Delphi 2.0J
機能評価ガイド

Borland

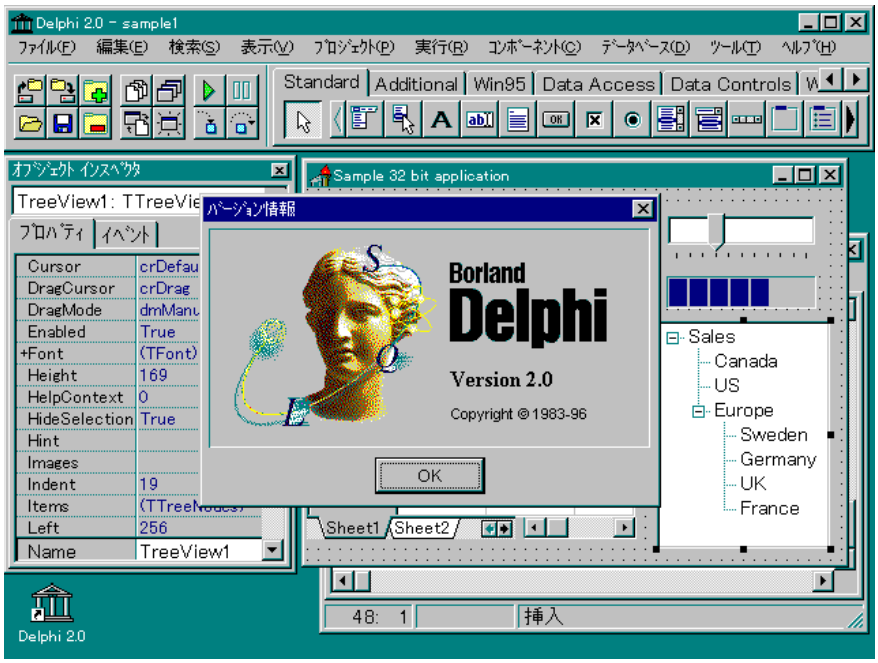
目次

Delphi 2.0 の概要

昨年の発売以来、Delphi は RAD(迅速なアプリケーション開発)の新しい標準の座を確保しました。ネイティブコードコンパイラ、ビジュアル 2Way-Tool、スケーラブルなデータベース技術を組み合わせた Delphi のユニークな特長は、結果として全世界で 20 以上の賞を受賞し、ビジュアルツールの分野での急成長をなしとげることになりました。Delphi は、数多くのライブラリやツールを提供するすばらしいサードパーティ製品や、いくつもの雑誌、書籍などによって支えられています。

Delphi 2.0 と呼ばれる新しい Delphi の 32 ビット版では、開発者の生産性やアプリケーションのパフォーマンスを向上させるためのいくつかの新しい技術が組み込まれています。Delphi 2.0 は、新しい 32 ビットの最適化ネイティブコードコンパイラを基礎とし、これまで以上にすぐれたパフォーマンスと大規模なメモリを利用できるようになります。加えて、新しいコンパイラのアーキテクチャは改良されたエラーチェックとモジュール管理によってプログラマが正しいコードを書く手助けをします。さらに、Delphi 2.0 は OLE コントロール(OCX)や OLE オートメーションを含む最新の OLE 技術の利点を活かし、Windows 95 や NT が持つオペレーティングシステムのすべての機能を利用できます。

図 1.　Delphi 2.0 の統合開発環境



この機能評価ガイドは、新しい 32 ビットの最適化ネイティブコードコンパイラの機能と利点を中心に紹介しています。Delphi 2.0 は、Windows 95 や Windows NT 上の特長を活かし、Windows 95 ユーザーインターフェースを完全にサポートし、多くのコンポーネントを備え、マルチスレッド、Unicode、MAPI などの最新の Windows 95 API のすべてをサポートします。Delphi 2.0 は、Windows 95 ロゴを取得する条件をすべて満たし、さらに開発者が作成するアプリケーションに対してもロゴを取得しやすいようにしています。

Delphi 2.0 は、以下の内容を目指して開発されました。

- 新しい 32 ビット最適化コンパイラによるパフォーマンスのさらなる向上
- 32 ビット Borland Database Engine によるデータベーススケーラビリティの向上
- OLE コントロール(OCX)や OLE オートメーションのサポートによる再利用性の向上
- Windows 95 ユーザーインターフェースや Win95/NT の API、32 ビット開発の完全なサポート

• 既存の Delphi アプリケーションとの高い互換性

今日の Delphi による開発

Delphi は、その高いパフォーマンスと生産性が評価され、さまざまな企業で活用されています。Delphi のユーザーには、 Alcatel, American Stores, Arthur Anderson, AT&T, BMW, BP Shipping, Bank of America, BBC Television, British Telecom, City of Los Angeles, Compaq, Conoco, Coopers & Lybrand, DHL, Dover Elevators, EDS, Ernst & Young, Fiat, First National Bank of Chicago, Glaxo, KPMG, Mercury Communications, Netscape, Sarah Lee Knitting, Standard & Poors, SwissBank SG Warburg, Union Bank, US Marine Corps といった企業が含まれます（海外）。

日本では、エアーニッポン(*1)、ユニシスソフトウェア、リオスコーポレーションなどで使われています。

- ある大手の石油化学会社は、全世界の 600 ユーザーを結ぶ１時間単位の処理および財務情報を提供するための経営システムを作成しました。
- ある民間の銀行は、世界中の主な金融センター間での安全なトランザクションによる 25 の通貨をサポートするグローバルな資金転送システムを作成しました。
- あるレンタル会社は、Delphi Client/Server(*2)を使って Windows NT 上の InterBase に対して全国の数カ所で 40 以上のユーザーが同時に接続する備品を追跡するアプリケーションを作成しました。
- 大手の繊物製造会社では、Delphi Client/Server(*2)を使って、Oracle 上の 1 憶件を超える収支、製造、販売情報を管理するシステムを開発しました。
- ある Internet のコンサルティンググループは、Delphi Client/Server(*2)を使って、代理店、小売業者、銀行へのアドバイスで使われる、World Wide Web 上のページを管理するためのアプリケーションを作成しました。
- ある技術書籍や雑誌を扱う出版社では、ソフトウェア出版を開始する際に Delphi を使って動画を使ったマルチメディアアプリケーションを作成しました。
- ある衛生機関は、Delphi を使ってラドンの放射レベルを測定する科学計器を通じてリアルタイムでデータを収拾するために、Paradox for Windows で使えるダイナミックリンクライブラリ(DLL)を作成しました。
- Fortune 100 に入るある機関では、Delphi Client/Server(*2)と MS-SQL Server を組み合わせることで、劇的な効率の改善と顧客満足を実現しています。

*1　詳細な資料をご希望の方は、弊社インフォメーションセンターまでお問い合わせください。
*2　Delphi Client/Server は、英語版のみ発売しています。

32 ビットコンパイラの新しいアーキテクチャ

Delphi 2.0 は、以下の新しい最適化ネイティブコードコンパイラの機能を含んでいます。

- C++と共通のバックエンド
- 最大 300〜400%のパフォーマンスの向上を実現する新しいコンパイラの最適化機能
- 高速化、最適化された新しいリンカ
- 実行時のインタープリタ DLL なしで、さらに 20〜25%の実行ファイルサイズの縮小
- C とのコードの共有化を容易にする OBJ ファイルのサポート

Delphi の新しい 32 ビット版は、すべて 32 ビットのネイティブコードコンパイラアーキテクチャをベースに構築されています。ボーランドは、10 年以上に渡るコンパイラの第一人者としての技術をこの製品に投入しました。新しいコンパイラは、実績のある Borland C++コンパイラと共通の"バックエンド"のコンパイラ技術を使って構築されています。

Delphi 2.0 は、32 ビット CPU の機能をフルに活用し、Delphi 1.0 に比べて 300〜400%もの実行速度の向上を達成しています。2〜999 までの素数の数を数える次の関数を 1 万回繰り返すために、Delphi 1.0 では 10.5 秒かかりますが、Delphi 2.0 では 2.5 秒で実行します(Pentium/133MHz)。

```
function SieveCount: Integer;
var
  flags: array [2..999] of Boolean;
  i, j: Longint;
begin
  for i := 2 to 999 do
    flags[i] := True;
  for i := 2 to 999 do begin
    j := i * i;
    while j <= 999 do begin
      flags[j] := false;
      Inc(j, i);
    end;
  end;
  Result := 0;
  for i := 2 to 999 do
    if Flags[i] then Inc(Result);
end;
```

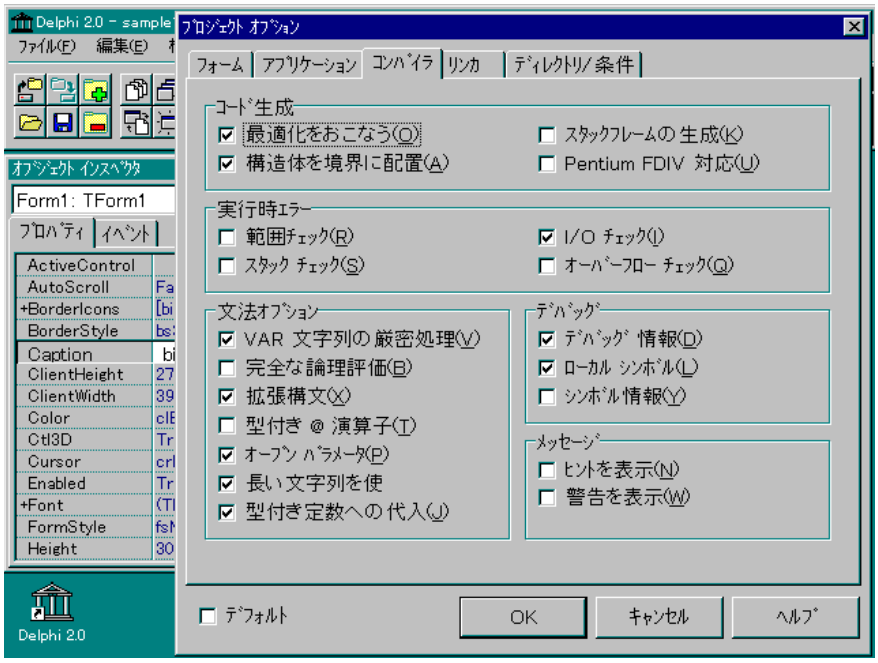
32 ビットコンパイラの最適化

新しい 32 ビットネイティブコードコンパイラは、いくつもの新しいコードの最適化技術を使ってパフォーマンスを向上させています。従来、最適化コンパイラは最高のパフォーマンスを引き出すために複雑なコンパイラ指令に対する経験が要求されました。さらに、ソースコードを何度も評価するために処理速度を低減させてしまい、迅速な開発の妨げにもなっていました。Delphi 2.0 のコンパイラは、推測に頼らない新しい最適化技法を自動的に適用し、毎分 35 万行(Pentium 使用時)という処理速度を維持しています。この結果、従来通りの迅速なアプリケーション開発と高いパフォーマンスという利点を得られます。

新しい 32 ビットネイティブコードコンパイラは、レジスタの最適化、コールスタックのオーバーヘッドの除去、共通部分式の除去、繰り返しにおける誘導変数を含む数多くの最適化を自動的に適用し、たいていのプログラムを高速化します。32 ビットコンパイラにおけるすべての最適化は、コードの本質的な意味を変化させないことが保障されています。

ベンチマークで比較するために最適化を禁止することもできます。さらに、Delphi 2.0 は"Pentium の FDIV 問題"に対応するコードも生成できます。

図 2.　Delphi 2.0 の最適化コンパイラのためのコンパイルオプション



レジスタによる最適化

頻繁に使われる変数や引数を自動的に CPU レジスタに配置することは、変数をアクセスする際に要求される機械語を減らすために非常に効果的です。メモリからレジスタに変数を読み込む必要もなくなるので、コードは高速かつ小さいものになります。この最適化は、どの変数や引数をレジスタに置き換えるかをいちいち指定しなくても、コンパイラが自動的に処理してくれます。さらに、コンパイラは変数の“ライフタイム”を解析して、レジスタを再利用します。たとえば、変数 I がコードのある部分で使われていて、変数 J がその後の部分で使われていれば、コンパイラは I と J に対してひとつのレジスタだけを使います。

関数引数のレジスタ化

可能であれば、関数や手続きに渡される引数も CPU レジスタに置き換えられます。このため、前述したメモリアクセスの手間がなくなるだけでなく、一時的に値を保持するための“スタックフレーム”を作成する必要もなくなります。関数呼び出しにおけるスタックフレームの作成・削除コードが除去でき、非常に効率的なコードが生成されます。

共通部分式の排除

コンパイラが複雑な算術式を評価する際に、何度も評価される共通部分式を一度だけで済ませます。これによって、プログラマーは共通部分式のためにコードを読みにくくする必要がなくなり、安心してわかりやすいコードを記述できます。

繰り返しにおける誘導変数

コンパイラは、繰り返しにおける配列や文字列の高速処理のために自動的に誘導変数を使います。たとえば、**for** 文の中で、変数が配列のためのインデックスのためだけに使われると、コンパイラは変数を“誘導”して、変数への操作を配列要素へアクセスするためにポインタに置き換えます。変数の大きさが 1、4、8 の場合には、CPU のスケールインデックス機能を使って効率化します。

最適化の例

以下のコードは、コンパイラのレジスタ変数と繰り返しにおける誘導を示したものです。式 Sums[X-1]は、繰り返し変数 X とともにインクリメントされるレジスタ誘導変数に置き換えられ、それぞれの繰り返しごとの計算しなくても済むようになっています。

```
program example1;
uses Windows;
var
  Sums : array [0..500] of Integer;
  X: Integer;
begin
  Sums[0] := 0;
  for X := 1 to High(Sums) do
    Sums[X] := Sums[X-1] + X;
  writeln(Sums[High(Sums)]);
end.
```

以下の生成コードは、Turbo Debugger を使って確認したものです。

```
ex1.9:  Sums[0] := 0;
    xor    eax,eax
    mov    [ex1.Sums],eax
ex1.10:  for X := 1 to High(Sums) do
    mov    edx,00000001      ; EDX = X, the loop control variable
    mov    eax,0040242C      ; EAX = the base memory address of Sums
ex1.11:  Sums[X] := Sums[X-1] + X;
    mov    ecx,[eax]         ; Get the value stored at Sums[X-1]
    add    ecx,edx           ; Add X to that value
    mov    [eax+04],ecx      ; Store the sum in Sums[X]
    inc    edx               ; increment the loop control variable
    add    eax,00000004      ; Increment the induction variable
    cmp    edx,000001F5      ; Test for loop end
    jne    ex1.11 (0040185D)
ex1.12:  writeln(Sums[High(Sums)]);
    mov    edx,[EX1.00402BFC]
    mov    eax,00402238
    call   @Write0Long
    call   @WriteLn
    call   @_IOTest
ex1.13: end.
```

新しい最適化リンカ

コンパイル処理において、Delphi はより高速に処理するいくつかの最適化を含む新しい 32 ビットのリンク技術を利用しています。新しいリンカは、新しいユニットキャッシュの手法を使って、20~50%もの高速化を実現しています。これは、アプリケーションを最初にコンパイルした後、変更されていないフォームやユニットをディスクではなく直接メモリ上でリンクするということです。さらに、.EXE は 20~25%も小さくなり、実行時のインタープリタ DLL は必要ありません。

リンカは、ユニット上で効率的なバージョンチェックを使い、できるだけユニットを再コンパイルしないようにしています。このため、ライブラリを更新するためにかかる時間は非常に短くなっています。たとえば、4 つの異なるフォームで使われているあるライブラリ関数が変更されたとします。従来は、すべてのフォームを再コンパイルする必要がありました。ディスク上にコンパイルされたバイナリ情報(DCU ファイル)が保存されており、そのユニットを利用するすべてのユニットを再コンパイルする必要があったからです。今回、改良されたリンク機能と組み込みバージョンチェックによって、DCU ファイルにコンパイルされたコードはより強固になり、再コンパイルする必要があるのは変更した関数があるユニットだけになりました。つまり、Delphi のバージョンが更新されても、サードパーティ製のライブラリを再コンパイルする必要はなくなる

ということです。

さらに、Delphi 2.0 は OBJ ファイル形式をサポートして、ダイナミックリンクライブラリ(DLL)を使えるという以上に、Delphi と C/C++間でのコードの共有を容易にしています。多くの市販の C ライブラリ関数は OBJ 形式で提供されています。このため、Delphi で利用できるサードパーティ製のライブラリが増えることになります(一般に処理系間でのオブジェクトの利用には制約があります)。

32 ビットによる容量の増大

新しい 32 ビットネイティブコードコンパイラは、従来の Windows 3.1 の 16 ビットセグメントアーキテクチャに依存する制約がなくなり、32 ビットのフラットアドレス空間上で実行されます。プログラマーは、Windows API の呼び出しに頼らずにマシンが持つすべてのメモリを活用できます。配列、文字列、レコード型などのデータ構造は OS の限界まで利用でき、たとえば Windows 95 上では 2GB までの文字列を使えます。これは、16 ビット版 Delphi での 64KB の制約を大幅に緩和するものです。

新しいデータ型やコンパイラの拡張

新しい 32 ビットネイティブコードコンパイラは、32 ビットのアドレス空間を活用し、柔軟性や 16 ビットコードとの移植性を持つ新しいデータ型を提供します。さらに、コンパイラ自身も大幅に拡張されています。

- OS のメモリだけに依存する長い文字列型のサポート
- アプリケーションの国際化のための Unicode 文字型に対応する WideChar 型
- データベースや OLE オートメーションに対応し実行時に型が変化するバリアント型
- 財務計算などで誤差が生じないようにするための通貨型
- DLL の関数をエクスポートしやすくするための予約語 stdcall
- マルチスレッドに対応するスレッド対応ライブラリ、スレッドローカル変数
- C++のような//による単一行コメント

OLE オートメーションと OLE コントロール(OCX)のサポート

Delphi 2.0 は、Windows 95 や NT 上での OLE の機能をサポートしています。

- OLE オートメーションのコントローラー・サーバーアプリケーションを作成できます。
- 将来のネットワーク OLE やリモートオートメーションと互換性があります。
- 既存のサードパーティ製の OLE コントロール(OCX)を利用できます。
- 継承によって OCX をカスタマイズできます。

OLE の重要性

マイクロソフトの OLE 技術は、開発者にモジュール性やアプリケーションの組み込み能力を高めるさまざまな重要な能力を含んでいます。ボーランドは、Delphi 2.0 は、たんにマイクロソフトの規格に単に適合させるだけではなく、完全なオブジェクト指向によって OLE 技術をより使いやすくすることも実現しています。

Delphi 2.0 の新しい OLE サポートは、OLE コントロール(OCX)を容易にインストールし、利用でき、OLE オートメーションのコントローラー・サーバーを容易に作成できるものです。Delphi 2.0 は、インプロセスとアウトプロセスの両方のサーバーを作成でき、最大限の柔軟性を提供しています。OLE オートメーションコントローラーとサーバーのサポートにより、Visual Basic 4.0 のリモートオートメーション技術や将来のネットワーク OLE 技術とも完全な互換性を保ち、さらに高速なパフォーマンスを提供します。Delphi 2.0 はネイティブコードコンパイラなので、Delphi のコンポーネントを作成するよりは難しいことですが、独自の OLE コントロール(OCX)を作成することも不可能ではありません。

OLE オートメーションの利用

Delphi 2.0 は、 **variant** と呼ばれる新しい型を使って OLE オートメーションのシームレスに組み込んでいます。Delphi 2.0 では、OLE オートメーションコントローラーとサーバーを容易に作成できます。たとえば、Delphi 2.0 のアプリケーションは、Word や Excel、Paradox など他の OLE アプリケーションを制御するために使えます。

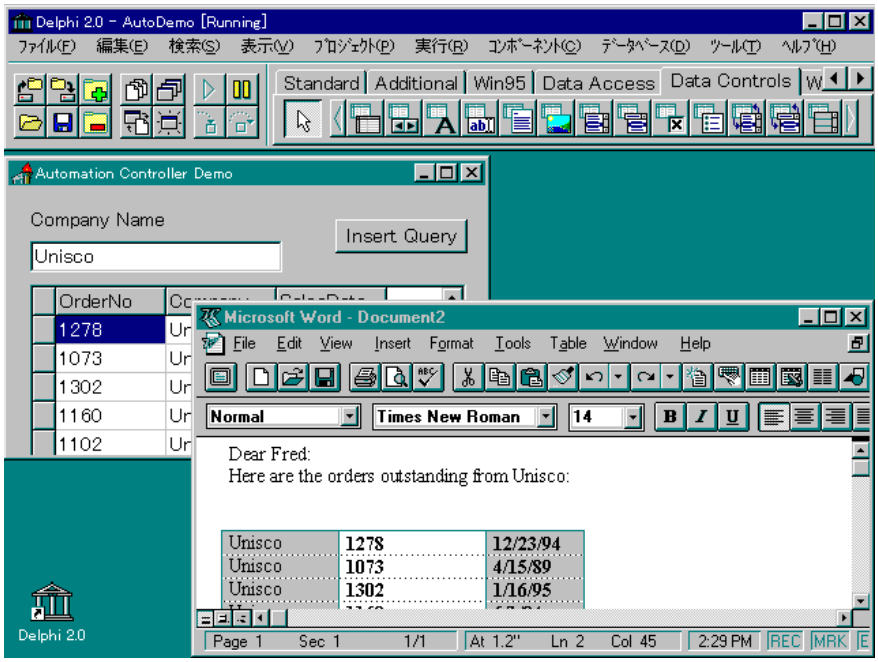
バリアント型を使えば、開発者は実行時に型が決定される変数を宣言でき、OLE オートメーションの柔軟性を活用できます。これにより、単一の変数を使って実行時に OLE オートメーションのさまざまな型を使うことができます。Delphi 2.0 は、OLE オートメーションサーバーを呼び出すときの名前付き引数もサポートしています。10 以上もの引数を持つ複雑な関数のために、開発者は必要な引数だけを指定することができます。

以下に示すコードは、Delphi 2.0 のアプリケーションで問い合わせを実行し、Word のドキュメントに挿入する例です。OLE オートメーションは、バリアント型の変数 MSWord の宣言を除き、4 行しか必要としていません。

```
procedure TForm1.InsertBtnClick(Sender: TObject);
var
  MSWord: Variant;
  S: string;
  L: Integer;
begin
  { MS-Word のオートメーションサーバーに接続し、問い合わせを実行します }
  MSWord := CreateOleObject('Word.Basic');
  with Query1 do
  begin
    Close;
    Params[0].Text := Edit1.Text;
    Open;
    try
```

```
        First;
        L := 0;
        while not EOF do
        begin
          S := S + Query1Company.AsString + ',' +
               Query1OrderNo.AsString + ',' + Query1SaleDate.AsString +
#13;
          Inc(L);
          Next;
        end;
        MSWord.Insert(S);
        MSWord.LineUp(L, 1);
        MSWord.TextToTable(ConvertFrom := 2, NumColumns := 3);
      finally
        Close;
      end;
    end;
  end;
```

図 3.　Delphi 2.0 を使った OLE オートメーションコントローラーとサーバー。

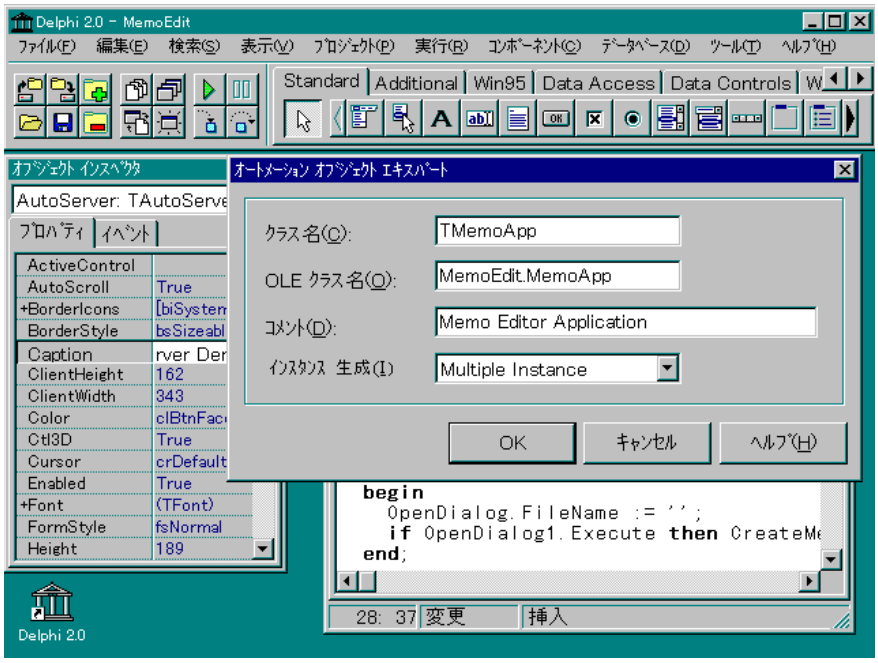


OLE オートメーションサーバーの作成

Delphi 2.0 は、独自の OLE オートメーションサーバーも作成でき、インプロセス・アウトプロセスの両方をサポートします。アプリケーションの関数やメソッドをエクスポーズできるため、MS-Word や Excel、Visual Basic、C++、Paradox、Delphi 2.0 などの他のアプリケーションから呼び出せます。Delphi 2.0 は、OLE オートメーションコントローラーとサーバーの両方をサポートでき、高度に最適化されたネイティブコードを生成するため、特にネットワーク OLE のような技術で重要なパフォーマンスを大きく向上させます。Delphi 2.0 の開発者は、クライアント側でもサーバー側でも高速にコンパイルされたアプリケーションを活用できます。

OLE オートメーションサーバーを新規に作成するために、[ｵｰﾄﾒｰｼｮﾝ ｵﾌﾞｼﾞｪｸﾄ ｴｷｽﾊﾟｰﾄ]が用意されています。このエキスパートは自動的に TAutoObject 型から新しいオブジェクトを継承し、プログラム ID、クラス ID、インスタンスオプションを含む OLE レジストレーションを準備します。

図 4.　OLE オートメーションサーバーも容易に作成できます。



その後、オブジェクトの **automated** セクションに自動化させたいプロパティやメソッドを定義します。**automated** セクションに記述された識別子の可視性は **public** 識別子と同じです。Smallint、Integer、Single、Double、WordBool、Boolean、Currency、TDateTime、String、Variant 型をプロパティ、引き数、関数の戻り値としてエクスポーズできます。

以下のプログラムは、Memo エディタを制御する OLE コントローラーアプリケーションに対応する **automated** セクションの使い方を示したものです。インプロセス(つまり DLL)とアウトプロセス(つまり EXE)のどちらのオートメーションサーバーも作成できます。

{ Delphi 2.0 での　OLE オートメーションサーバーの使い方 }

```
unit MemoAuto;

interface

uses
  OleAuto;

type
  { TMemoApp は、オートメーションサーバーオブジェクトと実装を定義する }
  TMemoApp = class(TAutoObject)
  private
    function GetMemo(Index: Integer): Variant;
    function GetMemoCount: Integer;
  automated
  { OLE は、以下のプロパティと関数を有効にする }
    procedure CascadeWindows;
    function NewMemo: Variant;
    function OpenMemo(const FileName: string): Variant;
    procedure TileWindows;
    property MemoCount: Integer read GetMemoCount;
    property Memos[Index: Integer]: Variant read GetMemo;
  end;
```

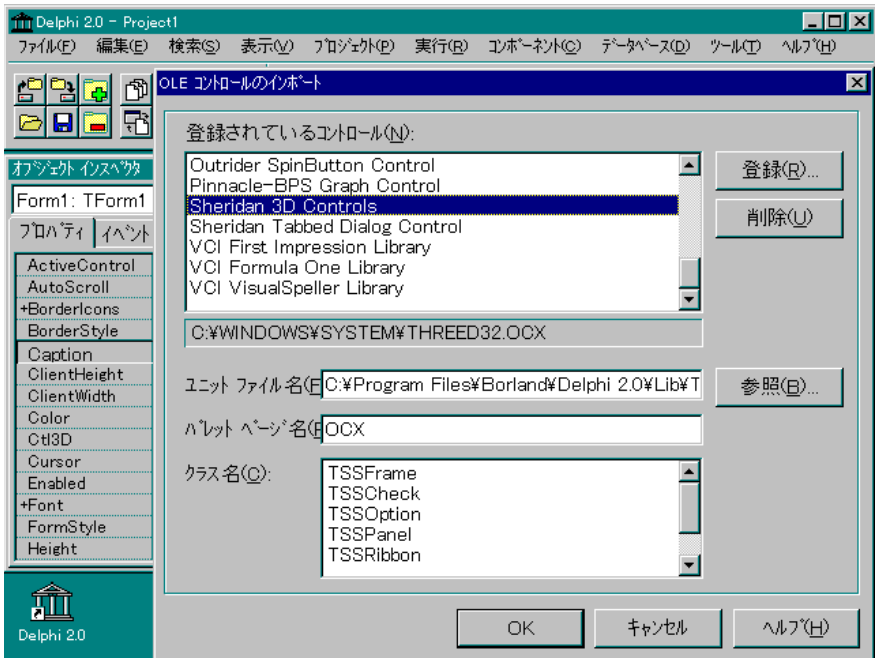
```
implementation

...

{ レジストレーション情報は、Automation Object Expert が自動的に作成する }
procedure RegisterMemoApp;
const
  AutoClassInfo: TAutoClassInfo = (
    AutoClass: TMemoApp;
    ProgID: 'MemoEdit.Application';
    ClassID: '{F7FF4880-200D-11CF-BD2F-0020AF0E5B81}';
    Description: 'Memo Editor Application';
    Instancing: acSingleInstance);
begin
  Automation.RegisterClass(AutoClassInfo);
end;

initialization
  RegisterMemoApp;
end.
```

OLE コントロール(OCX)の利用

Delphi 2.0 は、完全なオブジェクト指向環境であり、OLE コントロールをシームレスに組み込むことができます。サードパーティ製の OLE コントロール(OCX)を、Delphi で書かれたコンポーネントと同じようにインストールできます。Delphi 2.0 は、OLE システムのレジストリにもアクセスできるので、OLE コントロールを読み込み、単純なダイアログで登録することもできます。

OLE コントロール(OCX)をインストールするとき、Delphi は自動的に完全なオブジェクト指向を提供するオブジェクトラッパーを作成します。Delphi 2.0 だけが、OLE コントロールを完全にオブジェクト指向に対応させられる RAD 環境です。これは、開発者がどのコンポーネントも容易にサブクラス化し、継承によってカスタマイズできるということを表しています。

図 5.　Delphi のコンポーネントと同じように OLE コントロールを組み込めます。

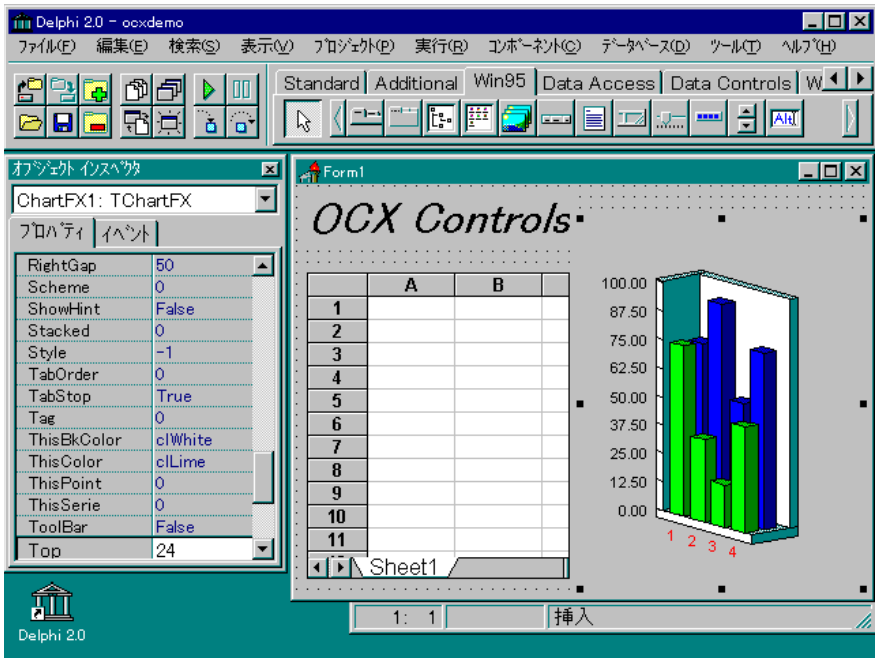


以下に、OLE コントロールをインストールするときに自動的に生成・コンパイルされるコードを示します。

```
{ どの OCX をインストールするときでも、Delphi は自動的にラッパーを生成します }
TGraph = class(TOleControl)
  private
    FOnHotHit: TGraphHotHit;
    function Get_Color(Index: Smallint): Smallint; stdcall;
    procedure Set_Color(Index: Smallint; Value: Smallint); stdcall;
    function Get_Data(Index: Smallint): Single; stdcall;
    procedure Set_Data(Index: Smallint; Value: Single); stdcall;
...
public
      property Color[Index: Smallint]: Smallint read Get_Color write
Set_Color;
       property Data[Index: Smallint]: Single read Get_Data write
Set_Data;
...
  published
    property TabOrder;
    property OnClick;
    property OnDblClick;
...
  end;
```

いったん、OLE コントロールが Delphi 2.0 のコンポーネントパレットに組み込まれると、Delphi のコンポーネントと同じように容易に利用できます。OLE コントロールのすべてのプロパティやイベントは、以下に示す通りオブジェクトインスペクタを使ってアクセスできます。

図 6.　Delphi 2.0 では、継承を使ってサードパーティ製の OLE をカスタマイズできます。



Windows 95 や NT の特長を活用する

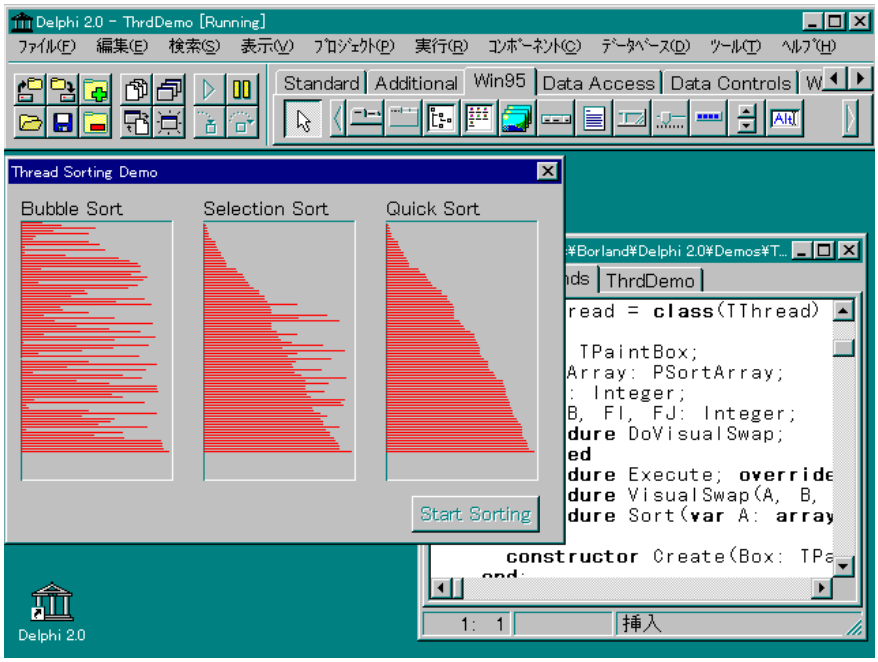
Delphi 2.0 は、ネイティブコードコンパイラなので、Windows 95 や NT が持っているプラットフォームとしての機能をすべてサポートします。これには、以下の機能が含まれます。

- マルチスレッドのサポート。
- ローカライズのための Unicode のダブルバイト文字列に対応。
- MAPI。

マルチスレッド

Delphi 2.0 はネイティブコードコンパイラなので、Windows 95 や Windows NT 上のプラットフォームの機能をすべて活用できます。これにはマルチスレッドのサポートが含まれます。どの API 関数も直接呼び出せるので、Delphi のアプリケーションで簡単にスレッドを作成したり優先順位を管理できます。たとえば、スレッドを作成するには実行させたい関数を引数として渡して CreateThread という API 関数を呼び出すことができます。さらに、Delphi には TThread というスレッド作成用のクラスが提供されています。Delphi の組み込みデバッガは、すべてのスレッドのステータスを表示できるため、マルチスレッドアプリケーションも容易にデバッグできます。

図 7.　Delphi 2.0 は、Windows 95 と NT のマルチスレッドをサポートします。



Delphi 2.0 は、スレッドローカルな変数を宣言するために ThreadVar という新しい予約語を導入し、異なるスレッドで使われる変数を完全に制御できます。Delphi の Visual Component Library (VCL)やライブラリルーチンも、マルチスレッドに対応しています。

正しいコードをプログラムするための支援

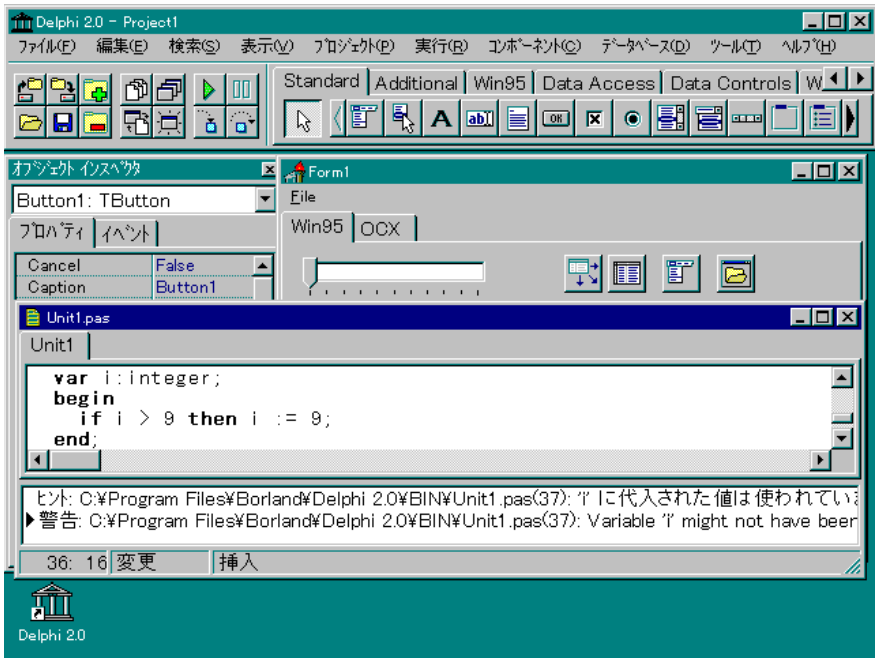
Delphi 2.0 のコンパイラには、正しいコードを記述するための機能が組み込まれています。

- 複数のエラーを検出。
- 一般的なコーディングミスを検出するヒントと警告。
- エラーメッセージの拡張。
- モジュール管理の効率化。
- 自動的なフォームのリンク。

コンパイラのエラーメッセージと診断メッセージの改良

見落としがちなネイティブコードコンパイラの利点のひとつに、プログラムを実行する前にすべてのコードが検査されるということがあります。コンパイラは、インタープリタでは検出できないような、あいまいさや不正なコードによる論理的な間違いを見つけやすくなります。Object Pascal 言語は強力に型付けされているので、間違ったスタイルで型を使おうとするような誤りも未然に防ぐことができます。新しい 32 ビットのコンパイラは“複数のエラー”を検出でき、多くのプログラムで正しいコードを記述しやすくなります。

図 8.　Delphi 2.0 は、改良されたエラーメッセージと診断メッセージを提供します。



Delphi 2.0 では、ヒントや警告を提供し、エラーメッセージやコンパイラの診断メッセージを改良しているため、次のような一般的なエラーを減らすことができます。

- 初期化されていない変数やポインタの使用。
- 使用されていない関数の戻り値。
- 空のループ。
- 型の不一致。

エラーメッセージの改良

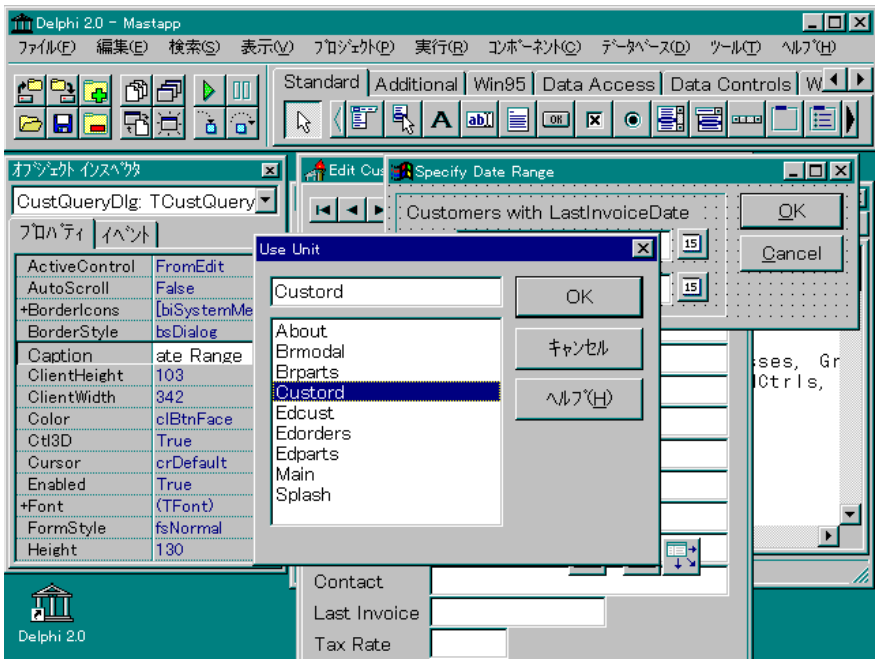
コンパイラは、文法エラーに対しても改良された診断メッセージを表示するので、新しく Object Pascal を覚える人にも Delphi をやさしく使えます。単に一般的な“構文にエラーがあります”というメッセージを表示するだけでなく、新しいコンパイラは問題を指し示すよりふさわしいメッセージを表示します。これには次のようなものがあります。

- セミコロンの不足
- else の直前にあるセミコロン

改良されたモジュール管理

Delphi は、大規模なプロジェクトの開発やテストをするために、ユニットと呼ばれるコードモジュールを分割コンパイルするための概念を使っています。ユニット間の厳密な検査を利用できる上、分割コンパイルにおいても外部参照のための膨大な宣言を必要としません。しかし、はじめて Delphi を使う人にとっては、いちいち **uses** 文を記述するのが面倒かもしれません。Delphi 2.0 では、[ﾌｧｲﾙ | ﾕﾆｯﾄを使う]コマンドとフォームリンクを使うことで自動的にモジュール管理ができます。

図 9.　Delphi 2.0 では、自動モジュール管理が実現されています。

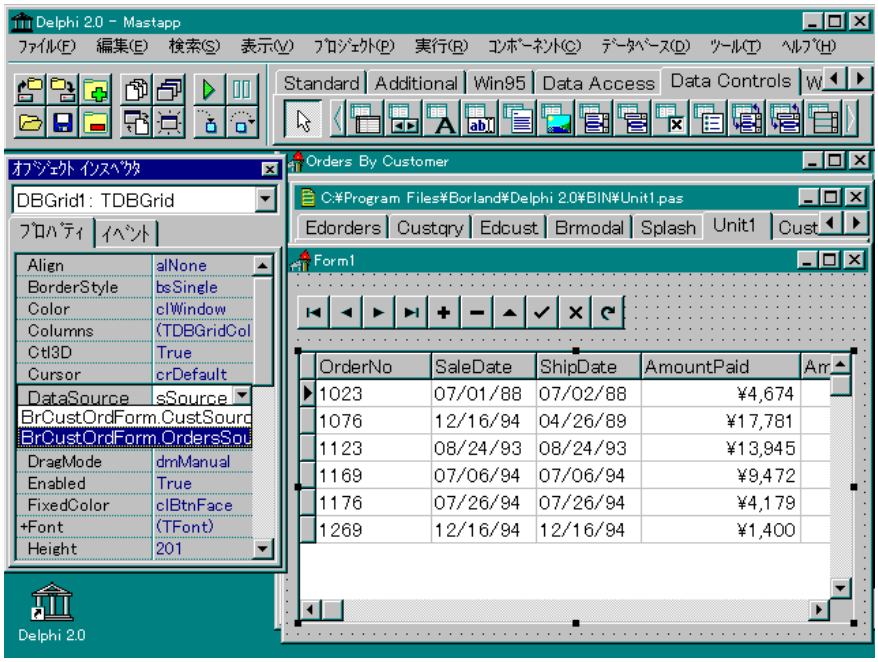


さらに、開発環境はよりインテリジェントになり、**uses** 文をいつ更新すべきかを認識するようになりました。たとえば、Form1 のユニットから Form2 を参照しようとすると、コンパイル時に **uses** 文に Unit2 を追加するかどうか問い合わせられます。

自動フォームリンク

次のステップは、異なるモジュール間のリンクを自動化して、プログラミングのモジュール化を容易にすることです。従来、異なるフォームにあるパブリックなオブジェクト、プロパティ、コードは、プログラムによって利用することはできましたが、設計時には利用できませんでした。たとえば、異なるフォームにあるデータソースや問い合わせ、テーブルを参照したいという要求がありました。このため、Delphi 2.0 では設計時にこれらのコンポーネントをリンクできるようになりました。ユーザーインターフェースコンポーネントから分離されているデータ処理やビジネスルールのカプセル化ができ、ますますモジュールの再利用が容易になります。

図 10.　フォームリンクを使えば、フォーム間でコンポーネントを共有できます。


Delphi 2.0 - Mastapp
ファイル(F) 編集(E) 検索(S) 表示(V) プロジェクト(P) 実行(R) コンポーネント(C) データベース(D) ツール(T) ヘルプ(H)
Standard Additional Win95 Data Access Data Controls
オブジェクト インスペクタ
DBGrid1: TDBGrid
プロパティ イベント
Align alNone
BorderStyle bsSingle
Color clWindow
Columns (TDBGridCol
Ctl3D True
Cursor crDefault
DataSource sSource
BrCustOrdForm.CustSourc
BrCustOrdForm.OrdersSou
DragMode dmManual
Enabled True
FixedColor clBtnFace
+Font (TFont)
Height 201
Orders By Customer
C:¥Program Files¥Borland¥Delphi 2.0¥BIN¥Unit1.pas
Edorders Custqry Edcust Brmodal Splash Unit1 Cust
Form1
OrderNo SaleDate ShipDate AmountPaid
1023 07/01/88 07/02/88 ¥4,674
1076 12/16/94 04/26/89 ¥17,781
1123 08/24/93 08/24/93 ¥13,945
1169 07/06/94 07/06/94 ¥9,472
1176 07/26/94 07/26/94 ¥4,179
1269 12/16/94 12/16/94 ¥1,400
Delphi 2.0

ビジュアルなフォームの継承

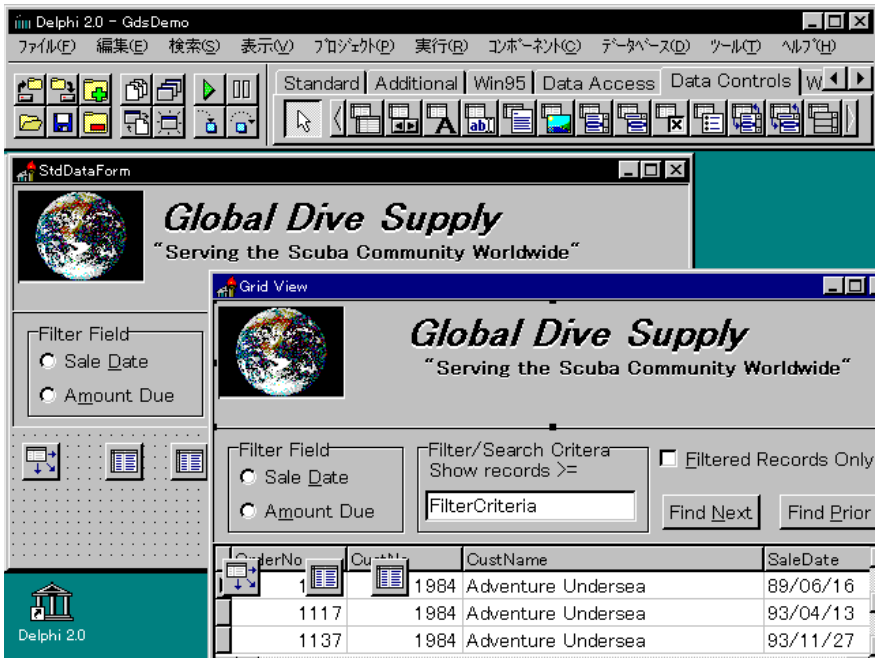
Delphi 2.0 は、ビジュアルなフォームの継承によってオブジェクト指向プログラミングへの対応を強化します。

Delphi のもっとも重要な能力のひとつに、オブジェクト指向プログラミングの完全なサポートがあります。Object Pascal 言語がカプセル化、多態性、継承を完全にサポートしていることで、Delphi は、開発者が Delphi の中だけで独自のオブジェクトを作成したり、既存のコンポーネントをサブクラス化したり、まったく新しい抽象ビジネスオブジェクトを表現するユニークな機能を持っています。Delphi は、Delphi 自身で開発されているため、開発者やサードパーティが作成するオブジェクトは、ボーランドが作成したものと同等のものです。実際、多くのサードパーティベンダーや Delphi の開発者が、こうしたコンポーネント市場に注目しています。

Delphi 2.0 は、継承の機能をよりやさしく使いやすいものにするためビジュアル環境の基本的な OOP の能力を活かし、強化しました。このバージョンでは、設計環境においてコードを書くことなくフォームからビジュアルに継承でき、変更がただちに反映されます。たとえば、多くの企業では、データ入力などで他のフォームの基礎として使われる標準的な“テンプレート”フォームを作成しています。ビジュアルなフォームの継承を使えば、標準フォームの変更が継承した他のフォームに直ちに反映されます。

ビジュアルなフォームの継承を使えば、実行時のパフォーマンスを損なうことなく、すべてのコードやオブジェクト、プロパティを複数のレベルで継承できます。同様の機能を提供しようとした他のシステムにはパフォーマンスを著しく低下させるものもあり、実用に向いていません。

図 11.　フォームの継承によって、再利用可能な共有フォームを容易に作成できます。



ビジュアルなフォームの継承の例

サンプルアプリケーション\DEMOS\DB\GDSDEMO\GDSDEMO.DPR は、ビジュアルなフォームの継承の例が含まれています。これは、ユーザーに表示するための 2 つのフォームを含んでいます。これらのフォーム GridViewForm と RecViewForm は、共通のデータセットを元にしてそれぞれ Grid ビューと Record ビューを表示しています。実際、これらの 2 つのフォームは多くの共通要素を持ち、上位フォーム型 StdDataForm から共通要素を継承しています。StdDataForm は、Customer と Orders テーブル、データソースを定義し、Orders テーブルのためのフィルタとフィルタに一致

する Orders の前後に移動するためのコードが記述されています。StdDataForm には多くの機能が定義されているため、RecViewForm にはほとんどコードが書かれていませんし、GridViewForm にはまったくコードが含まれません。

ビジュアルなフォームの継承によって、StdDataForm に対する変更は、コードであるかオブジェクトインスペクタやフォームデザイナによるビジュアルによるかに関係なく、自動的に下位の GridViewForm や RecViewForm に反映されます。たとえば、StdDataForm の Find Next ボタンを移動したりそのコードを変更すれば、直ちに両方の下位フォームに変更が反映されます。もちろん、GridViewForm に継承されたコンポーネントのビジュアルなプロパティやコードは自由にオーバーライドできます。

Delphi のビジュアルなフォームの継承は、継承するかオーバーライドするかをプロパティ単位のきめ細かさで利用できます。つまり、GridViewForm の Find Next ボタンの位置と Font プロパティだけを変更できるということです。キャプションや大きさ、OnClick イベントハンドラのコードなどはそのまま継承されます。

もし、すべての上位コンポーネントのプロパティに復帰させたい場合は、下位フォームのコンポーネントで右ボタンをクリックし、メニューから[継承元の値に戻す]を選ぶだけです。これは、すべてのプロパティに対する変更を無効にします。

継承したイベントハンドラの呼び出し

下位フォームのコンポーネントの動作を追加したりオーバーライドしたい場合でも、オブジェクトインスペクタを使って他のコンポーネントのイベントハンドラを結びつけることができます。たとえば、GridViewForm の FindNext ボタンの OnClick イベントハンドラをオーバーライドするためには、Find Next ボタンをダブルクリックするか、オブジェクトインスペクタで OnClick プロパティをダブルクリックするだけです。生成されるメソッドは、以下のように上位で定義されたメソッドの呼び出しを含んでいます。

```
procedure TGridViewForm.NextBtnClick(Sender: TObject);
begin
  inherited;
  { ここにコードを追加する }
end;
```

注意: コードのメンテナンスを容易にするためには、常に継承したルーチンを呼び出すべきです。これがないと、呼び出しは無視されてしまいます。

オブジェクトリポジトリから新しい継承フォームを作成する

新しい継承フォームを作成するためには、[ﾌｧｲﾙ | 新規作成]メニューコマンドを使い、オブジェクトリポジトリを呼び出します。すると、現在のプロジェクトのページ(GdsDemo など)や[ﾌｫｰﾑ]、[ﾀﾞｲｱﾛｸﾞ]、[ﾃﾞｰﾀﾓｼﾞｭｰﾙ]などのページから適当なフォームを選べます。なお、現在のプロジェクトのフォームでは、[継承]オプションだけが使えます。また、[ﾌｫｰﾑ]、[ﾀﾞｲｱﾛｸﾞ]、[ﾃﾞｰﾀﾓｼﾞｭｰﾙ]ページでは[ｺﾋﾟｰ]や[直接使用]オプションを選ぶこともできます。

どのようにフォームの継承が実装されているか

フォームから継承する際、現在のプロジェクトやオブジェクトリポジトリのどちらからでも上位クラスをメモリに読み込みます。コードエディタで継承したフォームの定義を見ると、継承したフォームは TForm の代わりに他のフォーム型から継承されていることがわかるでしょう。たとえば、GridViewForm の定義は以下のようになっています。

```
type
  TGridViewForm = class(TStdDataForm)
    DBGrid1: TDBGrid;
    procedure NextBtnEndDrag(Sender, Target: TObject; X, Y: Integer);
    procedure NextBtnClick(Sender: TObject);
```

```
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

ここで、DBGrid コントロールの定義が追加されていることがわかります。ボタンやテーブルなど GridViewForm の他のコンポーネントは、すべていかに示す上位型 TStdDataForm で定義されています。

```
TStdDataForm = class(TGDSStdForm)
    StdCtrlPanel: TPanel;
    FilterOnRadioGroup: TRadioGroup;
    Orders: TTable;
    Cust: TTable;
    OrdersSource: TDataSource;
    GroupBox1: TGroupBox;
    FilterOnLabel: TLabel;
    FilterCriteria: TEdit;
    FilterCheckBox: TCheckBox;
    NextBtn: TButton;
    PriorBtn: TButton;
...
    procedure FilterOnRadioGroupClick(Sender: TObject);
    procedure OrdersCalcFields(DataSet: TDataSet);
    procedure FilterCheckBoxClick(Sender: TObject);
    procedure PriorBtnClick(Sender: TObject);
    procedure NextBtnClick(Sender: TObject);
    procedure FilterCriteriaExit(Sender: TObject);
    procedure FilterCriteriaKeyPress(Sender: TObject; var Key: Char);
  protected
    FLastAmount: Double;
    FLastDate: TDateTime;
    function CalcAmountDue: Double;
    procedure ConvertFilterCriteria;
  end;
```

同様に、フォームデザイナで GridViewForm を右クリックし[ﾃｷｽﾄとして表示]を選ぶと、上位クラスから変更されたプロパティだけが表示されます。以下の例は、GridViewForm の Find Next ボタンが異なるフォントと位置になっていると想定した場合のテキスト表現です。

```
inherited GridViewForm: TGridViewForm
  Caption = 'Grid View'
  inherited StdCtrlPanel: TPanel
    inherited NextBtn: TButton
      Left = 223
      Top = 13
      Font.Height = -13
      Font.Style = [fsBold]
      ParentFont = False
    end
  end
  object DBGrid1: TDBGrid [2]
    Left = 0
```

```
      Top = 161
      Width = 460
      Height = 159
      Align = alClient
      DataSource = OrdersSource
      TabOrder = 2
      TitleFont.Color = clBlack
      TitleFont.Height = -11
      TitleFont.Name = 'MS Sans Serif'
      TitleFont.Style = []
    end
  end
```

ビジュアルなフォームの継承は、上位フォームと下位フォームのコンポーネントと、そのコンポーネントの差に基づいて設定されたプロパティをストリーム出力することで実現されています。コンポーネントの作成者は、ビジュアルなフォームの継承の差分解席ができるだけ有効に働くよう、コンポーネントが必要なプロパティだけを出力するようにしなければなりません。コンポーネントによっては、下位フォームでのプロパティの変更が大きなものになることがあります。たとえば、データベースグリッドコントロールは、カラム属性を保持する Collection プロパティを使います。このため、下位フォームの Collection プロパティを変更すると、上位の Collection は全体がオーバーライドされてしまいます。これは、継承したフォームの下位グリッドのカラム属性を変更すると、上位のカラム属性の変更が下位に伝えられなくなることを意味します。つまり、カラム属性の細分化は、個々のカラムではなくカラム属性全体にとどまるということです。しかし、Font や Color などの上位グリッドの他のプロパティに対する変更は下位に伝えられるため、柔軟性と効率性のすぐれたバランスが提供されます。

注意: ビジュアルなフォームの継承であまり深いレベルを使うと、遅いマシンでは設計環境でのサイズの変更や移動に対する複雑なフォームの再描画が若干遅くなってしまうことがあります。この設計環境におけるパフォーマンスの低下は、Delphi が上位と下位のプロパティの差を計算するためです。コンパイル処理によって、すべての情報はプロパティの静的な設定にコンパイルされフォームに読み込まれるため、実行時のパフォーマンスには影響しません。

ネイティブコードコンパイラ

Delphi は、別の C コードコンパイラでコンパイルするための C コードジェネレータを持つ 4GL システムと異なり、ただちに最適化されたネイティブなマシンコードにコンパイルします。このため、4GL システムのパフォーマンスの欠落を完全に補うばかりでなく、Delphi が常に最適化されたネイティブなマシンコードを直接生成することで、アプリケーションの開発を効率化し、信頼性を高める特長があります。Delphi は、常に最適化されたマシンコードを生成するため、2 段階の“コードジェネレータ”を上回る数多くの利点があります。この利点は次のようなものです。

- **ターンアラウンドの高速化** -- 世界最高速のネイティブコードコンパイラによって、Delphi は生産性を高め、迅速なアプリケーション開発を推進します。2 段階の“コードジェネレータ”と違い、コードジェネレータによる生成を待ったり、生成・コンパイル・リンクのサイクルに我慢する必要はありません。
- **容易なテスト** -- 開発環境で実行されるコードは、生成したアプリケーションを配布するときのコンパイルされたコードと同じものです。つまり、インタープリタによる実行がコンパイラで生成したコードと同じように動作しているだろうかと心配する必要ありません。
- **高度なデバッグ** -- コードがマシンコードに直接コンパイルされるため、環境の中でデバッガは書かれたコードそのものをデバッグできます。システムによって、C に暗号化されたプログラムを見る必要はなく、コードジェネレータが生成した C コードをデバッグしなくてもよいのです。
- **容易な保守** -- 単一の高級言語 Object Pascal だけを保守すればよく、4GL 言語と生成された C 言語の両方を保守する必要はありません。
- **容易な配布** -- アプリケーションを配布する際に、実行時のインタープリタ DLL は必要ありません。

歴史的に、中間コード (P-Code)システムとコードジェネレータという組み合わせは、そのギャップを証明してくれます。たとえば、Apple II コンピュータ上の初期の言語の実装は、UCSD の中間コードシステムによって実装され、これによって 64KB 以下のマシンで Basic の代わりに Pascal のような高級言語が使えるようになったのです。しかし、時が過ぎ中間コードシステムによるパフォーマンスの低下が嫌われるようになりました。

同様に、初期の C++言語の処理系は、C++コードを標準的な C コンパイラでコンパイルできるよう C コードに変換するソースコードトランスレータによるものがほとんどでした。しかし、2 段階のコンパイル処理による遅いターンアラウンドと生成された C コードのデバッグの複雑さの欠点から、多くの C++開発者が“ネイティブコード”の C++コンパイラに移行しました。

16 ビットコードとの互換性

Delphi 2.0 は、16 ビットコードとの完全な互換性を提供することも重視しています。Delphi では、16 ビット版で書かれたコードを 32 ビット版でもそのまま再コンパイルできるか、必要でも最低限の変更ですむようになっています。たいていの場合、開発者は単に 16 ビットアプリケーションを新しい環境で読み込み、32 ビットでコンパイルするだけで、32 ビットの能力が利用できます。場合によってはコードの変換が必要ですが、これは Windows 自身の変更によるものや、整数型が 16 ビットから 32 ビットになったというような物理的な実装の違いによるものです。Delphi は、(メッセージクラッキングと呼ばれる) Windows のメッセージ型の変更を処理するので、メッセージ処理コードは、たとえデータフィールドが 32 ビットに変わっているものでも変更する必要はありません。変更しなければならないコードは、次のようなものです。

- 16 ビットのインラインアセンブラを使っているもの
- Win32 で変更された Windows API を使っているもの
- 整数型の物理的なサイズに依存したレコードやルーチン

Delphi 2.0 では、整数型は従来の 16 ビットではなく 32 ビットになります。互換性のために、新しく Smallint という型が提供されます。

新しい 32 ビットのコンパイラは、“ユニットエリアス”の機能を持っていて、ユニットのために異なるシンボル名(エリアス)を使うことができます。この便利なテクニックは、新しい 32 ビットの機能の特長を利用するためにユニットの実装を変更させるために使われています。たとえば、Delphi の 16 ビット版では WinTypes と WinProcs の 2 つに分かれていました。これらは、32 ビット版では 32 ビット Windows の型や関数を含む Windows というひとつのユニットにまとめられています。完全な互換性を提供するためにユニットエリアスが提供され、古いコードをコンパイルするときに uses 文を変更する必要はありません。たとえば、次の文は、

```
Uses WinTypes, WinProcs;
```

次のように理解されるエリアスになります。

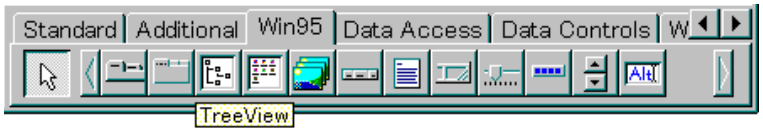
```
Uses Windows;
```

Delphi 2.0 で書かれたアプリケーションは、32 ビットの機能を使っていなければ 16 ビット版の Delphi で再コンパイルして Windows 3.1 のアプリケーションを作成できます。しかし、プログラムが 32 ビットの機能や Windows 95 のユーザーインターフェース要素、Win32 API 関数、32 ビットのフラットアドレス空間、他の機能などを利用していれば、16 ビットでコンパイルするために修正が必要です。

Windows 95 のコモンコントロール

Delphi 2.0 のコンポーネントパレットには、Windows 95 コントロール対応の 12 種類のコンポーネントが用意され、Windows 95 ロゴ準拠のアプリケーションを容易に設計できます。

図 12.　Delphi 2.0 のコンポーネントパレットに登録されている Windows 95 コントロール

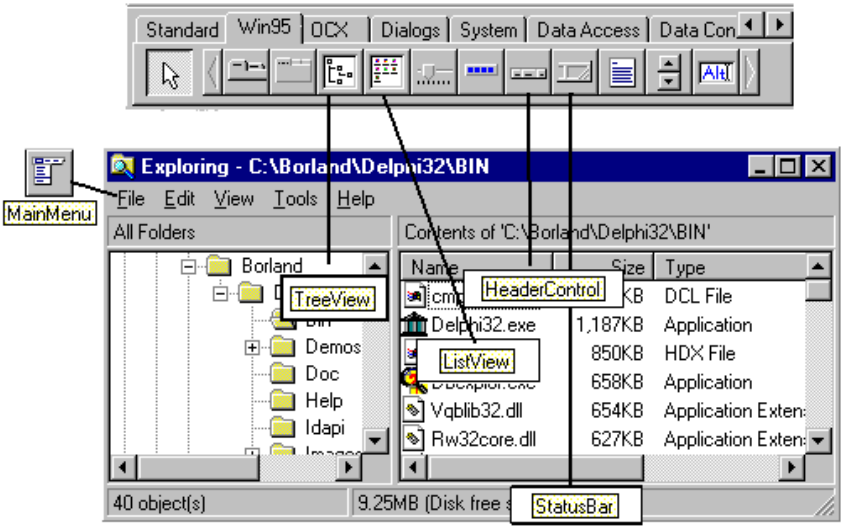


Delphi 2.0 が提供する Windows 95 コントロールは以下のとおりです。

- TabbedControl - TTabSet と同様のタブセット
- PageControl - マルチページダイアログを作成するためのページコントロール
- TreeView - オブジェクトのリストをインデント付きのアウトラインで表示します
- ListView - リストをカラム表示します
- ImageList - ひとまとまりのグラフィックイメージを管理します
- HeaderControl - THeader と同様の幅を変更できるヘッダ
- RichEdit -書式付きで編集できるリッチテキストフォーマットエディタ
- StatusBar - スクリーンの下端の領域に動作状態を示すコントロール
- TrackBar - 段階付きでスライドできるコントロール
- ProgressBar - TGauge と同様に進行状態を表示します
- UpDown - 値の増減を制御する矢印付きのボタン
- HotKey - コンポーネントへホットキーを割り当てるコントロール

Windows 95 スタイルのコントロールを使っている典型的な例は、エクスプローラです。Delphi のコンポーネントを使えば、このようなアプリケーションも簡単に作成できます。

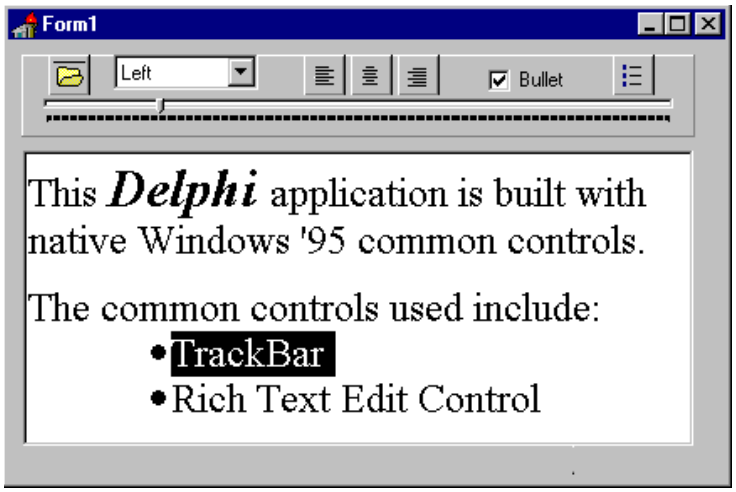
図 13.　エクスプローラの要素と Delphi のコンポーネントとの対応



TRichEdit（リッチテキスト）、書式付きメモ

図 14 は、リッチテキストとトラックバーの 2 つのコントロールを使っています。

図 14.　リッチテキストエディットとトラックバーの使用例



リッチテキストコントロールのプロパティは以下のとおりです。

DefaultText プロパティ	デフォルトテキストの属性を指定します。特に指定されていないテキストのための属性になります。DefaultText は TTextAttributes 型で、SelectedText プロパティの項目で説明します。
HideScrollBars プロパティ	選択していない場合には、スクロールバーは表示されません。
Lines プロパティ	テキストをあらわす文字列リストです。
Paragraph プロパティ	選択されたテキストのパラグラフ（段落）フォーマットです。パラグラフは TParaAttributes 型で、Alignment、FirstIndent、Numbering、Tab 配列などの独自のプロパティを持っています。
PlainText プロパティ	テキストを単純なテキストかリッチテキスト形式のどちらで処理するかを指示する論理型のプロパティです。
Print メソッド	TRichEdit の内容を印刷します。
OnResizeRequest イベント	テキストが現在の境界を超えて増大したか、resize イベントが発生していることを示します。境界のために必要な大きさを知るために、TRect 型の Rect 引数を受け取ることができます。
OnSelectionChange イベント	異なるテキストが選ばれたことを示します。

TTrackBar（トラックバー）、値を調節する

さきほどの例で、リッチテキストのインデント位置を設定するために使われているトラックバーは、アプリケーションの中で決められた範囲で値を設定したり、調節するために使われます。トラックバーは、範囲の定義と調節バーで構成されるコントロールです。トラックバーには、多くのオプションがあり、トラックバーの方向（垂直か水平）、インジケーターの種類、コントロールの目盛りの有無などを決めることができます。

ユーザーは、特定の位置にスライドインジケーターを移動したり、バーのホットゾーンをクリックすることで、インジケーターを移動できます。

トラックバーのプロパティは、以下のとおりです。

LineSize プロパティ	矢印キーを使って移動する目盛りの数

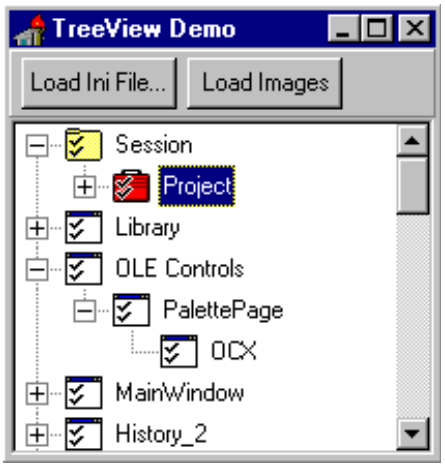
Max プロパティ	範囲の最大値
Min プロパティ	範囲の最小値
Orientation プロパティ	トラックバーの方向。水平(Horizontal)か垂直(Vertical)。
PageSize プロパティ	PageUp/PageDown キーを使って移動する目盛りの数。
Position プロパティ	トラックバーの現在値。
SelEnd プロパティ	選択中の範囲の終了点。
SelStart プロパティ	選択中の範囲の開始点。
TickMarks プロパティ	目盛りの位置。
TickStyle プロパティ	目盛りのスタイル（自動描画、手動描画）。
OnChange イベント	トラックバーが移動したことを示します。

TTreeView（ツリービュー）、情報をアウトラインで表示する

ツリービューコントロールは、一組のオブジェクトの論理的な階層関係をインデント付きのアウトラインで表示する特殊なリストボックスコントロールです。コントロールは、アウトラインを開いたり、閉じたりするボタンを持ちます。

それぞれの項目のテキストラベルに対してアイコンを表示することもできます。アプリケーションのユーザーがツリーの項目を開いたり閉じたりする際には、異なるアイコンが表示されます。さらに、項目の階層情報を反映するためにチェックボックスのようなグラフィックスも使えます。リストとボタンの間には階層関係を示すための線を描画できます。

図 16.　ツリービューの例



ツリービューは階層構造を持ち、ノードの集まりとして構成されます。ツリーの各ノードはプロパティを持ち、それぞれがイベントに応答できます。ノードは、次のような 3 種類のプロパティを持っています。

- ノードの状態を検査する（選択中であるか、広がっているか、など）
- ノードに関する情報収集（子ノードを持っているか、など）
- 第一の子ノード、最後の子ノード、ツリーの最上位などに新しいノードを割り当てる

ツリービューには、この他に以下のようなプロパティがあります。

Count プロパティ	ツリーの中の項目数。
Editable プロパティ	項目テキストをユーザーが（テキストをダブルクリックして）修正できるかどうかを示します。
Indent プロパティ	ノードとその親との間のピクセル数。
Item プロパティ	ツリービューの項目を返します。
Items プロパティ	ツリービューのすべての項目に対するプロパティ。
ShowButtons プロパティ	項目を開けるときに[+]ボタンを表示します。
ShowLines プロパティ	ノードを接続する線を表示します。
ShowRoot プロパティ	ルート（トップレベル）の項目を接続している線を表示します。
Sorted プロパティ	項目をアルファベット順にソートします。
TopItem プロパティ	ツリーの最初の項目を返します。
AddChildFirst メソッド	子項目を最初の項目として追加。
AddChildObjectFirst メソッド	子項目と関連付けられたオブジェクトを最初の項目として追加。
AddFirst メソッド	最初の項目を追加。
AddObject メソッド	項目と関連付けられたオブジェクトを追加。
AddObjectFirst メソッド	項目と関連付けられたオブジェクトを、最初の項目として追加。
Append メソッド	ツリーの最後に項目を追加。
AppendChild メソッド	最後の子として子項目を追加。
AppendChildObject メソッド	子項目を関連付けられたオブジェクトとともに最後の子として追加。
AppendObject メソッド	項目と関連付けられたオブジェクトを最後の子として追加。
FullCollapse メソッド	すべてのノードを閉じます。
FullExpand メソッド	すべてのノードを開きます。
GetFirstNode メソッド	最初の項目ノードを返します。
Insert メソッド	項目をツリーに挿入します。
InsertObject メソッド	項目を関連付けられたオブジェクトとともにツリーに挿入します。
SortChildren メソッド	項目の子項目をアルファベット順にソートします。
OnChange イベント	ツリービューが変更されたことを示します。
OnChanging イベント	ツリービューが変更されるとき、変更される前に発生します。
OnCollapse イベント	子項目を閉じたことを示します。
OnCollapsing イベント	子項目を閉じるときに、閉じる前に発生します。
OnDeletion イベント	項目が削除されるときに発生します。
OnEdited イベント	項目が編集されたことを示します。

OnEditing イベント	項目が編集モードに入るときに発生します。
OnExpanded イベント	子項目が開いたことを示します。
OnExpanding イベント	子項目を開くときに、開く前に発生します。

TProgressBar（プログレスバー）、進行状況を表示する

プログレスバーは、時間のかかる処理の進行状況を表示するためのコントロールです。これは、左から右に埋められていく長方形のバーです。Windows 95 では、エクスプローラでファイルをコピーするときに使われています。進行状況をあらわすインジケータは、情報を表示するだけであり、対話することはありません。

プログレスバーのプロパティは、以下のとおりです。

Max プロパティ	範囲の最大値。
Min プロパティ	範囲の最小値。
Position プロパティ	進行状況を示す位置。
Step プロパティ	StepIt メソッドで Position に加える値。
StepBy メソッド	指定された値を Position に加えます。
StepIt メソッド	Step プロパティで指定された値を、Position に加えます。

THeaderControl（ヘッダー）、移動可能なヘッダー

ヘッダーコントロールは、Windows 95 のエクスプローラで使われています。ヘッダーコントロールは、テキストや数値を表示する列の上に表示できます。複数の項目を表示するために、ヘッダーをいくつかの部分にわけることもできます。このとき、タイトルの要素を左寄せ、右寄せ、中央に合わせて表示できます。ユーザーが、この部分をクリックすれば特定の機能をサポートするボタンと同じく、それぞれの部分を再構成し表示を変更できます。エクスプローラで[表示 | 詳細]を選べば、[名前]、[ｻｲｽﾞ]、[ﾌｧｲﾙの種類]、[更新日時]をクリックしてソートのテストができます。

ヘッダーコントロールでは、ユーザーがセクションの幅を設定するためヘッダの分割位置をドラッグできます。

SectionCount プロパティ	セクションの数を示します。
Sections プロパティ	それぞれのセクションを指定します。
OnSectionResize イベント	セクションの幅が変更されるときに発生します。
OnSectionResized イベント	セクションの幅が変更された後に発生します。

TUpDown（アップダウン）、矢印ボタン

アップダウンコントロールは、指定された範囲の値を設定するためのテキストを制御します。アップダウンコントロールは、テキストボックスと組み合わせる一対のボタンです。ユーザーがテキストボックスやボタンをクリックすると、入力フォーカスがテキストボックスに設定されます。ここで、直接テキストを入力したり、ボタンを使って値を増減させることができます。変化の大きさは、設定した値に応じて変わります。

テキストボックスで時間、分、秒などを編集するために、スピンボタンボックスだけを利用することもできます。ボタンはフォーカスのあるテキストボックスにだけ影響します。

Associate プロパティ	アタッチするコントロールを示します。

Max プロパティ	範囲の最大値。
Min プロパティ	範囲の最小値。
Orientation プロパティ	表示する方向（垂直または水平）。
Position プロパティ	調整される数値。
Wrap プロパティ	値を循環させるかどうかを示します。True の場合、値が Max で上向きボタンをクリックすると Min に設定された値にリセットされます。
OnChanging イベント	上下の矢印ボタンをクリックして値が変更されるときに発生します。

TListView（リストビュー）、項目をリスト表示

リストビューは、データを様々なビューの中で表示します。Windows 95 のエクスプローラは、[表示]メニューから[大きいアイコン]、[小さいアイコン]、[一覧]、[詳細]を選ぶことで、リスト表示されている内容の形式をすばやく切り替えます。

リストビューのプロパティは、以下のとおりです。

Columns プロパティ	TListColumn オブジェクトのリスト
EditLabels プロパティ	項目ラベルを編集できるかどうかを示します。
IconArrangement プロパティ	アイコンの位置を示します。
Items プロパティ	TListItem オブジェクトのリスト。
ShareImageLists プロパティ	イメージリストの解放を防ぎます。
ViewStyle プロパティ	項目をビジュアルに表示します。
Arrange メソッド	項目を再配置します。
Scroll メソッド	リストビューウィンドウをスクロールします。
Sort メソッド	デフォルトまたは独自のソートを認めます。

TTabControl（タブコントロール）、タブセット

TTabControl は、TTabSet に似ています。マルチページダイアログボックスをつくるためには TPageControl を使い、タブセットを使いたい場合には TTabControl を使います。

タブコントロールのプロパティは、以下のとおりです。

Multiline プロパティ	タブに複数行を表示するかどうかを示します。
TabHeight プロパティ	タブの高さをピクセル値であらわします。
TabIndex プロパティ	選ばれたタブを指定します。最初のタブの TabIndex の値は 0 になります。
Tabs プロパティ	タブのテキストを指定する TStrings オブジェクトです。
TabWidth プロパティ	タブの幅をピクセル値であらわします。
OnChange イベント	新しいタブが選択され、フォーカスが新しいタブに移動した後で発生します。

OnChanging イベント	新しいタブが選択され、フォーカスが新しいタブに移動する前に発生します。

TPageControl（ページコントロール）、マルチページダイアログボックス

ページコントロールは、同じウィンドウの中で論理的な複数のページや情報のセクションを定義するために使います。

ページコントロールのプロパティは、以下のとおりです。

ActivePage プロパティ	アクティブな TTabSheet オブジェクトを指定します。
Multiline プロパティ	複数行を認めます。
PageCount プロパティ	ページの数を示します。
Pages プロパティ	インデックスを使い、TTabSheet オブジェクトのページをアクセスします。
TabHeight プロパティ	タブの高さをピクセル値であらわします。
TabWidth プロパティ	タブの幅をピクセル値であらわします。
FindNextPage メソッド	TTabSheet の次のページを返します。
SelectNextPage メソッド	TTabSheet の次か前のページを選択します。
OnChange イベント	新しいページが選択され、フォーカスが移動した後で発生します。
OnChanging イベント	新しいページが選択され、フォーカスが移動する前に発生します。