OBJECT PASCAL HANDBOOK

Delphi 11 Edition

Delphi 11 Alexandria のための Object Pascal プログラミング完全ガイド

Marco Cantù◉著

藤井 等◉監訳

エンバカデロ・テクノロジーズ◉翻訳

CUTT カットシステム

著者 :Marco Cantù
発行者 :Marco Cantù
編集者 :Peter W A Wood, Andreas Toth (second edition)

Object Pascal Handbook: Delphi 11 Alexandria Edition
この PDF 限定版は 2021 年 11 月に出版されました。
最新の印刷版（Delphi 10.4 Sydney 版）: ISBN-13: 9798554519963

はじめに

To my family, Raffaella, Benedetta, and Jacopo,
with all of my love and a big thank you for all you do
to take my life ahead of my expectations

強力かつ簡潔、表現力と可読性、学習にもプロの開発にも最適。これが現在の Object Pascal の端的な特徴です。この言語には長い歴史、活発な現在、そして明るい未来があります。

Object Pascal は多面的な言語です。伝統的な手続き型プログラミングスタイルのサポートを維持しながら、オブジェクト指向プログラミング（OOP: object-oriented programing）のパワー、ジェネリックプログラミングの強力なサポート、属性などといった動的な構文要素を併せ持っています。モバイル時代に対応したコンパイラと開発環境を備えた多才なツールであり、過去からの確固たるルーツを持ちながらも、将来にも対応した言語なのです。

Object Pascal 言語は、どのような場面で使われているのでしょうか。デスクトップアプリの記述からクライアントサーバーアプリケーション、大規模な Web サーバーモジュールからミドルウェア、オフィスオートメーションから最新の電話やタブレット用アプリ、産業用自動システムからインターネット仮想電話ネットワーク……この言語が、これらの用途に利用できる可能性があるという仮定の話ではなく、今まさに現実の世界で利用されているのです。

最新の Object Pascal 言語の中核は、1995 年の定義に由来します。この年に Java と JavaScript が考案されたことを考えると、1995 年はプログラミング言語にとってすばらしい年でした。Object Pascal のルーツは Pascal の前身に遡りますが、その進化は 1995 年で止まることはありませんでした。現在もコア機能の改良は続けられ、エンバカデロ・テクノロジーズによってデスクトップコンパイラとモバイルコンパイラが作成され、Delphi、RAD Studio で使用することができます。

現在の言語に関する書籍

この言語の役割の変化、長年にわたる拡張、そして今日新しい開発者からも注目を集めているという事実を考えると、現在の Object Pascal 言語について網羅した本を書くことは重要であると考えました。この書籍のゴールは、新たな開発者や他の似たような言語から移行した開発者だけでなく、この言語の最新の変更点について学習しようとしているさまざまな Pascal 系言語の古くからの開発者も対象とした、言語マニュアルを提供することです。

未経験者に基礎が必要なのは当然ですが、変更点が多岐にわたることを考えると、経験者も本書のいくつかの章で新たな発見があるでしょう。

Object Pascal 言語の歴史について簡単に説明した短い付録以外、本書では現在の Object Pascal 言語について解説しています。この言語の中核的な機能の大部分は、1995 年の Object Pascal の最初の実装である Delphi の初期バージョン以来あまり変わっていません。

本書を通して言及していますが、この言語はここ数年、停滞どころか非常に速いペースで進化しています。

過去に書いた他の本では、古典的な Pascal を最初に取り上げ、その後、登場順に拡張について説明するという「時系列的な」手法をとりました。しかし、本書ではより「論理的な」手法を用いています。この言語がどのように進化してきたかではなく、トピックごとに、この言語の現在とその最も良い使用方法について説明します。

たとえば、オリジナルの Pascal 言語から続くネイティブデータ型では、（組み込みの型ヘルパのおかげで）メソッドライクな機能が最近導入されました。第 2 章でこの機能の使用方法を紹介していますが、自分でそのカスタム型を拡張する方法については、後半で触れています。

つまり、本書では現在の Object Pascal 言語について基礎から順に説明し、歴史的な視点は非常に限られています。この言語を過去に使用したことがあっても、最後の数章だけでなく、本書全体にざっと目を通し、新しい機能を確認することをお勧めします。

実践的学習

本書では、まず中核的な概念を説明し、そのすぐ後に読者が実行したり、試してみたくなるような簡単なサンプルを示し、概念の理解と習得につなげることを基本的な考え方としています。本書はリファレンスマニュアルではありません。この言語が理論上なすべきことについて説明し、考えられる状況をすべて示します。正確であることを心がけていますが、言語について説明すること、実践的で段階的なガイドを提供することによりフォーカスしています。一度に 1 つの機能にフォーカスすることが目標であるため、そのサンプルは通常、非常に単純です。

ソースコード全体は、オンラインコードリポジトリである GitHub にあります。単一のファイルとしてダウンロードしたり、リポジトリのクローンを作成したり、オンラインで表示したり特定の

プロジェクトのコードのみをダウンロードしたりすることもできます。リポジトリのクローンを作成すると、私がサンプルの変更や追加を行ったときに、自分のコードの該当箇所を容易に特定して更新することができます。GitHub の場所は、Delphi 10.4 Sydney 向けの印刷版で仕様したのと同じです。

```
https://github.com/MarcoDelphiBooks/ObjectPascalHandbook104
```

サンプルコードをコンパイルしてテストするには、最新バージョンの Delphi が必要です（すべてを動かすには少なくとも 10.4 が必要です。ただし多くのサンプルは、10.x および 11 リリースでも動作します）。

Delphi のライセンスをお持ちでない方には、トライアル版も用意されており、通常 30 日間無料でコンパイラと IDE を使用できます。また、無料の Delphi Community Edition（現在バージョン 10.4 にアップデートされています）もあり、ソフトウェア開発によって収益を得ていない場合か限定的である場合には、無料で利用できます。

謝辞

他の本同様、本書は多くの人々のおかげで完成しました。多すぎてひとりひとり記載できません。本書初版に関わる仕事の大半をともに行ったのは編集者の Peter Wood で、刻々と変わる私のスケジュールに耐え、私の技術英語を改善し、本書を現在の姿にしてくれました。

初版を出してから、読者で開発者の Andreas Toth から初版に関する広範なフィードバックが届きました。その後、彼は第 2 版の編集者として参加してくれ、英文法、書籍としての一貫性、Object Pascal のコーディングスタイルまでをカバーする包括的なコンテンツレビューを行ってくれました。この新版の多くは、彼に依るところが多くあります。

新版は、他の数人の Delphi エキスパート（そのほとんどはエンバカデロ MVP です）によるレビューも受けています。特に François Piette は、最終版のテキストに反映された 100 を超える修正と提案を送ってくれました。

エンバカデロ・テクノロジーズの製品マネージャとしての現在の職のおかげで社内での数え切れないほどの会話、ミーティング、電子メールスレッドで知見が得られ、製品とそのテクノロジに対する理解がさらに深まりました。同僚全員と R&D チームのメンバーに非常に感謝しています。

全員の名前をここで言及することは難しいですが、本書初版に関して直接情報をくれた 3 人については、紹介しておこうと思います。デベロッパーリレーションの David I、開発ツール製品管理のトップであった John Thomas（JT）、そして RAD Studio のチーフアーキテクトの Allen Bauer です。

最近では、RAD Studio の製品マネージャに従事する他の 2 人（Sarina DuPont と David Millington）と、現在エバンジェリストを務める Jim McKeeth、そして R&D アーキテクトとしてす

ばらしい仕事をしてくれている現在のチームの協力も得ています。

エンバカデロ社外の人々も、直接情報が得られる重要な機会を与え続けてくれました。多くのイタリアの Delphi エキスパートから、数多くのお客様、エンバカデロのセールスやテクニカルパートナーの皆さん、Delphi コミュニティメンバー、MVP、さらには、私がしばしば会っている他の言語やツールを利用する開発者まで、いずれも重要です。

もし、このグループから 1 人の名前を挙げるとすれば、エンバカデロに加わる前に多くの時間を共に過ごした Cary Jensen を紹介したいと思います。彼とは、ヨーロッパと米国で、Delphi Developer Days を何度か実施しました。

最後に、私の旅行スケジュール、何夜にも及ぶミーティング、週末の他の書物の執筆に耐えてくれた家族に感謝します。ありがとう、Lella、Benny、Jacopo。

著者自身について

過去 25 年の大半を Object Pascal 言語を使用したソフトウェア開発についての執筆、教育、コンサルティングに費やしました。ベストセラーの「Mastering Delphi」シリーズ、自費出版 で作成した（Delphi 2007 から Delphi XE の各バージョンの）開発ツールに関する「Handbook」シリーズなどを執筆しています。

世界中の多くのプログラミングコンファレンスで講演したり、何百人もの開発者にトレーニングを提供してきました。さらに、Delphi 開発者イベント、企業向けのクラスルームトレーニング、オンライン Web セミナー、CodeRage カンファレンスなどでも講演しています。

2013 年、独立したコンサルタント／トレーナーとして長年働いてきた私のキャリアに突然変化が訪れました。すばらしい開発ツールを開発、販売している企業であるエンバカデロ・テクノロジーズでの Delphi および現在の RAD Studio の製品マネージャとしての職を引き受けました。

これ以上読者をわずらわせないよう、現在はイタリアに住んでいて（最近は頻度が減りましたが）カリフォルニアへ通っていること、魅力的な妻と 2 人のすばらしい子供がいること、できるだけプログラミングに携わり、それを楽しんでいることのみ付け加えておきます。

私がこの新版の執筆を楽しんだのと同じくらい、本書を楽しんでいただけたらと思います。さらなる情報については、次の Web サイト、ソーシャルメディアチャンネルを使用してください。

```
https://www.marcocantu.com
https://blog.marcocantu.com
https://twitter.com/marcocantu
```

目 次

第 III 部　高度な機能 …… 439

第 14 章　ジェネリクス……441

第 15 章　無名メソッド……503

第Ⅰ部
基礎

Object Pascal は、美しいプログラム構造を持ち、拡張可能なデータ型など中核的な基礎機能に支えられた大変強力なプログラミング言語です。これらの基礎部分の一部は、伝統的な Pascal 言語から来ていますが、中核の言語機能であっても、早い段階から多くの拡張が加えられています。

この第 I 部では、言語構文、コーディングスタイル、プログラムの構造、変数やデータ型の使用法、基本的な文（条件やループなど）、手続き、関数の使用法、そして、配列、レコード、文字列など中心的な型コンストラクタについて学習します。

これらは、本書の第 II 部と第 III 部で説明するクラスやジェネリック型などの高度な機能の基礎となります。言語の学習は家を建てるようなもので、しっかりとした土台と基礎から始める必要があります。そうしないと、その上の構造は美しいかもしれませんが、不安定になってしまいます。

第 I 部のサマリー

第１章
Pascal でのコーディング

この章では、コードと関連するコメントを記述する標準的な方法の説明、そしてキーワードやプログラムの構造の概説など、Object Pascal アプリケーションの基本的な要素から始めます。簡単なアプリケーションを作成してその動作を説明し、以降の章で詳細を説明する、その他の主な概念について概説します。

1.1　コードから始めよう

この章では、言語の基礎について説明します。アプリケーションの動作全体の詳細を説明するには、複数の章が必要になってしまいます。そのため、ここでは（構造が異なる）２つの単純なプログラムを取り上げますが、詳細の説明には立ち入りません。プログラムの各種要素を説明する前に、サンプルを作成するのに使用するプログラム構造を紹介し、特定の言語構文要素について紹介します。本書の情報をできるだけ早く実践できるようにするには、このサンプルを初めから確認するのが良いでしょう。

Object Pascal は、その統合開発環境（IDE）と緊密に連携するよう設計されています。Object Pascal がマシンフレンドリな言語の実行スピードを発揮しながら、プログラマフレンドリな言語の容易さと開発スピードを同時に実現できるのは、この強力な組み合わせによるものです。

IDE では、ユーザーインターフェイスの設計、コードの記述、プログラムの実行など、さまざまな作業を行えます。本書の Object Pascal 言語の説明では、この IDE を使用します。

最初のコンソールアプリケーション

まず初めに、単純な「Hello, World」コンソールアプリケーションのコードを示し、Pascal プログラムの構造上の要素を示しましょう。コンソールアプリケーションは、グラフィカルユーザーインターフェイスを持たないプログラムで、テキストを表示してキーボード入力を受け付ける、通常オペレーティングシステムコンソールまたはコマンドプロンプトから実行されるアプリケーションです。一般にコンソールアプリケーションはモバイルプラットフォームではほとんど意味をなしませんが、Windows（Microsoft は最近、cmd.exe の改善や PowerShell、ターミナルアクセスなどにも注力しています）でもまだ使われており、Linux では主流です。

次のコードの各要素について説明することは、最初の数章にわたる目的なので、ここではまだ触れずにおきます。以下が、HelloConsole サンプルのコードです。

```
program HelloConsole;

{$APPTYPE CONSOLE}

var  StrMessage: string;

begin
  StrMessage := 'Hello, World';
  Writeln(StrMessage);
  // Enterキーが押されるまで待機
  Readln;
end.
```

メモ

「はじめに」で説明したように、本書で取り上げる全サンプルの完全なソースコードは GitHub リポジトリにあります。サンプルの入手方法の詳細は、「はじめに」を参照してください。本文ではプロジェクト名で呼びます（この例では HelloConsole）が、これはサンプルの各種ファイルを含むフォルダの名前でもあります。プロジェクトフォルダは章ごとに分かれているため、この 1 つ目のサンプルは 01/HelloConsole にあります。

1 行目には、program という宣言の後ろにプログラム名、2 行目以降にコンパイラ指令（$ 記号で始まり、中かっこで囲まれています）、変数宣言（string という型名と名称）が続き、メインの begin-end ブロック内に 3 行のコードとコメントがあります。3 行のコードでは値を文字列にコピーし、システム関数を呼び出してそのテキスト行をコンソールに出力し、別のシステム関数を呼び出してユーザー入力行を読み取ります（この例ではユーザーが Enter キーを押すまで待ちます）。後述するように Object Pascal では独自の関数を定義することもできますが、あらかじめ定義された

数多くの関数が用意されています。

繰り返しになりますが、この最初のセクションの目的は、小さくても完全な Pascal プログラムがどのようなものかを示すことであるため、これらの全要素についての説明は後にまわします。もちろん、このアプリケーションは、開いて実行することができます。出力は次のとおりです（Windows 版の実例を図 1.1 に示します）。

```
Hello, World
```

図 1.1
Windows で実行したサンプル HelloConsole の出力

最初のビジュアルアプリケーション

最近のアプリケーションは、以前のコンソールプログラムとは異なり、通常、ウィンドウ（フォームと呼ばれます）に表示されるビジュアル要素（コントロールと呼ばれます）で構成されます。本書ではほとんどの場合（単純なテキストを表示することになる場合がほとんどですが）、FireMonkey（FMX）プラットフォームライブラリを使用してビジュアルサンプルを作成します。

メモ

Delphi のビジュアルコントロールには、VCL（Windows 用ビジュアルコンポーネントライブラリ）と FireMonkey（サポートされているすべてのプラットフォーム、デスクトップおよびモバイル用のマルチデバイスライブラリ）の 2 つの「種類」があります。このデモを Windows 固有の VCL ライブラリに適応させることは難しくありません。

ビジュアルアプリケーションの正確な構造について理解するには、本書の該当部分を参照しなければなりません。フォームは、特定のクラスのオブジェクトで、メソッド、イベントハンドラ、プロパティを持ち、いずれの機能の理解にも相応の時間がかかります。しかし、これらのアプリケーションを作成するには、エキスパートである必要はありません。新しいデスクトップアプリケーションまたはモバイルアプリケーションを作成するには、メニューコマンドを使用するだけです。本書ではまず、FireMonkey プラットフォームを使用して、フォームとボタン、そしてそのクリック操作を扱います。はじめに、任意の種類のフォーム（デスクトップでもモバイルでもどちらでも、私は通常、マルチデバイスアプリケーションで「空のアプリケーション」を選択します）を作成し、出力を表示する複数行のテキストコントロール（Memo）とボタンコントロール（Button）を配置

します。

図 1.2 に、モバイルアプリケーションの場合のフォームが Delphi IDE でどのように表示されるかを示します。Android スタイルプレビューを選択（設計画面の上部のコンボボックスをご覧ください）して、ボタンコントロールを配置した状態です。

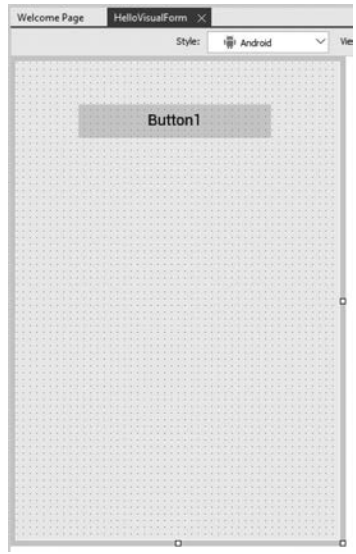

図 1.2
HelloVisual デモで使用されている、ボタンを
1 つ持つ単純なモバイルアプリケーション

同様のアプリケーションを作成するには、空のフォームにボタンを追加します。ここで実際にコードを追加する（それがここでのポイントです）には、ボタンをダブルクリックします。次のコードスケルトン（または同様のもの）が表示されます。

```
procedure TForm1.Button1Click(Sender: Tobject)
begin

end;
```

クラスのメソッド（Button1Click）を知らなくても、コードフラグメント内（begin キーワードと end キーワードの間）に入力すれば、ボタンを押したときにそのコードが実行されます。

最初の「ビジュアル」プログラムのコードは、1 つ目のコンソールアプリケーションのコードと同様の処理ですが、コンテキストと呼び出すライブラリ関数（ShowMessage）が異なります。HelloVisual サンプルのコードは次のとおりです。最初から再作成しても簡単です。

```
procedure TForm1.Button1Click(Sender: TObject)
var
  StrMessage: string;
begin
  StrMessage := 'Hello, World';
  ShowMessage(StrMessage);
end;
```

begin 文とその後の実際のコードの前に StrMessage 変数の宣言を配置する必要があることに注意してください。繰り返しになりますが、詳細は後で説明するため、不明な点があっても心配しないでください。

メモ このサンプルのソースコードは、この章の 01 コンテナ下のフォルダにあります。ただし、この場合、このサンプルと似た名前のプロジェクトファイルがありますが、プロジェクト名の後ろに "Form" が付いたセカンダリユニットファイルもあります。それが、本書での標準です。プロジェクトの構造は、この章の最後に説明します。

図 1.3 に、FMX モバイルプレビューモードを有効し Windows で実行した、この単純なプログラムの出力を示します（このデモは Android、iOS および macOS でも実行できますが、IDE に追加の設定が必要です）。

メモ FireMonkey モバイルプレビューを使用すれば、Windows アプリケーションをモバイルアプリのように見せることができます。本書のほとんどのサンプルで、このモードを有効にしています。これは、プロジェクトのソースコードの uses 節に MobilePreview ユニットを追加することで可能です。

Delphi プログラムの記述方法とテストのやり方がわかったので、この章の最初に述べたとおり、初めに戻ってアプリケーションの基本的な要素のいくつかを詳細に説明していきましょう。まず理解する必要があるのは、プログラムの「読み方」、各種要素の記述方法、そして作成したアプリケーションの構造（PAS ファイルと DPR ファイルによって構成）です。

図 1.3
HelloVisual デモで使用されている、ボタンを
1 つ持つ単純なモバイルアプリケーション

1.2　構文とコーディングスタイル

実際の Object Pascal 言語の文の記述について説明する前に、Object Pascal コーディングスタイルのいくつかの要素について見ていくことも重要です。ここで考えたいのは、（まだ説明していない）構文ルール以外のコードの記述方法です。個人の好みによってスタイルは異なるため、この質問に対する唯一の正解はありません。ただし、コメント、大文字、スペース、そして以前に「プリティプリント（コンピュータではなく人間にとってのプリティ）」と呼ばれていた現在では時代遅れの用語など、知っておくべき原則がいくつかあります。

一般的に、コーディングスタイルのゴールは明確さです。スタイルやフォーマットを決めるのは、特定のコードブロックがどのような目的であるかを表す略記のルールです。明確さに不可欠な要素は一貫性で、どのようなスタイルを選択しようとも、プロジェクト全体、そしてプロジェクト間でも、一貫性を保つ必要があります。

ヒント

IDE（統合開発環境）では、（ユニットレベルまたはプロジェクトレベルの）自動コードフォーマットがサポートされます。エディタで Ctrl + D キーを使用して、ルールのセットに従ってコードを再フォーマットできます。ルールのセットは、約 40 個の各種フォーマット要素（IDE の［オプション］にあります）を調整して変更でき、これらの設定をチームの他の開発者と共有してフォーマットの一貫性を保つこともできます。ただし、自動コードフォーマットは、いくつかの最新言語機能はサポートしていません。

コメント

コードを見ればわかるというケースもありますが、プログラムのソースコードに多くのコメントを追加し、なぜ、コードがこのように記述されているのか、その理由や想定を他者に（あるいは、将来コードを参照したときの自分自身にも）わかるようにすることは、適切な配慮です。

伝統的な Pascal では、コメントは、中かっこ、または丸かっことアスタリスクのペアによって囲まれていました。最近では、ダブルスラッシュの C/C++ スタイルの 1 行のコメントも使用できます。この場合、コメントは行の終わりまでで、コメントの終わりを示す記号は必要ありません。

```
// このコメントは1行の終わりで終了します

{ これは複数行の
  コメントです}

(* これも別の形式の
  複数行コメントです *)
```

1 つ目のコメント形式が最も一般的に使用されていますが、Pascal で独自に定義されたものではありませんでした。これは、C/C++ からの借用ですが、C#、Objective-C、Java、JavaScript 同様、複数行のコメント用に `/* comment */` も使用されます。

2 つ目の形式のほうが、3 つ目の形式よりもよく使用されています。ヨーロッパでは、キーボードに中かっこ記号がなかったため（あるいは、複数のキーの組み合わせとなり、面倒なため）、これが使用されていました。いいかえれば、この古い形式はちょっと時代遅れです。

行の終わりまでのコメントは、短いコメントの場合と 1 行のコードをコメントアウトする場合に有用です。このコメント形式は、Object Pascal 言語で最もよく使用されています。

ヒント

IDE エディタで、現在の行（または選択した行のグループ）を直接キーストロークでコメントにしたり、コメント解除できます。これは、US キーボードでは Ctrl + / の組み合わせですが、他のキーボードレイアウトでは異なります（物理的に / キーがどこにあるかに依存します）。実際のキーの組み合わせは、エディタのポップアップメニューに示されます。

3 つの異なるコメントの形式があることは、コメントをネストさせる場合に有用です。ソースコードの複数行をコメントアウトして無効にしたいときに、これらの行の中にコメントが含まれている場合、同じコメント識別子を使用することはできません。

```
{
  コード...
  {ネストしたコメントが問題を引き起こす}
  コード...
}
```

上記のコードは、最初に記述された右中かっこでコメントセクション全体の終了が示されるため、コンパイルエラーになってしまいます。3 つ目のコメント識別子を使用すれば、コードを次のように正しく記述できます。

```
{
  コード...
  // このコメントはOK
  コード...
}
```

あるいは、前述のように複数行を // でコメントアウトし、2 行目の // コメントが {} で囲まれたコメント行に追加した状態であれば、（元のコメントを保持しつつ）同じブロックを簡単にコメント解除できます。

メモ

1 つ目のサンプルの {$APPTYPE CONSOLE} の行のように、左中かっこまたは丸かっことアスタリスクの後にドル記号 ($) が続く場合は、コメントではなく、コンパイラ指令になります。コンパイラ指令はコンパイラへの特別な指示で、この章の後半で簡単に説明します。
実際、コンパイラ指令はコメントでもあります。たとえば、{$X+ This is a comment} は有効です。これは、有効な指令でもありコメントでもあります。ただし、ほとんどの「良識的な」プログラマは指令とコメントを分ける傾向にあります。

コメントと XML ドキュメント

Object Pascal には、他のプログラミング言語でも共通している、特殊なコメントが用意されています。これは、コンパイラによって特定の方法で処理されます。この特殊なコメントからはドキュメントが生成され、IDE のヘルプインサイトで直接利用されるほか、コンパイラによって生成される XML ファイルで使用されます。

メモ

Delphi IDE では、ヘルプインサイトが自動的にシンボルに関する情報（型や定義された場所など）を表示します。XML ドキュメントコメントを用いれば、ソースコード内に記述する特定の詳細情報で、この情報を補足することができます。

XML ドキュメント は、/// コメントまたは {! コメントで有効になります。これらのコメント内では、一般的なテキストまたは（より適切な）特定の XML タグを用いて、コメントしているシンボルについて、さらにはそのパラメータや戻り値などに関する情報を記述することができます。これは、自由形式テキストを用いた極めて単純な例です。

```
public
  /// This is a custom method, of course
  procedure CustomMethod;
```

XML ドキュメントの生成を有効にしていると、この情報はコンパイラによって生成される XML 出力に追加されます。以下がその例です。

```
<procedure name="CustomMethod" visibility="public">
  <devnotes>
    This is a custom method, of course
  </devnotes>
</procedure>
```

同じ情報は、IDE でシンボルの上にマウスを置くと表示されます。図 1.4 がその例です。

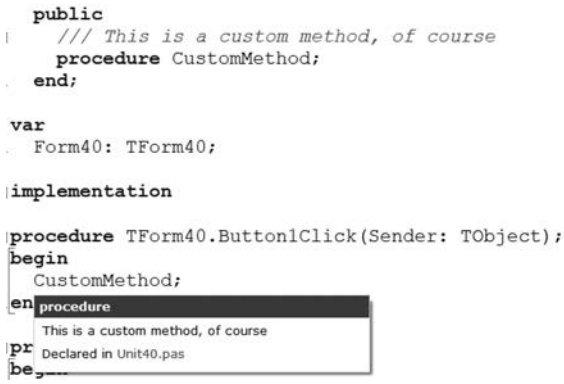

図 1.4
Delphi IDE のヘルプインサイトが /// コメントによって記述された XML ドキュメント情報を表示

ガイドラインに従ってこのコメントに summary セクションを記述しておくと、ヘルプインサイトウィンドウにもこの情報が表示されます。

```
public
  /// <summary>This is a custom method, of course</summary>
  procedure CustomMethod;
```

この機能のメリットは、パラメータ、戻り値、より詳細な情報の記述に用いることのできる多くの XML タグが用意されていることです。利用可能なタグの一覧は、以下に掲載されています。

```
https://docwiki.embarcadero.com/RADStudio/ja/XML_ドキュメント_コメント
```

シンボル識別子

プログラムは、さまざまな要素（データ型、変数、関数、オブジェクト、クラスなど）の名前を示すためのさまざまなシンボルによって構成されています。プログラムでは、任意の識別子を使用できますが、いくつかの従うべきルールがあります。

- 識別子にスペースを含めることはできません（スペースによって識別子と他の言語要素とを区別するため）
- 識別子には、全 Unicode 文字を含む文字および数字を使用できます。必要であれば、自国の言語でシンボルの名前を付けることができます（いくつかの文字は推奨されません。IDE のいくつかの箇所で、同じサポートが提供されているわけではないからです）。
- 従来の ASCII 記号のうち識別子に使用できるのはアンダースコア記号（_）のみです。文字および数字以外の ASCII 記号は使用できません。識別子に使用できない記号には、算術記号（+、-、*、/、=）、すべてのかっこ、句読点、特殊文字（@、#、$、%、^、&、\、| など）などがあります。使用できるのは、☼ ♪や∞などの Unicode 記号です。
- 識別子は文字またはアンダースコアで始まる必要があります。数字で始まることはできません（数字は使用できますが、先頭に使用することはできません）。ここでいう数字とは ASCII 数字（0 から 9）のことで、Unicode 表現の数字なら使用することができます。

次に、IdentifiersTest サンプルにリストされている典型的な識別子の例を示します。

```
MyValue
Value1
```

```
My_Value
_Value
Val123
_123
```

次に、有効な Unicode 識別子の例を示します（最後のは、やや極端な例です）。

```
Cantù（アクセント記号付きの文字）
结（簡体中国語）
画像（日本語）
☼（Unicodeの記号）
```

次に、「無効な」識別子の例をいくつか示します。

```
123
1Value
My Value
My-Value
My%Value
```

ヒント 実行時に識別子が有効かどうかをチェックする場合（他の開発者のためにツールを作成する場合以外あまり必要になることはありませんが）、`IsValidIdent` というランタイムライブラリの関数を使用できます。

大文字・小文字の区別無視と大文字の使用

C の構文に基づく言語（C++、Java、C#、JavaScript など）をはじめとする他の多くの言語とは異なり、Object Pascal コンパイラでは識別子の大文字／小文字の区別は無視されます。したがって、識別子 `Myname`、`MyName`、`myname`、`myName` そして `MYNAME` はすべてまったく同じです。大文字／小文字を区別する言語では、大文字／小文字の間違いによって構文エラーやささいなミスが起こることがあるため、大文字／小文字を区別しないことは、個人的には良い機能だと考えます。

ただし、識別子に Unicode を使用できることを考えると、同じ文字の大文字は同じ要素として扱われ、アクセント記号付きは別のシンボルとして扱われるため、状況は多少複雑になります。つまり、次のようになります。

```
cantu: Integer;
Cantu: Integer; // エラー：識別子が重複
cantù: Integer; // 正しい：異なる識別子と認識される
```

注意 大文字／小文字の区別には例外が 1 つのみあります。C++ の互換性の問題から、コンポーネントパッケージの *Register* 手続きは大文字の *R* で始まる必要があります。もちろん、他の言語によってエクスポートされた識別子（ネイティブオペレーティングシステム関数など）を参照する場合、大文字／小文字を適切に使用する必要がある場合があります。

ただし、ささいなデメリットがいくつかあります。まず、これらの識別子が同一であることを認識する必要があることです。異なる要素として使用しないようにしなければなりません。次に、コードの読みやすさを向上させるために、大文字の使用について一貫性を保つ必要があることです。

大文字／小文字の使用について一貫性を保つことはコンパイラによって実現されることではありません。習慣付けです。一般的な方法は、各識別子の最初の文字のみを大文字にすることです。識別子が複数の連続する語で構成される場合（識別子にスペースを含めることはできません）、各語の最初の文字を大文字にすることをお勧めします。

```
MyLongIdentifier
MyVeryLongAndAlmostStupidIdentifier
```

これは、いわゆるキャメル記法との対比で、しばしば「Pascal 記法」と呼ばれます。Java やその他の言語のキャメル記法は C の構文に基づいており、次のように内部の語を大文字で始め、先頭の文字は小文字にします。

```
myLongIdentifier
```

実際、ローカル変数にキャメル記法（先頭が小文字）が使用され、クラス要素、パラメータやその他のよりグローバルな要素に Pascal 記法が使用された Object Pascal コードがより一般的になりつつあります。いずれにしても、本書のソースコード例では、すべてのシンボルに一貫して Pascal 記法を使用します。

空白

コンパイラによって完全に無視されるその他の要素は、ソースコードに追加されたスペース、改行、タブです。これらのすべての要素はまとめて「空白」と呼ばれます。空白はコードの読みやすさを向上させるためにのみ使用されます。コンパイルには一切影響しません。

従来の BASIC とは異なり、Object Pascal では長い命令を複数行に分け、複数行にわたる文を記述することができます。複数行にわたる文を使用できることのデメリットは、文の終わりを示すために（より正確にいうと、文を次の文と区別するために）セミコロンを追加する必要があることで