C++Builder Reviewers Guide

C++Builder で始める ビジュアル プログラミング

2020 年 11 月
（バージョン 10.4 対応）

エンバカデロ・テクノロジーズ

バージョン 10.4 Sydney の概要や
ビジュアルプログラミングの手法を解説した評価ガイド

目次

C++Builder の世界へようこそ！

このたびは、C++Builder 10.4 Sydney を評価いただき、誠にありがとうございます。C++Builder は、強力な C++言語によるプログラミングをサポートしながら、Delphi や Visual Basic などでなじみのあるコンポーネントによる開発を実現しており、効率的に UI を設計して C++プログラミングを進めることができます。C++Builder は、Windows 10、iOS 向けの真のネイティブアプリケーションの構築に加え、クラウドや IoT とつながるアプリケーションの構築をサポートしています。

最新の C++Builder を用いれば、新しい C++言語の強力な機能を用いながら、Windows 10 VCL コントロール、プラットフォームスタイル、ユニバーサル Windows プラットフォームサービスコンポーネントなどを用いて、すばやく Windows 10 対応のネイティブアプリケーションを作成できます。さらに、FireMonkey フレームワークを用いることで、iOS モバイルプラットフォーム向けにもアプリケーションを展開することができます。

C++Builder は、接続性という点でも優れています。多様な RDBMS へのアクセスに加え、ビッグデータ、NoSQL、クラウドサービス、エンタープライズサービスへの接続もサポートしており、サービス指向のアプリケーションを構築するために役立てることもできます。

このガイドは C++Builder の機能評価の出発点であることにご留意ください。C++Builder には、このガイドで紹介しきれないほどの数々の機能があります。エンバカデロでは、これらの機能を最大限に活用するのに役立つ補足的な情報、ビデオ、ウォークスルー、ガイドなどを用意しています。本製品の最新情報については、機能一覧、製品情報ページなどを参照してください。

お問い合わせ先

C++Builder の評価に関してご不明な点、ご質問などがございましたら、下記までお問い合わせください。

エンバカデロ・テクノロジーズ インフォメーションサービスセンター
TEL：03-4540-4148　Email： japan.info@embarcadero.com

C++Builder 10.4 Sydney の新機能

C++Builder 10.4 Sydney では、最新の OS プラットフォームのサポートに加え、C++言語機能の強化、開発生産性、効率性を強化などが加えられています。

継続的なアップデート

近年の C++Builder は、およそ年 1 回のメジャーリリースの他に、継続的に年数回のアップデートリリースを実施しています。これは、日々変化する OS 環境への迅速な対応と、製品機能を常にブラッシュアップしていくという方針に基づくものです。

モバイル向けの OS は、頻繁なアップデートを行っており、ユーザーもこれらを積極的にインストールする傾向にあるため、モバイル向けアプリを開発する場合には、最新 OS に追従するための開発環境のアップデートは欠かせません。一方、Windows OS は、安定的に同じバージョンを利用する傾向にあったため、開発環境を継続的にアップデートするという考えは一般的ではありませんでした。しかし、Windows 10 は、年 2 回のアップデートがスケジュールされており、これらのアップデートでは大幅な機能拡張や仕様変更が加えられています。つまり、Windows 10 向けのアプリケーション開発では、モバイル向けと同様に、最新 OS 環境に追従する継続的な開発が必要となるのです。

エンバカデロでは、このような OS 環境の変化と開発需要に対応するため、上記のような継続的なアップデートを実施しています。そして、開発者の負担を最小化するため、製品にアップデートサブスクリプション（年間保守）を標準添付しています。

開発者は、アップデートサブスクリプションサービスを継続することで、常に最新環境を利用することができます（同時に多数の旧バージョンへのアクセスも可能です）。

メモリ管理の統合

Delphi / C++Builder のメモリ管理機構は、すべてのサポートプラットフォーム（モバイル、デスクトップ、サーバー）で、従来型のオブジェクトメモリ管理の方式に統一されました。ARC（Automatic Reference Counting：自動参照カウント）と比較して、既存コードとの互換性が高く、コンポーネント、ライブラリ、エンドユーザーアプリケーションで、よりシンプルなコーディングが可能になります。ARC モデルは、文字列管理やインターフェイス型の参照向けに、すべてのプラットフォームで残されます。この変更は C++コード内での Delphi スタイルのクラスの生成や定義が、ヒープに割り当てられる C++クラスと同様の通常のメモリ管理に従うこととなり、複雑性が大幅に低減することを意味します。

IDE の機能強化

10.4 では、IDE の GetIt パッケージマネージャを大幅に強化しました。パッケージをリリース日で並び変えたり、インストールパッケージ、アップデートサブスクリプションユーザー向けのコンテンツ、アップデートが提供されているパッケージなどの基準でフィルタリングできるようになりました。

また、GetIt インストーラテクノロジーを使用した統合インストーラを導入しました。これにより、（インターネット接続環境での）オンラインインストールと、（ISO イメージを用いた）オフラインインストールの双方で、単一のインストーラを使用するようになりました。オンラインとオフラインインストールのいずれでも、プログラミング言語とターゲットプラットフォームの選択を任意に選択可能。言語サポート、ヘルプリソースなど、インストールセットはいつでも追加／削除可能です。

VCL の機能強化

VCL（Visual Component Library）は、Delphi / C++Builder のファーストバージョンから利用されている Windows 向けのビジュアルコンポーネントです。VCL は、近年の Windows 環境の変化に対応し、数多くの機能強化が施されており、現在も進化を続けています。

ここ数年の大きな機能強化は、Windows 10 への対応と High DPI サポートの強化です。Windows 10 向けの新しい UI コントロールが追加されたほか、高解像度モニターでの表示などを改善しています。

以下は、10.4 で新たに追加された VCL の強化ポイントの一例です。

- High DPI における VCL スタイルの変更：High DPI および 4K モニターサポート向けに、VCL スタイルアーキテクチャが大幅に拡張され、VCL フォーム上のすべての UI コントロールは、フォームが表示されるモニターの適切な解像度に合わせて自動的にサイズ変更されます。スタイル API もアップデートされ、High DPI スタイルをサポートしました。
- 新しい High DPI スタイル：多数のビルトインおよびプレミアム VCL スタイルを更新し、新しい High DPI スタイル モードをサポートすることで、あらゆるモニター向けの完成度の高いアプリケーションの設計が可能になりました。
- コントロールスタイル単位での VCL：複数の VCL スタイルを、単一のアプリケーション内の異なるフォーム、あるいは同じフォーム上の異なるビジュアルコントロールに適用できるようになりました。これには、プラットフォームデフォルトテーマとの両立も含みます。その結果、スタイル設定にさらなる柔軟性が加わるだけでなく、スタイルをサポートしていないサードパーティコントロールを、スタイル付き VCL アプリケーションで使用可能にする効果も得られます。

- 新しい VCL コンポーネント：Edge Browser コントロール：TEdgeBrowser を用いれば、VCL アプリケーションで、Chromium ベースの新しい Microsoft Edge WebView2 を使用できます。これにより、よりモダンでセキュアな HTML エンジンを利用できるようになります。従来の TWebBrowser コンポーネントは、既存の Internet Explorer ベースのエンジンと Edge ブラウザを、必要に応じて動的に切り替えできるようになります。
- 新しい VCL コンポーネント：TTitleBarPanel および CustomTitleBar プロパティ：新しい TTitleBarPanel コントロールと TForm.CustomTitleBar プロパティにより、VCL フォームのネイティブ Windows タイトルバーのカスタマイズが可能になりました。これにより、Office、Explorer、Google Chrome などのアプリケーションと同様の、最新の拡張タイトルバーを構築できます。
- 新しい VCL コンポーネント：マルチ解像度をサポートするイメージコンポーネント：新しい TVirtualImage がマルチ解像度／DPI スケーリングをサポート。従来の TImage コンポーネントと置き換えることで、高品質なイメージスケーリングと表示が可能になります。
- Windows API のアプデート：RAD Studio が提供する優れたプラットフォーム統合をさらに向上させるため、多くの API 宣言を拡張し、さらに追加も行いました。

FireMonkey の機能強化

FireMonkey は、Delphi / C++Builder でマルチデバイスアプリケーションを開発する際に利用するクロスプラットフォームコンポーネントフレームワークです。

10.4 では、Professional、Enterprise、Architect のいずれのエディションでも、FireMonkey によるモバイル開発をサポートしています。単一コードにより、複数プラットフォーム向けのネイティブ開発をサポートしているので、プラットフォームごとに複数の言語やツールを習得する必要がなく、劇的な開発効率化が可能になります。

10.4 では、FireMonkey に関連して以下のような機能強化が施されています。

- Metal API のサポート：macOS プラットフォーム（Delphi）で、古い Quartz API や OpenGL（Apple で廃止予定）ではなく、Metal API に対してアプリケーションを構築できるようになりました。これにより、今後の要件へのスムーズな移行と、画面描画のより良いパフォーマンスが提供されます。
- 最新の iOS SDK のサポートに加え、組み込みの IDE サポートを通じ、Apple の起動画面のストーリーボード要件にも対応できます。

- 10.4 では、Windows プラットフォームでのスタイル付き TMemo コンポーネント用の新しい FMX 実装が含まれており、IME のサポートが改善され、追加の機能強化が行われています（macOS プラットフォームは 10.4.1 からサポート）。
- iOS 用 TWebBrowser コントロールは、WKWebView API を使用して実装されました。
- Media Player コントロール の macOS 実装は、AVFoundation の使用により可能となりました。
- FireMonkey フレームワークを使用する開発者は、サポート済みオペレーティング システムの最新バージョンをターゲットとすることができます。

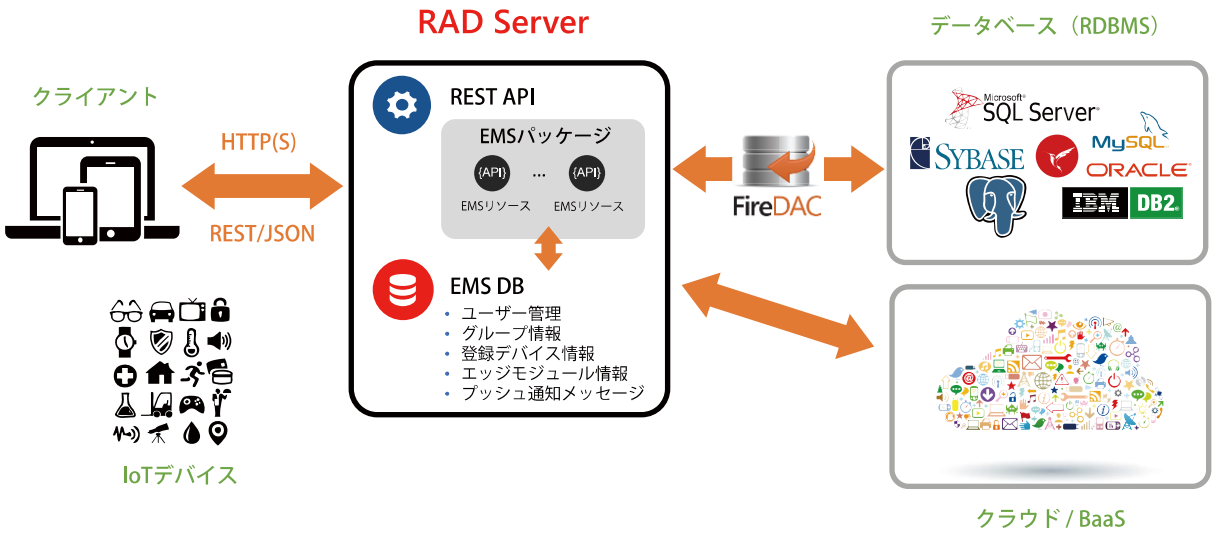
C++言語の機能強化

C++Builder には、従来からの Win32 向けクラシックコンパイラ（bcc32）と、Clang ベースの新しいコンパイラの双方が搭載されています。10.4 Sydney は、C++17 をサポートしており、マルチプラットフォームに対応するべく、各プラットフォーム向けにコンパイラが用意されています。

- C++ライブラリサポートの拡張：10.4 では、さまざまな人気の C++ライブラリを C++Builder 向けに移植し、C++Builder 内で利用できるように最適化しました。これには、すでにサポートされている Eigen などに加え、ZeroMQ、SDL2、SOCI、libSIMDpp、Nematode などのライブラリが含まれます（GetIt パッケージマネージャからダウンロード可能）。
- C++向け Win64 デバッガとリンク：10.4 では、新しい Windows 64-bit C++向けデバッガが導入されました。LLDB をベースとした新しいデバッガは、64-bit アプリケーションのデバッグ時の安定性を大幅に改善し、C++や Delphi の文字列型のほか、STL コレクション（std::vector や std::map など）など、複雑なデータ型の評価やインスペクトをサポートしています。新しいデバッガでは、従来とは異なるデバッグ情報のフォーマットを使用しています。これにより、より安定したデバッグが可能となり、従来よりも強力なデータの評価／インスペクトを実現しています。
- ツールチェーンのパフォーマンスと品質の向上: Dinkumware による多数の STL の改善といくつかの主要な RTL メソッドとその周辺が改善されました。これは、汎用 C++ライブラリとの互換性を改善させるために行われた作業をベースとしています。また CMake サポートにおける改善および多数の品質および安定性における改善されています。

RAD Server によるサービス指向アプリケーションの構築

RAD Server は、REST/JSON ベースのカスタム API を実装するための中間サーバー機能を提供します。開発者は、従来の Delphi / C++Builder でも使い慣れたデータアクセスコンポーネントやロジックコンポーネントを用いて、容易にサーバーサイドアプリケーションを実装できます。



RAD Server を用いれば、モバイルクライアントとバックエンドシステムを結び付け、業務システムにモバイルクライアントを加えることが可能になります。さらに、新たにエンバカデロの製品ファミリーに加わった Web 開発ソリューション Sencha のバックエンドとしても RAD Server を利用できるため、既存の Delphi / C++Builder アプリケーションを Web に拡張する手段としても有効となります。

無料からはじめよう！

C++Builder は、エンバカデロが提供する C++向けビジュアル開発ツールです。その歴史は古く、ボーランドが提供してきた Turbo C や Borland C++のコンパイラがベースとなっています。Delphi と同じようなビジュアル開発を C++で実現したいという要望に応えて 1997 年に最初のバージョンがリリースされて以降、多くの C++開発者に支持されてきました。

Windows 向けの C++開発ツールとして登場した C++Builder は、その後、マルチデバイス向け開発ツールへと進化し、現在では、Windows、iOS 向けのネイティブアプリケーション（※）を開発できます。

C++Builder による開発を体験するには、無料から始めることができます。30 日トライアル版をダウンロードすれば、C++Builder のすべての機能を 30 日間試用することができます。また、個人またはスタートアップ企業の方であれば、Community Edition をダウンロードすることで、Professional 版相当の機能を使って開発を始められます（商用開発には制限があります）。

個人またはスタートアップ企業の方は

以下のページから C++Builder Community Edition をダウンロードしてください。

https://www.embarcadero.com/jp/products/cbuilder/starter

C++Builder Community Edition には、Windows、iOS 向けアプリケーション（※）を単一の C++コードベースから開発できるマルチデバイス開発機能、ローカルデータベースアクセス機能などが搭載されています。個人開発者または 5 名以下の開発者の企業で利用（企業の年間売上が 5,000 US ドル未満、または個人開発者の場合、作成したアプリケーションの年間売上が 5,000 US ドル未満の場合に限る）することができます。

※　Android および macOS については、32bit アプリケーション開発のみの限定的なサポートが提供されています。

企業ユーザーの方は

Community Edition の利用規定に該当しない企業ユーザーの方は、以下のページから RAD Studio トライアル版をダウンロードしてください。

https://www.embarcadero.com/jp/products/rad-studio

RAD Studio トライアル版は、Delphi / C++Builder のすべての機能を 30 日間試用することができます。

C++Builder のビジュアル開発を体験しよう

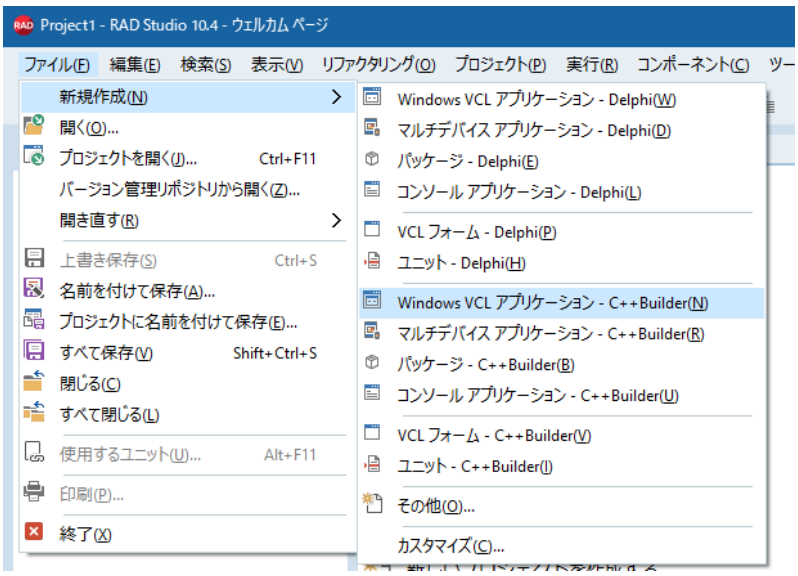
初めて C++Builder に触れる方は、複雑な C++による UI 設計を行うことなく、「コンポーネント」と呼ばれるパーツをドラッグ＆ドロップで配置するだけで簡単に UI を構築できることに驚くかもしれません。C++Builder は、ビジュアル操作で簡単にアプリケーションを開発できるツールですが、C++言語の機能を制約するものではありません。開発者は、コンポーネントによる生産性と C++言語のパワーを両立した開発が可能なのです。

はじめに、C++Builder によるビジュアル C++プログラミングの手法を理解するために、簡単なパーソナル Web ブラウザを作ってみましょう。

C++Builder を起動する

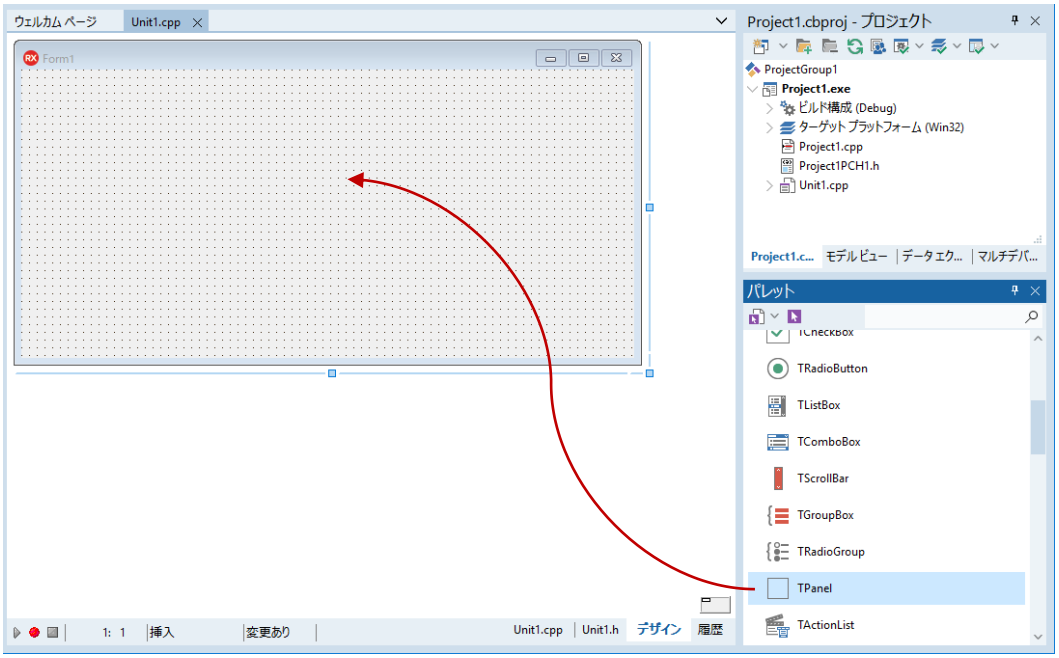
C++Builder、あるいは RAD Studio のアイコンをダブルクリックして統合開発環境（IDE）を起動します。C++Builder によるプログラミング作業は、この統合開発環境で行います。ユーザーインターフェイスの設計、コーディング、ビルド、デバッグ、実デバイスへの配置など、すべての操作をこの環境で実行できます。

新規にアプリケーションを作成するには、［ファイル(F)｜新規作成(N)｜Windows VCL アプリケーション - C++Builder］を順に選択します。

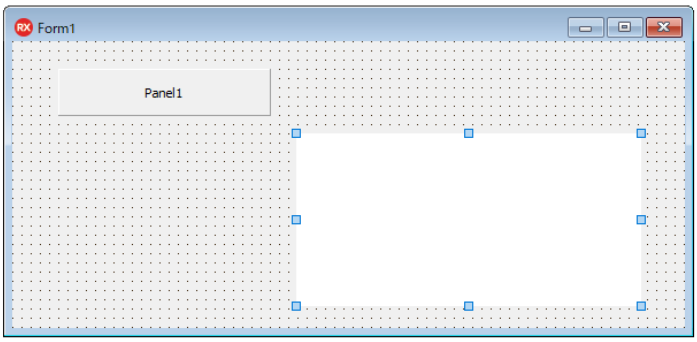


コンポーネント・UI パーツの配置

右下の「ツールパレット」から TPanel をドラッグし、画面中央のフォーム上にドロップします。ツールパレットの検索バーに「Panel」と入力すると、探しやすくなります。

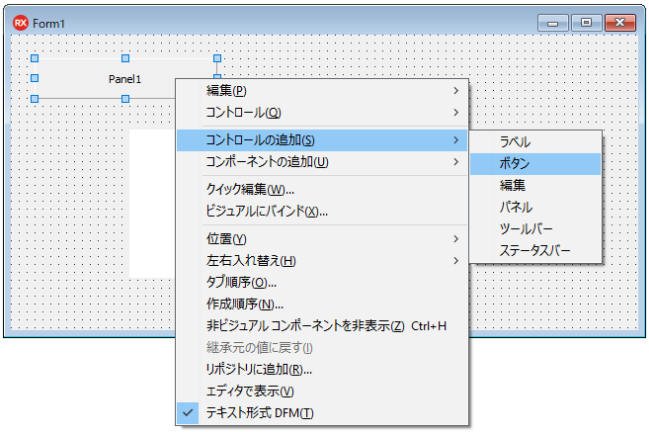


同様に、TWebBrowser をドラッグして、フォーム上の別の位置にドロップします。フォームは、以下のようになります。



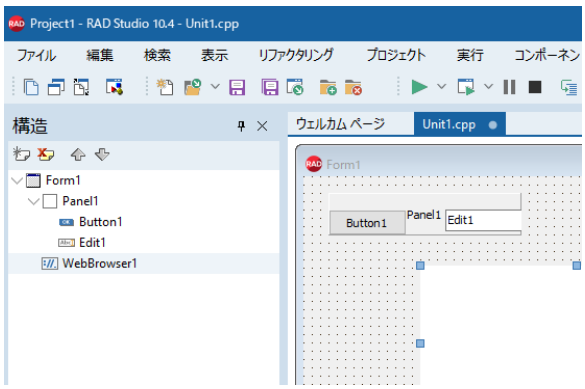
頻繁に使うコンポーネントは、簡単な方法で追加することもできます。フォーム上に配置した、TPanel（自動的に「Panel1」という名前が振られています）を右クリックすると、クイックメニューが表示されます。

ここから、［コントロールの追加(S)｜ボタン］を選択すると、Panel1 の上にボタンを追加することができます。



ボタン（Button1）を配置したら、再び Panel1 を右クリックして、［コントロールの追加(S)｜編集］を選択して、Panel1 の上に Edit1 を追加します。

ここで左上の「構造」ペインを確認してみましょう。Button1 と Edit1 は、Panel1 の下の階層（子項目）に位置していることが分かります。



Panel のように、別のコンポーネントをその上に置くことのできるコンポーネントを「コンテナ」といいます。コンテナの上に置かれたコンポーネントは、「構造」ペインでは、コンテナの子項目として表示されます。Panel を移動させると、その上に置かれた、Button1、Edit1 は一緒に移動します。コンテナを用いることで、複数のコンポーネントをグループ化してレイアウトすることができます。

もし、「構造」ペインで、**Button1** や **Edit1** が Panel の下に位置していない場合には、Panel の子項目にはなっていないということです。これでは、レイアウトをする上で困ったことになるので、修正しておく必要があります。次のように「構造」ペインで操作をしてください。

- **Edit1** を **Panel1** の上にドラッグアンドドロップします
- **Button1** を **Panel1** の上にドラッグアンドドロップします

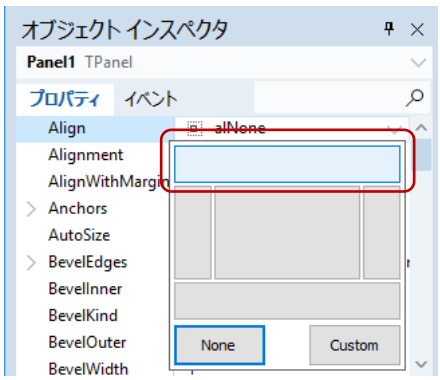
この操作は、必ず「構造」ペインで行います。フォーム上でコンポーネントを移動させても、親子関係は変わらないことに注意してください。

レイアウトを調整する

コンポーネントは、XY 座標で配置されています。しかし、実際のアプリケーションでは、ウィンドウの大きさを変えても、UI パーツは、適切にレイアウトされますよね。このようなレイアウトの調整を行うには、レイアウトを調整するためのプロパティを設定します。

「プロパティ」とは、コンポーネントの動作や外観などをカスタマイズできるさまざまなパラメータです。画面左下の「オブジェクト インスペクタ」には、コンポーネントに用意されたさまざまなプロパティの一覧が表示されています。C++Builder では、コードを記述しなくても、プロパティを設定するだけで簡単にコンポーネントの外観や振る舞いをカスタマイズできるのです。

では、最初に **Panel1** のレイアウトを調整してみましょう。「構造」ペインで、**Panel1** をクリックします。すると、「オブジェクト インスペクタ」の「プロパティタブ」内に、**Panel1** のプロパティが表示されます。ここから、「**Align**」を見つけ、右側の値列のドロップダウンリストをクリックし、**画面上部（alTop）**を選びます。



同様の操作で、Button1、Edit1、WebBrowser1 の「Align」プロパティも設定します。

Button1

項目	値
Align	左側（alLeft）

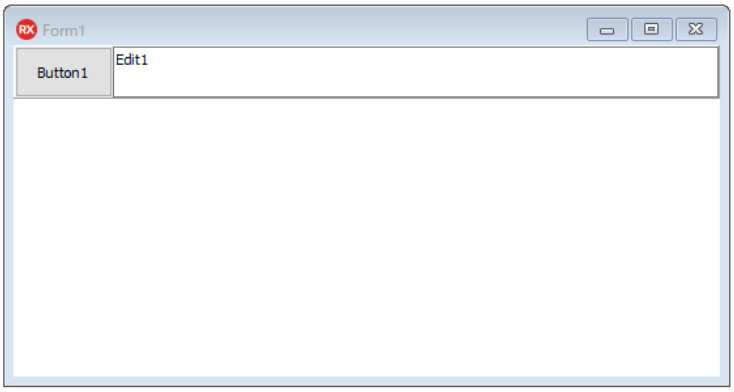
Edit1

項目	値
Align	画面中央（alClient）

WebBrowser1

項目	値
Align	画面中央（alClient）

以上で、フォームは次のような外観になります。



フォームの大きさを変更しても、Button1、Edit1 は適切な場所に配置されていますね。

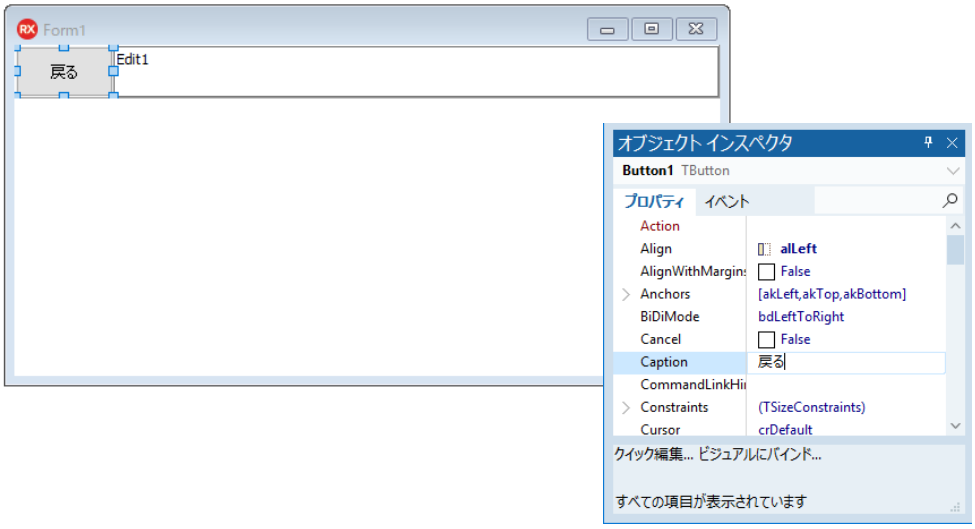
ボタンのキャプションを設定する

プロパティは、ボタンのキャプションを設定するためにも使えます。Button1 をクリックして選択状態にしたら、次のように「Caption」プロパティを設定します。

Button1

項目	値
Caption	戻る

この操作で、フォームのボタンのキャプションは、「戻る」に変わります。C++Builder では、プロパティの変更を設計フォームで直ちに確認できるため、直感的な開発が可能なのです。



さて、この［戻る］ボタンの機能はどのように実装するのでしょうか？ここで登場するのが「イベント」です。でも急がずに、簡単なところから始めてみましょう。

イベントを使う

通常、Web ブラウザは、起動するとホームページに設定されたページが表示されます。フォーム上に配置した TWebBrowser には、設定されたホームページを表示する機能が用意されています。ホームページは、Windows のインターネットオプションで設定されている URL です。

では、このホームページの表示は、いつ、どのように呼び出すのでしょうか？

「プロパティ」は、コンポーネントの状態を設定したり、参照することができます。例えば、Width プロパティは、そのコンポーネントの幅を表します。この値を参照すれば現在の幅が分かりますし、値を変更すれば実際に幅が変わります。

一方、コンポーネントの機能を呼び出すためには、「メソッド」が用意されています。TWebBrowser の `GoHome` メソッドは、「ホームページを表示しなさい」という指令をコンポーネントに送ります。`GoHome` メソッドを呼び出すコードを記述することで、ホームページが表示されるというわけです。

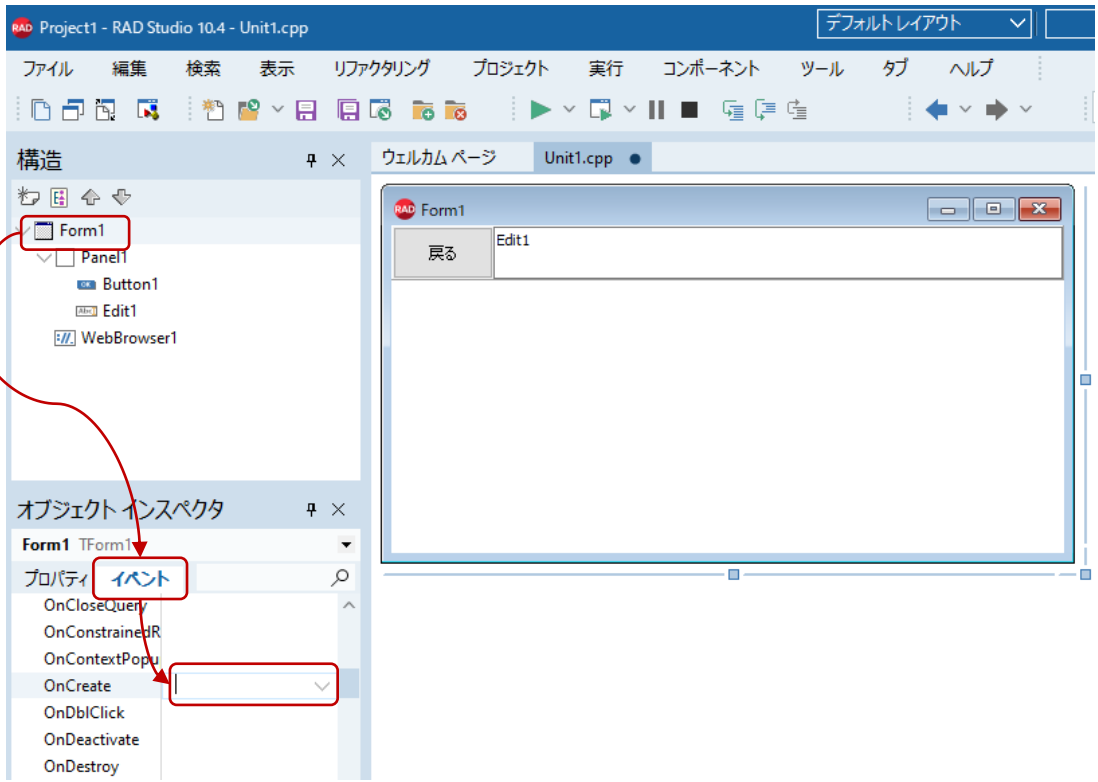
では、いつこれを呼び出せばいいのでしょうか？

そのために用意されたものが「イベント」です。コンポーネントには、複数のイベントが用意されています。例えば、ボタンをクリックすると「OnClick」イベントが発生します。マウスが上を通過すると「OnMouseOver」イベントが発生します。このように、コンポーネントに対してなされた「何か」によって、そのイベントが発生するのです。

そして、このイベントに対して「イベントハンドラ」と呼ばれるプログラムコードを記述することができます。これにより、さまざまなカスタムコードを定義できるのです。

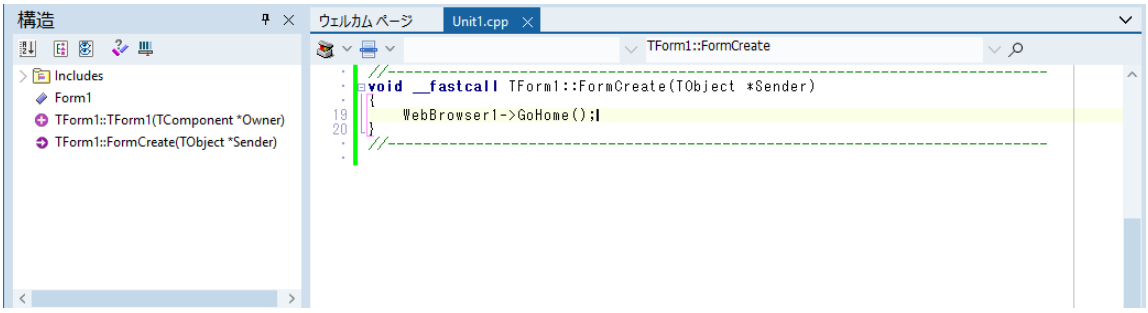
今回は、フォームが作られたときに、ホームページを表示させます。つまり、Form の「OnCreate」イベントです。

これを行うには、Form1 をクリックして「オブジェクト インスペクタ」の「イベント」タブ内の「OnCreate」の右横の空白部分（値列）をダブルクリックします。



値列に FormCreate と入力され、コードエディタが表示されます。これがイベントハンドラです。イベントハンドラの必要なコードは C++Builder が自動的に生成してくれるので、{ と } で囲まれた中身だけを記述すればいいのです。ここでは、次のように WebBrowser1->GoHome(); と、一行記述します。

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
   WebBrowser1->GoHome();
}
```



間違ったイベントをダブルクリックして、必要のないイベントハンドラができてしまっても気にしないでください。C++Builder は、空白のイベントハンドラを自動的に削除してくれます。手作業で削除すると、自動生成したコードと一致しなくなる恐れがあるので、やめたほうがいいでしょう。

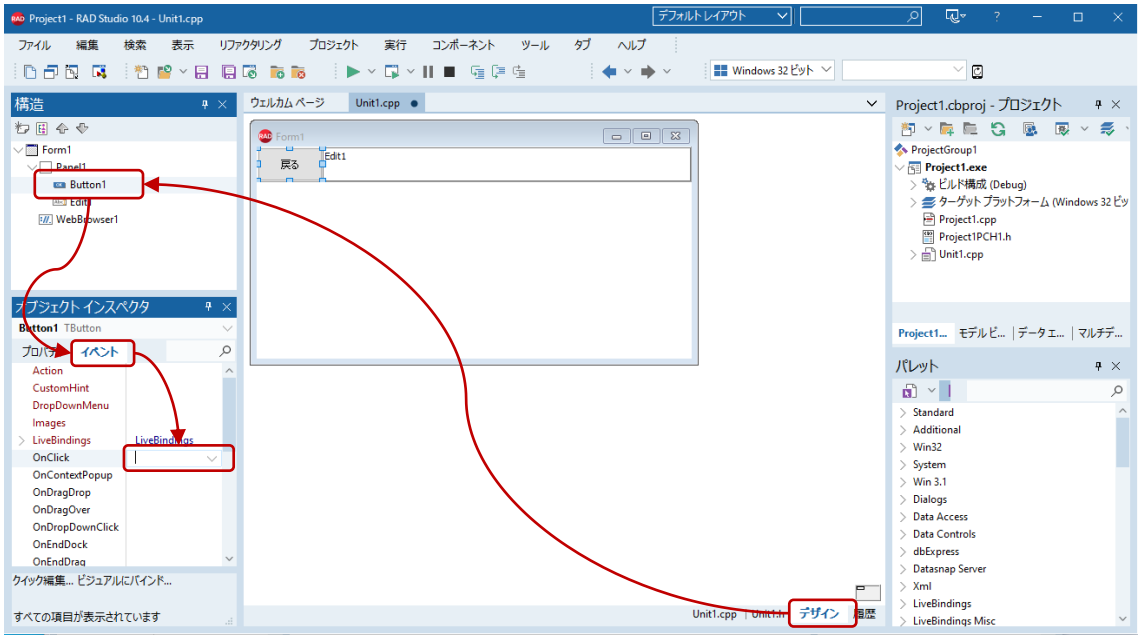
［戻る］ボタンに機能を実装する

では、この要領で［戻る］ボタンにも機能を実装してみましょう。

「戻る」という機能は、TWebBrowser の GoBack メソッドを使うだけです。このように、コンポーネントには、あらかじめいろいろな機能が用意されているので、コード量は各段に少なくなりますよね。これも C++Builder の利点のひとつなのです。

先ほど、コードエディタを使ったので、設計フォームは隠れてしまいました。再び設計フォームを表示するには、画面右下の「デザイン」タブをクリックします。

Button1（［戻る］ボタン）を選択したら、オブジェクトインスペクタの「イベント」タブ内にある「OnClick」の値列をダブルクリックします（もう、ボタンをクリックしたときの処理を実装するために、OnClick イベントを選択したことはお分かりですね）。



Button1Click のイベントハンドラには、以下のようにコードを記述します。

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    WebBrowser1->GoBack();
}
```

ところで、［戻る］ボタンを押したときに、戻るページがない（履歴がない）場合にはどうなるでしょうか？この場合はエラーが発生してしまいますが、一般的には、そのようなことのないように履歴がないときには、［戻る］ボタンが押せないように無効になっていますよね。

もちろん C++Builder でもそのように設定することができますが、今回は割愛してアプリケーションの完成を急ぎましょう。興味のある方は、あとで実装してみてください。

URL を入力してページを表示する

多くのブラウザがそうであるように、フォーム上部に配置した Edit1 に URL を入力して［Enter］キーを押すと、そのページが表示されるようにしようと思います。

Edit1 でキーが押されたときに発生するイベントは、OnKeyDown です。このイベントハンドラには、押されたキーの情報が渡されます。Enter キーかどうかは、あらかじめ定義された値「vkReturn」を使って調べます。では、この値はどこで定義されているのでしょうか？

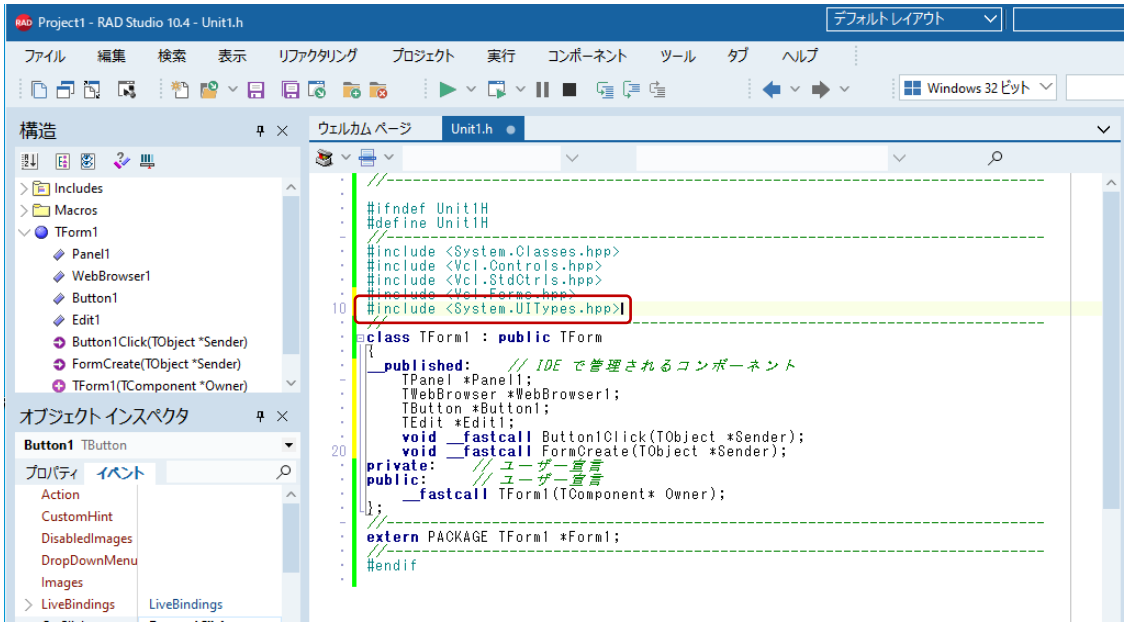
ちょっと視野を広げて、C++Builder のソースコード全体を眺めてみましょう。今回作成したアプリケーションのユニットは Unit1.cpp と言うファイル名ですが、cpp ファイルとは別に Unit1.h ファイルという名前のファイルもあり 2 つのファイルで構成されています。

```
#ifndef Unit1H
#define Unit1H
//---------------------------------------------------------------------------
#include <System.Classes.hpp>
#include <Vcl.Controls.hpp>
#include <Vcl.StdCtrls.hpp>
#include <Vcl.Forms.hpp>
#include <SHDocVw.hpp>
#include <Vcl.ExtCtrls.hpp>
#include <Vcl.OleCtrls.hpp>
//---------------------------------------------------------------------------
class TForm1 : public TForm
{
__published:    // IDE で管理されるコンポーネント
private:    // ユーザー宣言
public:     // ユーザー宣言
    __fastcall TForm1(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TForm1 *Form1;
//---------------------------------------------------------------------------
#endif
```

Unit1.h ファイル中には、インクルード指定とクラスの定義が記述されています。

`#include` は、他のユニットや C++ソース内容を取り込む場合に利用します。その後に書かれている `class TForm1` はフォームの宣言部です。つまり、先ほどビジュアル操作で設計していたフォームの定義です。`Unit1.cpp` では、`Unit1.h` に記述された `class` 宣言をもとに、他のユニットからアクセスできない実装部を記述しています。

インクルード指定に戻りましょう。`vkReturn` が定義されているのは、`System.UITypes.hpp` です。このファイルはインクルード指定に含まれていませんので、この値を使えるようにするために、`#include <System.UITypes.hpp>` を 1 行追加します。



設計フォームを表示して `Edit1` を選択し、「`OnKeyDown`」イベントを設定します。`Edit1KeyDown` のイベントハンドラを次のように記述します。

```
void __fastcall TForm1::Edit1KeyDown(TObject *Sender, WORD &Key, TShiftState Shift)
{
    if (Key == vkReturn) {
        WebBrowser1->Navigate(Trim(Edit1->Text));
    }
}
```

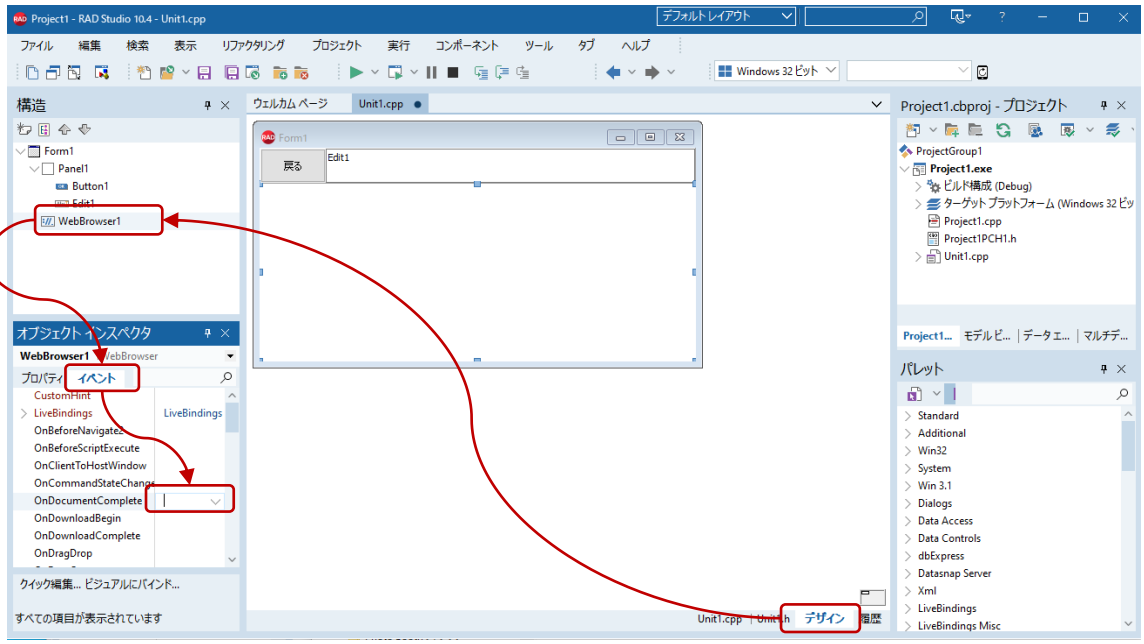
ここまでで Web ブラウザの基本的な動作を実装することができました。簡単でしたね。

ではもう少しだけ機能を追加して、より本物に近づけてみましょう。

表示されたページの URL とタイトルを表示する

URL を入力する `Edit1` は、現在の URL を表示するボックスとしても機能しなければなりません。そのため、ページが表示されたら、その URL を `Edit1` に設定し直すという動作が必要です。これを実装するのにちょうどよいイベントが、TWebBrowser に用意されています。`OnDocumentComplete` です。

`WebBrowser1` を選択して、「`OnDocumentComplete`」イベントのイベントハンドラを作成します。



作成されたイベントハンドラ `WebBrowser1DocumentComplete` に、以下のコードを記述します。

```
void __fastcall TForm1::WebBrowser1DocumentComplete(TObject *ASender,
    IDispatch * const pDisp, const OleVariant &URL)
{
    Edit1->Text = WebBrowser1->LocationURL;
}
```

もうひとつ実装すべきなのは、タイトルの表示です。Web ページにはタイトルが定義されていますが、多くの Web ブラウザでは、ウィンドウのタイトルに、ページのタイトルを表示するようにしています。

これを行うには、TWebBrowser の OnTitleChange イベントを使います。WebBrowser1 を選択して「OnTitleChange」イベントのイベントハンドラを作成、次のようにコードを記述します。

```
void __fastcall TForm1WebBrowser1TitleChange(TObject ASender,
    const WideString Text)
{
    this->Caption = Text;
}
```

さて、ここで新しい要素 this が出てきました。this は自分自身を表す変数です。ここでいう自分自身とは、TForm1 です。

今編集している Unit1.cpp の宣言部である Unit1.h には、TForm1 というクラスが宣言されています。TForm1 は、現在作成しているフォームです。C++Builder では、コンポーネントフレームワークを使うことで、他の C++開発ツールとは異なり、複雑なコードを組むことなく、ユーザーインターフェイスを構築できます。

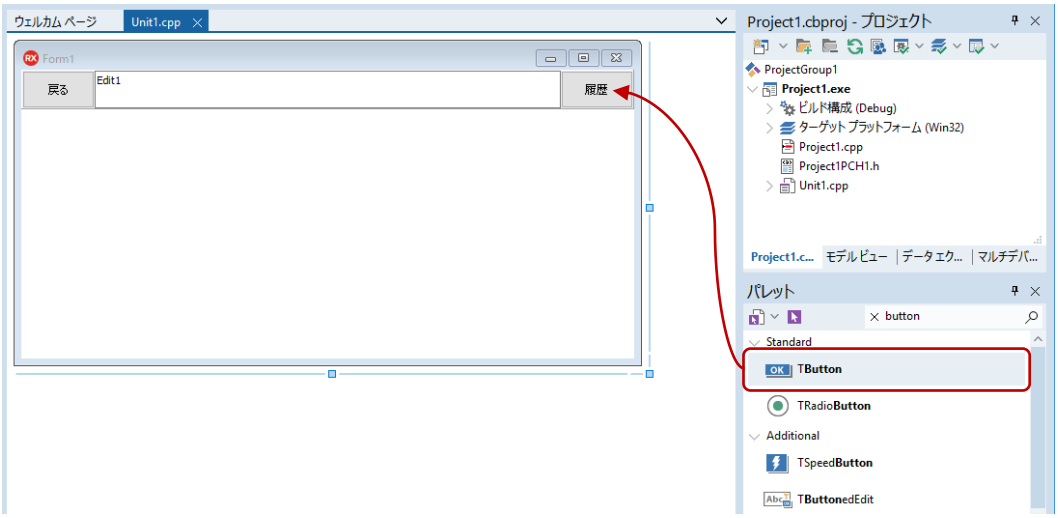
TForm1 は、TForm というからっぽのフォームを継承しています。Web ブラウザの機能を実装するために、ウィンドウの基本的な機能を作らなくてよかったのは、TForm の機能を利用してきたからです。ここで詳細に説明することはしませんが、C++Builder のコンポーネントフレームワークの構成を深く知るには役立つ知識です。

コードの説明に戻りましょう。this->Caption という記述は、自分自身の Caption プロパティという意味になります。TForm の Caption プロパティは、ウィンドウのタイトルを表しますので、ここに渡された Text（新しいタイトル）を設定すれば、ページのタイトルを表示することになるのです。

ブラウズ履歴を表示する

簡単な 1 行コードを記述し、フォーム画面タイトル表示することができました。さらに本格的ブラウザのようなアプリに仕上げていきしょう。次はブラウザにアクセスした URL の履歴をメモリに保存し、履歴ボタンをクリックすることで過去アクセスしたリストを呼び出します。C++の標準ライブラリを使って機能を実装していきますが、前章のようなシンプルな 1 行加えたようなコードでは実現できません。少し複雑になりますが履歴機能を作っていきましょう。

まず TButton を Panel1 上に配置し、履歴ボタンとします。



履歴ボタンのプロパティを設定します。

Button2

項目	値
Align	alRight
Caption	履歴

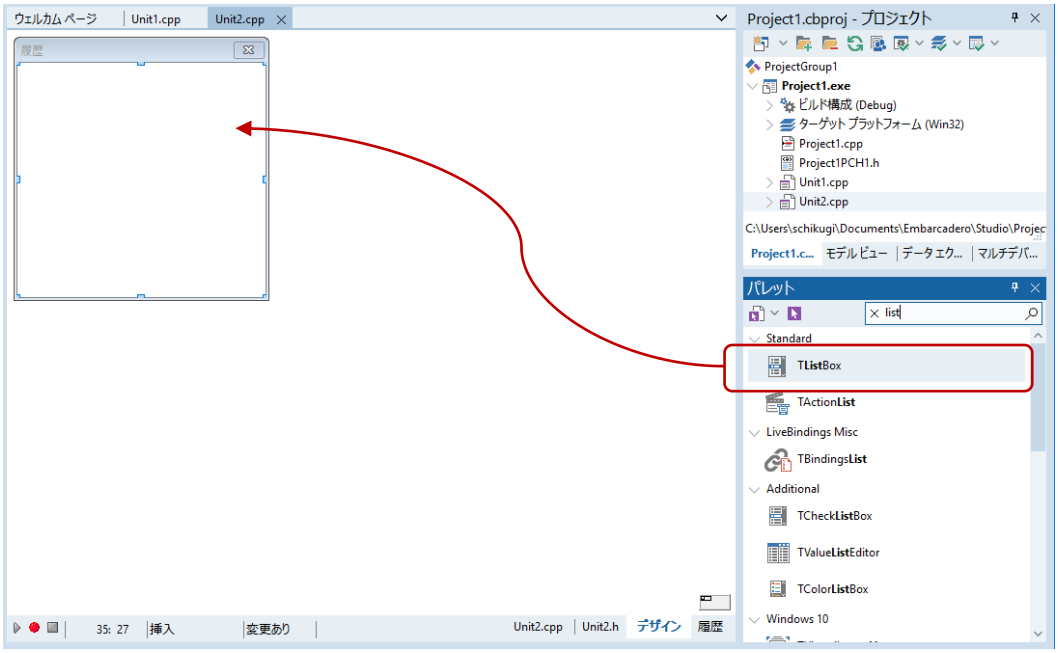
履歴リストを表示するために新しいフォームを作成します。メニューから［ファイル(F)｜新規作成(N)｜VCL フォーム］を選択すると、中央のフォームデザイナー画面に新たにフォーム（Unit2.cpp）が作られます。

まず Unit2.cpp フォームのプロパティを設定しましょう。

Form2

項目	値
BorderStyle	bsDialog
Caption	履歴
Width	265
Height	274

Unit2.cpp（Form2）に URL 履歴を表示するため TListBox コンポーネントを配置します。

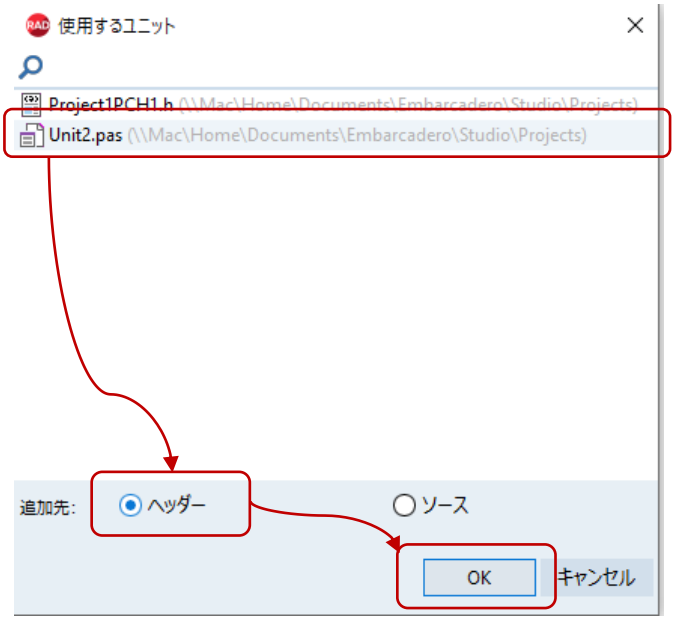


配置された TListBox のプロパティ設定を行います。Form2 画面全体に ListBox 表示したいので、Align を変更しましょう。

ListBox1

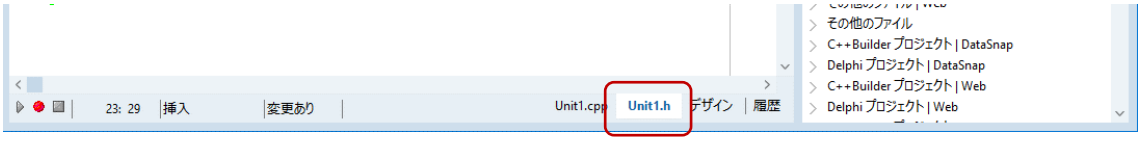
項目	値
Align	alClient

フォームデザイナー画面の上部タグをいったん `Unit1.cpp` に切り換え、さらにメニューから［ファイル(F) | 使用するユニット(U)］(Alt+F11) を選択すると、以下のダイアログが表示されます。



ダイアログ内のリスト `Unit2.cpp` を選択し、追加先で「ヘッダー」を選択したら［OK］ボタンをクリックします。これにより、`Unit1.cpp` から `Unit2.cpp` を参照できるようになります。

続いて C++コードを記述していきます。まず、フォームデザイナー上部のタブを切り換え、`Unit1.cpp` を選びます。次に画面下のタブで、`Unit1.h` を選択します。



C++ヘッダーが記述されたエディタ画面に切り替わります。#include という記述が先頭付近にたくさん書かれています。その最後に以下の 2 行を追加します。

```
#include <vector>
#include <memory>
```

記述した `vector` と `memory` は、C++の標準ライブラリの一部の型を呼び出すための宣言です。

次に履歴を収納するための動的配列 vector<T>型を宣言してみましょう。エディタ画面の中央行あたりに private:と書かれた行があります。その直下に以下のコードを記述します。

```
private:    // ユーザー宣言
    std::vector<UnicodeString> History;
```

以上で、C++コードを記述するための準備は完了です。

それでは Unit1.cpp イベントハンドラ内のコードを記述していきましょう。前章では TWebBrowser の OnTitleChange イベントで 「OnTitleChange」イベントのイベントハンドラを作成しフォームタイトルを表示しました。

その続きのコードを次のように記述します。

```
void __fastcall TForm1::WebBrowser1TitleChange(TObject *ASender,
    const WideString Text)
{
    this->Caption = Text;//ここは前章で記述したコードです。
    UnicodeString LastURL = History.empty() ? UnicodeString() : *History.rbegin();

    if (LastURL != WebBrowser1->LocationURL) {
        History.push_back(WebBrowser1->LocationURL);
    }
}
```

続いてブラウザ URL 履歴を呼び出せるように、履歴ボタンのイベントハンドラを追加しましょう。

ボタンイベントハンドラの作成手順は同様です。履歴ボタンを選択し、オブジェクトインスペクタからイベントタグで OnClick 項目をダブルクリックします。履歴画面を呼び出すためのコードは、次のように記述します。

```
void __fastcall TForm1::Button2Click(TObject *Sender)
{
    std::unique_ptr<TForm2> fHistory(new TForm2(this));

    fHistory->Position = poMainFormCenter;
    fHistory->ListBox1->Clear();
```

```
    std::vector<UnicodeString>::iterator iter;
    for (iter = History.begin() ;iter != History.end(); iter++) {
        fHistory->ListBox1->Items->Add(*iter);
    }

    fHistory->ShowModal();
}
```

少し難しいコードですが、Unit2.cpp のフォームデザイナーで作成した ListBox1 が配置されたフォームが表示され、リスト内に過去にアクセスした URL の履歴が入ったウィンドウを表示します。

ここでは、C++特有のコードを使っています。include <memory>で呼び出した unique_ptr<TForm2>型はスマートポインタの一種で、必要なくなったメモリを自動破棄してくれる型です。TForm2 を新たに作っていますが、必要がなくなった時点でメモリは解放されます。

もう 1 箇所は、最初にユーザー宣言した vector<UnicodeString> History に保存した内容を ListBox1 に追記していくコードです。

アプリケーションを実行し、履歴ボタンをクリックすると、図のように URL 履歴を表示するウィンドウが現れます。このウィンドウはフォームなので、「X」ボタンで終了します。

新しいウィンドウでページを表示する

TWebBrower には、標準的な Web ブラウザ機能の多くがあらかじめ用意されているので、ブラウザ上でマウスを右クリックして［新しいウィンドウで開く］といったメニューを選択することもできます。この動作はまだ定義していません。このままでは、別のブラウザを表示してしまうので、新たにもうひとつフォームを表示するように記述してみましょう。

「TForm1 はクラスである」と説明しました。クラスのインスタンスが、実際に表示されているフォームです。アプリケーションが起動すると、最初に TForm1 のインスタンスがひとつ作成され、表示されます。この部分のコードは記述していませんが、C++Builder が自動的に作成してくれているのです。

```
#include <vcl.h>
#pragma hdrstop
#include <tchar.h>
//---------------------------------------------------------------------------
USEFORM("Unit1.cpp", Form1);
//---------------------------------------------------------------------------
int WINAPI _tWinMain(HINSTANCE, HINSTANCE, LPTSTR, int)
{
    try
    {
        Application->Initialize();
        Application->MainFormOnTaskBar = true;
        Application->CreateForm(__classid(TForm1), &Form1);
        Application->Run();
    }
    catch (Exception &exception)
    {
        Application->ShowException(&exception);
    }
    catch (...)
    {
        try
        {
            throw Exception("");
        }
        catch (Exception &exception)
        {
            Application->ShowException(&exception);
```

```
        }
    }
    return 0;
}
```

このコードは通常意識する必要はありません（画面右側の「プロジェクト マネージャ」で「Project1.exe」を選択して右クリックし［ソースの表示(V)］を選択すると表示されます）。

「新しいウィンドウを開く」という動作を実装するには、フォームをもうひとつ、つまり TForm1 をもうひとつ作成する必要があります。これには new 演算子を呼び出します。［新しいウィンドウで開く］メニューが選択されると、TWebBrowser の「OnNewWindow3」イベントが発生します。これに対応するコードを記述するために、WebBrowser1 の OnNewWindows3 イベントにイベントハンドラを記述します。

記述するコードは、新しいフォームを作成し、これを表示します。このためには、作成したフォームを変数に代入する必要があります。C++では、関数内のローカル変数はスコープ内に宣言できます。ここでは、newForm というローカル変数を、イベントハンドラのスコープ内に宣言します。

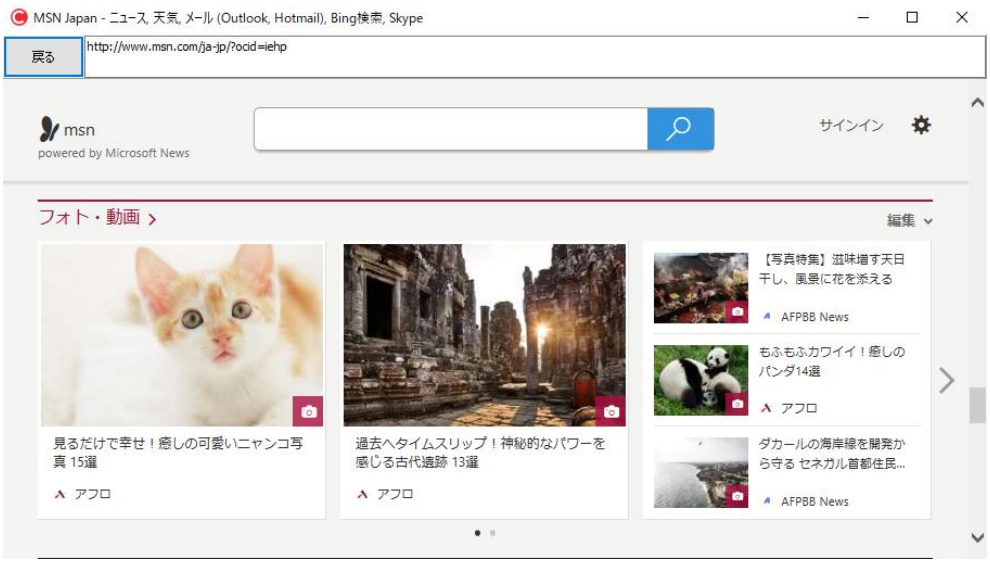
イベントハンドラのコード全体は、以下のようになります。

```
void __fastcall TForm1::WebBrowser1NewWindow3(TObject *ASender, IDispatch *&ppDisp,
       WordBool &Cancel, DWORD dwFlags, const WideString bstrUrlContext,
       const WideString bstrUrl)
{
    TForm1* newForm(new TForm1(this));
    newForm->Show();
    newForm->WebBrowser1->Navigate(bstrUrl);
    Cancel = true;
}
```

ここまでの作業が完了したら、［ファイル | すべて保存］でユニット、プロジェクトファイルを保存しておきましょう。これまでファイル名として Unit1、Project1 などのデフォルトの名前を使ってきましたが、BrowserUnit、MyBrowser などの名称を付けておくといいでしょう。

パーソナル Web ブラウザの完成

以上で、おおよそ Web ブラウザらしい動作が実装できました。F9 キーを押すか、［実行(R)｜実行(R)］メニューを選択して、プログラムを実行します。



しかし、まだまだ検討の余地は残っているようです。例えば、Java Script エラーが発生したときの処理など。また、多くの Web ブラウザでは、メインフォームという概念はなく、いくつもウィンドウを開いて、どの順番に閉じても構わないという動作をします。しかし、今回の実装では、最初のウィンドウをメインウィンドウとしており、複数のウィンドウを開いているときに、メインを閉じると、すべてのウィンドウが閉じられ、アプリケーションが終了してしまいます。

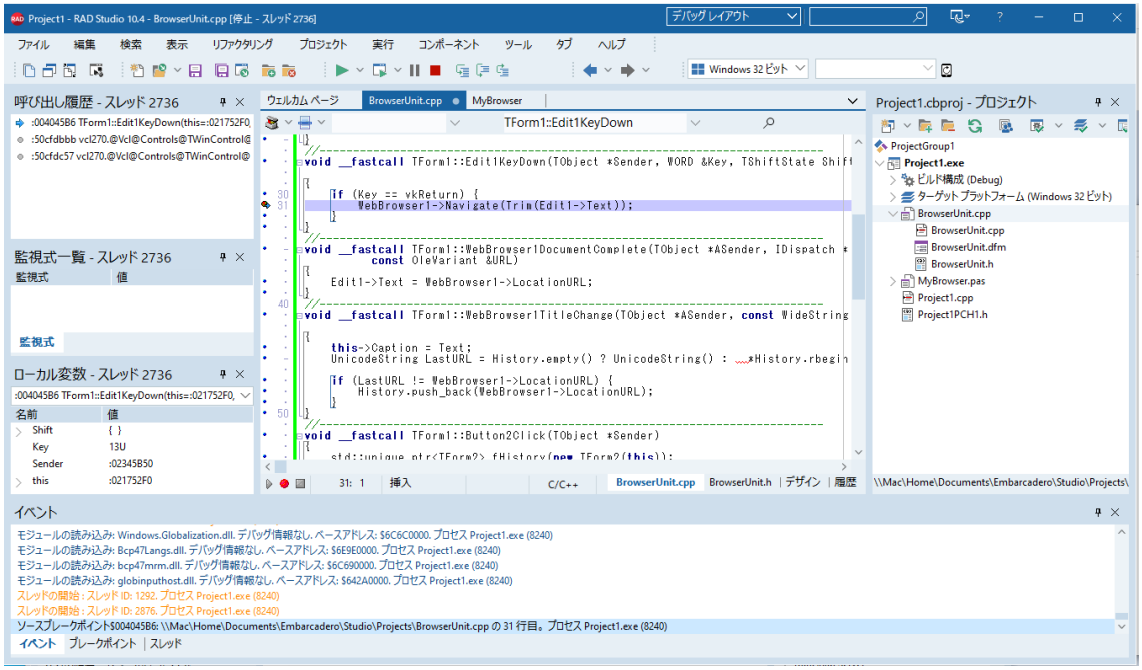
あらゆる操作を想定してプログラムを記述することは、困難なことかもしれません。しかし、C++Builder にはさまざまなエラーハンドリングのしくみが用意されているので、比較的容易にエラーに対応するコードを実装することができます。

デバッガを使ってみよう

C++Builder の統合開発環境には、ソースコードレベルでアプリケーションをステップ実行したり、変数の内容を監視できるビジュアルデバッガが搭載されています。デバッガを使えば、プログラムが意図したとおりに実行されているか、変数に正しく値が設定されているかを確認し、容易にバグを取り除くことができます。

作成した Web ブラウザアプリケーションをデバッグ実行してみましょう。メインメニューで［実行(R)｜実行(R)］を選択します。

ソースコード中のイベントハンドラ `TForm1::Edit1KeyDown` の最初の行の左余白をクリックして、ブレークポイントを設定します。アプリケーションで、`Edit1` に何か入力すると、ブレークポイントを設定した行で実行が停止します。

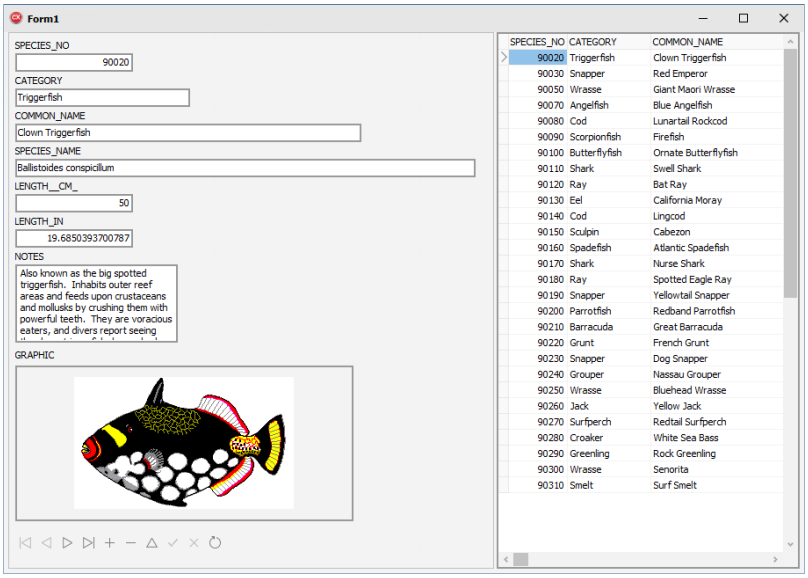


監視式に、引数として渡された `Key` などの変数を追加すると、現在どのような値がイベントハンドラに渡されているかを確認することができます。

データベースを利用してみよう

C++Builder と Delphi には、データベースアクセスのための強力な機能が搭載されています。これは、Delphi が元々Oracle データベースにアクセスするアプリケーションの開発を想定して開発されたことにも由来します。

最新バージョンでは、FireDAC と呼ばれるマルチデバイス対応の共通データベースアクセスフレームワークが用意されており、多様なデータベース（Oracle、SQL Server、IBM Db2、Sybase、InterBase、MySQL、PostgreSQL、Access、MongoDB など）に高速かつ直接、ネイティブアクセスできます。



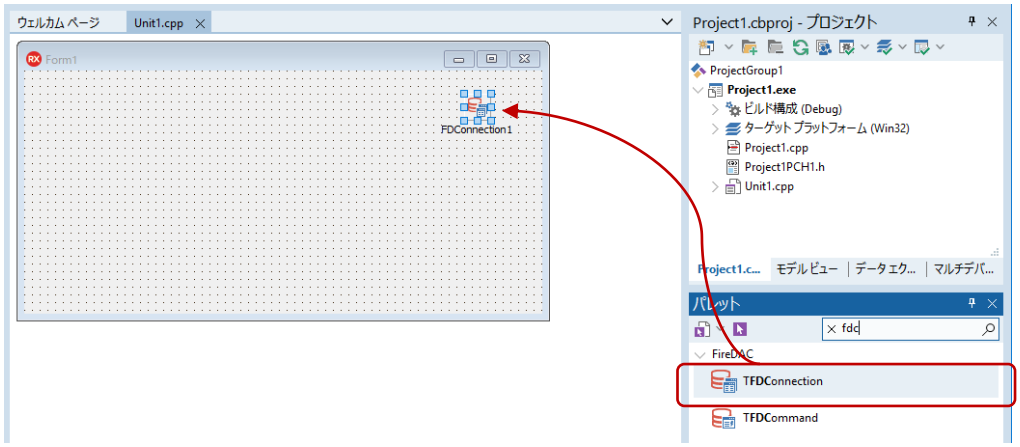
今回は、ご覧のような「魚図鑑」データベース（古くから Delphi や C++Builder のサンプルとしておなじみのものです）を使って、FireDAC によるデータアクセスの方法を紹介します。他のデータベースを使う場合も、基本的な手順は変わりません。

BDE などの古いデータベースアクセスの方法をご存じの方は、おなじみのデータベースを利用するアプリケーションの作成を通して、最新の FireDAC でどのようにデータアクセスを行うのか、その基本的な手順を理解できると思います。

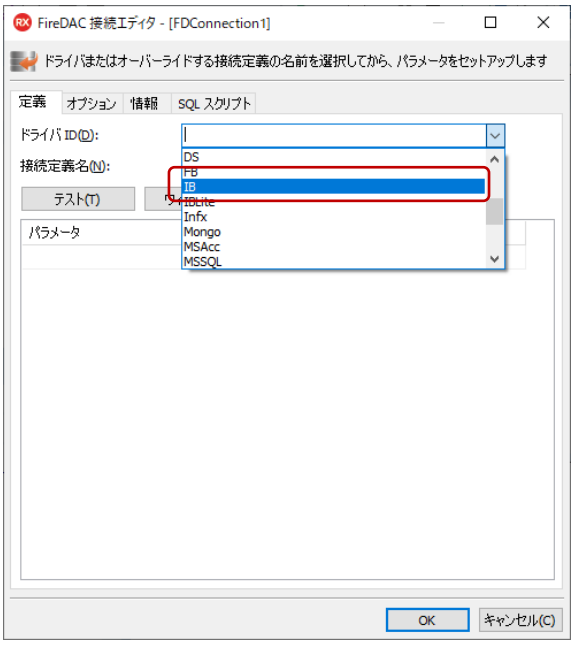
データベース接続を定義する TFDConnection

［ファイル(F)｜新規作成(N)｜Windows VCL アプリケーション - C++Builder］を選択し、新規アプリケーションを作成します。

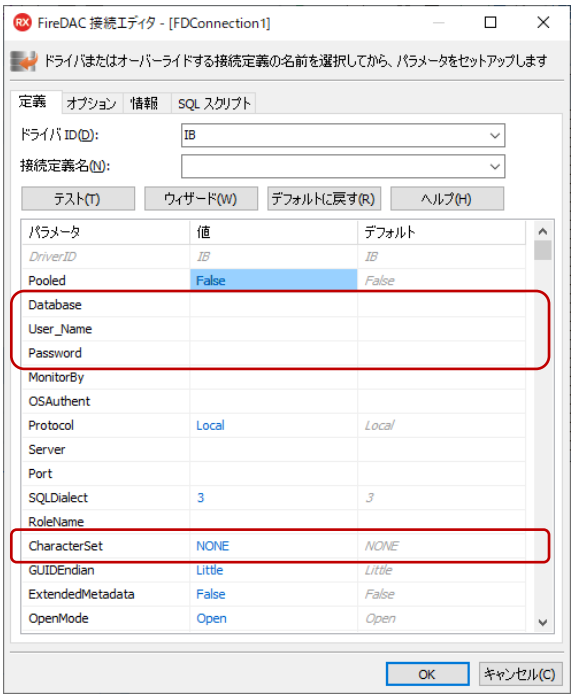
ツールパレットから、FireDAC カテゴリにある、TFDConnection コンポーネントをフォーム上にドラッグ&ドロップします。



配置した TFDConnection（**FDConnection1**）をダブルクリックすると、FireDAC 接続エディタが表示されるので、「ドライバ ID」項目で、接続するデータベースを選択します。今回は、「**IB**」（InterBase）を選びます。



ドライバ ID を選択すると、選択したドライバに応じて必要な設定パラメータが表示されます。InterBase の場合、この中で設定が必要となるのは、Database、User_Name、Password、CharacterSet です。



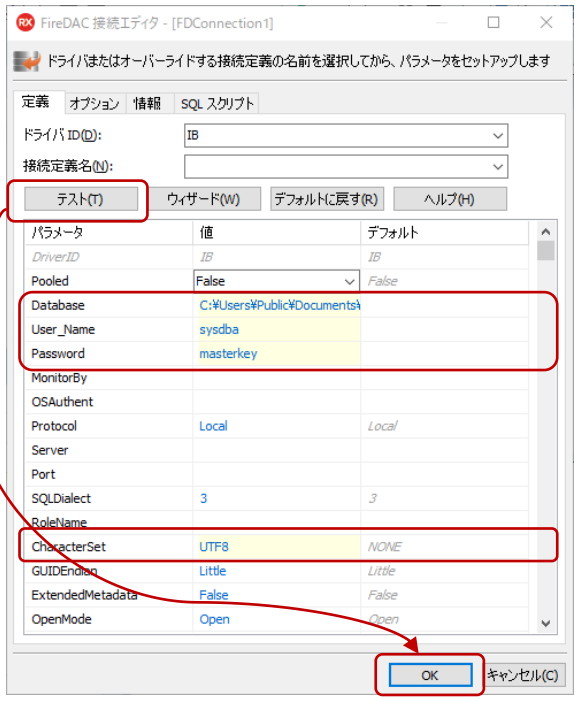
Database には、データベースファイルの場所を指定します。C++Builder をインストールする際にサンプルファイルもインストールしていると、デフォルトで以下の場所に、今回使用する InterBase のデータベースファイルが保管されています。

```
C:\Users\Public\Documents\Embarcadero\Studio\21.0\Samples\Data\dbdemos.gdb
```

各項目は以下のように設定します。

項目	値
Database	C:\Users\Public\Documents\Embarcadero\Studio\21.0\Samples\Data\dbdemos.gdb
User_Name	sysdba
Password	masterkey
CharacterSet	UTF8

設定が完了したら、［テスト］ボタンをクリックして接続テストを行います。接続に成功すれば、設定は完了です。［OK］をクリックして、接続エディタを閉じます。



このほかに、FDConnection1 に対して、いくつかのプロパティを設定します。

FDConnection1

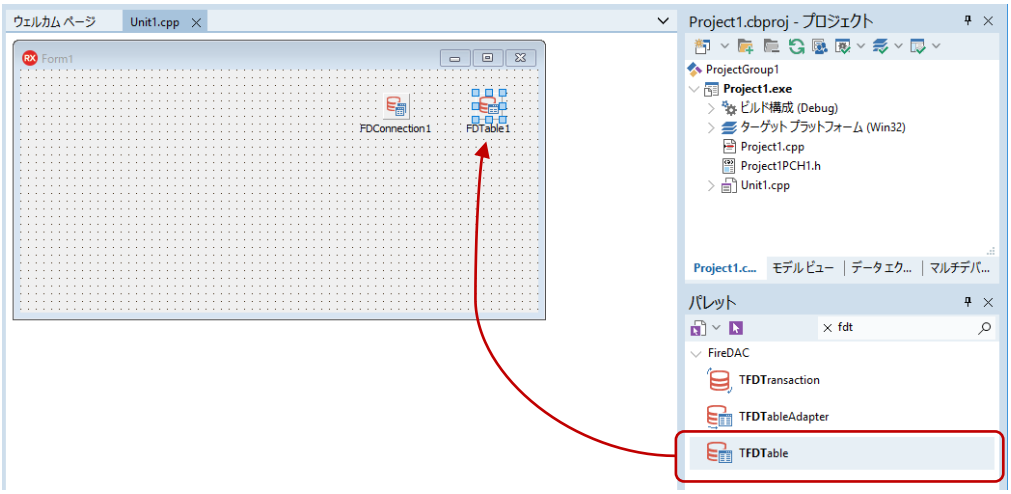
項目	値
Connected	True
LoginPrompt	False

これで、ユーザー名／パスワードを入力するログインプロンプトを表示することなく、アプリケーション起動と同時にデータベースに接続します。

TFDTable を使ってテーブルのデータセットを呼び出す

データベースのテーブルを扱うのは、TFDTable コンポーネントです。同じようなコンポーネントに TFDQuery があります。TFDQuery は、テーブルではなく、クエリー（SELECT 文）によってデータセットを取得する点が異なります。

ツールパレットから、TFDTable コンポーネントを選択し、フォーム上にドラッグ&ドロップします。

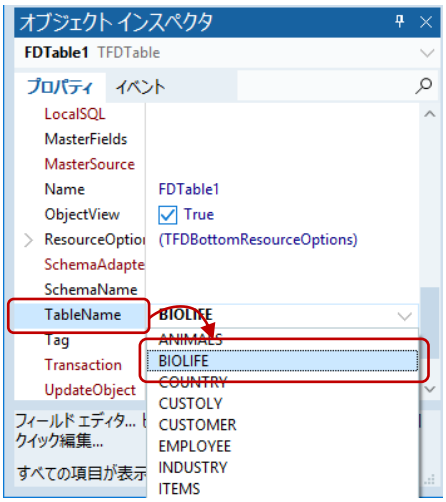


オブジェクト インスペクタで、配置した TFDTable（**FDTable1**）に対して、次のようにプロパティを設定します。

FDTable1

項目	値
TableName	BIOLIFE
Active	True

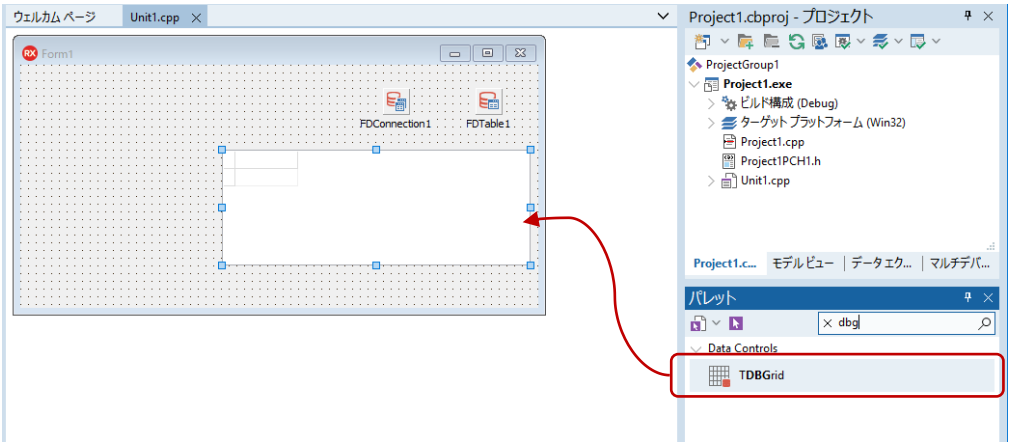
TFDTable を配置したときに、このコンポーネントの **Connection** プロパティは、自動的に **FDConnection1** に設定されているはずです。これにより、**TableName** プロパティを設定しようとすると、テーブルの一覧を取得し、下図のようにドロップダウンリストでテーブルの一覧を表示します。



ユーザーインターフェイスを設計する

データセットの準備ができたら、次にユーザーインターフェイスを作成していきます。データセットを表形式で表示、編集できる便利なグリッドコンポーネントを使いましょう。

ツールパレットから TDBGrid を選択し、フォーム上にドラッグ&ドロップします。



配置した TDBGrid（**DBGrid1**）に対して、以下のように **Align** プロパティを設定します。

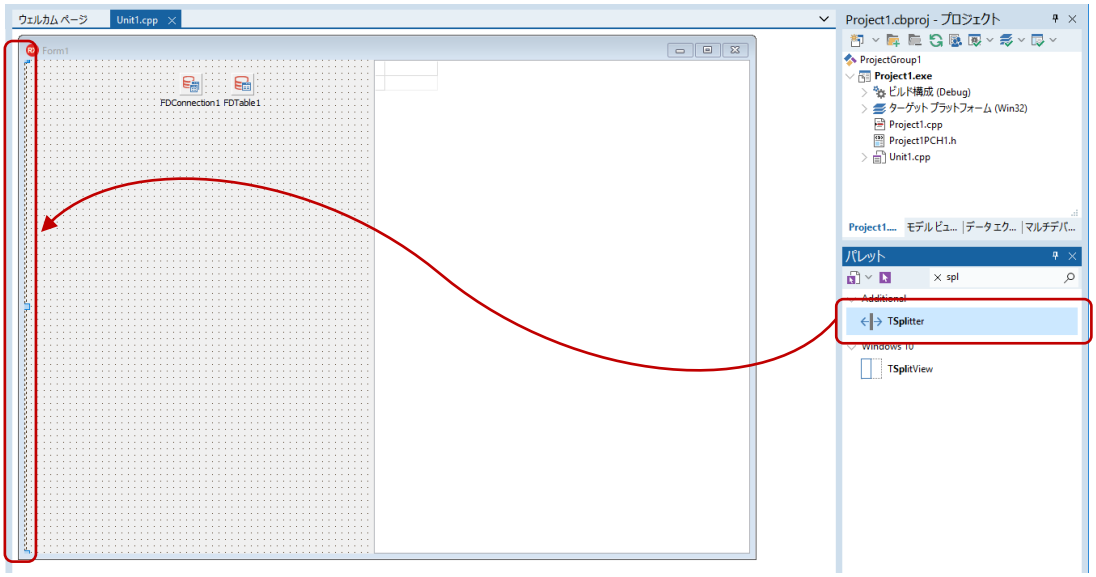
DBGrid1

項目	値
Align	alRight

Align プロパティを **alRight** に設定することで、フォーム左側の残りの部分に詳細データを表示するための領域を確保できました。ただ、ウィンドウの大きさによって、データの表示可能領域は変わってきてしまいます。そこで、左右の表示領域の大きさをユーザーが自由に変更できるようにしようと思います。

これを実現するのが、TSplitter コンポーネントです。TSplitter を使えば、左右の表示領域の分割位置をマウス操作で自由に変更できます。TSplitter は、フォームの一方の端に接しているコントロールと、それ以外のクライアント領域を占めているコントロールとの間に配置されます。ユーザーがスプリッタを移動すると、フォームの端に接しているコントロールのサイズが変化します。これによって、このフォームのクライアント領域が変化し、それに応じてクライアント領域の残りの部分を占めるコントロールのサイズが変化します。

ツールパレットから TSplitter を選択し、フォーム上にドラッグ&ドロップします。



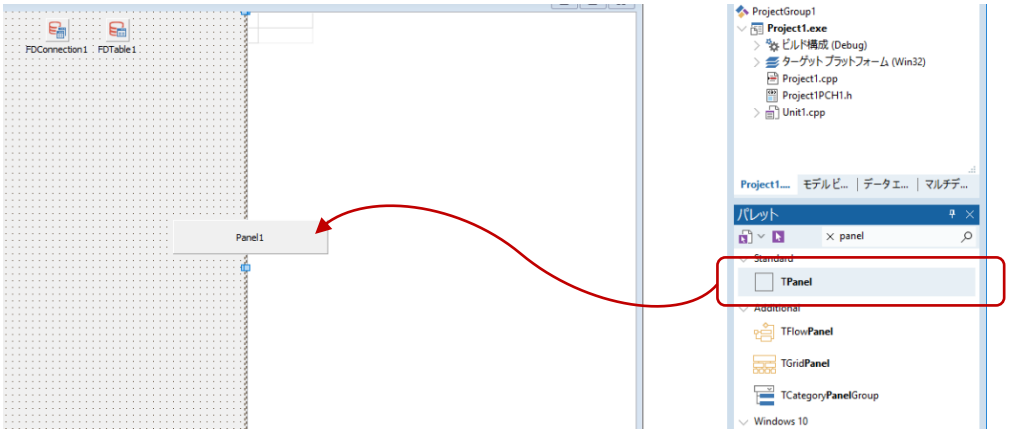
配置された TSplitter（**Splitter1**）の **Align** プロパティを次のように設定します。

Splitter1

項目	値
Align	alRight

これで、**Splitter1** は、右側のコントロール（**DBGrid1**）にくっつきました。

Splitter1 の左側には TPanel を配置します。最終的には、この上にいくつものUIコントロールを配置して、詳細データを表示するようにします。ツールパレットから TPanel を選択し、フォーム上にドラッグ&ドロップします。



配置した TPanel（**Panel1**）のプロパティを次のように設定します。

Panel1

項目	値
Align	alClient
ShowCaption	False

以上で分割線（スプリッタ）を持つユーザーインターフェイスが作成できました。後は詳細データを表示するコントロールの配置です。これには、データベースのフィールドデータに関する便利な機能を使ってみましょう。

データフィールド設定

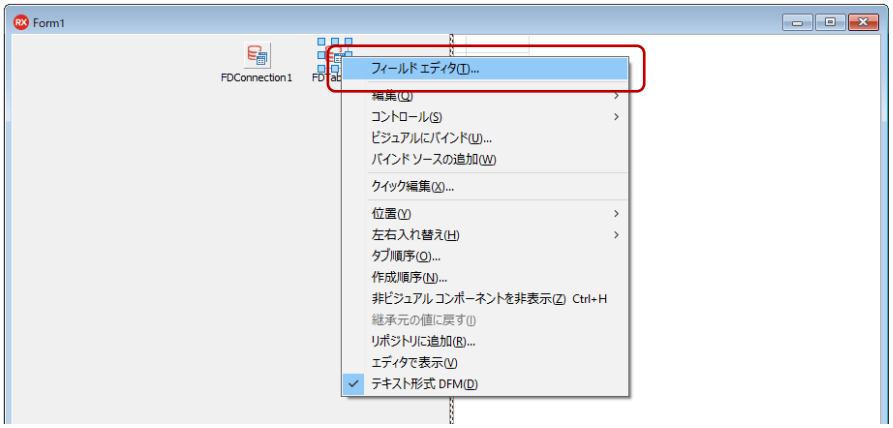
TFDTable がデータベースのテーブルを表すのに対し、テーブルの各フィールド（列）は、TField（およびその派生クラス）によって表されます。TField は、２種類の方法で作成できます。

- **静的フィールド**：設計時に「フィールドエディタ」を使って定義する
- **動的フィールド**：実行時に自動生成（TFDTable およびその他のデータセットの `Fields` プロパティや `FieldByName` メソッドを使ってアクセス）

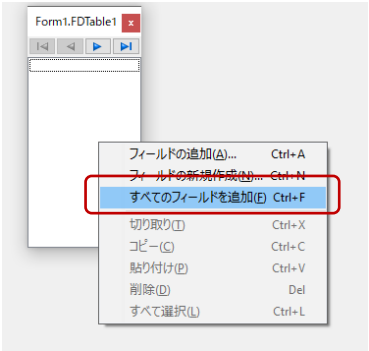
設計時に「フィールドエディタ」を使って定義すると、各フィールドのプロパティやイベントを定義することができます。例えば、`Alignment` プロパティを使ってデータの整列方法を変更したり、`OnChange` イベントを使って、フィールドデータが変更された時の処理を記述することができます。

「フィールドエディタ」では、フィールドを定義するだけでなく、定義したフィールドデータを表示するコントロールをドラッグ＆ドロップで簡単にフォーム上に配置することができます。今回は、この方法で、詳細データを表示するコントロールを配置することにします。

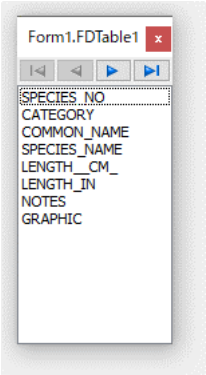
フォーム上の FDTable1 を選択し、右クリックして［フィールドエディタ］メニューを選択します。



「フィールドエディタ」が表示されるので、右クリックしてメニューを表示し、［すべてのフィールドを追加］を選択します。

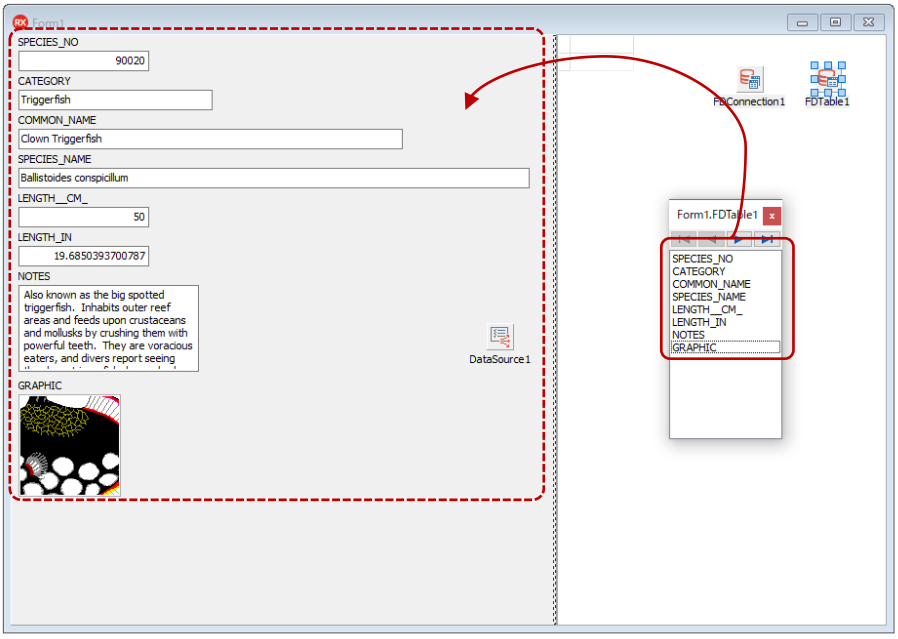


すると、次のように BIOLIFE テーブルのすべてのフィールドが追加されます。



「フィールドエディタ」または「構造」ペインで、フィールドを選択すると、そのフィールドのプロパティやイベントが、「オブジェクト インスペクタ」に表示されます。これらの値を設定することで、設計時にフィールドに関する設定やカスタマイズが可能になります。

さて、ここからがマジックのようですが、「フィールドエディタ」にリストされているフィールドをすべて選択し、設計フォーム右側に位置する **Panel1** の上にドラッグ&ドロップしてください。



たったこれだけの操作で、各フィールドに対応するラベル（TLabel）と入力ボックス（TDBEdit）、メモ（TDBMemo）、画像（TDBImage）が追加されます。これらのコントロールとデータセットをリンクするための TDataSource も、**DataSource1** として追加されます。

DataSource とは

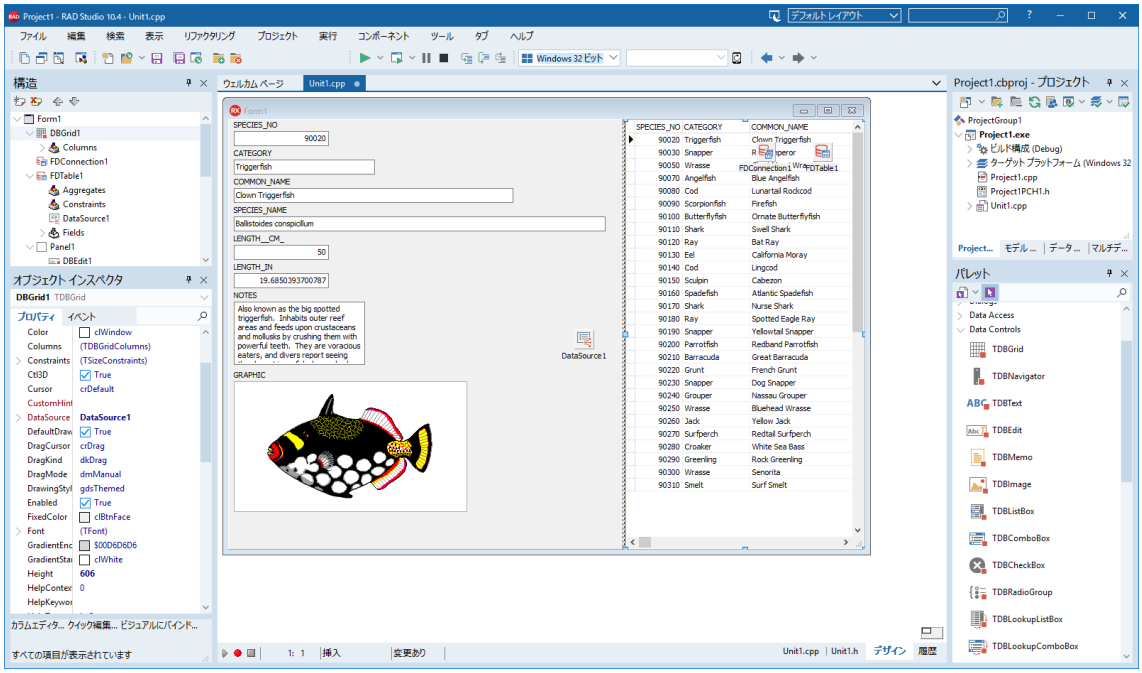
ドラッグ＆ドロップで配置したコントロールには、データベースの実データが表示されています。このように、設計時に実データを表示できるのも C++Builder の特長のひとつです。いちいちアプリケーションを実行しなくても、データを表示するのに最適なレイアウトを選べますから便利ですね。

さて、先に配置した **DBGrid1** には、まだデータが表示されていません。その理由は、**DataSource** プロパティが設定されていないからです。**DataSource** プロパティには、データセットとの仲介を行う TDataSource を指定します。今回は、BIOLIFE テーブルを表す **FDTable1** に **DataSource1** が結びつけられています（**DataSource1** は、先ほどのドラッグ＆ドロップ操作で自動生成されました）。

DBGrid1 の DataSource プロパティにも、同じ値を設定します。

DBGrid1

項目	値
DataSource	DataSource1

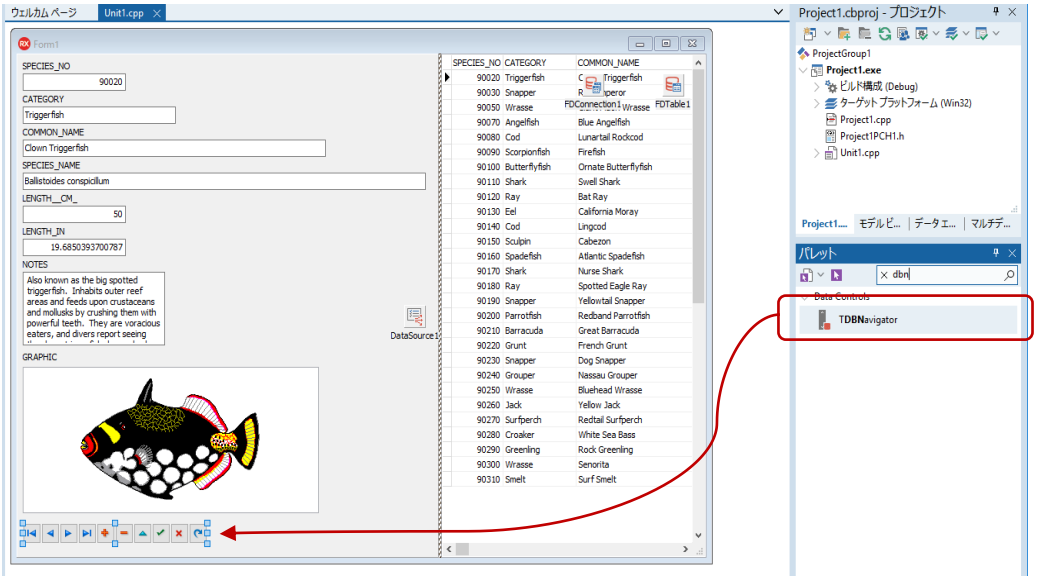


レコードの移動と操作

グリッドとは異なり、詳細データでは、現在のレコードの情報のみが表示されます。レコードを移動させたり、データの更新などを行うには、データセットに対して、レコードの移動や更新を行うメソッドを呼び出す必要があります。

こうした処理をボタンに割り当て、イベントハンドラやアクションをいちいち定義するのは面倒です。そこで、データセットの主要な操作をまとめたボタンセットのコンポーネント TDBNavigator を使います。TDBNavigator を用いれば、データセットにリンクされた TDataSource を指定するだけで、上記のような操作を実装できます。

ツールパレットから TDBNavigator を選び、フォーム上の **DBImage1** 画像の下あたりに配置します。



次のようにプロパティを設定すれば、ボタンが動作します。

DBNavigator1

項目	値
DataSource	DataSource1

データにコードからアクセスするには

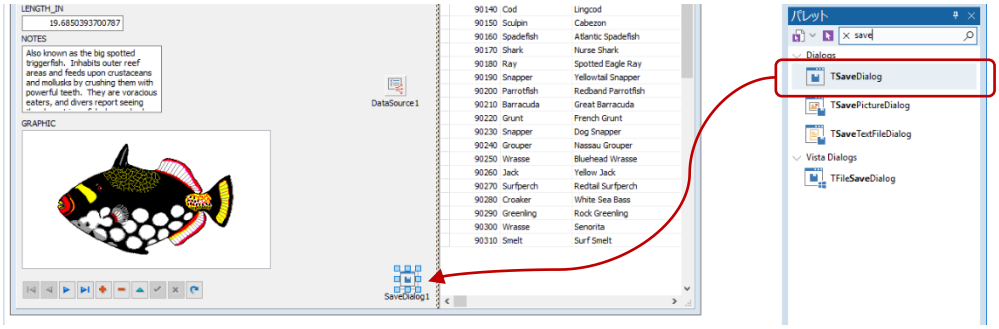
C++Builder では、ビジュアルコンポーネントとデータアクセスコンポーネントを結び付け、ユーザーインターフェイスとデータをリンクし表示させることができます。コードを記述する必要がまったくないことに驚かれたかもしれません。

しかし、実際のプログラムでは、データを操作したり、複雑な処理を実装する必要があるでしょう。C++Builder のコンセプトは、簡単でありながらも高度なことも実現できるようにすること。実は、ここで使用してきたデータアクセスコンポーネントについても、コードによって直接操作できるのです。この手法を用いれば、プログラム内で直接レコードのデータを取得したり、操作することもできます。

ここでは、データアクセスコンポーネントをプログラムコードによってアクセスし、表示中の画像を、ファイルとして保存できるようにしてみましょう。

ファイル保存用のダイアログを用意する

まずは保存ダイアログコンポーネント TSaveDialog を配置します。ツールパレットの「Dialog」から TSaveDialog を選び、フォーム上の **DBImage1** 画像の下あたりに配置します。



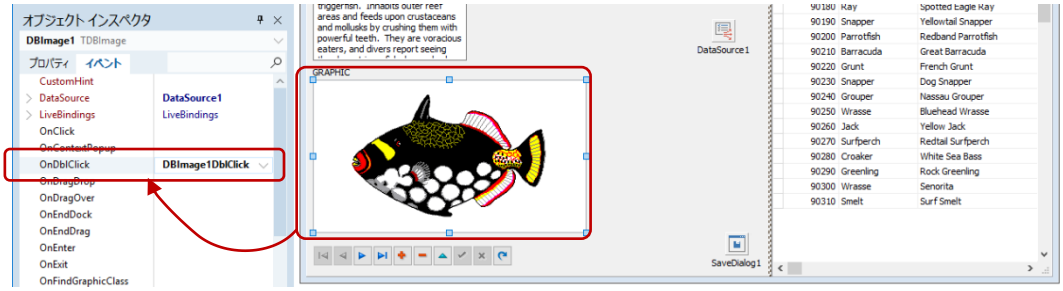
配置した TSaveDialog（**SaveDialog1**）のプロパティを次のように設定します。

SaveDialog1

項目	値
DefaultExt	bmp
Filter	ビットマップファイル|*.bmp
Options	[ofOverwritePrompt,ofHideReadOnly,ofEnableSizing]
	ofOverwritePrompt：既存のファイルを上書きするかどうかを尋ねます ofHideReadOnly：ダイアログから［読み出し専用ファイルとして開く］チェック ボックスを削除します ofEnableSizing：ダイアログのサイズを更できるようにします

画像を保存するコードの実装

画像のダブルクリックに応答して画像ファイルを保存するイベントハンドラを実装します。**DBImage1** を選択し、オブジェクトインスペクタで「OnDblClick」イベントを設定します。



イベントハンドラのコード全体は次のように記述します。

```
void __fastcall TForm1::DBImage1DblClick(TObject *Sender)
{
 // レコードから項目を取得し、SaveDialog にファイル名として設定する
 UnicodeString FileName = FDTable1->FieldByName("COMMON_NAME")->AsString;
 SaveDialog1->FileName = FileName;

 // ダイアログを表示して、画像の保存先を指定する
 if (SaveDialog1->Execute()) {
  // 画像ファイルを保存する。
  DBImage1->Picture->SaveToFile(SaveDialog1->FileName);
 }
}
```

このコードでは、レコードから画像ファイルと画像ファイル名を取得し保存しています。

'COMMON_NAME'フィールドから値を取得して保存先のファイルとし DBImage1 の Picture プロパティの SaveToFile メソッドを呼び出せば、SaveDialog1 で指定した保存先に画像が保存されます。

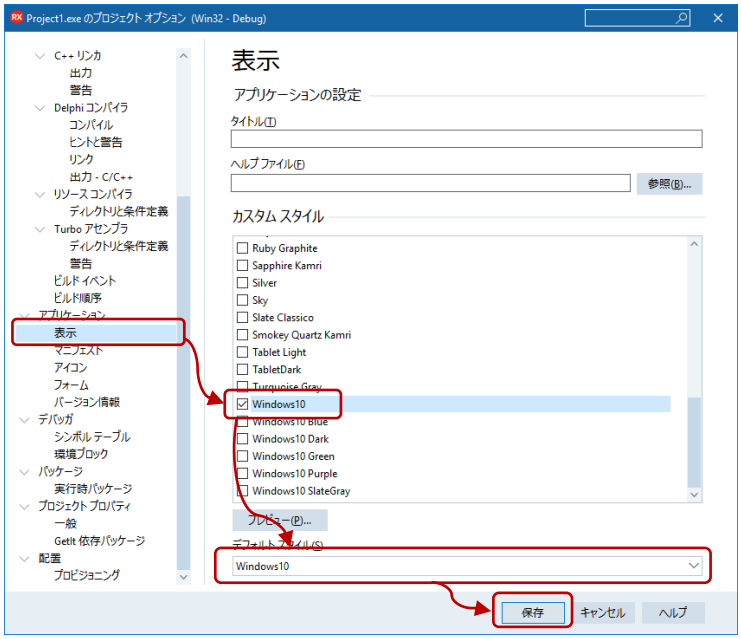
この結果、レコードから取得したデータをローカル保存先に画像として保存することができました。

Windows 10 スタイルの画面にしてみよう

以上で「魚図鑑」アプリケーションは完成ですが、最後にウィンドウの表示スタイルをカスタマイズしてみましょう。C++Builder には、スタイルと呼ばれる UI スタイル変更のしくみが用意されており、あらかじめ用意されたスタイルを切り替えるだけで、表示をモダンなイメージにしたり、ダークで落ち着いた感じにしたりと、カスタマイズできます。

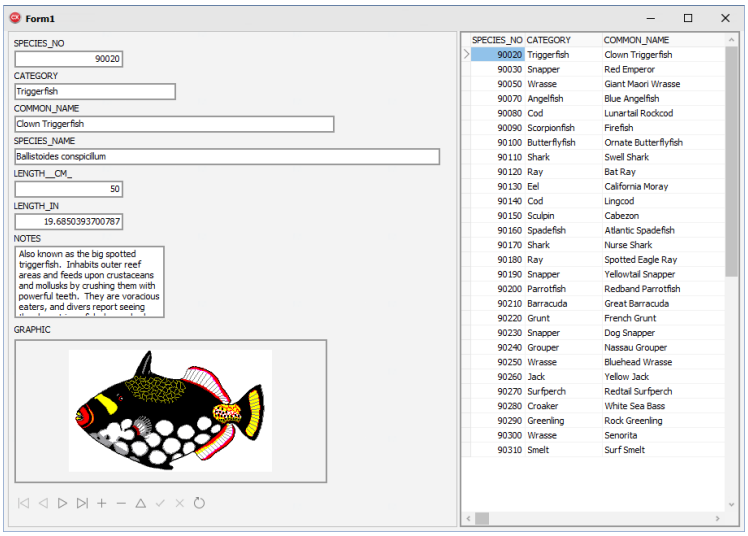
今回は、Windows 10 スタイルを適用してみましょう。メインメニューから［プロジェクト | オプション］を選択し、「プロジェクト オプション」ダイアログを表示します。

ここで、左側のツリーから「アプリケーション」下の「表示」を選択します。そして、「カスタムスタイル」項目で、「Windows 10」を選択し、その下の「デフォルトスタイル」で「Windows 10」を選択します。



［OK］ボタンをクリックして、ダイアログを閉じます。

［実行 | 実行］メニューを選択してアプリケーションを実行すると、次のように Windows 10 スタイルで UI が表示されます。



レコードの移動、データの更新などができることを確認してみましょう。