワードに対する変更を扱うことができます。

VARPG 環境では、アプリケーションの存続期間中は接続はアクティブになったままです。接続は、アプリケーションのプロセス内で実行されるコンポーネントで共用されます。この動作はユーザー制御の VARPG 接続の場合にも当てはまります。ユーザー制御のサインオンは、接続が確立される方法について、通常の VARPG サインオンとは異なります。

VARPG ランタイム制御接続の始動の場合は、VARPG ランタイムは、プロジェクトのリモート・サーバー・テーブル (.RST) ファイルからサーバー名を取得し、VARPG セキュリティー・ファイルからユーザー ID / パスワードを取得してから、接続を確立します。セキュリティー・ファイルの中に情報がない場合には、ランタイムはユーザー ID およびパスワードの入力プロンプトを出します。ユーザー制御の接続始動の場合は、プログラマーは、VARPG ランタイムが接続を確立できるように、「API へのサインオン」を使用してサーバー名、ユーザー ID、およびパスワードを識別します。これらの API によってインターフェースが記述されている関数

は、FVDCWVC9.DLL の中にあります。この DLL は VARPG ランタイムの一部であり、パスの中にあります。この API を使用するアプリケーションのパス環境を再編成する必要はありません。

サインオン設定関数 VARPG_Set_Signon_Info は、次のパラメーターを受け入れ、サインオン処理の成功または失敗を示す数値戻りコードを提供します。
- サーバー名
- ユーザー ID
- パスワード

パラメーターは参照によって受け渡されるヌル終了文字変数です。

次の例は、API の C シグニチャーおよび RPG IV プロトタイプをリストしたものです。

```
* サインオン・プロトタイプ
* extern "C" int VARPG_Set_Signon_Info(char * server, char * userid,
*   char * password);


D signon          pr             5I 0 dll('FVDCWVC9')
D                                     extproc('VARPG_Set_Signon_Info')
D system                         *   value
D userid                         *   VALUE
D password                       *   VALUE
```

パスワード変更関数 VARPG_Change_Password は、1 つの追加パラメーター (使用する新規のパスワード) をもっています。この関数も、API 実行の成功または失敗を示す数値を戻します。そのパラメーターは次の通りです:
- サーバー名
- ユーザー ID
- 旧パスワード
- 新規パスワード

これらのパラメーターは参照によって受け渡されるヌル終了文字変数です。次の例は、C API および対応する RPG プロトタイプを示したものです::

```
* 新規パスワード・プロトタイプ
* VARPG_ENTRY int __cdecl VARPG_Change_Password(char * server,
*    char * userid, char * password, char * newpassword);


D newpassw        pr             5I 0 dll('FVDCWVC9')
D                                     extproc('VARPG_Change_Password')
D system                         *   value
D userid                         *   VALUE
D oldpassword                    *   VALUE
D newpassword                    *   VALUE
```

サーバー / ユーザー ID 情報を収集する方法には、異なる多くの方法があります。ここに示したサンプル・プログラムは、VARPG で書かれたそれ自身のサインオン・ダイアログを使用します。 VARPG アプリケーションとそのコンポーネントの接続が確立されることを記憶しておいてください。別の VARPG アプリケーションを始動する場合には、この新規のアプリケーションの接続を確立するために、再びサインオン API を使用する必要があります。その他の場合は、VARPG ランタイムはその通常の接続始動メカニズムを使用します。

外部記述ファイルを使用する VARPG アプリケーションで「API へのサインオン」を使用するためには、その外部ファイルのファイル仕様に USROPN キーワードを指定してください。USROPN を指定しない場合には、SIGNON 関数を呼び出す機会を得る前に、サーバー接続が確立されることになります。接続セッションが確立された後で始動されたコンポーネントでは、これらのコンポーネントに USROPN キーワードを指定しなくても、ファイルの RPG 暗黙オープンを使用することができます。コンポーネントはアプリケーションの既存の接続を再利用します。

次の戻りコードが両方の関数に適用されます:

```
OK                                          0
INVALID_PARAMETER                           1
INTERNAL_ERROR                              2
FUNCTION_NOT_SUPPORTED                      3
COMMUNICATIONS_ERROR                        4
SERVER_INVALID                              101
USER_ID_UNKNOWN                             201
USER_ID_REVOKED                             202
NEW_PWD_LENGTH_LONGER_THAN_MAX              301
NEW_PWD_LENGTH_SHORTER_THAN_MIN             302
NEW_PWD_CONTAINS_CHAR_USED_THAN_ONCE        303
NEW_PWD_CONTAINS_ADJACENT_DIGIT             304
NEW_PWD_CONTAINS_REPEATED_CONSECUTIVELY     305
NEW_PWD_PREVIOUSLY_USED                     306
NEW_PWD_MUST_CONTAIN_ONE_NUMERIC            307
NEW_PWD_CONTAINS_INVALID_CHAR               308
NEW_PWD_CONATINS_DISALLOWED_WORD            309
NEW_PWD_CONTAINS_USERID                     310
PASSWORD_INCORRECT                          311
PASSWORD_DISABLED_NEXT_INVALID_ATTEMPT      312
PASSWORD_EXPIRED                            313
NEW_PWD_CONTAINS_CHAR_SAME_POSITION_AS_LAST 315
```

API へのサインオンを使用したサンプル・プログラム

サンプル・アプリケーションでは、「サーバーへのサインオン」プッシュボタンでウィンドウが表示されます。このボタンを押すことによって、ユーザー ID / パスワード情報を収集するサインオン・コンポーネントが始動します。コンポーネントは、接続が成功したかどうかの信号を戻します。このアプリケーションはコンポーネントの参照パーツを使用して、サインオン・コンポーネントが完了したかどうかをモニターします。

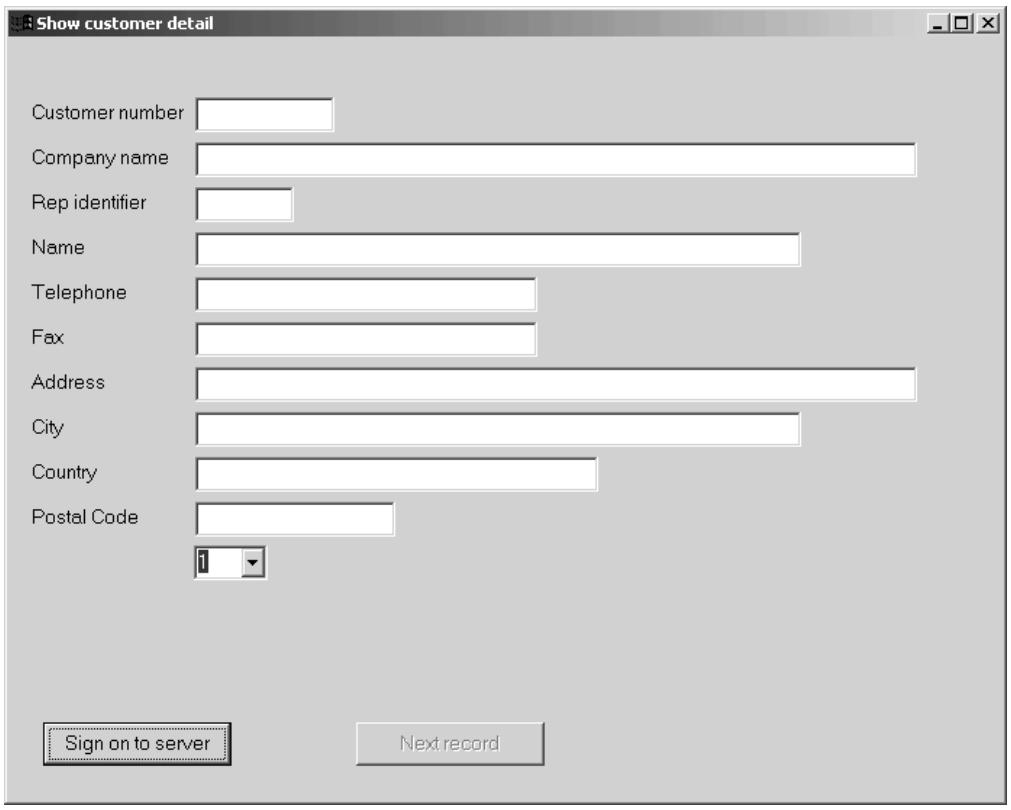
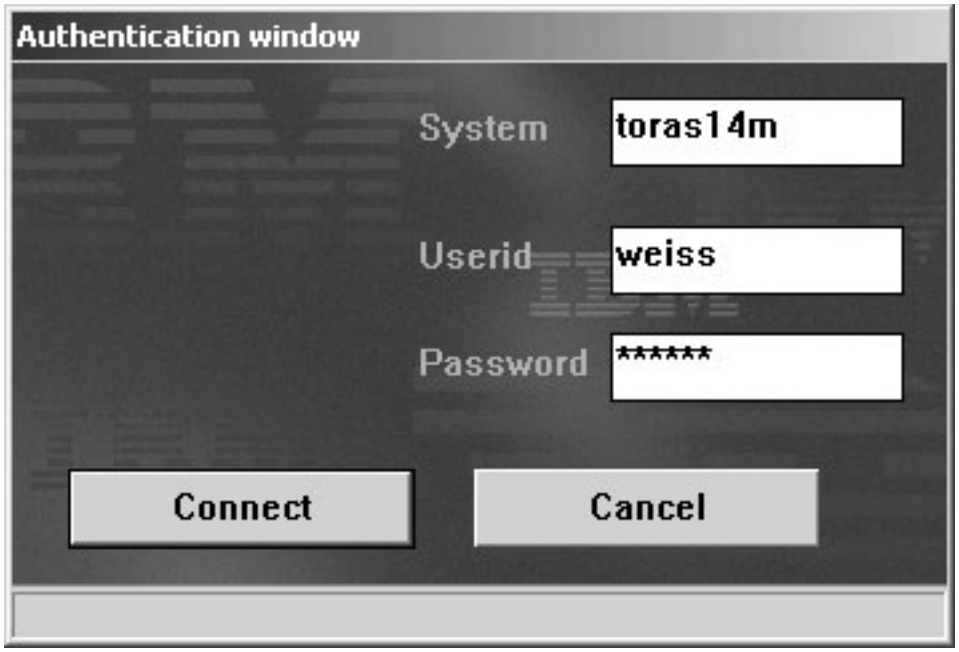
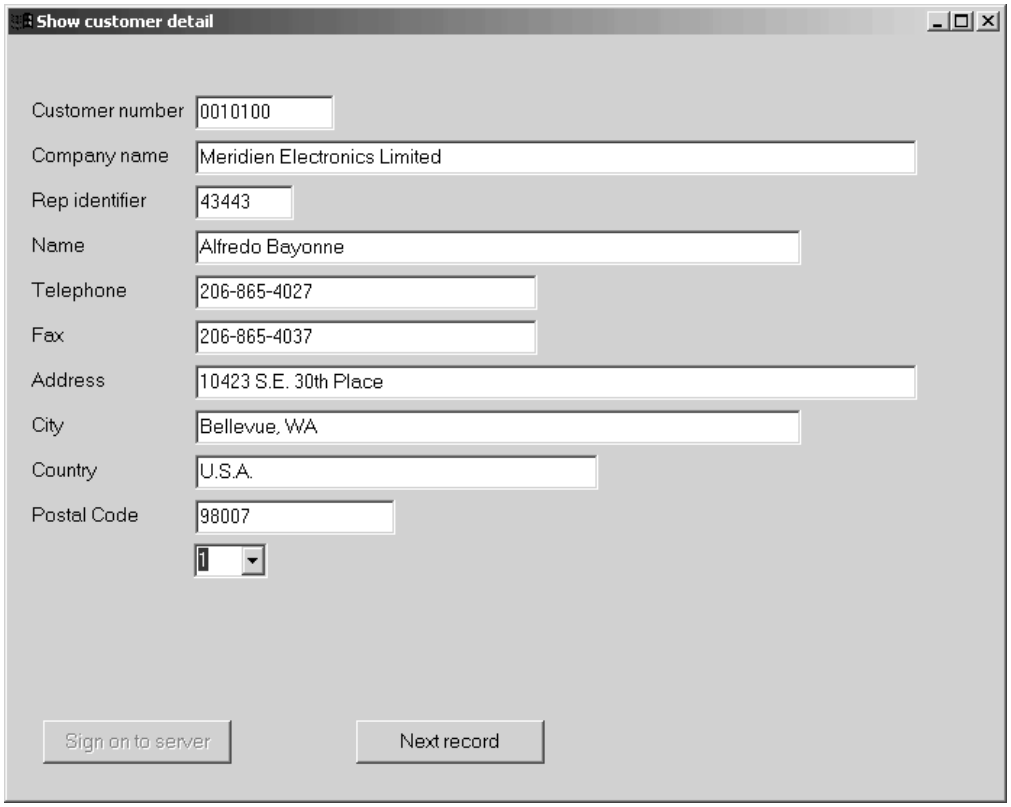


図 34. 「サインオン」プッシュボタンによる初期ウィンドウ

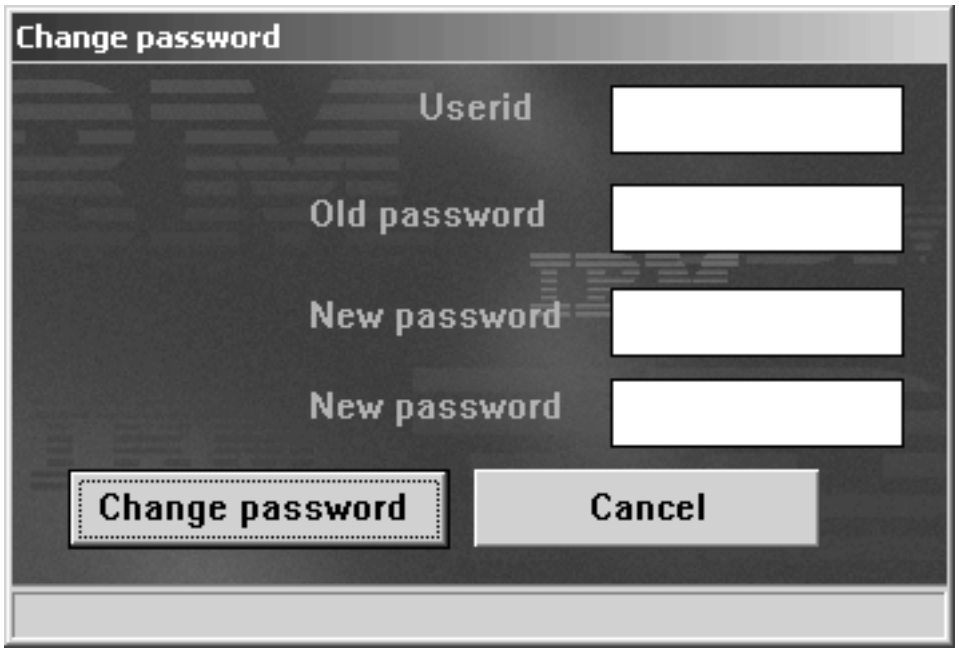
ファイルは USROPN キーワードで指定されているので、VARPG ランタイムでは接続要求は行われません。「サーバーへのサインオン」を押すことによって、コンポーネントが始動します。



これで、ユーザーはサーバーおよびユーザー ID / パスワード情報を指定することができます。コンポーネントはこのデータを使用して、サインオン API を始動し、サーバーとの接続を確立します。初期ウィンドウで、接続が正常に確立されたことが確認されたら、プログラムは iSeries サーバー上のカスタマイズ・データベースにアクセスします。



このサンプルでは特別に、ユーザーのパスワードが有効期限切れとなった場合に**パスワードの変更** ダイアログを表示します。この条件は、SIGNON 関数の戻りコードを検査することによって検出できます。



プログラムはこのダイアログを使用して新規のパスワードを取得してから、VARPG_Change_Password 関数を呼び出すことによって、そのパスワードを VARPG 通信レイヤーに送信します。これらの関数のコードはすべて **Runtime_control_of_server_connections** サンプル・プログラムに組み込まれています。

アプレット用のセキュリティー・ファイルの使用

VARPG アプリケーションがデータベース・ファイルなどの iSeries リソースを必要とする時は、 iSeries サーバーに接続するために、有効なユーザー ID およびパスワードが必要です。VARPG の場合は、ユーザー ID およびパスワードはクライアント・ワークステーションのセキュリティー・ファイルに保管されています。ただし、デフォルトのアプレットはクライアント・ワークステーション上のどのファイルにもアクセスできません。その結果、アプレットを実行するたびに、ユーザー ID およびパスワードの入力プロンプトが出されます。アプレットの実行時にセキュリティー・ファイルを使用し、このプロンプトが出されないようにしたい場合には、セキュリティー・ファイルを読み取るための *許可* を各アプレットに与える必要があります。これは、Sun Microsystem の J2SDK のパーツである **PolicyTool** ユーティリティーを使用して行います。PolicyTool の使用法の詳細については、J2SDK 1.2 解説書の中のツール資料のセクションを参照してください。

注: ここで述べられる手順は Windows クライアントでのみ有効です。

1. クライアント・ワークステーションにセキュリティー・ファイルを作成します。

   まだ作成されていない場合には、クライアント・ワークステーションに VARPG ランタイムをインストールしてください。**「スタート」**メニューから、**プログラム > VisualAge RPG Runtime > サーバー・ログオンの定義**を選択します。「サーバー・ログオンの定義」ダイアログが表示されます。特定の iSeriesサーバーのユーザー ID およびパスワードを追加して、それらをセキュリティー・ファイルに保管します。

2. 必要を作成します。

   許可 は、Java VM がセキュリティー・ファイルを読むために必要です。許可を作成するためには、次のようにしてください:

   a. MS DOS プロンプトから、**PolicyTool** を入力して、Enter (キー) を押します。「ポリシー・ツール」ダイアログ・ボックスが表示されます。

      初めて PolicyTool を使用する場合は、一定のディレクトリーの中にポリシー・ファイルが見つからないことを示すメッセージが表示されます。メッセージ中のポリシー・ファイルの名前およびディレクトリーをメモしてください。これは、後のステップでポリシー・ファイルを保管する場所となります。

   b. 「ポリシー・ツール」ダイアログから、**「ポリシー項目の追加」**ボタンをクリックします。「ポリシー項目」ダイアログが表示されます。CodeBase 入力フィールドに次のように入力してください:

      ```
      "http://xxxx/-"
      ```

      ここで、xxxx はセキュリティー・ファイルに対する許可を与えられるアプレットが入っている URLおよびディレクトリーです。

   c. **「許可の追加」**ボタンをクリックします。次のようにして「許可」ダイアログを完了します:
      - 「許可」組み合わせボックスから、**RuntimePermission** を選択します。
      - 「ターゲット」組み合わせボックスから、**loadLibrary.<library name>** を選択します。
      - 「ターゲット」組み合わせボックスの右隣の入力フィールドで、**loadLibrary.<library name>** を **loadLibrary.fvdcjava** に変更します。
      - **「OK」**を押して「ポリシー項目」ダイアログに戻ります。

   d. 「ポリシー項目」ダイアログから、再び**「許可の追加」**ボタンをクリックします。次のようにして「許可」ダイアログを完了します:
      - 「許可」組み合わせボックスから、**FilePermission** を選択します。
      - 「ターゲット」組み合わせボックスの右隣の入力フィールドに、セキュリティー・ファイルの名前を入力します。このファイルは Windows ディレクトリーの ibmcom サブディレクトリーに入っています。たとえば:

        ```
        c:\windows\ibmcom\fvdcsec.txt
        ```

        ここで、Windows ディレクトリーは *c:\windows* です。
      - 「アクション」ドロップダウン組み合わせボックスから、**読み取り**を選択します。
      - **「OK」**を押して「ポリシー項目」ダイアログに戻ります。
      - **「終了」**を押して「ポリシー・ツール」ダイアログに戻ります。

   e. 「ポリシー・ツール」ダイアログで、メニューから**「ファイル」**を選択し、次に**「別名保管」**を選択します。作成したポリシー・ファイルを、2a でメモしたファイル名およびディレクトリーに保管します。ここで**「ファイル」**を選択してから**「終了」**を選択して、「ポリシー・ツール」を終了します。

      まだユーザー ID およびパスワードの入力プロンプトが出される場合には、もう一度 PolicyTool を使用して、すべてのパラメーターを正しく指定していることを確認してください。

第 9 章 iSeries アプリケーションの再使用

VisualAge RPG アプリケーションを開発する場合には、既存の iSeries アプリケーションやその各部分を再使用することができます。この項では、iSeries 400 アプリケーションの再使用時に考慮する必要があるいくつかの事項について説明します。

再使用のシナリオ

VisualAge RPG を使用してサーバー上で実行されるアプリケーションを変更して、それらが PWS 上で実行され、ホスト上のデータにアクセスし、グラフィカル・ユーザー・インターフェースをもつようにすることができます。この項では、これらに関係するステップの概要を示します。

表示装置ファイルのインポート: インポート・ユーティリティーは、既存の表示装置ファイルを PWS のグラフィカル・ユーザー・インターフェースに変換します。表示装置ファイルをインポートした後で、レコード様式がユーザー定義パーツに変換されて、パーツ・カタログのインポート済みページに保管されます。アプリケーションでの作業時にパーツをパーツ・パレットに移動して、アプリケーションでの作業を終了したら、再びそれらを必要とするまで、パーツ・カタログに保管することができます。

たとえば、 222 ページの図 35 に示す 5250 画面をインポートすると、 222 ページの図 36 に示す GUI になります。レコードはパーツ・グループに変換され、フィールドは入力フィールド・パーツに変換され、固定情報は静的テキスト・パーツに変換されます。すべてのコマンド・キーはプッシュボタン・パーツに変換され、プッシュボタンのラベルは元のコマンド・キーのキーワードを反映します。

```
000 00 0000           000000000000000000000000000000      Status: 0000000

                    Purchase Order Header Maintenance   P.O. Number: 666666
-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.
       Vendor Number: 99999 0000000000000000000000000       000000000000000
                            0000000000000000000000000     66666666-666-6666
                            00000000000000000 0000000
                            000000000000000000000000000000

             Ship to: 9999  000000000000000000000000000000
                            000000000000000000000000000000
                            000000000000000000000000000000
              Print : B (Y,N)
            Ship Via: BBBBBBBBBB              Date Entered: 000 00 0000
              F.O.B.: BBBBBBBBBBBBBBB         Date Revised: 000 00 0000
         Terms Code : BBBB 000000000000000                  000 00 0000
            Password :                        Originator  : 000000000000000
  Prep./Collect/Chg : B (P,C,X)
     Confirm./Orig. : B (C,O)
           Warehouse: BB 000000000000000000000000000000
        Requested by: BBBBBBBBBBBBBBB         Work Order #: 9999999
-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.
Cmd1 Exit Cmd3 PO Notes Cmd4 Lookup Cmd5 Material Status Cmd9 Vendor Maint.
Cmd11 Delete            Cmd15 Vendor Notes  Cmd16 Vendor Quotes
```

図 35. 購入オーダー・アプリケーションからのサンプル 5250 画面

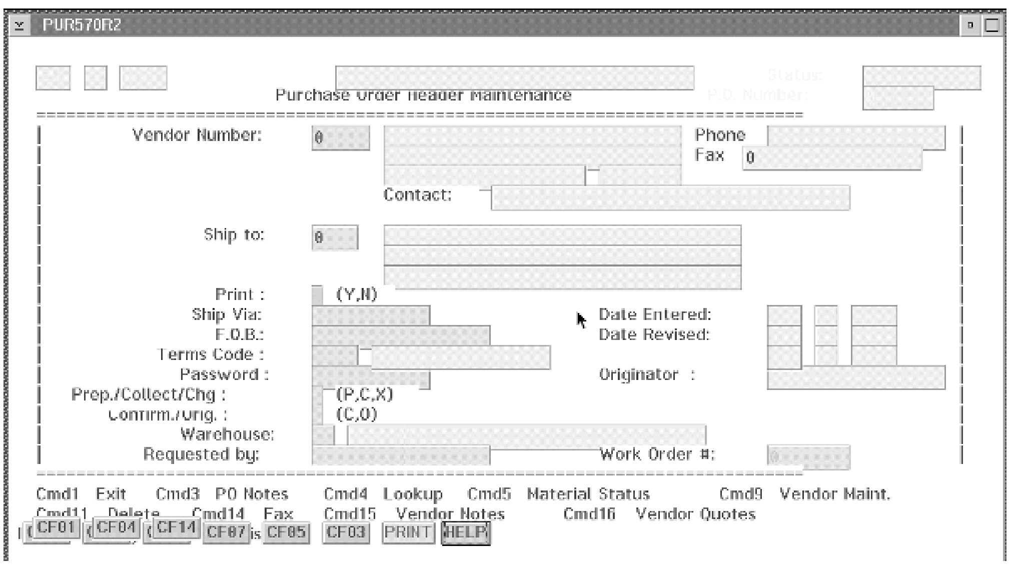


図 36. 購入オーダー画面のインポートの結果

注: 新しいパーツは元のフィールドめいを継承していますが、必要に応じてこれらをリネームすることができます。同じフィールド名を保持することで、再使用時にプログラム・ロジックを操作すれば生産性が向上します。

インポートしたパーツをカスタマイズして、21 ページの『第 2 章 アプリケーションの計画』 で説明した基本設計問題を活用することができます。

GUI のカスタマイズ:

図 37. 購入オーダー・アプリケーションの GUI のカスタマイズ

インターフェースに次の変更を行なった場合に、インポートしたウィンドウがどのようになるかを 図 37 に示します。

- コマンド・キーをメニュー・バーおよび関連メニュー項目と置き換えます。たとえば、**ベンダー**・メニューには、当初プッシュボタンに変換されたアクションが入っています。これは、頻繁に実行するアクションに容易にアクセスできるようにして、関連ウィンドウを起動します。
- 関連ウィンドウをグループ化し、グループ・ボックス・パーツを使用して、ユーザーに視覚的合図を送ります。たとえば、**出荷通知** とラベル付けされたグループ・ボックスには、出荷通知に関連した入力フィールドが入っています。
- ユーザーが既知の選択項目番号から選ぶ必要がある情報には、グループ化されたラジオ・ボタンを使用します。たとえば、3 種類の支払方法 (前払い、集金、課金) のみ可能な場合、ユーザーは使用する方法を指示するために 1 つのラジオ・ボタンを選択するだけで済みます。このラジオ・ボタンは**支払い**とラベル付けされたグループ・ボックスにあります。

これらの変更は、VisualAge RPG が提供するグラフィカル機能を活用しています。

オンライン・ヘルプの再使用: アプリケーションを再使用する場合には、既存のユーザー・インターフェース管理機能 (UIM) のヘルプも再使用することができます。

新しい GUI の外観を反映するために多少の変更は必要ですが、ゼロからヘルプを作成するために必要な努力は省くことができます。

変換された UIM のヘルプをカスタマイズし、情報表示機能 (IPF) を使用して新しいタイプのヘルプを追加することができます。ユーザーはそれぞれのウィンドウにヘルプを作成し、それぞれのパーツにヘルプ情報を書き、ヘルプのソースにハイパーテキスト・リンクを作成してヘルプ情報にリンクすることができます。詳細については、231 ページの『UIM ヘルプの再使用』 および 257 ページの『第 13 章 IPF を用いたオンライン・ヘルプ作成のヒント』 を参照してください。

プログラム・ロジックの作成: コンパイラーは RPG IV に基づいているので、RPG IV で書かれたロジックは再使用することができます。再使用するためには、単に既存のコードをカット・アンド・ペーストするだけです。

また、イベント・ドリブン・プログラミングを使用して、追加のプログラム・ロジックも書く必要があります。パーツに関連したすべてのイベントに対して、プログラムがイベントに応答する方法を記述したアクション・サブルーチンがあります。プログラム制御用の手続き型命令コードは不要です。プログラム制御は暗黙に行なわれます。 VisualAge RPG アプリケーションに固有の VisualAge RPG 命令コードの一部は次のとおりです。

BEGACT
アクション・サブルーチンを開始します。

ENDACT
アクション・サブルーチンを終了します。

SETATR
パーツ属性の値を設定します。

GETATR
パーツ属性の値を検索します。

SHOWWIN
ウィンドウを表示します。

CLSWIN
ウィンドウをクローズします。

225 ページの図 38 には、サンプル購入オーダーからのアクション・サブルーチンが入っています。特定のウィンドウから PUR570R2 ウィンドウを表示するために SHOWWIN が呼び出されると、 **作成**イベントにアクション・サブルーチンがコーディングされてユーザーの次のアクションのためのウィンドウが準備されます。

これが新しい購入オーダー (#PONUM = 0) の場合には、メニュー項目**変更**、**削除**、 **印刷**、および**ファクシミリ** が、MENUSELECT イベントに応答**しない**ように設定されます。メニュー項目のそれぞれに対して、%setatr 機能が **enabled** 属性を *0* に設定するために使用されます。BEGACT 命令コードは、アクション・サブルーチンの開始を示し、ENDACT はその終了を示します。

```
 ********************************************************************
 ********************************************************************
 *
 * Window . . :  PUR570R2 PO Header Maintenance                     *
 *                                                                  *
 ********************************************************************
 ********************************************************************-
 *
C     PUR570R2      BEGACT    CREATE        PUR570R2
c*
c*
C                   if        #PONUM = 0
C                   eval      %setatr('pur570r2':'m2_change':'enabled')=0
C                   eval      %setatr('pur570r2':'m2_delete':'enabled')=0
C                   eval      %setatr('pur570r2':'m2_print':'enabled')=0
C                   eval      %setatr('pur570r2':'m2_fax':'enabled')=0
c                   end
c                   exsr      POCHECK
c                   write     'PUR570R2'
C                   ENDACT
 ********************************************************************-
```

図 38. サンプル・アクション・サブルーチン

ホストへの接続: アプリケーションのビルド時にアプリケーションが iSeries 400 データベース・フィールドを使用するか、または iSeries 400 表示装置ファイルをインポートする場合には、使用される iSeries 400 サーバーを定義しなければなりません。サーバー・ログオン情報が正しく定義されている限り、ホストとはいつでも通信することができます。サーバー情報を定義する場合の詳細については、オンライン・ヘルプを参照してください。

表示装置ファイルのインポート

設計ウィンドウに組み込みたいレコード様式が含まれている表示装置ファイルがある場合には、それらのレコード様式を iSeries サーバーから VisualAge RPG 環境にインポートすることができます。インポートの完了後、レコードはユーザー定義パーツに変換され、パーツ・カタログのインポート済みページに格納されます。表示装置ファイルをインポートする前に:

- アクセスする サーバーの定義を行ってください。オンライン・ヘルプを参照してください。
- これらの表示装置ファイルに対して少なくとも *USE 権限が必要です。

表示装置ファイルをインポートする場合:

1. GUI Designer から**サーバー→表示装置ファイルのインポート**を選択します。
2. これらの方式の 1 つを使用しての表示装置ファイルの選択:
   - 表示装置ファイルの完全名 (*<server >library /file*) を**ファイル名**入力フィールドに入力して、カーソルがそのフィールドにまだある時に実行キーを押してください。
   - 次の通り実行してください。
     a. 表示されたサーバーの 1 つを選択します。ライブラリーのリストがサーバー名の下に表示されます。
     b. 表示されたライブラリーの 1 つを選択します。表示装置ファイル・オブジェクトのリストが表示されます。

c. 示された表示装置ファイル・オブジェクトの 1 つを選択します。ファイル中のレコードのリストが表示されます。

3. インポートしたいレコードを選択します。
4. デフォルト・パーツ名を使用したくない場合には、パーツ名を**パーツ名**入力フィールドに入力してください。
5. デフォルト・アイコンを使用したくない場合には、**「検索」**プッシュボタンを押して、「アイコンの検索」ウィンドウを使用してください。
6. 次の 1 つを選択して、そのパーツを入れたい位置を指示します。

   カタログのみ
   パーツはカタログのインポート済みページに追加されます。

   カタログとパレット
   パーツはカタログのインポート済みページに追加され、インポートされたパーツのアイコンがパレットに現れます。
7. **インポート**を選択します。

表示装置ファイルの変換

表示装置ファイルをインポートすると、同等のパーツや属性を持つレコード様式、フィールド、およびキーワードが変換されて、これらのパーツを他のパーツと同様に使用することができます。同等のパーツを持っていないレコード様式、フィールド、およびキーワードは変換されません。

一般には次のようになります。

- レコード様式は、そのレコード・タイプによって VisualAge RPG ウィンドウまたはパーツのグループに変換されます。
- フィールドは入力フィールドに変換されます。
- 固定情報は静的テキスト・パーツに変換されます。
- 条件付け標識オプションは無視されます。

注: 長さが 64 バイトを超えるフィールドは GUI では 64 バイトのサイズになりますが、プロパティー・ノートブックの長さ属性は元の長さに設定されます。

レコード様式

次のリストはディスプレイ・レコード様式がどのようにパーツに変換されるかを記述しています。

MNUBAR
MNUBAR レコード様式はメニュー・バー・パーツに変換され、キャンバス付きウィンドウにドロップすることができます。

PULLDOWN
PULLDOWN レコード様式は MNUBAR レコード様式と一緒に使用され、サブメニュー・パーツを作成します。PULLDOWN レコード様式は、それが参照する MNUBAR レコード様式のメニュー項目パーツに変換されます。

RECORD
RECORD レコード様式はパーツ・グループに変換され、キャンバス付きウィンドウにドロップすることができます。

SFL、SFLCTL

これらのレコード様式はサブファイル・パーツに変換され、キャンバス付きウィンドウにドロップすることができます。

SFL レコード内の固定情報は変換されません。

WINDOW

WINDOW 定義レコード様式はキャンバス付きウィンドウ・パーツに変換されます。WINDOW 参照レコード様式はパーツ・グループに変換され、キャンバス付きウィンドウにドロップすることができます。

次のレコード様式は変換されません: ERRSFL、SFLMSG、および USRDFN。サブファイル内の PULLDOWN および WINDOW キーワードは変換されません。

定位置項目

次のテーブルに、表示装置ファイルの作成に使用される DDS の定位置項目がどのように形式およびフィールドの変換方法を決定するかを示します。

表 6. 定位置項目および変換

桁	意味	項目および変換の結果	
8-16	標識		変換されません
17	レコード・タイプ	**R**	レコード様式に説明されている通りに変換されます
		H	変換されません
19-28	名前		名前フィールドの場合には、パーツの名前として使用されます
29	解説	**R**	変換されません
30-34	長さ		データ長を設定します
35	キーボード・シフト	**A**	文字に変換されます
		E	データ・タイプは DBCS 択一に設定されます
		G	データ・タイプは DBCS 専用に設定されます
		I	読み取り専用、使用不能
		J	データ・タイプは DBCS 専用に設定されます
		O	データ・タイプは DBCS 混用に設定されます
		D F M N S W X Y	変換されません
36-37	小数部分		変換されたパーツの小数点以下の桁数を決定します。指定した場合には、データ・タイプを数字に設定します。

表 6. 定位置項目および変換 (続き)

桁	意味	項目および変換の結果	
38	使用状況	**I B**	Enabled
		M P	変換されません
		O	読み取り専用、使用可能
		H	フィールドがサブファイル・フィールドの時変換されます
39-41	位置		ウィンドウ上のパーツの位置を決定します
45-80	キーワード	**固定情報**	静的テキスト・パーツを作成します

表示装置ファイル・キーワード

次のリストは表示装置ファイル・キーワードがパーツに変換される方法について記述しています。これらのキーワードと一緒に使用されるオプションによって、キーワードを変換するかどうかを決定することができます。

CFnn、CAnn
これらのキーワードは、キーワードの名前と同じラベルのプッシュボタンに変換されます。

CHOICE
MLTCHCFLD および SNGCHCFLD を参照してください。

CNTFLD
CNTFLD キーワードは複数行の編集パーツに変換されます。

COLOR
COLOR キーワードは、パーツの前景カラー属性を決定します。複数の COLOR キーワードがある場合には、最後のキーワードが使用されます。

COMP COMP キーワードは、パーツのプロパティー・ノートブックの比較属性を設定するために使用されます。

DATE DATE キーワードは静的テキスト・パーツに変換されます。

DFT DFT に対応するテキストは、パーツのラベルになります。
- 変換されるフィールドが固定情報フィールドの場合には、DFT 値がラベルに使用されます。
- 変換されるフィールドが名前フィールドに指定されている場合には、DFT キーワード値がパーツのデフォルトのテキストに変換されます。

DFTVAL
DFTVAL キーワード値は、パーツのデフォルト値に変換されます。

DSPATR
DSPATR 表示属性が次の場合:
- HI 前景カラーがさらに高輝度になります。
- ND 変換されたフィールドが非可視に設定されます。
- PR 変換されたフィールドが使用不能に設定されます。

その他の表示属性は変換されません。

HELP HELP キーワードは、ラベルが HELP のプッシュボタンに変換されます。ヘルプ機能は変換されません。

MLTCHCFLD

MLTCHCFLD キーワードが PULLDOWN レコード内で使用される場合には、これに対応するそれぞれの CHOICE キーワードがサブメニュー・パーツのメニュー項目パーツに変換されます。変換されたメニュー項目パーツには、横にそれが活動状態であることを示すチェック・マークが付いています。

MLTCHCFLD キーワードが PULLDOWN レコードの外で使用される場合には、関連する各 CHOICE キーワードは、チェック・ボックス・パーツに変換されます。チェック・ボックスは、同じデフォルト間隔で水平方向に配置されます。これらはグループ化されません。

MNUBAR

MNUBAR レコード様式はメニュー・パーツに変換されます。

MNUBARCHC

それぞれの MNUBARCHC はメニュー項目に変換されます。

PRINT PRINT キーワードは、ラベルが PRINT のプッシュボタンに変換されます。印刷機能は変換されません。

PSHBTNCHC、PSHBTNFLD

PSHBTNFLD はプッシュボタン・パーツに変換されます。PSHBTNCHC キーワードに関連するテキストは、プッシュボタン・パーツのラベルに変換されます。

PULLDOWN

PULLDOWN レコード様式は MNUBAR レコード様式と一緒に使用され、サブメニュー・パーツを作成します。PULLDOWN レコード様式は、それが参照している MNUBAR レコード様式のメニュー項目パーツに変換されます。

RANGE

RANGE キーワードはパーツの range 属性に変換されます。

SFL、SFLCTL

これらのレコード様式はサブファイル・パーツに変換されます。

SFLPAG

サブファイル・パーツの初期高さに影響します。

SNGCHCFLD

SNGCHCFLD キーワードが PULLDOWN レコード内で使用される場合には、これに対応するそれぞれの CHOICE キーワードがサブメニュー・パーツのメニュー項目パーツに変換されます。

SNGCHCFLD キーワードが PULLDOWN レコードの外で使用される場合には、関連する各 CHOICE キーワードは、チェック・ボックス・パーツに変換されます。ラジオ・ボタンは、同じデフォルト間隔で水平方向に配置されます。これらはグループ化されません。

SYSNAME
SYSNAME キーワードは静的テキスト・パーツに変換されます。

TIME TIME キーワードは静的テキスト・パーツに変換されます。

USER USER キーワードは静的テキスト・パーツに変換されます。

VALUES
VALUES キーワードによって、フィールドはドロップダウン組み合わせボックス・パーツに変換されます。VALUES キーワードに対応する値がドロップダウン・リストで使用されます。

WDWTITLE
WDWTITLE キーワードは、キャンバス・パーツ付きウィンドウのラベルおよび属性を決定するために使用されます。
- タイトル・テキストは、「プログラム - システム」フィールドに割り当てられている場合には変換されません。
- タイトル・テキストがリテラル・フィールドに割り当てられている場合には、キャンバス付きウィンドウ・パーツのラベルがこのテキストに設定されます。

WINDOW
WINDOW 定義レコード様式はキャンバス付きウィンドウ・パーツに変換されます。WINDOW 参照レコード様式はパーツ・グループに変換され、キャンバス付きウィンドウにドロップすることができます。

その他のキーワードはどれも変換されません。

変換カラー

コンピューター画面の文字基準の入力フィールドは、カラー・コード化された入力フィールド・パーツに変換されます。

注: 必ずしもすべての画面がこれらのカラーをサポートしているわけではありません。たとえば VGA 画面では、変換された入力フィールドは白色になります。

変換された各入力フィールドのカラーは、表示装置ファイルに定義されたタイプおよび定義によって異なります。

表 7. 変換前と変換後のフィールドの属性

フィールド・タイプ　フィールド属性	GUI 属性
I/O*	ReadOnly: オフ Enabled: オン Color: 淡黄色
出力	ReadOnly: オン Enabled: オン Color: 淡緑色
入力	ReadOnly: オフ Enabled: オン Color: 淡青色
入力または入出力　保護	ReadOnly: オフ Enabled: オフ Color: 淡赤色

表 7. 変換前と変換後のフィールドの属性 (続き)

入力または入出力	キーボード使用禁止	ReadOnly: オン Enabled: オン Color: 中間の赤色
入力または入出力	キーボード使用禁止保護	ReadOnly: オン Enabled: オフ Color: 暗桃色

注: I/O = 入出力

カラー・コードによって、入力フィールド・パーツに設定された属性を視覚的に判別することができます。たとえば、薄緑色のフィールドが表示されている場合には、それはプログラムがデータの表示に使用するもので、ユーザーの入力は受け入れられないことを知ることができます。淡赤色の入力フィールドが表示されている場合には、ユーザーの入力は受け入れられるが、元のアプリケーションの保護フィールドであるために使用できないことを知ることができます。

UIM ヘルプの再使用

VisualAge RPG ヘルプ・ファイルが **情報表示機能** (IPF) を使用して書かれていても、 **ユーザー・インターフェース管理機能** (UIM) を使用して書かれた iSeries 400 ヘルプを再使用することができます。UIM と IPF 形式は両方とも**汎用マークアップ言語** (GML) の原則を使用していて、高度の互換性があります。IPF の使用法の詳細については、*Information Presentation Facility Guide and Reference*(オンラインで使用可能)を参照してください。また、*IPF の制約事項*というタイトルのオンライン文書も参照してください。この文書には、Windows 環境に制限されている IPF タグのサブセットに関する詳細が記載されています。

UIM ヘルプ・ファイルを再利用するには、

1. エディターを使用して、UIM ヘルプが入っているメンバーをコピー・アンド・ペーストします。
2. UIM タグを適切な IPF タブに変更します。

次の項では、UIM タグと IPF タグのいくつかを比較します。

同じタグを使用する UIM および IPF 機能

UIM と IPF には同等の機能があり、まったく同じにタグ付けされます。これらの場合には、UIM タグをまったく同じ語で使用することができます。

表 8. 同一の UIM および IPF タグ

UIM タグ	タグ機能	IPF タグ
:DL.	定義リスト	:dl.
:FIG.	図	:fig.
:HP1.	強調表示句	:hp1.
:HP2.	強調表示句	:hp2.
:HP3.	強調表示句	:hp3.
:HP4.	強調表示句	:hp4.

表 8. 同一の UIM および IPF タグ (続き)

UIM タグ	タグ機能	IPF タグ
:HP5.	強調表示句	:hp5.
:HP6.	強調表示句	:hp6.
:HP7.	強調表示句	:hp7.
:HP8.	強調表示句	:hp8.
:HP9.	強調表示句	:hp9.
:LINES.	線	:lines.
:LI.	リスト項目	:li.
:LP.	リスト・パーツ	:lp.
:NT.	注	:nt.
:OL.	順序リスト	:ol.
:P.	段落	:p.
:PARML.	パラメーター・リスト	:parml.
:P	パラメーター記述	:pd.
:PT.	パラメーター用語	:pt.
:SL.	単純リスト	:sl.
:UL.	番号なしリスト	:ul.
:XMP.	例	:xmp.
&AMP.	アンパーサンド (&)	&.
&COLON.	コロン (:)	&colon.
&period.	ピリオド (.)	&period.
&SLR.	右スラッシュ (/)	&slr.

別のタグを使用する同等の UIM と IPF

UIM と IPF には同等でタグが異なる機能があります。この状態では、UIM のタグ付けをそれと同等の IPF のタグ付けに変更します。

表 9. 同等の UIM と IPF のタグ

UIM タグ	機能	IPF タグ
:CIT.	引用	:hp5.
:H1.	見出し	:h2.
:H2.	見出し	:h3.
:H3.	見出し	:h4.
:H4.	見出し	:h5.
:HELP.	見出し	:help.
:ISCH.	索引項目	:i1.
:ISCHSYN.	索引シノニム	:isyn.
:PK.	プログラミング・キーワード	:hp2.
:PK. :DEF 付き	デフォルトのプログラミング・キーワード	:hp7.
:PV.	プログラミング変数	:hp5.

IPF に同等のものがない UIM の機能

UIM には IPF にない機能があります。この状態では、この機能を削除するか、または IPF のタグ付けを使用して実行する別の方法を見つけます。

表 10. IPF に同等のものがない UIM のタグ

UIM タグ	機能	推奨される IPF の置き換え
:HP0.	強調表示なし	テキストの周囲に :hp*n* タグを使用しません。
:PC.	段落継続	タグを使用しません。段落のテキストに続きます。
:RT.	反転テキスト	:hp*n* タグを使用する別のタイプの強調表示を使用します。
:XH1.	全般ヘルプの見出し	IPF には全般ヘルプはありません。全般ヘルプを提供する別のウィンドウ (:h1.) へのハイパーテキスト・リンクを作成するために :link. タグを使用します。
:XH2.	全般ヘルプの見出し	IPF には全般ヘルプはありません。全般ヘルプを提供する別のウィンドウ (:h1.) へのハイパーテキスト・リンクを作成するために :link. タグを使用します。
:XH3.	全般ヘルプの見出し	IPF には全般ヘルプはありません。全般ヘルプを提供する別のウィンドウ (:h1.) へのハイパーテキスト・リンクを作成するために :link. タグを使用します。
:XH4.	全般ヘルプの見出し	IPF には全般ヘルプはありません。全般ヘルプを提供する別のウィンドウ (:h1.) へのハイパーテキスト・リンクを作成するために :link. タグを使用します。

RPG ソースの再使用

サーバー上の RPG ソース・コードを再利用するには、

1. ソース・コードが RPG IV の構文でない場合には、サーバー上の ILE RPG 変換ツール (**CVTRPGSRC**) を使用してそれを RPG IV の構文に変換します。
2. エディターを使用して、RPG ソースおよび通常使用されるサブルーチンが入っているメンバーを**コピー・アンド・ペースト**します。
3. 構文検査機能は、コンパイラーがサポートしない命令コードを強調表示します。サポートされる命令コードについては、*VisualAge for RPG WINDOWS 版 言語解説書* を参照してください。

   注: 命令コードの違いに加えて、 RPG IV 言語と VisualAge RPG コンパイラーの間には、RPG ソースを再使用する前に注意しなければならない他の多くの相違点があります。RPG IV 言語と VisualAge RPG 言語の相違点については、*VisualAge for RPG WINDOWS 版 言語解説書* を参照してください。

第 4 部 拡張トピック

第 10 章 アプリケーションのデバッグ

VisualAge RPG と一緒に提供されたデバッガーは、アプリケーション内のエラーの検出および診断を助けます。これは、複数言語アプリケーションのデバッグに使用することができます。次のことを行うことができます。

- アプリケーションおよび DLL の実行管理。
- ブレークポイントの設定および管理。
- ストレージ、レジスター、変数、および呼び出しスタック・ウィンドウを使用した、プログラム状態の表示および変更。

この項では、VisualAge RPG プログラムを使用したこれらの機能の一部を示します。

VARPG によって生成された Java コードをデバッグするには、Windows NT と分散デバッガーが必要です。

デバッガーの開始

デバッガーを開始するためには、プロジェクト・メニューから**デバッグ**・メニュー項目を選択してください。 2 つのウィンドウが表示されます。**デバッグ・セッション制御**ウィンドウと**VisualAge RPG ソース**・ウィンドウです。 238 ページの図 39 これらの 2 つのウィンドウを示します。

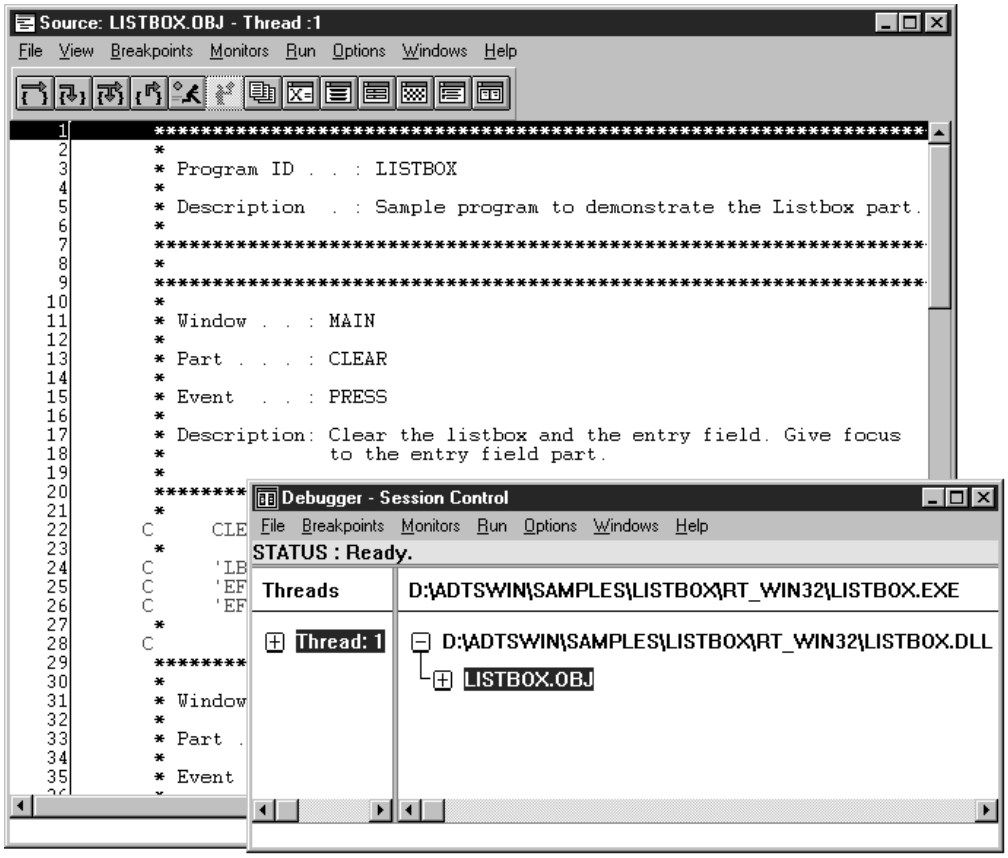


図 39. VisualAge RPG ソースおよびデバッグ・セッション制御ウィンドウ

デバッガーが開始されると、**VisualAge RPG ソース・メンバー**が検索されて、**ソース・ウィンドウ**に表示されます。ソースが表示されたら、デバッグ・タスクを実行することができます。

注: 実行中のプログラムの現在の位置が、強調表示された行番号で示されます。ここでは、ソース・メンバーの最初の行が強調表示されています。

アセンブリー・コードの表示

デバッガーが VisualAge RPG ソースを見つけられない場合には、プログラムがロードされて、VisualAge RPG ソース・コードの代わりに、**アセンブリー・ソース・コード**が表示されます。表示されるウィンドウは、 239 ページの図 40 で表示されるものと似ています。

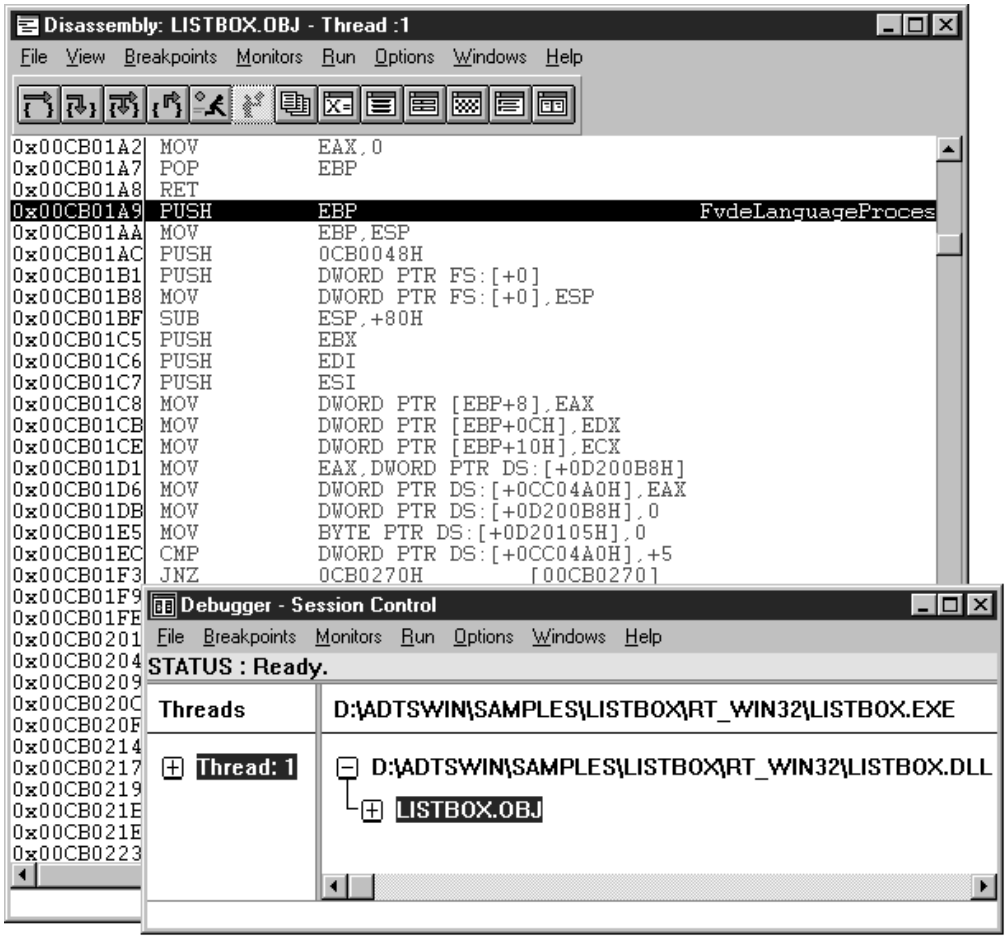


図 40. ディスアセンブリー・ウィンドウ :

この問題を訂正するために、VisualAge RPG ソース・コード (.VPG ファイル) がワークステーションに入っていることを確認してください。

DLL オカレンスのロード

アセンブリー・ソース・コードが表示された場合には、VisualAge RPG ソース・メンバーが見つからないことを意味します。これを解決するためには、**ブレークポイント**・プルダウン・メニューから**オカレンスのロード設定**を選択してください。「オカレンスのロード」ブレークポイント・ウィンドウが表示されます (240 ページの図 41)。このウィンドウから、**DLL オカレンス**をロードすることができます。次の情報を入力してから OK を選択してください。

```
application_name.DLL
```

これでデバッグ・セッションに戻ります。再度アセンブリー・ソース・ビューが表示されたら、R キーを押してプログラムの実行を再開してください。DLL がロードされると、システムはメッセージを表示し、制御ウィンドウにそのアプリケーションの名前が表示されます。ここでアプリケーションをクリックしてソースを表示することができます。ソースが表示されない場合には、ソースが失われたかまたは削除していることを意味しています。

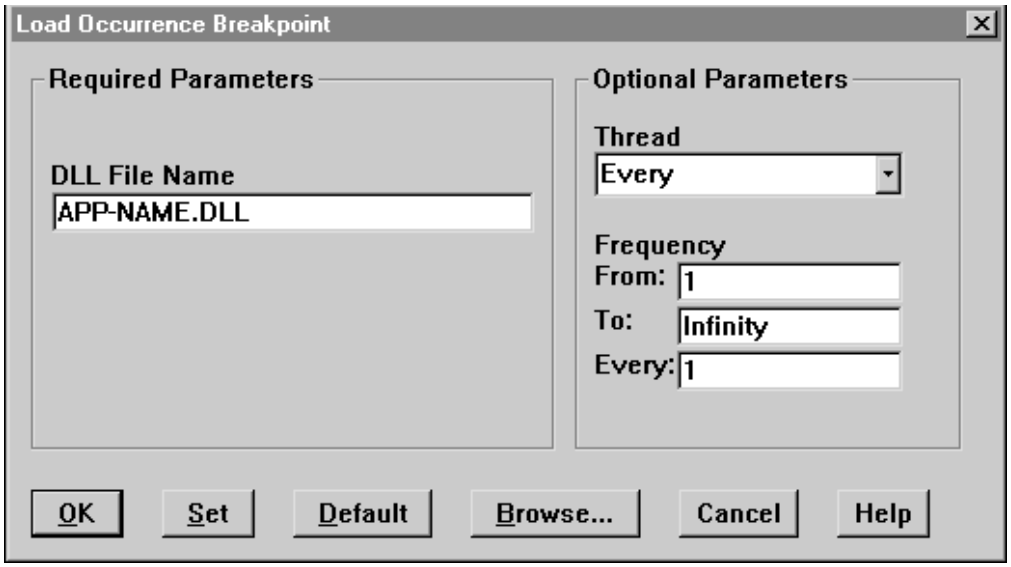


図 41. オカレンスのロード・ブレークポイントの設定:

アプリケーションが START 命令コードを使用して別のコンポーネントを開始する場合には、この手順を使用して他のコンポーネント DLL をロードする必要があります。これによって、他のコンポーネント内にブレークポイントを設定することができます。

デバッグ始動情報の入力

実行可能なファイルを見つけることができない場合には、デバッガーは下の 図 42 に似たウィンドウを表示します。このウィンドウにプログラム名とパラメーターを再入力することができます。

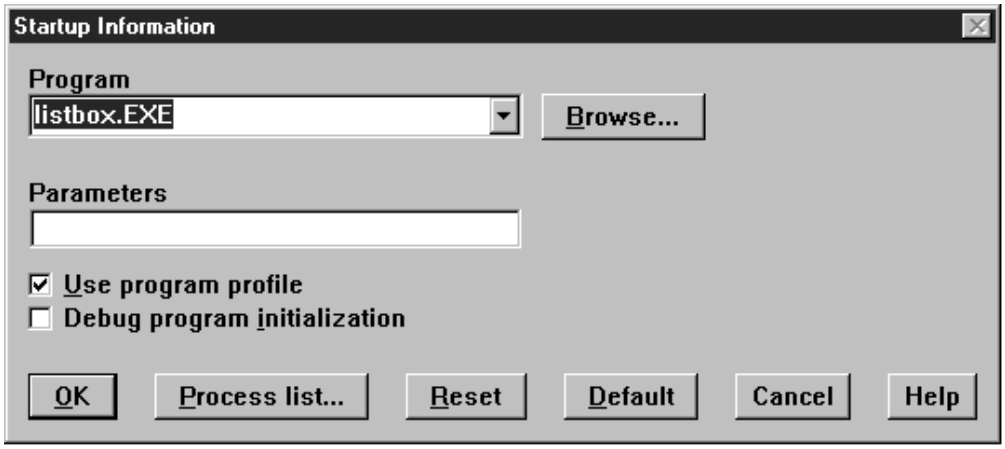


図 42. 始動情報:

ブレークポイントの設定

ブレークポイントを設定して、プログラムの実行方法を制御することができます。ブレークポイントは、特定の場所か特定のイベントが起こった時にプログラムの実行を停止します。ブレークポイントを設定するためには、中断したい行番号にカーソルを移動して、左マウス・ボタンをダブルクリックしてください。デバッガーは赤マークでその行番号を強調表示します。このプロセスは、中断したい、必要なすべての行にマークが付けられるまで何回でも繰り返すことができます。図 43 画面に複数のブレークポイントが設定されているところを示します。すべてのブレークポイントは、ブレークポイント・リスト・ウィンドウで表示することができます。

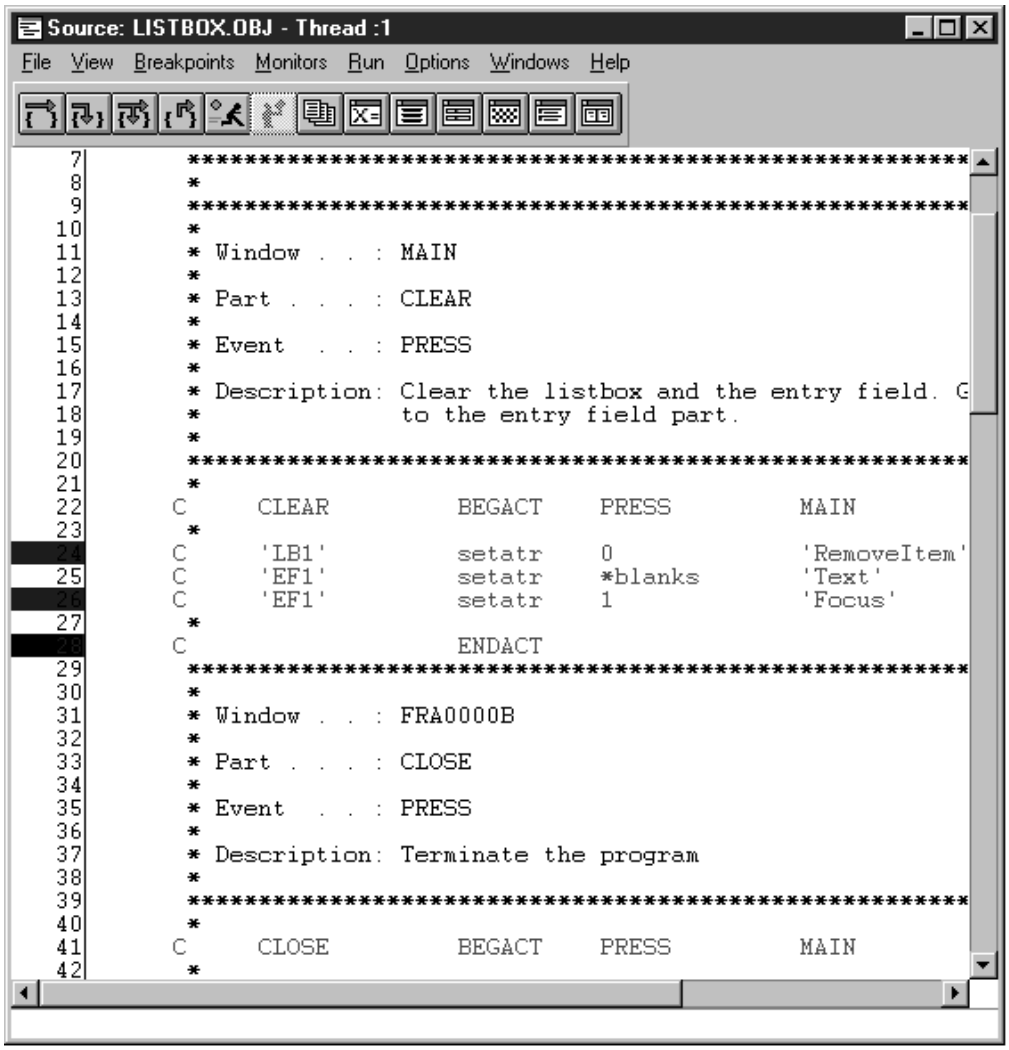


図 43. 複数のブレークポイントの設定:

ブレークポイント > リストを選択して、このウィンドウを表示します。それぞれのブレークポイントに対して次の情報も提供されます。

- 使用可能状態
- ブレークポイントのタイプ
- ブレークポイントの位置
- ブレークポイントが活動化される条件

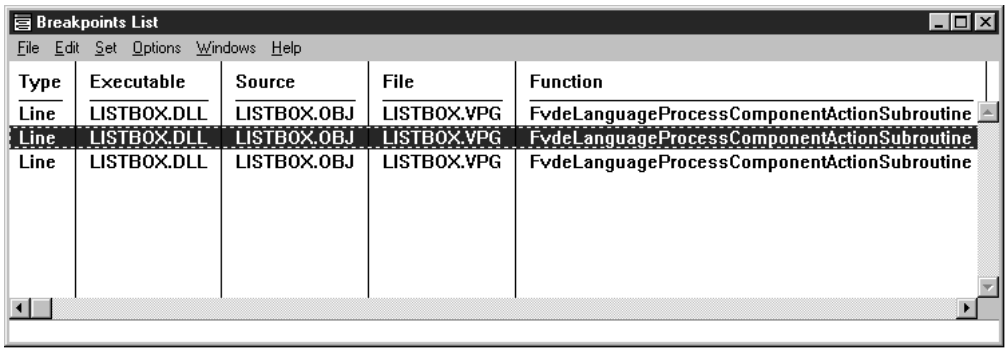


図 44. ブレークポイント・リスト・ウィンドウ :

ブレークポイントを伴う実行

R キーを押すと、プログラムが実行します。プログラムは、最初のブレークポイントを検出すると停止します。デバッガーはブレークポイントを検出すると停止して、下の 243 ページの図 45 に示すように行全体を強調表示します。これは実行中のプログラムが一時停止した位置を示します。

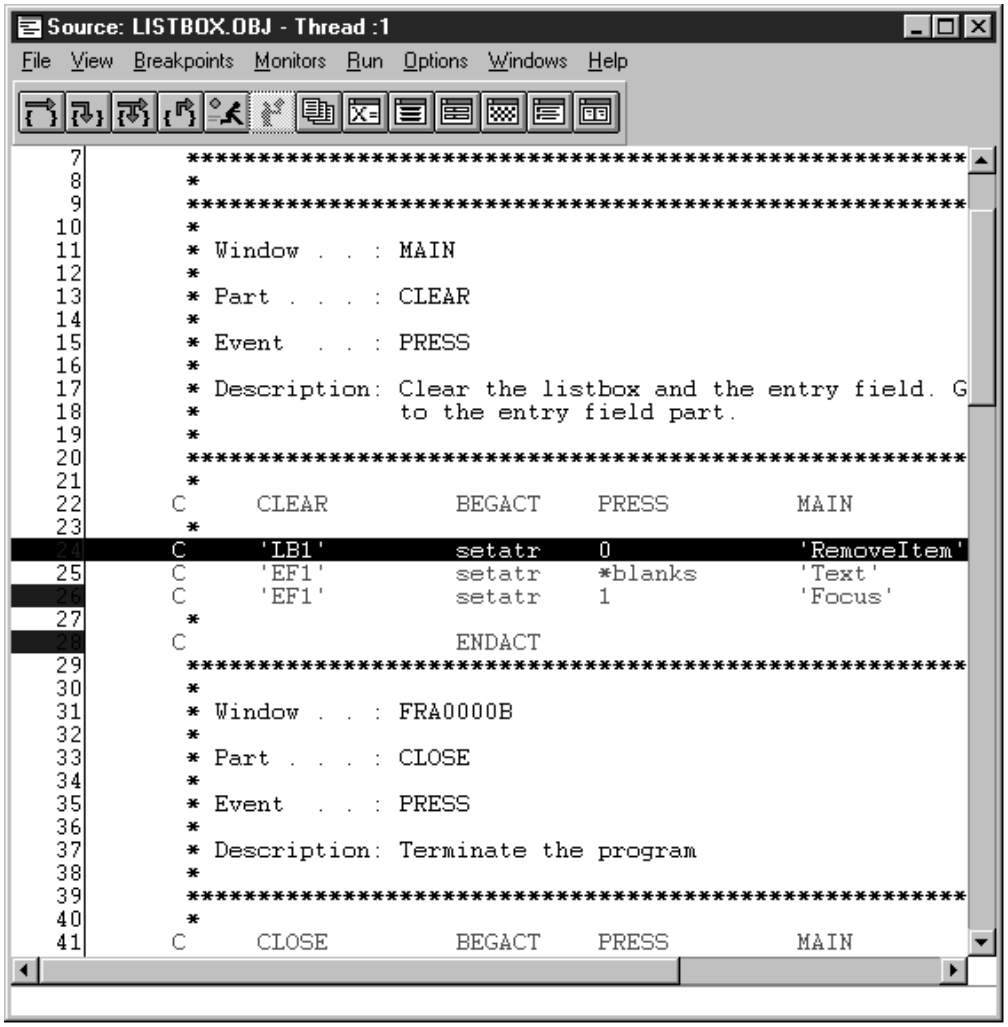


図 45. ブレークポイントを伴う実行 :

デバッグ機能を開始するためのマウスまたはキー・ボードの使用

ほとんどのデバッグ機能は、マウスまたはキーボードを使用して開始することができます。たとえば、ブレークポイントのロケーションを設定するには、行番号を左マウス・ボタンでダブルクリックします。同じことは、**ブレークポイント**・プルダウン・メニューから**行**を選択して行なうことができます。行を選択すると、行ブレークポイント・ウィンドウが表示されます。次に、行番号を入力しなければなりません。行番号を入力して Enter を押すと、選択した行が赤で強調表示されます。

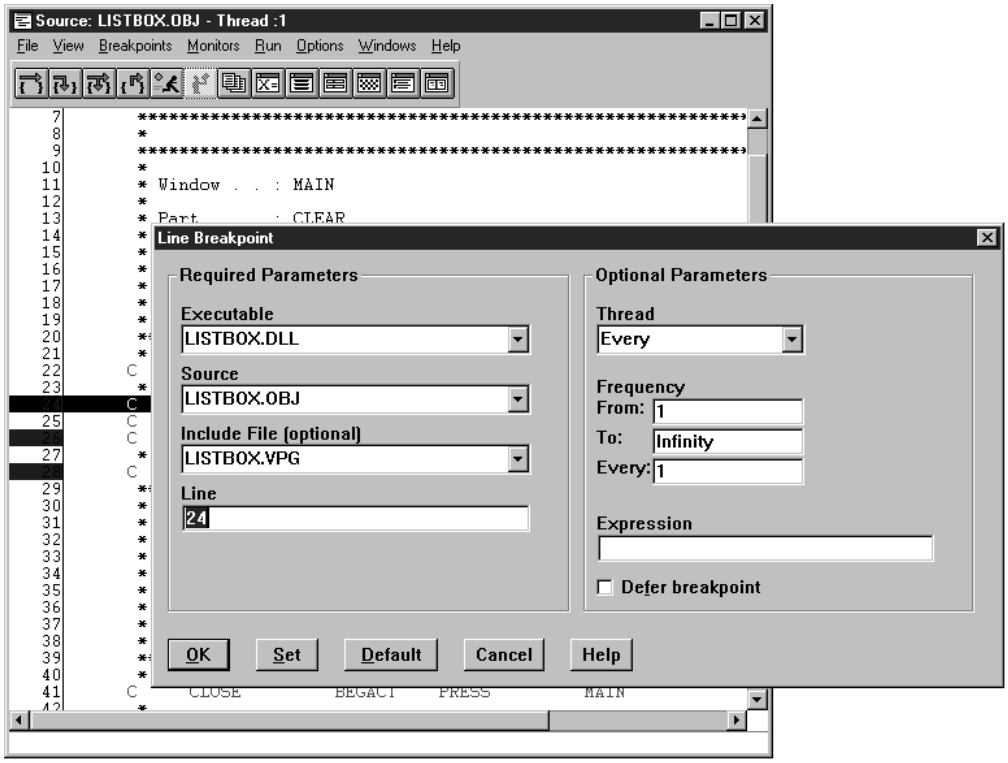


図 46. 行ブレークポイント・ウィンドウ :

プログラムの実行は別の方法でも再開することができます。次のいずれかを行なってください。

- 文字 R を押します。
- マウスを**実行**プルダウン・メニューに移動して、 **Run** を選択します。
- マウスを**ツールバー**の実行アイコンに移動して、**左マウス・ボタン** をシングルクリックします。

ツールバーからのオプションの選択

次の表に、ツールバーで使用可能なすべてのオプションをリストし、それぞれについて簡単に説明します。

図 47. ツールバーのオプション :

アイコン
: **機能**

ステップオーバー
: プログラムの現在行（強調表示されている）を実行しますが、呼び出された機能には入りません。

ステップイントゥ
プログラムの現在行（強調表示されている）を実行し、呼び出されたプログラムまたは機能に入ります。

ステップ・デバッグ
プログラムの現在行（強調表示されている）を実行します。デバッガーは、デバッグ情報を使用可能でない機能をステップオーバーし、デバッグ情報が使用可能な機能にステップイントゥします。

ステップ・リターン
現在の機能のリターン・ステートメント（を含む）までのコード行を自動的に実行します。

実行 現在行（強調表示されている）からプログラムの実行を開始します。

停止 プログラムの実行を停止します。

ビュー 次のビューに切り替えます。

式のモニター
モニター・ウィンドウに変数または式を表示します。

呼び出しスタック
スレッドの呼び出しスタックの活動機能を表示します。

レジスター
レジスター・ウィンドウのスレッド・レジスターを表示します。

ストレージ
ストレージ・ウィンドウにストレージの内容を表示します。

ブレークポイント
設定されているすべてのブレークポイントをリストします。

デバッグ・セッション制御
デバッグ・セッション制御ウィンドウを表示します。

変数、配列、および構造の表示および変更

デバッグ時に変数、配列、または有効なその他の VisualAge RPG 構造を表示するのは、通常使用される機能です。これを行なう一番容易な方法は、フィールドが使用可能な仕様にマウスを移動して、左マウス・ボタンをダブルクリックする方法です。たとえば、マウスを条件標識、演算項目 1、演算項目 2、結果フィールド、あるいは結果の標識に移動してダブルクリックします。これによって、その仕様の内容が表示されます。

注: 変数が EVAL 命令コードのオペランドの場合には、表示したい変数を選択してマウスで強調表示してからダブルクリックします。

表示または変更したいフィールドまたは構造がビューの中にある場合には、その内容を表示する一番簡単な方法はマウスを使用してダブルクリックする方法です。しかし、大きなプログラムを扱っていて、特定の変数または構造を容易に見つけることができない場合には、**モニター**・プルダウン・メニューから**式のモニター**を選びます (Ctrl-M を押しても同じことができることに注意してください)。

式のモニター・ウィンドウで、図 48 に示すように、表示したい式、フィールド、または構造を入力して Enter を押します。

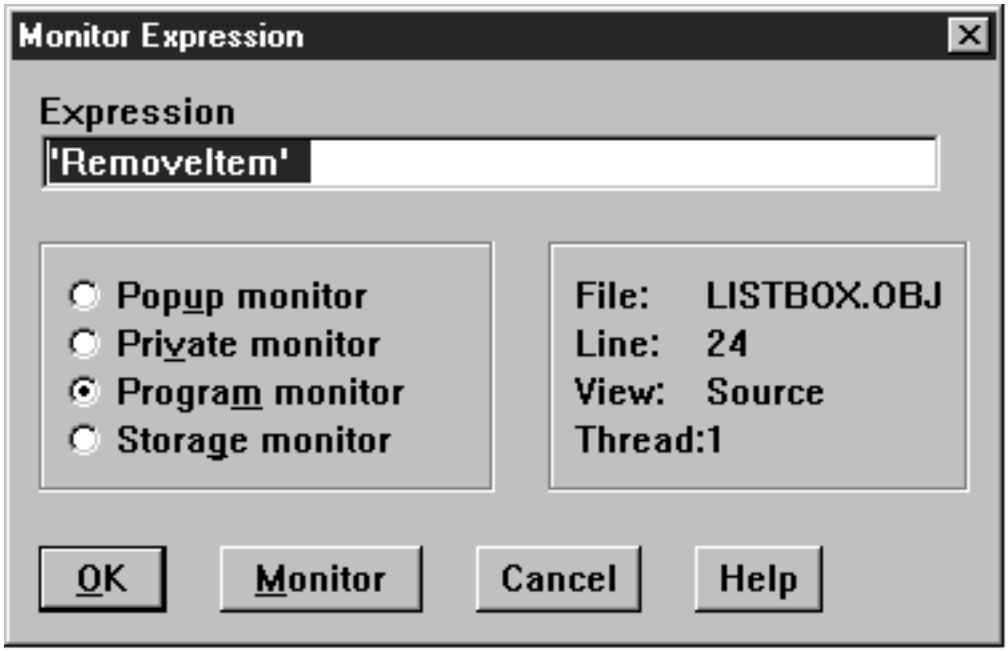


図 48. 式のモニター・ウィンドウ :

Enter を押すと、 247 ページの図 49 に示されるように、「プログラム・モニター」ウィンドウに VisualAge RPG フィールドまたは構造が表示されます。

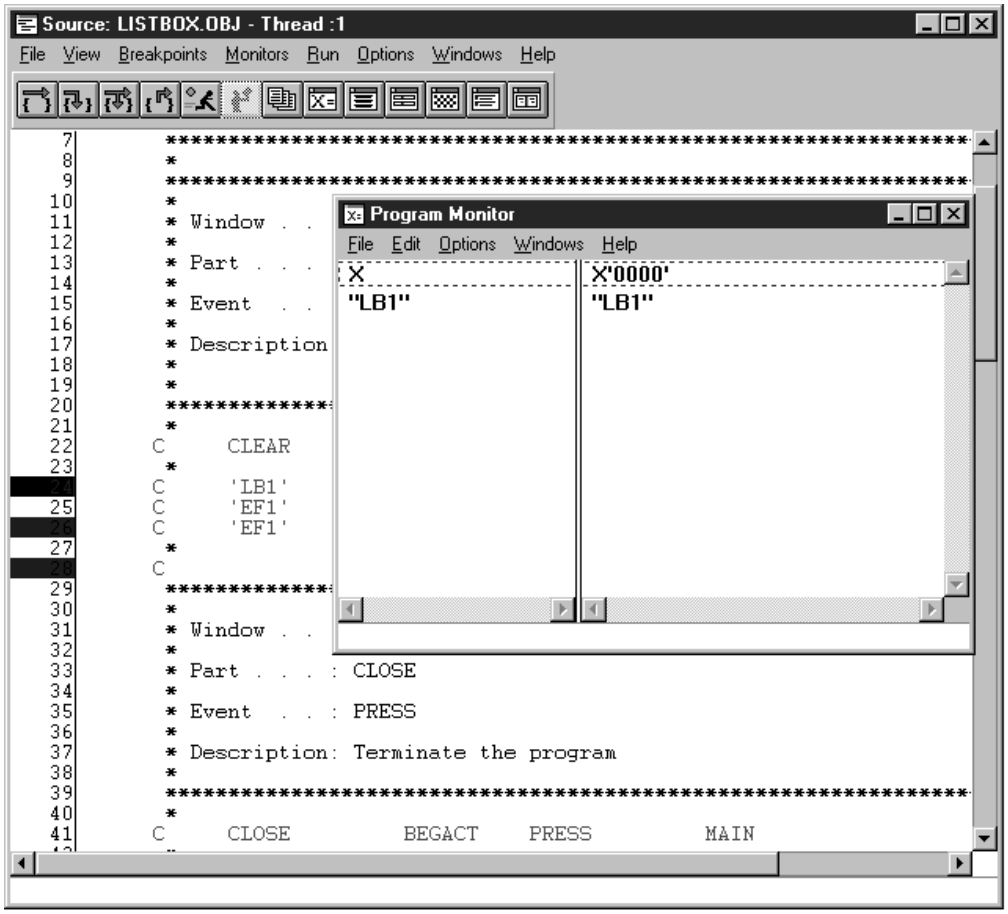


図 49. プログラム・モニター・ウィンドウ :

フィールドまたは構造の内容の変更

フィールドまたは VisualAge RPG 構造が表示されたら、その内容を変更することができます。このためには、プログラム・モニター・ウィンドウの値をダブルクリックして、新しい値を入力してから Enter を押します。

表現の変更

デバッガーによって、プログラム・モニターに表示された変数の表現を変更することができます。表現のタイプは 10 進数、16 進数、2 進数、またはストリング (その変数または構造 に有効な表現) とすることができます。このためには、プログラム・モニター・ウィンドウで変数を選択してください。その後で、**「編集 > 表現」**メニューから表現を選択します。これで、選択した表現に変数の内容が表示されます。

デフォルトの表現の変更

変数にはデフォルトの表現タイプがあります。たとえば、文字フィールドは 16 進数でなく文字としてプログラム・モニター・ウィンドウに表示されます。デバッガーによって、この動作を変更することができます。それぞれのデータ・タイプに、デフォルトの表現を設定するオプションがあります。フィールドのデフォルトの表現を変更するためには、**オプション > デバッガーの設定 > デフォルトのデータ表現 > システム**を選択します。デフォルトのデータ表現ウィンドウが表示されます。

ポインターおよびストレージの表示

VisualAge RPG によってサポートされるデータ・タイプの 1 つはポインターです。図 50 ストレージ・ウィンドウを使用してポインター値を表示する例を示します。「ストレージ」ウィンドウを表示するには、**モニター > ストレージ**を選択します。

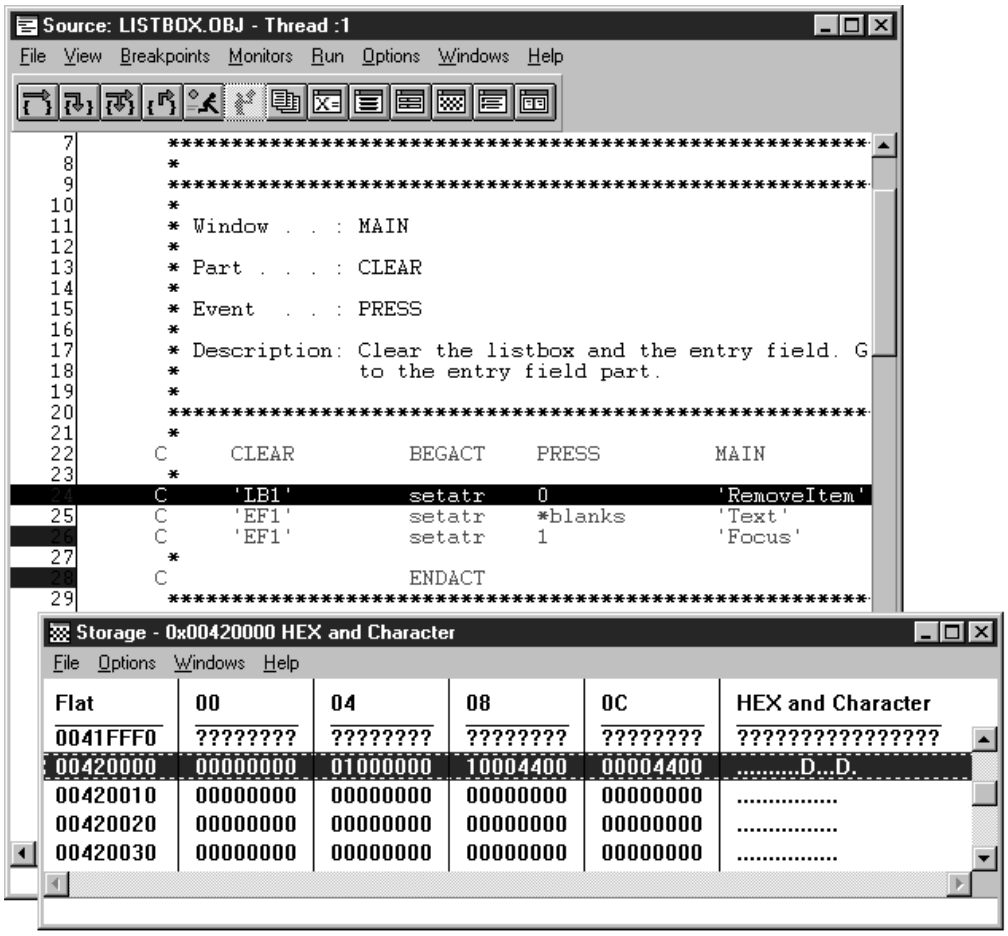


図 50. ポインター値の表示:

デバッガー・ビューの変更

この項の例のほとんどは、VisualAge RPG ソース・ビューを示しています。デバッガーは、他のビューも表示します。ディスアセンブリーおよび混合ビューです。ビューを変更するには、図 51 に示すように、**ビュー**・メニューからオプションを選択します。

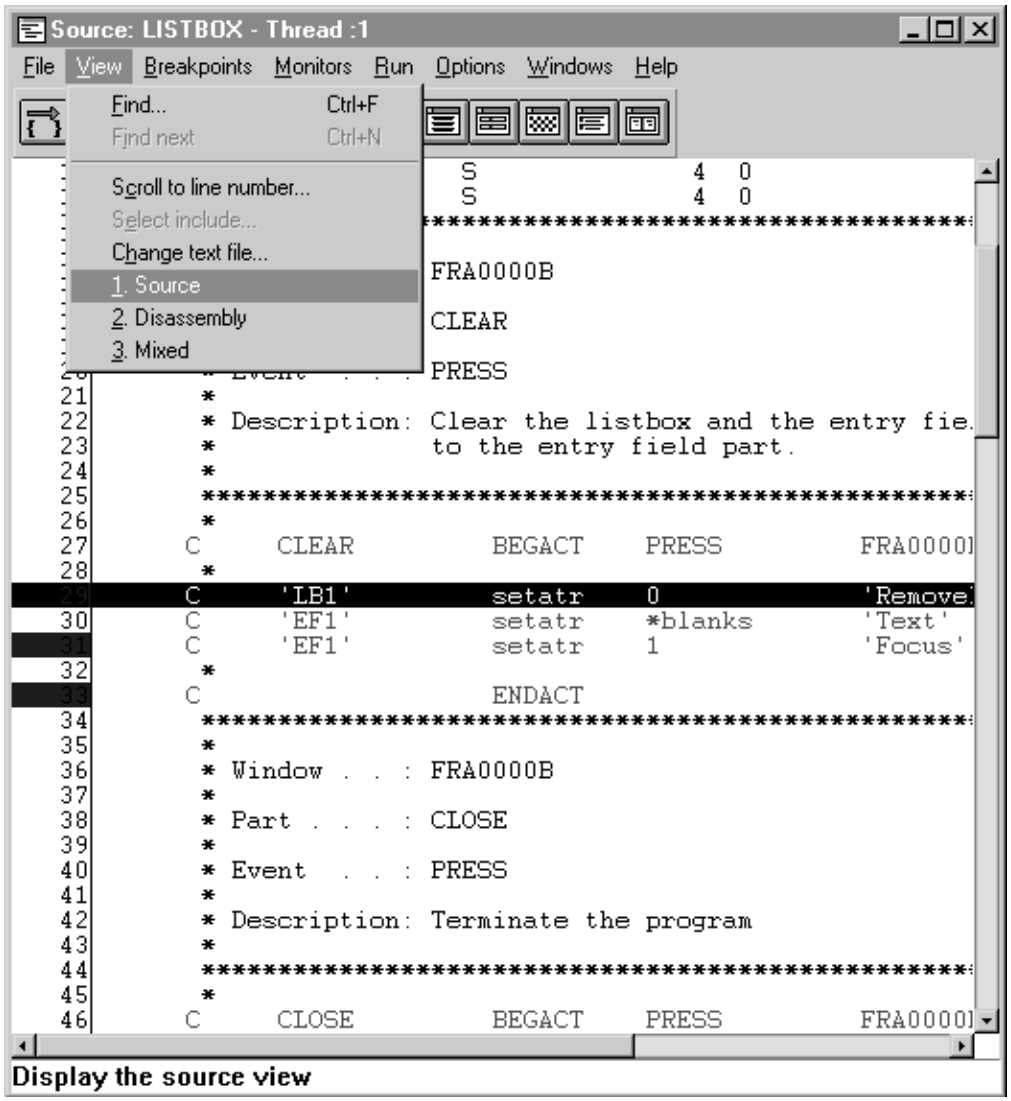


図 51. デバッグ・ビューの変更 :

フォントの設定

デバッガーでは、デバッグ・セッションをカスタマイズできる多くのオプションを使用することができます。たとえば、フォントを設定することができます。図 52 フォント・ウィンドウを表示します。フォント・ウィンドウを表示するには、**オプション > ウィンドウの設定 > フォント**を選択します。フォント・ウィンドウで、必要なフォント、スタイル、およびサイズを選んでから OK を選択します。デバッグ・セッションのフォント変更が表示されます。

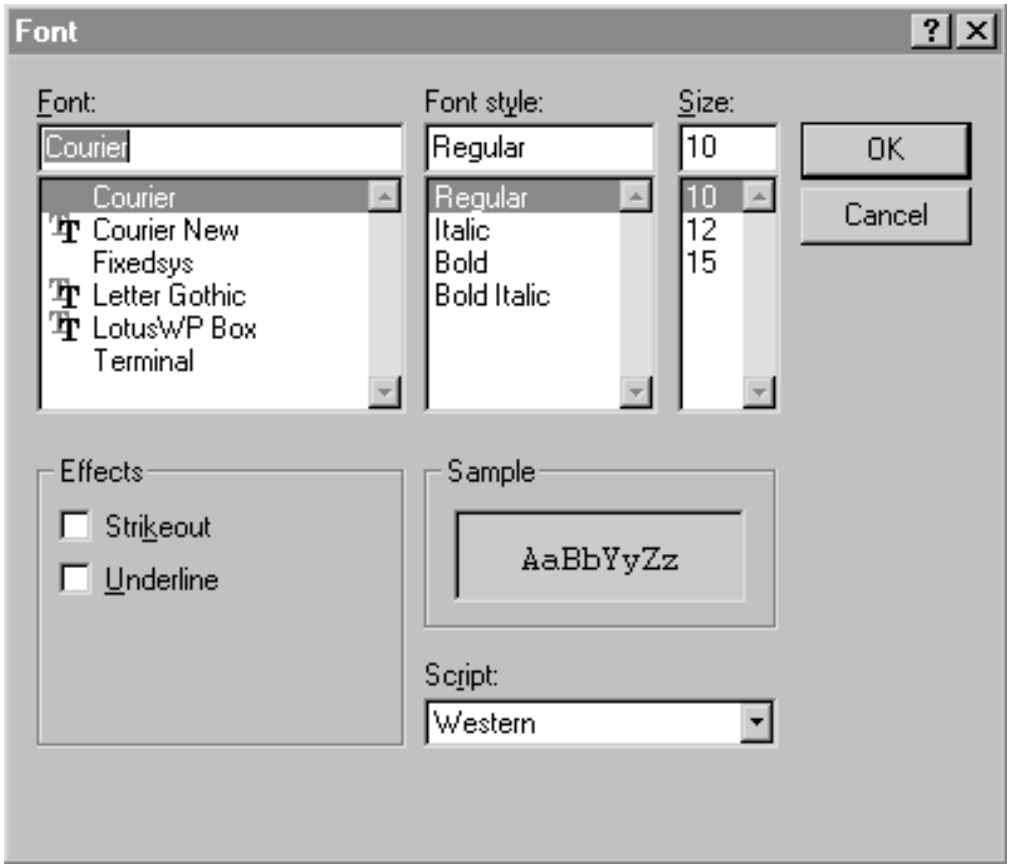


図 52. フォントの設定：

第 11 章 出力の編集

コンパイラーは、入力フィールドおよび静的テキスト・パーツに表示される時に、データの形式設定方法を決定する編集機能をサポートします。出力を編集するために、これらのパーツのプロパティー・ノートブックで**編集コード** または**編集語**を設定することができます。

編集コードではユーザーが事前定義形式に従ってデータを形式設定できますが、編集語ではユーザーが独自の形式を定義します。パーツには編集コードまたは編集語のいずれかを指定できますが、両方を指定することはできません。

編集コードおよび編集語は数字入力フィールドおよび数字静的テキスト・パーツのみに指定することができます。

定様式入力フィールドからユーザー・プログラムにデータが読み込まれる場合には、コンパイラーはデータをユーザー・プログラムに戻す前にすべての編集文字を除去します。

注: アプリケーションの制御仕様の編集コード項目は無視されます。これらは編集コードの出力に影響を与えません。

編集コード

データを事前定義形式に形式設定するためにいくつかの編集コードがサポートされています。これらの形式は正しい千単位の区切り文字と 10 進数区切り記号を挿入し、固定または浮動のマイナス符号あるいは CR (クレジット) 記号を提供して負の数の表示方法を決定します。

ユーザーは任意選択で、編集コードによりアスタリスク保護または浮動通貨記号を指定することができます。アスタリスク保護を指定すると、抑制されるそれぞれのゼロでアスタリスクが表示されます。浮動通貨記号を指定すると、最初の有効数字の左側に記号が表示されます。ゼロ残高を抑制する編集コードを使用すると、ゼロ残高ではこの記号は表示されません。

千単位の区切り文字と 10 進数区切り記号および通貨記号に実際に使用される文字は、アプリケーションの実行時にオペレーティング・システムによって決定されます。

次の表に、サポートされる編集コードとそれらが提供する編集の要約を示し、いくつかの例を提示します。

注: コンパイラーはユーザー定義の編集コードをサポートしません。ユーザー定義の編集コードは AS/400 システムで定義され保管されて、VisualAge RPG プログラムでは使用できません。

表 11. VisualAge RPG 編集コード

編集コード	コンマ	小数点	負バランスの符号	正の数の例	負の数の例	ゼロ・バランス
なし	いいえ	はい	はい	0123456789	0123456789-	
1	はい	はい	符号なし	124,567.89	124,567.89	.00
2	はい	はい	符号なし	124,567.89	124,567.89	
3	いいえ	はい	符号なし	124567.89	124567.89	.00
4	いいえ	はい	符号なし	124567.89	124567.89	
A	はい	はい	CR	124,567.89	124,567.89CR	.00
B	はい	はい	CR	124,567.89	124,567.89CR	
C	いいえ	はい	CR	124567.89	124567.89CR	.00
D	いいえ	はい	CR	124567.89	124567.89CR	
J	はい	はい	- (マイナス)	124,567.89	124,567.89-	.00
K	はい	はい	- (マイナス)	124,567.89	124,567.89-	
L	いいえ	はい	- (マイナス)	124567.89	124567.89-	.00
M	いいえ	はい	- (マイナス)	124567.89	124567.89-	
N	はい	はい	- (浮動マイナス)	124,567.89	-124,567.89	.00
O	はい	はい	- (浮動マイナス)	124,567.89	-124,567.89	
P	いいえ	はい	- (浮動マイナス)	124567.89	-124567.89	.00
Q	いいえ	はい	- (浮動マイナス)	124567.89	-124567.89	
Y (2.)				1984-12-25		
Z (3.)	いいえ	いいえ	符号なし	1234567	1234567	

注:

1. すべての編集コードは先行ゼロを抑制します。
2. Y 編集コードは日付フィールドに使用されます。日付フィールドは数値フィールドとして定義しなければなりません。この編集コードの出力は *nnnn-nn-nn* の形式です。この形式を変更することはできません。日付区切り記号は、アプリケーションの実行時にオペレーティング・システムによって決定されます。
3. Z 編集コードは + または – 符号を除去します。

編集語

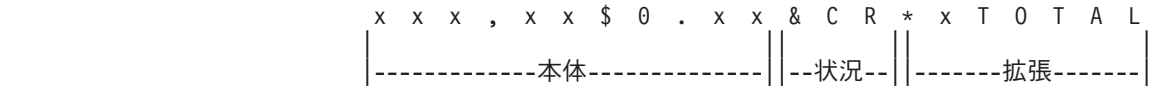
提供された編集コードがどれもユーザーの編集要件に適合しない場合には、編集語を使用することができます。編集語は、パーツに入れる前にデータに適用されるテンプレートです。編集語を使用して、以下を指定することができます。

- 先行ゼロの抑制
- 先行アスタリスク
- 固定/浮動通貨記号
- 千単位の区切り文字と 10 進数区切り記号の位置

注: 編集語を使用する場合には、通貨、10 進数、および千単位の記号を正しく指定していることを確認してください。記号が編集語と一致しない場合には、正しく形式設定されていない出力が得られますが、実行時エラーにはなりません。これらの記号は、アプリケーションの実行時にオペレーティング・システムの実行時の値と置き換えられます。

編集語の部分

編集語は、本体、状況、および拡張から構成されます。これらのパーツを次の例で示します。




```
ここで BLANK              = x
      CURRENCY SYMBOL     = $
      THOUSAND SEPARATOR = ,
      DECIMAL SYMBOL      = .
```

編集語の本体

本体は、データ・フィールドからパーツに転送される数字のためのスペースです。これは編集語の最左端位置から始まります。これには、ブランクの数に 1 個のゼロまたはアスタリスクを加えたものが入り、その合計は編集するデータ・フィールドの数字の数に等しくなります。

次の文字は、編集語の本体で使用すると特別の意味を持ちます。

ブランク
ブランクは、データ・フィールドの対応する位置からの数字と置き換えられます。

アンパーサンド
アンパーサンドは、編集画面でブランクになります。

ゼロ ゼロはゼロ抑制を停止します。ゼロはそれ自体が数字部分です。データ・フィールドのゼロ抑制停止文字の右側のゼロは表示されます。抑制されるそれぞれのゼロはブランクと置き換えられます。

アスタリスク
ゼロの代わりのアスタリスクは、ゼロ抑制停止文字として使用することができます。これはアスタリスク保護と呼ばれ、抑制されるそれぞれのゼロがアスタリスクと置き換えられます。ゼロ抑制停止文字の右側のアスタリスクまたはゼロは定数で、そのまま表示されます。

通貨記号

ゼロ抑制停止文字のすぐ左に通貨記号をコーディングすると、最初の有効数字の左側の位置に通貨記号が挿入されます。これは、そのように使われる場合には、浮動通貨記号と呼ばれます。通貨記号を編集語の最左端位置にコーディングした場合には、それは固定で、同じ場所に表示されます。これは固定通貨記号と呼ばれます。

千単位の区切り文字と10 進数区切り記号

千単位の区切り文字と 10 進数区切り記号は、編集語でコーディングしたものと同じ相対位置に表示されます。他のすべての文字は、編集語の有効数字の右側にあれば表示されます。それらが編集語の高位有効数字の左側にある場合には、ブランクにされるか、またはアスタリスク保護が使用されている場合にはアスタリスクと置き換えられます。

編集語の状況

状況位置にはデータの符号が表示されます。状況は本体の右側の、クレジット (CR) またはマイナス (-) 符号のいずれかに続きます。これらの 2 つの符号は、フィールドが負の場合に表示されるだけです。アンパーサンド (&) はブランクの表示に使用されます。

編集語の拡張

拡張位置は編集操作では変更されません。拡張位置は、状況の右側 (または、状況が指定されていない場合には、本体) の右側の最初の位置から始まります。拡張部分にブランクを入れることはできません。 ブランクが必要な場合には、編集語でアンパーサンドを使用してください。

第 12 章 ピクチャー、サウンド、およびビデオ・ファイルの使用

アニメーション・コントロール、キャンバス、グラフィック・プッシュボタン、イメージ、メディア、およびメニュー項目パーツによって、**FileName** 属性に有効なイメージ名を指定して、ウィンドウにイメージを表示することができます。

有効な **Windows** イメージ形式には、次が含まれます。

OS/2 および Windows ビットマップ
ファイル拡張子 .BMP、.VGA、.BGA、.RLE、.DIB、.RL4、および .RL8 は、OS/2 または Windows ビットマップとして認識されます。

アイコン・フォーマット
.ICO ファイル拡張子は、アイコン・ファイルとして認識されます。

CompuServe グラフィック交換形式
.GIF ファイル拡張子は GIF ファイルとして認識されます。

ZSoft PC Paintbrush イメージ・ファイル形式
.PCX ファイル拡張子は Paintbrush ファイルとして認識されます。

Microsoft/Aldus タグ付きイメージ・ファイル形式
.TIF および .TIFF ファイル拡張子は TIFF ファイルとして認識されます。

Truevision Targa/Vista ビットマップ
ファイル拡張子 .TGA、.VST、および .AFI は Targa/Vista ファイルとして認識されます。このクラスは 8 ビット/プレーンと 24 ビット/プレーンのイメージしかサポートしていません。

Amiga IFF/ILBM インターリーブド・ビットマップ形式
ファイル拡張子 .IFF および .LBM はインターリーブ・ビットマップ・ファイルとして認識されます。

X Windows ビットマップ
.XBM ファイル拡張子は X ビットマップ・ファイルとして認識されます。このクラスは X10 および X11 1bpp ビットマップをサポートしています。内部にテキスト・ストリングをもつ .XBM ファイルのなかにはスプライトまたはアイコンとして見えるものがあり、サポートされていません。

IBM プリンター・ページ・セグメント
ファイル拡張子 .PSE、.PSEG、.PSEG38PP、および .PSEG3820 は PSEG ファイルとして認識されます。 PSEG ファイルは BookMaster 文書にイメージ・データを組み込むために使用されます。 PSEG ファイルには、1 ビット/プレーン（常に用紙上のインク、すなわち、白地に黒を表す）しか入りません。

有効な **Java** イメージ形式には、次が含まれます。
- CompuServe グラフィック交換形式 (GIF)
- Joint Photographic Experts Group 形式 (JPG, JPEG)

さらに、WAV、.MID、.MPG、.MOV、.DAT、および .AVI ファイルをサポートするメディア・パーツを使用してサウンドおよびビデオを追加することができます。

ピクチャー、サウンド、またはビデオを含む VisualAge RPG アプリケーションの開発時には、パーツの **FileName** 属性のハードコーディングは避けてください。ユーザーは、アプリケーションを開発時のものとは異なるディレクトリーにインストールすることになります。

これらのファイルが実行時に見つかるようにするには、**現行ディレクトリー**・ストリング (.\) を使用します。すなわち、ドットと円記号にファイル名を続けて指定します。実行時に、ファイルはアプリケーションが実行された現行ディレクトリー内で見つかります。

たとえば、グラフィック・プッシュボタンのプロパティー・ノートブックで、EXIT.ICO という名前のアイコンが実行時に現行ディレクトリー内で見つかるように、ファイル名として次のように指定します。

```
.\\EXIT.ICO
```

注: Windows で実行されるアプリケーションの場合は、現行ディレクトリーは AUTOEXEC.BAT ファイル中に指定しなければなりません。

ビルド時に、ピクチャー・ファイルをビルド時ディレクトリーにコピーして、これらのファイルにアクセスしなければなりません。

アプリケーションを配布用にパッケージする前に、関連するすべてのピクチャーおよびサウンド・ファイルをプロジェクトの適切な実行時サブディレクトリー (RT_JAVA または RT_WIN32) にコピーしてください。パッケージされてユーザーに配布されるのはこのディレクトリーだからです。アプリケーションをパッケージ化する際の指示については、 437 ページの『第 5 部 アプリケーションの配布』を参照してください。

Windows 用アイコンの作成

VisualAge for C++ for Windows がある場合には、Resource Workshop ユーティリティーを使用してWindows アイコンを作成することができます。

OS/2 アイコンの Windows 形式への変換

VisualAge RPG には、OS/2 アイコンとビットマップを Windows バージョンに変換するユーティリティー・プログラムが入っています。このユーティリティーとそのパラメーターの詳細については、DOS プロンプトに進み、IBMPCNV -H. と入力してください。

第 13 章 IPF を用いたオンライン・ヘルプ作成のヒント

情報表示機能 (IPF) によって、ユーザー・アプリケーション用のオンライン・ヘルプ・ファイルを作成して管理することができます。また、IPF を使用して、チュートリアルおよびオンライン文書を作成することもできます。 VisualAge RPG では、作成するオンライン・ヘルプは Windows 固有のヘルプとなります。

この項では、IPF を紹介して、ユーザー・アプリケーション用のオンライン・ヘルプを作成するためのヒントをいくつか提供します。 IPF を使用する場合の詳細については、*Information Presentation Facility Guide and Reference*(オンラインで使用可能) を参照してください。また、*IPF 制約事項* というタイトルのオンライン文書も参照してください。この文書には、Windows 環境で限定されている IPF タグのサブセットについての詳細が記載されています。

同様に AS/400 システムからの UIM ヘルプ・ソースを再使用することができます。231 ページの『UIM ヘルプの再使用』を参照してください。

オンライン・ヘルプの作成

ユーザー・アプリケーション内のパーツ用にオンライン・ヘルプを追加するためには、次を実行します。

1. パーツのポップアップ・メニューを表示します。
2. **「ヘルプ・テキスト」**を選択します。編集セッションがオープンされます。
3. パーツについての項目ヘルプ・テキストを入力します。
4. IPF タグ言語を使用してヘルプ・テキストにタグ付けます。
5. **ファイル**・メニューから**保管**を選択することによって、ヘルプを保管します。

IPF の使用

VisualAge RPG アプリケーション・ヘルプ・モジュールのソースは IPF 形式になっています。IPF によって、オンライン情報を作成したり、それを画面上に表示する方法を指定したり、情報の各種のパーツを結合したり、さらにユーザーによって要求されることのあるヘルプ情報を提供することができます。IPF の機能には次が含まれています。

- テキストの書式を整え、情報単位を結合する方法を提供し、さらにウィンドウをカスタマイズするタグ付け言語
- オンライン文書およびヘルプ・ウィンドウを作成するコンパイラー
- 様式設定された文書を表示する表示プログラム

その他の言語のヘルプのサポート

任意のテキスト・エディターを使用して .VPF ファイルをコピーして手操作で編集することができます。しかし、それを実行しないで、次のいずれか一方を実行してください。

- `res=` テキストの後に現れる番号の変更または除去。その番号はリソース ID であり、パーツ用のヘルプの作成時に GUI Designer によって生成されます。リソー

ス ID は、該当のヘルプ・テキストを見つけるために使用されます。リソース ID を削除または変更した場合には、それに属しているヘルプ・テキストは見つからないことになります。
- 見出し情報の除去。見出し情報は変換済みテキストで置き換えることができます。

オンライン・ヘルプへのグラフィックスの追加

必要に応じて、:artwork. タグを使用して、ソース・ファイルの内部にグラフィックスを組み込みます。このグラフィックスはビットマップ形式 (.BMP ファイル) になっていなければなりません。

提供するヘルプのタイプの決定

ユーザーは、VisualAge RPG アプリケーション内で 3 つの異なる方法でヘルプにアクセスすることができます:

ヘルプ情報

選択項目またはパーツの現在の文脈に適合しているヘルプ情報。ユーザーは、選択項目またはパーツにフォーカスがある時に F1 キーを押すことによってこのヘルプにアクセスします。このヘルプはパーツのポップアップ・メニューから提供することができます。

ウィンドウ・レベルのヘルプ

ウィンドウの目的に関する情報。ユーザーは、「ヘルプ」プッシュボタンを押すことによってこのヘルプにアクセスします。このヘルプは、**「ヘルプ」**プッシュボタンを作成して、関連した ヘルプ情報を追加することによって提供することができます。

タスク・ヘルプ

ユーザーがアプリケーションで実行できるタスクについての情報。ユーザーは、ヘルプ・パネル内から他のヘルプ情報にハイパーテキスト・リンクすることによってこのヘルプにアクセスします。この情報は、オンライン情報を作成して、ウィンドウ・レベルまたはヘルプ情報からそこへのハイパーテキスト・リンクを作成することによってこの情報を提供することができます。ハイパーテキスト・リンクによって、ユーザーは、あるヘルプ・パネルから別のヘルプ・パネルにジャンプするか、あるいはヘルプ・パネル内で選択されたテキストから関連情報にジャンプすることができます。

ツール・ヒント・ヘルプ

使用可能なツールに関する吹き出しヘルプ・スタイルの情報。このヘルプを作成するためには、パーツのプロパティー・ノートブックの「一般」ページに進んで、入力フィールドにツールの説明 (15 文字まで) を入力します。また、メッセージ ID (たとえば *MSG0001) を使用して、ヘルプ・テキストを指定することもできます。

ヘルプ情報の追加

パーツ用のヘルプ情報を追加するためには、そのパーツのポップアップ・メニューから**「ヘルプ・テキスト」**を選択します。これで、図 53に示されているような情報が入っている編集セッションが開始されます。

```
:h1 res=01.PSB0000C:p.Help
```

図 53. オンライン・ヘルプを追加するための編集セッション

`:h1 res=01.` は、自動的に生成されるリソース ID です。このテキストは編集**しないでください**。このタグの後にヘルプ・パネルの目的を識別する見出しを入力して、 `:p.` タグの後にヘルプ・テキストを入力します。

「ヘルプ」プッシュボタンの作成

「ヘルプ」プッシュボタンを作成するには、右マウス・ボタンでパーツ・パレットからプッシュボタンを選択し、マウス・ポインターを設計ウィンドウに移動してから、再び右クリックします。プッシュボタンのポップアップ・メニューから**「ヘルプ・テキスト」**を選択して、ヘルプ情報を編集します。そのプッシュボタンに **Help Enable** 属性をセットして、 **Label** 属性を語 ヘルプにセットします。

ハイパーテキスト・リンクの作成

ユーザーが適切な情報を迅速かつ容易に見つけることができるように、ヘルプ情報の関連のある断片をリンクするためには、ヘルプ・テキストで link タグを使用します。ヘルプ・パネルへのリンクは resid または refid を使用して作成することができます。

`id=` を用いて定義されたヘルプ・パネルにリンクする場合は、次の通りです。

```
:link reftype=hd refid=search.検索ウィンドウ:elink.
```

`res=` を用いて定義されたヘルプ・パネルにリンクする場合は、次の通りです。

```
:link reftype=hd res=15433.検索プッシュボタン:elink.
```

第 14 章 Windows ヘルプの作成および使用のヒント

VisualAge RPG の機能の 1 つは、アプリケーション用のカーソル移動に影響されるヘルプを作成する機能です。設計ウィンドウのパーツを右クリックして**「ヘルプ・テキスト」**を選択することによって、ヘルプを作成します。これによりエディターが開始されます。情報表示機能 (IPF) タグ言語を使用して、ヘルプ・テキストを作成します。作成プロセス時に、ヘルプ・ソースがコンパイルされて **HLP** ファイルが作成されます。IPF タグ言語は、結果的に特徴的な OS/2 の外観をもつヘルプ・ファイルになります。このセクションでは、アプリケーション用の真の Windows ヘルプを作成する方法を説明します。

必要なもの

Windows ヘルプ・ファイルを作成するためには、次の 2 つのツールが必要です:
- ファイルを**リッチ・テキスト・フォーマット (RTF)** 形式で保管できるワード・プロセッサー
- Windows ヘルプ・コンパイラー

ヘルプ・コンパイラーは、RTF で保管されたヘルプ・ソース・ファイルを入力として使用します。いくつかのワード・プロセッサー (Lotus WordPro、 Microsoft Word、およびWordPerfect など) が RTF 形式でファイルを保管できます。Windows WordPad エディターはRTF 形式でファイルを保管できることに注意してください。ただし、この RTF 形式はヘルプ・ファイルの作成に使用することはできません。これには、ヘルプ・ファイルを作成するためにヘルプ・コンパイラーが必要とするだけのフォーマット・オプションが保存されていません。

Help Compiler Workshop は Microsoft から提供されたツールで、ヘルプ・コンパイラーと同様に、ヘルプ・ファイルを管理するための IDE から成っています。これは次の URL の Microsoft ヘルプ・コンパイラー FTP からダウンロードできます:

```
ftp://ftp.microsoft.com/softlib/mslfiles/hcwsetup.exe
```

完全なヘルプ・オーサリング環境を提供する多くのツールが、営利的およびシェアウェアとして市場で利用できます。さらに、ヘルプ・コンパイラー・ワークショップを使用する方法を記述したいくつかの書物が存在します。これらの書物の多くには、*Microsoft Windows 95 Help Authoring Kit ISBN1-55615-892-0* などのヘルプ・コンパイラー・ワークショップが組み込まれた CD-ROM が付いています。

Windows ヘルプを作成するためのステップ

アプリケーションで Windows ヘルプを使用するために従う基本的なステップは次の通りです:
1. ヘルプをもつ各パーツごとに**リソース ID** を確立する。
2. ヘルプ・テキストを作成する。
3. ヘルプ・**プロジェクト・ファイル**を作成する。
4. ヘルプ・ファイルをコンパイルする。

リソース ID の確立

入力フィールド、プッシュボタン、またはウィンドウなどのすべてのパーツは、通常**リソース ID** と呼ばれる ID が割り当てられています。 VisualAge RPG は、ユーザーに代わってリソース ID を割り当て、それらは変更できません。パーツのリソース ID を表示するには、設計ウィンドウでそのパーツを右クリックしてください。そのプロパティー・ノートブックを表示するには、**プロパティー**を選択してください。リソース ID は「一般」ページの最上部にある数値です。次の例では、パーツ ID の横の数値 **12** です:

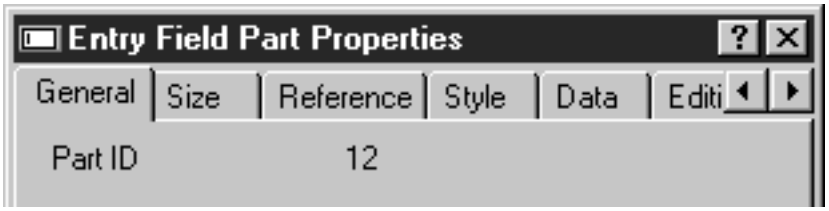


図 54. リソース ID の表示

作成プロセス時に、VisualAge RPG は、ユーザーがパーツのポップアップ・メニューから**「ヘルプ・テキスト」**メニュー項目を使用するためのヘルプを作成した各パーツごとに、リソース ID テーブル項目を作成します。 Windows ヘルプ・エンジンはこのテーブルを使用してパーツのリソース ID を判別するので、正しいヘルプを表示することができます。Windows ヘルプをもつパーツについて、この方法で各パーツごとにヘルプ・テキストを作成しなければなりません。現在は、VisualAge RPG はユーザーに代わって自動的にこのテーブル項目を作成しません。このプロセスに従った場合には、ヘルプは表示されず、エラー・メッセージも生成されません。

ヘルプ・テキストの作成

ヘルプを作成する前に、いくつかの Windows ヘルプ用語を知っておく必要があります。 Windows ヘルプ・ファイル (HLP 拡張子) を作成するには、次のファイルが必要です:

トピック・ファイル
このファイルにはユーザーのヘルプ・テキストが入っています。ヘルプ・プロジェクトは 1 つまたは複数のトピック・ファイルで構成することができます。トピック・ファイルには、1 つまたは複数の**トピック**が入っています。トピック・ファイルはワード・プロセッサーで作成し、それを RTF 形式 (RTF 拡張子) で保管します。

プロジェクト・ファイル
プロジェクト・ファイルにはヘルプ・ファイルの情報が入っています。これには、トピック・ファイルが組み込まれることになるものが入っています。プロジェクト・ファイルはヘルプ・ワークショップ IDE によって保守されます。一般的に、これを直接変更することはありません。

目次ファイル

ヘルプ・ファイルを表示する時に**目次**タブを付けたい場合は、目次ファイルが必要です。目次ファイルもヘルプ・ワークショップ IDE によって作成され、保守されます。

次の例は、1 つのトピックをもつトピック・ファイルを作成する場合の基本的なステップを概説したものです。これは入力フィールド・パーツのヘルプ・テキストをもっています。トピック・ファイルを作成するために Lotus WordPro を使用します。ワード・プロセッサーで新規の文書がオープンされたら、ページの最上部に入力フィールドのヘルプ などのタイトルを入力してください。タイトルに続いて、ヘルプ・テキストの本文を入力します。

各トピックはトピック ID をもっていなければなりません。トピック ID は、**#** 記号をもつ脚注です。

必要な脚注を WordPro で作成するステップは次の通りです。ワード・プロセッサーで、# 記号をもつ脚注を作成するためのステップに従ってください:

1. カーソルをトピック見出しの直前に位置付けます。
2. **脚注の作成...**を選択します。
3. 脚注ダイアログで **OK** を押します。
4. カーソルが脚注セクションの文書の最下部に位置付けられます。
5. トピック ID を入力します: HelpForEF。
6. カーソルをトピック ID の先頭に位置付けます。
7. マウスを右クリックして、ポップアップ・メニューから**テキスト・プロパティー**を選択します。
8. 「プロパティー」ダイアログで、**中黒および番号**タブを選択します。
9. **ページで編集**チェック・ボックスにチェックします。
10. トピック IDの前に **#** 文字を入力します。
11. 「テキスト・プロパティー」ダイアログをクローズします。

文書は次のようになるはずです。行の後のデータは脚注です:

```
#入力フィールドのヘルプ

This is help for the entry field part.

more stuff ...

_____________________________________

#HelpForEF
```

1 つのトピック・ファイルの中に複数のトピックを入れることができます。各トピックは新しいページから始まらなければなりません。ヘルプ・テキストの入力を完了したら、トピック・ファイルを RTF 形式で保管してください。

ヘルプ・プロジェクト・ファイルの作成

以下は、最小のプロジェクト・ファイルを作成するための基本ステップです。
Microsoft Help Workshop を始動して、次のことを実行してください:

1. **ファイル - 新規**を選択し、次に**新規プロジェクト**を選択することによって、新らしいプロジェクト・ファイルを作成します。新しいプロジェクトが作成されます。
2. **「ファイル」**プッシュボタンを押します。
3. 「トピック・ファイル」ダイアログで、**追加...**を押して、作成したばかりのトピック・ファイルを追加します。**OK** を押して、「トピック・ファイル」ダイアログをクローズします。
4. **「ウィンドウ」**プッシュボタンを押します。
5. 「ウィンドウ・プロパティー」ダイアログで、**追加**を押して、「新規ウィンドウ・タイプの追加」ダイアログを表示します。
6. **メイン**という名前のウィンドウを作成し、すべてのダイアログをクローズしてから、Help Workshop に戻ります。
7. トピック・ファイルの中のトピック ID (HelpForEF) を、入力フィールド・パーツ (12) のリソース ID にマップします。
8. **「マップ」**プッシュボタンを押します。
9. 「マップ」ダイアログで、**追加**を押します。
10. 「マップ項目の追加」ダイアログが表示されたら、トピック ID フィールドに **HelpForEF** と入力し、マップされた数値フィールドに **12** を入力します。 **OK** を押します。
11. **OK** を押して「マップ」ダイアログをクローズします。
12. プロジェクト・ファイルを保管してコンパイルします。これによって、ヘルプ (HLP) ファイルが作成されます。

新規の HLP ファイルを VARPG プロジェクトの RT_WIN32 ディレクトリーにコピーしてください。

VARPG プログラムのコンパイル

作成プロセス時に、VisualAge RPG は RT_WIN32 ディレクトリーに HLP ファイルを作成します。これは、もちろん、コピーしたばかりの HLP ファイルを上書きします。また、RTF ファイルがプロジェクトのソース・ディレクトリーに作成されます。トピック・ファイルを同じ名前でここに保管している場合には、それは上書きされます。これを避けるためには、プロジェクトの「作成オプション」プロパティー・ノートブックをオープンして、ヘルプ・ファイル・ページに進みます。**RTF ヘルプ・ファイルの作成**チェック・ボックスのチェックを外します。これで、VisualAge RPG はヘルプを作成したり、RTF および HLP ファイルを作成することはありません。

パーツにヘルプを追加するたびに、VARPG プログラムを再コンパイルしなければなりません。

ヘルプのテスト

VARPG アプリケーションを開始します。ウィンドウが表示されたら、入力フィールドにタブを付けて F1 を押します。Windows ヘルプ・ウィンドウにヘルプが表示されるはずです。

これは何 ? ヘルプとしてヘルプを表示することもできます。そのためには、ウィンドウのプロパティー・ノートブックをオープンします。**スタイル**・ページで、**コンテキスト**・チェック・ボックスにチェックを付けます。最小化および最大化ボタンのチェック・ボックスをクリアしなければなりません。

ヘルプ・ウィンドウではなくポップ・ウィンドウにヘルプを表示するためには、**ポップアップ**選択項目にチェックしてください。

目次ファイルの作成

ヘルプに「ヘルプ・トピック」ダイアログ・ボックスを付けたい場合には、**目次ファイル**を作成する必要があります。目次ファイルは、**ファイル - 新規**および**新規の目次ファイル**を選択すると、Help Workshop IDE の中に作成されます。目次ファイルにヘルプ・ファイルと同じ名前を付けて、それを同じディレクトリーに保管します。

第 15 章 JavaHelp 作成のヒント

VisualAge RPG の機能の 1 つは、VARPG Java アプリケーションのコンテキスト・センシティブ JavaHelp に役立つ機能です。(VisualAge RPG は現在は JavaHelp 1.1 リリースをサポートしています。) JavaHelp が組み込まれた VARPG Java アプリケーションを作成して実行するためには、以下のことが必要です:

- HTML 3.2 タグの基本的な知識。
- アプリケーション用の JavaHelp メタデータ・ファイル:
  - ナビゲーション・データ - 目次ファイル (TOC)
  - HelpSet データ - HelpSet およびマップ・ファイル
  - HTML トピック・ファイル
- Java 2 ソフトウェア開発キット、標準版 (J2SDK) バージョン 1.2 またはそれ以降のコピーがワークステーションにインストールされていること。(J2SDK は次の URL の Sun から利用できます `http://java.sun.com/products/`)

このセクションは、VARPG Java アプリケーション用の基本的なコンテキスト・センシティブ JavaHelp の作成方法について要約します。JavaHelp システムの詳細については、*JavaHelp システム・ユーザーズ・ガイド* を参照してください。すべての JavaHelp 資料は JavaHelp システムで利用でき、これらは URL `http://java.sun.com/products/javahelp` からダウンロードできます。

以下のステップは、JavaHelp を作成してそれをアプリケーションに取り込む方法を要約したものです:

1. HelpSet の作成:
   - HTML トピックを作成します。
   - HelpSet ファイルを作成します。
   - マップ・ファイルを作成します。
   - 目次 (TOC) ファイルを作成します。

     オプショナルで、索引ファイルおよび全文検索データベースを作成することができます。これらのステップおよび検索をインプリメントするのに必要なツールの詳細については、*JavaHelp システム・ユーザーズ・ガイド* を参照してください。
   - ヘルプ・ファイルを圧縮して JAR ファイルにカプセル化します。
2. 必要なすべてのファイルをもつ JavaHelp を作成し、それらを JAR ファイルにパッケージした後で、JAR ファイルをプロジェクトの RT_JAVA サブディレクトリーにコピーします。
3. プロジェクトを作成して実行します。

JavaHelp システムはファイル・ベースです - トピックはファイルに入っており、一度に1 つのファイルが適切なビューアーに表示されます。関連トピックをまとめてグループ化し、それらを編成しておき、可能な限り簡単にトピック相互間をリンクさせることは、良い方法です。また、アプリケーションの圧縮 JAR に簡単にパッケージできるように、トピックを編成することも重要です。通常は、取り外して JAR ファイルに入れることができるフォルダー階層にトピックを編成するのが最良です。

ビデオ・ストア・カタログ・アプリケーションにはサンプルの JavaHelp ファイルが入っています。これらは *WDSC\samples\vidcust* ディレクトリーの *javahelp* および *help* サブディレクトリーに入っています。ユーザー自身の JavaHelp を開発するためのテンプレートとしてこれらのファイルを使用してください。

注: Java では、ファイルおよびフォルダー名は大文字小文字を区別します。提供されたサンプルに示されている通りに、正確に名前を入力してください。

HelpSet ファイルの作成

JavaHelp がアプリケーションによって開始されると、最初に行なうことは、HelpSet ファイルを読み取ることです。HelpSet ファイルはユーザーのアプリケーションの HelpSet (ヘルプ・システムを構成するデータのセット) を定義します。HelpSet ファイルには次の情報が組み込まれています。

マップ・ファイル
: マップ・ファイルは、トピック ID を HTML トピック・ファイルの URL またはパス名と関連付けます。

ビューの情報
: HelpSet で使用されるナビゲーターについて説明します。標準のナビゲーターは、目次、索引、および全文検索です。

HelpSet タイトル
: 最上位 TOC フォルダーの名前。

ホーム ID
: ID を指定しないでヘルプ・ビューアーが呼び出された時に表示される (デフォルト) ID の名前。

HelpSet ファイル (filename.hs) は、拡張マークアップ言語 (XML) 形式でコーディングされます。以下は HelpSet ファイルの例です:

```
<?xml version='1.0' encoding='ISO-8859-1'  ?>
<!DOCTYPE helpset
  PUBLIC "-//Sun Microsystems Inc.//DTD JavaHelp HelpSet Version 1.0//EN"
         "http://java.sun.com/products/javahelp/helpset_1_0.dtd">

<helpset version="1.0">

  <!-- title -->
  <title>Video Store Catalog - Help</title>

  <!-- maps -->
  <maps>
     <homeID>11</homeID>
     <mapref location="Map.jhm"/>
  </maps>

  <!-- views -->
  <view>
    <name>TOC</name>
    <label>Table Of Contents</label>
    <type>javax.help.TOCView</type>
    <data>VIDCTOC.xml</data>
  </view>

</helpset>
```

ここで:

<title> HelpSet に命名します。これはヘルプ・ウィンドウのタイトルに対応します。

<homeID>
ID が明示的に指定されていない場合に、ヘルプが呼び出された時に表示される (デフォルト) ID の名前を指定します。

<data>
ナビゲーターで使用されるデータへのパスを指定します。この例では、これは TOC ビューです。TOC ファイル名は大文字で、xml 拡張子は小文字です。TOC ファイルは、ヘルプ・ディレクトリーの中に存在していなければなりません。

マップ・ファイルの作成

アプリケーションが JavaHelp を活動化すると、最初にアプリケーションの HelpSet を読み取ります。次に、HelpSet ファイルにリストされているマップ・ファイルを読み取ります。マップ・ファイルはトピック ID を URL (HTML トピック・ファイルへのパス) と関連付けます。規則により、マップ・ファイル名には*jhm* 接尾部が組み込まれます。マップ・ファイルは XML 形式です。

以下はマップ・ファイルの例です:

```
<?xml version='1.0' encoding='ISO-8859-1'  ?>
<!DOCTYPE map
  PUBLIC "-//Sun Microsystems Inc.//DTD JavaHelp Map Version 1.0//EN"
         "http://java.sun.com/products/javahelp/map_1_0.dtd">

<map version="1.0">
        <mapID target="11" url="help/welcome.htm" />
        <mapID target="18" url="help/catalog.htm" />
        <mapID target="14" url="help/browse.htm" />
        <mapID target="15" url="help/new.htm" />
        <mapID target="16" url="help/top10.htm" />
        <mapID target="17" url="help/search.htm" />
</map>
```

target VARPG パーツのパーツ ID を指定します。パーツ ID は、GUI Designer によって自動的にパーツに割り当てられます。これはパーツのプロパティー・ノートブックから検索することができます。

url ヘルプ・テキストを含む HTML トピック・ファイルへのパスを指定します。パスは相対または絶対とすることができます。

TOC ファイルの作成

目次 (TOC) ファイルは、TOC の内容およびレイアウトについて TOC ナビゲーターに説明します。TOC ファイルは XML 形式です。以下は TOC ファイルの小さな例です:

```
<?xml version='1.0' encoding='ISO-8859-1'  ?>
<!DOCTYPE toc
  PUBLIC "-//Sun Microsystems Inc.//DTD JavaHelp TOC Version 1.0//EN"
         "http://java.sun.com/products/javahelp/toc_1_0.dtd">
<toc version="1.0">
<tocitem text="Video Store Catalog - help">
```

```
   <tocitem text="Welcome" target="11"/>
   <tocitem text="Help" target="22"/>
   <tocitem text="Browse" target="14"/>
   <tocitem text="New" target="19"/>
   <tocitem text="Top 10" target="20"/>
   <tocitem text="Search" target="21"/>

</tocitem>
</toc>
```

ここで:

tocitem
最初の TOC 項目は目次のタイトルを指定します。 (TOC 項目を高位の項目にネストすることができます。)

text 後続の TOC 項目で使用するテキストを指定します。

target ユーザーが TOC でこの項目を選択した時に表示する HTML トピックの ID を指定します。この ID はマップ・ファイルで識別されたパーツ ID と対応します。

JAR ファイルの作成

必要なすべてのヘルプ・ファイルが作成されたら、jar コマンドを使用してファイルをカプセル化して圧縮します。jar ファイル名は次の通りでなければなりません:

```
SOURCE_FILE_NAMEHS
```

ここで、SOURCE_FILE_NAME パーツは、「アプリケーションとして保管 - VisualAge RPG」ウィンドウのソース・ファイル ・フィールドに指定された名前です。ファイル名は HS で終わり、大文字でなければなりません。jar 拡張子は小文字です。

ヘルプ階層が入っている最上位のディレクトリーからコマンドを出します。たとえば、ヘルプ・ディレクトリー構造が次のような場合には:

```
javahelp (directory)
    Map.jhm
    CATALOG.hs
    VIDCTOC.xml

    help (subdirectory)
        browse.htm
        catalog.htm
        new.htm
        search.htm
        top10.htm
        welcome.htm
```

javahelp ディレクトリーから次のように jar コマンドを出します:

```
jar -cf VIDCUSTHS.jar *.*
```

結果として生ずる jar ファイルをプロジェクトの RT_JAVA サブディレクトリーにコピーします。 Java オプションを使用してプロジェクトを作成および実行します (**作成 > Java** または**実行 > Java**)。

第 16 章 メッセージの処理

ユーザー VARPG アプリケーションのメッセージを作成、表示、編集、および削除することができます。

既存のメッセージをメッセージの定義ウィンドウから直接表示し削除することができます。新しいメッセージを作成したり既存のメッセージを変更することができるメッセージの編集ウィンドウにアクセスするためには、メッセージの定義ウィンドウを使用します。

メッセージは VARPG で次の 2 つのグループに分けられます。すなわち、実行時にユーザー・コードで参照できないグループと、参照できるグループです。

最初のグループは、置換ラベル (たとえば、プッシュボタンまたはウィンドウ) の置き換えに使用されるラベル・タイプのメッセージから構成されます。

2 番目のグループには 4 つのタイプのメッセージがあり、アクション、重要、通知、および警告です。これらのメッセージは、メッセージ・ウィンドウまたはメッセージ・サブファイル・パーツに表示することができます。これらは、実行時にユーザー・インターフェースのテキストを動的に更新する (たとえば、インストール進行状況メッセージを表示する) ために使用することができます。

置換ラベルのテキストの定義

置換ラベルとテキストを関連付ける場合:

1. パーツに置換ラベルが定義されていることを確認してください。オンライン・ヘルプに説明されている手順に従ってください。
2. GUI Designer から「**プロジェクト→メッセージの定義**」を選択します。「メッセージの定義」ウィンドウがオープンします。
3. 表示されたリストからラベル-タイプ・メッセージを選択します。
4. 「**編集**」プッシュボタンを選択します。「メッセージの編集」ウィンドウがオープンして、選択したラベルが表示されます。
5. 「**メッセージ**」フィールドでは、そのラベルで置き換えたいテキストを入力します。
6. ユーザー変更を保管するためには「**保管**」を選択し、それを破棄するためには「**キャンセル**」を選択 (あるいは、ウィンドウのシステム・メニューをダブルクリック) してください。

注: GUI Designer の置換ラベルを持つパーツのサイズを設定する場合には、変換されたテキストが元の長さよりも長くなる場合があることに留意してください。

新規メッセージの作成

新しいメッセージを作成する場合:

1. GUI Designer から「**プロジェクト→メッセージの定義**」を選択します。「メッセージの定義」ウィンドウがオープンします。
2. **作成**を選択します。「メッセージの編集」ウィンドウがオープンします。
3. **メッセージング別名**フィールドに、10 文字までの長さのストリングを入力します。これにブランクを入れることはできません。コードには、メッセージを表示するためのメッセージ ID の代わりにメッセージ別名を使用することができます。
4. 「**タイプ**」ドロップダウン・ボックスからメッセージ・タイプを選択します。次の 4 つのタイプがあり、この中から選択します。

   メッセージ・タイプ
   意味

   アクション
   置換ではこのタイプのメッセージを使用しますが、その場合に**特定の**アクションを行なって状態を訂正するか、あるいは代替アクションを選択しなければなりません。

   重要 置換ではこのタイプのメッセージを使用しますが、その場合に**即時**アクションを行なって状態を訂正するか、あるいは代替アクションを選択しなければなりません。

   通知 ユーザーに単に状態を告げるだけで、ユーザーがアクションを行なう必要がないような状態にはこのタイプのメッセージを使用してください。

   警告 ユーザーが修正なしに元の要求を続行することができるが、ある種の状態が存在することを認識する必要がある時にはこのタイプのメッセージを使用してください。
5. メッセージ・テキストを「**メッセージ**」フィールドに入力します。
6. メッセージのヘルプを提供したい場合には、「**メッセージ・ヘルプ**」フィールドにこれを入力してください。

   メッセージ・ヘルプを作成し、DSPLY 命令コードを使用してメッセージを表示する時には、「**ヘルプ**」プッシュボタンがメッセージ・ウィンドウの下部が表示されます。ユーザーがこのプッシュボタンをクリックすると、追加の情報としてヘルプ・テキストが表示されます。
7. メッセージを背景に移動して、そのメッセージでアクションを行なう前にその他のタスクを続行したい場合には、「**移動可能**」チェック・ボックスを選択してください。
8. 「**ボタン**」ドロップダウン・ボックスから、メッセージ・ウィンドウの下部に表示したいプッシュボタンの組み合わせを選択します。

   選択項目
   現れるボタン

   abortRetryIgnoreButton
   中止、再試行、および無視

   okButton
   OK

okCancelButton
OK およびキャンセル

retryCancelButton
再試行およびキャンセル

yesNoButton
はい、およびいいえ

yesNoCancelButton
はい、いいえ、およびキャンセル

9. **ボタン 1**、**ボタン 2**、または**ボタン 3** のラジオ・ボタンを選択することによって、デフォルトのプッシュボタンを選択します。メッセージ・ウィンドウが表示されて、ユーザーが Enter キーを押すと、デフォルトのプッシュボタンに関連付けられたアクションが実行されます。

   たとえば、「**ボタン**」ドロップダウンから enterCancelButton を選択して、デフォルトのプッシュボタンを「キャンセル」にしたい場合には、「**ボタン 2**」ラジオ・ボタンを選択してください。
10. メッセージを保管するためには**保管**を選択し、これを破棄するには **キャンセル** を選択します。

注: メッセージ ID (メッセージ ID) MSG0001 から MSG9999 の範囲で、VisualAge RPGによって割り当てられます。この範囲のすべてのメッセージ ID が使用された場合には、新規メッセージを作成しようとすると、VisualAge RPG がエラーを通知して、それを削除するまで新規メッセージを作成することができません。メッセージを削除した後で、削除したメッセージの ID を使用する新しいメッセージを作成することができます。

メッセージの編集

メッセージを編集する場合:
1. GUI Designer から「**プロジェクト→メッセージの定義**」を選択します。「メッセージの定義」ウィンドウが現れます。
2. 表示されたリストからメッセージを選択します。必要なメッセージを検索できない場合には、「 276 ページの『メッセージの検索』.
3. 「メッセージの定義」ウィンドウから「**編集**」を選択します。「メッセージの編集」ウィンドウがオープンして、選択したメッセージが表示されます。
4. メッセージの別名、タイプ、テキスト、ヘルプ、またはメッセージ・ウィンドウの情報を変更します。
5. 変更を保管するためには「**保管**」を選択し、これを破棄するためには「 **キャンセル**」を選択します。

メッセージの削除

メッセージを削除する場合:
1. GUI Designer から「**プロジェクト→メッセージの定義**」を選択します。「メッセージの定義」ウィンドウがオープンします。
2. 表示されたリストからメッセージを選択します。必要なメッセージを検索できない場合には、「 276 ページの『メッセージの検索』.

3. 「**削除**」プッシュボタンを選択します。

メッセージの検索

メッセージの検索についてのヒントは次の通りです。

- 正確なメッセージ ID が判明している場合には、「メッセージの定義」ウィンドウの「**メッセージ ID によるソート**」機能を使用してください。メッセージは、メッセージ ID の昇順で表示されます。
- 検索しているメッセージのタイプが判明している場合には、「メッセージの定義」ウィンドウの「**タイプによるソート**」機能を使用してください。メッセージは、次のグループ内でメッセージ ID の昇順にソートされます。
  1. 実行時に設定できるメッセージ:
     a. 情報
     b. 警告
     c. アクション
     d. 重大
  2. 実行時に設定できないメッセージ (置換ラベル)

矢印キーまたはスクロール・バーを使用して、メッセージのリストを移動することができます。リストが長い場合には、スクロールが検索するものを見つける一番早い方法です。

ロジック付きメッセージの使用

実行時にメッセージ・ウィンドウにメッセージを表示するのは通常の方法です。メッセージが作成された後で、それを表示する 1 つの方法は、DSPLY 命令コードおよびメッセージ・サブファイル・パーツの **AddMsgID** 属性を使用することです。

AddMsgID 属性については、*ADTS/CS VisualAge for RPG パーツ解説書*, SD88-5040-03を参照してください。

定義指定で MSGDATA および MSGNBR キーワードを使用し、置換変数によってメッセージを定義することができます。置換変数は、メッセージの作成時に、パーセント (%) 文字の後に数字を続けて入力する (たとえば *%1 %2 %3*) ことによって定義します。置換変数は、MSGDATA キーワードに定義された対応のフィールドで置き換えられます。たとえば、%1 は MSGDATA に定義された最初のフィールドで置き換えられ、%2 は MSGDATA に定義された 2 番目のフィールドで置き換えられる、などとなります。 MSGNBR キーワードには、8 文字のメッセージ識別コード (たとえば *MSG0001) が入っていなければなりません。

DSPLY 命令コードでメッセージ置き換えを使用するには、D 仕様でメッセージ・データ・タイプを定義してください。たとえば、次のようになります。

```
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
 *
D notFound        M                   MSGNBR(*MSG0001)
D                                     MSGDATA(cusno: file)
 *
```

フィールド CUSNO と FILE はプログラム内のどこかで定義されています。メッセージ *MSG0001 のメッセージ・テキストは次のようになっているとします。

顧客番号 %1 がファイル %2 に見つかりません。

DSPLY 命令でメッセージを表示して置き換えを実行するには、C 仕様で次のようにコーディングしてください。

```
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq
C     notFound      DSPLY                   rc                9 0
```

DSPLY 命令コードの詳細については、*VisualAge for RPG WINDOWS 版 言語解説書*, SC88-5608-04 を参照してください。

メッセージ・ファイルの変換

ユーザー・アプリケーションを再コンパイルして、変換されたメッセージを取り込む必要はありません。

メッセージ・ファイルのそれぞれに異なったファイル拡張子を割り当てることによって、実行時ディレクトリーに複数のメッセージ・ファイルを入れることができます。たとえば、コンパイルされたメッセージ・ファイルの英語バージョンには SAMPLE.ENG の名前を付け、ドイツ語バージョンには SAMPLE.GER の名前を付けることができます。アプリケーションを実行する前に、SAMPLE.MSG に対して適切なメッセージ・ファイルに名前変更するよう、ユーザーに指示することができます。

メッセージ・ファイルの手作業による変更

変換のために .TXM ASCII ファイルを手作業で変更することができます。このファイルには、ユーザー・アプリケーション用に作成したメッセージが入っています。これは、アプリケーションの作成時に指定したソース・ディレクトリーに作成されます。

ファイルのレコード設計の例は 図 55 に示されています。

```
MSG
MSG0001I: ファイルはユーザーの現行作業ディレクトリーに保管されました。
MSG0002W: 別のユーザーが編集のためにこのファイルをすでにオープンしています。
```

図 55. TXM ファイルのサンプルのレコード設計

編集するのは、レコード設計のコロン (:) の後に表示されたテキストだけにしてください。

最初のレコードがメッセージ接頭語を識別し、次のレコードはそれぞれアプリケーション中のメッセージを表します。

各メッセージには、メッセージ接頭語、*MSG* ; 4 桁の ID または ID 番号; およびメッセージのタイプを記述する英字があります。この例では、メッセージ番号 1 は通知メッセージであり、メッセージ番号 2 は警告メッセージです。

次のいずれも実行**しない**でください。

- メッセージ ID の変更。メッセージ ID を変更すると、予測できない結果が起こります。メッセージ ID がないと、そのメッセージを表示することはできません。

- メッセージの追加。メッセージ・スタイルが定義されず、「メッセージの定義」ウィンドウにメッセージは表示されません。
- メッセージの削除。「メッセージの定義」ウィンドウには、そのメッセージ・テキストを除いて、メッセージに関するすべてがまだ表示されます。

ラベルとしてのメッセージの使用

LABEL 属性を持つパーツのラベルは、メッセージ・ファイルのメッセージ・テキストから設定することができます。プレフィックス '*MSG' をもつラベルは、メッセージ・ファイルからのメッセージ・テキストを示します。 図 56の例では、プッシュボタン PB1 のラベルは、メッセージ・ファイ ルのメッセージ番号 0001 からのテキストで設定されます。

```
C    'PB1'      SETATR   '*MSG0001'    'Label'
```

図 56. メッセージ・ファイルからのパーツ・ラベルの動的設定

メッセージ番号をコンポーネント・メッセージ・ファイルの中で検出できない場合に、アプリケーションは、そのメッセージ番号の *Component MsgFile 属性によって指示されたメッセージ番号を検索します。このメッセージ番号がいずれのメッセージ・ファイルにも存在しない場合には、メッセージ ID (この例では、MSG0001) がラベル・テキストとして表示されます。

第 17 章 オブジェクト間の通信

VisualAge RPG では、オブジェクト間で各種の通信を実行できます。

パーツ間

VisualAge RPG のパーツは、1 つのパーツが他のパーツに変更されたことを通知して、受信側パーツがこの変更を通知された時にイベントを発行するようにパーツ間をリンクすることができます。

VisualAge RPG アプリケーションと他の PWS アプリケーション間

ユーザー・アプリケーションが DDE プロトコルをサポートする別のアプリケーションと情報を交換できるようにすることができます。 VisualAge RPG アプリケーションは、この交換のクライアントまたはサーバーとすることができます。クライアント機能については、80 ページの『DDE クライアント』 を参照してください。サーバー機能については、この項で説明します。

コンポーネント間

1 つのコンポーネントが別のコンポーネントと通信することができます。

また、命令コードを使用して以下を行うこともできます。
- ローカル機能の呼び出し
- ローカル・プログラムの呼び出し
- コンポーネントの開始および停止
- リモート・プログラムの呼び出し

この項では、それぞれのタイプの通信に有益なヒントを提供し、いくつかの例を示します。

パーツのリンク

次のパーツは VisualAge RPG を使用してリンクすることができます。
- チェック・ボックス
- 入力フィールド
- イメージ
- リスト・ボックス
- メディア
- メディア・パネル
- スライダー
- タイマー

変更時に別のパーツに通知するパーツは**ソース**・パーツと呼ばれ、この変更について通知を受けるパーツは**ターゲット**・パーツと呼ばれます。

ソース・パーツとターゲット・パーツの間で通信を設定する 1 つの方法は、ソース・パーツのプロパティー・ノートブックのリンク・ページを使用する方法です。提供されたフィールドに、ターゲット・パーツの名前とそれが入っているウィンド

ウの名前を入力してください。ソース・パーツによって通知された時にターゲットに**リンク**・イベントを出させたい場合には、**ターゲット通知可能**チェック・ボックスを選択してください。

また、 *WindowName|PartName* の形式で **AddLink** 属性とターゲットを設定して、通信リンクを設定することもできます。ターゲットに**リンク**・イベントを発行させたい場合には、**AllowLink** 属性を *1* に設定します。図 57 に、メディア・パネル・パーツ MMP1 をメディア・パーツ AUDIO1 にリンクするために使用するコードの例を示します。

```
 *
C     'MMP1'        SETATR    'WIN2|AUDIO1'  'AddLink'
C     'MMP1'        SETATR    1              'AllowLink'
 *
```

図 57. 別のパーツにリンクした 1 つのパーツを示すサンプル・コード

注: GUI Designer ではソース・パーツに 1 つのリンクしか 設定できませんが、ユーザー・コードでは複数のリンクを設定することができます。

DDE サーバーとしての VisualAge RPG アプリケーションの使用

VisualAge RPG アプリケーションは、動的データ交換 (DDE) 会話のサーバーとして機能することができます。

LINK イベントのソースになることができるパーツは DDE データを作成することができます。DDE クライアント・パーツは、同じアプリケーションまたは別のアプリケーションのコンポーネントからデータを取得することができます。DDE クライアント・パーツの詳細については、 80 ページの『DDE クライアント』 を参照してください。

たとえば、CLIENT と呼ばれるアプリケーションをビルドしているとします。これは WINDOW_C と呼ばれるウィンドウ、DDECLI_C と呼ばれる DDE クライアント・パーツ、および STTEXT_C と呼ばれる静的テキスト・パーツから構成されます。

このアプリケーションは SERVER と呼ばれるサーバー・アプリケーションからのデータを必要としているとします。このサーバー・アプリケーションには、WINDOW_S と呼ばれるウィンドウと ENTRY_S と呼ばれる入力フィールド・パーツがあります。サーバー・アプリケーションの入力フィールドの値が変更されると、クライアント・アプリケーションの静的テキスト・パーツが更新されてこの変更が反映されます。

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間でホット・リンクを確立するには、クライアント・アプリケーションの DDECLI_C DDE クライアント・パーツの次の属性を指定します。

AppName

これはサーバー・アプリケーション SERVER.EXE の名前です。

Topic

これは、サーバー・コンポーネントの後に縦線が続き、さらにコンポーネントのインスタンス名が続く名前です。VisualAge RPG では、ほとんどの場合に、コンポーネント名はコンポーネントのインスタンス名と同じで、実行可能な名前とも同じです。この例では、コンポーネント名は SERVER|SERVER です。

Item

これはサーバー・パーツの名前です。VisualAge RPG プログラムの場合には、これはウィンドウ名の後に縦線が続き、さらにパーツ名が続く名前です。この例では、この項目の属性値は WINDOW_S|ENTRY_S です。

DDEAddLink

これは、クライアント・パーツの名前です。これはウィンドウ名の後に縦線が続き、さらにパーツ名が続く名前です。この例では、 **DDEAddLink** 属性は WINDOW_C|STTEXT_C です。

DDEMode

サーバーとクライアントとの間で会話を開始し、ホット・リンクを始めるには、**DDEMode** を *1* に設定します。会話を終了するには、**DDEMode** を *2* に設定します。これでクライアント・アプリケーションに**終了**イベントが通知されます。

コンポーネント間の通信

コンポーネントは VisualAge RPG のプロジェクトです。これらは GUI Designer で作成された 1 つまたは複数のアプリケーション・ウィンドウを表します。例は、ユーザーにイメージ・ファイル名の入力を求めるプロンプトを出すウィンドウで、その後でイメージを表示します。別のコンポーネントと通信する 1 つの VisualAge RPG コンポーネントを使用可能にするためには、コンポーネント参照パーツを使用してください。詳細については、 71 ページの『コンポーネント参照』 を参照してください。

ローカル・コールの実行

この項では、これらの命令コードを使用して行なうことができるローカル・コールについて説明します。:

命令コード
　目的

CALLB
　ローカル機能を呼び出します。この機能はオブジェクト・コード・ファイル (OBJ) に入っているか、またはダイナミック・リンク・ライブラリー (DLL) からエクスポートすることができます。

CALLP
　ローカル・プログラムまたは機能 (プロシージャー) を呼び出します。この機能はダイナミック・リンク・ライブラリー (DLL) からエクスポートしな

ければなりません。詳細については、 290 ページの『複数のプロシージャーの使用』を参照してください。CALLP の使用は CALLB の使用に優先します。

START

アプリケーションで新しいコンポーネントを開始するか、またはローカル・プログラムを呼び出します。

CALLB 命令の使用

CALLB 命令コードは、VisualAge RPG アプリケーションから機能を呼び出すために使用します。RPG 以外の言語でコンパイルされた OBJ にリンクしている場合には、コンパイラーの実行時環境が正しく初期化され終了されていることを確認してください (詳細については、コンパイラーの資料を参照してください) 。

次の例は、CALLB を使用して C の機能を呼び出す別の方法を示しています。図 58 には、呼び出されるサンプルの C 機能が入っています。

```
#include <stdio.h>
 *
/*The following two lines are required only if you compile */
/*the OBJ with the IBM C/C++ compiler.  These lines  */
/*are not required if the function is exported from a DLL. */
int  _CRT_init(void);
void _CRT_term(void);
 *
/* print the str and age parameters to a file */
void MYFUNC(char *str, int *age) {
   FILE *fp;
   int j;
 *
/*The following line is required only if you compile       */
/*the OBJ with the IBM C/C++ compiler.  This line    */
/*is not required if the function is exported from a DLL.  */
   _CRT_init();
  *
   fp=fopen("myfunc.log", "a");
  *
   /* print the character data to a file*/
   for (j=0; j<10; ++j) {
      fprintf(fp, "%c", str[j]);
   }
  *
   /* if an age is given, print the age */
   if ( age == NULL ) {
      fprintf(fp, "no age is given\n");
   } else {
      fprintf(fp, "num = %d\n", *age);
   }
  *
   fclose(fp);
  *
/*The following line is required only if you compile       */
/*the OBJ with the IBM C/C++ compiler.  This line    */
/*is not required if the function is exported from a DLL.  */
   _CRT_term();
}
```

図 58. サンプルの C 機能 MYFUNC

名前の付いた固定情報またはリテラルを使用した機能の呼び出し

次の例は、名前の付いた固定情報またはリテラルを使用した機能の呼び出し方法を示しています。

```
DConst1          C                   CONST('MYFUNC')
Dwilma           s             80a   inz('mydata')
Dage             s              9b 0 inz(32)
 *
 *
C     *inzsr       begsr
c*************************************************************************
c***********  *** PLIST を指定した VRPG の CALLB ***  *****************
c*************************************************************************
C     myplist      plist
C                  parm                    wilma
C                  parm                    age
C                  CALLB     Const1        myplist
C                  seton                                        lr
C                  endsr
```

図 59. 名前の付いた固定情報を使用した機能の呼び出し

```
 *
Dwilma           s             80a   inz('mydata')
Dage             s              9b 0 inz(32)
C     *inzsr       begsr
C                  callb     'MYFUNC'
C                  parm                    wilma
C                  parm                    age
C                  seton                                        lr
C                  endsr
```

図 60. リテラルを使用したライブラリー機能の呼び出し

プロシージャー・ポインターを使用した機能の呼び出し

次の例は、プロシージャー・ポインターを使用した機能の呼び出し方法を示しています。CALLB でプロシージャー・ポインターが使用されると、プログラム状況データ構造 (PSDS) の *ROUTINE フィールドは、呼び出された機能の名前には更新されません。このフィールドはブランクに設定されます。

```
 *
Dp2              s               *   procptr inz(%paddr('MYFUNC'))
Dwilma           s             80a   inz('mydata')
Dage             s              9b 0 inz(32)
C     *inzsr       begsr
C                  callb     p2
C                  parm                    wilma
C                  parm                    age
C                  seton                                        lr
C                  endsr
```

図 61. プロシージャー・ポインターを使用した機能の呼び出し

必要なパラメーターなしの機能の呼び出し

次の例は、必要な数より少ないパラメーターでの機能の呼び出し方法を示しています。NULL ポインターにマップする *OMIT パラメーターを使用します。

```
 *
Dp2               s                 *   procptr inz(%paddr('MYFUNC'))
Dwilma            s               80a   inz('mydata')
Dage              s                9b 0 inz(32)
C     *inzsr        begsr
C                   callb     p2
C                   parm                    wilma
C                   parm                    *OMIT
C                   seton                                        lr
C                   endsr
```

図 62. 必要なパラメーターなしの機能の呼び出し

CALLP を使用したローカル・プログラムの呼び出し

CALLP は、ローカル・プログラムの同期呼び出しを行ないます。このことは、CALLP の後で呼び出されたプログラムが VisualAge RPG ステートメントを実行する前に実行を完了することを意味します。

CALLP を使用して呼び出すそれぞれのプログラムにはプロトタイプが必要です。このプロトタイプは、呼び出されたプログラムのシステム名とそのプログラムが必要とするパラメーターの数とタイプを定義します。このプロトタイプは、PR タイプの定義仕様を使用して指定してください。この仕様は以下から構成されます。

桁 **説明**

6 D

7-21 VisualAge RPG プログラムで使用するプログラムの名前

24-25 PR

44-80 キーワード

パラメーターとしてプログラムのシステム名と CLTPGM キーワードを使用します。

プログラムにパラメーターが必要な場合には、PR 定義仕様の直後に それぞれのパラメーターに 1 つの定義仕様を使用してください。これらの定義仕様は、名前、長さ、およびパラメーターのタイプから構成されなければなりません。数値パラメーターの精度を指定してください。常に VALUE キーワードを指定してください。パラメーター定義には ASC、DATFMT、DESC、DIM、LIKE、NOOPT、OPTIONS、および TIMFMT キーワードも指定することができます。

285 ページの図 63 は pgm1 を VisualAge RPG に定義します。1 つのパラメーターをプログラムに渡すことができます。

```
D pgm1              PR                  CLTPGM('testprog')
D  parm1                          20A   VALUE
```

図 63. ローカル・プログラムの呼び出し時の定義仕様パラメーターの指定

図 64 では、CALLP 命令コードはパラメーターが f1d1 と 22.4 の pgm1 を呼び出します。

```
C                   CALLP       pgm1(f1d1:22.4)
```

図 64. CALLP を使用したローカル・プログラムの呼び出し

プロシージャーの詳細については、 290 ページの『複数のプロシージャーの使用』を参照してください。

START を使用したローカル・プログラムの呼び出し

START 命令コードを使用してプログラムを呼び出すと、VisualAge RPG は呼び出されたプログラムが実行を終了するまで待機せずに、呼び出しを行なって続行します。この時点から、呼び出されたプログラムは呼び出した VisualAge RPG プログラムとは別個に実行します。

START の使用時には、プロトタイプのローカル・プログラムは必要ありません。

F2 は、文字リテラル、名前の付いた固定情報、または変数名とすることができます。

F2 が文字リテラルの場合には、それはコンポーネントと見なされます。これが固定情報名で固定情報の定義に LINKAGE(*CLIENT) を指定している場合には、それはローカル・プログラムと見なされます。 図 65を参照してください。

```
D test1             C                   'component'
D test2             C                   'testprog'   LINKAGE(*CLIENT)
 *
*To start a component:
C                   START               'xxx'
 *
*To start a component:
C                   START               test1
 *
*Starts local program testprog.exe:
C                   START               test2
```

図 65. START を使用してローカル・プログラムを呼び出す例

F2 が変数名の場合には、LINKAGE(*CLIENT) を指定した定義仕様で 変数を定義しない限り、それはコンポーネント名と見なされます。このように定義された変数は他の RPG フィールドと同様に使用することができます。 286 ページの図 66 では、最初の START 命令コードはコンポーネントを開始しようとし、 2 番目の START 命令コードはローカル・プログラムを開始しようとします。

```
D name1               S             20A
D name2               S             20A        LINKAGE(*CLIENT)
 *
C                   START         name1
C                   START         name2
```

図 66. START 命令コードの変数名の定義

START は結果フィールドに PLIST を指定するか、あるいは PARMS のリストを続けることができます。これらの PARMS はコンポーネントまたはローカル・プログラムに渡されます。

CALLP および START の制約事項

ローカル・プログラムで CALLP および START 命令コードを使用する場合には、これらの制約事項に注意してください。

- プログラム名が完全なパス名と一緒に指定されていない場合には、ローカル・プログラムを見つけるために PATH 環境変数が使用されます。
- 通常、プログラムは最大 20 のパラメーターを持つことができます。場合によっては、コマンド・ストリングが 1024 バイトを超えることができないので、この最大が 20 以下になることもあります。(コマンド・ストリングは、文字に変換されたプログラム名とパラメーターから構成されます。)
- ポインターおよびプロシージャー・ポインターはパラメーターとして使用することができません。すべてが値で渡されなければなりません。
- エラー標識付きの START を使用してローカル・プログラムを呼び出している場合には、ローカル・プログラムを開始できなければエラー標識が ON に設定されます。
- START 命令コードで LINKAGE(*SERVER) は無効です。
- 呼び出すプログラムの名前を指定する場合には、EXE 以外では拡張子を含めてください。拡張子を指定しないと、EXE と見なされます。たとえば、次のようになります。

  CLTPGM('superc2')
  SUPERC2.EXE を呼び出します。

  CLTPGM('rexxpgm')
  REXXPGM.EXE を呼び出します。

  CLTPGM('rexxpgm.cmd')
  REXXPGM.CMD を呼び出します。

  これは、プログラムめいを名前の付いた固定情報として指定するか、またはプログラムを変数として渡す場合に適用されます。

START を使用したコンポーネントの開始

START 命令コードはアプリケーションの新しいコンポーネントを開始するために使用し、**STOP** 命令コードはその実行を終了するために使用します。これらの 2 つの命令コードの構文の詳細については、*VisualAge for RPG WINDOWS 版 言語解説書* を参照してください。

次の項では、アプリケーションのコンポーネントの START および STOP の性質について説明します。

コンポーネントの開始

START 命令コードは、アプリケーションの新しいコンポーネントを開始します。この命令が実行されると、開始と開始されたコンポーネントの両方がそのアプリケーションの他の活動状態のコンポーネントと一緒に、すべてのコンポーネントによって現在使用可能になっているすべてのパーツに対するユーザーのアクションを受け取る用意ができます。

START 命令コードは、次の点で CALL 命令コードと似ています。

- パラメーターをコンポーネントに渡すことができます。
- パラメーターは、ターゲット・コンポーネントの *ENTRY PLIST のパラメーターにマップされます。
- ソース・コンポーネントでは、PARM 命令コードの演算項目 2 が同じ PARM 命令コードの結果フィールドにコピーされます。ソース・コンポーネントに制御が戻されると、結果フィールドが演算項目 1 にコピーされます。
- ターゲット・コンポーネントでは、結果フィールドが演算項目 1 にコピーされます。制御がソース・コンポーネントに戻されると、ターゲット・コンポーネントが正常な始動を完了していれば、演算項目 2 が結果フィールドにコピーされます。
- パラメーターに検査や変換は行なわれません。

START 命令コードは、CALL 命令コードとは次のように違っています。

- CALL 命令コードでは、**呼び出し先**および**呼び出し** という用語が使用されます。**呼び出し先プログラム**は、その実行が別のプログラムによって要求されているプログラムです。**呼び出し側プログラム** は、別のプログラムの実行を要求するプログラムです。 START 命令コードでは、**ターゲット**（呼び出し先）および**ソース**（呼び出し）という用語が使用されます。
- CALL はプログラムを呼び出し、これを実行して、上記のようにコピーされた演算項目 1、演算項目 2、および結果フィールドを呼び出し側プログラムに戻します。START はコンポーネントを初期設定し、その *INZSR を実行して、上記のようにコピーされた演算項目 1、演算項目 2、および結果フィールドをソース・コンポーネントに戻します。相違は、START 命令コードでは、ターゲット・プログラムの演算項目 2 が、プログラムの終わりではなく、*INZSR の終わりに (*INZSR が成功した場合に）結果フィールドにコピーされる点です。
- START 命令がターゲット・コンポーネントの初期設定を終了すると、ソース・コンポーネントのアクション・サブルーチンは実行を続行し、ターゲット・コンポーネントはイベントを受け取るために使用可能にされたそのアクション・サブルーチンを活動状態のままにします。
- パラメーターはアドレスで渡されるので、渡されたパラメーターは初期 START の終了後にソースとターゲット・コンポーネントの両方がアクセスできます。このことは、ソースとターゲット・コンポーネントの両方がパラメーター・フィールドを使用して情報の共用を継続できることを意味します。

コンポーネントの終了

STOP 命令コードは、コンポーネントの実行を終了します。演算項目 2 にコンポーネント名を指定しないと、現在実行中のコンポーネントが終了されます。コンポーネントが終了される時には、開始されている子コンポーネントが最初に終了されます。

現在実行中のコンポーネントに影響を与える STOP 命令が実行されると、 STOP の後の命令は実行されません。言いかえれば、STOP の結果は即時です。たとえば、COMPA が COMPB を開始して COMPB が現在実行中のコンポーネントで、これが COMPA の STOP を出したとすれば、最初に COMPB が終了し続いて COMPA が終了します。STOP の後の命令は実行されません。

STOP でのコンポーネントの終了は通常の終了と考えられ、最終ユーザー処理のために *TERMSR が呼び出されます。

リモート・プログラムの呼び出し

この項では、VisualAge RPG アプリケーションが iSeries 400 プログラムを呼び出す方法と、iSeries 400 サーバーで実行中の RPG アプリケーションが VisualAge RPG アプリケーションを呼び出す方法について説明します。

iSeries 400 プログラムの呼び出し

アプリケーションが iSeries サーバーを呼び出す前に、サーバーをセットアップする必要があります。

呼び出し先プログラムの名前は、iSeries サーバー・プログラム名（任意選択でライブラリー修飾）か一時変更名のどちらかにすることができます。プログラムの一時変更は、「iSeries 情報の定義」ノートブックのプログラム・ページを使用して定義することができます。ノートブック・ページにデータ域の一時変更名が入っていない場合にどうなるかについては、 205 ページの『ノートブックについての考慮事項』 を参照してください。

表 12 および 289 ページの図 67 は、一時変更名を使用して iSeries プログラムを呼び出す方法を図示したものです。 289 ページの図 67 のプログラムが SERVER01 の MYLIB/LOOKUP を呼び出します。

表 12. この情報はプログラム・ページに入力します。

プログラム一時変更名:	REMPGM
リモート・プログラム名:	MYLIB/LOOKUP
サーバーの別名:	SERVER01

```
 ********************************************************************
 *                                                                  *
 * プログラム ID. : rcallex.vpg                                     *
 *                                                                  *
 * 説明 . . . . . : AS/400 のリモート・プログラムを呼び出すコード・  *
 *                  セグメント。                                    *
 *                                                                  *
 ********************************************************************
 *
 * REMPGM はリモート・プログラムの別名です
D as400pgm        S                6A   INZ('REMPGM') LINKAGE(*SERVER)
 * 次の変数は、リモート・プログラムに渡されるパラメーター
 * です
 *    student_id   - input
 *    name         - output
D student_id      S                6S 0 INZ(32533)
D name            S               20A
 ********************************************************************
 *                                                                  *
 * ウィンドウ : WIN1                                                *
 *                                                                  *
 * パーツ . . : PSB0000C                                            *
 *                                                                  *
 * イベント . . : PRESS                                             *
 *                                                                  *
 * 説明 . . . : 生徒 ID と関連付ける担当者の名前を取得するために    *
 *               AS/400 のリモート・プログラムを呼び出します。      *
 *                                                                  *
 ********************************************************************
 *
C     PSB0000C      BEGACT    PRESS         WIN1
C                   CALL      as400pgm
C                   PARM                    student_id
C                   PARM                    name
C                   ENDACT
```

図 67. iSeries 400 プログラムの呼び出し

注: iSeries サーバー上のプログラムにワークステーション・ファイルが入っている場合には、システムがこれをオープンしようとすると失敗します。リモート・システムの呼び出しコマンドは DDM サーバーを介して実行されるので、ワークステーション・データ管理には表示装置が認識されません。使用できるのは、iSeries サーバー上に表示装置の値をセッションの名前 (OMXxxxx) に設定し、装置の最大数パラメーターを 1 より大きい値に設定してワークステーション・ファイルを作成する方式です。これによって、パラメーターを iSeries サーバー・プログラムを渡すことができます。 ACQ ステートメントで明示的にセッションを獲得してはいけません。これでは矛盾が起こってエラーになります。また、ワークステーションをリブートしなければ終了できないデッドロックが起こるので、ワークステーション上で 5250 エミュレーター表示装置を獲得することもできません。

iSeries サーバーからのワークステーション・プログラムの開始

サーバーで実行中の RPG アプリケーションがあって、Windows ワークステーションで VisualAge RPG アプリケーションを開始したい場合には、 STRPCCMD コマンドを使用してください。

複数のプロシージャーの使用

複数のプロシージャーをコードできれば、モジュラー・アプリケーションをコードする機能は大幅に向上します。

VisualAge RPG プログラムは、1 つまたは複数のモジュールから構成されます。プロシージャーは、バインド呼び出しで呼び出すことができるコード部分です。VisualAge RPG には 2 種類のプロシージャーがあります: メイン・プロシージャーおよびサブプロシージャー。メイン・プロシージャーは、プログラムの入り口プロシージャーとして指定でき、最初に呼び出された時に制御を受け取るプロシージャーです。メイン・プロシージャーが作成されるのは EXE の作成時だけであることに注意してください。

サブプロシージャーは、メイン・ソース・セクションの後で指定されるプロシージャーです。サブプロシージャーは、主として次の点でメイン・プロシージャーとは異なります。

- サブプロシージャー内で定義された名前にはサブプロシージャーの外部からアクセスできません。
- 呼び出しインターフェースはプロトタイプ化されていなければなりません。
- サブプロシージャーへの呼び出しはバインド済みプロシージャー呼び出しでなければなりません。
- P、D、および C 仕様しか使用できません。

サブプロシージャーは、データ項目がローカルであるために、他のプロシージャーとの独立性を提供することができます。通常、ローカル・データ項目は自動ストレージに記憶され、このことはプロシージャー呼び出し間でローカル変数の値が保存されないことを意味します。

サブプロシージャーは別の機能を提供します。サブプロシージャーにはパラメーターを値で渡すことができ、サブプロシージャーは値を戻す式で呼び出すことができます。

プロトタイプ呼び出し

サブプロシージャーを呼び出すためには、プロトタイプ呼び出しを使用しなければなりません。プロトタイプ呼び出しを使用することによって、任意の言語で書かれたプログラムやプロシージャーを呼び出すことができます。プロトタイプ呼び出しは、コンパイル時にプロトタイプを使用することによって呼び出しインターフェースが検査される呼び出しです。プロトタイプは、呼び出しインターフェースの定義です。これには、次の情報が組み込まれています。

- 呼び出しはバインド (プロシージャー) か動的 (プログラム) か
- プログラムまたはプロシージャーをどのようにして見つけるか (外部名)
- パラメーターの数とプロパティー
- 渡さなければならないパラメーターと、任意に渡すパラメーター
- 戻り値（もしあれば）のデータ・タイプ (プロシージャーの場合)

プロトタイプは、プログラムまたはプロシージャーを正しく呼び出し、呼び出し側が正しいパラメーターを渡したことを確認するために使用されます。 291 ページの図 68 は、レコードの各種フィールド読み取り可能な形式に形式設定するプロシージ

ャー FmtCust のプロトタイプを示しています。これには 2 つの出力パラメーターがあります。

```
 * Prototype for procedure FmtCust  (Note the PR on definition
 * specification.)  It has two parameters.
D FmtCust         PR
D  Name                         100A
D  Address                      100A
```

図 68. FmtCust プロシージャーのプロトタイプ

定様式の出力フィールドを作成するために、FmtCust はプロシージャー NumToChar を呼び出します。 NumToChar には、値で渡されて文字フィールドを戻す数値入力パラメーターがあります。図 69 は NumToChar のプロトタイプを示しています。

```
 * Prototype for procedure NumToChar
 * The returned value is a character field of length 31.
D NumToChar          PR            31A
 * The input parameter is packed with 30 digits and 0 decimal
 * positions, passed by value.
D   NUMPARM                     30P 0   VALUE
```

図 69. NumToChar プロシージャーのプロトタイプ

プログラムまたはプロシージャーがプロトタイプの場合には、 CALLP または（戻り値を使用したい場合には）式の中で呼び出します。プロトタイプの名前に続くリスト内のパラメーターを渡しますたとえば、 *name (parm1 : parm2 : ...)*。

図 70 は、FmtCust の呼び出しを示しています。パラメーターの名前（図 68 に示されている）が call ステートメントに一致していないことを注意してください。プロトタイプのパラメーター名は文書化のためだけのものです。プロトタイプは、呼び出しインターフェースの属性の記述を助けます。実際の呼び出しパラメーターの定義は、プロシージャー自体の中で行なわれます。

```
C                   CALLP     FmtCust(RPTNAME : RPTADDR)
```

図 70. FmtCust プロシージャーの呼び出し

プロトタイプ呼び出しを使用して、以下を（同じ構文で）呼び出すことができます。

- 実行時にシステム上にあるプログラム
- 他のモジュール内のエクスポートされたプロシージャー
- 同じモジュールのサブプロシージャー

名前およびアドレスを正しく形式設定するために、FmtCust は NumToChar を呼び出してカスタマー番号を文字フィールドに変換します。FmtCust は戻り値を使用する必要があるので、NumToChar の呼び出しは式の中で行なわれます。 292 ページの図 71 は、この呼び出しを示しています。

```
     *-------------------------------------------------------------
     * CUSTNAME and CUSTNUM are formatted to look like this:
     *     A&P Electronics     (Customer number 157)
     *-------------------------------------------------------------
    C                   EVAL      Name = CUSTNAME + ' '
    C                               + '(Customer number '
    C                               + %trimr(NumToChar(CUSTNUM)) + ')'
```

図 71. NumToChar プロシージャーの呼び出し

上図のように、値を戻すプロシージャーを使用することによって、必要な任意のユーザー定義機能を作成することができます。さらに、プロトタイプ呼び出しインターフェースの使用によって、パラメーター引き渡し用の多数のオプションが利用できます。

- プロトタイプ・パラメーターは、参照や値（プロシージャーの場合のみ）によったり、読み取り専用の参照によるなどのいくつかの方法で渡すことができます。RPG の場合のデフォルトは参照による受け渡し方式です。しかし、値や読み取り専用の参照による受け渡しでは、パラメーターの受け渡しにさらに多くのオプションが提供されます。
- プロトタイプを所定のパラメーターに使用できることが示されれば、次の 1 つまたは複数を行なうことができます。
  - *OMIT の受け渡し
  - パラメーター全体の除外
  - 指定されたものより短いパラメーターの受け渡し（文字およびグラフィックス・パラメーターと配列パラメーターの場合）

プロシージャーについての考慮事項

- メイン・プロシージャーに戻り値を定義することはできません。パラメーターは値で渡さなければなりません。
- メイン・プロシージャーは EXE 内に入っているだけです。そのパラメーターは値で渡すように指定してください。
- どのような演算命令もサブプロシージャーでコーディングすることができます。しかし、すべての入出力仕様をメイン・ソース・セクションで定義するように、ファイルはすべてグローバルに定義しなければなりません。同様に、すべてのデータ域もメイン・プロシージャーで定義しなければなりませんが、それらはサブプロシージャーでも使用することができます。
- 制御仕様は、全モジュールを制御するので、メイン・ソース・オプションでしかコーディングできません。
- サブプロシージャーは反復して呼び出すことができます。 それぞれの反復呼び出しによって、新しいプロシージャー呼び出しが呼び出しスタックに入れられます。新しい呼び出しは自動ストレージのすべてのデータ項目に対して新しいストレージを持ちます。このストレージはローカルであるため、他の呼び出しに使用することはできません。(自動ストレージを使用するサブプロシージャーで定義されたデータ項目は、定義に STATIC が指定されていない限り、自動ストレージを使用します。)

  前の呼び出しに関連付けられている自動ストレージは、後の呼び出しでは影響を受けません。すべての呼び出しは同じ静的ストレージを共用するので、後の呼び出しは静的ストレージに変数で保持された値に影響を及ぼすことがあります。

- サブプロシージャー内の例外処理は、サブプロシージャーにデフォルトの例外処理プログラムがないので、本質的にはメイン・プロシージャーのものとは異なります。メイン・プロシージャーでデフォルトの処理プログラムが呼び出されるような状態では、サブルーチンが異常終了します。
- VisualAge RPG プロシージャー名は大文字です。これらのプロシージャーを呼び出す場合には、大文字小文字がプロシージャーの大文字小文字と一致するようにしてください。

プロシージャー関連

プログラマーとしてユーザーは、可能な次の 3 つのターゲット・オブジェクトを作成するオプションを持っています。

- VisualAge RPG DLL (には GUI 命令コードが入っています。)
- GUI 命令コードを含まない RPG サブプロシージャーだけが入っているユーティリティー DLL。
- GUI 命令コードを含まない RPG EXE。

VisualAge RPG DLL についての考慮事項

- VisualAge RPG DLL サブプロシージャーは外部化されていません。

  これらのサブプロシージャーは、コンパイラーによって内部専用として指示されています。入り口点は他のモジュールに対して外部化されていません。これらのサブプロシージャーにリンクしようとすれば、リンク・ステップが失敗することになります。
- EXPORT キーワードは、プロシージャー仕様では使用できません。プロシージャーは VisualAge RPG DLL からエクスポートできないためです。

ユーティリティー DLL についての考慮事項

この DLL は、制御仕様でキーワード NOMAIN が提供された時にビルドされます。

コンパイラーは、コンパイルの結果として DLL と LIB ファイルの両方を作成します。LIB ファイルには、開始 P 仕様に EXPORT キーワードを持つすべてのプロシージャーが入ります。LIB ファイルにより、DLL が入っているサブプロシージャーにリンクすることができます。

- この DLL はプロシージャーだけで構成されます。

  すべてのサブルーチン (BEGSR) はプロシージャーに対してローカルでなければなりません。
- ソースで使用できる GUI 命令コードはありません。

  これには、START、STOP、SETATR、GETATR、%SETATR、%GETATR、SHOWWIN、CLSWIN および READS が含まれます。DSPLY は使用できますが、これの入ったプロシージャーが VisualAge RPG DLL から呼び出されると、DSPLY 命令コードは何もしません。
- *inzsr および *termsr は使用できません。
- *ENTRY パラメーターは使用できません。
- 例外処理は、次の点で VisualAge RPG DLL とは異なります。
  - 呼び出し元がユーティリティー DLL に入っていない場合には、例外についての情報が呼び出し元に通知されません。

– ユーザーがユーティリティー DLL で例外を処理するために推奨される方式は、エラー標識を持つか、またはそれぞれのルーチンで呼び出し元に適切な戻りコードを返すローカル *PSSR を持つ方式です。
– プロシージャーに例外が起こった時にデフォルトの例外処理プログラムは呼び出されないので、デフォルトの例外処理プログラムはユーティリティー DLL からは呼び出されません。ユーティリティー DLL で例外が起こってエラー標識または *PSSR がないと、終了() が実行されて、例外についての情報が FVDCERRS.LOG ファイルに書き出されます。

EXE についての考慮事項

- EXE は、制御仕様でキーワード EXE が指定されている時にビルドされます。
- EXE はプロシージャーだけで構成されます。

  すべてのサブルーチン (BEGSR) はプロシージャーに対してローカルでなければなりません。EXE には、名前がソース・ファイルの名前と一致するプロシージャーが入っていなければなりません。これは EXE の主入り口点 (すなわち、メイン・プロシージャー) となります。
- ソースで使用できる GUI 命令コードはありません。

  これには、START、STOP、SETATR、GETATR、%SETATR、%GETATR、SHOWWIN、CLSWIN および READS が含まれます。DSPLY は使用できます。
- *inzsr および *termsr は使用できません。
- *ENTRY パラメーターは許可されません。

  入り口点パラメーターがある場合には、それらはメイン・プロシージャーのパラメーター定義に指定され、それらは VALUE (各パラメーターごとに VALUE キーワードを指定しなければならない) によって渡されなければなりません。
- 開始 P 仕様では EXPORT キーワードは使用できません。
- 例外処理は VRPG DLL とは異なります。 EXE からはデフォルトの例外処理プログラムは呼び出されません。 EXE で例外が起こってエラー標識または *PSSR がない場合には、終了() が実行されて、例外についての情報が FVDCERRS.LOG ファイルに書き出されます。

第 18 章 VisualAge RPG プログラムからの Java メソッドの呼び出し

この項では、Java ソース・コードに変換された VARPG プログラムから Java メソッドを呼び出す方法およびそれをサポートする VARPG 言語への付加機能について説明します。Java メソッドを呼び出すためには、VARPG コンパイラーに次の情報が必要です。

- メソッドの名前
- メソッドが入っているクラス
- メソッドがオブジェクトを戻す場合には、戻されるオブジェクトのクラス
- メソッドが静的メソッドであるかどうか
- メソッドに渡されるパラメーターのデータ・タイプ

さらに、メソッドが静的メソッドでない場合には、メソッドを呼び出すために、オブジェクトがインスタンス化されなければなりません。メソッドがオブジェクトを戻す場合には、コンパイラーにはそのオブジェクトを保管する場所が必要です。メソッドがオブジェクトをパラメーターとして受け入れる場合には、そのオブジェクトを作成する何らかの方法が必要です。

以上の要件により、VARPG 言語に以下が追加されています。

- オブジェクトのデータ・タイプ
- CLASS キーワード
- EXTPROC キーワードの拡張

オブジェクトのデータ・タイプおよび CLASS キーワード

オブジェクトを保管できるフィールドは、**O** データ・タイプを使用して宣言されます。タイプ O のフィールドを宣言するには、D 仕様の 40 桁目に O をコーディングし、CLASS キーワードを使用してオブジェクトのクラスを指定します。 CLASS キーワードは 2 つのパラメーターを受け入れます。

```
CLASS(*JAVA:class_name)
```

*JAVA は、オブジェクトを Java オブジェクトとして識別します。Class_name は、オブジェクトのクラスを指定します。これは文字リテラルで、クラス名は完全修飾されなければなりません。クラス名は大文字・小文字が区別されます。

たとえば、タイプ BigDecimal のオブジェクトを保持するフィールドを宣言するには:

```
D bdnum                 S               O   CLASS(*JAVA:'java.math.BigDecimal')
```

タイプ String のオブジェクトを保持するフィールドを宣言するには:

```
D string                S               O   CLASS(*JAVA:'java.lang.String')
```

両方のクラス名が完全修飾され、大文字小文字が Java クラスと完全に一致することに注意してください。

タイプ O のフィールドは、データ構造のサブフィールドとして定義することはできません。タイプ O フィールドの配列を使用することはできますが、タイプ O のテーブルは実行時に事前ロードする必要があるので、使用することはできません。

次のキーワードを CLASS キーワードと一緒に使用することはできません。

```
ALIGN, ALT, ASCEND, BASED, BUTTON, CLTPGM, CONST, CTDATA, DATFMT,
DESCEND, DTAARA, EXTFLD, EXTFMT, EXTNAME, FROMFILE, INZ, LINKAGE,
MSGDATA, MSGNBR, MSGTEXT, MSGTITLE, NOOPT, NOWAIT, OCCURS, OPTIONS,
OVERLAY, PACKEVEN, PERRCD, PREFIX, PROCPTR, STYLE, TIMFMT, TOFILE,
VALUE, VARYING
```

Java メソッドのプロトタイピング

サブプロシージャーと同様に、Java メソッドは、正しく呼び出すためにプロトタイピングされなければなりません。VARPG コンパイラーは、メソッドの名前、それが属するクラス、パラメーターのデータ・タイプと戻される値のデータ・タイプ（もしあれば）、およびメソッドが静的メソッドであるかどうかを認識しなければなりません。

拡張された EXTPROC キーワードは、メソッドの名前およびそれが属するクラスを指定するために使用できます。Java メソッドをプロトタイピングする場合には、必要な EXTPROC キーワードの形式は次のようになります。

```
EXTPROC(*JAVA:class_name:method_name | *JAVARPG:class_name:method_name)
```

*JAVARPG は、メソッドを VARPG 生成 Java メソッドとして識別します。*JAVA は、メソッドを VARPG 生成でなく、最初に Java で書かれたコードから生成された Java メソッドとして識別します。*JAVARPG から生成されたメソッドでは、通常は Java で参照によって渡すことができない特定のデータ・タイプを参照によって渡すことができるので、この相違は重要です。これにより、Windows をターゲットにする場合および Java ソース・コードを生成する場合に、同じソース・コードを使用することができます。

クラス名とメソッド名は両方とも文字リテラルでなければなりません。クラス名は完全修飾された Java クラス名でなければならず、大文字小文字が区別されます。メソッド名は呼び出されるメソッドの名前でなければならず、大文字小文字が区別されます。

EXTPROC キーワードの拡張形式を使用できるのは、Java メソッドを呼び出す場合だけです。Windows をターゲットにしている場合には、EXTPROC キーワードのこの形式を使用するとコンパイラー・エラーになります。

パラメーターのデータ・タイプおよびメソッドの戻り値のデータ・タイプは、サブプロシージャーをプロトタイピングする時と同様に指定します。これが該当しないのは、データ・タイプが実際には Java データ・タイプにマップされることだけです。コンパイラーは、VARPG データ・タイプを次のように Java データ・タイプにマップします。

Java データ・タイプ	VARPG データ・タイプ
char[]	グラフィックまたはユニコード
boolean	標識 (N)

Java データ・タイプ	VARPG データ・タイプ
byte[]	英字 (任意の長さの A)
byte	整数 (3I)
int	整数 (10I)
short	整数 (5I)
long	整数 (20I)
float	浮動 (4F)
double	浮動 (8F)
任意のオブジェクト	オブジェクト (O)

ゾーン、パック、2 進数、および符号なしデータ・タイプは Java では使用できません。ゾーン、パック、2 進数、または符号なしフィールドをパラメーターとして渡すと、コンパイラーは適切な変換を行いますが、ほとんどが切り捨てや精度に欠けるものとなります。

呼び出そうとしているメソッドが VARPG 生成メソッドで、*JAVARPG が EXTPROC キーワードの最初のパラメーターとして指定されている場合には、パック、ゾーン、2 進数、および符号なしデータ・タイプは、パラメーターおよび戻される値のデータ・タイプとして指定することができます。最初に Java で書かれたコードから生成されたメソッドは、パック、ゾーン、2 進数、および符号なしデータ・タイプを、パラメーターまたは戻り値のプロトタイプで使用することはできません。

メソッドを呼び出す時には、コンパイラーは、パラメーターが DIM キーワードを使用してプロトタイピングされていれば配列をパラメーターとして受け入れます。そうでない場合にはスカラー・フィールド、データ構造、またはテーブルだけが受け入れられます。

現在は、次の Java データ・タイプを必要とするか、または次の値を戻すメソッドを呼び出すことはできません。すなわち、byte、char、および long です。

メソッドの戻り値がオブジェクトの場合には、プロトタイプで CLASS キーワードをコーディングして、オブジェクトのクラスを指定しなければなりません。指定されたクラス名は、戻されるオブジェクトのクラス名となります。呼び出されるメソッドのクラスを指定するには、EXTPROC キーワードを使用してください。

呼び出されるメソッドが静的メソッドの場合には、プロトタイプで STATIC キーワードを指定する必要があります。

Java では、次のデータ・タイプは値で渡すことができるだけです。

```
byte
int
short
long
float
double
```

これらのタイプのパラメーターには、プロトタイプで VALUE キーワードが指定されていなければなりません。

呼び出そうとしているメソッドが VARPG 生成メソッドの場合で、EXTPROC キーワードの最初のパラメーターとして *JAVARPG が指定されている場合には、これらのデータ・タイプは参照によって渡すことができ、VALUE キーワードは不要です。

オブジェクトは参照によってしか渡せないことに注意してください。タイプ O で VALUE キーワードを指定することはできません。配列は Java によってオブジェクトとして見られるので、配列へのパラメーター・マッピングも参照によって渡さなければなりません。これにはバイト配列が含まれます。

Java メソッドのプロトタイピングの例

この項では、Java メソッドのプロトタイピングの例を示します。

例 1

Java 整数クラスには、*toString* と呼ばれる静的メソッドが入っていて、これは *int* パラメーターを受け入れてストリング・オブジェクトを戻します。これは Java で次のように宣言されます。

```
String  Integer.toString(int)
```

このメソッドは次のようにプロトタイピングされます。

```
D tostring            PR              O   EXTPROC(*JAVA:
D                                          'java.lang.Integer':
D                                          'toString')
D                                          CLASS(*JAVA:'java.lang.String')
D                                         STATIC
D    num                              10I 0 VALUE
```

EXTPROC キーワードは、メソッドを非 VARPG 生成メソッドとして識別します。また、メソッド名が 'toString' で、クラス 'java.lang.Integer' にあることも示しています。

40 桁目の O と CLASS キーワードは、コンパイラーにこのメソッドがオブジェクトを戻し、そのオブジェクトのクラスが 'java.lang.String' であることを指示しています。

STATIC キーワードは、このメソッドが静的メソッドで、メソッドの呼び出しに整数オブジェクトは不要であることを示しています。

パラメーターのデータ・タイプは 10I として指定され、これは Java *int* データ・タイプにマップされます。パラメーターが int であるため、値で渡される必要があり、VALUE キーワードが必要です。

例 2

Java 整数クラスには、*getInteger* と呼ばれる静的メソッドが入っていて、これはストリングおよび整数オブジェクトをパラメーターとして受け入れて、整数オブジェクトを戻します。これは Java で次のように宣言されます。

```
Integer Integer.getInteger(String, Integer)
```

このメソッドは次のようにプロトタイピングされます。

```
D getint               PR              O   EXTPROC(*JAVA:
D                                           'java.lang.Integer':
D                                           'getInteger')
D                                          CLASS(*JAVA:'java.lang.Integer')
D                                          STATIC
D   string                                 O   CLASS(*JAVA:'java.lang.String')
D   num                                    O   CLASS(*JAVA:'java.lang.Integer')
```

このメソッドはパラメーターとして 2 つのオブジェクトを受け入れます。O は D 仕様の 40 桁目にコーディングされ、CLASS キーワードはそれぞれのオブジェクト・パラメーターのクラスを指定します。

例 3

Java 整数クラスには *shortValue* と呼ばれるメソッドが入っていて、これはメソッドの呼び出しに使用される整数オブジェクトの短い表記を戻します。これは Java で次のように宣言されます。

```
short shortValue()
```

このメソッドは次のようにプロトタイピングされます。

```
D shortval             PR            5I 0 EXTPROC(*JAVA:
D                                           'java.lang.Integer':
D                                           'shortValue')
```

このメソッドは静的メソッドでないため、STATIC キーワードは指定されません。このメソッドはパラメーターを取りません。

戻される値は 5I として指定され、これは短縮データ・タイプにマップされます。

例 4

Java 整数クラスには *equals* と呼ばれるメソッドが入っていて、これはパラメーターとしてオブジェクトを受け入れてブールを戻します。これは Java で次のように宣言されます。

```
boolean equals(Object)
```

このメソッドは次のようにプロトタイピングされます。

```
D equals               PR              N   EXTPROC(*JAVA:
D                                           'java.lang.Integer':
D                                           'equals')
D  obj                                 O   CLASS(*JAVA:'java.lang.Object')
```

戻される値は N として指定され、これは Java ブール・データ・タイプにマップされます。

オブジェクトの作成

非静的メソッドを呼び出すためには、オブジェクトが必要です。オブジェクトのクラスは、メソッドが入っているクラスと同じでなければなりません。オブジェクトは、クラス・コンストラクターを呼び出して、インスタンス化され、あるいは作成されます。クラス・コンストラクターは静的メソッドではありませんが、呼び出しにオブジェクトを必要としません。コンストラクターをプロトタイピングする場合には、特殊なメソッド名 *CONSTRUCTOR が使用されます。

たとえば、浮動値から *BigDecimal* オブジェクトを構成するためには、次のように浮動パラメーターを必要とするコンストラクターを呼び出す必要があります。

```
BigDecimal(float)  returns a new BigDecimal object
```

コンストラクターは次のようにプロトタイピングされます。

```
D bdcreate             PR              O   EXTPROC(*JAVA:
D                                           'java.math.BigDecimal':
D                                            *CONSTRUCTOR)
D                                           CLASS(*JAVA:'java.math.BigDecimal')
D    dnum                              4F   VALUE
```

Java 浮動データ・タイプにマップされるので、値によってパラメーターを渡す必要があることに注意してください。

Java メソッドの呼び出し

Java メソッドは、既存の命令コード CALLP（戻り値が必要ない場合）および EVAL（戻り値が必要な場合）を使用して呼び出すことができます。新しい構文は不要です。

静的メソッドを呼び出す場合には、呼び出しを行うためにオブジェクトは必要ありません。非静的メソッドを呼び出す場合には、オブジェクトが必要です。使用するオブジェクトは、その呼び出しの最初のパラメーターとしてコーディングしなければなりません。このパラメーターは、プロトタイプでは指定されませんが、静的以外のすべてのメソッドでは暗黙に指定されます。このことは、静的でないメソッドを呼び出す場合には、少なくとも 1 つのパラメーターを指定する必要があることを意味しています。

例 1

この例では、目標は 2 つの BigDecimal 値を一緒に加えることにあります。これを行うためには、BigDecimal クラスのコンストラクターを呼び出して 2 つの BigDecimal オブジェクトをインスタンス化し、BigDecimal オブジェクトを保管するフィールドを宣言して、BigDecimal クラスの add() メソッドを呼び出す必要があります。

```
 *
 *   Prototype the BigDecimal constructor that accepts a String
 *   parameter.  It returns a new BigDecimal object.
 *
D bdcreate1            PR              O   EXTPROC(*JAVA:
D                                           'java.math.BigDecimal':
D                                            *CONSTRUCTOR)
D                                           CLASS(*JAVA:'java.math.BigDecimal')
D    str                               O   CLASS(*JAVA:'java.lang.String')
 *
 *   Prototype the BigDecimal constructor that accepts a double
 *   parameter.  8F maps to the Java double data type and so must
 *   be passed by VALUE.  It returns a BigDecimal object.
 *
D bdcreate2            PR              O   EXTPROC(*JAVA:
D                                           'java.math.BigDecimal':
D                                            *CONSTRUCTOR)
D                                           CLASS(*JAVA:'java.math.BigDecimal')
D   double                             8F   VALUE
 *
```

```
 *   Define fields to store the BigDecimal objects.
 *
D bdnum1            S                 O    CLASS(*JAVA:'java.math.BigDecimal')
D bdnum2            S                 O    CLASS(*JAVA:'java.math.BigDecimal')
 *
 *
 *   Since one of the constructors we are using requires a String object,
 *   we will also need to construct one of those.  Prototype the String
 *   constructor that accepts a byte array as a parameter.  It returns
 *   a String object.
 *
D makestring           PR             O   EXTPROC(*JAVA:
D                                          'java.lang.String':
D                                           *CONSTRUCTOR)
D                                          CLASS(*JAVA:'java.lang.String')
D    bytes                          10A
 *
 *  Define a field to store the String object.
 *
D string            S                 O    CLASS(*JAVA:'java.lang.String')
 *
 *  Prototype the BigDecimal add method.  It accepts a BigDecimal object
 *  as a parameter, and returns a BigDecimal object (the sum of the parameter
 *  and of the BigDecimal object used to make the call).
 *
D add               PR                O    EXTPROC(*JAVA:
D                                           'java.lang.BigDecimal':
D                                           'add')
D                                          CLASS(*JAVA:'java.math.BigDecimal')
D   bd1                               O    CLASS(*JAVA:'java.math.BigDecimal')
 *
 *  Define a field to store the sum.
 *
D sum               S                 O    CLASS(*JAVA:'java.math.BigDecimal')
D
D double            S                 8F   INZ(1.1)
D fld1              S                10A
```

呼び出しを行うコードは次のとおりです。

```
C                       MOVEL     'mystring'    fld1            10
     C*
C*  Call the constructor for the String class, to create a String
C*  object from fld1.  Since we are calling the constructor, we
C*  do not need to pass a String object as the first parameter.
C*
C                       EVAL      string = makestring(fld1)
C*
C*  Call the BigDecimal constructor that accepts a String
C*  parameter, using the String object we just instantiated.
C*
C                       EVAL      bdnum1 = bdcreate1(string)
C*
C*  Call the BigDecimal constructor that accepts a double
C*  as a parameter.
C*
C                       EVAL      bdnum2 = bdcreate2(double)
C*
C*  Add the two BigDecimal objects together by calling the
C*  add method.  The prototype indicates that add accepts
C*  one parameter, but since add is not a static method, we
C*  must also pass a BigDecimal object in order to make the
C*  call, and it must be passed as the first parameter.
C*  bdnum1 is the object we are using to make the
```

```
C*  call, and bdnum2 is the parameter.
C*
C                       EVAL      sum = add(bdnum1:bdnum2)
C*  sum now contains a BigDecimal object with the value
C*  bdnum1 + bdnum2.
```

例 2

この例では、VARPG %TRIM 表示の代わりに trim() メソッドを使用して、Java で TRIM を実行する方法を示します。ストリング・クラスの trim() メソッドは静的メソッドではないので、これを呼び出すためにはストリング・オブジェクトが必要です。

```
 *
 *  Define a field to store the String object we wish to trim
 *
D str                 S               O   CLASS(*JAVA:'java.lang.String')
 *
 *  Prototype the constructor for the String class.  The
 *  constructor expects a byte array.
 *
D makestring          PR              O   EXTPROC(*JAVA:
D                                          'java.lang.String':
D                                           *CONSTRUCTOR)
D                                          CLASS(*JAVA:'java.lang.String')
D   parm                              10A
D
 *
 *  Prototype the String method getBytes which converts a String to a byte
 *  array.  We can then store this byte array in an alpha field.
 *
D makealpha           PR            10A   EXTPROC(*JAVA:
D                                          'java.lang.String':
D                                          'getBytes')
 *
 *  Prototype the String method trim.  It doesn't take any parameters,
 *  but since it is not a static method, must be called using a String
 *  object.
 *
D trimstring          PR              O   EXTPROC(*JAVA:
D                                          'java.lang.String':
D                                          'trim')
 *
D fld               S                 10A     INZ('   hello    ')
```

呼び出しは次のようにコーディングされます。

```
     C*
C* Call the String constructor
C*
C                       EVAL      str = makestring(fld)
C*
C* Trim the string by calling the String trim() method.
C* We will reuse the str field to store the result.
C*
C                       EVAL      str = trimstring(str)
C*
C* Convert the string back to a byte array and store it
C* in fld.
C*
C                       EVAL      fld = makealpha(str)
```

静的メソッドは、呼び出しを行うためにオブジェクトが必要ないことを除けば、同じように呼び出されます。上記のmakealpha() メソッドが静的の場合には、呼び出しは次のようになります。

```
C                       EVAL      fld = makealpha()
```

このメソッドが値を戻さない場合には、CALLP 命令コードを使用してください。

追加の考慮事項

コンパイラーは、コンパイル時にクラスを解決しようとしません。実行時にクラスが見つからないと、実行時エラーが起こります。これは、Java 環境から *UnresolvedLinkException* オブジェクトが受け取られたことを示します。

コンパイラーは、コンパイル時にパラメーターのタイプ検査を行いません。プロトタイプと呼び出されるメソッドの間に矛盾があると、実行時にエラーを受け取ることになります。

呼び出されるメソッドが VARPG 生成メソッドである場合には、EXTPROC の最初のパラメーターとして *JAVARPG を指定することが非常に重要です。これを行わないと、上記の 2 つのエラー状態のいずれかが起こる可能性があります。

第 19 章 Java でのコンパイル時の考慮事項

このセクションでは、VARPG ソースの制限、VARPG ソースで必要と考えられる変更、および Java ソースを作成するために Java 作成オプションを使用する時の実行時の動作の違いについて説明します。

プロジェクト・ファイル命名規則

Java アプリケーションのプロジェクト・ファイル名は、Java 命名規則に従っていなければなりません。最初の文字は英字でなければなりません。プロジェクトの名前が正しくない場合は、*プロジェクトの名前変更* ユーティリティーを使用して名前を変更することができます。 (プロジェクトのアイコンのポップアップ・メニューから**プロジェクトの名前変更**を選択してください。)

条件付きコンパイル指示

2 つの条件付きコンパイラー指示がコンパイラーによって定義されていて、これらは Windows コンポーネントと Java ソース・コードの両方を作成するために使用できる単一のソース・ファイルを保守するのに役立ちます。それらの指示は次の通りです:

- **COMPILE_WINDOWS** は、Windows のビルドが要求された時にコンパイラーによって定義されます。
- **COMPILE_JAVA** は、Java のビルドが要求された時にコンパイラーによって定義されます。

コンパイラーがこれらの 2 つの名前を定義するので、/DEFINE 指示を使用してそれらを定義する必要はありません。

Java ソース・コードの制約事項

次の言語エレメントは、Java ソース・コードを生成する時はサポートされません:

キーワード:

- ALIGN
- EXPROPTS
- フィールド定義における STATIC。STATIC は Java メソッド・プロトタイプでサポートされます。

命令コード:

- ALLOC
- CABxx
- CALLB
- DEALLOC
- DSPLY (NOMAIN および EXE の場合のみ、その他の場合はサポートされる)
- GOTO
- REALLOC

- TAG

命令拡張:
- M
- R

言語エレメント:
- 組み込み SQL

データ・タイプ:
- ポインター・データ・タイプ

ファイル・タイプ:
- SPECIAL

ファイル操作:
- 相対レコード番号によるレコードの書き込み

可能な VARPG ソースの変更

このセクションでは、Java ソース・コードを生成するために VARPG ソースで必要となる可能性のある変更について要約します。

1. コンパイラーがサブフィールド定義を妥当性検査できるようにするために、PSDS および INFDS のサブフィールドを定義する時は、to/from 表記を使用しなければなりません。INFDS および PSDS のサブフィールドの定義は、VARPG 言語解説書に指定されている定義と一致していなければなりません。一致していない場合は、コンパイル時エラーが出されます。
2. ループ中の最後の命令以外に、無条件 LEAVE または ITER 命令があってはなりません。それがある場合は Java コンパイラーはエラーを出します。ループ中に無条件 LEAVE またはITER がある場合には、ループ中のそれ以降のすべての命令が決して実行されないように、それらを削除しなければなりません。
3. 日付 / 時刻 / タイム・スタンプの期間を加算および減算する時は、maxint (2 147 483 647) および -maxint (-2 147 483 648) の間の値しか使用できません。
4. Java では int (10I)、short (5I)、float (4F)、および double (8F) 値を参照によって受け渡しすることができないので、Java に変換されるサブプロシージャーに対してこれらの機能性を保存するためには、Java コードを VARPG で生成しなければなりません。これを達成するために生成されたコードは、VARPG ソースが複数のリターン・ポイントのあるサブプロシージャーを含んでいて、参照によって受け渡された整数または浮動パラメーターを受け取る場合に、Java コンパイラー・エラーの原因となることがあります。

   Java コンパイラー・エラーの原因となることのあるサンプル・コードは次の通りです:

```
C                   IF        x = 1
C                   ...
C                 RETURN      1
C                   ELSE
C                   ...
C                   RETURN    0
C                   ENDIF
```

   上記のコードは次のように変更しなければなりません:

```
C                   IF        x = 1
C                   ...
C                 RETURN     1
C                   ELSE
C                   ...
C                   ENDIF
C                   RETURN    0
```

5. 文字 '*'、'#'、および '@' は Java ID には使用できません。そのために、VARPG 名の中で使用されたすべての '*'、'#'、および '@' は '_' に変更されます。この変換によって重複した名前ができる可能性があります。
6. ファイルをもたないアプリケーションの中で COMMIT または ROLBK 命令がコーディングされている場合には、重大度 30 のメッセージ (RNF7833) が出されます。
7. ローカル *PSSR が Java に変換される方法のために、ローカル *PSSR を呼び出すことはできません。また、GOTO がサポートされていないので、ローカル *PSSR をそのままにして、デフォルト・ハンドラーを避ける唯一の方法は RETURN 命令をコーディングすることであることに注意してください。
8. 論理式の短絡はありません。これは、複合論理式が実行される順序が信頼できないことを意味します。
9. 可変長フィールドは Java への変換時にクラスとしてインプリメントされます。すなわち、それらは VARPG 言語解説書に文書として保管されないということです。一定の方法で保管されるそれらのものに依存するコードは機能しません。
10. 初期値が提供されていない場合には、データ構造サブフィールドはブランクに初期設定されませんが、サブフィールドのデータ・タイプに応じてデフォルトの値に初期設定されます。デフォルト値は、数値の場合は 0、文字の場合はブランクで、日付、時刻、タイム・スタンプの場合は *LOVAL です。可変長フィールドの長さは 0 に設定されます。
11. *HIVAL および *LOVAL 値はグラフィックおよび UCS-2 フィールドでは使用できません。
12. データ構造に長さを指定する場合には、含まれるサブフィールドの合計長と一致する長さでなければなりません。そうでない場合には、コンパイラーは重大度 30 の診断メッセージを出します。
13. サブルーチンをサブプロシージャー内で定義することはできません。唯一の例外として、*PSSR はサブプロシージャー内で定義できます。サブプロシージャー内のすべてのサブルーチンは、サブプロシージャー外に移動しなければなりません。サブルーチンがサブプロシージャー内のローカル・フィールドにアクセスする場合は、そのフィールドをグローバル・フィールドに変更するか、あるいはサブルーチンを、ローカル・フィールドをパラメーターとして受け入れるサブプロシージャーに変更する必要があります。
14. DO ループ内の無条件 LEAVE ステートメントはサポートされません。この状態が存在する場合には、Java コンパイラー・エラーが起こります。DO ループ内の無条件 LEAVE は、ループが 1 回しか実行されないことを意味するので、LEAVE を除去し、コードを変更してループ命令コードを除去しなければなりません。
15. イベント属性を固定した複合条件ステートメントで使用すると、現在は Java コンパイル・エラーの原因となります。代わりに、同等のフリー・フォームの式を使用しなければなりません。

Java コンパイラー・エラーの原因となることのあるサンプル・コードは次の通りです:

```
C     %mousex       IFEQ      x
C     %mousey       ANDEQ     y
C                   ...
C                   ENDIF
```

上記のコードは次のように変更しなければなりません:

```
C                   IF        %mousex = x AND
C                             %mousey = y
C                   ...
C                   ENDIF
```

16. 無条件 RETURN 命令は、それがユーザー・サブルーチン、アクション・サブルーチン、またはサブプロシージャーの中の最後のステートメントでない限り、コーディングすることはできません。そうでない場合は、Java コンパイラーがエラーを報告する可能性があります。
17. 無条件 LEAVESR 命令は、それがユーザー・サブルーチンまたはアクション・サブルーチンの中の最後のステートメントでない限り、コーディングすることはできません。そうでない場合は、Java コンパイラーがエラーを報告する可能性があります。
18. SELECT ステートメントは、サブプロシージャーの中にある場合、RETURN 命令を含んでいる場合、およびサブプロシージャーのメイン・ボディ内にコーディングされている場合は、Java コンパイラー・エラーの原因となることがあります。

    Java コンパイラー・エラーの原因となることのあるサンプル・コードは次の通りです:

    ```
    C                   SELECT
    C     x             WHENEQ    y
    C                   RETURN    1
    C     x             WHENEQ    z
    C                   RETURN    2
    C                   OTHER
    C                   RETURN    0
    C                   ENDSL
    ```

    上記のコードは次のように変更しなければなりません:

    ```
    C                    SELECT
    C     x             WHENEQ    y
    C                   RETURN    1
    C     x             WHENEQ    z
    C                   RETURN    2
    C                   ENDSL
    C                   RETURN    0
    ```

    一般的に、RETURN 命令はサブプロシージャーの考えられるすべてのコード・パスに対してコーディングしなければならず、そうでない場合は、Java コンパイラーがエラーを報告する可能性があります。
19. 配列を値によってサブプロシージャーに渡すことはできません。

ランタイムの違い

Windows と Java 環境の違いにより、Java でのアプリケーションの実行は Windows での実行と異なる場合があります。次の範囲が影響を受けます:

1. %SCAN 組み込み関数は整数の結果を戻します。Windows では、符号なし結果を戻します。
2. 数字の切り捨てオプションは信頼性がないので、それを当てにすべきではありません。
3. 入出力例外が起こった場合に、ユーザーはその操作を再試行するオプションが与えられません。
4. Java アプリケーションの実行時に、データ構造が 1 つの大きな文字フィールドとして扱われません。そのために、使い方によっては予期しない結果となる場合があります。
5. 2 進数、整数、および符号なしデータ・タイプの形式は、ローカル・ファイルによって異なる処理をされます。ローカル・ファイルの読み取りおよび書き込み時には、Java 形式が使用されます。すなわち、高順位バイトが左端となります。一方 Windows アプリケーションとして実行される時は、右端となります。
6. サブプロシージャーの例外処理は、アクション・サブルーチンの場合と同じ動作をします。ローカル *PSSR または INFSR がない場合、および操作のエラー標識がない場合には、デフォルト・エラー処理プログラムが呼び出されます。
7. ファイルとの間でフィールドの読み取りまたは書き込みを行っている時に無効な日付、時刻、タイム・スタンプ値が見つかった場合は、そのフィールドはデフォルト値 (*LOVAL) に設定されます。エラーは報告されません。
8. Java はタイム・スタンプの 3 桁のミリ秒部分しか処理できません。6 桁すべてのミリ秒部分を使用するタイム・スタンプ (すなわち、ミリ秒の形式が 000xxx ではない) で計算を行う時は、結果は予測通りにはならない可能性があります。
9. 式の中間結果は 30 桁に制限されていません。実際に、VARPG 環境で実行する時は、中間結果の精度に対して注意が払われません。
10. メモリーをコンポーネント相互間で共用することはできません。コンポーネントが、START を使用して別のコンポーネントによって開始された場合でも、渡されたパラメーターに対して行った変更はコンポーネント相互間にまたがって反映されることはありません。
11. 整数のオーバーフローまたはアンダーフローは報告されません。浮動オーバーフローまたはアンダーフローは状況 9999 として報告されます。
12. NOMAIN または EXE アプリケーションの中のサブプロシージャーの実行時にエラーが起こった場合で、エラー標識または *PSSR がない場合には、そのエラーは呼び出し側に戻され、呼び出し側によって処理されます。Windows で実行している時は、アプリケーションが終了します。
13. Java アプリケーションの実行時には、状況 50 のエラーが出されることはありません。 Java は、処理できない文字変換の診断メッセージを出しません。Java は、変換が正常に行われなかった場合に状況 100 を出すか、あるいは変換済みストリングの使用時に ArrayIndexOutOfBoundsException を出す場合があります。
14. ALWNULL(*NO) が指定されている時に、ホスト・ファイルをヌル値レコードに位置指定すると、結果は CPF5035 となります。

アプレットの制約事項

次の言語エレメントは VARPG アプレットの実行時はサポートされておらず、実行時に Java エラーとなります:

- 印刷装置ファイル
- ローカル・ファイル
- 呼び出し側の C 関数、外部サブプロシージャー、EXE。
- NOMAIN および EXE アプリケーションをアプレットとして実行することはできません。

J2SDK 1.2 印刷上の問題

Java 2 ソフトウェア開発キット (J2SDK)、バージョン 1.2 またはそれ以降は、現在、テキストを印刷装置に送信する時に問題があります。この問題を避ける 1 つの方法は、Java アプリケーションを次のように実行することです:

```
java -Djava2d.font.usePlatformFont=true -ms32m -mx32m <classname>
```

ただし、テキストは期待通りに印刷されない可能性があります。この問題は、J2SDK 1.2 の既存の問題が修正された時は、VARPG を更新しなくても解決されます。

第 20 章 VisualAge RPG アプレットの作成および実行

ビジュアル・インターフェースおよび関連の VARPG 論理をワークステーションに作成した後は、適切な Java 仮想マシン (JVM) を使用して Web ブラウザーで実行できる Java アプレットとして、アプリケーションをビルドおよび展開することができます。これによって、アプリケーションをインターネット上で広く使用できるようにするための格段の柔軟性が得られます。Web サイトにブラウザーでアクセスする多くのユーザーは、ブラウザー内でアプレットを実行することができ、iSeries サーバー上のデータと通信することもできます。

この章では、このような VARPG アプレットをビルドおよび展開する方法について説明します。

アプレットの作成

注: プロジェクトの Java バージョンをビルドするためには、Java 2 ソフトウェア開発キット (J2SDK)、バージョン 1.2 またはそれ以上がワークステーションにインストールされていなければなりません。J2SDK は次の URL にある Sun Microsystems から利用できます。

```
http://java.sun.com/products/
```

アプレットを実行するには、Java 2 実行時環境 (J2RE) の***国際*** バージョンがインストールされていなければなりません。

アプレットは、Web ページのコンテキスト内で実行される Java アプリケーションです。アプレットを含む Web ページをロードすると、そのアプレットのコードが HTTP サーバーからワークステーションにダウンロードされ、Java アプレットが始動します。一般的に、アプレットはメインの Web ページに組み込まれており、ブラウザーに Web ページが表示されると実行されます。アプレットを別のウィンドウに表示することもできます。

VARPG アプレットを作成するために必要な特別な設計または環境ステップはありません。VARPG プロジェクトの設計およびコーディングは、Windows アプリケーションに関する限り、どのアプレットでも同じです。ただし、長時間にわたるダウンロードを避けるために、アプレットとしてのターゲットを決める時に、***シン・クライアント*** を作成することを考慮することができます。 (詳細については、 459 ページの『付録 B. シン・クライアント・アプリケーションの書き込み』 を参照してください。).

アプレットには、気を付けなければならないセキュリティー上の制限があります。これらのセキュリティー上の制限は VisualAge RPG の制限ではありませんが、アプレットに対する Java 言語実行時指定の一部です。アプレットは次のことはできません。

- ファイル・システムおよびプリンターなどの、クライアント上のローカル・リソースへのアクセス

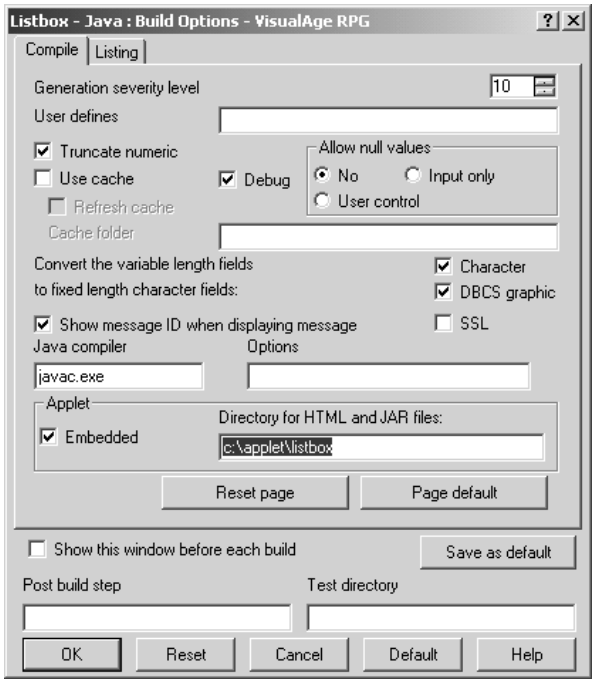
- アプレットが常駐するホストとは異なる別のホストへのソケット接続のオープン。すなわち、1 つの iSeries 400 サーバーからアプレットをロードして、別のサーバーのファイルにアクセスすることはできません。

ただし、これらのセキュリティー制限の一部を緩和するために、ポリシー・ファイルをセットアップすることができます。ポリシー・ファイルのセットアップについては、219 ページの『アプレット用のセキュリティー・ファイルの使用』 および Java 資料を参照してください。

アプレットを設計およびコーディングするためには、典型的な Windows アプリケーションを設計およびコーディングするステップと同じステップを使用してください。ただし、Java 環境でコーディングする時に適用される制限に注意してください。アプレットのビジュアル・インターフェースおよび関連 VARPG 論理を完了したら、アプリケーションをビルドします。

Java: ビルド・オプション・ノートブックでアプレットのビルド・オプションを制御することはできません。VisualAge RPG 設計ウィンドウから**プロジェクト > ビルド・オプション > Java** を選択して、プロジェクト用の Java ビルド・オプションを表示します。

図 72. リスト・ボックスのサンプル - Java: ビルド・オプション・ノートブック



設定値の大部分は、次の例外を除いて、Windows アプリケーションのビルドに使用するものと類似しています。

SSL iSeries サーバーと Java アプレットまたはアプリケーションの間のすべての TCP/IP 接続を、Secure Sockets Layer テクノロジーを使用して暗号化したい

場合には、SSL を選択します。 (SLL セットアップについては、 481 ページの『付録 D. Secure Sockets Layer (SSL) セットアップ』 を参照してください。)

Java コンパイラー
VisualAge RPG は、プロジェクトから Java ソース・ファイル (.java) を生成し、外部 Java コンパイラーを信頼して、ソースからクラス・ファイル (.class) を作成します。 IBM または Sun Microsystem の Java 2 SDK を使用しない場合には、ここで使用する Java コンパイラーを指定する必要があります。

オプション
必要なコマンド行オプションを Java コンパイラーに渡します。

アプレット - 組み込み
アプレットは、それが組み込まれている HTML ページが Web ブラウザーに表示された時に実行されるか、あるいはアプレットは外部ウィンドウで実行されるかを決定します。

アプレット - HTML および JAR ファイルのディレクトリー
アプレットに必要なすべての実行時ファイルを入れるディレクトリーを指定することができます。これらのファイルは、Java 用のプロジェクトをビルドする時に生成されます。デフォルトでは、これらのファイルはワークステーション上のプロジェクトのソース・ディレクトリーに入ります。

ヒント: ネットワーク・ドライブを iSeries サーバーにマップし、アプレットの配置元の IFS ディレクトリーを入力します。

アプレット用のオプションを構成しているので、プロジェクトをビルドする準備ができています。プロジェクトの設計ウィンドウから、**プロジェクト > ビルド > Java** を選択します。ビルドが成功すると、アプレットの実行時ファイルがプロジェクトのソース・ディレクトリーに作成されるか、あるいは ***HTML および JAR ファイルのディレクトリー*** Java オプションに指定されているディレクトリーに作成されます。

たとえば、Java アプリケーションを作成するためにリスト・ボックスのサンプルをビルドし、アプレットの実行時ファイルを入れるディレクトリーとして `c:\applet\Listbox` を指定した場合には、このディレクトリーに次のファイルが表示されるはずです。

```
listbox.htm
listbox_applet.htm
LISTBOX.jar
vapplet.jar
```

これらのファイルは、次のように、Web サーバーからアプレットを展開するために使用されます。

listbox.htm
Java プラグインを使用してアプレットを立ち上げます。

listbox_applet.htm
必要な VisualAge RPG Java 実行時 (varpg.jar) ファイルについて、ユーザーのワークステーションを検査します。ワークステーションに正しいバージ

ョンの実行時ファイルがある場合には、ブラウザーは ***listbox.htm*** ページをオープンします。そうでない場合は、ユーザーは正しい実行時ファイルをダウンロードしてインストールするようにプロンプトが出されます。

LISTBOX.jar
プロジェクトで使用される LISTBOX.class、LISTBOXApplet.class、LISTBOX.ODX、LISTBOX.RST および任意の *Resources.properties (アプリケーション用のメッセージを定義している場合) が含まれています。

vapplet.jar
Web サーバー上で必要な VisualAge RPG Java 実行時ファイルの小さいサブセットが含まれています。

アプレットのテスト

このセクションでは、VisualAge RPG アプレットをテストするのに必要なセットアップについて説明します。

1. IBM または Sun Microsystem の Java 2 Runtime Environment (J2RE) をインストールします。

   VisualAge RPG 生成のアプレットを正しく実行するためには、**国際**バージョンの Java 2 Runtime Environment (J2RE) が必要です。 IBM または Sun Microsystem の Java 2 SDK (または JRE) をインストールすると、Java プラグインが自動的インストールされます。開発マシンで実行する場合には、アプレットを開発するためにインストールした J2SDK で十分です。
2. VisualAge RPG Java 実行時 (varpg.jar) ファイルを JRE の拡張ディレクトリーに追加します。

   VisualAge RPG Java 実行時ファイルが各クライアント・マシンにコピーされ、ローカル JRE の拡張ディレクトリーに追加されるはずです。一般的に、これは `jre\lib\ext\` という名前のサブディレクトリーです。これにより、Web ページからアプレットを実行するたびに、HTTP サーバーから実行時ファイルをダウンロードする必要がなくなります。***varpg.jar*** ファイルは、WDSC インストール・ディレクトリーのもとの JAVA サブディレクトリーの中にあります。たとえば、`c:\wdsc\java\varpg.jar` です。
3. アプレットの実行時ファイルを iSeries IFS ディレクトリーにコピーし、ここからアプレットを実行します。リスト・ボックスのサンプルでは、これらのファイルは次のようになります。

   ```
   listbox.htm
   listbox_applet.htm
   LISTBOX.jar
   vapplet.jar
   ```

   ヒント: ネットワーク・ドライブを iSeries サーバーにマップし、アプレットを作成する前に、IFS ディレクトリーを***HTML および JAR ファイル用のディレクトリー*** Java ビルド・オプションに入力します。
4. iSeries HTTP サーバーをセットアップして、アプレットが入っているディレクトリーにアクセスできるようにします。

まだそのようにしていない場合には、IBM HTTP サーバーを始動して構成する必要があります。HTTP サーバーの構成については、*HTTP Server for iSeries Webmaster's Guide* を参照してください。

アプレット実行時ファイルを入れた IFS ディレクトリーへのアクセスを可能にする***PASS*** ステートメントを iSeries HTTP 構成ファイルに追加します。この例では、アプレット・ファイルは IFS ディレクトリー `/applets` にあります。したがって、次の PASS ステートメントを追加してください。

```
Pass /applets/* /applets/*
```

5. Web ブラウザーからアプレットを実行します。たとえば、次の URL で Web ブラウザーをオープンします。

   ```
   http://Toras14m:999/Listbox.htm
   ```

   ここで、`Toras14m` は iSeries サーバー名、`999` は HTTP ポート番号、そして `Listbox.htm` はアプレットが入っている Web ページです。

次は、Windows ブラウザー内で実行されるリスト・ボックス・アプレットです。

図 73. Internet Explorer 内で実行されるリスト・ボックス・アプレット

トラブルシューティング

以下は、アプレットが実行されない原因となることのある共通の構成上の問題をリストしたものです。

- 適切な J2RE がインストールされていない。Java プラグインとともに、Java 2 SDK または***国際***バージョンの J2RE をインストールしてください。

- アプレットの必須実行時ファイルがサーバーの正しいディレクトリーの中にない。ファイル *AppName*.htm、*AppName*_applet.htm、vapplet.jar、および *APPNAME*.jar が、HTTP サーバーの Pass ステートメントに対する 2 番目のパラメーターで指定されているディレクトリーの中に入っている必要があります。
- Java ファイル名は**大文字小文字を区別**します。これが構成上の問題の主な原因です。すべての .jar ファイルが大文字小文字を正しく区別されていることを確認してください。Windows Explorer は必ずしもファイル名を実際の大文字小文字の区別で表示するとは限りません。OS/400 wrklnk コマンドを使用して、IFS に保管されているファイル名の大文字小文字の区別を検査してください。

それでもアプレットが実行されない場合には、Java プラグイン・コンソールを使用可能にして、エラー・メッセージが表示されていないか確かめてください。プラグイン・コントロール・パネルは、***「スタート」 > 「プログラム」 > 「Java プラグイン・コントロール・パネル」***を選択することによって開始します。基本タブから、**「Java コンソールの表示」**を選択して、***「適用」***をクリックします。変更を有効にするには、オープンしているすべての Web ブラウザー・ウィンドウをクローズしてから、再び Web ブラウザーを始動します。次回にアプレットを実行すると、メッセージを表示した Java コンソール・ウィンドウが表示されます。

アプレットを別のアプレットから実行する

呼び出し側アプレットのメイン ***.htm*** ファイルに呼び出し先のアプレットを組み込むように変更することによって、VisualAge RPG 生成のアプレットを別のアプレットから実行することができます。たとえば、AppletB を AppletA から始動するには、***AppletA.htm*** ファイルを次のように変更します。

- APPLETA.jar および varpg.jar が含まれている行を探します。一般的には次の行です。

```
<PARAM NAME = "archive" VALUE = "APPLETA.jar , varpg.jar">

 ...
     archive = "APPLETA.jar , varpg.jar"
```

- ***varpg.jar*** と右引用符の間に ***, APPLETB.jar*** を挿入します。そうすると、***VALUE*** および ***archive*** の値が次のようになります。

```
<PARAM NAME = "archive" VALUE = "APPLETA.jar , varpg.jar , APPLETB.jar">

 ...
    archive = "APPLETA.jar , varpg.jar , APPLETB.jar"
```

  コンマ区切り文字の前後には必ずブランク文字を入れてください。

Web ブラウザーで AppletA.htm ページを表示すると、今度は AppletB も実行されるはずです。

呼び出し先のアプレット (この例では AppletB) もサーバー上のデータにアクセスする場合は、ワークステーション上に常駐するセキュリティー・ファイルを読み取る*許可*を各アプレットに与える必要があります。そうしないと、ユーザーは AppletB を実行するたびに有効なユーザー ID およびパスワードを入力するようにプロンプトが出されます。詳細については、219 ページの『アプレット用のセキュリティー・ファイルの使用』 を参照してください>

この例では、次の行を追加することによって、ローカル・***ポリシー・ファイル*** を変更する必要があります。

```
        permission java.lang.RuntimePermission "modifyThreadGroup";
        permission java.lang.RuntimePermission "modifyThread";
```

ポリシー・ファイル (java.policy) は、Java 実行時 lib\security サブディレクトリーにあります。

ソケット許可の場合は、次のような行を追加する必要がある場合があります。

```
        permission java.net.SocketPermission
"server_name:port_number", "connect,resolve";
```

このポリシー・ファイルの名前または位置を変更する必要がある場合には、同じディレクトリー内の ***java.security*** ファイルを変更してください。

第 21 章 Java コンパイル時のシステム機能呼び出し

VisualAge RPG コンパイラーには、Java ネイティブ・インターフェース (JNI) を通して Windows プラットフォームでダイナミック・リンク・ライブラリー (DLL) の機能入り口点として実行される外部プロシージャー呼び出しサポートが含まれています。この項では、このサポートの使用法について説明します。

前提事項については、Java 2 ソフトウェア開発きっと (J2SDK) の資料の Java ネイティブ・インターフェース (JNI) の項を参照してください。

単純呼び出し

最初のコードの例では、パラメーターなしで戻り値なしの外部プロシージャーに対する単純呼び出しを示します。単純 VisualAge RPG アプリケーションは、サンプルのダイナミック・リンク・ライブラリーで外部プロシージャーを呼び出します。JNI の指定は、呼び出されるネイティブ機能の機能名およびインターフェースを指示します。機能はコード化され、呼び出しのターゲットとなる新しい DLL にコンパイルされます。次の例は、C 言語でコード化されたネイティブ機能を示しているだけですが、示されているコーディング原則は他の言語の実装にも等しく適用されます。ネイティブ機能が一度制御権を持つと、システム API などの他のネイティブ機能を自由に呼び出すことができます。

呼び出されるネイティブ機能が入っている DLL の名前を指定する DLL キーワードを指定して、VisualAge RPG のプロシージャー用のプロシージャー・プロトタイプをコーディングしてください。EXTPROC キーワードは、VisualAge RPG プログラム内のプロシージャー名と異なる機能名を指定するために、オプショナルでコード化することができます。

注: EXTPROC キーワードの値は大文字と小文字が区別されます。VisualAge RPG ソースのプロシージャーに対する呼び出しをコーディングしてください。

```
 ********************************************************************
 *  Source File:  VCOMP1.VPG
 *
 *  Demonstrate calling an external procedure thru JNI.
 *
 ********************************************************************


 * This declares a procedure named 'sub1' which refers to
 * a function named 'proc1' in a Dynamic Link Library 'VSUB.DLL'

d sub1            pr                  dll('VSUB') extproc('proc1')

 * Without the EXTPROC keyword
d sub2            pr                  dll('VSUB')


C     *INZSR        BEGSR

C                   callp     sub1
C                   callp     sub2

c                   seton                                          lr

C                   ENDSR

 * This action subroutine is linked to a Create event for the Window.
 * It causes the component to end after running the INZSR.

C     CREATE1       BEGACT
C                   seton                                      LR
C                   ENDACT
```

図 74. サンプル・ファイル VCOMP1.VPG

Windows プラットフォームの場合には、ネイティブ機能は、StdCall プログラム・リンケージを使用するためにコード化されます。この機能は、DLL のエクスポート機能としてコード化されます。エクスポート機能名は、JNI の指定で指示された名前と一致しなければなりません。 JNI 指定の形式は次のとおりです。

```
Java_VARPGComponentName_ExternalProcedureName_OverloadedNativeMethods
```

完全なネイティブ機能名は、この例では 'Java_VCOMP1_proc1' および "Java_VCOMP1_SUB2' です。この例では多重定義されたネイティブ・メソッドは必要ありません。

JNI インターフェースは、最初の 2 つのパラメーター（インターフェース・ポインターとこのオブジェクト・ポインター）を指示します。追加のパラメーターは、プロシージャー宣言パラメーターに対応します。jni.h ヘッダー・ファイルは、インターフェース定義を提供するために、C 言語のソース・プログラムに含まれています。このファイルは J2SDK で提供されます。J2SDK の C 言語ヘッダー・ファイルが入っているディレクトリーを組み込むには、C コンパイラーで INCLUDE 環境変数を更新する必要があります。

最後に、サンプル C ソース・ファイルが DLL にコンパイルされます。

```
// Source File:  VSUB.C

// Add   (d:\jdk12\include;d:\jdk12\include\win32) to the INCLUDE setting
//      in order to find jni.h when compiling.

// Compiled with:  IBM VisualAge(TM) for C++ for Windows(R), Version 3.5
// Compile command:  icc  /q /ss /ge- /fe vsub.dll  vsub.c


#include <stdio.h>
#include <string.h>

#include <jni.h>


//-------------------------------------------------------------

void _Export __stdcall   Java_VCOMP1_proc1( void *je , void *jc)

{

  printf(" proc1 called successfully.\n");
}
//-------------------------------------------------------------

void _Export __stdcall   Java_VCOMP1_SUB2( void *je , void *jc)
{

  printf(" SUB2 called successfully.\n");
}
```

図 75. サンプル・ファイル VSUB.C

パラメーターの受け渡し

JNI 仕様は Java プリミティブ・データ・タイプ・ディレクトリーを渡しますが、VisualAge RPG はクラスを通してすべての VARPG データ・タイプを処理します。このことは、ネイティブ機能を呼び出す VARPG は、常にオブジェクトの引き渡しを伴うことを意味します。JNI 仕様は、ネイティブ機能のインターフェース機能を提供して、渡されたオブジェクトの値にアクセスします。それぞれの VARPG データ・タイプごとにクラス・メソッドが異なるので、それぞれのタイプについて個々に説明します。

パラメーター・タイプ

Character

VisualAge RPG は、長さが 1 の文字フィールドを含む Java のバイト配列として文字フィールドを実装します。JNI インターフェース機能 GetByteArrayElements は、バイト配列パラメーターの値を戻します。この値は、ReleaseByteArrayElements インターフェース機能を通して変更され、呼び出し側機能に戻されます。

注: リリース機能の呼び出し後に、この値をネイティブ機能で使用してはいけません。

C 言語ソースのネイティブ機能の最初のパラメーターは、ボイド・ポインターから JNIEnv ポインターに変更されています。これは、JNI インターフェース機能の機能ポインターのテーブルをポイントします。この 2 つの標準 JNI ポインターの後で、ネイティブ機能パラメーターに外部プロシージャーのプロトタイプ・パラメーターが追加されます。ネイティブ機能では、文字パラメーターは jbyteArray タイプとして宣言されます。

GetByteArrayElements インターフェース機能は、VARPG 文字フィールドの Java バイト配列の値を取得するために使用されます。

取得された値は、ReleaseByteArrayElements インターフェース機能によって変更され、Java 呼び出し側に戻されます。

リリースした後で、取得された値にアクセスしてはいけません。

注: 取得された値は、メモリー内のコピーでなく、Java オブジェクトの実際の値である可能性があります。この値の変更は、リリース機能を呼び出さなくても、Java 呼び出し側に反映されます。詳細については、JNI の資料の GetByteArrayElements 機能を参照してください。

```
 ***********************************************************************
 *  Source File:  VCOMP1.VPG
 *
 *  Demonstrate calling an external procedure thru JNI.
 *
 ***********************************************************************

 * This declares a procedure named 'sub1c' which refers to
 * a function named 'proc1c' in a dynamic Load Library 'VSUBC.DLL'

 * With 1 character parameter
d sub1c           pr                  dll('VSUBC') extproc('proc1c')

d                                  1

 * Without the EXTPROC keyword
 * With 2 character parameters
d sub2c           pr                  dll('VSUBC')
d                                  4
d                                 10

d c1              s                 1    inz('J')
d c4              s                 4    inz('blue')
d c10             s                10    inz('abcdefghij')

d mb1             m                       style(*info) button(*OK)

d rc              s                 9  0

C     *INZSR        BEGSR

C                   callp     sub1c(c1)
C                   callp     sub2c(c4:c10)

 * Display the changed values from the calls
c     c4            dsply     mb1           rc
c     c10           dsply     mb1           rc

c                   seton                                            lr
C                   ENDSR


 * This action subroutine is linked to a Create event for the Window.
 * It causes the component to end after running the INZSR.

C     CREATE1       BEGACT
C                   seton                                        LR
C                   ENDACT
```

図 76. サンプル・ファイル VCOMPC.VPG

```
// Source File:  VSUBC.C
//               Native function with Character parameters

// Compiled with:  IBM VisualAge(TM) for C++ for Windows(R), Version 3.5
// Compile command:  icc  /q /ss /ge- /fe vsubc.dll  vsubc.c

#include <stdio.h>
#include <string.h>

#include <jni.h>

//-------------------------------------------------------------

void _Export __stdcall   Java_VCOMPC_proc1c(  JNIEnv *je , void *jc,
                                              jbyteArray p1)

{
  char  *c1;

  printf(" proc1c called successfully.\n");

  c1 = (char *) (*je)->GetByteArrayElements( je, p1, NULL);

  printf(" c1 = \'%c\'\n", c1[0]);
}
//-------------------------------------------------------------

void _Export __stdcall   Java_VCOMPC_SUB2C( JNIEnv *je , void *jc,
                                            jbyteArray p1, jbyteArray p2)
{
  char  *c4;
  char  *c10;

  printf(" SUB2C called successfully.\n");


  c4  = (char *) (*je)->GetByteArrayElements( je, p1, NULL);
  c10 = (char *) (*je)->GetByteArrayElements( je, p2, NULL);

  printf(" c4  = %.4s.\n",  c4);
  printf(" c10 = %.10s.\n", c10);
```

図 77. サンプル・ファイル VSUBC.C (1/2)

```
  // Now change the values

  memcpy( c4, "Gray", 4);
  memcpy(c10, ">Received<", 10);

  // Update the values back to the Java Caller

  // Fourth Parameter = 0 also causes the variable's storage to be freed,
  // so can not access the variables after this function call.



  (*je)->ReleaseByteArrayElements( je, p1, (signed char *) c4, 0);
  (*je)->ReleaseByteArrayElements( je, p2, (signed char *) c10, 0);
}
```

図 77. サンプル・ファイル VSUBC.C (2/2)

ゾーン数値

```
 ********************************************************************
 *  Source File:  VCOMPN.VPG
 *
 *  Demonstrate calling an external procedure thru JNI.
 *
 ********************************************************************

 * With a Zoned(4,0)  parameter
d subz            pr                  dll('VSUBN')
d                                4S 0

 * With a Packed(9,2) parameter
d subp            pr                  dll('VSUBN')

d                                9P 2

 * With  Binary(4,0), Binary(9,0) parameter2
d subb            pr                  dll('VSUBN')
d                                4B 0
d                                9B 0

d z4              s               4S 0 inz(1234)
d p92             s               9P 2 inz(1234567.89)
d b4              s               4B 0 inz(1234)
d b9              s               9B 0 inz(123456789)
```

図 78. サンプル・ファイル VCOMPN.VPG (1/2)

```
d mb1             m                   style(*info) button(*OK)
d rc              s               9  0

C     *INZSR        BEGSR

C                   callp     subz(z4)
C                   callp     subp(p92)
C                   callp     subb(b4:b9)
 * Display the changed values from the calls
c     z4            dsply     mb1           rc
c     p92           dsply     mb1           rc

c     b4            dsply     mb1           rc
c     b9            dsply     mb1           rc

c                   seton                                          lr
C                   ENDSR

 * This action subroutine is linked to a Create event for the Window.
 * It causes the component to end after running the INZSR.

C     CREATE1       BEGACT
C                   seton                                      LR

C                   ENDACT
```

図 78. サンプル・ファイル VCOMPN.VPG (2/2)

```
// Source File:  VSUBN.C

//               Native function with Character parameters

// Add   (d:\jdk12\include;d:\jdk12\include\win32) to the INCLUDE setting
//      in order to find jni.h when compiling.

// Compiled with:  IBM VisualAge(TM) for C++ for Windows(R), Version 3.5
// Compile command:  icc  /q /ss /ge- /fe vsubn.dll  vsubn.c

#include <stdio.h>
#include <string.h>

#include <jni.h>

static void SwapBin2( short *b2);

static void SwapBin4( int *b4);

//-------------------------------------------------------------

void _Export __stdcall   Java_VCOMPN_SUBZ(  JNIEnv *je , void *jc,
                                            jobject p1)

{
  jclass     cls;
  jmethodID  mid;
  jobject    aryobj;
  char       *zd;

  printf(" SUBZ called successfully.\n");

  // p1:  Zoned
  // Call the method to get the zoned value

  cls = (*je)->GetObjectClass(je, p1);


  mid = (*je)->GetMethodID( je, cls, "zonedValue", "()[B");

  if ( mid == NULL)
  {
    printf(" ERROR: GetMethod.\n");
    return;
  }

  aryobj = (*je)->CallObjectMethod( je, p1, mid);

  zd = (char *) (*je)->GetByteArrayElements( je, aryobj, NULL);

  printf(" zd = %.4s.\n", zd);
```

図 79. サンプル・ファイル VSUBN.C (1/2)

```
  // Now change the values

  memcpy( zd, "9876", 4);

  // Returning the Zoned parameter

  // 1. Update the Byte array object with the changed value.


  (*je)->ReleaseByteArrayElements( je, aryobj, (signed char *) zd, 0);

  // 2. Prepare to call the method from the RpgNumeric class which
  //    takes a byte array object and assigns it's value into the
  //    RpgNumeric object.  Obtain the method ID.

  // cls = (*je)->GetObjectClass(je, p1);
  // (clS) still identifies the second parameter.  Re-use value

 mid = (*je)->GetMethodID( je, cls, "assignFromNative", "([BII)V");


  if ( mid == NULL)
  {
    printf(" ERROR 2: GetMethod.\n");
    return;
  }

  (*je)->CallVoidMethod( je, p1, mid,
                         aryobj,
                         (int) 1,    // = Component.ZONED_TYPE
                         0           // precision
                        );
}
```

図 79. サンプル・ファイル VSUBN.C (2/2)

RpgZoned オブジェクトはパラメーターとして渡されます。RpgZoned::zonedValue メソッドは、ゾーン 10 進フォーマットのオブジェクトの数値が入っているバイト配列を取得するために使用されます。インターフェースの Java サイドでバイト配列が取得されたら、GetByteArrayElements JNI インターフェース関数が呼び出されて、インターフェースのネイティブ・サイドのバイト配列にアクセスします。

オブジェクトで zonedValue メソッドを起動するために、GetObjectClass、GetMethodID、および CallObjectMethod インターフェース関数が使用されます。 GetMethodID 呼び出しで正しいメソッド・シグニチャーを指定するように注意しなければなりません。(パラメーターがなく、この場合にはバイト配列が戻されます。)

Java サイドで値を変更するために、バイト配列値が変更され、ReleaseByteArrayElements インターフェース関数が呼び出されて Java サイドでのバイト配列変更が設定され、Java サイドで適切な RpgZoned クラス・メソッドが呼び出されて RpgZoned オブジェクトの値がバイト配列で表された値で設定されます。この場合には、パラメーターとしてバイト配列および 2 つの整数を取得するのは assignFromNative メソッドです。(サンプル・コードの中で RpgNumeric クラスと呼ばれているものは RpgZoned クラスに対する親クラスです。)

パック数値

```
void _Export __stdcall   Java_VCOMPN_SUBP( JNIEnv *je , void *jc,
                                           jobject p1)  // P(9,2)
{
  jclass     cls;
  jmethodID  mid;
  jobject    aryobj;
  char       *packednum;

  printf(" SUBP called successfully.\n");

  // p1:  Packed 9,2
  // Call the method to get the zoned value

  cls = (*je)->GetObjectClass(je, p1);

  mid = (*je)->GetMethodID( je, cls, "packedValue", "()[B");


  if ( mid == NULL)
  {
    printf(" ERROR: GetMethod.\n");
    return;
  }

  aryobj = (*je)->CallObjectMethod( je, p1, mid);

  packednum = (char *) (*je)->GetByteArrayElements( je, aryobj, NULL);

  // Now change the values

  memcpy( packednum, "\x98\x76\x54\x32\x1C", 5);
```

図 80. サンプル・ファイル VSUBN.C（続き）(1/2)

```
  // Returning the Packed parameter

  // 1. Update the Byte array object with the changed value.

  // Fourth Parameter = 0 also causes the variable's storage to be freed,

  // so can not access the variables after this function call.

  (*je)->ReleaseByteArrayElements( je, aryobj, (signed char *) packednum, 0);

  // 2. Prepare to call the method from the RpgNumeric class which
  //    takes a byte array object and assigns it's value into the
  //    RpgNumeric object.  Obtain the method ID.

  // cls = (*je)->GetObjectClass(je, p1);
  // (cls) still identifies the second parameter.  Re-use value

  mid = (*je)->GetMethodID( je, cls, "assignFromNative", "([BII)V");


  if ( mid == NULL)
  {
    printf(" ERROR 2: GetMethod.\n");
    return;
  }


  (*je)->CallVoidMethod( je, p1, mid,
                         aryobj,
                         (int) 2,    // = Component.PACKED_TYPE
                         2           // precision  (Number of decimal places)
                        );
}
```

図 80. サンプル・ファイル VSUBN.C（続き）(2/2)

パック 10 進数パラメーターの場合は、ゾーン 10 進数に似ています。RpgPacked オブジェクトをバイト配列値に変換する適切なメソッドが使用されるだけです。

RpgPacked::packedValue メソッドは、パック 10 進数形式の RpgPacked パラメーター・オブジェクトの数値が入っているバイト配列を取得するために使用され、次に JNI インターフェース機能が呼び出されて Java バイト配列をインターフェースのネイティブ側からアクセスできるようにします。

ネイティブ側でバイト配列を変更し、ReleaseByteArrayElements インターフェース機能を呼び出して Java 側にバイト配列を戻した後で、assignFromNative メソッドが再び呼び出されて Java バイト配列から RpgPacked オブジェクトの値が設定されます。

2 進数

```
void _Export __stdcall   Java_VCOMPN_SUBB( JNIEnv *je , void *jc,
                                            jobject p1  // B(4,0)
                                           ,jobject p2) // B(9,0)
{
  jclass     cls;
  jmethodID  mid;
  jobject    aryobj;
  jobject    aryobj2;

  char       *binarynum;
  char       *b9;

  short      binary2;
  int        binary4;

  printf(" SUBB called successfully.\n");

  // p1:  Binary 4,0

  // Call the method to get the binary value

  cls = (*je)->GetObjectClass(je, p1);

  mid = (*je)->GetMethodID( je, cls, "binaryValue", "()[B");

  if ( mid == NULL)
  {
    printf(" ERROR: GetMethod.\n");
    return;
  }

  aryobj = (*je)->CallObjectMethod( je, p1, mid);

  binarynum = (char *) (*je)->GetByteArrayElements( je, aryobj, NULL);

  // Must reverse the byte order of the value received

  memcpy( &binary2, binarynum, 2);

  SwapBin2( &binary2);
```

図 81. サンプル・ファイル VSUBN.C（続き）(1/5)

```
printf(" binary = %hd\n", (short ) binary2);

// p2:  Binary 9,0
// Call the method to get the binary value

cls = (*je)->GetObjectClass(je, p2);

mid = (*je)->GetMethodID( je, cls, "binaryValue", "()[B");

if ( mid == NULL)
{
  printf(" ERROR: GetMethod.\n");
  return;
}

aryobj2 = (*je)->CallObjectMethod( je, p2, mid);

b9 = (char *) (*je)->GetByteArrayElements( je, aryobj2, NULL);


// Must reverse the byte order of the value received

memcpy( &binary4, b9, 4);
SwapBin4( &binary4);

printf(" binary = %d.\n", (int ) binary4 );
```

図 81. サンプル・ファイル VSUBN.C（続き）(2/5)

```
// Now change the values

binary2 = 5;

// Swap it back for returning to the Java value

SwapBin2( &binary2);
memcpy( binarynum, &binary2, 2);

// Returning the parameter

// 1. Update the Byte array object with the changed value.


// Fourth Parameter = 0 also causes the variable's storage to be freed,
// so can not access the variables after this function call.

(*je)->ReleaseByteArrayElements( je, aryobj, (signed char *) binarynum, 0);

// 2. Prepare to call the method from the RpgNumeric class which
//    takes a byte array object and assigns it's value into the
//    RpgNumeric object.  Obtain the method ID.

cls = (*je)->GetObjectClass(je, p1);

// (cls) still identifies the second parameter.  Re-use value

mid = (*je)->GetMethodID( je, cls, "assignFromNative", "([BII)V");

if ( mid == NULL)
{
  printf(" ERROR 2: GetMethod.\n");
  return;
}
```

図 81. サンプル・ファイル VSUBN.C（続き）(3/5)

```
(*je)->CallVoidMethod( je, p1, mid,
                       aryobj,
                       (int) 3,    // = Component.BINARY_TYPE
                       0           // precision  (Number of decimal places)
                     );


// Now change the values

binary4 = 19981999;

// Swap it back for returning to the Java value

SwapBin4( &binary4);
memcpy( b9, &binary4, 4);

// Returning the parameter

// 1. Update the Byte array object with the changed value.

// Fourth Parameter = 0 also causes the variable's storage to be freed,
// so can not access the variables after this function call.

(*je)->ReleaseByteArrayElements( je, aryobj2, (signed char *) b9, 0);


// 2. Prepare to call the method from the RpgNumeric class which
//    takes a byte array object and assigns it's value into the
//    RpgNumeric object.  Obtain the method ID.

cls = (*je)->GetObjectClass(je, p2);
// (cls) still identifies the second parameter.  Re-use value

mid = (*je)->GetMethodID( je, cls, "assignFromNative", "([BII)V");

if ( mid == NULL)
{
  printf(" ERROR 2: GetMethod.\n");
  return;
}
```

図 81. サンプル・ファイル VSUBN.C（続き）(4/5)

```
  (*je)->CallVoidMethod( je, p2, mid,
                         aryobj2,
                         (int) 3,    // = Component.BINARY_TYPE
                         0           // precision  (Number of decimal places)
                        );
}
//---------------------------------------------------------------------
static void SwapBin2( short *b2)
{
   char  tmp;
   char  *p;

   p = (char *) b2;

   tmp = p[0];
   p[0] = p[1];
   p[1] = tmp;

}
//---------------------------------------------------------------------
static void SwapBin4( int *b4)
{
   char  tmp;
   char  *p;

   p = (char *) b4;

   tmp = p[0];
   p[0] = p[3];
   p[3] = tmp;

   tmp = p[1];
   p[1] = p[2];
   p[2] = tmp;
}
```

図 81. サンプル・ファイル VSUBN.C（続き）(5/5)

このケースはゾーン 10 進数に似ています。該当する RpgBinary オブジェクトのメソッドだけが使用されます。複雑になったのは、ネイティブ Intel アーキテクチャー・プラットフォームが 2 進整数を低位バイト最左端形式で保管し、Java 側がそれを低位バイト最右端形式で処理する点だけです。相互間で 2 バイトおよび 4 バイトの 2 進整数を変換するときにバイト順を逆にするために、SwapBin2 および SwapBin4 機能が採用されています。

ネイティブ 2 進数形式で RpgBinary パラメーター・オブジェクトの数値の入ったバイト配列を取得するために、RpgBinary::binaryValue メソッドが使用されます。その後で JNI インターフェース機能が呼び出されて、Java バイト配列をインターフェースのネイティブ側からアクセスできるようにします。ネイティブ側でバイト配列を変更し、ReleaseByteArrayElements インターフェース機能を呼び出してバイト配列を Java 側に戻した後で、assignFromNative メソッドが再び呼び出されて Java バイト配列から RpgBinary オブジェクトの値が設定されます。

整数、符号なし

```
      * With Parameters: Integer, unsigned
     d subiu           pr                  dll('VSUBO')
     d                                5i 0
     d                               10i 0
     d                                5u 0
     d                               10u 0
```

図 82. サンプル VJNIO.VPG

```
static void SwapBin2( char *);
static void SwapBin4( char *);

void _Export __stdcall   Java_VJNIO_SUBIU(  JNIEnv *je , void *jc,
                      jobject p1, jobject p2, jobject p3, jobject p4)
{
  jclass     cls, cls2;
  jmethodID  mid;
  jshort     i2;
  jint       i4;
  jobject    aryobj3, aryobj4;
  unsigned short *u2;
  unsigned int   *u4;

  printf(" SUBIU called successfully.\n");


  // p1:  Integer, 2 byte
  // Call the method to get the value


   cls = (*je)->GetObjectClass(je, p1);
   mid = (*je)->GetMethodID( je, cls, "getValue", "()S");

   if ( mid == NULL)
   {
      printf(" ERROR: GetMethod.\n");
      return;
   }

   i2 = (*je)->CallShortMethod( je, p1, mid);

   printf(" i2 = %hd\n", (short) i2);


  // p2:  Integer, 4 byte
  // Call the method to get the value

   cls = (*je)->GetObjectClass(je, p2);

   mid = (*je)->GetMethodID( je, cls, "getValue", "()I");
```

図 83. サンプル VSUBO.C (1/6)

```
 if ( mid == NULL)
 {
    printf(" ERROR: GetMethod.\n");
    return;
 }

 i4 = (*je)->CallIntMethod( je, p2, mid);

 printf(" i4 = %d\n", (short) i4);


// p3:  Unsigned 2-byte.
// Call the method to get the double value

 cls = (*je)->GetObjectClass(je, p3);

 mid = (*je)->GetMethodID( je, cls, "unsignedValue", "()[B");

 if ( mid == NULL)
 {
    printf(" ERROR: GetMethod.\n");

    return;
 }

 aryobj3 = (*je)->CallObjectMethod( je, p3, mid);

 u2 = (unsigned short *) (*je)->GetByteArrayElements( je, aryobj3, NULL);

 // Must reverse the byte order of the value received
 SwapBin2( (char *) u2);

 printf(" u2 = %hu\n", *u2 );
```

図 83. サンプル VSUBO.C (2/6)

```
// p4:  Unsigned 4-byte.
// Call the method to get the double value

 cls = (*je)->GetObjectClass(je, p4);

 mid = (*je)->GetMethodID( je, cls, "unsignedValue", "()[B");


 if ( mid == NULL)
 {
    printf(" ERROR: GetMethod.\n");
    return;
 }

 aryobj4 = (*je)->CallObjectMethod( je, p4, mid);

 u4 = (unsigned int *) (*je)->GetByteArrayElements( je, aryobj4, NULL);

 // Must reverse the byte order of the value received
 SwapBin4( (char *) u4);

 printf(" u4 = %u\n", *u4 );



 // Now change the values

 i2 = 99;
 i4 = 88;
 *u2 = 77;
 *u4 = 66;


 // Must reverse the byte order of the value being returned
 SwapBin2( (char *) u2);
 SwapBin4( (char *) u4);

 // Return the array memory to Java.  Used later to set return
 // values for parameters

 (*je)->ReleaseByteArrayElements( je, p3, (signed char *) u2, 0);
 (*je)->ReleaseByteArrayElements( je, p4, (signed char *) u4, 0);
```

図 83. サンプル VSUBO.C (3/6)

```
// Returning P1: Integer 2-byte

//    Invoke the RpgShortRef::setValue method to set the object

//    value with a short parameter value

//    Obtain the method ID so it can be invoked.

cls = (*je)->GetObjectClass(je, p1);
mid = (*je)->GetMethodID( je, cls, "setValue", "(S)V");

if ( mid == NULL)
{
   printf(" ERROR 5: GetMethod.\n");
   return;
}

(*je)->CallVoidMethod( je, p1, mid, i2);

// Returning P2: Integer 4-byte

//    Invoke the RpgIntRef::setValue method to set the object

//    value with an integer parameter value

//    Obtain the method ID so it can be invoked.

cls = (*je)->GetObjectClass(je, p2);
mid = (*je)->GetMethodID( je, cls, "setValue", "(I)V");

if ( mid == NULL)
{
   printf(" ERROR 6: GetMethod.\n");
   return;
}

(*je)->CallVoidMethod( je, p2, mid, i4);
```

図 83. サンプル VSUBO.C (4/6)

```
   // Returning P3: Unsigned 2-byte

   //    Invoke the RpgNumeric::assignFromNative method to set the object

   //    value with an unsigned parameter value

   //    Obtain the method ID so it can be invoked.
   cls = (*je)->GetObjectClass(je, p3);
   mid = (*je)->GetMethodID( je, cls, "assignFromNative", "([BII)V");

   if ( mid == NULL)
   {
      printf(" ERROR 7: GetMethod.\n");
      return;
   }

   // Pass (aryobj3) as first parameter to method because the
   // method expects a Java byte array object

   (*je)->CallVoidMethod( je, p3, mid,

                                  aryobj3,
                                  (int) 5,   // = Component.UNSIGNED_TYPE
                                  (int) 0);  // 0 decimal places


   // Returning P4: Unsigned 4-byte

   //    Invoke the RpgNumeric::assignFromNative method to set the object
   //    value with an unsigned parameter value

   //    Obtain the method ID so it can be invoked.

   cls = (*je)->GetObjectClass(je, p4);

   mid = (*je)->GetMethodID( je, cls, "assignFromNative", "([BII)V");

   if ( mid == NULL)
   {
      printf(" ERROR 8: GetMethod.\n");
      return;
   }

   (*je)->CallVoidMethod( je, p4, mid, aryobj4,
                                  (int) 5,   // = Component.UNSIGNED_TYPE
                                  (int) 0);  // 0 decimal places

}
```

図 83. サンプル VSUBO.C (5/6)

```
static void SwapBin2( char *p)
{
   char  tmp;

   tmp = p[0];
   p[0] = p[1];

   p[1] = tmp;
}

static void SwapBin4( char *p)
{
   char  tmp;

   tmp = p[0];
   p[0] = p[3];
   p[3] = tmp;

   tmp = p[1];
   p[1] = p[2];
   p[2] = tmp;
}
```

図 83. サンプル VSUBO.C (6/6)

2 バイト整数は RpgShortRef::getValue および setValue メソッドを使用して、ネイティブ側の短縮値でそれらの値にアクセスします。同様に、4 バイト整数は RpgIntRef::getValue および setValue メソッドを使用して、ネイティブ側の int 値間で受け渡しします。

符号なしパラメーターは、符号なしの値と一致する Java プリミティブが欠けることによって複雑になっています。符号なしのオブジェクト値にはバイト配列のプリミティブを通してアクセスされます。このパラメーター・アクセスでは、符号なしの値を表すバイト配列を取得するメソッドが呼び出され、次にネイティブ側の配列エレメントにアクセスするために GetByteArrayElements インターフェース機能が呼び出されます。さらに、ネイティブの Intel/Windows プラットフォームでは、このバイト値は、低位バイト最左端形式に変更するために最初にバイトが逆順にされなければなりません。パラメーターの戻しは逆のプロセスに従います。

浮動 (4/8)

```
     * With Parameters: Float 4, Float 8.
    d subf            pr                  dll('VSUBO')
    d                                 4f
    d                                 8f
```

図 84. サンプル VJNIO.VPG

```
void _Export __stdcall   Java_VJNIO_SUBF(  JNIEnv *je , void *jc,
                                             jobject p1, jobject p2)
{
  jclass     cls, cls2;
  jmethodID  mid;
  jfloat     f4;
  jdouble    f8;

  // p1:  Float
  // Call the method to get the float value

   cls = (*je)->GetObjectClass(je, p1);
   mid = (*je)->GetMethodID( je, cls, "getValue", "()F");

   if ( mid == NULL)
   {
      printf(" ERROR: GetMethod.\n");

      return;
   }

   f4 = (*je)->CallFloatMethod( je, p1, mid);

   printf(" f4 = %f\n", (float) f4);

  // p2:  Double
  // Call the method to get the double value

   cls2 = (*je)->GetObjectClass(je, p2);

   mid = (*je)->GetMethodID( je, cls2, "getValue", "()D");

   if ( mid == NULL)
   {
      printf(" ERROR: GetMethod.\n");
      return;
   }

   f8 = (*je)->CallDoubleMethod( je, p2, mid);


   printf(" f8 = %lf\n", (double) f8);
```

図 85. サンプル VSUBO.C (1/2)

```
  // Now change the values

  f4 = 999.888;
  f8 = 98789.65456;

   // Returning the Float parameter

   //    Invoke the method from the RpgFloatRef class which
   //    assigns a Float parameter value to the object

   //    Obtain the method ID so it can be invoked.

   mid = (*je)->GetMethodID( je, cls, "setValue", "(F)V");

   if ( mid == NULL)
   {
      printf(" ERROR 2: GetMethod.\n");

      return;
   }


   (*je)->CallVoidMethod( je, p1, mid, f4);

   // Returning the Double parameter

   //    Invoke the method from the RpgDoubleRef class which
   //    assigns a Double parameter value to the object

   //    Obtain the method ID so it can be invoked.

   mid = (*je)->GetMethodID( je, cls2, "setValue", "(D)V");

   if ( mid == NULL)
   {
      printf(" ERROR 2: GetMethod.\n");
      return;

   }

   (*je)->CallVoidMethod( je, p2, mid, f8);
}
```

図 85. サンプル VSUBO.C (2/2)

浮動およびダブル・パラメーターのケースは、前のデータ・タイプに似ています。パラメーター値にアクセスするメソッドは Java プリミティブ・データ・タイプを処理するだけで、通常のバイト配列でなく、対応するネイティブ・プリミティブにマップします。これらの特定のプリミティブを処理する JNI インターフェース機能は、パラメーター値にアクセスするメソッドを呼び出すために使用されます。

RpgFloatRef::getValue、setValue、RpgDoubleRef::getValue、および setValue メソッドが使用されます。

日付、時刻、タイム・スタンプ

```
     * With Parameters: Date, Time, Timestamp.
    d subdtz          pr                  dll('VSUBO')
    d                                10d
    d                                 8t
    d                                26z

    d fd              s              10d   inz(D'1999-12-31')
    d ft              s               8t   inz(T'09.00.00')
    d fts             s              26z   inz(Z'2001-01-01-08.01.01')

    C                   callp     subdtz(fd:ft:fts)
```

図 86. サンプル VJNIO.VPG

```
void _Export __stdcall   Java_VJNIO_SUBDTZ(  JNIEnv *je , void *jc,
                  jbyteArray p1, jbyteArray p2, jbyteArray p3)
{
  char   *fd, *ft, *fz;

  fd  = (char *) (*je)->GetByteArrayElements( je, p1, NULL);
  ft  = (char *) (*je)->GetByteArrayElements( je, p2, NULL);
  fz  = (char *) (*je)->GetByteArrayElements( je, p3, NULL);

  printf(" fd = %.10s.\n", fd);
  printf(" ft = %.8s.\n",  ft);
  printf(" fz = %.26s.\n", fz);

  // Now change the values


  memcpy( fd, "2000-01-01",10);
  memcpy( ft, "17.00.00",  8);
  memcpy( fz, "2222-22-22-02.02.02", 19);

  // Update the values back to the Java Caller

  // Fourth Parameter = 0 also causes the variable's storage to be freed,
  // so can not access the variables after this function call.

  (*je)->ReleaseByteArrayElements( je, p1, (signed char *) fd, 0);
  (*je)->ReleaseByteArrayElements( je, p2, (signed char *) ft, 0);
  (*je)->ReleaseByteArrayElements( je, p3, (signed char *) fz, 0);


}
```

図 87. サンプル VSUBO.C

日付、時刻、およびタイム・スタンプ・パラメーターは、Java 側でバイト配列として実行されるので、文字パラメーターと同じ働きをします。

配列の受け渡し

配列パラメーターの処理は、データ・タイプによって 2 つの方法のうちのいずれかで行われます。 GetObjectArrayElement インターフェース機能を呼び出して配列内の個々のオブジェクト・エレメントに対するアドレスを取得して、スカラー・パラメーター・メソッドと同様に処理します。あるいは、Java プリミティブの配列の場合

には、ネイティブ・プリミティブの配列としてそれらにアクセスする特定のインターフェース機能があって、それらを Java に戻してリリースします。

```
d subca           pr                  dll('VSUBA')
d                                4
d                               10    dim(4)

d c1              s                1    inz('J')
d c4              s                4    inz('blue')
d c10             s               10    inz('abcdefghij') dim(4)

d subz            pr                  dll('VSUBA')
d                                4S 0 dim(4)

d subp            pr                  dll('VSUBA')

d                                9P 2 dim(4)

d subb            pr                  dll('VSUBA')
d                                4B 0 dim(4)
d                                9B 0 dim(4)

d z4              s                4S 0 dim(4)
d p92             s                9P 2 dim(4)
d b4              s                4B 0 dim(4)
d b9              s                9B 0 dim(4)

d subf            pr                  dll('VSUBA')

d                                4f   dim(4)
d                                8f   dim(4)
```

図 88. サンプル VJNIA.VPG (1/2)

```
d subdtz          pr                  dll('VSUBA')
d                                10d   dim(4)
d                                 8t   dim(4)
d                                26z   dim(4)

d subiu           pr                  dll('VSUBA')
d                                 5i 0 dim(4)
d                                10i 0 dim(4)

d                                 5u 0 dim(4)
d                                10u 0 dim(4)

d f4              s                4f   dim(4) inz(1234.56)
d f8              s                8f   dim(4) inz(1111.2222)
d fd              s               10d   dim(4) inz(D'1999-12-31')
d ft              s                8t   dim(4) inz(T'09.00.00')
d fts             s               26z   dim(4) inz(Z'2001-01-01-08.01.01')

d fi2             s                5i 0 dim(4) inz(1)

d fi4             s               10i 0 dim(4) inz(2)
d fu2             s                5u 0 dim(4) inz(3)
d fu4             s               10u 0 dim(4) inz(4)

C     *INZSR        BEGSR

C                   callp     subca(c4:c10)
C                   callp     subz(z4)
C                   callp     subp(p92)
C                   callp     subb(b4:b9)
C                   callp     subf(f4:f8)
C                   callp     subdtz(fd:ft:fts)

C                   callp     subiu(fi2:fi4:fu2:fu4)
c                   seton                                          lr
C                   ENDSR
```

図 88. サンプル VJNIA.VPG (2/2)

```
// Source File:  VSUBA.C
//               Native function with Character parameters
// Add   (d:\jdk12\include;d:\jdk12\include\win32) to the INCLUDE setting
//      in order to find jni.h when compiling.
// Compiled with:  IBM VisualAge(TM) for C++ for Windows(R), Version 3.5
// Compile command:  icc  /q /ss /ge- /fe vsuba.dll  vsuba.c
#include <stdio.h>
#include <string.h>
#include <jni.h>
static void SwapBin2( char *);
static void SwapBin4( char *);

void _Export __stdcall   Java_VJNIA_SUBCA( JNIEnv *je , void *jc,

                                           jbyteArray p1, jobjectArray p2)
{
  char  *c4;
  char  *c10;
  jobject  p2e;

  // Resolve to 2nd element of array parameter
  p2e = (*je)->GetObjectArrayElement( je, p2,
                                      1);     /* Array index, first element = 0.  */

  c4  = (char *) (*je)->GetByteArrayElements( je, p1, NULL);
  c10 = (char *) (*je)->GetByteArrayElements( je, p2e, NULL);

  printf(" c4  = %.4s.\n",  c4);

  printf(" c10 = %.10s.\n", c10);

  // Now change the values

  memcpy( c4, "Gray", 4);
  memcpy(c10, "Changed   ", 10);

  // Update the values back to the Java Caller

  (*je)->ReleaseByteArrayElements( je, p1, (signed char *) c4, 0);
  (*je)->ReleaseByteArrayElements( je, p2e, (signed char *) c10, 0);
}

void _Export __stdcall   Java_VJNIA_SUBZ(  JNIEnv *je , void *jc,
                                           jobject p1)
```

図 89. サンプル VSUBA.C (1/14)

```
{
  jclass     cls;
  jmethodID  mid;
  jobject    aryobj;
  char       *zd;
  jobject    pe;

  // Resolve to element of array parameter
  pe = (*je)->GetObjectArrayElement( je, p1,
                                     0);     /* Array index, first element = 0.  */

  // p1:  Zoned
  // Call the method to get the zoned value

   cls = (*je)->GetObjectClass(je, pe);

   mid = (*je)->GetMethodID( je, cls, "zonedValue", "()[B");


   aryobj = (*je)->CallObjectMethod( je, pe, mid);

   zd = (char *) (*je)->GetByteArrayElements( je, aryobj, NULL);

   printf(" zd = %.4s.\n", zd);


  // Now change the values

  memcpy( zd, "9876", 4);


   // Returning the Zoned parameter

   // 1. Update the Byte array object with the changed value.

  (*je)->ReleaseByteArrayElements( je, aryobj, (signed char *) zd, 0);
```

図 89. サンプル VSUBA.C (2/14)

```
   // 2. Prepare to call the method from the RpgNumeric class which

   //  takes a byte array object and assigns it's value into the
   //  RpgNumeric object.  Obtain the method ID.

   // cls = (*je)->GetObjectClass(je, p1);
   // (clS) still identifies the second parameter.  Re-use value

   mid = (*je)->GetMethodID( je, cls, "assignFromNative", "([BII)V");

   (*je)->CallVoidMethod( je, pe, mid,
                          aryobj,
                          (int) 1,    // = Component.ZONED_TYPE
                          0           // precision

                        );
}

void _Export __stdcall   Java_VJNIA_SUBP( JNIEnv *je , void *jc,
                                          jobject p1  // P(9,2)
                                          )
{
  jclass     cls;
  jmethodID  mid;
  jobject    aryobj;
  char       *packednum;
  char       tmp[80];         // For tracing
  jobject    pe;

  // Resolve to element of array parameter

  pe = (*je)->GetObjectArrayElement( je, p1,

                                     1);     /* Array index, first element = 0.  */

  // p1:  Packed 9,2
  // Call the method to get the zoned value

   cls = (*je)->GetObjectClass(je, pe);

   mid = (*je)->GetMethodID( je, cls, "packedValue", "()[B");

   aryobj = (*je)->CallObjectMethod( je, pe, mid);

   packednum = (char *) (*je)->GetByteArrayElements( je, aryobj, NULL);
```

図 89. サンプル VSUBA.C (3/14)

```
  // Now change the values
  memcpy( packednum, "\x98\x76\x54\x32\x1C", 5);


   // Returning the Packed parameter

   // 1. Update the Byte array object with the changed value.

   // Fourth Parameter = 0 also causes the variable's storage to be freed,
   // so can not access the variables after this function call.

  (*je)->ReleaseByteArrayElements( je, aryobj, (signed char *) packednum, 0);

   // 2. Prepare to call the method from the RpgNumeric class which
   //  takes a byte array object and assigns it's value into the

   //  RpgNumeric object.  Obtain the method ID.

   // cls = (*je)->GetObjectClass(je, p1);
   // (clS) still identifies the second parameter.  Re-use value

   mid = (*je)->GetMethodID( je, cls, "assignFromNative", "([BII)V");

   (*je)->CallVoidMethod( je, pe, mid,
                          aryobj,
                          (int) 2,    // = Component.PACKED_TYPE
                          2           // precision  (Number of decimal places)

                        );
}
//-------------------------------------------------------------
```

図 89. サンプル VSUBA.C (4/14)

```
void _Export __stdcall   Java_VJNIA_SUBB( JNIEnv *je , void *jc,
                                           jobject p1  // B(4,0)
                                          ,jobject p2  // B(9,0)
                                         )
{
  jclass     cls;
  jmethodID  mid;
  jobject    aryobj;
  jobject    aryobj2;
  char       *binarynum;
  char       *b9;

  short      binary2;
  int        binary4;
  jobject    pe,p2e;


  // Resolve to element of array parameter

  pe = (*je)->GetObjectArrayElement( je, p1,
                                     2);     /* Array index, first element = 0.  */

  // Resolve to element of array parameter

  p2e = (*je)->GetObjectArrayElement( je, p2,
                                     3);     /* Array index, first element = 0.  */

  // p1:  Binary 4,0

  // Call the method to get the binary value

   cls = (*je)->GetObjectClass(je, pe);

   mid = (*je)->GetMethodID( je, cls, "binaryValue", "()[B");

   aryobj = (*je)->CallObjectMethod( je, pe, mid);

   binarynum = (char *) (*je)->GetByteArrayElements( je, aryobj, NULL);
```

図 89. サンプル VSUBA.C (5/14)

```
 // Must reverse the byte order of the value received

 memcpy( &binary2, binarynum, 2);
 SwapBin2( (char *) &binary2);

 printf(" binary = %hd\n", (short ) binary2);


// p2:  Binary 9,0
// Call the method to get the binary value

 cls = (*je)->GetObjectClass(je, p2e);

 mid = (*je)->GetMethodID( je, cls, "binaryValue", "()[B");

 aryobj2 = (*je)->CallObjectMethod( je, p2e, mid);

 b9 = (char *) (*je)->GetByteArrayElements( je, aryobj2, NULL);

 // Must reverse the byte order of the value received

 memcpy( &binary4, b9, 4);
 SwapBin4( (char *) &binary4);


 printf(" binary = %d.\n", (int ) binary4 );

// Now change the values

binary2 = 5;

 // Swap it back for returning to the Java value

 SwapBin2( (char *) &binary2);
```

図 89. サンプル VSUBA.C (6/14)

```
 memcpy( binarynum, &binary2, 2);


 // Returning the Packed parameter

 // 1. Update the Byte array object with the changed value.

 // Fourth Parameter = 0 also causes the variable's storage to be freed,

 // so can not access the variables after this function call.

(*je)->ReleaseByteArrayElements( je, aryobj, (signed char *) binarynum, 0);

 // 2. Prepare to call the method from the RpgNumeric class which
 //  takes a byte array object and assigns it's value into the
 //  RpgNumeric object.  Obtain the method ID.

  cls = (*je)->GetObjectClass(je, pe);
 // (clS) still identifies the second parameter.  Re-use value


 mid = (*je)->GetMethodID( je, cls, "assignFromNative", "([BII)V");

 (*je)->CallVoidMethod( je, pe, mid,
                        aryobj,
                        (int) 3,    // = Component.BINARY_TYPE
                        0           // precision  (Number of decimal places)
                      );


// Now change the values

binary4 = 19981999;

 // Swap it back for returning to the Java value

 SwapBin4( (char *) &binary4);


 memcpy( b9, &binary4, 4);
```

図 89. サンプル VSUBA.C (7/14)

```
   // Returning the Packed parameter

   // 1. Update the Byte array object with the changed value.

   // Fourth Parameter = 0 also causes the variable's storage to be freed,
   // so can not access the variables after this function call.

  (*je)->ReleaseByteArrayElements( je, aryobj2, (signed char *) b9, 0);

   // 2. Prepare to call the method from the RpgNumeric class which
   //  takes a byte array object and assigns it's value into the

   //  RpgNumeric object.  Obtain the method ID.

    cls = (*je)->GetObjectClass(je, p2e);
   // (clS) still identifies the second parameter.  Re-use value

   mid = (*je)->GetMethodID( je, cls, "assignFromNative", "([BII)V");

   (*je)->CallVoidMethod( je, p2e, mid,
                          aryobj2,
                          (int) 3,    // = Component.BINARY_TYPE
                          0           // precision  (Number of decimal places)

                        );
}

//-------------------------------------------------------------
```

図 89. サンプル VSUBA.C (8/14)

```
void _Export __stdcall   Java_VJNIA_SUBF(  JNIEnv *je , void *jc,
                                             jfloatArray p1, jdoubleArray p2)
{
  jclass     cls, cls2;
  jmethodID  mid;
  jfloat     *f4;
  jdouble    *f8;
  jobject    p1e,p2e;

  printf(" SUBF called successfully.\n");

  f4  =  (*je)->GetFloatArrayElements( je, p1, NULL);

  f8  =  (*je)->GetDoubleArrayElements( je, p2, NULL);

   printf(" f4 = %f\n", (float) f4[0]);


  // p2:  Double
  // Call the method to get the double value

   printf(" f8 = %lf\n", (double) f8[0]);

   // Now change the values

  f4[0] = 999.888;
  f8[1] = 98789.65456;

   // Returning the Float parameter

  (*je)->ReleaseFloatArrayElements( je, p1, f4, 0);
  (*je)->ReleaseDoubleArrayElements( je, p2, f8, 0);

}
```

図 89. サンプル VSUBA.C (9/14)

```
//-------------------------------------------------------------

void _Export __stdcall   Java_VJNIA_SUBDTZ(  JNIEnv *je , void *jc,
                  jbyteArray p1, jbyteArray p2, jbyteArray p3)
{
  char   *fd, *ft, *fz;
  jobject p1e, p2e, p3e;

  printf(" SUBDTZ called successfully.\n");

  // Resolve to element of array parameter

  p1e = (*je)->GetObjectArrayElement( je, p1,
                                      2);     /* Array index, first element = 0.  */


  p2e = (*je)->GetObjectArrayElement( je, p2,
                                      3);     /* Array index, first element = 0.  */
  p3e = (*je)->GetObjectArrayElement( je, p3,
                                      0);     /* Array index, first element = 0.  */


  fd  = (char *) (*je)->GetByteArrayElements( je, p1e, NULL);
  ft  = (char *) (*je)->GetByteArrayElements( je, p2e, NULL);
  fz  = (char *) (*je)->GetByteArrayElements( je, p3e, NULL);


  printf(" fd = %.10s.\n", fd);
  printf(" ft = %.8s.\n",  ft);
  printf(" fz = %.26s.\n", fz);

  // Now change the values

  memcpy( fd, "2000-01-01",10);
  memcpy( ft, "17.00.00",  8);
  memcpy( fz, "2222-22-22-02.02.02", 19);

  // Update the values back to the Java Caller

  (*je)->ReleaseByteArrayElements( je, p1e, (signed char *) fd, 0);
  (*je)->ReleaseByteArrayElements( je, p2e, (signed char *) ft, 0);
  (*je)->ReleaseByteArrayElements( je, p3e, (signed char *) fz, 0);


}
```

図 89. サンプル VSUBA.C (10/14)

```
//-------------------------------------------------------------

void _Export __stdcall   Java_VJNIA_SUBIU(  JNIEnv *je , void *jc,
                     jshortArray p1, jintArray p2, jobject p3, jobject p4)
{
  jclass     cls, cls2;
  jmethodID  mid;
  jshort     *i2;
  jint       *i4;
  jobject    aryobj3, aryobj4;
  unsigned short *u2;
  unsigned int   *u4;
  jobject    p1e, p2e, p3e, p4e;

  printf(" SUBIU called successfully.\n");


  // Resolve to element of array parameter

  i2  =  (*je)->GetShortArrayElements( je, p1, NULL);
  i4  =  (*je)->GetIntArrayElements( je, p2, NULL);

  p3e = (*je)->GetObjectArrayElement( je, p3, 2);
  p4e = (*je)->GetObjectArrayElement( je, p4, 3);


   printf(" i2 = %hd\n", (short) i2[0]);


   printf(" i4 = %d\n", (short) i4[1]);


  // p3:  Unsigned 2-byte.
  // Call the method to get the double value


   cls = (*je)->GetObjectClass(je, p3e);

   mid = (*je)->GetMethodID( je, cls, "unsignedValue", "()[B");

   aryobj3 = (*je)->CallObjectMethod( je, p3e, mid);

   u2 = (unsigned short *) (*je)->GetByteArrayElements( je, aryobj3, NULL);
```

図 89. サンプル VSUBA.C (11/14)

```
 // Must reverse the byte order of the value received
 SwapBin2( (char *) u2);

 printf(" u2 = %hu\n", *u2 );

// p4:  Unsigned 4-byte.
// Call the method to get the double value


 cls = (*je)->GetObjectClass(je, p4e);

 mid = (*je)->GetMethodID( je, cls, "unsignedValue", "()[B");

 aryobj4 = (*je)->CallObjectMethod( je, p4e, mid);

 u4 = (unsigned int *) (*je)->GetByteArrayElements( je, aryobj4, NULL);

 // Must reverse the byte order of the value received
 SwapBin4( (char *) u4);

 printf(" u4 = %u\n", *u4 );
```

図 89. サンプル VSUBA.C (12/14)

```
// Now change the values

i2[0] = 99;
i4[1] = 88;

*u2 = 77;
*u4 = 66;

// Must reverse the byte order of the value being returned
SwapBin2( (char *) u2);
SwapBin4( (char *) u4);

// Return the array memory to Java.  Used later to set return
// values for parameters

(*je)->ReleaseByteArrayElements( je, p3e, (signed char *) u2, 0);
(*je)->ReleaseByteArrayElements( je, p4e, (signed char *) u4, 0);


(*je)->ReleaseShortArrayElements( je, p1, i2, 0);

(*je)->ReleaseIntArrayElements( je, p2, i4, 0);



// Returning P3: Unsigned 2-byte

//    Invoke the RpgNumeric::assignFromNative method to set the object
//    value with an unsigned parameter value

//    Obtain the method ID so it can be invoked.
cls = (*je)->GetObjectClass(je, p3e);
mid = (*je)->GetMethodID( je, cls, "assignFromNative", "([BII)V");

// Pass (aryobj3) as first parameter to method because the

// method expects a Java byte array object

(*je)->CallVoidMethod( je, p3e, mid,
                                aryobj3,
                                (int) 5,   // = Component.UNSIGNED_TYPE
                                (int) 0);  // 0 decimal places
```

図 89. サンプル VSUBA.C (13/14)

```
   // Returning P4: Unsigned 4-byte

   //    Invoke the RpgNumeric::assignFromNative method to set the object
   //    value with an unsigned parameter value

   //    Obtain the method ID so it can be invoked.


   cls = (*je)->GetObjectClass(je, p4e);
   mid = (*je)->GetMethodID( je, cls, "assignFromNative", "([BII)V");

   (*je)->CallVoidMethod( je, p4e, mid, aryobj4,
                                  (int) 5,   // = Component.UNSIGNED_TYPE
                                  (int) 0);  // 0 decimal places
}

static void SwapBin2( char *p)
{
   char  tmp;

   tmp = p[0];
   p[0] = p[1];
   p[1] = tmp;
}

static void SwapBin4( char *p)

{
   char  tmp;

   tmp = p[0];
   p[0] = p[3];
   p[3] = tmp;

   tmp = p[1];
   p[1] = p[2];
   p[2] = tmp;
}
```

図 89. サンプル VSUBA.C (14/14)

文字値の戻し

```
     d subrc           pr            10    dll('VSUBR')

     d fc              s             10    inz('ibm varpg ')

     C                   eval      fc = subrc
```

図 90. サンプル VJNIR.VPG

```
jbyteArray  _Export __stdcall   Java_VJNIR_SUBRC(  JNIEnv *je , void *jc)
{
  jbyteArray ba;
  char       *p;

  printf(" SUBRC called successfully.\n");

  // Create a new byte array object so it can be returned.

  ba = (*je)->NewByteArray( je, 10 /* = byte array length */ );

  // Pin the byte array element memory so native side can access it.
  p  = (char *) (*je)->GetByteArrayElements( je, ba, NULL);

  memcpy( p, "Success   ",10);


  // Update the values back to the Java Caller

  // Fourth Parameter = 0 also causes the variable's storage to be freed,
  // so can not access the variables after this function call.

  (*je)->ReleaseByteArrayElements( je, ba, (signed char *) p, 0);

  return ba;
}
```

図 91. サンプル VSUBR.C

機能から値を戻すことには、該当する Java オブジェクトを取得して、次にそれを戻すことが含まれます。このサンプルでは、新しいオブジェクト (Character(10) フィールドに一致する）が作成されて、その値が割り当てられました。RPG の文字フィールドは Java のバイト配列として実装されるので、長さが 10 の Java バイト配列が作成されて、ネイティブ側でそのバイト配列エレメントにアクセスするために GetByteArrayElements インターフェース機能が使用され、その後で Java に戻されてリリースされ、最後にこの機能から戻すために使用されています。

該当する Java 場合と配列オブジェクトが入力パラメーターの 1 つからすでに使用可能な場合には、新規オブジェクトを作成しないでそれを使用することができます。

ゾーン値の戻し

```
     d subrs           pr             5s 0 dll('VSUBR')
     d fs              s              5s 0

     C                   eval      fc = subrc
```

図 92. サンプル VJNIR.VPG

```
jobject  _Export __stdcall  Java_VJNIR_SUBRS( JNIEnv *je , void *jc)
{
  jclass     cls;
  jmethodID  mid;
  jobject    rzo;

  jbyteArray ba;
  char       *p;

  printf(" SUBRS called successfully.\n");

  // Create a new RpgZoned object.

  cls = (*je)->FindClass( je, "com/ibm/varpg/rpgruntime/RpgZoned");

  mid = (*je)->GetMethodID( je, cls, "<init>", "(II)V");

  rzo = (*je)->NewObject( je, cls, mid,
                          (int) 5, /* # of digits */

                          (int) 0  /* # of decimal places */
                        );

  // To set the zoned object value, we need a Java byte array to use
  // as an input parameter to the method for setting the zoned object.
  // Could constuct a new byte array object, or get one by retrieving
  // the zoned value from the object.

  // Will construct a byte array.

  // Create a new byte array object

  ba = (*je)->NewByteArray( je, 5 /* = byte array length */ );


  // Pin the byte array element memory so native side can access it.

  p  = (char *) (*je)->GetByteArrayElements( je, ba, NULL);

  memcpy( p, "55555", 5);
```

図 93. サンプル VSUBR.C (1/2)

```
  // Update the values back to the Java Caller

  // Fourth Parameter = 0 also causes the variable's storage to be freed,
  // so can not access the variables after this function call.

  (*je)->ReleaseByteArrayElements( je, ba, (signed char *) p, 0);


   //  Prepare to call the method from the RpgNumeric class which

   //  takes a byte array object and assigns it's value into the
   //  RpgNumeric object.  Obtain the method ID.

   // cls = (*je)->GetObjectClass(je, p1);
   // (cls) still identifies the second parameter.  Re-use value

   mid = (*je)->GetMethodID( je, cls, "assignFromNative", "([BII)V");


   (*je)->CallVoidMethod( je, rzo, mid,
                          ba,
                          (int) 1,    // = Component.ZONED_TYPE
                          0           // precision

                        );
  return rzo;
}
```

図 93. サンプル VSUBR.C (2/2)

RpgZoned オブジェクトは戻すことができるように構成されます。その値は、次にメソッド呼び出しを通して設定されます。しかし、値を設定するメソッドでは値を提供するバイト配列オブジェクトが入力パラメーターとして必要になるので、最初にバイト配列オブジェクトが構成されます。

RpgZoned オブジェクトは、クラスを探し、次にそのクラスのコンストラクター・メソッド、その後でコンストラクター・メソッドを呼び出して構成されます。Java バイト配列オブジェクトが次に構成され、ゾーン形式のバイト値が設定されます。その後で、RpgZoned オブジェクトの値を設定するメソッドが解決されて呼び出され、そのパラメーターの 1つとしてバイト配列オブジェクトが渡されます。

パック値の戻し

```
     d subrp           pr             5p 0 dll('VSUBR')
     d fp              s              5p 0

     C                   eval      fp = subrp
```

図 94. サンプル VJNIR.VPG

```
jobject  _Export __stdcall  Java_VJNIR_SUBRP( JNIEnv *je , void *jc)
{
  jclass     cls;
  jmethodID  mid;
  jobject    ro;

  jbyteArray ba;
  char       *p;

  printf(" SUBRP called successfully.\n");

  // Create a new RpgPacked object.

  cls = (*je)->FindClass( je, "com/ibm/varpg/rpgruntime/RpgPacked");

  mid = (*je)->GetMethodID( je, cls, "<init>", "(II)V");

  ro = (*je)->NewObject( je, cls, mid,
                         (int) 5, /* # of digits */

                         (int) 0  /* # of decimal places */
                       );

  // To set the packed object value, we need a Java byte array to use
  // as an input parameter to the method for setting the packed object.
  // Could constuct a new byte array object, or get one by retrieving
  // the packed value from the object.

  // Create a new byte array object

  ba = (*je)->NewByteArray( je, 3 /* = byte array length */ );
```

図 95. サンプル VSUBR.C (1/2)

```
  // Pin the byte array element memory so native side can access it.

  p  = (char *) (*je)->GetByteArrayElements( je, ba, NULL);

  memcpy( p, "\x55\x55\x5C", 3);

  // Update the values back to the Java Caller

  (*je)->ReleaseByteArrayElements( je, ba, (signed char *) p, 0);

   //  Prepare to call the method from the RpgNumeric class which
   //  takes a byte array object and assigns it's value into the
   //  RpgNumeric object.  Obtain the method ID.


   mid = (*je)->GetMethodID( je, cls, "assignFromNative", "([BII)V");

   (*je)->CallVoidMethod( je, ro, mid,
                          ba,         // The byte array object
                          (int) 2,    // = Component.PACKED_TYPE
                          0           // decimal places
                        );
  return ro;
}
```

図 95. サンプル VSUBR.C (2/2)

パック値の戻しは、上記のゾーンのケースに似ています。

RpgPacked オブジェクトは、クラスを探し、次にそのクラスのコンストラクター・メソッド、その後でコンストラクター・メソッドを呼び出して構成されます。Java バイト配列オブジェクトが次に構成され、パック形式のバイト値が設定されます。その後で、RpgPacked オブジェクトの値を設定するメソッドが解決されて呼び出され、そのパラメーターの 1 つとしてバイト配列オブジェクトが渡されます。

2 進数値の戻し

```
     d subrb           pr             5b 0 dll('VSUBR')
     d fb              s              5b 0

     C                   eval      fb = subrb
```

図 96. サンプル VJNIR.VPG

```
jobject  _Export __stdcall  Java_VJNIR_SUBRB( JNIEnv *je , void *jc)
{
  jclass     cls;
  jmethodID  mid;
  jobject    ro;

  jbyteArray ba;
  char       *p;

  printf(" SUBRB called successfully.\n");

  // Create a new RpgPacked object.

  cls = (*je)->FindClass( je, "com/ibm/varpg/rpgruntime/RpgBinary");

  mid = (*je)->GetMethodID( je, cls, "<init>", "(II)V");

  ro = (*je)->NewObject( je, cls, mid,
                          (int) 5, /* # of digits */

                          (int) 0  /* # of decimal places */
                        );
```

図 97. サンプル VSUBR.C (1/2)

```
  // To set the object value, we need a Java byte array to use
  // as an input parameter to the method for setting the object.

  // Create a new byte array object

  ba = (*je)->NewByteArray( je, 4 /* = byte array length */ );

  // Pin the byte array element memory so native side can access it.

  p  = (char *) (*je)->GetByteArrayElements( je, ba, NULL);


  memcpy( p, "\x00\x00\xD9\x03", 4);      // 55555 = 0xD903

  // Update the values back to the Java Caller

  (*je)->ReleaseByteArrayElements( je, ba, (signed char *) p, 0);

   //  Prepare to call the method from the RpgNumeric class which
   //  takes a byte array object and assigns it's value into the
   //  RpgNumeric object.  Obtain the method ID.

   mid = (*je)->GetMethodID( je, cls, "assignFromNative", "([BII)V");


   (*je)->CallVoidMethod( je, ro, mid,
                          ba,         // The byte array object
                          (int) 3,    // = Component.Binary_TYPE
                          0           // decimal places
                        );
  return ro;
}
```

図 97. サンプル VSUBR.C (2/2)

2 進数値の戻しは上記のゾーンまたはパックのケースに似ていて、RpgBinary オブジェクトが戻されるだけです。

整数値の戻し

```
     d subri2          pr             5i 0 dll('VSUBR')
     d subri4          pr            10i 0 dll('VSUBR')

     d fi2             s              5i 0
     d fi4             s             10i 0

     C                   eval      fi2= subri2
     C                   eval      fi4= subri4
```

図 98. サンプル VJNIR.VPG

```
jshort  _Export __stdcall  Java_VJNIR_SUBRI2( JNIEnv *je , void *jc)
{
  jshort  rc;
  rc = -5555;
  return rc;
}

jint    _Export __stdcall  Java_VJNIR_SUBRI4( JNIEnv *je , void *jc)
{
  return -55555;
}
```

図 99. サンプル VSUBR.C

2 バイトまたは 4 バイトの 2 進整数値の戻しは簡単です。これは、タイプが Java プリミティブとしてサポートされるためです。

符号なし値の戻し

```
     d subru           pr             10u 0 dll('VSUBR')
     d fu              s              10u 0
     C                   eval      fu = subru
```

図 100. サンプル VJNIR.VPG

```
jobject  _Export __stdcall  Java_VJNIR_SUBRU( JNIEnv *je , void *jc)
{
  jclass     cls;
  jmethodID  mid;
  jobject    ro;

  jbyteArray ba;
  char       *p;

  printf(" SUBRU called successfully.\n");

  // Create a new RpgUnsigned object.

  cls = (*je)->FindClass( je, "com/ibm/varpg/rpgruntime/RpgUnsigned");

  mid = (*je)->GetMethodID( je, cls, "<init>", "(II)V");

  ro = (*je)->NewObject( je, cls, mid,
                         (int) 5, /* # of digits */

                         (int) 0  /* # of decimal places */
                       );

  // To set the object value, we need a Java byte array to use
  // as an input parameter to the method for setting the object.

  // Create a new byte array object

  ba = (*je)->NewByteArray( je, 4 /* = byte array length */ );

  // Pin the byte array element memory so native side can access it.

  p  = (char *) (*je)->GetByteArrayElements( je, ba, NULL);


  memcpy( p, "\x00\x00\xD9\x03", 4);      // 55555 = 0xD903
```

図 101. サンプル VSUBR.C (1/2)

```
  // Update the values back to the Java Caller

  (*je)->ReleaseByteArrayElements( je, ba, (signed char *) p, 0);

   //  Prepare to call the method from the RpgNumeric class which
   //  takes a byte array object and assigns it's value into the
   //  RpgNumeric object.  Obtain the method ID.

   mid = (*je)->GetMethodID( je, cls, "assignFromNative", "([BII)V");


   (*je)->CallVoidMethod( je, ro, mid,
                          ba,         // The byte array object
                          (int) 5,    // = Component.UNSIGNED_TYPE
                          0           // decimal places
                        );
  return ro;
}
```

図 101. サンプル VSUBR.C (2/2)

2 バイトまたは 4 バイトの符号なし 2 進数値の戻しは上記のゾーンまたはパックのケースに似ていて、RpgUnsigned オブジェクトが使用されるだけです。

日付、時刻、またはタイム・スタンプの戻し

```
     d subrd           pr             10d   dll('VSUBR')
     d fd              s              10d
     C                   eval      fd = subrd
```

図 102. サンプル VJNIR.VPG

```
jbyteArray  _Export __stdcall   Java_VJNIR_SUBRD(  JNIEnv *je , void *jc)
{
  jbyteArray ba;
  char       *p;

  // Create a new byte array object so it can be returned.

  ba = (*je)->NewByteArray( je, 10 /* = byte array length */ );

  // Pin the byte array element memory so native side can access it.

  p  = (char *) (*je)->GetByteArrayElements( je, ba, NULL);

  memcpy( p, "2000-01-01",10);

  // Update the values back to the Java Caller


  (*je)->ReleaseByteArrayElements( je, ba, (signed char *) p, 0);

  return ba;
}
```

図 103. サンプル VSUBR.C

日付、時刻、およびタイム・スタンプの値は、必要な長さの Java バイト配列として戻されます。これは文字データ・タイプに似ています。

浮動値の戻し

```
     d subrf           pr             4f   dll('VSUBR')
     d subrf8          pr             8f   dll('VSUBR')
     d ff              s              4f
     d ff8             s              8f

     C                   eval      ff = subrf
     C                   eval      ff8= subrf8
```

図 104. サンプル VJNIR.VPG

```
jfloat   _Export __stdcall  Java_VJNIR_SUBRF( JNIEnv *je , void *jc)
{
  jfloat  rc;

  rc = -4444.4444;
  return rc;
}

jdouble   _Export __stdcall  Java_VJNIR_SUBRF8( JNIEnv *je , void *jc)
{
  return -7777777.55555;
}
```

図 105. サンプル VSUBR.C

浮動またはダブル (8 バイト浮動) 値の戻しは直接実行されます。これは、タイプが Java プリミティブとしてサポートされるためです。

可変長文字値の戻し

```
     d subrcv          pr            10    dll('VSUBR')  varying
     d fcv             s             10    varying
     C                   eval      fcv= subrcv
```

図 106. サンプル VJNIR.VPG

```
jbyteArray  _Export __stdcall  Java_VJNIR_SUBRCV( JNIEnv *je , void *jc)
{
  jbyteArray ba;
  char       *p;

  // Return a byte array of the current data length

  // Create a new byte array object so it can be returned.

  ba = (*je)->NewByteArray( je, 4 /* = byte array length */ );

  // Pin the byte array element memory so native side can access it.

  p  = (char *) (*je)->GetByteArrayElements( je, ba, NULL);


  memcpy( p, "abcd",4);

  // Update the values back to the Java Caller

  (*je)->ReleaseByteArrayElements( je, ba, (signed char *) p, 0);

  return ba;
}
```

図 107. サンプル VSUBR.C

可変長文字値は Java バイト配列を通して戻され、この場合には配列の長さが現在の値の長さと一致します。

配列値の戻し

配列オブジェクトの割り振りには JNI インターフェース機能が呼び出されます。配列のエレメントが Java プリミティブ・データ・タイプの場合には、これらの配列オブジェクトのタイプを割り振るインターフェース機能が使用されます。(それぞれのプリミティブ・タイプに特定の機能があります。) これらは配列のエレメントも割り振ります。次に、配列のエレメントをネイティブ・メモリーにマップするインターフェース機能を呼び出すだけで、設定し、リリースして Java に戻して、この機能から戻すことができます。

配列のエレメントが Java プリミティブ・データ・タイプ以外のケースでは、Java オブジェクトは配列の各エレメントごとに割り振る必要があり、その値は必要に応じて設定されて、最後に配列の特定のエレメントにオブジェクトが割り当てられます。エレメントの個々のオブジェクトの割り振りは、そのデータ・タイプのスカラー戻り値のケースと同様です。

```
  *  Source File:  VJNIRA.VPG

d subrca          pr            10    dll('VSUBRA') dim(4)
d subrsa          pr             5s 0 dll('VSUBRA') dim(4)
d subrpa          pr             5p 0 dll('VSUBRA') dim(4)
d subrba          pr             5b 0 dll('VSUBRA') dim(4)
d subri2a         pr             5i 0 dll('VSUBRA') dim(4)
d subri4a         pr            10i 0 dll('VSUBRA') dim(4)
d subrua          pr            10u 0 dll('VSUBRA') dim(4)

d subrda          pr            10d   dll('VSUBRA') dim(4)
d subrfa          pr             4f   dll('VSUBRA') dim(4)
d subrf8a         pr             8f   dll('VSUBRA') dim(4)
d subrcva         pr            10    dll('VSUBRA')  varying dim(4)

d fc              s             10    dim(4)
d fs              s              5s 0 dim(4)
d fp              s              5p 0 dim(4)
d fb              s              5b 0 dim(4)

d fi2             s              5i 0 dim(4)
d fi4             s             10i 0 dim(4)
d fu              s             10u 0 dim(4)
d fd              s             10d   dim(4)
d ff              s              4f   dim(4)
d ff8             s              8f   dim(4)
d fcv             s             10    varying dim(4)

d mb1             m                   style(*info) button(*OK)
d rc              s              9  0
```

図 108. サンプル VJNIRA.VPG (1/2)

```
C     *INZSR        BEGSR

C                   eval      fc = subrca
c     fc(2)         dsply     mb1           rc
C                   eval      fs = subrsa
c     fs(2)         dsply     mb1           rc

C                   eval      fp = subrpa
c     fp(2)         dsply     mb1           rc

C                   eval      fb = subrba
c     fb(2)         dsply     mb1           rc


C                   eval      fi2= subri2a
c     fi2(2)        dsply     mb1           rc

C                   eval      fi4= subri4a
c     fi4(2)        dsply     mb1           rc

C                   eval      fu = subrua
c     fu(2)         dsply     mb1           rc

C                   eval      fd = subrda
c     fd(2)         dsply     mb1           rc

C                   eval      ff = subrfa

c     ff(2)         dsply     mb1           rc

C                   eval      ff8= subrf8a
c     ff8(2)        dsply     mb1           rc

C                   eval      fcv= subrcva
c     fcv(2)        dsply     mb1           rc
C                   eval      rc = %len(fcv(2))
c     rc            dsply     mb1           rc
c                   seton                                          lr
C                   ENDSR
```

図 108. サンプル VJNIRA.VPG (2/2)

```
// Source File:  VSUBRA.C

//               Native function which returns Array values

// Add   (d:\jdk12\include;d:\jdk12\include\win32) to the INCLUDE setting
//      in order to find jni.h when compiling.

// Compiled with:  IBM VisualAge(TM) for C++ for Windows(R), Version 3.5
// Compile command:  icc  /q /ss /ge- /fe vsubra.dll  vsubra.c


#include <stdio.h>
#include <string.h>

#include <jni.h>

static void SwapBin2( char *);

static void SwapBin4( char *);

//-------------------------------------------------------------

jobjectArray  _Export __stdcall   Java_VJNIRA_SUBRCA(  JNIEnv *je , void *jc)
{
  jobjectArray  oa;
  jclass     cls;
  jbyteArray ba;
  char       *p;
  int           i;

  printf(" SUBRCA called successfully.\n");


  // Create the object array

  cls = (*je)->FindClass( je, "java/lang/Object");
   if ( cls == NULL)
   {

      printf(" ERROR 1: FindClass.\n");
      return NULL;
   }
```

図 109. サンプル VSUBRA.C (1/22)

```
  oa = (*je)->NewObjectArray( je, 4 /* array length */, cls, NULL );

   if ( oa == NULL)
   {
      printf(" ERROR 2: Newobj\n");
      return NULL;
   }

  // Populate the array
  for (i=0; i<4; i++)
  {
    // Create a new byte array object so it can be returned.

    ba = (*je)->NewByteArray( je, 10 /* = byte array length */ );

    // Set value of 2nd element

    if ( 1 == i)
    {
      // Pin the byte array element memory so native side can access it.

      p  = (char *) (*je)->GetByteArrayElements( je, ba, NULL);

      memcpy( p, "Success   ",10);

      // Update the values back to the Java Caller

      // Fourth Parameter = 0 also causes the variable's storage to be freed,
      // so can not access the variables after this function call.

      (*je)->ReleaseByteArrayElements( je, ba, (signed char *) p, 0);

     }


    (*je)->SetObjectArrayElement( je, oa, i /* array element, starting at 0 */
                                  , ba);
  }  // for i

  return oa;
}
//-------------------------------------------------------------
```

図 109. サンプル VSUBRA.C (2/22)

```
jobjectArray  _Export __stdcall  Java_VJNIRA_SUBRSA( JNIEnv *je , void *jc)
{
  jobjectArray  oa;
  int           i;
  jclass     cls;
  jmethodID  mid;
  jobject    rzo;

  jbyteArray ba;
  char       *p;


  printf(" SUBRSA called successfully.\n");

  // Create the object array

  cls = (*je)->FindClass( je, "com/ibm/varpg/rpgruntime/RpgZoned");
   if ( cls == NULL)
   {
      printf(" ERROR 1: FindClass.\n");
      return NULL;
   }

  oa = (*je)->NewObjectArray( je, 4 /* array length */, cls, NULL );

   if ( oa == NULL)
   {
      printf(" ERROR 2: Newobj\n");
      return NULL;
   }

  // Populate the array
  for (i=0; i<4; i++)

  {

  // Create a new RpgZoned object.

  cls = (*je)->FindClass( je, "com/ibm/varpg/rpgruntime/RpgZoned");
   if ( cls == NULL)
   {
      printf(" ERROR 1: FindClass.\n");
      return NULL;
   }
```

図 109. サンプル VSUBRA.C (3/22)

```
mid = (*je)->GetMethodID( je, cls, "<init>", "(II)V");

 if ( mid == NULL)
 {
    printf(" ERROR: GetMethod.\n");
    return NULL;
 }

rzo = (*je)->NewObject( je, cls, mid,
                        (int) 5, /* # of digits */

                        (int) 0  /* # of decimal places */
                      );

 if ( rzo == NULL)
 {
    printf(" ERROR3: \n");
    return NULL;
 }

  // Set value of 2nd element
  if ( 1 == i)
  {


// To set the zoned object value, we need a Java byte array to use
// as an input parameter to the method for setting the zoned object.
// Could constuct a new byte array object, or get one by retrieving

// the zoned value from the object.

// Will construct a byte array.


// Create a new byte array object

ba = (*je)->NewByteArray( je, 5 /* = byte array length */ );


 if ( ba == NULL)
 {
    printf(" ERROR4: \n");
    return NULL;
 }
```

図 109. サンプル VSUBRA.C (4/22)

```
// Pin the byte array element memory so native side can access it.

    p  = (char *) (*je)->GetByteArrayElements( je, ba, NULL);

memcpy( p, "55555", 5);



    // Update the values back to the Java Caller

    // Fourth Parameter = 0 also causes the variable's storage to be freed,
    // so can not access the variables after this function call.

(*je)->ReleaseByteArrayElements( je, ba, (signed char *) p, 0);



 //  Prepare to call the method from the RpgNumeric class which
 //  takes a byte array object and assigns it's value into the
 //  RpgNumeric object.  Obtain the method ID.


 // cls = (*je)->GetObjectClass(je, p1);
 // (clS) still identifies the second parameter.  Re-use value

 mid = (*je)->GetMethodID( je, cls, "assignFromNative", "([BII)V");

 if ( mid == NULL)
 {
    printf(" ERROR 2: GetMethod.\n");
    return NULL;
 }


 (*je)->CallVoidMethod( je, rzo, mid,
                        ba,
                        (int) 1,    // = Component.ZONED_TYPE
                        0           // precision

                      );


  }
```

図 109. サンプル VSUBRA.C (5/22)

```
    (*je)->SetObjectArrayElement( je, oa, i /* array element, starting at 0 */
                                  , rzo);
  }  // for i

  return oa;

}
//-------------------------------------------------------------

jobjectArray  _Export __stdcall  Java_VJNIRA_SUBRPA( JNIEnv *je , void *jc)
{
  jobjectArray  oa;
  int           i;
  jclass     cls;
  jmethodID  mid;
  jobject    ro;


  jbyteArray ba;
  char       *p;

  printf(" SUBRPA called successfully.\n");

  // Create the object array

  cls = (*je)->FindClass( je, "com/ibm/varpg/rpgruntime/RpgPacked");
   if ( cls == NULL)
   {
      printf(" ERROR 1: FindClass.\n");
      return NULL;
   }

  oa = (*je)->NewObjectArray( je, 4 /* array length */, cls, NULL );

   if ( oa == NULL)
   {
      printf(" ERROR 2: Newobj\n");
      return NULL;

   }

  // Populate the array
  for (i=0; i<4; i++)
  {
```

図 109. サンプル VSUBRA.C (6/22)

```
  // Create a new RpgPacked object.
#if 0
  cls = (*je)->FindClass( je, "com/ibm/varpg/rpgruntime/RpgPacked");
   if ( cls == NULL)
   {
      printf(" ERROR 1: FindClass.\n");
      return NULL;
   }
#endif

  mid = (*je)->GetMethodID( je, cls, "<init>", "(II)V");

   if ( mid == NULL)
   {
      printf(" ERROR: GetMethod.\n");
      return NULL;
   }

  ro = (*je)->NewObject( je, cls, mid,

                         (int) 5, /* # of digits */
                         (int) 0  /* # of decimal places */
                        );

   if ( ro == NULL)
   {
      printf(" ERROR3: \n");
      return NULL;
   }


    // Set value of 2nd element
    if ( 1 == i)
    {

  // To set the packed object value, we need a Java byte array to use
  // as an input parameter to the method for setting the packed object.
  // Could constuct a new byte array object, or get one by retrieving

  // the packed value from the object.

  // Will construct a byte array.
```

図 109. サンプル VSUBRA.C (7/22)

```
// Create a new byte array object

ba = (*je)->NewByteArray( je, 3 /* = byte array length */ );


 if ( ba == NULL)
 {
    printf(" ERROR4: \n");
    return NULL;
 }


// Pin the byte array element memory so native side can access it.

p  = (char *) (*je)->GetByteArrayElements( je, ba, NULL);

memcpy( p, "\x55\x55\x5C", 3);



// Update the values back to the Java Caller

// Fourth Parameter = 0 also causes the variable's storage to be freed,
// so can not access the variables after this function call.

(*je)->ReleaseByteArrayElements( je, ba, (signed char *) p, 0);




 //  Prepare to call the method from the RpgNumeric class which
 //  takes a byte array object and assigns it's value into the
 //  RpgNumeric object.  Obtain the method ID.


 // cls = (*je)->GetObjectClass(je, p1);
 // (clS) still identifies the second parameter.  Re-use value

 mid = (*je)->GetMethodID( je, cls, "assignFromNative", "([BII)V");

 if ( mid == NULL)
 {
    printf(" ERROR 2: GetMethod.\n");
    return NULL;
 }
```

図 109. サンプル VSUBRA.C (8/22)

```
   (*je)->CallVoidMethod( je, ro, mid,
                          ba,         // The byte array object
                          (int) 2,    // = Component.PACKED_TYPE

                          0           // decimal places
                        );


    }

    (*je)->SetObjectArrayElement( je, oa, i /* array element, starting at 0 */
                                  , ro);
  }  // for i

  return oa;
}
//-------------------------------------------------------------

jobjectArray  _Export __stdcall  Java_VJNIRA_SUBRBA( JNIEnv *je , void *jc)
{
  jobjectArray  oa;
  int           i;
  jclass     cls;

  jmethodID  mid;
  jobject    ro;

  jbyteArray ba;
  char       *p;

  printf(" SUBRBA called successfully.\n");

  // Create the object array

  cls = (*je)->FindClass( je, "com/ibm/varpg/rpgruntime/RpgBinary");
   if ( cls == NULL)
   {
      printf(" ERROR 1: FindClass.\n");
      return NULL;
   }
```

図 109. サンプル VSUBRA.C (9/22)

```
  oa = (*je)->NewObjectArray( je, 4 /* array length */, cls, NULL );

   if ( oa == NULL)
   {
      printf(" ERROR 2: Newobj\n");

      return NULL;
   }

  // Populate the array
  for (i=0; i<4; i++)
  {

  // Create a new RpgPacked object.
#if 0
  cls = (*je)->FindClass( je, "com/ibm/varpg/rpgruntime/RpgBinary");
   if ( cls == NULL)
   {
      printf(" ERROR 1: FindClass.\n");
      return NULL;
   }
#endif

  mid = (*je)->GetMethodID( je, cls, "<init>", "(II)V");

   if ( mid == NULL)
   {
      printf(" ERROR: GetMethod.\n");
      return NULL;

   }

  ro = (*je)->NewObject( je, cls, mid,
                         (int) 5, /* # of digits */
                         (int) 0  /* # of decimal places */
                       );
```

図 109. サンプル VSUBRA.C (10/22)

```
 if ( ro == NULL)
 {
    printf(" ERROR3: \n");
    return NULL;
 }

  // Set value of 2nd element
  if ( 1 == i)
  {


// To set the packed object value, we need a Java byte array to use
// as an input parameter to the method for setting the packed object.

// Could constuct a new byte array object, or get one by retrieving
// the packed value from the object.

// Will construct a byte array.


// Create a new byte array object

ba = (*je)->NewByteArray( je, 4 /* = byte array length */ );


 if ( ba == NULL)
 {
    printf(" ERROR4: \n");
    return NULL;
 }


// Pin the byte array element memory so native side can access it.

p  = (char *) (*je)->GetByteArrayElements( je, ba, NULL);


memcpy( p, "\x00\x00\xD9\x03", 4);      // 55555 = 0xD903


    // Update the values back to the Java Caller

    // Fourth Parameter = 0 also causes the variable's storage to be freed,
    // so can not access the variables after this function call.

    (*je)->ReleaseByteArrayElements( je, ba, (signed char *) p, 0);
```

図 109. サンプル VSUBRA.C (11/22)

```
   //  Prepare to call the method from the RpgNumeric class which
   //  takes a byte array object and assigns it's value into the

   //  RpgNumeric object.  Obtain the method ID.

   // cls = (*je)->GetObjectClass(je, p1);
   // (clS) still identifies the second parameter.  Re-use value

   mid = (*je)->GetMethodID( je, cls, "assignFromNative", "([BII)V");

   if ( mid == NULL)
   {
      printf(" ERROR 2: GetMethod.\n");
      return NULL;
   }


   (*je)->CallVoidMethod( je, ro, mid,
                          ba,         // The byte array object
                          (int) 3,    // = Component.Binary_TYPE

                          0           // decimal places
                        );

    }

    (*je)->SetObjectArrayElement( je, oa, i /* array element, starting at 0 */
                                , ro);
  }  // for i

  return oa;
}
```

図 109. サンプル VSUBRA.C (12/22)

```
//-------------------------------------------------------------

jshortArray   _Export __stdcall  Java_VJNIRA_SUBRI2A( JNIEnv *je , void *jc)
{
  jshortArray  rc;
  jshort      *n;


  printf(" SUBRI2A called successfully.\n");

  rc = (*je)->NewShortArray( je, 4 /* = array length */ );

  // Pin the array element memory so native side can access it.

  n  =  (*je)->GetShortArrayElements( je, rc, NULL);

  n[1] = -5555;

  // Update the values back to the Java Caller

  (*je)->ReleaseShortArrayElements( je, rc, n, 0);

  return rc;
}
//-------------------------------------------------------------


jintArray   _Export __stdcall  Java_VJNIRA_SUBRI4A( JNIEnv *je , void *jc)
{
  jintArray  rc;
  jint       *n;
  printf(" SUBRI4A called successfully.\n");

  rc = (*je)->NewIntArray( je, 4 /* = array length */ );

  // Pin the array element memory so native side can access it.

  n  =  (*je)->GetIntArrayElements( je, rc, NULL);

  n[1] = -5555;

  // Update the values back to the Java Caller

  (*je)->ReleaseIntArrayElements( je, rc, n, 0);


  return rc;
}
```

図 109. サンプル VSUBRA.C (13/22)

```
//-------------------------------------------------------------

jobjectArray  _Export __stdcall  Java_VJNIRA_SUBRUA( JNIEnv *je , void *jc)
{
  jobjectArray  oa;
  int           i;
  jclass     cls;
  jmethodID  mid;
  jobject    ro;

  jbyteArray ba;
  char       *p;

  printf(" SUBRUA called successfully.\n");

  // Create the object array

  cls = (*je)->FindClass( je, "com/ibm/varpg/rpgruntime/RpgUnsigned");

   if ( cls == NULL)
   {
      printf(" ERROR 1: FindClass.\n");
      return NULL;
   }

  oa = (*je)->NewObjectArray( je, 4 /* array length */, cls, NULL );

   if ( oa == NULL)
   {
      printf(" ERROR 2: Newobj\n");
      return NULL;
   }

  // Populate the array
  for (i=0; i<4; i++)
  {

  // Create a new RpgPacked object.
#if 0
  cls = (*je)->FindClass( je, "com/ibm/varpg/rpgruntime/RpgUnsigned");
   if ( cls == NULL)

   {
      printf(" ERROR 1: FindClass.\n");
      return NULL;
   }
#endif
```

図 109. サンプル VSUBRA.C (14/22)

```
mid = (*je)->GetMethodID( je, cls, "<init>", "(II)V");

 if ( mid == NULL)
 {
    printf(" ERROR: GetMethod.\n");
    return NULL;
 }

ro = (*je)->NewObject( je, cls, mid,
                        (int) 5, /* # of digits */
                        (int) 0  /* # of decimal places */
                      );

 if ( ro == NULL)

 {
    printf(" ERROR3: \n");
    return NULL;
 }

  // Set value of 2nd element
  if ( 1 == i)
  {


// To set the packed object value, we need a Java byte array to use
// as an input parameter to the method for setting the packed object.
// Could constuct a new byte array object, or get one by retrieving
// the packed value from the object.

// Will construct a byte array.
```

図 109. サンプル VSUBRA.C (15/22)

```
// Create a new byte array object

ba = (*je)->NewByteArray( je, 4 /* = byte array length */ );

 if ( ba == NULL)
 {
    printf(" ERROR4: \n");
    return NULL;
 }

// Pin the byte array element memory so native side can access it.

p  = (char *) (*je)->GetByteArrayElements( je, ba, NULL);

memcpy( p, "\x00\x00\xD9\x03", 4);      // 55555 = 0xD903

// Update the values back to the Java Caller

// Fourth Parameter = 0 also causes the variable's storage to be freed,

// so can not access the variables after this function call.

(*je)->ReleaseByteArrayElements( je, ba, (signed char *) p, 0);
```

図 109. サンプル VSUBRA.C (16/22)

```
   //  Prepare to call the method from the RpgNumeric class which
   //  takes a byte array object and assigns it's value into the
   //  RpgNumeric object.  Obtain the method ID.

   // cls = (*je)->GetObjectClass(je, p1);
   // (clS) still identifies the second parameter.  Re-use value

   mid = (*je)->GetMethodID( je, cls, "assignFromNative", "([BII)V");


   if ( mid == NULL)
   {
      printf(" ERROR 2: GetMethod.\n");
      return NULL;
   }


   (*je)->CallVoidMethod( je, ro, mid,
                          ba,         // The byte array object
                          (int) 5,    // = Component.UNSIGNED_TYPE
                          0           // decimal places
                        );


    }

    (*je)->SetObjectArrayElement( je, oa, i /* array element, starting at 0 */

                                  , ro);
  }  // for i

  return oa;
}
//-------------------------------------------------------------
```

図 109. サンプル VSUBRA.C (17/22)

```
jobjectArray  _Export __stdcall   Java_VJNIRA_SUBRDA(  JNIEnv *je , void *jc)
{
  jobjectArray  oa;
  jclass     cls;
  jbyteArray ba;
  char       *p;
  int           i;

  printf(" SUBRD called successfully.\n");

  // Create the object array

  cls = (*je)->FindClass( je, "java/lang/Object");

   if ( cls == NULL)
   {
      printf(" ERROR 1: FindClass.\n");
      return NULL;
   }

  oa = (*je)->NewObjectArray( je, 4 /* array length */, cls, NULL );

   if ( oa == NULL)
   {
      printf(" ERROR 2: Newobj\n");
      return NULL;
   }

  // Populate the array
  for (i=0; i<4; i++)
  {
    // Create a new byte array object so it can be returned.

    ba = (*je)->NewByteArray( je, 10 /* = byte array length */ );
```

図 109. サンプル VSUBRA.C (18/22)

```
    // Set all elements to a valid date value.

      // Pin the byte array element memory so native side can access it.

      p  = (char *) (*je)->GetByteArrayElements( je, ba, NULL);

      memcpy( p, "2000-01-01",10);

      // Update the values back to the Java Caller

      // Fourth Parameter = 0 also causes the variable's storage to be freed,
      // so can not access the variables after this function call.


      (*je)->ReleaseByteArrayElements( je, ba, (signed char *) p, 0);


    (*je)->SetObjectArrayElement( je, oa, i /* array element, starting at 0 */
                                , ba);
  }  // for i

  return oa;
}
//-------------------------------------------------------------

jfloatArray   _Export __stdcall  Java_VJNIRA_SUBRFA( JNIEnv *je , void *jc)
{
  jfloatArray  rc;
  jfloat      *n;

  printf(" SUBRF called successfully.\n");


  rc = (*je)->NewFloatArray( je, 4 /* = array length */ );

  // Pin the array element memory so native side can access it.

  n  =  (*je)->GetFloatArrayElements( je, rc, NULL);

  n[1] = -4444.4444;

  // Update the values back to the Java Caller

  (*je)->ReleaseFloatArrayElements( je, rc, n, 0);

  return rc;
}
```

図 109. サンプル VSUBRA.C (19/22)

```
//-------------------------------------------------------------

jdoubleArray   _Export __stdcall  Java_VJNIRA_SUBRF8A( JNIEnv *je , void *jc)

{
  jdoubleArray  rc;
  jdouble      *n;

  printf(" SUBRF8 called successfully.\n");

  rc = (*je)->NewDoubleArray( je, 4 /* = array length */ );

  // Pin the array element memory so native side can access it.

  n  =  (*je)->GetDoubleArrayElements( je, rc, NULL);

  n[1] = -7777777.55555;

  // Update the values back to the Java Caller

  (*je)->ReleaseDoubleArrayElements( je, rc, n, 0);

  return rc;

}
//-------------------------------------------------------------

jobjectArray  _Export __stdcall  Java_VJNIRA_SUBRCVA( JNIEnv *je , void *jc)
{
  // This is similar to fixed length character, only the individual
  // array elemnts can be created as byte arrays of different lengths
  // to reflect the current length of the varying length values.
```

図 109. サンプル VSUBRA.C (20/22)

```
jobjectArray  oa;
jclass     cls;
jbyteArray ba;
char       *p;
int           i;


printf(" SUBRCVA called successfully.\n");


// Create the object array

cls = (*je)->FindClass( je, "java/lang/Object");
 if ( cls == NULL)
 {
    printf(" ERROR 1: FindClass.\n");
    return NULL;
 }

oa = (*je)->NewObjectArray( je, 4 /* array length */, cls, NULL );

 if ( oa == NULL)
 {
    printf(" ERROR 2: Newobj\n");
    return NULL;
 }
```

図 109. サンプル VSUBRA.C (21/22)

```
  // Populate the array
  for (i=0; i<4; i++)

  {
    // Create a new byte array object so it can be returned.

    ba = (*je)->NewByteArray( je,
                         /* = byte array length */
                         (1==i) ? 4 : 10 );

    // Set value of 2nd element
    if ( 1 == i)
    {
      // Pin the byte array element memory so native side can access it.

      p  = (char *) (*je)->GetByteArrayElements( je, ba, NULL);

      memcpy( p, "abcd",4);

      // Update the values back to the Java Caller


      // Fourth Parameter = 0 also causes the variable's storage to be freed,
      // so can not access the variables after this function call.

      (*je)->ReleaseByteArrayElements( je, ba, (signed char *) p, 0);
     }



    (*je)->SetObjectArrayElement( je, oa, i /* array element, starting at 0 */
                                  , ba);
  }  // for i

  return oa;
}
```

図 109. サンプル VSUBRA.C (22/22)

第 22 章 非 GUI VisualAge RPG プログラムの作成

この項では、スタンドアロン VARPG アプリケーションおよび**ダイナミック・リンク・ライブラリー** (DLL) の作成方法について説明します。スタンドアロン VARPG アプリケーションにはユーザー・インターフェースはなく、ローカルおよび AS/400 ファイルにアクセスして AS/400 プログラムを呼び出すことができます。DLL は直接実行できないモジュールです。これには、他の VARPG アプリケーションによって呼び出されるプロシージャーが入っています。また DLL は、ローカル・ファイルや AS/400 ファイルおよびプログラムにアクセスすることもできます。DLL は、AS/400 サービス・プログラムと同様に考えることができます。

VARPG GUI Designer 内で、あるいは MS-DOS コマンド・プロンプトでコマンドを出して、スタンドアロン VARPG アプリケーションを作成することができます。（コマンドについては、477 ページの『付録 C. 非 GUI プログラムを MS-DOS から作成してコンパイルする』 を参照してください。）この項では、GUI Designer を使用して非 GUI プログラムを作成する方法について説明します。

スタンドアロン・プログラムまたは DLL を作成する場合には、次の制約事項が適用されます。

- すべてプロシージャーで構成されなければなりません。
- *ENTRY は使用できません。
- 特殊のサブルーチン *INZSR および *TERMSR は使用できません。
- すべてのサブルーチンがプロシージャーに対してローカルでなければなりません。
- スタンドアロン・アプリケーションを作成する場合には、EXPORT キーワードは使用できません。
- スタンドアロン・アプリケーションおよび DLL はユーザー・インターフェースを持たないので、%GETATR および %SETATR 組み込み関数や GUI 命令コードは使用できません。これには以下が含まれます。
  - CLSWIN、GETATR、SETATR、START、STOP、SHOWWIN、READS

  DSPLY 命令コードは使用できます。しかし、これに含まれているプロシージャーが VisualAge RPG DLL から呼び出される場合には、DSPLY 命令コードは何も実行しません。また、DSPLY 命令コードは演算項目 1 のメッセージ・データ・タイプをサポートしません。

スタンドアロン VARPG プログラムの作成

スタンドアロン VARPG プログラムは、制御仕様で EXE キーワードが指定された場合に作成されます。

```
H  EXE
```

プログラム・ソースには、名前がソース・ファイルの名前と一致するプロシージャーが入っていなければなりません。これは、プログラムの主入り口点となります。プログラムに渡されるパラメーターがある場合には、それらをメイン・プロシージャーのパラメーター定義で指定しなければならず、それらは値によって渡されなけ

ればなりません。すなわち、それぞれのパラメーターに **VALUE** キーワードを指定する必要があります。コマンド行からアプリケーションを呼び出す場合には、パラメーターをスペースで区切ってください。指定された数よりパラメーターが多くても少なくても、エラー・メッセージは表示されません。

GUI Designer でスタンドアロン・プログラムを作成するには、「プロジェクト」ウィンドウから**「プロジェクト」>「新規」>「非 GUI プロジェクト」**を選択してください。エディターが、H 制御仕様テンプレートを持つ新しいソース・ファイルをオープンします。H * EXE 仕様のコメントを外して、プログラムをコーディングしてください。完了したら、そのプロジェクトを保管して、アプリケーションをビルドしてください。「プロジェクト」ウィンドウから必要なビルド・オプションも設定できます。

次の例では、スタンドアロン VARPG プログラムは単一のパラメーターを受け入れます。実行すると、このプログラムはパラメーターを大文字に変換し、DSPLY 命令コードを使用して結果を表示します。メイン（唯一の）プロシージャーの名前は *MyPgm* であることに注意してください。このサンプルを試す場合には、保管時にファイル名として MYPGM を指定するようにしてください。

```
 * Sample standalone VARPG program
H EXE
 *
 * Prototype for the main procedure
D MyPgm           PR
D                                64A   Value
 *
 * Procedure definition for MYPGM
PMyPgm            B
 *
D MyPgm           PI
D  InString                      64A   Value
 *
D OutString       S               64A
 *
D LC              C                    'abcdefghijklmnopqrstuvwxyz'
D UC              C                    'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
 *
 * Translate input parameter to uppercase and display it
C     lc:uc         Xlate     InString      OutString
C     OutString     Dsply                   I                 1
 *
PMyPgm            E
```

DLL の作成

DLL は、制御仕様でキーワード NOMAIN が指定された場合に作成されます。

```
H NOMAIN
```

GUI Designer で DLL を作成するには、「プロジェクト」ウィンドウで**「プロジェクト」>「新規」>「非 GUI プロジェクト」**を選択してください。エディターが、H 制御仕様テンプレートを持つ新しいソース・ファイルをオープンします。H * NOMAIN 仕様のコメントを外して、プログラムをコーディングしてください。完了したら、そのプロジェクトを保管して、DLL をビルドしてください。「プロジェクト」ウィンドウから必要なビルド・オプションも設定できます。

DLL をビルドすると、コンパイラーは DLL と LIB ファイルを作成します。LIB ファイルは、DLL を他のアプリケーションとリンクするために使用されます。LIB ファイルは、ソースと同じディレクトリーに入れられて、DLL と同じ名前になります。LIB ファイルには、開始 P 仕様に EXPORT キーワードを持つすべてのプロシージャーが入ります。

次の例は、DLL のプロシージャーとして小文字のストリングを大文字に変換するプログラム *MyPGM* の一部をコーディングする方法を示しています。この DLL のソースには、*ToUpper* という名前の 1 つのプロシージャーがあります。このプロシージャーを他のプログラムから呼び出せるように、プロシージャー定義に **Export** キーワードを追加してください。

```
 * Sample VARPG DLL
H NOMAIN
 *
 * Prototype ToUpper procedure
D ToUpper         PR            64A
D                               64A   Value
 *
 * The ToUpper procedure
PToUpper          B                   Export
 *
D ToUpper         PI            64A
D  InString                     64A   Value
 *
D OutString       S             64A
 *
D LC              C                   'abcdefghijklmnopqrstuvwxyz'
D UC              C                   'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
 *
 *
C     lc:uc         Xlate     InString      OutString
C                   Return    OutString
 *
PToUpper          E
```

DLL を作成しビルドしたら、これに任意の名前を付けることができます。この例では、*MyFunc* を使用しています。ビルドが正常に行われると、ソース・ディレクトリーに次のファイルが作成されます。

- MyFunc.VPG - プログラム・ソース
- MyFunc.DLL - DLL
- MyFunc.LST - コンパイラー・リスト
- MyFunc.LIB - ライブラリー・ファイル
- MyFunc.EVT - イベント・ファイル (GUI Designer がエラー・フィードバック・ウィンドウを表示するために使用します。プログラムの実行には必要ありません。)

作成した MyFunc.DLL で ToUpper プロシージャーを呼び出すように、MyPGM ソースを編集し変更してください。変更されたソースは次のようになります。

```
0000  * Calling a procedure in a VARPG DLL
0001 H EXE
0002  *

0003 D ToUpper         PR            64A   DLL('MyFunc')
0004 D                                     ExtProc('TOUPPER')
```

```
0006 D                                64A   Value

0007  *
0008 D MyMain          PR
0009 D                                64A   Value
0010  *
0011 PMyMain           B
0012  *
0013 D MyMain          PI
0014 D  InString                      64A   Value
0015  *
0016 D Upper           S               64A
0017  *

0018 C                   Eval      Upper=ToUpper(Instring)

0019 C     Upper         Dsply                   I                1
0010  *
0011 PMyMain           E
```

行　　変更の記述

0003　ToUpper プロシージャーのプロトタイプを定義し、このプロシージャーがパラメーター (64 バイトの長さの英字フィールド) を戻すように指定してください。DLL キーワードは、このプロシージャーが MyFunc.DLL という名前の DLL に入っていることを示します。

0004　ExtProc キーワードは、呼び出されるプロシージャーの名前を指定します。この名前は定義仕様で使用される名前と同じである (0003 行目) ため、このキーワードは省略することができます。しかし、名前を指定する場合には、示されているように、名前は大文字でなければなりません。

0006　このステートメントは、このプロシージャーが 1 つのパラメーター - 64 バイトの英字フィールド - を必要としていることを示します。この場合には、パラメーターは VALUE によって渡されます。

0018　これは、プロシージャーの呼び出しです。

呼び出そうとしているプロシージャーが値を戻さない場合には、呼び出しに CALLP 命令コードを使用する必要があります。

```
C             CALLP     SomeFunc(parm1:parm2)
```

例外処理

例外処理は、GUI VARPG アプリケーションと次の点で異なります。

- 呼び出し元がユーティリティー DLL に入っていない場合には、例外についての情報が呼び出し元に通知されません。
- デフォルトの例外処理プログラムは DLL からは呼び出されません。プロシージャー内で例外が起こったときにデフォルトの例外処理プログラムは呼び出されないためです。DLL で例外が起こってエラー表示または *PSSR がない場合には、その DLL は終了します。例外についての情報は FVDCERRS.LOG ファイルに書き込まれます。
- ユーティリティーDLL で例外を処理する方法として推奨されるのは、呼び出し元に適切な戻りコードを返すそれぞれのルーチンに、エラー標識またはローカル *PSSR を持つ方式です。

アプリケーションのデバッグ

VARPG プログラムのデバッグを行うためには、アプリケーションのビルド時にデバッグ・コンパイラー・オプションを使用するようにしてください。デバッグ・オプションが設定されていない場合でも、プログラムでデバッガーを開始することはできますが、そのプログラムのアセンブラー・ビューを処理しなければなりません。

ソースに対してデバッガーを実行するには、最初にそのアプリケーションをビルドしなければなりません。プロジェクト・ビューから**「プロジェクト」>「ビルド」>「Windows NT/95/98」**を選択します。デバッガーを開始するためには、プロジェクト・メニューから**デバッグ**・メニュー項目を選択してください。デバッグの詳細については、237 ページの『第 10 章 アプリケーションのデバッグ』 を参照してください。

プロシージャーのデバッグ

DLL のコードをデバッグしたい場合には、いくつかの余分のステップに従う必要があります。:

1. メイン・アプリケーション（この例では MyMain) のデバッガーを開始します。
2. **「デバッガー - セッション制御」**ダイアログで、**「オカレンスのロード・ブレークポイントの設定...」**を選択します。
3. **「オカレンスのロード・ブレークポイント」**ダイアログが表示されたら、**「DLL ファイル名」**項目に DLL の名前 (MyFunc) を入力して**「OK」**を押します。
4. プログラムを実行します。

DLL でプロシージャーが呼び出されると、DLL がロードされることを示すデバッガー・メッセージ・ダイアログが表示されます。**「OK」**を押して、以下を実行します。

1. 「デバッグ - セッション制御」ダイアログを探し、右側パネルに DLL の名前の新規項目があることに注意してください。

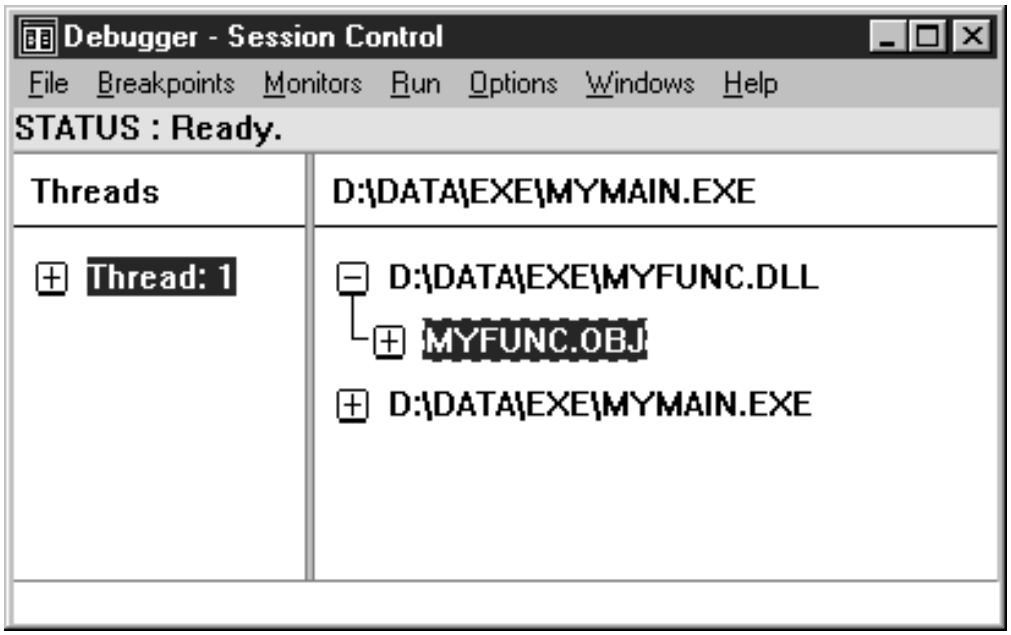


図 110. MyFunc.DLL オブジェクト・ファイルの選択

2. DLL 名の横の **+** 符号をクリックします。拡大されて、オブジェクト・モジュールの名前 MyFunc.obj が表示されます。
3. このオブジェクト・モジュール名をダブルクリックします。
4. デバッガーのソース・ビューに、DLL MYFUNC のプロシージャー ToUpper のソースが表示されます。

これで、ブレークポイントを追加し、DLL でプログラム変数を表示することができます。また、現在他の VARPG コンポーネントを **START** しようとしている場合、またはユーザー独自の 'C' 機能を呼び出そうとしている場合にも、上記のプロシージャーを使用してそれらをデバッグすることができます。

第 23 章 DBCS の考慮事項

2 バイト文字セット (DBCS) システムで VisualAge RPG を使用する計画がある場合には、以下を考慮する必要があります。

- コンパイラーは、リテラルではシフトインおよびシフトアウト文字を許可しません。ソースを VisualAge RPG プログラムにコピーするために VisualAge RPG エディターを使用して AS/400 メンバーをオープンする場合には、すべてのリテラルからシフトインおよびシフトアウト文字を除去しなければなりません。これらを除去しないと、コンパイル・エラーが起こります。
- コンパイラーは、リモート /COPY 機能を使用して検索する時に、VisualAge RPG ソース・メンバーからシフトインおよびシフトアウト文字を除去します。
- DBCS 文字は、アプリケーションのアイコン・ファイル名拡張子に使用することはできません。
- 非 DBCS 文字を含む VisualAge RPG アプリケーション名はビルド障害の原因となります。

VisualAge RPG DBCS データ・タイプのサポート

VisualAge RPG は多数の DBCS データ・タイプをサポートします。アプリケーションの実行時に DBCS データ・タイプを使用する場合には、AS/400 サーバーとワークステーションの間でデータが正しく転送されたことを確認するために、一定の規則に従う必要があります。次の DBCS データ・タイプがサポートされます。

DBCS 専用
このデータ・タイプのフィールドには DBCS データしか入っていないので、AS/400 データベースを使用する時に使用してください。これは、AS/400 データベースによってサポートされる J データ・タイプと同等です。

DBCS 択一
このデータ・タイプのフィールドには、すべての 1 バイトまたはすべての DBCS データが入っています。これは、AS/400 データベースを使う時に使用してください。これは、 AS/400 データベースによってサポートされる E データ・タイプと同等です。

DBCS 混用
このデータ・タイプのフィールドには、すべての 1 バイトまたはすべての DBCS データが入っています。これは、データを AS/400 データベースと交換する時に使用してください。これは AS/400 データベースによってサポートされる 0 データ・タイプと同等です。

AS/400 J、O、および E データ・タイプでは、DBCS データを SO (シフトアウト) および SI (シフトイン) 文字で囲む必要があります。ワークステーションの DBCS 択一、DBCS 混用、および DBCS ONLY フィールドでは、SO および SI 文字を使用してはいけません。これらのフィールドがサーバーへのデータの転送に使用される場合には、 SO および SI 文字が適宜追加されます。データがサーバーから検索

される場合には、SO および SI 文字は除去されて、VisualAge RPG フィールドには 2 個の単一バイトのブランクが埋め込まれます。

DBCS 択一、DBCS 混用、および DBCS ONLY フィールドは、VisualAge RPG アプリケーション内のパーツ名と同じ名前の文字フィールドとして表されます。

次の例では、DBCS データがサーバーに転送され（サーバーから転送され）る時にどのようにデータが変換されるかを示します。この例では、VisualAge RPG を使用して 10 バイトの DBCS ONLY フィールドが作成されます。このことは、それぞれの DBCS 文字には 2 バイトが必要であるので、フィールドに 4 文字の DBCS 文字を入れられることを意味します。余分の 2 バイトは、このフィールドがサーバーに転送される前に SO および SI 文字を挿入するために使用されます。サーバーに転送する前に、フィールドに次のデータが入っているとします。

```
DBDBDBDBblbl

ここで DB = 1 個の 2 バイト文字。
      bl = 1 個の単一バイトのブランク文字。
```

このフィールドはサーバーに転送される前に、DBCS データが SO および SI 文字で囲まれるように変換されます。単一バイトのブランクは無意味なものとして処理され、適切な SO および SI 文字と置き換えられます。したがって、このフィールドはサーバーに転送される前に次のように表示されます。

```
SODBDBDBDBSI
```

同じデータがサーバーから検索される場合には、SO および SI 文字が除去されて、フィールドには 2 個の単一バイトのブランクが埋め込まれます。

```
DBDBDBDBblbl

ここで DB = 1 個の 2 バイト文字
      bl = 1 個の単一バイトのブランク文字
```

注: DBCS ONLY、DBCS 混用、または DBCS 択一のデータ・タイプを表す文字フィールドには、フィールドをサーバーに転送するためと、フィールド内のデータをウィンドウに正しく表示するために、適切な数の単一バイトのブランクが埋め込まれなければなりません。

VisualAge RPG は、十分な単一バイトのブランクがあるようにします。 SETATR および GETATR 命令コードを使用してそれぞれ DBCS フィールドを設定し、あるいはDBCS フィールドから情報を検索する場合には、 SETATR および GETATR 命令のフィールドの長さがウィンドウのフィールドの長さと同じであることを確認する必要があります。同じでない場合には、サーバーとワークステーションの間で転送されないことがあります。

DBCS ONLY データ・タイプ

VisualAge RPG は、DBCS ONLY データ・タイプの使用時には、データがウィンドウ上のフィールドを介して追加されるか、SETATR 命令コードを使用して入力されるかに関係なく、以下になるようにします。

- 最小フィールド長は 2 です。これは、サーバーにデータを転送する時に SO および SI 文字を追加する十分な余地ができるようにしています。

- フィールドには有効な DBCS 文字が入れられます。それぞれの 2 バイトの対を検査して、有効な DBCS 文字が使用されるようにします。
- フィールドには適切にブランク文字が埋め込まれます。フィールドの長さより小さな値を入力すると、フィールドに最大 2 の長さまで 2 バイトのブランクが埋め込まれます。フィールドの最後の 2 バイトには単一バイトのブランクが埋め込まれます。

DBCS 択一データ・タイプ

DBCS 択一データ・タイプには、すべて 1 バイトのデータかすべて 2 バイトのデータのいずれかが入れられなければなりません。DBCS と単一バイトの混合データは使用できません。単一バイトのデータを使用する場合には、フィールドの最大長を使用して単一バイトのデータを入れることができ、データの最大長をサーバーに（サーバーから）転送することができます。

VisualAge RPG は、フィールドの最初の 2 バイトが DBCS 文字を表している場合には、データがウィンドウのフィールドを介して追加されるか、SETATR 命令コードを使用して入力されるかに関係なく、次の規則に従います。

- 最小フィールド長は 2 です。これは、サーバーにデータを転送する時に SO および SI 文字を追加する十分な余地ができるようにしています。
- フィールドには有効な DBCS 文字だけが入れられます。それぞれの 2 バイトの対を検査して、有効な DBCS 文字が使用されるようにします。
- フィールドには適切にブランク文字が埋め込まれます。フィールドの長さより小さな値を入力すると、フィールドに最大 2 の長さまで 2 バイトのブランクが埋め込まれます。フィールドの最後の 2 バイトには単一バイトのブランクが埋め込まれます。

DBCS 混用データ・タイプ

このフィールドには、任意の数の DBCS または単一バイト文字を交互に入れることができます。 VisualAge RPG は次の規則に従います。

- この文字フィールドには、常に単一バイトのブランクが埋め込まれます。
- DBCS モードに変わるたびに SO および SI 文字を考慮しなければなりません。DBCS 文字の入力と単一バイト文字の入力を切り替えるたびに、入力可能な最大長から 2 が引かれます。たとえば、VisualAge RPG を使用して長さが 20 の DBCS 混用フィールドが作成されるとします。このフィールドは次の値を持ちます。

```
DBsbDBsbDBsbDBsb.

ここで DB = 1 個の DBCS 文字。
      sb = 1 個の単一バイト文字。
```

これは、フィールドがサーバーに転送される前に次のように変換されるので、このフィールドの最大長です。

```
SODBSIsbSODBSIsbSODBSIsbSODBSIsb.

ここで SO = 1 個のシフトアウト文字。
      sb = 1 個のシフトイン文字。
```

20 バイトのフィールド全部が使用されます。

純粋の DBCS についての考慮事項

VisualAge RPG 言語と AS/400 データベースの両方が純粋の DBCS データ・タイプをサポートします。G またはグラフィック・データ・タイプです。 AS/400 サーバーまたはワークステーションでは、純粋な DBCS データには SO (シフトアウト) または SI (シフトイン)文字は不要です。 AS/400 サーバーとワークステーションの間でグラフィック・データが変換される場合には、SO および SI 文字は追加されたり除去されたりしません。

GUI 入力フィールドは、VisualAge RPG 言語でサポートされるグラフィック・データ・タイプを直接マップしません。フィールドを最大限に使用するためには、ウィンドウ上に文字項目を作成することをお奨めします。実行すると、GUI Designer と同じ名前で VisualAge RPG 文字フィールドが作成されます。次に、別個のグラフィック・フィールドを使用して、GUI Designer で作成された文字入力フィールドと対話することができます。入力フィールドと対話するためには、SETATR または GETATR 命令コードを使用してください。この方法で、SO および SI 文字を気にせずに、入力フィールドの前長を使用して DBCS 文字を保管することができます。

第 24 章 ユーザー・アプリケーションでのコードのマージ

プログラミング時には、プロジェクトまたはコンポーネントの 2 つ以上のパーツと関連するコードを一緒にマージすることができます。そのためには**マージ**機能を使用することができます。**プロジェクト**・プルダウン・メニューから「マージ」メニュー項目を選択してください。これで、**「コンポーネントのオープン - VisualAge RPG」**ダイアログ・ボックスが表示され、マージしたいプロジェクトを選択することができます。

このダイアログは、外観と機能が**「フォルダー／プロジェクトの検索」**ダイアログ・ボックスに似ています。入力フィールドに完全なパスを含むプロジェクトを指定するか、またはリスト・ボックスを使用してドライブを選択し、必要なプロジェクトを検索するフォルダーの選択に進むことができます。どちらの方法でも、**「GUI オブジェクト・ツリー・ビュー」**ウィンドウがオープンします。また、プロジェクト・オーガナイザーの**「ビュー」**プルダウン・メニューから**「GUI オブジェクト」**メニュー項目を選択することもできます。

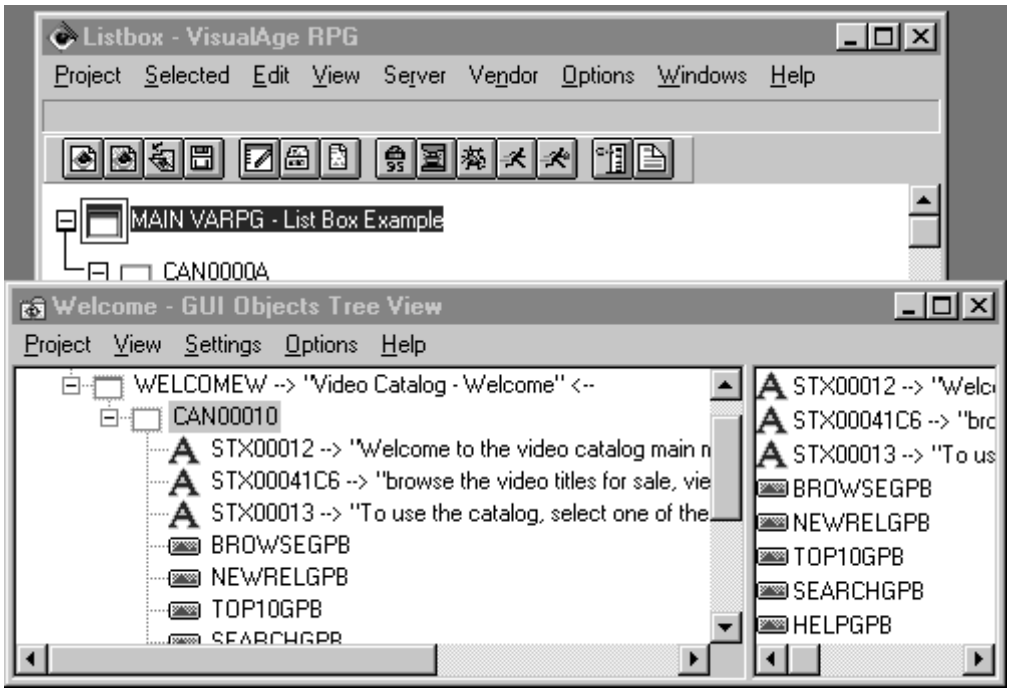


図 111. コード・マージ GUI オブジェクト・ツリー・ビュー

このウィンドウには 2 つのビューが表示されます。1 つは左側にあってマージを選択したプロジェクトのツリー・ビューが入っています。もう 1 つは右側にあって、左側のツリー・ビューで選択したパーツのすべての子が入っています。 Windows Explorer でできるように、ウィンドウの右側では複数のパーツを選択することができます。このビューは、ここ（左側または右側ペインのいずれか）から項目を選択して、現行プロジェクトのツリー・ビューまたは設計ウィンドウに指示してクリックできるので、追加のパーツ・パレットとして使用できます。これは、パーツはフ

レーム・ベースのパーツにしか入れられず、フレーム・ベースのパーツはプロジェクト・ツリーのルートにしか入れられない点で、パーツ・パレットと同様に機能します。GUI と関連コードをマージすると、ビルド・プログラムは、マージ結果に満足しない場合にその作業のバックアップを提供するために、作業中の現行プロジェクトを強制保管します。

GUI のレイアウトに加えて、マージでは、リンクされたアクション・サブルーチン、ヘルプ・パネル、技術上の説明、メディア・ファイルに対する参照、参照されたユーザー・サブルーチン、およびユーザー・メッセージがコピーされます。このような特定の場合のコードのマージについては、留意しなければならないいくつかの規則があります。

- リンクされたすべてのアクション・サブルーチンがコピーされます。
- 参照されたメディア・ファイルは、その参照と一緒にはコピーされません。これはユーザーの責任において実行します。
- ファイル仕様および定義仕様は現行プロジェクトにはコピーされません。これもユーザーの責任において実行します。
- コピーされるアクション・サブルーチンで参照されているユーザー・サブルーチン、RPG プロシージャー、およびユーザー・メッセージもコピーされます。これには、EXSR または CASxx 命令コードで使用されたユーザー・サブルーチン、CALLP 命令コードで参照された RPG プロシージャー、および DSPLY 命令コードで参照されたユーザー・メッセージに対するすべての参照が含まれます。
- 名前変更されたパーツの場合には、そのパーツを参照し、標準形式に適合した名前を持つすべてのアクション・サブルーチンが名前変更されます。たとえば、次のソース・コードは標準形式に従うものとして名前変更されます。この形式の要件は、パーツ名およびウィンドウ名がそのパーツの見つかる場所に直接対応していることです。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq----
C     PSB000000C    BEGACT    PRESS         FRA000000B
    .
    .
    .
C                   ENDACT
```

- コピーされるユーザー・メッセージは、最初にコピーされたメッセージから順番に名前変更されます。マージされたすべてのユーザー・メッセージは、現行プロジェクトの最後のメッセージの直後から始めて順次に番号付けされます。メッセージ ID は、それらを参照する DSPLY 命令コードで変更されます。
- ユーザー・サブルーチンで名前の競合が検出された場合には、その名前は変更されないで、マージ・ログ・ファイルに入っているリストに追加されます。このログは、「コードのマージ結果」ウィンドウにも表示されます。マージ・ログ・ファイルは、プロジェクト・ディレクトリー **projectname.mrg** のファイル名で入れられます。この場合の projectname はそのプロジェクトの名前です。このファイルは、同じプロジェクトで複数のマージが実行されると上書きされます。このファイルは自動的には付加されません。次の例に、サンプル・マージ・ファイルのリストを入れてあります。

以下のパーツがターゲット・プロジェクトにコピーされました。

ソース名	ターゲット名
SEARCHW:CAN00023	SEARCHW:CAN00023
SEARCHW:SEARCHW	SEARCHW:SEARCHW
SEARCHW:SEARCHGB	SEARCHW:SEARCHGB
SEARCHW:STX00071	SEARCHW:STX00071
SEARCHW:TITLECB	SEARCHW:TITLECB
SEARCHW:STX00073	SEARCHW:STX00073
SEARCHW:STX00074	SEARCHW:STX00074
SEARCHW:STX00075	SEARCHW:STX00075
SEARCHW:CATCB	SEARCHW:CATCB
SEARCHW:DIRCB	SEARCHW:DIRCB
SEARCHW:ACTORCB	SEARCHW:ACTORCB
SEARCHW:SEARCHPB	SEARCHW:SEARCHPB
SEARCHW:CANCELSEPB	SEARCHW:CANCELSEPB
SEARCHW:HELPPB	SEARCHW:HELPPB
SEARCHW:STX00082	SEARCHW:STX00082

以下のヘルプ・パネルがターゲット・プロジェクトにコピーされました。

ソース名	ターゲット名
24.SEARCHW	88.SEARCHW
79.SEARCHPB	99.SEARCHPB
80.CANCELSEPB	100.CANCELSEPB
81.HELPPB	101.HELPPB

ソース・コードのマージ:

Action Subroutine	CATW+CLOSE+CATW
Renaming To	SEARCHW+CLOSE+SEARCHW
Action Subroutine	TITLECB+CREATE+SEARCHW
Action Subroutine	DIRCB+CREATE+SEARCHW
Action Subroutine	ACTORCB+CREATE+SEARCHW
Action Subroutine	CATCB+CREATE+SEARCHW
Action Subroutine	TITLECB+ENTER+SEARCHW
Action Subroutine	CATCB+SELECT+SEARCHW
Action Subroutine	DIRCB+SELECT+SEARCHW
Action Subroutine	ACTORCB+SELECT+SEARCHW
Action Subroutine	CANCELSEPB+PRESS+SEARCHW
Action Subroutine	SEARCHPB+PRESS+SEARCHW
Message	*MSG0001 -> *MSG0003
Message	*MSG0001 -> *MSG0003
Message	*MSG0001 -> *MSG0003
Message	*MSG0001 -> *MSG0003
User Subroutine	WRTBRSFSR
User Subroutine	CASECAT
User Subroutine	CKCRITERIA
User Subroutine	DSPBROWSE
Message	*MSG0001 -> *MSG0003
User Subroutine	BRACTION
User Subroutine	BRCHILDREN
User Subroutine	BRSCIFI
User Subroutine	BRCOMEDY
User Subroutine	BRHORROR
User Subroutine	BRWESTERN
User Subroutine	BRROMANCE
User Subroutine	BRCLASSIC

次のメッセージがターゲット・プロジェクトにコピーされました。

```
    Source Message                  Target Message

    1                               3
```

次のパーツの技術上の説明がターゲット・プロジェクトにコピーされました。

ソース名	ターゲット名
SEARCHW:SEARCHPB	SEARCHW:SEARCHPB
SEARCHW:CANCELSEPB	SEARCHW:CANCELSEPB
SEARCHW:HELPPB	SEARCHW:HELPPB

パーツ名競合の解決には次の規則が適用されます。
- マージされたパーツまたはウィンドウは名前変更されます。
- ウィンドウが名前変更されると、そこに入っているすべてのパーツが新しいウィンドウ名を継承します。
- 名前変更されたウィンドウまたはパーツにリンクされたアクション・サブルーチンは、標準命名形式に従っていれば、名前変更されて再リンクされます。
- 名前変更されたパーツを参照する GETATR または SETATR 命令コードが入っている演算仕様は変更されます。
- マージ・プロセスでは、マージされるコードのパーツ参照をパーツ名変更をアドレスするように訂正しようとします。

マージされたパーツに使用されるユーザー・メッセージもコピーされます。次の規則が適用されます。
- ユーザー・メッセージに名前の競合がある場合には、名前変更が行なわれます。
- 名前変更されたユーザー・メッセージに対する参照が更新されます。

第 25 章 ベンダー・プラグイン

プラグインは、サード・パーティーの開発者が作成したアプリケーションで、VisualAge RPG に追加の機能を提供するために作成されます。名前の付いたパーツのリストから選択した属性の値を設定するコード行や、プログラマーがページ見出しやフッターを組み込むように形式設定された RPG ソースを印刷できるプロシージャーを挿入するなどの、広範なタスクを自動化することができます。

ベンダー・プラグインの追加

ユーザーが作成し、あるいはサード・パーティーの開発者から取得したプラグインを追加するには、以下を実行してください。

1. **「ベンダー」**プルダウン・メニューをクリックします。
2. **「プラグイン」**項目を強調表示して、プラグイン・サブメニューを表示します。
3. **Add plugin...** コマンドをクリックして、**「プラグインの追加」**ウィンドウをオープンします。
4. ウィンドウに表示されたファイルから追加したいベンダー・プラグインを選択します。表示されたファイルには .plg の拡張子が付いています。

ベンダー・プラグインの呼び出し

VisualAge RPG GUI Designer にプラグインを追加したら、その機能を呼び出す必要があります。これを行うための一番簡単な方法は、ベンダーが定義したプラグイン・メニュー項目を選択する方法です。このメニュー項目は**「ベンダー」**プルダウン・メニューまたは**「選択済み」**プルダウン・メニューにあるか、あるいはパーツのポップアップ・メニューからです。場合によってはプラグインにメニュー項目が定義されていないこともあり、GUI Designer の外部からこのプラグインを呼び出すことが必要になります（この呼び出し方法の例は、LPEXSAMP サンプル・プラグインを参照してください）。一例として、VisualAge RPG で提供される 1 つまたは複数のベンダー・プラグインのサンプルを追加します。これらのプラグインを呼び出すには、以下を実行してください。

1. **「ベンダー」**プルダウン・メニューをクリックします。
2. **「プラグイン」**項目を強調表示して、プラグイン・サブメニューを表示します。
3. プラグインと一緒に追加されたメニュー項目を強調表示します。このメニュー項目の名前はさまざまです。プラグインの開発者によって作成されるためです。開発者／ベンダーによって、別のサブメニューが作成されることもあります。その場合にはステップ 4 に進み、それ以外はメニュー項目をクリックしてプラグインを呼び出します。
4. サブメニューから該当するメニュー項目をクリックしてプラグインを呼び出します。

ベンダー・プラグインの管理

VisualAge RPG で機能するベンダー・プラグインがある場合には、開発者、プラグインが実行する内容についての開発者の記述、あるいはプラグインに関連した dll などの、プラグインに関する情報を見つけることができます。この情報は、**「プラグインの管理」**ウィンドウで取得できます。

「プラグインの管理」ウィンドウをオープンするには、以下を実行してください。

1. **「ベンダー」**プルダウン・メニューをクリックします。
2. **「プラグイン」**項目を強調表示して、プラグイン・サブメニューを表示します。
3. **Manage plugins...** コマンドをクリックして、**「プラグインの管理」**ウィンドウをオープンします。

第 26 章 プラグインの作成

ユーザー自身の特別のニーズを指示したり、VisualAge RPG の補足としてそのプラグインから利益を受ける他のプログラマーに送るために、プラグインを作成することができます。プラグインは、VisualAge RPG, VisualAge C++ または REXX のいずれかを使用して作成することができます。以下のステップは、VisualAge RPG コードを使用したプラグインの作成プロセスを略述したものです。VisualAge C++または REXX でプラグインを作成するための例外や追加のガイドラインは、後の追加の項で示します。

VisualAge RPG を使用したプラグインの作成

プラグインを作成するには 2 つのコンポーネントが必要です。実行可能ファイルとプラグイン情報ファイルです。実行可能ファイルは、必要な機能を実行するコンパイル済みソース・コードから構成されます。プラグイン情報ファイル (.plg) は、GUI Designer と実行可能ファイルの間のインターフェースとしての働きをします。これには、GUI Designer に追加されるプルダウン・メニュー項目の定義や、実行可能ファイルの実際の呼び出しなどの重要な情報が入っています。

この 2 つのどちらのファイルを先に作成するかは問題ではなく、プラグインが機能するためには両方が存在しなければなりません。この例では .plg ファイルからスタートします。

.plg ファイルの作成

.plg ファイルは、プラグインとDesigner の間のインターフェースとしての働きをします。必要な DLL の所在、関連するヘルプ・ファイルの所在、およびプラグイン自体とベンダーの名前などの編成上の情報が入っているのは ASCII ファイルです。.plg ファイルには、プラグインとDesigner とのインターフェースに必要な情報も入っています。これには、コマンド行ストリング、メニュー項目に必要なテキスト、およびアクセラレーターとしての働きをするキーの組み合わせが含まれます。次のキーワードのリストで、各種のパラメーター（適用される規則、パラメーターが必須か任意か、およびユーザーが提供する必要がある情報を含む）について説明します。このキーワードのリストの後の例は、終了時にこのファイルがどのようになるかを示しています。

注: プラグインの値に使用されるキーワードとパラメーター値の間のスペースは任意です。例で使用しているスペースは表示目的のみで設計されています。

Alternate_Paths

これは、以下で説明するプラグイン DLL のロード時に使用する相対パスを示すストリングです。このフィールドは任意指定です。

ストリングの形式は次のとおりです。

```
"&1\relativePath1;&1relativePath2;...;&1\relativePathN;"
```

ここで、&1\ は .plg ファイルへの絶対パスに変換されます。このことは、c:\myplugins などのディレクトリーにインストールする一組のプラグインがある場合に、それらはすべて共通 DLL c:\myplugins\plugutil\plugutil.dll にリンクされて、個々のそれぞれのプラグインが c:\myplugins\plugN\plugN.plg にある場合には、代替パス &1\..\util; を指定することができ、これは c:\myplugins\plugN\..\util; に変換されて PATH に付加されてからその DLL がロードされることを意味します。このキーワードを使用する場合には、引用符が必要です。

ストリングの代わりに、外部ソースで指定されたパスをポイントするリソース id を使用することもできます。

DLL_Names

このフィールドの値は、このプラグインの必須 DLL の名前を定義するストリングです。これは任意指定フィールドですが、使用する場合には、プラグインで使用する名前を持った DLL がなければなりません。また、mri.dll も命名することができます。mri.dll 名は任意指定ですが、plugin.dll なしで組み込むことはできません。両方が組み込まれると、それらはスペースで区切られます。DLL ファイルの名前には、プラグイン・ファイルの所在に対するパスを含めることができます。

plugin.dll
これは、プラグインのコードが入っている DLL の名前です。プラグインが DLL 内の機能呼び出しでない場合には、これは省略することができます。

mri.dll mri（変換可能ストリング）が別の DLL 内にある場合には、ここでその DLL に名前を付けます。 DLL は、後に続くストリングを DLL 内に入れることができるように、最初に指定します。

.plg ファイル内でさらにストリングが必要な場合には、引用符で囲まれたストリングが指定されていなければ、指定された値は mri.dll または plugin.dll のいずれかのストリング・リソース id と見なされます。

Vendor_Name

これは、二重引用符で囲まれたベンダーの名前か、ストリングのリソース id のいずれかです。このフィールドは任意指定ですが、指定することをお奨めします。このストリングの例は次のとおりです。

```
Vendor_Name:            "Plug-Me-In Inc."
```

Plugin_Name

これは、引用符で囲まれたプラグインの名前か、ストリングのリソース id のいずれかです。このフィールドは任意指定ですが、指定することをお奨めします。ストリングを使用したこのフィールドの例は次のとおりです。

```
Plugin_Name:            "Who Am I?"
```

Help_File

このフィールドは任意指定で、メニュー項目のヘルプを表示するために使用する Windows の .hlp ファイルを識別します。これは、引用符で囲まれていないストリングで、ヘルプ・ファイルへの相対パスを含みます。

Unloading_Function

このフィールドはオプションです。**Unloading_Function** フィールドは、**Unloading_Command_Line** フィールドと一緒に使用することはできません。このフィールドは、プラグインが終了または除去された後で変更しあるいは除去する必要がある情報または表示エレメントがある場合に使用されるだけです。引用符で囲まれたこのストリングは、使用される機能を指示します。この機能は、プラグインに付随する DLL に入っていなければなりません。

Unloading_Function は、プラグインがアンロードされるときに呼び出される機能の名前です。これは次のシグニチャーを持ちます。

```
unsigned long
      unloadFunctionName(
            const char*     ppluginPath_,
            const char*     ppluginStub_,
            const char*     pdllPath_,
            const char*     builderId_,
            int             remove_)
```

パラメーターは次のとおりです。

ppluginPath_
呼び出されるプラグインへの完全修飾パス。最後の円記号 (\) まで。

ppluginStub_
プラグインの残りのファイル名（たとえば、″myplug.plg″）。

pdllPath_
VARPG の公開メソッドが入っている DLL への完全修飾パス。

builderId_
VARPG がビルダーを識別するために使用するストリングで、ビルダーとの連絡時にプラグインによって使用されます。

remove_

0 ビルダーがシャットダウン中です。

1 プラグインを一緒に除去するようにユーザーが要求しています。この場合には、プラグインは、この時点でレジストリーに保管されているすべての情報を除去します。

Return value

0 成功

1 失敗または拒否

プラグインがこの機能から 1 を戻すと、ビルダーはその DLL をアンロードしてプラグインを強制的に除去するオプションをプロンプトで出すことがあります。その場合には、プラグインがDesigner を破壊している可能性があります。その場合には、Designer を再始動してください。

Unloading_Command_Line

上記のように、このフィールドは **Unloading_Function** フィールドと同時に使用することはできません。

このオプションを使用する場合には、コマンド行から実行されたかのように実行されるストリングを指定します。たとえば、 Netscape はストリング netscape.exe を指定して開始することができます。

このメソッドにより、DLL の機能に使用可能なものと同じパラメーターのセットを取得することができます。これは、置き換え変数を定義することによって達成されます。指定されたストリングに '&0'、'&1'、'&2'、 '&3'、'&4'、または '&5' が見つかると、それは以下と置き換えられます。

&0 ppluginPath_

&1 ppluginStub_

&2 pdllPath_

&3 builderId_

&4 remove_

&5 GUI Designer のルート・ディレクトリーへのパス

IBM_PluginInterface | PluginInterface

これは不必要な拡張機能です。このフィールドによって、プラグインをプログラマブル・コンポーネントとして公開することができます。この 2 つのオプションを同じ .plg ファイルで使用することはできません。これらのインターフェースのどちらも指定する理由がない場合には、指定しないでください。

これらのオプションの一方または他方を使用すると、他方のプラグインがターゲット／コマンド／パラメーター・インターフェースを通してこのプラグインと対話する場合に、指定された機能名が使用されます。

この機能のシグニチャーは次のとおりです。

```
unsigned long __stdcall IBMtargetCommandFunction(
                    const IString&  pluginPath_,
                    const IString&  dllPath_,
                    const IString&  builderId_,
                    const IString&  target_,
                    const IString&  command_,
                    IString&        arguments_);
```

IBM_ style 機能の場合には、入出力に arguments_ is が使用されます。

非 IBM style 機能の場合には、シグニチャーは次のようになります。

```
unsigned long __stdcall targetCommandFunction(
                    const char*     ppluginPath_,
                    const char*     pdllPath_,
                    const char*     pbuilderId_,
                    const char*     ptarget_,
                    const char*     pcommand_,
                    const char*     parguments_,
                    char**          ppreturnString_);
```

このケースでは、戻りストリングがあると ppreturnString_ は GlobalAlloc(GMEM_FIXED, [bufferSize]) を使用してプラグインがメモリーを割り振るので、VARPG は終了時にそのメモリーを割り振り解除することができます。戻りストリ

ングが必要ない場合には、このパラメーターは無視することができます。以下は、このコマンドの例です。

```
{
  IString returnString = ...;
  ...

  *ppreturnString = GlobalAlloc( returnString. length() + 1);
  strcpy( *ppreturnString, returnString);
}
```

IBM_PluginInterface 機能をサポートするプラグインの例は、x:\...\WDSC\samples\vndplugs\lpexsamp ディレクトリーにある LPEXSAMP サンプルを見てください (x は、VisualAge RPG をインストールしたドライブ名に対応します)。

Begin_Details ... End_Details

このタグ間に、**「プラグインの管理」**ダイアログにこのプラグインの情報が表示されるときに表示したい内容を入力します。プラグインの目的および使用法についての簡単な説明を入れることができます。ここにはテキストを入れたり、mri.dll 用に記述した String/Resid 形式を使用することができます。このフィールドは任意指定ですが、指定することをお奨めします。

Function_Name

これは、メニュー項目が起動されたときに plugin.dll で呼び出される機能の名前です。このフィールドを使用するか、または **Command_Line** フィールドを使用することができます。両方を使用することはできません。シグニチャーは次のとおりでなければなりません。

```
unsigned long
      pluginFunctionName(
            const char*     ppluginPath_,
            const char*     pdllPath_,
            const char*     builderId_,
            unsigned long   menuContextId_,
            const char*     partsIds_);
```

パラメーターは次のとおりです。

ppluginPath_
unloadFunctionName() と同じ意味です。

pdllPath_
unloadFunctionName() と同じ意味です。

builderId_
unloadFunctionName() と同じ意味です。

menuContextId_
このプラグインが呼び出されるメニューのタイプを表す符号なしの長い値。この値は partsIds_ の意味を決定します。使用できる値は次のとおりです。

1 メニュー・バーからプラグインが呼び出され、(すなわち、プロジェクト範囲プラグイン）partsIds_ は空のストリングです。

2 単一選択パーツとしてプラグインが呼び出され、(すなわち、単一選択範囲プラグイン) partsIds_ には選択したパーツの ID が入っています。

4 共同選択パーツのグループとしてプラグインが呼び出され、(すなわち、複数選択範囲プラグイン) partsIds_ は選択したパーツのブランクで区切られたパーツ ID のセットが入っているストリングです。

8 このプラグインは GUI Designer が開始されると呼び出されます。

partsIds_

これは、menuContextId_ で指示されたように機能呼び出しが適用される 1 つまたは複数のパーツを表すストリングです。partsIds_ 内で、個々のそれぞれのパーツ ID は所定のパーツの子と親の階層を表す、一連のドット区切りの、符号なしの長い値 (たとえば、432.5632.612) です。示された例では、612 はパーツの ID で、5632 はその親の ID、432 はその親の親の ID です。

Command_Line

このオプションを使用する場合には、コマンド行から実行されたかのように実行されるストリングを指定します。たとえば、Netscape はストリング `netscape.exe` を指定して開始することができます。

このメソッドにより、DLL の機能に使用可能なものと同じパラメーターのセットを取得することができます。これは、置き換え変数を定義することによって達成されます。指定されたストリングに '&0'、'&1'、'&2'、 '&3'、'&4'、または '&5' が見つかると、それは以下と置き換えられます。

&0 ppluginPath_

&1 pdllPath_

&2 builderId_

&3 menuContextId_

&4 partsIds_

&5 GUI Designer のルート・ディレクトリーへのパス

上の Netscape の例を使用して、ベンダーがプラグインと一緒に HTML ファイルを提供し、その特定のメニュー項目がその HTML ファイルを表示するためのものであるとします。また、そのプラグイン・ファイルは d:\vendor\plugins に入っていて、HTML ファイルは d:\vendor\plugins\htmlsrc\plugpage.html にあるとします。プラグインをこの Web ページに表示するためには、コマンド行の定義は次のようになります。

```
netscape &0htmlsrc\plugpage.html
```

これは次のように拡張されて実行されます。

```
netscape d:\vendor\plugins\htmlsrc\plugpage.html
```

Menu_Name

これは、メニュー項目がどうあるべきかを示すストリングまたはストリング・リソース id です。これらのストリングの形式は次のとおりです。

```
submenu1/submenu2/.../submenuN/menuitem
```

ここで、submenu1 から submenuN までは任意指定のサブメニューです。

たとえば、次のようになります。

```
Menu_Name:              "Plug-Me-In Inc./Who am I?"
```

ここで、″Plug-Me-In Inc.″ はサブメニューで、″Who am I?″ はメニュー項目です。

Menu_Info_Strings

これは、**Menu_Name** で指定された対応するサブメニュー／メニュー項目に関連づけられたストリングまたはストリング id のリストです。関連づけは逆方向に行われます。

たとえば、**Menu_Name** に 1 つのサブメニューと 1 つのメニュー項目を指定して、**Menu_Info_Strings** に 1 つのストリングしか指定しないと、**Menu_Info_Strings** に指定したストリングはメニュー項目と関連づけられてサブメニューは無視されます。(前のメニュー項目の追加で所定のメニュー項目の info-area ストリングを定義することは可能です。)

Supported_Menus

Function_Name で述べているように、menuContextId_ はメニューのタイプを示します。**Supported_Menus** は、この特定の項目を追加するメニューを示します。

Help_Id

ヘルプ・ファイルが指定されていて、このコマンドに関連づけられたヘルプがある場合には、**Help_Id** の ulong_panel パラメーターでここにヘルプ id を指定します。optional_force_window_parameter が指定されて、それがゼロ以外の値である場合には、ヘルプはデフォルトのコンテキスト・ポップアップでなく、完全なヘルプ・ウィンドウに表示されます。このフィールドは次の形式を取ります。

```
Help_Id:                ulong_panel optional_force_window_parameter
```

そして、実際のコーディングのサンプルは次のようになります。

```
Help_Id:       1000 1
```

アクセラレーター

この任意指定フィールドは、項目と関連づけられるアクセラレーターを指定します。これは、F1 から F12 までのうちの 1 つと、その後に続く 1 つまたは複数の修飾子 (SHIFT、ALT、CONTROL) から構成されます。

注: <F1/10>、<Alt-F5/7/8/9/10>、および <Shift-F9/10> はすでにDesigner によって予約されているので、指定しても無視されます。

この機能を使用するには、次の情報を指定します。

```
Accelerator:            [F1 | F2 | F3 | ... | F12] [SHIFT] [CONTROL] [ALT]
```

このフィールドは、使用時には次のサンプルのようになります。

```
Accelerator:            F8 Shift
```

End_of_Definition

これは、構文解析プログラムに対して機能定義が終了して新しい機能定義が開始されることを示します。

.plg ファイルのテンプレートおよびサンプル

前に説明したフィールドを使用して GUI Designer の .plg ファイルを作成する場合には、以下の形式に従う必要があります。

注: 少なくとも 1 つの Function_Name または Command_Line 定義がなければなりません。許可される最大数に制限はありません。

```
// Lines that begin with double forward-slashes are ignored
// (that is, treated as comments)

Alternate_Paths:        string_or_resId
dll_Names:              plugin.dll mri.dll
Vendor_Name:            string_or_resId
Plugin_Name:            string_or_resId
Help_File:              helpfile.hlp
Unloading_Function:     "unloadingFunction"
       (or)
Unloading_Command_Line: "command line invocation with substitution symbols &0, &1, &2, &3, &4, &5"
IBM_PluginInterface:    "IBMtargetCommandFunction"
       (or)
PluginInterface:        "targetCommandFunction"


Begin_Details:
   .
   .
   Optional text outlining the function of the plugin.
   .
   .
End_Details:


Function_Name:          "functionName1"
       (or)
Command_Line:           "command line invocation1 with substitution symbols &0, &1, &2, &3, &4"
Menu_Name:              string_or_resId
Menu_Info_Strings:      string_or_resId string_or_resId ...
Supported_Menus:        menuContextId1 menuContextId2 ...
Help_Id:                ulong_panel optional_force_window_parameter
Accelerator:            [F1 | F2 | F3 | ... | F12] [SHIFT] [CONTROL] [ALT]
End_of_definition:
```

注:

- ファイル名はすべて .plg ファイルの所在に対して示されます。
- アンローダーは任意選択ですが、アンローダーを持つことにした場合には、持てるのは 1 つだけです。
- 機能名またはコマンド行のいずれかを指定することができます。

以下に、簡単な .plg ファイルの特別の例を示します。VisualAge RPG では、いくつかのプラグインのサンプルが提供されています。ファイルは、VisualAge RPG がインストールされたワークステーションの X:\...\WDSC\samples\vndplugs\ ディレクトリーに入っています (x はドライブ名に対応します)。

```
// Print project plugin

Vendor_Name:            "Plug-Me-In Inc."
Plugin_Name:            "Who Am I?"
Begin_Details:

Who Am I?

  This plugin will display information about the current
  project including its directory name and file name.

End_Details:

Command_Line:           "d:\myproj\whoami\rt_win32\whoami.exe &1 &2 &4"
Menu_Name:              "Plug-Me-In Inc./Who am I?"
Supported_Menus:        1
Accelerator:            F7 Shift
End_of_Definition
```

.EXE ファイルの作成

GUI Designer の .EXE ファイルを作成するためには、以下に注意する必要があります。

VisualAge RPG を使用してプラグインを作成する場合には、*component パーツを使用してDesigner と対話します。**PlugDLL**、**PlugId**, **PlugCmd**, **PlugRC**、および **PlugResult** 属性の値を設定して、設計機能とプラグインの間で必要なすべての情報を伝えることができます。

作業プラグインを作成するには、設計機能に次の情報を提供して、適切な通信を確立する必要があります。

builderId_
: これは、プラグインの呼び出し時に設計機能が提供したものと同じ id です。

target_
: これは、ユーザーが対話したい設計機能の局面を表すストリングです。

command_
: これは、設計機能に行わせたい特定のアクションです。

parameters_
: コマンドで必要な引き数。

注: VARPG プログラムでは、builderId_ は *component の **PlugId** 属性に対応します。プラグイン・インターフェースを呼び出すためには、最初に **PlugId** を設定し、**PlugDLL** も設定する必要があります。**PlugDLL** は、VARPG ランタイムにビルダーのプラグイン・インターフェースが入っている dll の所在を示します。コマンドを出すと、最初に区切り文字にブランクを使用して

target_、command_、および parameters_ の値を連結し、次にその結果を使用して *component の **PlugCmd** 属性を設定します。

この結果およびエラー・コードが戻されます。機能呼び出しの場合には、結果のストリングを入れるためにパラメーターの 1 つが設定されて、エラー・コードが入った符号なしの長い値が戻されます。以下は、すべてのコマンドに共通の基本的な戻りコードの一部です。追加のエラー・コードがあれば、ターゲットおよびコマンドのそれぞれのテーブルで定義されます。

戻りコード

意味

0 すべてが正常で、コマンドが処理されました。

1 ターゲットが認識されませんでした。

2 ターゲットがコマンドを認識していません。

4 ビルダーが見つかりません。

5 何らかの不明エラーが起こって、コマンドの結果は信頼できません。

ターゲット／コマンドおよび関連した戻りコード

以下に、有効なターゲットとコマンドのリスト、それにパラメーターと戻り値の意味を示します。

表 13. ターゲット: Project

コマンド	パラメーター	意味／戻り値
ビルド	[win32|java] デフォルト: win32	プラットフォームが ″win32″ か ″java″ かによって、プロジェクトの win32 または java バージョンをビルドします。戻り値はなく、即時に（すなわち、ビルドが完了する前に）戻します。
BuildOptions	[win32|java] デフォルト: win32	プラットフォームが ″win32″ か ″java″ かによって、win32 または java のビルド・オプションを示します。戻り値はありませんが、（モーダル）ダイアログが消えるまで戻しません。
CursoredPart	なし	現在カーソルが置かれているパーツの partId が入っているストリングを戻します。設計ウィンドウがオープンしていない場合、またはオープンしている設計ウィンドウがアクティブの設計ウィンドウでない場合には、これは空のストリングになります。
ExpandAll	[1]	このパラメーターが 1 の場合には、ツリー・ビュー全体が拡大され、そうでない場合には縮小されます。

表 13. ターゲット: Project (続き)

ForceOpen	[projectName]	現行のプロジェクトを保管する必要があるかどうかを調べないで、指定されたプロジェクトをオープンします。ForceOpen が正常に実行されたことを示す 1 または ForceOpen が失敗したことを示す 0 を戻します。
Get	ProjectDir	現行のプロジェクトのルート・ディレクトリーを戻します。
	ProjectFileName	現行のプロジェクトの完全修飾 .IVG ファイル名を戻します。
	ProjectTargetName	プロジェクトのビルド時に生成されるファイルの名前（たとえば、″myproj.exe″）を戻します。
	ProjectTitle	現行のプロジェクトのタイトルを戻します。
	ProjectFileStub	現行のプロジェクトの名前のファイル名（拡張子を除いた）（たとえば、″myproj″）を戻します。
IsSaveRequired	なし	プロジェクトが変更されたことを示す 1 またはオープンされてから触れられていないことを示す 0 を戻します。
IsTemporary	なし	これが名前のないプロジェクトの場合には 1 その他の場合には 0 を戻します。
MostRecentlyUsed	n	n が 1 以上の場合には、n 番目の最近オープンされたプロジェクトを戻します。この指標が範囲外の場合には、空のストリングを戻します。
オープン	projectName	別のプロジェクトをオープンする前に、このプロジェクトを保管するかどうかを確認します。プロジェクトが正常にオープンされた場合には 1 を、そうでない場合には 0 を戻します。

表 13. ターゲット: Project (続き)

PartId	partName [windowName |[0|1|2]]	パーツ名が指定された場合に partId を戻します。 windowName を指定するとパーツの id を戻し、該当するパーツがない場合には空のストリングを戻します。 searchTypeを指定すると、指定された名前のパーツを検索するときに、次の規則が使用されます。 0（デフォルト）- 指定された名前の最初のパーツを戻します。 1 - 指定された名前のすべてのパーツを戻します。 2 - その名前のパーツが 1 つしかない場合にはそれを戻し、そうでない場合には何も戻しません。
PromptedSave	なし	プロジェクト名を求めるプロンプトを出して、そのプロジェクトを保管します。正常に保管が行われたことを示すためには 1 を、そうでない場合には 0 を戻します。
PromptExisting	なし	既存のプロジェクトを求めるプロンプトを出します。プロジェクトのファイル名を戻します。
実行	なし	現行のプロジェクトを実行します。
保管	なし	現行のプロジェクトを保管します。
別名保管	projectName	指定されたプロジェクト名で現行のプロジェクトを保管します。
SelectedParts	なし	プロジェクトのツリー・ビューで現在選択されているすべてのパーツの partIds が入ったストリングを戻します。

表 14. ターゲット: PartClass

コマンド	パラメーター	意味／戻り値
AllAttributes	ClassName	指定された ClassName でサポートされる属性のリストを戻します。
AllClasses	なし	使用可能なすべてのパーツ・クラスのリストを戻します。それぞれのリスト項目は、複数のワードから構成されている場合があるので（たとえば、ベンダー・パーツの場合）、二重引用符で囲まれます。

表 14. ターゲット: PartClass (続き)

AllEvents	ClassName	指定されたクラスの登録済みのすべてのイベントを戻します。
IBMClasses	なし	IBM 提供の、ベンダー以外の、すべてのパーツ・クラスを戻します。
IconDll	ClassName	指定されたパーツ・クラスを表すアイコンが入っている dll のパスを戻します。
IconId	ClassName	指定されたクラスの (″IconDll″ で指定された dll の) アイコンのリソース id を戻します。
IsType	TypeName	指定された ClassName のクラスが指定されたタイプであることを示すためには 1 を、そうでない場合には 0 を戻します。TypeName に使用できる値には、Frame、Canvas、MenuBar、NoteBook、 NoteBookPage、PopUpMenu、 SubMenu、MenuItem、 Subfile、および SubfileEntryField が含まれます。
VendorClasses	なし	使用可能なすべてのベンダー・パーツのパーツ・クラスを戻します。

表 15. ターゲット: Part

コマンド	パラメーター	意味／戻り値
ActionSubroutine	partId eventName	リンクされたアクション・サブルーチン eventName を探すか、またはまだ存在しない場合にはリンクを作成してスキャンします。
ActionSubroutines	partId	このパーツに定義されたアクション・サブルーチンのリストを戻します。
AllEvents	partId	指定されたパーツの登録済みのすべてのイベントを戻します。
Children	[partId]	指定されたパーツのすべての子を列挙するブランクで区切られた partIds のリストを戻します。partId が指定されていない場合には、プロジェクトのすべてのウィンドウのリストが戻されます。
ClassName	partId	指示されたパーツのクラス名を戻します。

表 15. ターゲット: Part (続き)

CreateChild	partId className	指定されたパーツの子として、指定されたクラス名 className のパーツを作成します。新しく作成されたパーツの partID を戻します。
CreateFrame	ClassName	指定されたクラスのパーツを作成します。このクラスはフレーム・ベースのクラスでなければなりません。新しく作成されたパーツの partID を戻します。
DataInfo	dataType dataLength decimalPlaces ここで: dataType は '0'=Numeric または '1'=Character dataLength はデータ長 decimalPlaces は小数点以下の桁数	それぞれブランクで区切られた 3 つの数字ストリングを戻します。適用可能なパーツには、入力フィールド、静的テキスト、およびサブファイル入力フィールドが含まれます。
ExtraColorAreas 以下の注を参照	partId	このパーツがフォアグラウンドおよびバックグラウンド以外のカラー・エリアをサポートする場合には、このパーツがサポートするカラー・エリアのカウントを戻します。
FileName	partId	このパーツに設定されたファイル名を取得します。パーツがファイルをサポートしない場合には、戻り値は空ストリングになります。
GetColor	partId [x] ここで、x は指示されたパーツの colorArea に対応します。	指定されたエリアのカラーを取得します。ブランクで区切られた 4 つの数字を持つストリングを戻します。 useDefault - (0 or 1) redMix - (0 - 255) greenMix - (0 - 255) blueMix - (0 - 255)

表 15. ターゲット: Part (続き)

GetFont	partId [x] ここで、x は指示されたパーツの fontArea に対応します。	パーツのフォントを取得します。フォントがサポートされない場合には、空ストリングを戻します。そうでない場合には、戻されるストリングの最初のパーツはデフォルトのフォントが使用されるかどうかを示す 0 または 1 になり、ストリングの 2 番目のワードはポイント・サイズ、3 番目のワードは適用可能なフォント・スタイルと共に (OR) 以下の数字となります。 1 - 太字体 2 - イタリック体 4 - 下線 8 - 取り消し線 16 - アウトライン 残りのストリングはフォントの書体名です。
GetRect	partId	親に対するパーツの座標 (x y width height)
HasFile	partId	パーツがファイル（たとえば、キャンバス、イメージ、メディアなど）をサポートすることを示すためには 1 を、そうでない場合には 0 を戻します。
IsColorArea 以下の注を参照	partId [x] ここで、x は指示されたパーツの colorArea に対応します。	カラー・エリアがサポートされることを示すには 1 を、そうでない場合には 0 を戻します。
IsFontArea	partId [x] ここで、x は指示されたパーツの fontArea を表します。	フォント・エリアがサポートされることを示すには 1 を、そうでない場合には 0 を戻します。
Label	partId	パーツのラベル（もしあれば）を戻します。
LinkedEvents	partId	このパーツがアクション・リンクを持っているイベントのリストを戻します。
Name	partId	ツリー・ビューおよび設定ノートブックに表示されたパーツの名前を戻します。
OpenDesignWindow	partId [1]	1 に設定すると、指示されたパーツが属する設計ウィンドウをオープンしてフォーカスを設定します。0 に設定すると、設計ウィンドウはクローズされます。

表 15. ターゲット: Part (続き)

OpenPart	partId	パーツの設定ノートブックをオープンするか、またはパーツがフレームの場合にはパーツの対応する設計ウィンドウをオープンします。
OpenSettings	partId	パーツの設定ノートブックをオープンします。
SetColor 以下の注を参照	partId colorArea useDefault redMix greenMix blueMix	指定されたエリアのカラーを設定します。それぞれのパラメーターに使用できる値の詳細については、GetColor を参照してください。
SetCursored	partId	パーツの設計ウィンドウがオープンされるとパーツはアクティブ・パーツになりますが、選択状態は変わりません。設計ウィンドウがオープンされない場合には、何の影響もありません。
SetDataInfo	partId dataType dataLength decimalPlaces ここで: partId はパーツの id dataType は '0'=Numeric または '1'=Character dataLength はデータ長 decimalPlaces は小数点以下の桁数	パーツのデータ・プロパティーを設定します。すでにオープンされているか、または使用中のプロパティー・ノートブックは更新されません。プログラマーは、新しい値がそのパーツにすでに定義されている既存の値に対応するかどうかを確認する必要があります。適用可能なパーツには、入力フィールド、静的テキスト、およびサブファイル入力フィールドが含まれます。
SetFileName	partId newFileName	このパーツのファイル名を設定します。パーツがファイルをサポートしない場合には、何も起こりません。
SetFont	partId fontArea setToDefault pointSize styles faceName	パーツのフォントを設定します。
SetLabel	partId newLabel	パーツのラベルを設定しようとします。指定されたラベルが無効の場合には、エラー・メッセージが表示されます。ラベルが設定されたことを示すには 1 を、そうでない場合には 0 を戻します。

表 15. ターゲット: Part (続き)

SetName	partId newName	パーツの名前を設定しようとします。設定が失敗すると、エラー・メッセージが表示されます。パーツに関連づけられたアクション・リンクがある場合には、リンクを外すかどうかを尋ねるメッセージが表示されます。成功を示すには 1 を、失敗を示すには 0 を戻します。
SetRect	partId x y width height	親に対するパーツの座標 (x y width height) を設定します。
SetSelected	partId [0|1] [0|1]	指定されたパーツを選択／選択解除します。ストリングの最初のパラメーターは turnOn で、2 番目のパラメーターは exclusive です。 turnOn または exclusive を指定しないと、値 ″1″ と見なされます。 Exclusive はパーツを選択したときに他のすべてのパーツを選択解除するかどうかを、また turnOn はパーツの選択状態を変更するかどうかを示します。
SetStyles	partId styles extendedStyles [0|1]	指定されたパーツのスタイルおよび拡張スタイルを設定します。これらの設定では、プロパティー・ノートブックまたは設計ウィンドウはオープンされていても、必ずしも更新されないことに注意してください。このコマンドは、パーツが作成され初期化されるときに使用します。このストリングの終わりの ″0″ は値が 10 進数形式であることを示し、″1″ は 16 進表記が使用されることを示します。
スタイル	partId [0|1]	指定されたパーツのスタイルおよび拡張スタイルを表す、スペースで区切られた2 つの数値を戻します。このストリングの終わりの ″0″ は値が 10 進数形式であることを示し、″1″ は 16 進表記が使用されることを示します。
ズーム	partId [0|1]	ツリー・ビューを拡大して、指示されたパーツまでスクロールします。″1″ を指定すると、ツリー・ビューにもフォーカスが設定されます。

注: パーツは、フォアグラウンド (1) カラー・エリアを持つか、バックグラウンド (0) カラー・エリアを持つか、カラー・エリアを持たないか、あるいは特別のカラー・エリアを持つことになります。ウィンドウ・パーツはカラー・エリアを持ちません。チェック・ボックスはフォアグラウンドとバックグラウンドのカラー・エリアを持ちます。グラフは特別のカラー・エリアを持ちます。したがって、パーツが特別のカラーを持たない場合には、0 と 1 はバックグラウンドとフォアグラウンドのカラーを指示するだけです。

以下は、ソース・ファイルを LPEX でオープンするように要求します。

表 16. ターゲット: Subroutine

コマンド	パラメーター	意味／戻り値
DeleteActionSub	routineName	指定された名前のアクション・サブルーチンを削除します。
DeleteUserSub	routineName	指定された名前のユーザー・サブルーチンを削除します。
UserSubroutine	routineName	サブルーチンが存在しない場合には、指定された名前のユーザー・サブルーチンを作成して、ソース・ファイルに入れます。存在する場合には、ソース・ファイルに入っています。
UserSubroutines	なし	ユーザー・サブルーチンのリストを戻します。

表 17. ターゲット: Grid

コマンド	パラメーター	意味／戻り値
IsOn	なし	グリッドが現在オンになっている場合には 1 を、オフの場合には 0 を戻します。
TurnOn	[0|1]	1 に設定されると、グリッドをオンにします。0 に設定されると、オフにします。(デフォルトはオンです。)

表 18. ターゲット: Lpex

コマンド	パラメーター	意味／戻り値
DoIt	Any_LPEX_command	パラメーターを LPEX の ″DoIt″ に渡します。
IsSourceFileOpen	なし	ソース・ファイルがオープンされていることを示すには 1 を、そうでない場合には 0 を戻します。
OpenSourceFile	なし	ソース・ファイルを LPEX でオープンします。
Query	Any_LPEX_query	パラメーターを LPEX の ″Query″ に渡します。

表 19. ターゲット: Plugin

コマンド	パラメーター	意味／戻り値
AddPlugin	filename	指定されたプラグインを追加しようとします。成功すると ″0″ を戻します。
get	PluginCount	現在インストールされているプラグインの数を戻します。
	Plugin oneBasedIndex	oneBasedIndex 番目のプラグインの完全修飾パスを戻します。n が 1 より小さいかまたはプラグインの数より大きい場合には、ヌル・ストリングが戻されます。
	Plugins	すべてのプラグインの完全修飾パスのリストを戻します。
InvokePlugin	oneBasedIndex ターゲット・コマンド・パラメーター	ターゲット／コマンド・インターフェースを使用してプラグインを呼び出します。

表 20. ターゲット: Registry

コマンド	パラメーター	意味／戻り値
DeleteKey	キー	このコマンドは、レジストリーから指定されたキー（サブキーを含む）を削除します。
Get	key [″defaultValue″]	キーが存在しない場合には戻り値は defaultValue で、存在する場合には戻り値はレジストリー内のキーの値です。入力時には、'defaultValue' ストリングをユーザーが選択するストリングと置き換えます。二重引用符が必要です。
GetRect	key [″x y width height″]	このコマンドは、レジストリーから長方形を検索し、指定されたキーを持つエレメントが見つからない場合には、提供されたデフォルト値が代わりに戻されます。二重引用符が必要です。
Set	キー値	このコマンドは、レジストリーにストリング値を設定するために使用します。戻り値はありません。
SetRect	key ″x y width height″	このコマンドは、正規化された座標を使用して、指定された長方形をレジストリーに保管します。二重引用符が必要です。

レジストリー・コマンドの商法に関するコメント。

プラグインは、固有のものとなる初期サブキーを使用して、他のプラグインのレジストリー項目と干渉しないようにすることを強くお奨めします。

これらのコマンドを使用して作成されたすべてのレジストリー項目は、VARPG のレジストリー項目の共通サブセクションに制限されますが、プラグインをまたいでオーバーラップすることが可能です。

そのようなオーバーラップを防ぐために、プラグインは、以下のように初期サブキーとして .PLG ファイルのパス名にバリエーションを使用することができます。

プラグインのパス名が次の場合:

```
"c:\plugins\My_Plugins\myplug.plg"
```

ウィンドウ位置に格納するためにレジストリー項目を使用すると、その値に使用する適切なキーは次のようになります。

```
"c__plugins_my plugins_myplug.plg\Window Position"
```

（キーのパス部分から大文字小文字の区別が除かれて、コロンと \ マークが下線に変換されていることに注意してください。）キーにはスペースを含めることができるので、指定するキーと値は引用符で囲む必要があります。したがって、ストリング値を設定しようとする場合には、以下を使用することになります。

```
Set( "c__plugins_my_plugins_myplug.plg\Some relevant keyname" "The new value.")
```

組み込み引用符には前に \ マークが付けられます。

```
Set( "c__plugins_my_plugins_myplug.plg\Some relevant keyname" "The new \"quoted\"value.")
```

GUI Designer 独自の構成ウィンドウ（の一部）に適用される他のいくつかのコマンドがあります。(たとえば、パーツ・カタログです。)

適用可能なターゲット:

MainWindow
: これは GUI Designer のメイン・ウィンドウです。

Catalog
: パーツ・カタログ。

DBRefDlg
: 参照フィールドの定義ウィンドウ。

ImportDlg
: 表示装置ファイルのインポート・ウィンドウ。

LPEX エディター・ウィンドウ。

これらは、指示されたウィンドウがオープンされるときに適用されるだけであることに注意してください。

表 21. ターゲット: GUI Designer constituent windows

コマンド	パラメーター	意味／戻り値
GetHandle	なし	指定されたウィンドウの Windows HANDLE を戻します。
GetIWindowPointer	なし	指定されたウィンドウの IWindow ポインターを戻します。

表 21. ターゲット: GUI Designer constituent windows (続き)

MoveSizeTo	X Y Width Height	ウィンドウのサイズおよび位置を設定します。
MoveTo	X Y	ウィンドウを位置 (X、Y) へ移動します。
Position	なし	ウィンドウの位置を ″X,Y″ の形で戻します。
Rect	なし	ウィンドウの長方形を ″X,Y,Width,Height″ の形で戻します。
SetFocus	なし	指示されたウィンドウにフォーカスを設定します。
SetSize	Width Height	ウィンドウのサイズを設定します。
ShowSetFocus	なし	ウィンドウを（まだ表示されていない場合に）表示して、フォーカスを設定します。
サイズ	なし	ウィンドウのサイズを ″X Y″ の形で戻します。
NotifyOnClose	ウィンドウ・ハンドル	GUI Designer のクローズ時に通知されるウィンドウを指定します。

サンプル・プラグインのソース・コード

以下は、上記の項で使用した plg ファイルに対応するプラグインのソース・コードです。

```
*********************************************************************
*                                                                   *
* Program ID . . : WhoAmi                                           *
*                                                                   *
* Description  . : Sample program to illustrate the Vendor plugin   *
*                  interface of VARPG.                              *
*                                                                   *
*                  When invoked from the Vendor menu item on the    *
*                  GUI Designer this program will use the plugin    *
*                  interface to gather information about the        *
*                  current project and display it on a window       *
*                  named MAIN.                                      *
*                                                                   *
*  The following plugin file, WHOAMI.PLG, was specified when adding *
*  this plugin to the GUI designer                                  *
*                                                                   *
*  // WhoAmi.plg plug in file                                       *
*  Vendor_Name:         "Plug-Me-In Inc."                           *
*  Plugin_Name:         "Who Am I?"                                 *
*  Begin_Details:                                                   *
*      Who Am I?                                                    *
*      This plug-in will display information about the current      *
*      project including its directory name and file name.          *
*  End_Details:                                                     *
*  Command_Line:        "d:\myproj\whoami\rt_win32\whoami.exe &1 &2"*
*  Menu_Name:           "Plug-Me-In Inc./Who am I?"                 *
*  Supported_Menus:     1                                           *
*  Accelerator:         F7 Shift                                    *
*  End_of_Definition                                                *
*                                                                   *
*********************************************************************
 *
D Cmd             S            255A
 *
C     *Entry        Plist
C                   Parm                    PlugDLL          64
C                   Parm                    PlugID           64
```

```
      ****************************************************************
      *                                                              *
      * Window . . : Main                                            *
      *                                                              *
      * Part . . . : PB_Cancel                                       *
      *                                                              *
      * Event  . . : Press                                           *
      *                                                              *
      * Description: Terminate the program                           *
      *                                                              *
      ****************************************************************
      *
     C     PB_CANCEL     BEGACT    PRESS         MAIN
      *
     C                   Move      *on           *inlr
      *
     C                   ENDACT
      *
      ****************************************************************
      *                                                              *
      * Window . . : Main                                            *
      *                                                              *
      * Part . . . : Main                                            *
      *                                                              *
      * Event  . . : Create                                          *
      *                                                              *
      * Description: Set up the PLUGDLL and PLUGID values of the     *
      *              *COMPONENT part to establish communication with the  *
      *              GUI builder.                                    *
      *                                                              *
      *              Execute PLUGCMD attributes to collect information    *
      *              about the current project                       *
      *                                                              *
      ****************************************************************
      *
     C     MAIN          BEGACT    CREATE        MAIN
      *
     C     '*Component'  Setatr    PlugDll       'PlugDLL'
     C     '*Component'  Setatr    PlugID        'PlugID'
      *
     C                   Eval      Cmd='Project Get ProjectDir'
     C     '*Component'  Setatr    Cmd           'PlugCmd'
     C     '*Component'  Getatr    'PlugResult'  DirName
      *
     C                   Eval      Cmd='Project Get ProjectFileStub'
     C     '*Component'  Setatr    Cmd           'PlugCmd'
     C     '*Component'  Getatr    'PlugResult'  File
      *
     C                   Eval      Cmd='Project Get ProjectTargetName'
     C     '*Component'  Setatr    Cmd           'PlugCmd'
     C     '*Component'  Getatr    'PlugResult'  TAR



      *
     C                   Eval      Cmd='Project Get ProjectTitle'
     C     '*Component'  Setatr    Cmd           'PlugCmd'
     C     '*Component'  Getatr    'PlugResult'  Title
      *
     C                   Eval      Cmd='Project Get ProjectFileName'
     C     '*Component'  Setatr    Cmd           'PlugCmd'
     C     '*Component'  Getatr    'PlugResult'  Folder
      *
    C                  Write     'Main'
      *
     C                   ENDACT
```

ユーザー・アプリケーションのパッケージング

プラグイン作成の最終ステップは .EXE ファイルのコンパイルです。**「ビルド」**を**「ファイル」**プルダウン・メニューから選択して、プラグインを使用したいプラットフォームを選択します。ファイルはコンパイルするとすぐに使用することができます。プラグインを追加するためには、戻って**ベンダー・プラグインの追加**に関する説明を参照してください。この時点から、プラグインを使用するか、あるいはさらに必要なテストを実行することができます。

VisualAge for C++ を使用してプラグインを作成するときの考慮事項

VisualAge for C++ を使用してプラグインを作成するためのプロセスは VisualAge RPG を使用する場合と同じです。相違点は、VisualAge for C++ を使用してプラグインを作成するときには、*component パーツを直接使用しないことだけです。代わりに、VisualAge for C++ プログラムをプラグインとして使用することができ、IBMExecuteVDECommand() 機能が提供されています。その使用法はサンプル・プラグイン ″TreeSamp″ で示します。

IBMExecuteVDECommand() の正しい呼び出しを作成するために、必要な場合には、x:\...\WDSC\samples\vndplugs\treesamp (x は VisualAge RPG をインストールしているドライブ名に対応）および x:\...\WDSC\samples\vndplugs\plugutil ディレクトリーからコードをカット・アンド・ペーストすることができます。

REXX を使用してプラグインを作成するときの考慮事項

REXX を使用してプラグインを作成するためのプロセスは、VisualAge RPG の例とほとんど同じです。REXX プログラマーは、*component パーツを使用して GUI Designer に直接アクセスすることはしません。プラグインとしての REXX スクリプトの使用を容易にするために、この一連の機能の中に RexxExecuteVDECommand() 機能が含まれています。REXX ファイルでのこの機能の使用法の例は、サンプル・プラグイン ″RexxSamp″ に入っています。

REXXExecuteVDECommand() の正しい呼び出しを作成するために、必要な場合には x:\...\WDSC\samples\vndplugs\rexxsamp (x は VisualAge RPG をインストールしているドライブ名に対応）ディレクトリーからコードをカット・アンド・ペーストすることができます。

第 5 部 アプリケーションの配布

第 27 章 ランタイム・コードおよびアプリケーションのパッケージング

アプリケーションをビルドしてテストした後で、これをパッケージして VisualAge RPG ランタイム・コードをインストールしている他のワークステーションに配布することができます。

この項では、VisualAge RPG ランタイム・コードと VisualAge RPG アプリケーションのパッケージ方法について説明します。

注: アプリケーションが開発サイクル中にアクセスしたサーバー以外の iSeries サーバーを使用する場合には、そのアプリケーションが使用するすべてのサーバー・オブジェクトもパッケージして新しいサーバーに復元しなければなりません。これらは、VisualAge RPG アプリケーション・パッケージング・ユーティリティーではパッケージされません。

開始する前に

アプリケーションに必要なファイルが適切なランタイム・ディレクトリー (Windows の場合は RT_WIN32、Java の場合は RT_JAVA) に保管されていることを確認してください。一部のファイルは、アプリケーションのビルド後に自動的に実行時ディレクトリーに入れられます (たとえば、.MSG、.HLP、.DLL、.BND、.RST、および .EXE ファイル) 。他のファイルはユーザー自身で入れる必要があります (たとえば、.BMP、.GIF、.ICO、.JPG、.MID、および .WAV ファイル) 。

アプリケーションをパッケージする前に追加のファイルを実行時ディレクトリーに入れる場合には、ファイル名が既存のアプリケーション・ファイルの名前と同じでないことを確認してください。

すべてのファイルを実行時ディレクトリーに入れたら、名前とファイル拡張子の最初 2 文字が同じ 2 つのファイルがないことを確認してください。たとえば、FILEA.ABC と FILEA.ABB という 2 つのファイルが RT_WIN32 ディレクトリーにあると、パッケージング・プロセス中に一方が上書きされてしまいます。

ディスケットにパッケージする場合には、パッケージング・プロセスを開始する前に、事前にフォーマットされたディスケットがあると便利です。

VisualAge RPG ランタイム・コードおよびアプリケーションのパッケージング

この項では、VisualAge RPG ランタイム・コード、アプリケーション、または共用コンポーネントをパッケージするために従う必要があるプロセスについて説明します。

注: RST ファイルのリモート・ロケーション名がユーザーの使用するサーバーと同じであることを確認してください。同じでない場合には、「iSeries 情報の定

義」ノートブックの「サーバー」ページで、この項目のリモート・ロケーション欄を変更してください。GUI Designer からこのノートブックにアクセスするには、**「サーバー」**プルダウン・メニューから**「iSeries 情報の定義」**を選択してください。実行時にこれにアクセスするには、「iSeries 情報の定義」アイコンを使用してください。パッケージング時にこれにアクセスするには、**「サーバーの変更」**ボタンを使用してください。

RST ファイルで、ユーザーが使用する正しいプロトコルも指定する必要があります。

SNA の場合には、リモート・ロケーション名はクライアント・アクセスで定義されたルーターの名前です。

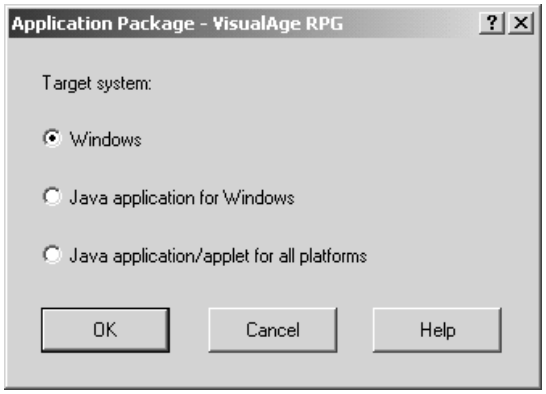
TCP/IP の場合には、リモート・ロケーション名は TCP/IP サーバー・リストで定義された iSeries サーバー・ホスト名です。

パッケージング・ユーティリティーの開始

パッケージング・ユーティリティーを開始するには、以下の方法の 1 つを使用してください。

- プロジェクト・オーガナイザーから**「プロジェクト」>「パッケージ」**を選択します。
- プロジェクト・アイコンのポップアップ・メニューから**パッケージ**・オプションを選択します。
- **「スタート」 > 「プログラム」 > 「IBM WebSphere Development Studio Client for iSeries」 > 「VisualAge RPG」**メニューから、**「アプリケーション・パッケージ・ユーティリティー」**を選択します。

「アプリケーションのパッケージ」ウィンドウが表示されます。



アプリケーションのターゲット・システムを指定します。

- **Windows**

  アプリケーションの Windows バージョンを Windows プラットフォーム用にパッケージします。
- **Java アプリケーション Windows 用**

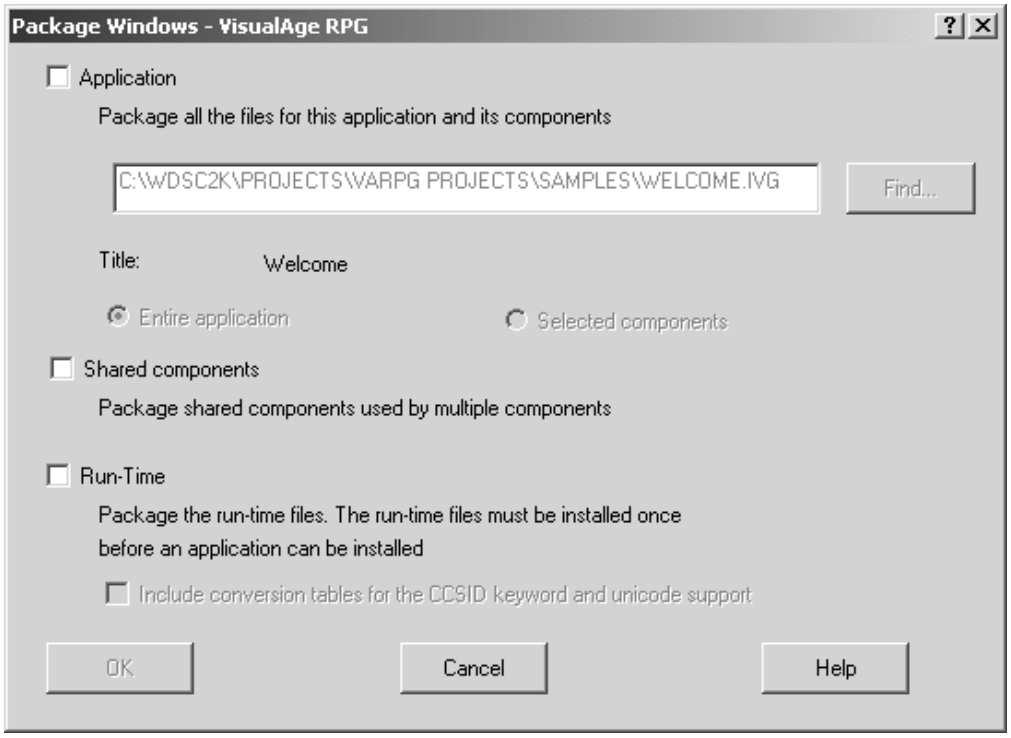
  アプリケーションの Java バージョンを Windows プラットフォーム用にパッケージします。

- **Java アプリケーション / アプレット 全プラットフォーム用**

  アプリケーションの Java バージョンを他のオペレーティング・システム用にパッケージします。

Windows アプリケーションを Windows 用にパッケージ

「Windows」を選択して「OK」を押します。「Windows パッケージ」ダイアログが表示されます。



以下を指定してください。
- パッケージ対象
- アプリケーション・パッケージ情報
- 実行時パッケージ情報

パッケージ対象の指定

パッケージ・ウィンドウで、次の情報を指定してください。

アプリケーション名

完全修飾アプリケーション・プロジェクト名。デフォルト (もしあれば) に上書きするか、または **「検索」**プッシュボタンを使用して「パッケージ用プロジ ェクトの検索」ウィンドウを呼び出します。アプリケーション・プロジェクト名を指定すると、アプリケーションのタイトルが参考用に表示されます。

パッケージ対象

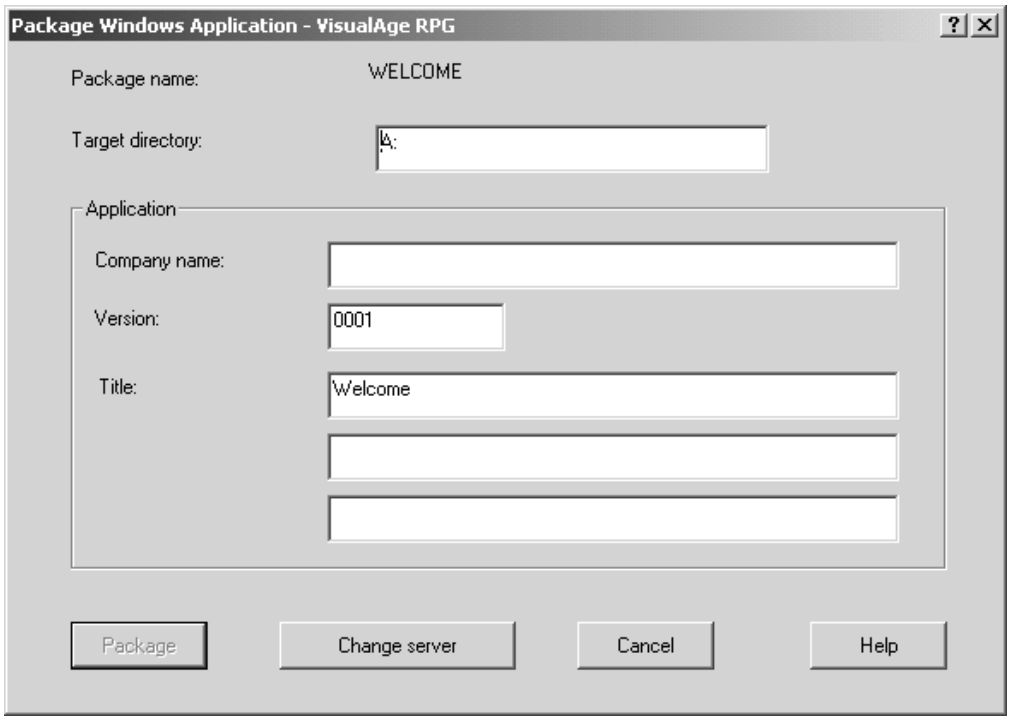
このチェック・ボックスは、アプリケーション、共用コンポーネント、ランタイム・コード、またはこの 3 つ全部のいずれをパッケージするかを指示するために使用します。共用コンポーネントを選択すると、共用コンポーネ

ントのリストが入っているウィンドウがオープンされます。リストのどの共用コンポーネントをパッケージするかを選択できます。

このラジオ・ボタンは、全アプリケーションをパッケージするか、それとも選択したコンポーネントだけをパッケージするかを指示するために使用します。選択したコンポーネントのパッケージを選択すると、アプリケーション内の選択可能なコンポーネントのリストが入っているウィンドウがオープンします。他のアプリケーションで作成された追加の共用コンポーネントをパッケージすることもできます。これらの共用コンポーネントは、次回にコンポーネント別のパッケージを選択すると自動的にリストに追加されます。

アプリケーション・パッケージ情報の指定

「パッケージ」ウィンドウで選択を行ったら「OK」を押します。パッケージ情報ウィンドウが表示されます。



「パッケージ情報」ウィンドウに、次の情報を入力してください。

ターゲット・ディレクトリー
パッケージ用のターゲット・ディレクトリー。ディスケットにパッケージする場合には、ターゲット・ディレクトリーはディスケットのルート・ディレクトリーだけで、サブディレクトリーはありません。ディレクトリーにパッケージする場合には、そのディレクトリーに他のファイルが入っていては**いけません**。

会社名 アプリケーションが登録される会社名。

注: 以前に配布されたアプリケーションの更新バージョンをパッケージする場合には、改訂されたアプリケーションに同じ会社名を使用してください。そうしないと、改訂されたアプリケーションは新規アプリケーションとして処理されます。

バージョン
アプリケーションのバージョン。

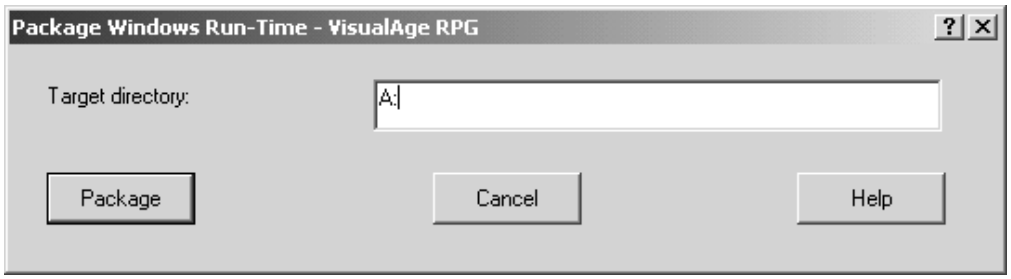
Title アプリケーションのタイトル。

ここで、**「サーバーの変更」**ボタンを使用して、ユーザーのアプリケーションで使用するサーバーのリスト（リモート・ロケーション）を表示することができます。パッケージに新しい名前を使用するように、リストを変更することができます。

パッケージを開始するには、**パッケージ**を選択します。進行状況表示ウィンドウが現れます。各種のディスケットを作成した時にどのラベルを付けるかを指示するメッセージが表示されます。パッケージが完了すると、完了メッセージが表示されます。

ランタイム・パッケージ情報の指定

ランタイム・コードをパッケージするように指示した場合には、「ランタイムのパッケージ」ウィンドウでターゲット・ディレクトリーを指定しなければなりません。:



ディスケットにパッケージする場合には、ターゲット・ディレクトリーはディスケットのルート・ディレクトリーだけで、サブディレクトリーはありません。ディレクトリーにパッケージする場合には、そのディレクトリーに他のファイルが入っていては**いけません**。

パッケージを開始するには、**パッケージ**を選択します。進行状況表示ウィンドウが現れます。完了メッセージに、パッケージングが完了した時点が示されます。

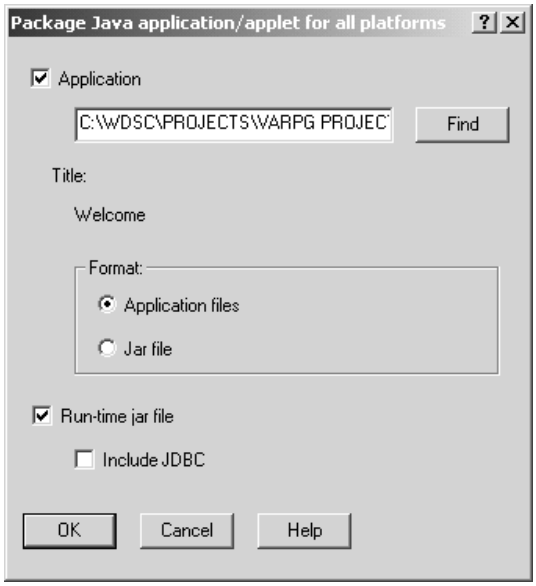
Windows 用 Java アプリケーションのパッケージ

アプリケーションの Java バージョンを Windows プラットフォーム用にパッケージするには、Windows バージョンと同様の一連のダイアログを使用します。

パッケージ対象およびランタイムをパッケージするかどうかを指定します。アプリケーションをパッケージしたい場合には、そのアプリケーションのプロジェクト名を指定して、アプリケーション・チェック・ボックスを選択します。ランタイムをパッケージするためには、ランタイム・チェック・ボックスを選択します。

他のプラットフォーム用の Java アプリケーションのパッケージング

「アプリケーション・パッケージ」ウィンドウで**すべてのプラットフォーム用の Java アプリケーション／アプレット**を選択して「OK」を押します。「Java パッケージ」ウィンドウが表示されます。



以下を指定してください。

- パッケージ対象: アプリケーション、ランタイム、またはその両方。
- パッケージの形式: アプリケーション・ファイル（アプリケーション選択の場合のみ有効）**または** Jar ファイル。

パッケージ対象の指定

「Java パッケージ」ウィンドウで、以下の情報を指定してください。

アプリケーション名

完全修飾アプリケーション・プロジェクト名。デフォルト (もしあれば) に上書きするか、または**「検索」**プッシュボタンを使用して「パッケージング用のプロジェクトの検索」ウィンドウを表示します。アプリケーション・プロジェクト名を指定すると、アプリケーションのタイトルが参考用に表示されます。

パッケージ対象

アプリケーションをパッケージするか、ランタイム jar ファイルをパッケージするかを指示します。

アプリケーション選択の場合には、次の形式の 1 つを選択することができます。

- **アプリケーション・ファイル**

  実行時ディレクトリーに入っているすべてのファイルを含み、それらをターゲット・ディレクトリーに入れます。
- **JAR ファイル**

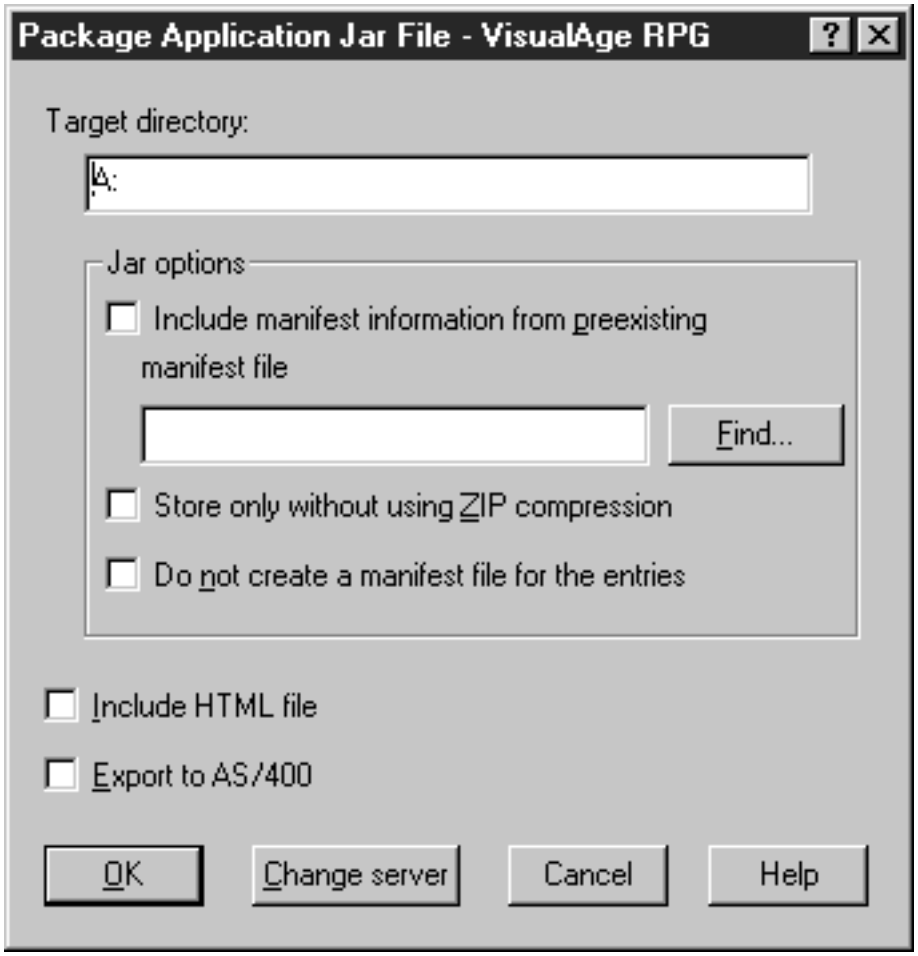
独自の jar ファイルに入っているコンポーネントのすべてのファイルを含みます。(GIF イメージ・ファイルはコピーされるだけで、jar ファイルには含まれません。)

- **JDBC を含む**

  jar の JDBC クラス機能ファイルを含む。

アプリケーション jar ファイルのパッケージング

Jar ファイルとしてパッケージされたアプリケーションを持つように選択すると、次のウィンドウが表示されます。

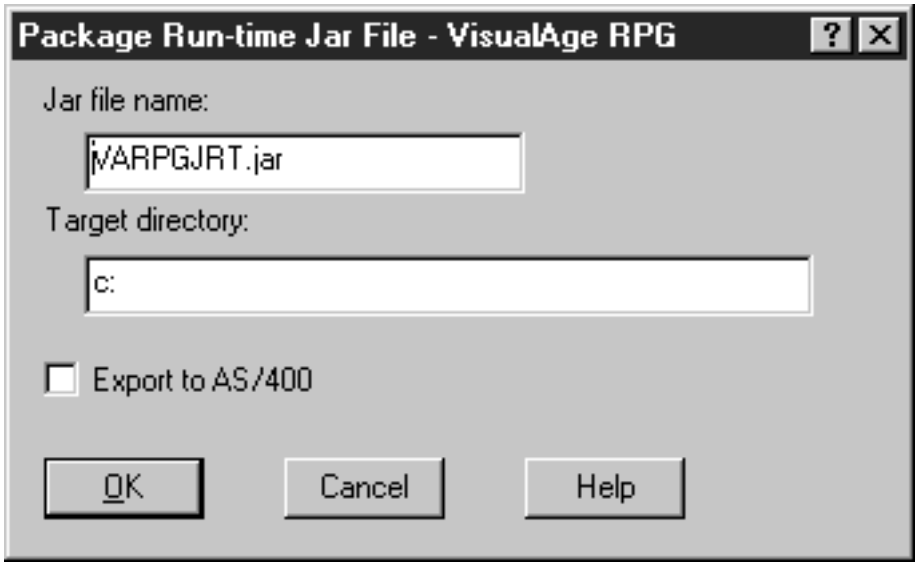


ターゲット・ディレクトリーを指定します。Jar オプションも指定できます。**「HTML ファイルを含む」**を選択すると、アプリケーションのデフォルトの HTML ページがターゲット・ディレクトリーにコピーされます。**「iSeries にエクスポート」**を選択すると、ファイルをサーバーにエクスポートするためのスマート・ガイドが表示されます。

注: アプリケーションに複数のコンポーネントがある場合には、それぞれのコンポーネントに独自の jar ファイルがあります。ここでも GIF イメージ・ファイルはコピーされるだけで、jar ファイルには含まれません。

ランタイムのパッケージング

ランタイム jar ファイルを持つように選択すると、次のウィンドウが表示されます。



Jar ファイル名とターゲット・ディレクトリーを指定します。**「iSeries にエクスポート」**を選択すると、ファイルをサーバーにエクスポートするためのスマート・ガイドが表示されます。

第 28 章 Windows NT/95/98ランタイム・コードおよびアプリケーションのインストール

この項では、InstallShield を使用した Windows NT/95/98 のランタイム・コードおよびアプリケーションのインストールについて説明します。

注: ランタイム・コードは、アプリケーションをインストールする**前に** 必ずインストールされていなければなりません。インストールされているアプリケーションの数に関係なく、ランタイム・コードのコピーは 1 つしかワークステーションにインストールされません。

インストール・コードのインストール

```
setup.exe
```

コマンドを実行して、インストール・ユーティリティーを開始します。

サーバー・ログオン定義、AS/400 情報定義ユーティリティー、および TCP/IP サーバー・リスト定義が、ランタイム・コードでインストールされます。これらのユーティリティーを使用して、実行時に AS/400 リソースの名前と位置を保守および更新します。詳細については、205 ページの『第 8 章 iSeries の接続性』 を参照してください。

組み込み SQL に関する注

アプリケーションに組み込み SQL があって、アプリケーションがビルド時にバインドされなかったデータベースを参照している場合には、そのアプリケーションをアクセス権を持っているデータベースに再バインドする必要があります。

アプリケーションのインストール

パッケージで

```
setup.exe
```

を呼び出してアプリケーションをインストールし、ダイアログ・ボックスに示されているステップに従います。

AS/400 情報定義ユーティリティーは、アプリケーションと一緒に任意にインストールされます。このユーティリティーを使用して、実行時に AS/400 リソースの名前と位置を保守および更新します。詳細については、205 ページの『第 8 章 iSeries の接続性』 を参照してください

ランタイム・コードおよびアプリケーションの保守

ランタイム・コードまたはアプリケーションを更新するには、同じセットアップを使用してください。

ランタイム・コードまたは VisualAge RPG アプリケーションを除去するには、次のようにしてください。

1. タスクバーの Windows NT/95/98 スタート・ポップアップ・メニューから、「設定」を選択し、次に「コントロール・パネル」を選択します。
2. プログラムの追加／除去ユーティリティーを呼び出します。

LAN からのインストール

この項は、Windows NT/95/98 で実行中の Windows アプリケーションに適用されます。

LAN から実行するには、次のようにしてください。

1. ランタイム・コードまたはアプリケーションを LAN サーバーにパッケージします。
2. 同じサーバーのルート・ディレクトリーにランタイム・コードまたはアプリケーションをインストールします。ディレクトリー名は、ランタイム・コードの場合には VRPGRT_LAN、アプリケーションの場合には XXX_LAN です。XXX はアプリケーションの実行可能ファイルの名前です。
3. ステップ 1 のパッケージを使用して、ランタイム・コードをクライアントのワークステーションにインストールします。コンパクト・オプションを選択します。
4. ステップ 1 のパッケージを使用して、アプリケーションをインストールします。コンパクト・オプションを選択します。

LAN からのサイレント・インストール

この項では、LAN サーバーからサイレントにランタイム・コードまたはアプリケーション・コードをインストールする方法について説明します。基本のステップは次のとおりです。

1. パッケージング・ユーティリティーを使用して、ランタイム・コードまたはアプリケーション・コードを LAN サーバーにパッケージします。(説明は、 439 ページの『VisualAge RPG ランタイム・コードおよびアプリケーションのパッケージング』 を参照してください。)
2. **-r** セットアップ・オプションを使用して、ランタイムまたはアプリケーションを LAN サーバーにインストールします。:

   ```
   setup -r
   ```

   r パラメーターによって、システムはインストール・プロセス中にユーザーのキー・ストロークを記録することができます。この情報は、Windows ディレクトリーに作成された setup.iss ファイルに保管されます。このキー・ストロークは、サイレント・インストールに使用されます。
3. **setup.iss** ファイルを、Windows ディレクトリーからランタイムまたはアプリケーションをパッケージした LAN ディレクトリーにコピーします。たとえば、c:\winnt が Windows ディレクトリーの場合には、setup.iss ファイルは、c:\winnt の下で見つけることができます。

   注: アプリケーションを r オプションで LAN ディレクトリーにインストールする前に、ランタイム setup.iss ファイルをコピーしていることを確認してく

ださい。コピーしていない場合には、アプリケーションのインストールからの setup.iss ファイルが、実行時インストールのセットアップで作成された setup.iss ファイルを上書きします。

4. ランタイムまたはアプリケーションをパッケージした LAN ディレクトリー setup.iss ファイルのコピーを変更してください。szDir 項目を、ターゲット・ディレクトリーおよびランタイムまたはアプリケーション・パッケージのインストール先ディレクトリーをポイントするように変更してください。
5. クライアント・ワークステーションから、ランタイムまたはアプリケーションがパッケージされた LAN ディレクトリーに進みます。次のコマンドを実行してください。

   ```
   setup -s
   ```

   ランタイムのインストール後に、オペレーティング・システムをシャットダウンしてオペレーティング・システムを再始動してください。

第 6 部 付録

付録 A. アプリケーション・ファイル

この項では、GUI を作成し、ロジックを書き、アプリケーションをビルドする時に VisualAge RPG が作成するすべてのファイルについて説明します。指示のない限り、これらのファイルを編集したり、名前変更したり、ファイルが作成されたディレクトリーからファイルを除去したりしてはいけません。

注: Java アプリケーションの場合は、これは RT_JAVA です。Windows NT/95/98 アプリケーションの場合には、RT_WIN32 です。

表 22. アプリケーション・ファイル

ファイル名	フォーマット	説明
filename.CLASS	2 進数	実行時ディレクトリーには filename.CLASS ファイルが入っていて、これはプロジェクトが Java 用にコンパイルされる時に作成されます。
filename.DLL	2 進数	実行時ディレクトリーには filename.DLL ファイルが入っていて、これは .VPG ファイルを使用して作成されます。VisualAge RPG DLL はアプリケーションのプログラム・オブジェクトです。コンパイラーは、ユーティリティー DLL とそれに付随する .LIB ファイルも作成することができます。
filename.EVT	ASCII	アプリケーションのソース・ディレクトリーには filename.EVT ファイルが入っていて、これにはコンパイラー・フィードバック・エラーが含まれています。
filename.EXE	2 進数	実行時ディレクトリーには filename.EXE ファイルが入っていて、これには実行時メインラインが含まれています。コンパイラーは自己完結型の EXE も作成することができます。
filename.HLP	2 進数	実行時ディレクトリーには filename.HLP ファイルが含まれており、これは .IPF、.IPM、.VPG、および .TXM ファイルを使用して作成されたコンパイル済みヘルプ・ファイルです。
filename.HTM	ASCII	アプリケーションのソース・ディレクトリーには filename.HTM ファイルが入っていて、これには、コンパイル済みの Java プログラムをアプレットとして立ち上げるための HTML コードが含まれています。

表 22. アプリケーション・ファイル (続き)

ファイル名	フォーマット	説明
filename_applet.HTM	ASCII	アプリケーションのソース・ディレクトリーには filename_applet.HTM ファイルが入っていて、これには、ユーザーが正しいバージョンのランタイムを拡張子としてインストールするように、VARPG Java ランタイムを検査する HTML コードが含まれています。
filename.IPF	ASCII	アプリケーションのソース・ディレクトリーには filename.IPF ファイルが入っていて、これにはオンライン・ヘルプの作成に必要なすべての制御情報が入っています。
filename.IPM	ASCII	アプリケーションのソース・ディレクトリーには filename.IPM ファイルが入っていて、これにはウィンドウとそのパーツのメッセージに対してユーザーが作成する 2 次レベルのすべてのヘルプが含まれています。 • このファイルを名前変更したり、それが作成されたディレクトリーから除去してはいけません。 • このファイルの編集には GUI Designer（メッセージの定義ウィンドウ）を使用してください。このファイルを GUI Designer の外側で編集しなければならない場合には、ユーザーによる変更を、文法やスペルの間違いを訂正するなどの簡単なテキスト編集に制限してください。メッセージ ID を除去または変更したり、メッセージを追加したり、メッセージを削除したりしてはいけません。
filename.JAVA	ASCII	アプリケーションのソース・ディレクトリーには filename.JAVA ファイルが入っていて、これには Java コンパイルの結果として生成された Java ソースが含まれています。
filename.LIB	2 進数	filename.LIB ファイルには、ユーティリティーfilename.DLL のパーツであるエクスポート済みのすべてのプロシージャーが入っています。
filename.LST	ASCII	アプリケーションのソース・ディレクトリーには filename.LST ファイルが入っていて、これにはコンパイル・リストが含まれています。

表 22. アプリケーション・ファイル (続き)

ファイル名	フォーマット	説明
filename.ODF	2 進数または ASCII	アプリケーションのソース・ディレクトリーには filename.ODF ファイルが入っていて、これにはユーザーのアプリケーションのウィンドウとそのパーツに関するすべての情報が含まれています。 • このファイルを名前変更したり、それが作成されたディレクトリーから除去してはいけません。 • このファイルは、GUI Designer を使用しなければ編集できません。
filename.ODX	ASCII	ソース・ディレクトリーには filename.ODX ファイルが入っていて、これは filename.ODF を使用して作成され、実行時に使用されます。
filenameResources.properties	ASCII	アプリケーションの実行時ディレクトリーには filenameResources.properties ファイルが入っていて、これには、ユーザーがウィンドウおよびそのパーツ用に Java 形式で作成するすべてのメッセージが含まれます。
filename.RST	ASCII	ソース・ディレクトリーには、このファイルのマスター・コピーが入っています。filename.RST には、ユーザーが自分のアプリケーション用に定義するすべてのサーバーの別名、ファイル指定変更、データ域指定変更、プログラム指定変更、およびロック・レベル情報が含まれます。その内容は、ビルド時に GUI Designer を使用するか、または実行時に AS/400 情報定義ユーティリティーを使用して変更することができます。
filename.TXC	ASCII	アプリケーションのソース・ディレクトリーには filename.TXC ファイルが入っていて、技術上の説明を保管するために提供されている複数行編集フィールドにユーザーが保存するすべてのプログラミング上の注意事項が含まれます。 • このファイルを名前変更したり、それが作成されたディレクトリーから除去してはいけません。 • その設定値を変更するには、パーツのプロパティーを使用してください。

表 22. アプリケーション・ファイル (続き)

ファイル名	フォーマット	説明
filename.TXM	ASCII	アプリケーションのソース・ディレクトリーには filename.TXM ファイルが入っていて、これにはユーザーがウィンドウとそのパーツ用に作成するすべてのメッセージが含まれます。 • このファイルを名前変更したり、それが作成されたディレクトリーから除去してはいけません。 • このファイルの編集には GUI Designer（メッセージの定義ウィンドウ）を使用してください。このファイルを GUI Designer の外側で編集しなければならない場合には、ユーザーによる変更を、文法やスペルの間違いを訂正するなどの簡単なテキスト編集に制限してください。リソース ID (resid) を除去または変更したり、メッセージを追加したり、あるいはメッセージを削除したりしてはいけません。
filename.VCX	2 進数	アプリケーションの実行時およびソース・ディレクトリーには filename.VCX ファイルが入っていて、これにはアプリケーションで使用される ActiveX パーツの永続性情報が含まれています。
filename.VPF	ASCII	アプリケーションのソース・ディレクトリーには filename.VPF ファイルが入っていて、これには、ユーザーがウィンドウとそのパーツ用に作成するすべてのヘルプ・テキストが含まれます。 • このファイルを名前変更したり、それが作成されたディレクトリーから除去してはいけません。 • このファイルは、GUI Designer のエディターを使用して編集することができます (パーツのポップアップ・メニューを使用するか、またはプロパティー・ノートブックを使用)。
filename.VPG	ASCII	アプリケーションのソース・ディレクトリーには filename.VPG ファイルが入っていて、これにはユーザーが作成するすべての VisualAge RPG アプリケーション・ソース・コードが含まれます。 • このファイルを名前変更したり、それが作成されたディレクトリーから除去してはいけません。 • ソース・コードを編集するには、GUI Designer を使用してください。

付録 B. シン・クライアント・アプリケーションの書き込み

主としてワークステーション・リソースで実行し使用される VisualAge RPG アプリケーションは、**シック・クライアント**・アプリケーションと呼ばれます。**シン・クライアント**・アプリケーションは、全面的に iSeries サーバーを信頼してその処理を実行し、GUI処理をクライアントだけに任せます。

シック・クライアント・アプリケーションは今日の RPG III や RPG IV アプリケーションに見られるものとほとんど同じプログラミング・スタイルをとりますが、iSeries サーバーではなく主としてワークステーションで実行します。アクセスするデータベース・ファイルの指定にはファイル仕様を使用し、サーバー上のデータにアクセスするためには READ や CHAIN などのネイティブな RPG 操作を使用します。iSeries サーバーはデータ・サーバーとして機能し、最小のコンピューティングを実行して VARPG アプリケーションをサポートします。

シック・クライアント・モデルには、対するシン・クライアントと比較していくつかの欠点があります。そのモジュール再利用の機能は非常に限定されていて、変更管理に関連したオーバーヘッド・コストは高くなります。それだけでなく、クライアント・ワークステーションへの処理の移動では、サーバーの処理パワーが使用されます。

アプリケーションのクライアント部分をシンにすることによって、次のような利点が得られます。
- クライアントで実行するコード量は容易に削減できます。
- アプリケーションの再利用性がかなり高くなります。
- 複雑なコードの保守が容易になります。

この項では、シン・クライアント・モデルで可能な 2 つの実装について説明します。両方とも iSeries サーバーとの統合を提供する VARPG の複数の機能を活用するものです。この 2 つの例には、以下を使用する可能性が含まれています。
- 直接のファイル・アクセスを使用しないで容易にデータ構造の外部記述データを定義するための、データ構造の外部記述
- サーバー・プログラムを呼び出してデータを渡す簡単な方法を提供するための、リモート呼び出しインターフェース
- GUI Designer の参照フィールド。ユーザー・インターフェースのサブファイルのフィールドは参照ファイルとして定義され、データベース・フィールドの追加の定義は必要ありません。

VARPG シン・アプリケーション・モデルの実装

シン VARPG アプリケーション・モデルは、いくつかの異なった方法で実装できます。ここでは 2 つの方法について説明します。1 つの方法では iSeries サーバーへのリモート呼び出しを使用し、もう 1 つの方法では iSeries サーバーのデータ待ち行列を使用します。両方の例で、同じユーザー・インターフェースを使用します。

単純なクライアント・アプリケーションは、カスタマー・ファイルからデータを読み取って、一度に 10 レコードずつサブファイルに入れます。以下に、このアプリケーションのユーザー・インターフェースを示します。

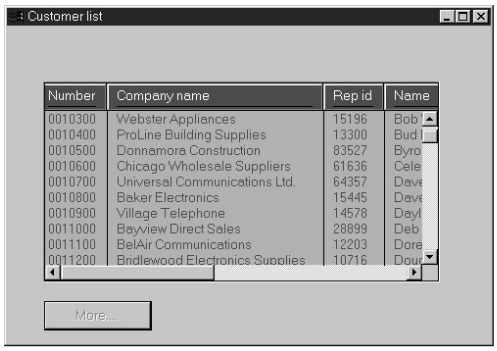


図 112. クライアント GUI インターフェース

このインターフェースは、キャンバス、サブファイル、およびさらに 1 ページの続きページをサブファイルにロードするプッシュボタンから構成されています。この特定の例のサブファイルのレコード数は 10 です。これは、サブファイル・パーツの高さを変えることによって変更することができます。

この例では、次の名前を使用しています。

パーツ 名前

ウィンドウ
WIN1

サブファイル
SUB1

プッシュボタン
PSBMORE

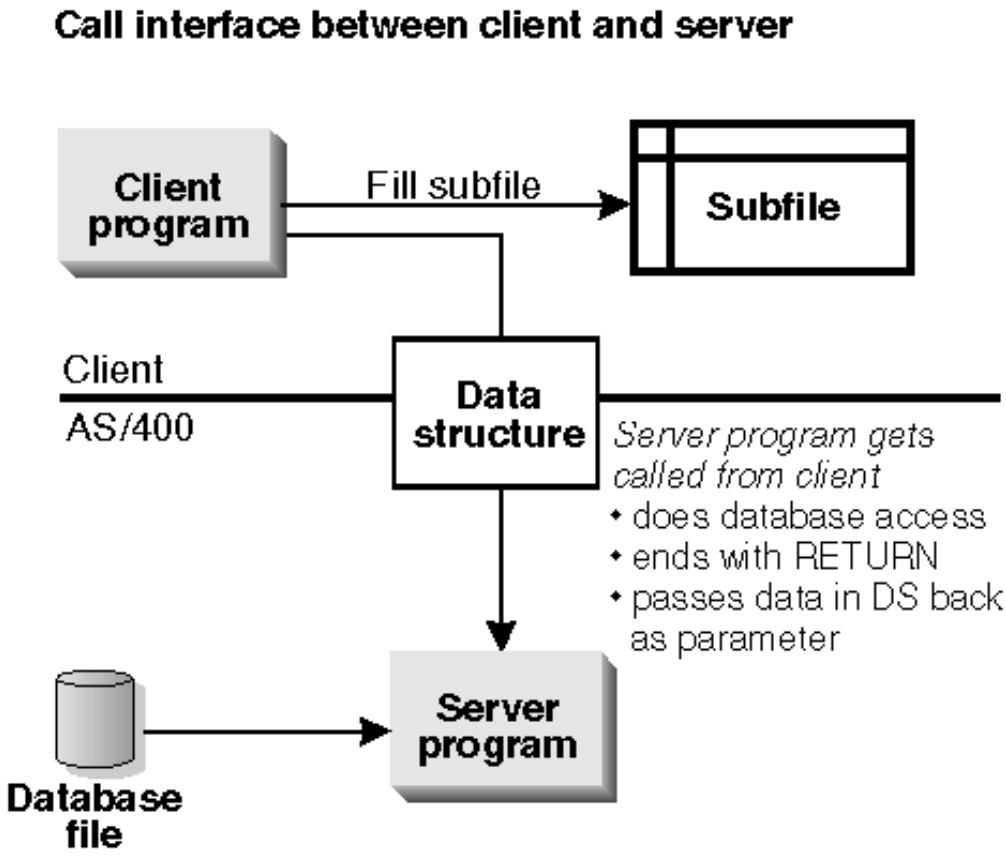
リモート呼び出しを使用するサンプル・アプリケーション

従来の RPG プログラムでは、ユーザー・インターフェース・コードとデータベース・アクセス・ロジックが 1 つのモジュールに混在していました。この構造は、RPG の歴史からの部分と、プログラマー良好なパフォーマンスを達成させるオリジナル・プログラム・モデル (OPM) の使用部分からきています。シン・アプリケー

ション・モデルを実装する 1 つの方法は、ユーザー・インターフェース・ロジックをデータベース・アクセス・ロジックから完全に分割して、それぞれを別個のシステムで実行する方法です。ユーザー・インターフェース・ロジックは Windows クライアントで実行し、データベース・アクセス・ロジックは iSeries サーバーで実行します。

このサンプル・アプリケーションは、iSeries データベースからデータ・レコードを読み取って、このデータを GUI サブファイルに入れるためのサポートの仕組みを示しています。iSeries サーバーのプログラムは、完全なデータベース・アクセス (READおよび WRITE) もサポートできます。これは、異なるそれぞれのアクセス方式に対して 1 つのプログラムを提供したり、必要な操作をパラメーターとして単一のサーバー・プログラムに渡すことによって実装することができます。

次の図に、このサンプルの動作を示します。



クライアント・プログラムは、ユーザー・インターフェースから要求を受け取ります。このプログラムは、データベース・プログラムからレコードを読み取って、このデータをパラメーターを介してクライアントに戻すサーバー・プログラムを呼び出します。サブファイルには戻されたデータが入れられます。

クライアント・プログラム

クライアント側のプログラムの主要部分はユーザー・インターフェースです。これは、すべての VARPG プログラムと同様に作成され、データベース参照フィールドを使用してサーバーの外部データベース記述を使用することができます。データベ

ースに指定された妥当性検査は、VARPG ランタイムによって自動的にクライアントで実行されます。クライアント・プログラムは、サーバー・データ・アクセス・プログラムを呼び出してサーバーにデータを要求し、データ自体はパラメーターを介して渡されます。クライアント・プログラムはファイル仕様を使用しません。その代わりに、外部記述データ構造を通してデータ定義を実行します。このようにして、プログラマーは VARPG プログラムで外部フィールド記述の利点を取得します。

クライアント側のサンプル RPG ソース

VARPG プログラムは、D 仕様と C 仕様から構成されています。D 仕様には次のデータ定義が入っています。

- パラメーターとして使用されるフィールド:
  - *cust*、マルチ・オカレンス構造
  - *custelem*、最大要求レコード数が入っている数値フィールド
  - *eof*、サーバー・プログラムでファイルの終わり標識が ON に設定されたときに渡される名前付き標識
  - *nrecords*、戻されたレコード数が入っている数値フィールド
- 2 つの作業フィールド:
  - *fileend*、ファイルの終わり条件を保存するための名前付き標識
  - *counter*、DO ループのカウンター
- *getrec*、サーバー上で呼び出されるプログラムを定義する固定情報。サーバーとサーバー・プログラムの実際の名前とのリンケージを定義します。プログラム名は UPPERCASE で指定しなければなりません。

```
H
D cust          e ds                    extname(customer)
D                                       occurs(10)
D                                       inz
D eof             s                n    inz
D nrecords        s                2  0
D fileend         s                n    inz
D getrec          c                     linkage(*server)
D                                       const('GETREC')
D counter         s                2  0
D custelem        s                2  0 inz(%elem(cust))
```

C 仕様には、3 つのイベントにリンクされる 1 つのアクション・サブルーチンが入っています。

- **Press** イベント。プッシュボタン *PSBMore* から。
- **Create** イベント。ウィンドウ *Win1* から (PSBmore/press アクション・サブルーチンにリンク)。
- **Pageend** イベント。サブファイル *Subf1* から。

最初のステートメントは、さらにレコードを取り出すためのサーバー・プログラムの呼び出しです。残りのロジックでは、パラメーターを介して渡されたデータを処理して、これをマルチ・オカレンス・データ構造からサブファイルへ移動します。サブファイルに一連のレコードが入れられると、サブファイルの一番大きいレコード番号に SETTOP 属性が適用されて、このレコード・セットがサブファイルの見える部分に移動します。

最後にファイルの終わりに到達すると、**「続き」**プッシュボタンが使用不可に設定されます。「次ページ」キーはまだ機能することに注意してください。このアクション・サブルーチンを起動するイベントは、プッシュボタンが使用不可になっても、使用することができます。

```
 *
C     PSBmore       begact    PRESS         win1
C                   call      GETREC
C                   parm                    cust
C                   parm                    custelem
C                   parm                    eof
C                   parm                    nrecords
C                   eval      counter=1
C                   dow       counter<=nrecords and not fileend
C     counter       occur     cust
C                   write     sub1
C                   eval      counter=counter+1
C                   enddo
C                   eval      %setatr('win1':
C                             'sub1':'settop')=%getatr('win1'§
C                             'sub1':'count')

C                   if        eof
C                   eval      %setatr('win1':
C                             'psbmore':'enabled')=0
C                   eval      fileend=*on
C                   endif
 *
C                   endact
```

お分かりのように、このコードのクライアント側は直裁的で、ワークステーション上の処理を最小にします。

サーバー・プログラム

VARPG クライアント・プログラムはデータベース・アクセス・ロジックを除外するので、この機能はサーバー・プログラムによって提供されます。サーバー・プログラムには、データベース処理を扱うすべての FILE 定義と操作が入っています。データは、クライアントとサーバー・プログラムの間でパラメーターとしてデータ構造を移動することによって交換されます。データ構造には、データ・ファイル・レコード形式のフィールド定義が入っています。この例では、マルチ・オカレンス・データ構造はレコードの集まりにアクセスするために使用されます。オカレンスの数は、渡されるレコードの数（この例では 10) に等しくなります。たとえば、エラー情報などの操作上の情報も、パラメーターによって渡すことができます。サーバー・プログラムは、VARPG クライアント・プログラムの呼び出しによって呼び出され、それぞれの要求の後で終了します。戻り操作コードは、プログラムを終了して呼び出し環境を維持するために使用されます。これは、初期化を必要としないので、以後の呼び出しのパフォーマンスに有利です。また、*NEW では呼び出し環境が破壊されて直ちにストレージが解放されるので、名前つきの活性化グループで実行するためには、プログラムを作成する必要もあります。

サーバー側のサンプル RPG ソース

ファイル仕様は、外部データベース・ファイル・カスタマーを定義します。データ定義仕様は、渡されるパラメーターを定義します。これらは、クライアント側と同じに定義しなければなりません。カウントは、DO ループのカウンターの作業変数

を表します。Custelem にはデータ構造 CUST のエレメントの数が入っていて、これは DO ループの制限として使用されます。

```
 * Program to read a set of records into a data structure
 ******************************************************
Fcustomer  if   e             disk
D cust          e ds                    extname(customer)
D                                       occurs(20)
D eof             s                n
D count           s               2  0
D custelem        s               2  0
```

演算仕様の上部で、PLIST 命令コードは、このプログラムに渡されるパラメーターを定義します。DO ループは、データベース・ファイルから読み取って、データをデータ構造 CUST に入れます。これは、パラメーターとしてクライアント・プログラムに戻されます。他の 2 つのパラメーターはデータベース・アクセスの状況を示すだけです。

- EOF は、標識 99 が READ ステートメントによって設定されると、ON に設定されます。
- カウントには、データ構造 CUST のクライアントに戻されるレコードの数が入っています。

```
C    *entry         plist
C                   parm                    cust
C                   parm                    custelem
C                   parm                    eof
C                   parm                    count
C                   eval      count=1
C     count         occur     cust
C                   read      customer                              9999
C                   dow       count<custelem and not *in99
C                   eval      count=count+1
C     count         occur     cust
C                   read      customer                              9999
C                   enddo
C                   if        *IN99
C                   eval      count=count-1
C                   eval      eof=*on
C                   endif
C                   return
```

サーバー・プログラムのコンパイル時には、活性化グループに *NEW を指定しないようにしてください。*NEW が指定されると（デフォルト）、RETURN の実行時に、このプログラムによって割り振られたストレージが解放されます。このシン・クライアントの例の利点の 1 つは、別のアプリケーションによるサーバー・アプリケーションの最利用可能性です。従来の 5250 アプリケーションでも、データベース・アクセスにサーバー・モジュールを使用できます。サーバー・モジュールの変更はそれを使用するすべてのアプリケーションに反映されるので、このアプローチでは確かにアプリケーションの保守が容易になります。

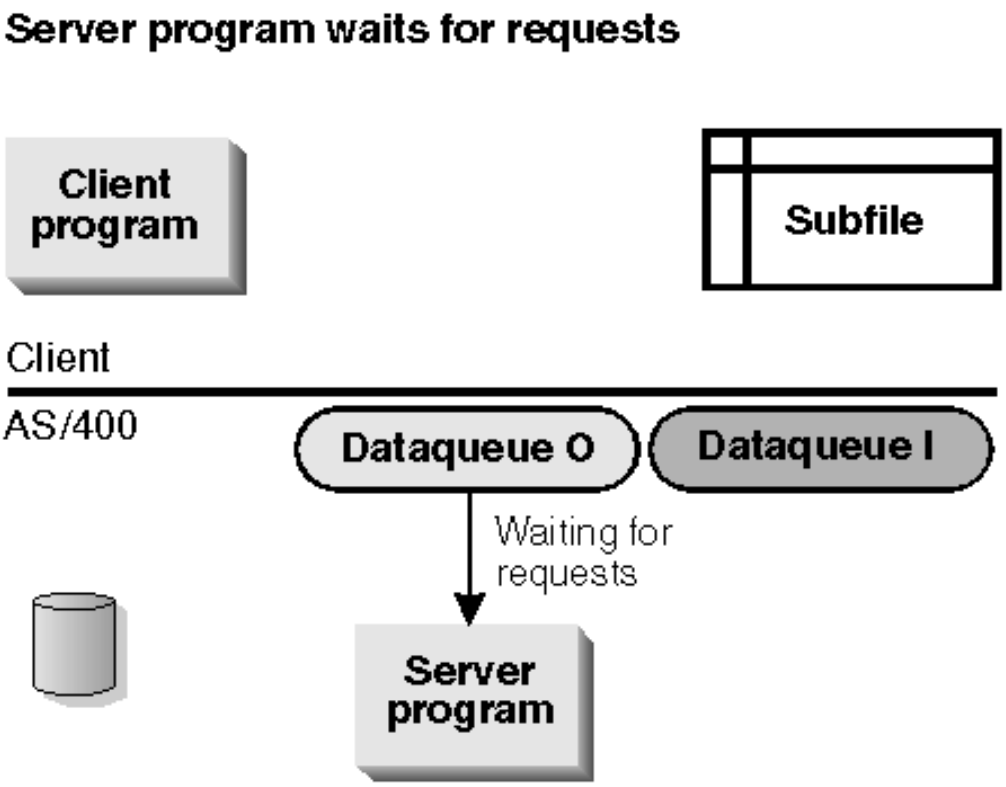
データ待ち行列を使用するサンプル・アプリケーション

iSeries サーバーでは、アプリケーションが相互に非同期的に通信できるように、データ待ち行列の組み込みサポートが提供されます。このサンプル・アプリケーションは、データベースからのデータを VARPG クライアント・プログラムと交換するために、パラメーターの受け渡しでなく、データ待ち行列を活用しています。この

アプリケーションは、クライアントとサーバー・プログラムが使用する、サーバー上の 2 つのデータ待ち行列に基づいています。この例のサーバー・プログラムは、クライアント・プログラムの D 仕様の NOWAIT キーワードを使用して、サーバー上で独立プログラムとして起動されます。

次の図に、この例の動作を示します。

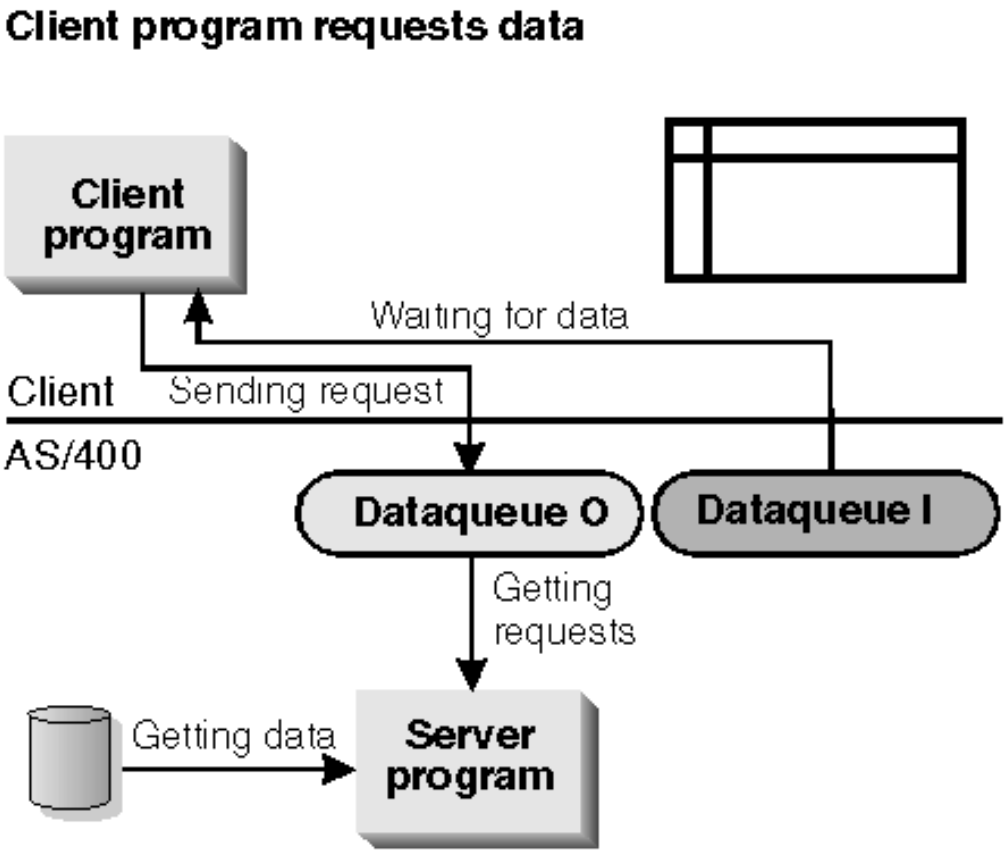
最初に 2 つのデータ待ち行列が作成されて、サーバー・プログラム DATAQ が開始されます。サーバー・プログラムはデータ待ち行列 'O' へのデータ要求を開始して無限の待機状態のままとなります。



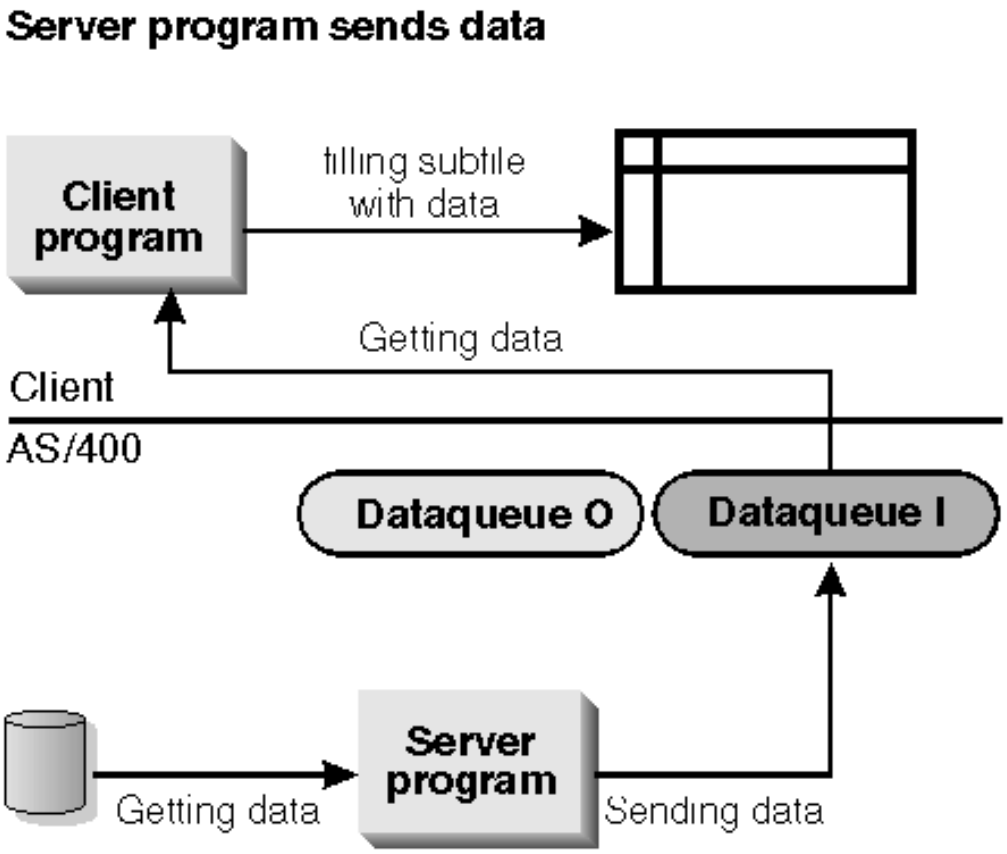
GUI イベントがさらにデータを要求すると、次の状態に入ります。(クライアント・インターフェースについては、 460 ページの図 112 を参照してください。) アクション・サブルーチンを起動する 3 つのイベントは次のとおりです。

- ウィンドウからの Create イベント
- 「続く...」プッシュボタンからの Press イベント
- サブファイルからの Pageend イベント

次に、クライアント・プログラムはデータ待ち行列 'I' のデータを待機します。サーバー・プログラムはデータベース・ファイルにアクセスして、データを取得します。



3 番目の状態では、サーバー・プログラムはデータ待ち行列 'I' に入れます。クライアント・プログラムがアクティブになって、データをサブファイルに入れます。この後でプログラムは初期状態に戻ってもう一度プロセスを開始します。



クライアント・アプリケーション

ユーザー・インターフェースは前のアプリケーションと同じで、基本的には iSeries サーバー・データベースからサブファイルにデータが入れられます。サブファイルへの埋め込みは、ウィンドウの create イベントで開始されて、「続く...」プッシュボタンが押されるかまたは「次ページ」キーを使用して *pageend* イベントが行われると続行されます。これは本質的には前の例と同じです。

データ待ち行列のセットアップはウィンドウ操作作成サブルーチンで実行され、サーバー上のプログラムを呼び出して iSeries サーバーのライブラリーに 2 つのデータ待ち行列を作成します。各クライアントごとに固有のデータ待ち行列を作成するためには、*component パーツのホスト名属性を使用して、クライアントのホスト名および IP アドレスを検索します。戻されたストリングの IP アドレス部分の最後の 5 文字が、データ待ち行列の名前にタグ付けされます。データ待ち行列名の最後の文字 **'I'** または **'O'** は、入力または出力待ち行列に固有の名前を提供します。

サーバー・ジョブは **'O'** データ待ち行列からコマンドを受け取り、クライアント・プログラムはコマンドを **'O'** データ待ち行列に送ります。

データ待ち行列の作成後に、クライアント・プログラムはサーバー・プログラムを呼び出してこの 2 つのデータ待ち行列名を渡します。サーバー・プログラムは、データ待ち行列 **'O'** でクライアント・プログラムからのコマンドを待機します。

クライアント・プログラムは GUI イベントを活動化して、要求をデータ待ち行列 **'O'** に送ります。そして、このデータ待ち行列がサーバー・ジョブによって埋められるまで、データ待ち行列 **'I'** で待機します。

クライアント・プログラムが終了要求を受け取ると、*TERMSR サブルーチンが呼び出されてサーバー・プログラムに終了するように信号を送り、2 つのデータ待ち行列が削除されます。

クライアント・サンプル・ソース

このプログラムは、中でデータ待ち行列環境も管理するので、少し大きくなっています。

データ定義

クライアント・プログラムのデータ定義:

```
D*
D* このプログラムは *component パーツ属性のホスト名変数を使用
D* して、クライアントのホスト名および IP アドレスを保管します。
D entipadd        s            100a
D*
D* データ待ち行列を作成および削除するコマンド・ストリング
D QCMDEXC         s             10    inz('QCMDEXC')
D                                     linkage(*server)
D* コマンド情報を入れる変数
D cmd             s            256    INZ
D cmdlen          s             15p 5 inz(%size(cmd))
D cmd1            s            256    INZ('CRTDTAQ DTAQ(QGPL/')
D cmde            s            256    INZ('DLTDTAQ DTAQ(QGPL/')
D cmd2            s              9    inz(') MAXLEN(')
D*
D* データ待ち行列名の接頭部
D qname1          s              4    inz('CUSQ')
D*
D* 1 つのクライアントで使用される 2 つのデータ待ち行列名が入っている変数
D qnamei          s             10
D qnameo          s             10
D
D* RCVDTAQ および SNDDTAQ プログラムをサーバー・プログラムとして定義します
D QRCVDTAQ        s             10    inz('QRCVDTAQ')
D                                     linkage(*server)
D QSNDDTAQ        s             10    inz('QSNDDTAQ')
D                                     linkage(*server)
D* RPG IV サーバー・プログラム定義
D DATAQ           s             10    inz('DATAQ')
D                                     linkage(*server) nowait
D*
D* カスタマー・データベースが入っているデータ構造
D CustDS        e ds                  extname(customer) occurs(10)
D*
D* 処理情報が入っているデータ構造
D rinfo           ds
D eof                               n
D nrecords                        2  0
D filler                         20
D* ループの限界
D custelem        s               2  0 inz(%elem(CustDS))
D* ファイルの終わりに達した標識
D fileend         s                n
D*
D* データ待ち行列 API のパラメーター
```

```
D msg_sz          s              5  0
D Name_of_Q       s             10
D Name_of_Lb      s             10
D count           s              2  0
D maxlen          s             10  0  inz(%size(custds:*all))
D wait_time       s              5  0
```

ウィンドウ操作作成サブルーチン

データ待ち行列が作成され、サーバー RPG プログラム DATAQ が開始され、このプログラムが NOWAIT キーワードで呼び出されると、クライアント・プログラムは終了するまで待機しません。両方のプログラムはまったく非同期的に機能します。

```
C*
C     WIN1          BEGACT    CREATE        WIN1
C* 固有のデータ待ち行列名をビルドするために、クライアント IP アドレスを入手します。
C* entipadd にはクライアントの完全ホスト名および IP アドレスが入ります。
C*
C                   eval      entipadd =
C                              %GetAtr('*component':
C                             '*component':'hostname')
C* hostname はストリング 'hostname IPAddress' を戻します。
C* 戻されたストリングの IPAddress 部分だけを使用します。
C* 戻されたストリングの 2 番目のパーツをサブストリングとします:
C                   eval      entipadd = %subst(entipadd: %scan(' ':
C                             entipadd)+1)
C* 'I' および 'O' データ待ち行列の名前を作成します。
C* IPAddress の最後の 5 文字を使用し、'I' または 'O' を追加します。
C                   eval      qnameI= qname1 +
C                             %subst(entipadd:%len
C                             (%trim(entipadd))-4:5) + 'I'
C                   eval      qnameO= qname1 +
C                             %subst(entipadd:%len
C                             (%trim(entipadd))-4:
C                             5) + 'O'
C* データ待ち行列を作成するためのコマンド・パラメーターをセットアップします。
C                   eval      cmd=%trim(%trimr(cmd1) +
C                             qnamei + cmd2 +
C                             %editc(%size(
C                             CustDS:*all):'Z') + ')')
C* データ待ち行列を作成します。
C                   call      QCMDEXC                           98
C                   parm                cmd
C                   parm                cmdlen
C*
C                   eval      cmd=*blank
C                   eval      cmd=%trim(%trimr(cmd1) +
C                             qnameo + cmd2 +
C                             %editc(%size(
C                             CustDS:*all):'Z') + ')')
C                   call      QCMDEXC                           98
C                   parm                cmd
C                   parm                cmdlen
C*
C* サーバー上のデータベースにアクセスするため、サーバー・プログラムを呼び出します。
C                   call      DATAQ                             98
C                   parm                qnamei
C                   parm                qnameo
C* 初期化が実行されました。今度はイベント処理を行います。
C*
C                   exsr      callhost
C                   endact
C*
```

さらなるデータの要求を処理するアクション・サブルーチン

要求はデータ待ち行列 'O' に送られます。クライアント・プログラムは、データ待ち行列 'I' でサーバー・プログラム DATAQ からの応答を待機します。データを受け取ると、サブファイルはループに入ります。

```
C*
C* アクション・サブルーチンが呼び出されます:
C*   - PSBMORE プッシュボタンの Press イベント
C*   - サブファイルからの Pageend イベント
C     PSBmore       BEGACT    PRESS         win1
C*
C* サーバーからデータを取得し、それをサブファイルに表示します。
C                   EXSR      callhost
C                   ENDACT
C*

C*
C* ユーザー・サブルーチン 'callhost' がサーバー・データを要求します。
C*
C     callhost      begsr
C* データがある限り、さらにデータを取得します。
C                   if        not fileend
C*
C*
C* データを取り出すために、データ待ち行列 'O' に対して要求を発信します。
C*
C                   eval      nrecords=10
C                   call      QSNDDTAQ
C                   parm                    qnameo
C                   parm      'QGPL   '     NAME_OF_LB       10
C                   parm      23            MSG_SZ            5 0
C                   parm                    rinfo
C* データ待ち行列 'I' からのデータを待ちます。
C* DS rinfo で処理データを待ちます。
C                   eval      wait_time=-1
C                   eval      MSG_sz=23
C                   call      QRCVDTAQ
C                   parm                    qnamei
C                   parm      'QGPL      '  NAME_OF_LB
C                   parm                    MSG_SZ
C                   parm                    rinfo
C                   parm                    WAIT_TIME
C* ここでデータベースからのデータを待ちます。
C                   eval      Msg_sz=%size(custds:*all)
C                   call      QRCVDTAQ
C                   parm                    QNAMEi
C                   parm      'QGPL      '  NAME_OF_LB
C                   parm                    MSG_SZ
C                   parm                    CustDS
C                   parm                    WAIT_TIME
C* サーバーが読み取ったレコードの数だけ、サブファイルを充てんします。
C                   eval      count=1
C                   dow       count<=nrecords and not fileend
C    count          occur     CustDS
C                   write     sub1
C                   eval      count=count+1
     C                   enddo
C                   eval      %setatr('win1':
C                             'sub1':'settop')=
C                             %getatr('win1':
C                             'sub1':'count')
C* ファイルの終わりのシグナルが出された場合は、プッシュボタンを使用不可にします。
C                   if        eof
C                   eval      %setatr('win1':
C                             'psbmore':'enabled')=0
```

```
C                   eval      fileend=*on
C                   endif
C                   endif
C* ユーザー・サブルーチンの終わり
C                   endsr
```

サブルーチンの終了

終了要求がサーバー・プログラム DATAQ に送られて、2 つのデータ待ち行列が削除されます。

```
C* When client app ends, clean up server environment
C*
C     *termsr       begsr
C* Indicate end of program to server program and send data to dataq 'O'
C                   eval      nrecords=0
C                   call      QSNDDTAQ                                98
C                   parm                    qnameo
C                   parm      'QGPL   '     NAME_OF_LB       10
C                   parm      23            MSG_SZ            5 0
C                   parm                    rinfo
C* Delete both data queues
C                   eval      cmd=*blank
C                   eval      cmd=%trim(%trimr(cmde) +
C                             qnamei + ')')
C                   call      QCMDEXC                                 98
C                   parm                    cmd
C                   parm                    cmdlen
C                   eval      cmd=*blank
C                   eval      cmd=%trim(%trimr(cmde) +
C                             qnameo + ')')
C                   call      QCMDEXC                                 98
C                   parm                    cmd
C                   parm                    cmdlen
C* Application ends
C                   endsr
```

サーバー・プログラム

サーバー・プログラムが起動されると、ループに入ってクライアント・プログラムから要求を受け取るまで、データ待ち行列 **'O'** で待機します。この例では、2 つの異なる要求が可能です。プログラムはどちらの要求が送られたかを判別します。さらにデータを読み取るか、あるいは終了するかです。

さらにデータを読み取る要求の場合には、データベースからさらに 10 レコードを読み取って、2 項目をデータ待ち行列 **'I'** に送ります。

最初の項目には、実際にどれだけの情報が読み取られたか、ファイルの終わり状態になったかどうかなどのプロセス情報が入っています。

2 番目の項目には、データベース・ファイルからのデータと一緒にマルチ・オカレンス・データ構造が入っています。

クライアント・プログラムは、データ待ち行列 **'I'** からこれらのレコードを受け取り、それぞれに応じてサブファイルに入れます。

サーバー・プログラムが終了要求の信号を受け取ると、LR 標識がオンに設定されて DO ループが終了します。これでプログラムが終了します。その他の終結処理はクライアント・プログラムによって管理されます。

サーバー・サンプル・ソース

ファイルおよびデータ定義

```
Fcustomer  if   e             disk
D* data structure containing database data to be passed to client
D CustDS        e ds                  extname(customer) occurs(10)
D* data structure to pass control information between client and server
D rinfo           ds
D eof                              n
D count                           2  0
D fill                           20
D* number of occurs in DS for loop limit
D custelem        s               2  0 inz(%elem(CustDS))
D* library name for dataq and data size to be send to dataq and wait time
D Name_of_LB      s              10    inz('QGPL')
D msg_sz          S               5  0
D wait_time       s               5  0
D* name of dataq's passed from client
D qnamei          s              10
D qnameo          s              10
```

メイン行プログラム

DO ループを処理し、要求が到着するまでデータ待ち行列 'O' で待機し、データベースからさらにレコードを読み取り、データをデータ待ち行列 'I' に送り、再び続きの要求を待機します。

```
C* Beginning of mainline
C    *entry         plist
C                   parm                    qnamei
C                   parm                    qnameo
C* DO loop runs forever until client program signals that it
C* terminates
C                   dow       not *inlr
C* Wait for client program to signal that it needs data
C                   eval      wait_time=-1
C                   eval      MSG_sz=23
C                   call      'QRCVDTAQ'
C                   parm                    qnameo
C                   parm                    NAME_OF_LB
C                   parm                    MSG_SZ
C                   parm                    rinfo
C                   parm                    WAIT_TIME
C*
C* Read 10 records from database file
C* count = 0 means client program is terminating
C                   if        count >0
C                   eval      count=1
C    count          occur     CustDS
C                   read      customer                              9999
C                   dow       count<custelem and not *in99
C                   eval      count=count+1
C    count          occur     CustDS
C                   read      customer                              9999
C                   enddo
C* Determine whether there is more data in file
C                   if        *IN99
C                   eval      count=count-1
C                   eval      eof=*on
C                   endif
```

```
C* Send information to the data queue.
C* Send one record with information on how many records are read and
C* whether end-of-file was reached
C*
C                   call      'QSNDDTAQ'                              98
C                   parm                    QnameI
C                   parm                    NAME_OF_LB
C                   parm      23            MSG_SZ
C                   parm                    rinfo
C* Send the data in DS from database file to dataq
C                   eval      msg_sz=%SIZE(custds:*all)
C                   call      'QSNDDTAQ'                              98
C                   parm                    qnamei
C                   parm                    NAME_OF_LB
C                   parm                    MSG_SZ
C                   parm                    CUSTds
C* When client program ends, it sends nrecords 0, then ends this
C* program as well
C                   else
C                   eval      *inlr=*on
C*
C                   end
C                   enddo
C*
C* End of MAINLINE
```

可能なその他のインプリメンテーション

これらの特定の例に加えて、両方のインプリメンテーションのバリエーションや組み合わせが可能です。目標は、クライアントの処理を最小にし、サーバーのパワーを使用してこれらのアプリケーションを実行することにあります。5250 および GUI アプリケーションは同じサーバー・プログラムを使用できるので、サーバー上でのモジュールの再利用が達成されます。

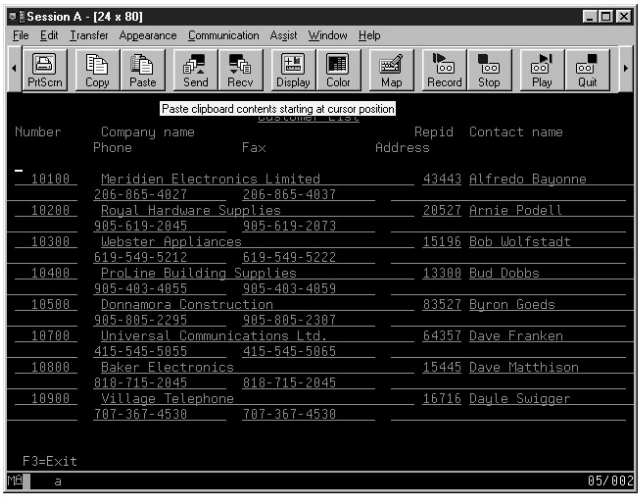
可能な 1 つのインプリメンテーションは、要求をサーバー・プログラムに渡される SQL ステートメントの形で使用するものです。このサーバー・プログラムは SQL ステートメントを出して、受け取ったデータをデータ待ち行列に送ります。データ待ち行列上で待機しているクライアント・プログラムは、戻されたデータを使用してエンド・ユーザーの要求を満たします。この特定のアプリケーションでは、複数のデータ待ち行列でなく、単一のキー付きデータ待ち行列が使用されます。

もう一つのインプリメンテーションでは、ユーザー・インターフェースからのすべてのデータがサーバー・プログラムに渡されて、サーバー上でエラー検査と処理が実行されます。エラー条件があれば、クライアントに戻されます。このアプローチでは、5250 アプリケーションと GUI アプリケーションとの間で高度なビジネス・ロジックの再利用が可能となり、よりシンなクライアントが提供されます。

再利用可能サーバー・プログラムの例

5250 サブファイル・アプリケーションの次の RPG IV コードは、同じサーバー・プログラムを使用しますが、5250 インターフェースをドライブします。この例は、GUI および 5250 アプリケーションでサーバー・プログラムを再利用する可能性を示しています。5250 サーバー・プログラムに入っているビジネス・ロジックは、シン・クライアント GUI プログラムで容易に使用することができます。次の図に、こ

の章の初めの GUI サンプルと同じデータベース情報を表示している 5250 画面のサンプルを示します。



各レコードがサブファイルの 2 行を占めるので、このサブファイルには 8 レコードしか入っていません。このアプリケーションのプログラム・ソースは次のとおりです。

```
H* Program to list customer records
F* Workstn file containing DSPF
Fgetrecs   cf   e             workstn  sfile(sub1:recnum)
D* Data structure to pass data from server program to the subfile
Dcust           e ds                    extname(customer) occurs(8)
D                                       inz
Deof              s              1    inz(*off)
Dnrecords         s              2  0
Dfileend          s              1    inz(*off)
Dcount            s              2  0
Dcustelem         s              2  0 inz(%elem(cust))
Drecnum           s              5  0 inz(1)
 * Main program to invoke subfile sub routine and end the program
 * In91 indicates that database file has not reached the end
C                   EVAL      *IN91=*ON
C                   exsr      more
C                   eval      *inlr=*on
c
C     more          BEGsr
 * LOOP to fetch more data and display in subfile
C                   dow       not *in03
 * Call server program to get data first time and when page down
 * is used.
C                   if        *in91
C                   call      'GETREC'
C                   parm                    cust
C                   parm                    custelem
```

```
C                   parm                    eof
C                   parm                    nrecords
C                   eval      count=1
 * recnum1 is sbfrcdnbr in subfile control record, to position top record
 * to be shown on screen
C                   EVAL      RECNUM1=RECNUM
 * Loop to fill subfile with new records
C                   dow       count<=nrecords
C     count         occur     cust
C                   write     sub1
C                   eval      count=count+1
C                   eval      recnum=recnum+1
C                   enddo
 * After the set of records is added to subfile, display it, plus header
 * and footer record formats
C                   write     record1
C                   write     footer
C                   exfmt     sub1ctl
 * Handle none pagedown keys
C                   else
C                   read      sub1ctl                                99
C                   endif
C                   if        eof=*on
 * IN90 enables PAGEDOWN key for additional records to be read.
 * At file end it gets disabled.
C                   eval      *in90=*on
C                   eval      recnum=recnum1
C                   endif
C                   enddo
 * Leave the LOOP when Exit requested
C                   ENDSR
```

付録 C. 非 GUI プログラムを MS-DOS から作成してコンパイルする

VARPG GUI Designer 内で、あるいは MS-DOS コマンド・プロンプトでコマンドを出して、スタンドアロン VARPG アプリケーションを作成することができます。このセクションでは、コマンド・プロンプトで使用できるコマンドについて説明します。 GUI Designer を使用するには、395 ページの『第 22 章 非 GUI VisualAge RPG プログラムの作成』 を参照してください。

エディターを始動して、MS-DOS コマンド・プロンプトでソースを作成するためには、次のように入力します:

```
codeedit filename.VPG
```

エディターのトークン化およびソースの構文検査を利用できるように、ファイル名に .VPG 拡張子を組み込むようにしてください。

FVDFNFE コンパイラーを実行するには、MS-DOS コマンド・プロンプトからコンパイラー・コマンドを出します。コマンドの構文は次の通りです:

```
fvdfnfe filename.VPG [/compiler_option1 ... /compiler_optionn]
```

注: ユーザーはソース・ファイル名に .VPG 拡張子を組み込まなければなりません。最初から組み込まれているものとは見なされません。

コンパイラー・オプション (大文字または小文字で入力可能) はオプショナルです。それらは次の通りです:

オプション
　　説明

/BL 名
　　リンク・ライブラリー名。複数ある場合には、二重引用符で囲んでください。

/D　　デバッグ情報を生成します

/GL 1-99
　　生成重大度レベル

/L　　出力リストを生成します

/LI　　字下げ

/LX　　相互参照 (XREF) リストを生成します

/LV　　可視の相互参照 (XREF) リストを生成します

/LD　　リスト中の DDS を展開します

/LC　　リスト中の /COPY を展開します

/LE　　外部参照を表示します

/LM2　　リスト中に 2 次レベル・メッセージを表示します

/LS リスト中の除外される行を表示します

/LP 10-99
1 ページ当りの行数 (リスト)

/HCU ホスト・キャッシュが使用可能になります

/HCR ホスト・キャッシュのリフレッシュ

/RF 数字の修正

/RN ヌル使用可能

/RNU ユーザー制御のもとでヌルが使用できます

/RT 数字の切り捨て

/TI デバッグ情報を生成します (/D と同じ)

/SB 名
SQL バインド・ファイル名

/SF XX
日付 / 時刻欄の SQL 形式

/SI XX
(RR RS CS UR) SQL 分離レベル

/SN 名
SQL データベース名

/SP 名
SQL パッケージ・ファイル名

/SR SQL レコード・ブロック

/SU データベース・ユーザー ID

/SUP データベース・パスワード

/SVC 可変文字を変換します

/SVG 可変グラフィックを変換します

AS/400 システムへのアクセス

プログラムが AS/400 システム上のデータにアクセスしたり、あるいは AS/400 プログラムを呼び出す場合には、**リモート・サーバー・テーブル** (RST) ファイルを作成しなければなりません。RST ファイルは、使用するシステムの情報およびファイルの場所を判別するために、コンパイラーおよび VARPG ランタイムによって読み取られる ASCII ファイルです。RST ファイルは、**「AS/400 情報の定義」**ダイアログの使用時に GUI Designer によって作成されます。コンパイル時には、RST ファイルはプログラム・ソースと同じディレクトリーの中になければなりません。実行時には、RST ファイルはプログラム実行可能ファイルと同じディレクトリーの中になければなりません。

以下は RST ファイルの例です:

```
DEFINE_SERVER  SERVER_ALIAS_NAME(MYAS400)
               REMOTE_LOCATION_NAME(TORAS40Z)
               NETWORK_PROTOCOL(*TCP)
```

```
                TEXT(Development - RST)

 DEFINE_FILE  FILE_ALIAS_NAME(CUSTOMER)
              REMOTE_FILE_NAME(PRODUCT/CUSTMAST)
              SERVER_ALIAS_NAME(MYAS400)
              TEXT()
```

DEFINE_FILE ステートメントは、'F' 仕様で定義されたファイルが AS/400 システム上のどこにあるかを示します。この例では、'F' 仕様の CUSTOMER という名前のファイルは、実際にはライブラリー PRODUCT の中のファイル CUSTMAST を指します。このファイルがライブラリー・リストにある場合には、DEFINE_FILE ステートメントを省略することができます。その場合には、ファイルを見つけるためにライブラリー・リスト *LIBL が検索されます。

TEXT キーワードはコメント用に使用され、ブランクとすることができます。

付録 D. Secure Sockets Layer (SSL) セットアップ

Secure Sockets Layer (SSL) は、クライアントと iSeries 400 サーバー・セッションとの間で交換されるデータを暗号化し、サーバー認証を実行することによって、セキュア接続を提供します。SSL は、OS/400 を実行する SSL 可能な iSeries 400 サーバー (バージョン 4 リリース 4 以降) でのみ使用できます。SSL 接続を使用すると、暗号化なしの接続を使用する場合と比較して、製品のパフォーマンスが著しく低下します。パフォーマンスの低下よりも、転送されるデータの重要度が優先される場合にのみ、SSL を使用することをお勧めします。

この項では、iSeries 400 サーバーを Secure Sockets Layer サポート用に構成する場合の指示事項についてを述べます。

注: VARPG ランタイムに対して修正を適用する場合には、次の手順を繰り返さなければなりません。SSL が iSeries 400 サーバーで正しく動作するためには、QSECOFR ユーザー・プロファイル・パスワードの有効期限が切れていてはいけません。

SSL についての考慮事項

- セットアップには、SSL に精通していることが必要です。システム証明書の生成などのように、SSL 関連タスクを実行するためには、iSeries 400 サーバーでのディジタル証明書管理機能 (DCM) プログラムの使用法を理解している必要があります。 DCM プログラムについては、そのオンライン・ヘルプを呼び出すか、あるいは URL http://www.ibm.com/eserver/iseries/infocenter の Information Center に進んでください。
- 輸出入の規則を必ず守るようにしてください。IBM iSeries クライアント暗号化製品は、米国およびカナダ使用では輸出禁止の 128 ビットを使用する SSL バージョン 3.0 暗号化サポートを提供し、国際使用では輸出可能な 56 ビットを提供します。国境を越えてクライアント暗号化製品をダウンロードできるカスタマー構成では、カスタマーは、輸出禁止の製品が米国およびカナダ以外で使用されることがないようにする責任があります。輸出禁止および輸出可能な両方のクライアント暗号化製品を組み合わせて使用することによって、適切なクライアント暗号化製品を別の Web サイトでダウンロードすることができます。

前提条件

SSL をセットアップする前に、次の前提条件が満たされていなければなりません。

- ソフトウェア:
  - 暗号アクセス・プロバイダーのライセンス・プログラム (5769-AC1、5769-AC2、または 5769-AC3)
  - IBM iSeries クライアント暗号化プログラム(5722-CE2 または 5722-CE3) ここで、5722-CE2 (56 ビット) は米国およびカナダ以外の国での使用、および 5722-CE3 (128-bit) は米国およびカナダのみでの使用です。
- ハードウェア:
  - IBM HTTP Server for iSeries (5722-DG1)

– 基本オペレーティング・システム・オプション 34 ディジタル証明書管理機能 (DCM)
- 権限:

  ユーザーが SSL ファイルにアクセスするための適切な権限を提供します。権限を変更するには、以下のステップに従ってください。
  1. 次のコマンドを入力します:

     ```
     wrklnk '/QIBM/ProdData/HTTP/Public/jt400/*
     ```
  2. 正しいディレクトリー (SSL56、または SSL128) でオプション **9** を選択します。
  3. ディレクトリーに対する ***RX** 権限をユーザーに与えます。

     注: 個々のユーザーまたはユーザーのグループを認可することができます。

iSeries 400 サーバー用の SSL セットアップ

iSeries 400 サーバーを SSL 用にセットアップするには、下記の指示に従ってください。

DCM の開始:
1. 次のコマンドを使用して HTTP サーバーを開始します:

   ```
   STRTCPSVR SERVER(*HTTP) HTTPSVR (*ADMIN)
   ```
2. サーバーの URL アドレスおよびポート番号を入力することによって、iSeries 400管理サーバーにアクセスします。たとえば、以下のようになります。

   ```
   http://your.server.name:2001/
   ```

   適切なセキュリティー担当者権限に加えて、 *secadm および *allobj 権限が必要です。
3. iSeries 400 ユーザー ID およびパスワードを入力します。
4. 「AS/400 タスク」ページから、DCM リンクを選択して DCM にアクセスします。

ディジタル証明書管理プログラムから、iSeries 400 サーバーのシステム証明書を獲得します。

システム証明書を獲得する方法については、**?**アイコンをクリックしてください。信頼のある認証局 (CA) からシステム証明書を取得するか、あるいは自分自身のものをビルドすることができます。
- 信頼のある CA からシステム証明書を取得する場合:

  注: アプリケーションがインターネット・アプリケーションであって、ランタイムを配布する必要がある場合は、これを使用すべきです。

  CA を選択します。以下の会社の 1 つから証明書を入手することができます。
  – VeriSign, Inc.
  – Integration Financial Network
  – Thawte Consulting
  – RSA Data Security

DCM から信頼のある CA に対する実行依頼のための要求データを取得します。実行する正確なステップについては、DCM ヘルプを参照してください。CA はユーザーからの要求フォームを処理して、ユーザーに証明書を提供します。それをシステムにインストールするには、**システム証明書の受信**オプションを使用します。

- 自分自身のシステム証明書を取得する場合:

  注: 作成されたランタイムはユーザー固有の iSeries 400 サーバーでしか使用できないので、これはイントラネットの場合にのみ使用してください。

  1. iSeries 400 サーバーに自分の CA を作成します。
  2. 自分自身の CA からシステム証明書を生成します。

  CA の作成については、DCM ページで**認証局**をクリックしてください。

システム証明書を以下のサーバー・アプリケーションに適用します。

```
QIBM_OS400_QZBS_SVR_CENTRAL
QIBM_OS400_QZBS_SVR_DATABASE
QIBM_OS400_QZBS_SVR_DTAQ
QIBM_OS400_QZBS_SVR_NETPRT
QIBM_OS400_QZBS_SVR_RMTCMD
QIBM_OS400_QZBS_SVR_SIGNON
QIBM_OS400_QZBS_SVR_FILE
QIBM_OS400_QRW_SVR_DDM_DRDA
```

ワークステーション用の SSL セットアップ

ワークステーション用の SSL をセットアップするには、以下のステップに従ってください。

iSeries 400 サーバーから、適切なバージョンの SSL クライアント暗号化ソフトウェアをワークステーションにダウンロードします。

製品	ダウンロードするファイル	ダウンロード先の場所
5722-CE2	sslightx.zip	/QIBM/ProdData/HTTP/Public/jt400/SSL56
5722-CE3	sslightu.zip	/QIBM/ProdData/HTTP/Public/jt400/SSL128

自分自身の証明書をビルドする場合には、以下のステップを実行して CA 証明書をダウンロードしてください。

1. SSL Client Encryption をダウンロードしたディレクトリーから SSLTools.jarをダウンロードします。
2. SSLTools.jar および sslightu.zip を CLASSPATH ステートメントに追加します。
3. 一時的なディレクトリーを作成します。たとえば、c:\tempkey など。
4. c:\tempkey の下に次のサブディレクトリーを作成します:

   ```
   com\ibm\as400\access
   ```

5. tempkey ディレクトリーから、次のコマンドを 1 行で実行します:

   ```
   java com.ibm.sslight.nlstools.keyrng com.ibm.as400.access.KeyRing
   connect <systemname>:<port>
   ```

   ここで <systemname> は iSeries 400 サーバーの名前で、<port> はポート番号です。

   注: サーバー・ポートは、アクセスできる任意のホスト・サーバーとすることができます。たとえば、iSeries 400 サーバーにおけるセキュア・サインオン・サーバーのデフォルトのポートである 9476 を使用することができます。パスワードの入力プロンプトが出されたら、toolbox と入力します。これが唯一の有効なパスワードです。SSLツールが iSeries 400 サーバーに接続し、検出した証明書をリストします。
6. CA 証明書番号を入力します。サイト証明書ではなく、必ず CA 証明書を使用してください。証明書を com.ibm.as400.access.KeyRing.class に追加中であることを示すメッセージが表示されます。
7. ファイル SSLTools.jar を削除します。

KeyRing.class ファイルがディレクトリーに作成されました。

以下の指示に従って、カスタマイズした varpg.jar ファイルを作成してください。

1. 一時的なディレクトリーを作成します。たとえば、c:\tempjar など。
2. ファイル sslightu.zip を c:\tempjar にコピーします。

3. 次のコマンドを実行します:

   ```
   jar xvf sslightu.zip
   ```

   c:\tempjar に meta-inf および com\ibm\sslight サブディレクトリーが作成されていることが分かります。
4. 自分自身の証明書をビルドする場合:
   a. サブディレクトリー c:\tempjar\com\ibm の下にサブディレクトリー \as400\access を追加します。
   b. c:\tempkey\com\ibm\as400\access サブディレクトリーの中の c:\tempjar\com\ibm\as400\access サブディレクトリーに、KeyRing.class ファイルをコピーします。
5. varpg.jar ファイルを c:\tempjar にコピーします。
6. c:\tempjar から次のコマンドを実行します:

   ```
   jar uvf varpg.jar -C ./ com
   ```

   注: C は大文字です。
   このコマンドを実行すると varpg.jar ファイルが更新され、SSL 可能の VARPG アプリケーションで使用できるようになります。これは通常の SSL 不可アプリケーションでも機能します。

アプリケーションを SSL 使用可能にします:
1. VARPG インストール・ディレクトリー \WDSC\JAVA の中の varpg.jar を名前変更します。
2. **ビルド・オプション**・ダイアログで、**/SSL** ユーザー定義オプションを使用してアプリケーションをコンパイルします。
3. カスタマイズしたファイル varpg.jar を使用してアプリケーションを実行します。

用語集

この用語集には、次のものから引用した用語とその定義が含まれています。

- The *American National Dictionary for Information Systems* ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). 米国規格協会 American National Standards Institute, 1430 Broadway, New York, New York, 10018 から購入することができます。これらの定義は定義の後のシンボル (A) によって明示されています。
- The *Information Technology Vocabulary* developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Committee (ISO/IEC JTC1/SC1). この用語集の出版済み部分の定義は、定義の後のシンボル (I) によって示されています。国際標準草案、委員会草案、および ISO/IEC JTC1/SC1 により開発中の作業文書から引用した定義は、定義の後のシンボル (T) によって示され、これは SC1 の関係国際本部間でいまだ最終合意に至っていないことを示します。
- *IBM Dictionary of Computing* , New York: McGraw-Hill, 1994.
- *Object-Oriented Interface Design IBM Common User Interface Guidelines*, SC34-4399-00, Carmel, IN: Que Corporation, 1992.

［ア行］

アイコン. オブジェクトのグラフィック表現であり、イメージ、イメージの背景、およびラベルで構成されます。

アイコン・ビュー. コンテナーに含まれている各オブジェクトがアイコンとして表示される標準的な目次ビュー。

アクション. (1) アクション・サブルーチンの同義語。 (2) プロジェクトのパーツを操作するために使用したり、あるいはビルドにかかわるために使用する実行可能なプログラムまたはコマンド・ファイル。

アクション・サブルーチン. 特定のイベントに応答するために作成するロジック。

アクティブ・ウィンドウ. 現在ユーザーが対話しているウィンドウ。 これがキーボード入力を受け取るウィンドウです。

アニメーション制御パーツ. Windows で AVI という拡張子を持つビデオ・ファイルの再生、または Java アプリケーションでアニメーション GIF シーケンスの再生を可能にするパーツ。

アプリケーション. コンピューター上で特定のユーザー・タスクを実行するために使用するソフトウェア・コンポーネントの集合。

アプリケーション・プログラミング・インターフェース (API). オペレーティング・システムによって提供される機能インターフェースまたは個別に注文するライセンス・プログラムであり、高水準言語で書かれたアプリケーション・プログラムがオペレーティング・システムまたはライセンス・プログラムの特定のデータまたは機能を使用できるようにするのもの。

アプレット. Java で書かれ、Java 互換ブラウザーまたは appletviewer の内部で実行されるプログラム。

アンカー. 他のパーツを調整、サイズ決め、およびスペーシングする場合の参照点として使用する任意のパーツ。

移行. (1) 変更された操作環境に移動することであり、通常は新しいリリースまたはバージョンのシステムに移動します。 (2) ある階層のストレージから別の階層のストレージにデータを移動すること。

イベント. パーツの状態に対する変更の結果として生成されるシグナル。たとえば、ボタンを押すと *Press* イベントが生成されます。

イメージ・パーツ. ウィンドウ上で、BMP または ICO ファイルからピクチャーを表示するのに使用するパーツ。

インポート. AS/400 表示装置ファイル・オブジェクトを該当する VARPG パーツに変換する機能。エクスポートと対比。

ウィンドウ・パーツ. 可視境界を持つ領域で、オブジェクトのビューを表したり、あるいはこれを使用してユーザーはコンピューター・システムでダイアログを管理します。パーツ・パレットまたはパーツ・カタログ内のウィンドウ・パーツを指してクリックし、それをプロジェクト・ウィンドウ上に置くことができます。

ウェーブ・ファイル. ウェーブ形式装置のオーディオ・サウンド用に使用するファイル。

エクスポート. アプリケーションの外側で使用するために内部ファイルを何らかの標準ファイル形式に変換する機能。 *インポート*と対比。

エラー・ログ. エラー・ログにエラーのトラックを保存します。エディターを使用すると、ソース内でエラーが起こった場所を表示できます。

オブジェクト. (1) それ自身について記述し、場合によってはデータについて記述する一組のプロパティーから成る名前の付いた記憶スペース。 オブジェクトはストレージの中に存在し、そのスペースを占有するもので、それについて操作を実行することができます。 オブジェクトの例としては、プログラム、ファイル、ライブラリー、およびフォルダーなどがあります。 (2) ユーザー・インターフェースのビジュアルなコンポーネントであり、ユーザーはタスクを実行するためにこれで作業することができます。 オブジェクトはテキストまたはアイコンとして表示されることがあります。

オブジェクト指向プログラミング. 他のオブジェクトと対話することのできるオブジェクトのデータおよび操作について記述するための階層的に編成されたクラスとして、プログラムを構造化するためのメソッド。(T)

オブジェクト-アクション・パラダイム. 対話の 1 つのパターンであり、このパターンでユーザーはオブジェクトを選択し、そのオブジェクトに適用するアクションを選択します。

オブジェクト・プログラム. 実行に適したターゲット・プログラム。 オブジェクト・プログラムは、リンクを必要とする場合と、そうでない場合があります。(T)

オペレーティング・システム. コンピューター・システムの全体の操作を制御するシステム・プログラムの集合。

［カ行］

カーソル. ユーザーとキーボードの対話によって現れる可視位置の指示。

カラー・パレット. アプリケーションの GUI の中の任意のパーツのカラーを変更するために使用するカラーのセット。

カレンダー・パーツ. カレンダーを追加するパーツであり、ユーザーはテキスト、カラー、およびその他の属性を組み込むためにこれを変更することができます。

簡略記号. 選択項目のテキスト内の 1 つの文字で、その文字の下に下線を引いて示されます。 *略号の選択*も参照してください。

キャンバス付きウィンドウ・パーツ. ウィンドウ・パーツとキャンバス・パーツの組み合わせ。ウィンドウ・パーツおよびキャンバス・パーツも参照してください。

キャンバス・パーツ. 他のいろいろなパーツを指示してクリックし、それらを位置指定して、グラフィカル・ユーザー・インターフェースを作成するようにそれらを編成するパーツ。キャンバス・パーツは、ウィンドウ・パーツまたは

ノートブック・ページ・パーツのいずれかのクライアント域を占めます。キャンバス・パーツ付きノートブック・ページおよびキャンバス・パーツ付きウィンドウも参照してください。

キャンバス・パーツ付きノートブック・ページ. ノートブック・ページ・パーツとキャンバス・ページ・パーツの組み合わせ。ノートブック、キャンバス・パーツも参照してください。

強調. カラー変更を強調表示したり、あるいはオブジェクトまたは選択項目と対話するユーザーの能力に影響するそれらのオブジェクトまたは選択項目に関連する条件の可視な指示を強調表示すること。 また、強調により、選択項目またはオブジェクトの状態についての追加情報がユーザーに与えられます。

共通ユーザー・アクセス・アーキテクチャー (CUA アーキテクチャー). 人間とワークステーションまたは端末装置の間のダイアログの指針。

共用コンポーネント. 複数のプロジェクトがアクセスできるコンポーネント。

組み合わせボックス. 入力フィールドとリスト・ボックスの機能を結合する制御。組み合わせボックスには、入力フィールドを完成させるためにユーザーがスクロールして選択することのできるオブジェクトのリストが入っています。これがない場合は、ユーザーはテキストを直接入力フィールドに入力することができます。 VisualAge RPG では、パーツ・パレットまたはパーツ・カタログ内の組み合わせボックス・パーツを指してクリックし、それを設計ウィンドウ上に置くことができます。

クライアント. (1) サーバーにデータを提供するために、サーバーに依存しているシステム。 (2) VARPG アプリケーションが実行される PWS 。*DDE クライアント*も参照してください。

クライアント域. ユーザーのワークスペースであるウィンドウ内の部分であり、ここでユーザーは情報を入力したり、選択フィールドから選択項目を選択します。プライマリー・ウィンドウにおいて、ユーザーが作業するオブジェクトをアプリケーション・プログラマーが提供する領域。

クライアント／サーバー. 分散データ処理における相互作用の形式であり、一方のサイトのプログラムが別のサイトのプログラムに要求を送信し、応答を待機します。 要求側のプログラムをクライアントと呼び、応答側のプログラムをサーバーと呼びます。クライアント、サーバー、*DDE クライアント*、*DDE サーバー*も参照してください。

グラフィカル・ユーザー・インターフェース (GUI). 高解像度グラフィックスを利用したユーザー・インターフェースのタイプ。グラフィカル・ユーザー・インターフェースには、グラフィックスの組み合わせ、オブジェクト - アクション・パラダイム、ポインティング・デバイスやメニュー・バーおよびその他のメニューの使用、オーバーラップ・ウィンドウ、およびアイコンが組み込まれています。

グラフィック・プッシュボタン・パーツ. グラフィックのラベルが付いたプッシュボタンで、ユーザーがこれを選択した時に開始されるアクションを表します。プッシュボタン・パーツと対比。

グラフ・パーツ. ユーザーが GUI にグラフを追加するために使用できるパーツ。使用可能なグラフ・スタイルは折れ線グラフ、棒グラフ、折れ線と棒グラフ、または円グラフです。

クリック. 選択項目またはオブジェクトからポインターを外さないでマウス・ボタンを押して放すこと。 *ダブルクリック*も参照してください。

クリップボード. データを一時的に保持するためにシステムによって提供された記憶領域。クリップボード中のデータは他のアプリケーションにも使用可能です。

グループ・ボックス・パーツ. 制御グループの回りを取り囲む四角形のフレームであり、これらの制御が関連していることを示し、そのグループに対して任意指定のラベルを提供します。 VisualAge RPG では、パーツ・パレットまたはパーツ・カタログ内のグループ・ボックス・パーツを指してクリックし、それを設計ウィンドウ上に置くことができます。

グループ・マーカー. パーツをグループ内の最初のものであるとして認識するマーク。 パーツのグループ内でカーソルを移動している時に、最後のパーツに達すると、カーソルはグループ内の最初のパーツに戻ります。

コールド・リンク会話. DDE において、クライアント・プログラムからサーバー・プログラムに対して行われる明示的な要求。サーバー・プログラムはこの要求に応答します。*ホット・リンク会話*と対比。

構成. 情報処理システムのハードウェアおよびソフトウェアが編成され、相互接続される方法 (T)。

構文検査. ソースの編集中に、各行の構文が正しいことを確認します。確認することによって、コンパイル・エラーを回避することができます。このオプションは、オンまたはオフに設定することができます。一定の仕様タイプ (C 仕様など) または特定のストリングを持つ行だけを表示することができます。

項目. 動的データ交換におけるデータの単位。 たとえば、スプレッドシートにおける 左上のセルの位置は、行 1、列 1 です。このセル位置は、項目 R1C1 と呼ばれることもあります。

コンテナー・パーツ. 関連レコードを保管するパーツであり、それらを詳細、アイコン、またはツリー・ビューで表示します。

コンパイル. ソース・プログラムを実行可能プログラム (オブジェクト・プログラム) に変換すること。

コンポーネント. プロジェクト内の関連ファイルの機能上のソート。 コンポーネントは、制御仕様書に NOMAIN および EXE キーワードがない時に作成されます。

コンポーネント参照パーツ. 1 つのコンポーネントが VARPG アプリケーション中の別のコンポーネントと通信できるようにするパーツ。

［サ行］

サーバー. ネットワーク中にあり、他のシステム (クライアントと呼ばれる) の要求を処理するシステム。

サーバーの別名. サーバー名の代わりに使用することができるユーザー定義の名前。

最小化ボタン. タイトル・バーの右端のボタンの隣にあるボタンで、ウィンドウを最小サイズに縮小します。 *最大化ボタンおよび[隠す] ボタン*と対比。

最大化ボタン. タイトル・バーの右端にあるボタンで、ウィンドウをその最大サイズに拡大するためにクリックします。 *最小化ボタン*、 *[隠す] ボタン*と対比。

索引. リスト・ボックスまたは組み合わせボックスなどの VARPG パーツ中の項目の ID。

サブファイル・パーツ. レコードのリストを表示するために使用するパーツであり、沢山のフィールドから成っています。 このパーツは AS/400 サブファイルに類似しています。*サブファイル・フィールド*も参照してください。

サブファイル・フィールド. サブファイル・パーツ内のフィールドを定義するために使用するフィールド。サブファイル・パーツも参照してください。

サブプロシージャー. サブプロシージャーはメイン・ソース・セクションの後に指定されるプロシージャーです。メイン・ソース・セクションの定義仕様の中に対応するプロトタイプが入っていなければなりません。

サブメニュー. 別のメニューの中のカスケード選択項目から表示され、それに関連した選択項目が入っているメニュー。サブメニューはプルダウン・メニューまたはポップアップ・メニューの長さを縮小するために使用します。 サブメニュー・パーツも参照してください。

サブメニュー・パーツ. メニュー項目または既存のメニューからサブメニューを開始したり、あるいはメニュー・バー上のメニュー項目からプルダウン・メニューを開始するために使用するパーツ。サブメニューおよび*メニュー項目・パーツ*も参照してください。

参照解除. パーツと AS/400 データベース・フィールドとの間の関連を除去するアクション。

参照フィールド. AS/400 のデータベース・フィールドであり、入力フィールド・パーツがそのプロパティーを継承することができます。

状況バー. ウィンドウのパーツであり、現行のビューまたはオブジェクトの状態を示す情報を表示します。 *情報域*も参照してください。

状況バー・パーツ. ウィンドウ上にあり、そのウィンドウの処理またはアクションについての追加情報を表示できるパーツ。

詳細ビュー. オブジェクトについての説明情報を提供するテキストが小アイコンに結合されている標準目次ビュー。

情報域. カーソルが置かれているオブジェクトまたは選択項目についての情報が表示されるウィンドウのパーツ。 情報域には、処理の通常の完了についてのメッセージも表示されます。 *状況バー*も参照してください。

情報表示機能 (IPF). プログラマブル・ワークステーション上でオンライン・ヘルプを作成するのに使用するツール。

情報表示機能 (IPF) ファイル. アプリケーションのヘルプ・ソースが保管されるファイル。

進行状況バー・パーツ. ファイルのコピー、データベースのロード、などの処理の進行をグラフィカルに示すために使用できるパーツ。

進行状況表示. 処理の進行状況をユーザーに知らせるために使用される 1 つまたは複数の制御。

水平方向のスクロール・バー・パーツ. ウィンドウに水平方向のスクロール・バーを追加するパーツ。このパーツによって、ユーザーは、情報のペインを左から右または右から左にスクロールできるようになります。

スクロール・バー. 特定の方向にまだ情報があることをユーザーに示し、マウスまたはページ・キーを使用して移動しその情報を表示できることを示すパーツ。

スピン・ボタン・パーツ. 入力フィールドの 1 つのタイプであり、関連があるが互いに排他的な選択項目のリングを表示し、ユーザーはこれをスクロールして 1 つの選択項目を選択することができます。 ユーザーは入力フィールドに正しい選択を入力することもできます。 パーツ・パレットまたはパーツ・カタログ内のスピン・ボタン・パーツを指してクリックし、それを設計ウィンドウ上に置くことができます。

スライダー・アーム. スライダー内の可視標識であり、ユーザーは数値を変更するためにこれを移動することができます。

スライダー・パーツ. ユーザー・インターフェースのビジュアル・コンポーネントであり、数量を表し、その数量に使用できる値の範囲に対するリレーションシップを表します。ユーザーはこの数量の値を変更することもできます。パーツ・パレットまたはパーツ・カタログ内のスライダー・パーツを指してクリックし、それを設計ウィンドウ上に置くことができます。

スレッド. プロセス中で実行される操作の最小の単位。

静的テキスト・パーツ. 他のパーツのラベルとして使用されるパーツ。入力フィールド・パーツに対するプロンプトなど。

セカンダリー・ウィンドウ. プライマリー・ウィンドウの中の情報に依存した情報が入っているウィンドウで、プライマリー・ウィンドウ中の対話を補足するために使用されます。プライマリー・ウィンドウも参照してください。ポップアップ・ウィンドウの同義語。

設計ウィンドウ. GUI Designer の中のウィンドウであり、ユーザー・インターフェースを作成するためのパーツが置かれています。

選択ボタン. *左マウス・ボタン* を参照してください。

選択枠. VARPG パーツまたはオーダーメイド・パーツの回りに現れる可視枠で、マウスまたはキーボードを使用して移動できます。

ソース・ディレクトリー. VARPG アプリケーションのすべてのソース・ファイルが保管されているディレクトリー。

ソース・パーツ. ソース・パーツの状態が変更された時には常にターゲット・パーツに通知できるパーツ。 ソース・パーツは複数のターゲットを持つことができます。

操作ボタン. *右マウス・ボタン* を参照してください。

外枠パーツ. パーツのグループの回りを囲む長方形ボックスのパーツで、その中のすべてのパーツが関連していることを示します。

［タ行］

ターゲット・ディレクトリー. ビルド後にコンパイル済みの VARPG アプリケーションが保管されるディレクトリー。 *ターゲット・フォルダー*と対比。

ターゲット・パーツ. ソース・パーツの状態が変更された時には常に、ソース・パーツからリンク・イベントを受信するパーツ。

ターゲット・フォルダー. VARPG アプリケーションを表すアイコンが入っているオブジェクト。

ターゲット・プログラム. プロジェクトによってビルドされるオブジェクト。ダイナミック・リンク・ライブラリー (DLL) など。

タイトル・バー. 各ウィンドウの最上部にある領域で、システム・メニュー・シンボルが入っています。

ダイナミック・リンク・ライブラリー (DLL). リンク時ではなくロード時または実行時にプログラムにバインドされる実行可能コードおよびデータが入っているファイル。 ダイナミック・リンク・ライブラリー中のコードおよびデータは複数のアプリケーションで同時に共用することができます。

タイマー・パーツ. 2 つのイベント間の時間間隔を追跡し、その間隔が経過した時に 2 番目のイベントを起動するために使用するパーツ。

縦方向のスクロール・バー・パーツ. ウィンドウに縦方向のスクロール・バーを追加するパーツ。このパーツによって、ユーザーは、情報のペインを縦方向にスクロールできるようになります。

ダブルクリック. マウス・ボタンを素早く 2 回押すこと。

タブ・ストップ. ユーザーがインターフェースの中を移動するためにタブ・キーを使用する時に、パーツにフォーカスできるように、そのパーツのタブ・ストップを設定するために使用する属性。

チェック・ボックス・パーツ. 選択項目を表している関連テキストを持つ正方形のボックス。ユーザーが選択項目を選択すると、その選択項目が選択されたことを示す標識がチェック・ボックスの中に現れます。 その選択項目を再び選択することによって、チェック・ボックスを取り消すことができます。 VisualAge RPG では、パーツ・パレットまたはパーツ・カタログ内のチェック・ボックス・パーツを指してクリックし、それを設計ウィンドウ上に置くことができます。

直接編集. ユーザーがオブジェクトをマウスでドラッグしたり、オブジェクトのポップアップ・メニューと対話することによって、そのオブジェクトを処理することのできる手法を使用すること。

ツールバー. ユーザーがマウスを使用して実行することのできるアクションを表す 1 つまたは複数のグラフィカル選択項目が入っているメニュー。

ツリー・ビュー. オブジェクトの目次を階層的に表示する方法。

データベース. (1) 複数ユーザーのためのデータをオンデマンドで受け入れ、保管し、および提供するための指定の構造体を持つデータの集合。(T) (2) システム中に保管されているすべてのデータ・ファイル。

データ・オブジェクト. テキスト、グラフィックス、オーディオ、またはビデオなどの情報を運ぶオブジェクト。

デフォルト. ユーザーによって値が指定されていない時に、システムまたはプログラムによって自動的に供給または想定される値。デフォルト値はプッシュボタンまたはグラフィック・プッシュボタンに割り当てることができます。

デフォルト・アクション. 何らかのアクション (たとえば Enter キーを押す) が取られた時に実行されるアクション。

トークン強調表示. コードの読みやすさを向上させます。異なるカラーまたはフォントを使用して、異なる言語コンポーネントの強調表示を構成して、プログラム構造を識別することができます。トークンの強調表示はオンにしたりオフにしたりすることができます。

動的データ交換 (DDE). プログラム間またはプログラムとデータ・ファイル・オブジェクトの間のデータの交換。 1 つのプログラムまたはセッションの中の情報に対して行われた変更が、他のプログラムによって作成された同じデータに適用されます。 *DDE 会話*、*DDE クライアント*、*DDE サーバー*も参照してください。

トピック. 動的データ交換 (DDE) において、DDE 会話のサブジェクトとなるデータのセット。

ドラッグ. マウスを使用してオブジェクトを移動またはコピーすること。たとえば、マウス・ボタンを押したままで移動することにより、ウィンドウ・ボーダーをドラッグしてそのウィンドウを大きくすることができます。 *ドラッグ・アンド・ドロップ* も参照してください。

ドラッグ・アンド・ドロップ. マウスを使用してオブジェクトを直接移動し、別の場所に置くこと。

ドロップダウン組み合わせボックス. 組み合わせボックスの変形であり、ユーザーが明示的にアクションを実行して可視にするまでは、リスト・ボックスは隠されています。

ドロップダウン・リスト. 現在の選択項目だけが可視になっている単一の選択フィールド。他の選択項目が入っているリスト・ボックスを表示するためのアクションを明示的に実行するまでは、他の選択項目は隠されたままです。

[ナ行]

ナビゲーション・パネル. サブファイル中のレコードの可視選択を制御するために使用するボタンのグループ。

入出力 (I/O). コンピューターに提供されるデータ、またはコンピューター処理の結果としてのデータ。

入力フィールド・パーツ. ユーザーが情報を入力できる画面上の領域 (そのフィールドが読み取り専用でない場合)。通常、入力フィールドの境界が指示されています。 VisualAge RPG では、パーツ・パレットまたはパーツ・カタログ内の入力フィールド・パーツを指してクリックし、それを設計ウィンドウ上に置くことができます。

入力フォーカス. キーボードまたはマウスのどちらからでもユーザーの対話が可能なウィンドウの領域。

ノートブック・パーツ. ノートブックのグラフィカル表現。 ノートブック・ページをノートブック・パーツに追加し、その後でそれらのページをタブ付きの区切り記号によって分離されたセクションにグループ化することができます。 Windows では、ノートブックのことを Windows タブ制御と呼ぶことがあります。*ノートブック・ページ・パーツ*、*キャンバス・パーツ付きノートブック・ページ*も参照してください。

ノートブック・ページ・パーツ. *ノートブック・パーツ*にページを追加するために使用するパーツ。*ノートブック*も参照してください。

[ハ行]

パーツ. VARPG アプリケーションの GUI を構成するオブジェクト。

パーツ・カタログ. VARPG アプリケーション用のグラフィカル・ユーザー・インターフェースを作成するために使用するすべてのパーツのための記憶スペース。

パーツ・パレット. アプリケーション用の現在のグラフィカル・ユーザー・インターフェースをビルドするのに最も適しているパーツの集合。 1 つの GUI を終了したら、パレットをきれいにふき取り、次のアプリケーションに必要なパーツをパーツ・カタログから追加することができます。

パッケージ. VARPG アプリケーションのすべてのパーツを配布用に収集するために使用する機能。

非活動ウィンドウ. 指定の瞬間にキーボード入力を受け入れることができないウィンドウ。

左マウス・ボタン. デフォルトとして、選択のために使用するマウス上の左のボタン。

ビルド. VARPG アプリケーションのコンポーネントを構成するいろいろなソース・コードが、実行可能なバージョンのアプリケーションを作成するためにコンパイルされ、リンクされるプロセス。

ファイル. 関連データの集合であり、割り当てられた名前で保管されたり検索されます。ファイルには、プログラム (プログラム・ファイル・オブジェクト) を開始する情報、テキストまたはグラフィック (データ・ファイル・オブジェクト) を含む情報、あるいは一連のコマンド (バッチ・ファイル) を処理する情報を組み込むことができます。

フィールド. (1) ウィンドウ内の識別可能な領域であり、たとえば、ユーザーがテキストを入力する入力フィールドなど。 (2) 名前や容量などの関連したバイトのグループであり、レコード中の単位として扱われます。

フォーカス. *入力フォーカス*の同義語。

フォント・パレット. アプリケーション GUI の中のパーツのフォントを変更するために使用できるフォントのセット。

復元ボタン. ウィンドウが最大化された後に、タイトル・バーの右端の隅に現れるボタン。復元ボタンを選択すると、ウィンドウは最大化される前のサイズと位置に戻ります。*最大化ボタン*も参照してください。

複数行編集 (MLE) パーツ. ユーザーが複数行のテキストを入力することのできる入力フィールドを表すパーツ。

プッシュボタン・パーツ. ユーザーがプッシュボタンを選択した時に開始されるアクションを表すテキストがラベルとして付いているボタン。 パーツ・パレットまたはパーツ・カタログ内のプッシュボタン・パーツを指してクリックし、それを設計ウィンドウ上に置くことができます。「グラフィック・プッシュボタン・パーツ」も参照してください。

プライマリー・ウィンドウ. ユーザーとアプリケーションの間の主たる会話が行われるウィンドウ。メイン・ウィンドウ同義。

プラグイン. ユーザーまたは外部のベンダーによって作成された機能で、 VARPG プログラムで使用することができます。

プルダウン・メニュー. メニュー・バーの選択した項目またはシステム・メニュー・シンボルから拡張されるメニュー。 プルダウン・メニューの中の選択項目はある意味で互いに関連しています。

プログラマブル・ワークステーション (PWS). ある程度の処理能力を持っているワークステーションで、ユーザーがその機能を変更することができるワークステーション。

プロシージャー. プロシージャーは、CALLP 命令コードで呼び出すことのできるコードの任意の部分です。

プロシージャー・インターフェース定義. プロシージャー・インターフェース定義は、プロシージャーの定義の中のプロトタイプ情報の反復です。 これは、プロシージャーの入力パラメーターを宣言したり、プロシージャーの内部定義が外部定義 (プロトタイプ) と矛盾しないことを保証するために使用します。

プロジェクト. ダイナミック・リンク・ライブラリー (DLL) または実行可能ファイル (EXE) などの単一のターゲットをビルドするのに必要なデータおよびアクションの完全なセット。

プロトタイプ. プロトタイプは、呼び出しインターフェースの定義です。これには次のような情報が組み込まれています : 呼び出しがバインド (プロシージャー) であるかダイナミック (プログラム) であるか ; 外部名 ; パラメーターの数および特長 ; どのパラメーターを渡さなければならないか ; 戻り値のデータ・タイプ (プロシージャーの場合) 。

プロパティー・ノートブック. タブ区分ページで複数のセクションに分離されたページを含むバウンド・ノートブックに類似したグラフィカル表現。 1 つのセクションから別のセクションに移動するためには、ノートブックのタブを選択してください。

プロンプト. (1) ユーザーの応答を要求するために、プログラムによって送信される視覚または音によるメッセージ。(T) (2) ユーザーからの入力を要求したり、操作情報を提供する表示記号またはメッセージ。ユーザーは続行するためには、このプロンプトに応答しなければなりません。

ポイント／クリック. (1) パーツをパーツ・パレットまたはパーツ・カタログから GUI 設計ウィンドウ、アイコン・ビュー、またはツリー・ビューにコピーするために使用される選択方式。 (2) パーツを希望するいずれかのビューに入れるためには、そのパーツを指示してクリックし、次にカーソルを選択したウィンドウに移動して、そのカーソルを指示して、パーツを表示したい位置でクリックします。アイコンおよびツリー・ビューでは、パーツは親パーツ上に置かれるので、ユーザーはそれを設計ウィンドウ中の希望する位置に移動しなければなりません。

ぼかし表示. ユーザーがそのパーツを選択したり直接操作することができないことを示すために、明度が減光されて表示されているもの。

ボタン. (1) アクションを要求したり開始するために使用するマウスなどのポインティング・デバイス上のメカニズム。 (2) ウィンドウ内のグラフィカル・メカニズムであり、これを選択するとアクションが実行されます。 ボタンの 1 つの例に [OK] プッシュボタンがあり、これを選択するとアクションが開始されます。

ホット・リンク会話. DDE において、サーバー上でデータが変更された時の、サーバー・プログラムによるクライアント・プログラムの自動更新。 *コールド・リンク会話*と対比。

ポップアップ・ウィンドウ. 固定サイズの移動可能なウィンドウであり、アプリケーションがユーザーの要求の処理を続行できるように、アプリケーションに必要な情報をユーザーがこのウィンドウに提供します。 *セカンダリー・ウィンドウ*と同義。

ポップアップ・メニュー. オブジェクトに関連するメニューを要求した時に、そのオブジェクトの横に表示されるメニュー。 これには、オブジェクトの現在のコンテキストに適した選択項目が含まれています。

ポップアップ・メニュー・パーツ. オブジェクトに関して要求した時に、そのオブジェクトの横に表示されるパーツ (そのパーツがユーザー・インターフェース上のオブジェクトに追加されている場合)。パーツ・パレットまたはパーツ・カタログ内のポップアップ・メニュー・パーツを指してクリックし、それを設計ウィンドウ上に置くことができます。

[マ行]

マウス. キーボードを使用しないで画面上のポインターを位置付けるために使用し、1 つまたは複数のプッシュボタンを持つ装置。実行する選択項目または機能を選択するために使用したり、画面上で操作を実行するために使用します。たとえば、ある場所から別の場所に線をドラッグしたり描いたりします。

マウス・ポインター. *カーソル*の同義語。

マウス・ボタン. 選択項目を選択したり、アクションを開始したり、あるいはポインターでオブジェクトを操作するために使用するマウス上の機構。 *左マウス・ボタン* および*右マウス・ボタン* も参照してください。

右マウス・ボタン. デフォルトとして、操作のために使用するマウス上の右のボタン。

メイン・ウィンドウ. *プライマリー・ウィンドウ*を参照してください。

メイン・ソース・セクション. VARPG プログラムでは、メイン・ソース・セクションには、モジュールのすべてのグローバル定義が含まれています。コンポーネントの場合には、このセクションには、アクションおよびユーザー・サブルーチンも含まれます。

メイン・プロシージャー. メイン・プロシージャーは、プログラムの入り口プロシージャーとして指定でき、最初に呼び出された時に制御を受け取るサブプロシージャーです。 メイン・プロシージャーは EXE の作成時にしか作成されません。 *EXE モジュール*を参照してください。

メッセージ. (1) ユーザーによって要求された情報ではなく、予期しないイベントに応答してプロダクトによって表示されるか、あるいは何か不都合なことが起こった時に表示される情報。 (2) ある人またはプログラムから別の人またはプログラムに対して送信される通信。

メッセージ・サブファイル・パーツ. プログラム・ロジックに提供されている事前定義メッセージまたはテキストを表示できるパーツ。

メッセージ・ファイル. アプリケーション・メッセージが入っているファイル。このファイルは、ビルド・プロセス時にメッセージ・ソース・ファイルから作成されます。*ビルド*も参照してください。

メディア・パーツ. プログラムに、サウンド・ファイルおよびビデオ・ファイルを処理する機能を与えるパーツ。

メディア・パネル・パーツ. ユーザーに他のパーツ全般の制御を与えるために使用するパーツ。たとえば、メディア・パネル・パーツは、メディア・パーツのボリュームを制御するために使用することができます。

メニュー. オブジェクトに適用することのできる選択項目のリスト。 メニューには、一定のコンテキストの下では選択できない選択項目も含まれることがあります。 これらの選択項目はぼかし表示されます。

メニュー項目パーツ. メニュー上のグラフィカルまたはテキスト項目であるパーツ。ユーザーは何らかの方法でオブジェクトを処理するためのメニュー項目を選択します。

メニュー・バー・パーツ. ウィンドウの上部近くで、タイトル・バーの下で、ウィンドウの他の部分より上にある領域で、他のメニューにアクセスできる選択項目が含まれています。 VisualAge RPG では、パーツ・パレットまたはパーツ・カタログ内のメニュー・バー・パーツを指してクリックし、それを設計ウィンドウ上に置くことができます。

［ヤ行］

ユーザー定義パーツ. ユーザーがカスタマイズした 1 つまたは複数のパーツから成るパーツであり、再使用のためにこれをパーツ・パレットまたはパーツ・カタログに保管します。パレットまたはカタログの中にある時には、他の VARPG パーツと同様に、このパーツをポイントおよびクリックして設計ウィンドウに入れることができます。

ユーティリティー DLL. *NOMAIN モジュール* を参照してください。

［ラ行］

ラジオ・ボタン・パーツ. 横にテキストが付いている円。 ラジオ・ボタンは、固定した選択項目のセットから 1 つの選択項目を選択できる組み合わせです。選択項目が選択されると、円の一部が塗りつぶされます。 パーツ・パレットまたはパーツ・カタログ内のラジオ・ボタン・パーツを指してクリックし、それを設計ウィンドウ上に置くことができます。

リスト・ボックス・パーツ. ユーザーが選択できるスクロール可能な選択項目が入っている制御。 VisualAge RPG では、パーツ・パレットまたはパーツ・カタログ内のリスト・ボックス・パーツを指してクリックし、それを設計ウィンドウ上に置くことができます。

略号の選択. 選択項目の簡略記号を入力することにより、その選択項目を選択する選択手法。

リンク・イベント. ソース・パーツの状態が変更された時に、常にターゲット・パーツが受け取るイベント。

例外. (1) プログラム言語において、実行時に起こる可能性のある異常な状態、通常の実行順序から逸脱する原因となる可能性のある異常な状態であり、それを定義したり、引き起こしたり、認識、無視、および処理するための機能がプログラム言語中に存在しています。(I) (2) VisualAge RPG において、ユーザーが要求したアクションが予定通りに完了するのを妨げたり、妨げる可能性のあるイベントまたは状態。 例外は、プロダクトがユーザーの入力を解釈できない時に起こります。

［ワ行］

ワークエリア. ユーザーのタスクにしたがってオブジェクトを編成するために使用する領域。ユーザーがワークエリアをクローズすると、このワークエリアにあるオブジェクトからオープンされたすべてのウィンドウがワークプレースから除去されます。

ワークステーション. ユーザーが作業することのできる装置。*プログラマブル・ワークステーション*も参照してください。

ワークプレース. 画面全体を埋める領域で、ユーザー・インターフェースを構成するすべてのオブジェクトを保持する領域。

枠のサイズ変更. マウスまたはキーボードを使用してパーツ (またはパーツのセット) をサイズ変更するために選択するパーツ (またはパーツのセット) の回りの枠またはフレーム。

［数字］

1 バイト文字セット (SBCS). 各文字が 1 バイトのコードで表される文字セット。 *2 バイト文字セット (DBCS)* と対比。

2 バイト文字セット (DBCS). 各文字が 2 バイトで表される文字のセット。 256 個のコード・ポイントでは表しきれない記号が入っている、日本語、中国語、および韓国語、などの言語は 2 バイト文字セットを必要とします。各文字が 2 バイトを必要とするので、DBCS 文字の入力、表示、および印刷では、 DBCS をサポートするハードウェアおよびソフトウェアが必要です。システムは次の 4 つの 2 バイト文字セットをサポートします: 日本語、韓国語、中国語 (簡体字)、および中国語 (繁体字)。 *1 バイト文字セット (SBCS)* と対比。

A

activeX パーツ. プロジェクトに ActiveX 制御オブジェクトを追加するパーツ。これにより、VARPG アプリケーションは、その属性にアクセスして、イベントをモニターすることができます。

API. アプリケーション・プログラミング・インターフェース。

ASCII (情報交換用米国標準コード). 7 ビット (パリティー検査を組み込む場合は 8 ビット) のコード化文字から成るコード化文字セットを使用する標準コードであり、データ処理システム、データ通信システム、および関連機器の間で情報交換用に使用されます。 ASCII セットは制御文字とグラフィック文字から成ります。(A)

B

BMP. ビットマップ・ファイルのファイル拡張子。

C

CONFIG.SYS. DOS、OS/2、または Windows オペレーティング・システムのブート・ドライブのルート・ディレクトリーにある構成ファイル。 この中には、ハードウェアおよびソフトウェアをインストールし実行するために必要な情報が入っています。

CUA アーキテクチャー. 共通ユーザー・アクセス・アーキテクチャー。

D

DBCS. 2 バイト文字セット。

DDE. 動的データ交換。

DDE 会話. DDE クライアントと DDE サーバーの間のデータの交換。 *コールド・リンク会話*および*ホット・リンク会話*も参照してください。

DDE クライアント. DDE 会話を開始するアプリケーション。 *DDE サーバー*と対比。*DDE クライアント・パーツ*、*DDE 会話*も参照してください。

DDE クライアント・パーツ. 動的データ交換 (DDE) プロトコルをサポートする他のアプリケーション (スプレッドシート・アプリケーションなど) とデータを交換するために使用するパーツ。

DDE サーバー. データを別の DDE 可能アプリケーションに提供するアプリケーション。 *DDE クライアント*と対比。*DDE 会話*も参照してください。

DLL. ダイナミック・リンク・ライブラリー。

E

EBCDIC. 拡張 2 進化 10 進コード。 256 個の 8 ビット文字のコード化文字セット。

EXE. 実行可能ファイルの拡張子。

EXE モジュール. EXE モジュールはメイン・プロシージャーとサブプロシージャーから成ります。これは、 制御仕様に EXE キーワードがある場合に作成されます。すべてのサブルーチン (BEGSR) はプロシージャーに対してローカルでなければなりません。 EXE には、名前がソース・ファイルの名前と一致するプロシージャーが入っていなければなりません。これが EXE のメイン入り口点であり、すなわちメイン・プロシージャーとなります。

G

GUI Designer . パーツをパーツ・パレットから設計ウィンドウにドラッグ／ドロップすることによってインターフェースを作成するのに使用する一組のツール。

I

ICO. アイコン・ファイルのファイル拡張子。

INI. OS/2 または Windows オペレーティング・システムにおいて、アプリケーションから別のアプリケーションを呼び出すために保持する必要のあるアプリケーション固有の情報が入っているファイルのファイル拡張子。

IPF. 情報表示機能

J

JAR ファイル (.jar). Java では、Java ARchive の省略形。多くのファイルを 1 つのファイルに集めるために使用されるファイル形式。

Java. リモート・オブジェクト間の対話をサポートする移植可能解釈コードのためのオブジェクト指向プログラミング言語。Java は、Sun Microsystems, Incorporated によって開発され、明示されたものです。

Java 2 Software Development Kit (J2SDK). Sun Microsystems 社が Java 開発者のために配布するソフトウェア。このソフトウェアには、Java インタープリター、Java クラス、および Java 開発ツールが組み込まれています。開発ツールには、コンパイラー、デバッガー、逆アセンブラー、AppletViewer、スタブ・ファイル生成プログラム、および文書生成プログラムが組み込まれています。

Java Bean パーツ. VARPG アプリケーションが Sun Microsystem の JavaBeans にアクセスできるようにするパーツ。

Java Runtime Environment (JRE). JRE を再配布したいエンド・ユーザーおよび開発者のための Java Developer Kit のサブセット。JRE は、Java 仮想マシン、Java コア・クラス、およびサポート・ファイルからなります。

Java 仮想マシン (JVM). Java Runtime Environment (JRE) の一部で、Java バイトコードの解釈を受け持ちます。

Java データベース接続性 (JDBC). Java と広範囲のデータベースの間のデータベースから独立した接続のための業界標準。JDBC は、SQL 基本データベース・アクセスのための呼び出しレベルのアプリケーション・プログラミング・インターフェース (API) を提供します。

Java ネイティブ・インターフェース (JNI). Java 仮想マシン (JVM) の内部で実行する Java コードが他のプログラム言語で書かれた機能と相互に関連し合って機能できるようにするプログラミング・インターフェース。

JavaBeans. Java では、移植可能で、プラットフォームから独立した再使用可能コンポーネント・モデルです。

M

MID. MIDI ファイルのファイル拡張子。

MIDI ファイル. 電子楽器ディジタル・インターフェース・ファイル。

N

NOMAIN モジュール. サブプロシージャーだけが入っているモジュール。 この中には、アクションまたはスタンドアロン・ユーザー・サブルーチンはありません。 NOMAIN モジュールは、制御仕様の中に NOMAIN キーワードがある場合に作成されます。

O

odbc/jdbc パーツ. VAPRG アプリケーションが Windows ODBC API または Sun Microsystem 社の JDBC API をサポートするデータベース・ファイルにアクセスして、それを処理できるようにするパーツ。

P

PWS. プログラマブル・ワークステーション。

S

SBCS. 1 バイト文字セット。

secure sockets layer (SSL). Netscape Communications Corp. および RSA Data Security, Inc によって開発された一般的なセキュリティー機構。SSL によって、クライアントはサーバーを認証できるようになり、すべてのデータおよび要求が暗号化できるようになります。SSL によって保護されたセキュア・サーバーの URL は、http ではなく、https で始まっています。

SSL. Secure sockets layer。

W

WAV. ウェーブ・ファイルのファイル拡張子。

［特殊文字］

***component パーツ.** コンポーネントの『パーツ表現』であるパーツ。 各コンポーネントごとに 1 つの *component パーツが自動的に作成され、それは目には見えません。

[隠す] ボタン. タイトル・バーにあるボタンで、ユーザーがウィンドウをクローズしないでワークプレースからそのウィンドウを除去する場合にクリックします。 ウィンドウが隠されると、ウィンドウの状態 (ウィンドウ・リストに表示される) が変わります。*最大化ボタン*および*最小化ボタン*と対比。

参考文献

WebSphere Development Studio Client と関連したトピックの追加情報については、以下の IBM 資料を参照してください。

WebSphere Development Studio Client マニュアル:

- *WebSphere Development Studio Client for iSeries ご使用に際して*, SD88-5054-03。WebSphere Development Studio Client for iSeries についての情報を提供し、各種コンポーネントの概要、それらがどのように関連して機能するか、およびそれを使用することによる業務上の利点を示します。

VisualAge RPG マニュアル:

- *VisualAge for RPG プログラミング*, SC88-5607-04 には、VisualAge RPG を使用してアプリケーションを作成する場合の具体的な情報が記載されています。これには、アプリケーション開発サイクルのすべての段階で実行する必要があるステップが、設計からパッケージおよび分配まで説明されています。VARPG アプリケーション開発の概念と工程を明らかにするために、プログラミング例が収録されています。
- *ADTS/CS VisualAge for RPG パーツ解説書*, SD88-5040-03 には、VARPG の各パーツ、パーツ属性、パーツ・イベント、およびイベント属性が説明されています。本書は、VisualAge RPG を使用してアプリケーションを開発しているすべての方々の参照資料です。
- *VisualAge for RPG WINDOWS 版 言語解説書*, SC88-5608-04 には、VARPG 言語およびコンパイラーに関する参照情報が提供されます。
- *Java for RPG プログラマー*。Java 言語 (および RPG IV) を RPG 言語と比較しながら紹介しています。これは、Java を使って様々なことを経験するための最初のステップとして格好なものです。これには、Java および VisualAge for Java に関する対話式 CD チュートリアルも MINDQ 別に含まれています。
- *Experience RPG IV Tutorial*。RPG IV および ILE について肩の凝らない段階的方法によって学習するための対話式 CD チュートリアルです。この資料は、この興味深い RPG の新規バージョンでの実技経験を得る上で役立つ質問および演習が記載されたハンドブックです。
- VisualAge RPG ユーザーの関心を引く IBM 以外の資料に *VisualAge for RPG by Example* があります。

インターネットにアクセスできる場合には、以下の Web サイトから iSeries および AS/400e についての最新の情報および資料を入手できます。

`http://www.ibm.com/eserver/iseries/infocenter`

Application Development Manager マニュアル:

- *AS/400 V3 適用業務開発ツール (ADT) セット /400 適用業務開発管理 /400 入門および計画の手引き*, GC88-5245-00 には、Application Development Manager 機能を効率的に使用するために必要な基本概念および計画が説明されています。
- *DTS/ Application Development Manager User's Guide*, SC09-2133-02 には、Application Development Manager 機能に対して定義されるプロジェクトを作成し管理する方法が説明されています。

- *AS/400 アドバンスト・シリーズ V3.6 適用業務開発ツール (ADT) セット OS/400 用 適用業務開発管理 OS/400 用 自習書*, SC88-5417-00。Application Development Manager 機能を使用する実践的な体験を得ることができます。この手引書には、ステップ順に練習することによって、Application Development Manager 機能の使用法が説明されています。
- *ADTS/400: Application Development Manager API Reference*, SC09-2180-00。アプリケーション・プログラマーが Application Development Manager 機能に対する固有のインターフェースを作成する方法が説明されています。

情報表示機能マニュアル:

- *Information Presentation Facility Programming Guide* G25H-7110 には、情報表示機能 (IPF) を構成するエレメントが説明されています。IPF は、オンライン文書機能およびオンライン・ヘルプ機能の設計および開発をサポートするツールです。

SQL マニュアル:

- *IBM SQL Reference Version 2* (SC26-8416) Volume 2 には、以下に示すものの機能比較があります。
  - DB2
  - SQL/DS™
  - DB2/400™
  - DB2/6000™
  - IBM SQL
  - ISO-ANSI (SQL92E)
  - X/Open™ (XPG4-SQL).
- *DB2 Universal Database Administration Guide* S10J-8157 には、DB2 製品の使用および管理に必要な情報が示されています。
- *DB2 Universal Database Embedded SQL Programming Guide* S10J-8158 には、DB2 クライアント／サーバーのファミリー・サーバー (DB2 または DB2/400) にアクセスするアプリケーション・プログラムの設計およびコーディングの方法が説明されています。これには、構造化照会言語 (SQL) の使用およびアプリケーションでの API 呼び出しに関する詳細情報が説明されています。

特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものであり、本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権（特許出願中のものを含む）を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒106-0032
東京都港区六本木 3-2-31
IBM World Trade Asia Corporation
Licensing

以下の保証は、国または地域の法律に沿わない場合は、適用されません。 IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム（本プログラムを含む）との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

Lab Director
IBM Canada Ltd. Laboratory
B3/KB7/8200/MKM
8200 Warden Avenue
Markham, Ontario, Canada L6G 1C7

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、 IBM より提供されます。

本書には、日常のビジネス・プロセスで用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。 これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

この情報をソフトコピーでご覧になっている場合は、写真やカラーの図表は表示されない場合があります。

プログラミング・インターフェース情報

本書は、クライアント / サーバー環境において、VisualAge RPG アプリケーションおよびワークステーションのユーザー・インターフェースの作成と管理するための援助を目的としています。 本書は IBM WebSphere Development Studio Client for iSeries が提供する汎用プログラミング・インターフェースおよび関連ガイダンス情報を文書化したものです。

商標

以下は、IBM Corporation の商標です。

Application System/400	AS/400	AS/400e
Common User Access	CUA	DATABASE 2
DB2	DB2 Connect	DB2 Universal Database
IBM	OS/400	SQL/DS
VisualAge	WebSphere	400

Java およびすべての Java ベースの商標およびロゴは、Sun Microsystems, Inc. の米国およびその他の国における商標です。

Lotus は、Lotus Development Corporation の商標です。

ActiveX、 Microsoft、Windows、および Windows NT は Microsoft Corporation の米国およびその他の国における商標です。

他の会社名、製品名およびサービス名などはそれぞれ各社の商標または登録商標です。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

[カ行]

[サ行]

[タ行]

[ナ行]

[ハ行]

[マ行]

[ヤ行]

[ラ行]

[数字]

A

B

C

D

E

F

T

U

V

W

[特殊文字]

Printed in JAPAN

SC88-5607-05

日本アイ・ビー・エム株式会社
〒106-8711　東京都港区六本木3-2-12