【A5】Delphiテクニカルセッション

EMBARCADERO TECHNOLOGIES

DEVELOPER CAMP

VCL で紐解く Win32 API 入門

株式会社シリアルゲームズ 取締役
細川 淳

アジェンダ

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- OS のおさらい
  - OS の動き
  - Windows API
- Windows API の実例を見てみよう
  - TButton の動き
- Windows との通信
  - メッセージについて
- 付録 TForm
  - TForm.CreateWnd の一端に触れる

EMBARCADERO TECHNOLOGIES

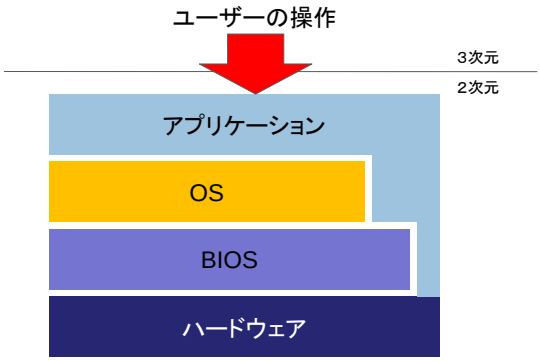
DEVELOPER CAMP

OS のおさらい

OS は何をしている？ (1)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

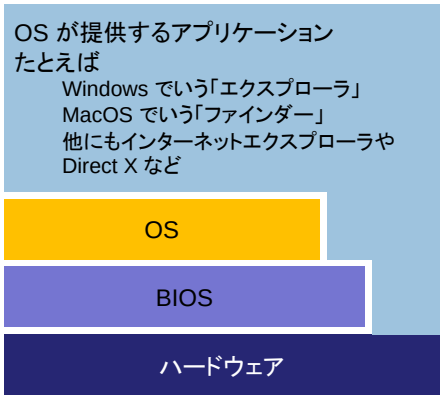
- OS とは
  - OS(Operating System)とは、ハードウェアを抽象化しハードウェアへのアクセスをアプリケーションに提供するソフトウェアのこと。



OS は何をしている？ (2)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

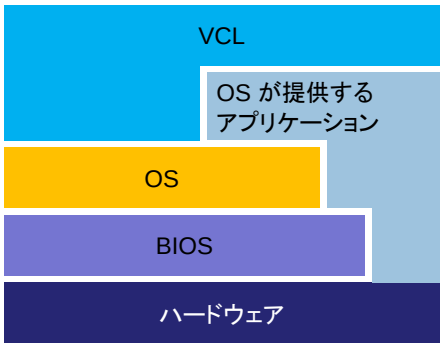
- OS のアプリケーション
  - ハードウェアへのアクセスを提供するOSのソフトウェアを含めて、広義のOSと呼ぶこともある



OS は何をしている？ (3)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

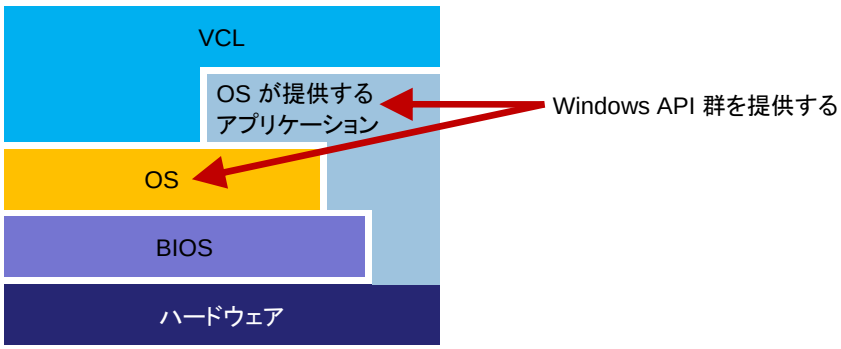
- VCL (Visual Component Library)
  - VCL が提供している機能
    - OSの機能
    - OS標準のアプリケーションの機能



OS は何をしている？ (4)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- OS が提供する機能
  - OS が提供する機能とはなんだろうか？
    - →Windows で言うと Windows API 群のこと



OS は何をしている？ (5)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- Windows API とは？
  - Wikipedia Windows API より引用

> **Windows API**（**ウィンドウズ エーピーアイ**）とは、Microsoft WindowsのAPIのことである。特に32ビットプロセッサで動作するWindows 95以降やWindows NTで利用できるものはWin32 APIと呼ばれる。また、それらのWindowsにおけるWin32 APIの実装をWin32と呼ぶ。

  - Windows API は、Windows が制御しているハードウェア・ソフトウェアに対してのアクセスを提供する
    - 例えば
      - ハードで言えば、ファイルシステム（HDD）だったり、表示（ディスプレイ）といった物
      - ソフトで言えば、TCP/IP プロトコルスタックだったり、セキュリティ関連といった物
  - ※ API とは Application Programming Interface の略
    - アプリケーションと OS の入出力を提供する機能のこと

OS は何をしている？ (6)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- Windows API が提供する大まかな機能
  - ファイルシステム
    - ファイルへのアクセスや、ファイルの情報、それらにとどまらず HDD の具体的な情報や、ボリュームなどの基本的な情報も取得できる
  - プロセス
    - プロセスやスレッドの起動・終了や、プロセスの情報や、スレッド間通信の機能を提供する
  - GDI (Graphics Device Interface)
    - ディスプレイやプリンタといった外部出力デバイスへの描画や、情報の取得できる
  - GUI (Graphical User Interface) の提供
    - ウィンドウやボタンなどの基本的なコントロールや、マウス・キーボードといった外部入力機器との通信などを提供する
  - INet API
    - インターネット関連の API で、IE が提供している

OS は何をしている？ (7)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- Windows API をラップする VCL クラスの例
  - ファイルシステム
    - TFileStream, TIniFile など
  - プロセス
    - TApplication, TThread など
  - GDI (Graphics Device Interface)
    - TCanvas, TBitmap など
  - GUI (Graphical User Interface) の提供
    - TForm, TButton など
  - INet API
    - TWebBrowser, Indy など

OS は何をしている？ (8)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- まとめ
  - OS は、ユーザーの入力と、ユーザーへの出力を担当する
    - ある機能に限った局所な視点でそうは見えなくても、大局的な視野で見れば結局の所ハードとユーザーの橋渡しを行っている
  - OS はアプリケーションに対して OS が管理している機能へのアクセスを提供する
    - アクセス手段のことを API と呼ぶ
  - VCL は、アプリケーションを構築する際に必須となる API をラップし、プログラマにわかりやすい形で再提供する
    - たとえば、BUTTON は、Windows 上ではウィンドウ（後述）の1つの形態として実装されているが、VCL での TButton を使えば、TForm とは違う「コントロール」として考えることができる

Windows API バリエーション

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- Windows API には、様々なバリエーションがある。
  - 例えば
    - Win16　　Windows 3.1 などの 16 bit システム用
    - Win32　　Windows 9x, NT 以降の 32 bit システム用
    - Win64　　Windows Xp 64, Vista 64 といった 64 bit システム用
    - Win32s　　Windows 3.1 などの 16bit システムに Win32 の機能を追加するアドオン(Win32 Subset)
    - Win32 for CE　　Windows CE 用の API
  - といった具合である
- 本セッションでは Windows API の中でも **Win32 API** について述べる。
- 特に注意せずに API と言った場合は Win32 API を指す。

EMBARCADERO TECHNOLOGIES

DEVELOPER CAMP

Windows API の実例を見てみよう

Windows と通信する

EMBARCADERO TECHNOLOGIES

DEVELOPER CAMP

- Windows API が何かは分かった。
- では、具体的にはどうやって使用するのだろう？
- VCL の TButton を元に動きを追ってみよう

TButton

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- TButton の継承を見る

TObject	全てのクラスの基底クラス
↓	
TPersistent	ストリーミング機能を持つオブジェクトの基本クラス
↓	
TComponent	コンポーネントの基本クラス
↓	
TControl	コントロールの基本クラス
↓	
TWinControl	Windows コントロールの基本クラス
↓	
TButtonControl	Button コントロールの基本クラス
↓	
TButton	Button コントロールクラス

TButton の生成 (1)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- TButton をファイルにドロップ

```
type
  TForm1 = class(TForm)
    Button1: TButton;
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

- このようなコードが生成される
  - ここでドロップされた TButton は、アプリケーション実行時にシリアライズ機構により自動的に生成される
- 生成時には TButton.Create が呼ばれる。
- しかし、TButton.Create を呼んだだけでは、実体は存在しない
  →では、どういう流れで我々が普段見る Button となるのだろうか？

TButton の生成 (2)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- TButton は親が生成されると親によって生成される
  - 簡単な流れ
    - TForm が生成される
    - TForm に乗っているコントロールが生成される
    - TForm の Visible が True になり、表示されようとする
    - TForm 上のコントロールも表示されようとする
    - TForm は、コントロールに**ハンドル**を要求する
    - TButton は、**ハンドル**を取得するために、自身の**ウィンドウ**を作る
      - この段階で初めて実体が出現する
    - TButton は、自身を表示する
    - TForm は、自身を表示する

TButton の生成 (3)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- ウィンドウとは
  - Win32 API プログラマーズリファレンスより引用

    Microsoft(R) Windows(TM) オペレーティング システム用に記述されているアプリケーションでは、アプリケーションが出力を表示したりユーザーからの入力を受け取ったりする画面上の長方形の領域のことを、ウィンドウと呼んでいます。

  - ユーザーの入出力を受け持つ領域のこと
    - TButton で言えば、まさにここのこと

    - TButton は、クリック(入力)とCaption・状態(通常・Down・Over)といった視覚効果(出力)を有する
    - 全てのコントロールは、ユーザーの入出力・もしくは出力を持つので、全てウィンドウを持つ

TButton の生成 (4)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- ウィンドウとハンドル
  - ウィンドウを生成すると、Windows は、そのウィンドウを管理する
    - ウィンドウの属性（タイトルバーがあるかなど）や状態（最大化や最小化など）を管理
  - 管理のためにつけた番号を**ハンドル**と呼ぶ
  - 逆に言えば、ハンドルがないウィンドウは存在せず、存在したとしても Windows は何ら感知しないためユーザーとの入出力は行われず、表示さえされない
  - ※ハンドルは、ウィンドウ固有の物ではなく、Windows が管理するデータには、ほぼ全てハンドルがある
    - 例えば、DeviceContext ハンドルや、ファイルハンドルなどがある。

TButton の生成 (5)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- TButton の生成順序をコードベースで見てみる
  - TForm によってハンドルを要求されるたあと
  - TWinControl.CreateHandle
  - TButton.CreateWnd
  - TWinControl.CreateParams （1）
  - TWinControl.CreateSubClass （2）
  - TWinControl.CreateWindowHandle （3）
  - という経過をたどる。
  - ここで、重要な部分は、（1）～（3）の部分

TButton の生成 (6)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- TWinControl.CreateParams
  - ウィンドウの外観を決定するパラメータを指定するメソッド
  - Windows にウィンドウの制作を依頼する場合、次の API を呼ぶ
  - **CreateWindow** 関数 または **CreateWindowEx** 関数
    - VCL では、全て CreateWindowEx 関数を用いる
      - CreateWindowEx 関数は CreateWindow 関数のスーパーセットである
    - 定義 (Win32 API プログラマーズリファレンスより引用)

```
HWND CreateWindowEx(dwExStyle, lpszClassName, lpszWindowName, dwStyle, x, y, nWidth, nHeight, hwndParent, hmenu, hinst, lpvParam)

DWORD dwExStyle;        /* 拡張ウィンドウ スタイル      */
LPCTSTR lpszClassName;  /* 登録されたクラス名のアドレス */
LPCTSTR lpszWindowName; /* ウィンドウ名のアドレス        */
DWORD dwStyle;          /* ウィンドウ スタイル          */
int x;                  /* ウィンドウの水平座標の位置    */
int y;                  /* ウィンドウの垂直座標の位置    */
int nWidth;             /* ウィンドウの幅                */
int nHeight;            /* ウィンドウの高さ              */
HWND hwndParent;        /* 親ウィンドウまたはオーナー ウィンドウのハンドル          */
HMENU hmenu;            /* メニューのハンドルまたは子ウィンドウのID     */
HINSTANCE hinst;        /* アプリケーション インスタンスのハンドル      */
LPVOID lpvParam;        /* ウィンドウ作成データのアドレス               */
```


TButton の生成 (7)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- TWinControl.CreateParams は
  - CreateWindowEx 関数の
    - dwExStyle
    - dwStyle
  - 2つのパラメータを変更する手段を与える
  - どのような値を入れられるか、いくつか示す
    - dwExStyle
      - WS_EX_ACCEPTFILES　ドラッグドロップされたファイルを受け入れる
      - WS_EX_TOPMOST　常に最前面に表示される
    - dwStyle
      - WS_CAPTION　タイトルバーを持つウィンドウを作成する
      - WS_CHILD　子ウィンドウを作成する
      - WS_SYSMENU　タイトルバー内にコントロールメニューを持つウィンドウを作成する
  - 実際にこれらを変更する例は後述する

TButton の生成 (8)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- TWinControl.CreateParams のコード

```
procedure TWinControl.CreateParams(var Params: TCreateParams);
begin
  FillChar(Params, SizeOf(Params), 0);
  with Params do
  begin
    Caption := FText;
    Style := WS_CHILD or WS_CLIPSIBLINGS;
    AddBiDiModeExStyle(ExStyle);
    if csAcceptsControls in ControlStyle then
    begin
      Style := Style or WS_CLIPCHILDREN;
      ExStyle := ExStyle or WS_EX_CONTROLPARENT;
    end;
    if not (csDesigning in ComponentState) and not Enabled then
      Style := Style or WS_DISABLED;
    if FTabStop then Style := Style or WS_TABSTOP;
    X := FLeft;
    Y := FTop;
    Width := FWidth;
    Height := FHeight;
    if Parent <> nil then
      WndParent := Parent.GetHandle else
      WndParent := FParentWindow;
    WindowClass.style := CS_VREDRAW + CS_HREDRAW + CS_DBLCLKS;
    WindowClass.lpfnWndProc := @DefWindowProc;
    WindowClass.hCursor := LoadCursor(0, IDC_ARROW);
    WindowClass.hbrBackground := 0;
    WindowClass.hInstance := HInstance;
    StrPCopy(WinClassName, ClassName);
  end;
end;
```

太い赤字で示した部分でスタイルを指定している

TWinControl.CreateParams は仮想メソッドとして提供されているので継承先で、このパラメータを変更すると、状態や属性を変更できる

変更の実例は、後述

TButton の生成 (9)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- TWinControl.CreateSubClass
  - このメソッドは定義済みウィンドウのデフォルトの属性を取得するために利用される
  - Windows は、いくつかのウィンドウについてあらかじめひな形（定義済みコントロールクラス名）を持っており、それは文字列で示される
    - 例えば
      - BUTTON　　ボタン(TButton)
      - EDIT　　エディットボックス(TEdit)
      - LISTBOX　　リストボックス(TListBox)

      など
  - TButton は、当然 'BUTTON' を指定して、BUTTON ウィンドウに必要な情報をこのメソッドで取得する。

TButton の生成 (10)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- TWinControl.CreateSubClass のコード

```
procedure TWinControl.CreateSubClass(var Params: TCreateParams;  ControlClassName: PChar);
const
  CS_OFF = CS_OWNDC or CS_CLASSDC or CS_PARENTDC or CS_GLOBALCLASS;
  CS_ON = CS_VREDRAW or CS_HREDRAW;
var
  SaveInstance: THandle;
begin
  if ControlClassName <> nil then
    with Params do
    begin
      SaveInstance := WindowClass.hInstance;
      if not GetClassInfo(HInstance, ControlClassName, WindowClass) and
        not GetClassInfo(0, ControlClassName, WindowClass) and
        not GetClassInfo(MainInstance, ControlClassName, WindowClass) then
        GetClassInfo(WindowClass.hInstance, ControlClassName, WindowClass);
      WindowClass.hInstance := SaveInstance;
      WindowClass.style := WindowClass.style and not CS_OFF or CS_ON;
    end;
end;
```

  - 分かりづらいが、上記コードの背景色がある部分で、API の GetClassInfo 関数を呼んで WindowClass と呼ばれるウィンドウの属性を取得している

TButton の生成 (11)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- TWinControl.CreateWindowHandle
  - このメソッドで、実際にウィンドウが作成される。
  - TWinControl.CreateParams, CreateSubClass などで設定された情報を元に CreateWindowEx 関数を呼び、実際にウィンドウを作成する
- TWinControl.CreateWindowHandle のコード

```
procedure TWinControl.CreateWindowHandle(const Params: TCreateParams);
begin
  with Params do
    FHandle := CreateWindowEx(ExStyle, WinClassName, Caption, Style,
      X, Y, Width, Height, WndParent, 0, WindowClass.hInstance, Param);
end;
```

  - CreateParams や CreateSubClass で情報は取得済みのため、このメソッドは非常に簡潔で、CreateWindowEx を呼んでいるだけである
  - CreateWindowEx を呼んだ戻りとして**ハンドル**を手に入れる

TButton の生成 (12)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- まとめ
  - TButton は、自らを表現するために以下の3つのメソッドを呼ぶ
    - CreateParams　　属性や見た目を指定する
    - CreateSubClass　　定義済みクラスの属性を取得する
API の GetClassInfo を呼ぶ
    - CreateWindowHandle　　実際にウィンドウを作成しハンドルを取得する
API の CreateWindowEx を呼ぶ
  - VCL を使うと非常に簡単にコントロール（ウィンドウ）を作成できるが、中ではかなり複雑なコードが動いていることが分かるだろう。

実例

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- ここでは、TButton で出てきた CreateParams, CreateSubClass を使ったコードの実例を紹介する
- 筆者が RichEdit 2.0 の機能を使うために使っているコード

```
procedure TRichEditEx.CreateParams(var ioParams: TCreateParams);

  function LoadRichEdit(const iFileName: String): Boolean;
  begin
    if (FRichEditModule = 0) then begin
      FRichEditModule := LoadLibrary(PChar(iFileName));
      if (FRichEditModule <= HINSTANCE_ERROR) then
        FRichEditModule := 0;
    end;

    Result := (FRichEditModule <> 0);
  end;

begin
  if not (csDesigning in ComponentState) then
    LoadRichEdit('RICHED20.DLL');

  inherited CreateParams(ioParams);

  if (FAutoScroll) then
    ioParams.Style := ioParams.Style or ES_AUTOVSCROLL
  else
    ioParams.Style := ioParams.Style and not ES_AUTOVSCROLL;

  if (FRichEditModule <> 0) then begin
    CreateSubClass(ioParams, 'RichEdit20A');
    Fls20 := True;
  end;
end;
```

太字の赤字部分で Style の変更と使用する RichEdit クラスを変更している

普通の TRichEdit は 'RICHEDIT' を指定している。
RichEdit20A を指定すると RichEdit の新しいバージョンを使用できる

EMBARCADERO TECHNOLOGIES

DEVELOPER CAMP

Windows との通信

メッセージ (1)

EMBARCADERO TECHNOLOGIES

DEVELOPER CAMP

- Windows と何かをやり取りするには API で提供される関数だけではなく、様々な方法がある
  - Windows API の関数を呼ぶ
  - コールバック関数を登録する
  - COM オブジェクトと通信する
  - メッセージのやり取り

などがある。
ここでは特に重要で Windows の根幹をなすメッセージについて見てみよう

メッセージ (2)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- メッセージとは？
  - Win32 API プログラマーズリファレンスより引用

    情報通信や要求のために使われるデータ パケット。メッセージは、オペレーティング システムとアプリケーション間、異なるアプリケーション間、アプリケーション中の「スレッド」間、および、アプリケーション中の「ウィンドウ」間で、受け渡しができます。
  - メッセージとは、異なる動作主体同士の通信手段
  - それは、どのような組み合わせもありうる
    - OSとアプリケーション
    - アプリケーションとアプリケーション
    - コントロールとコントロール
    - スレッドとスレッド

    などなど。
  - メッセージは、パラメータを2つ渡すことができる
    - 第1パラメータを WParam
    - 第2パラメータを LParam

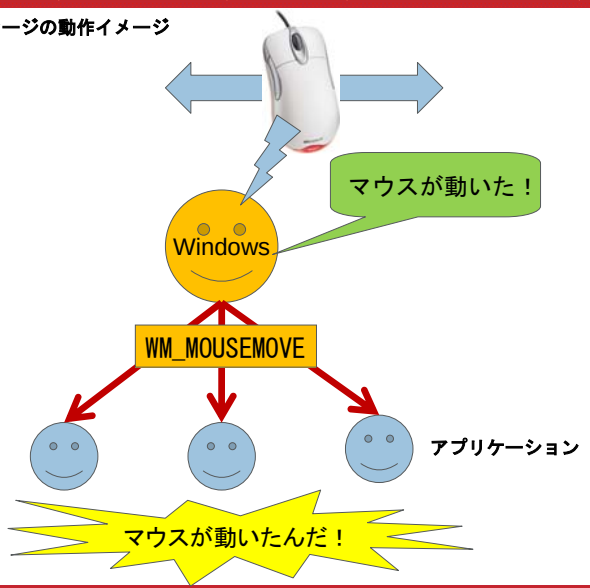
    と呼ぶ

    両方とも 32bit の値

メッセージ (3)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

イベント主導型メッセージの動作イメージ



メッセージ (4)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- メッセージの受け渡し
  - では、実際はどのようにメッセージを受け渡すのだろうか？
- メッセージの送信
  下記の2つが代表的な API
  - SendMessage 関数　　メッセージをウィンドウプロシージャに送る
  - PostMessage 関数　　メッセージをメッセージキューに送る

メッセージ (5)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- SendMessage 関数
  - Win32 API プログラマーズリファレンスより引用

```
LRESULT SendMessage(hwnd, uMsg, wParam, lParam)

HWND hwnd;              /* 送り先のウィンドウのハンドル     */
UINT uMsg;              /* 送るメッセージ                   */
WPARAM wParam;          /* 第1メッセージ パラメータ         */
LPARAM lParam;          /* 第2メッセージ パラメータ         */

SendMessage関数は、指定された1つまたは複数のウィンドウに、指定されたメッセージを送ります。この
関数は指定されたウィンドウのウィンドウ プロシージャを呼び出し、プロシージャがメッセージを処理し
終わるまで制御を戻しません。これに対して、PostMessage関数は、メッセージをスレッドのメッセージ
キューにポストして、直ちに制御を戻します
```

  - SendMessage 関数は
    - メッセージの送り先をウィンドウハンドルで示す
    - ウィンドウプロシージャを呼ぶ
    - 呼び出し先の実行が終わらないと処理が戻らない
    - 結果が戻り値で手に入る

メッセージ (6)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- PostMessage 関数
  - Win32 API プログラマーズリファレンスより引用

```
BOOL PostMessage(hwnd, uMsg, wParam, lParam)

HWND hwnd;              /* 受け取り側のウィンドウのハンドル */
UINT uMsg;              /* ポストするメッセージ             */
WPARAM wParam;          /* 第1メッセージ パラメータ         */
LPARAM lParam;          /* 第2メッセージ パラメータ         */

PostMessage関数は、スレッドに関連付けられたメッセージ キューにメッセージを置き（ポストし）、対応
するスレッドがメッセージを処理する前に制御を戻します。メッセージ キュー内のメッセージは、
GetMessage関数かPeekMessage関数を呼び出して取得します。
```

  - PostMessage 関数は
    - メッセージの送り先をウィンドウハンドルで示す
    - メッセージはキューに置かれる
    - 結果が戻り値で手に入らない

メッセージ (7)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

メッセージの送信については分かったが、所々で出てきた
「ウィンドウプロシージャ」とは？

- ウィンドウプロシージャとは、メッセージを処理する関数
  - ウィンドウを持つ全てのコントロールが持つ
    - CreateWindowEx に渡す WindowClass 構造体の中にウィンドウプロシージャを指定するパラメータがある
    - TButton.CreateParams のコードでも以下のように指定されている

```
WindowClass.lpfnWndProc := @DefWindowProc;
```

VCL が用意しているデフォルトのウィンドウプロシージャ

  - VCL で言えば TControl 以下のコンポーネントはデフォルトの WndProc メソッドで処理されている

メッセージ (8)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- メッセージの受信
  - ウィンドウプロシージャで受け取る
  - ウィンドウはすべからくウィンドウプロシージャを持つ
  - Delphi 言語の場合メッセージディスパッチ機構(※)があるためウィンドウプロシージャ自身をいじることなく、メッセージに応答できる
  - TButton のメッセージ応答メソッドの宣言部

```
TButton = class(TButtonControl)
private
  procedure CMDialogKey(var Message: TCMDialogKey); message CM_DIALOGKEY;
  procedure CMDialogChar(var Message: TCMDialogChar); message CM_DIALOGCHAR;
  procedure CMFocusChanged(var Message: TCMFocusChanged); message CM_FOCUSCHANGED;
  procedure CNCommand(var Message: TWMCommand); message CN_COMMAND;
  procedure CNCtlColorBtn(var Message: TWMCtlColorBtn); message CN_CTLCOLORBTN;
  procedure WMEraseBkgnd(var Message: TWMEraseBkgnd); message WM_ERASEBKGND;
```

  - このように書くことで該当メッセージが来たときに特定の処理を実行できる

※Dispatch メソッドで実現される

メッセージ (9)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- TButton.WMEraseBkGnd を見る

```
// 宣言部
TButton = class(TButtonControl)
private
  procedure WMEraseBkgnd(var Message: TWMEraseBkgnd); message WM_ERASEBKGND;
end;

// 実現部
procedure TButton.WMEraseBkgnd(var Message: TWMEraseBkgnd);
begin
  if ThemeServices.ThemesEnabled then
    Message.Result := 1
  else
    DefaultHandler(Message);
end;
```

  - WM_ERASEBKGND とは、背景を消去する必要があるとき Windows から送られるメッセージ
  - このコードでは、Xp のテーマが有効な場合は何もしないようになっている
  - 無効の場合、TWinControl.DefaultHandler を呼ぶようになっている
    - 最終的には Windows が自動的に背景を消去する

メッセージ (9)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- Windows から送られる代表的なメッセージ
  - WM_ACTIVATE　アクティブになった
  - WM_COMMAND　メニューを選ばれた
  - WM_CREATE　ウィンドウが生成された
  - WM_DESTROY　ウィンドウが破棄された
  - WM_KEYDOWN　キーが押された
  - WM_LBUTTONDOWN　マウスの左ボタンが押された
  - WM_RBUTTONDOWN　マウスの右ボタンが押された
  - WM_MOUSEMOVE　マウスが移動した
  - WM_PAINT　描画要求
  - WM_QUIT　アプリエーション終了要求
  - WM_TIMER　タイマーで設定した時間が経過
  - WM_WINDOWPOSCHANGING　ウィンドウが移動している

メッセージを送信する例

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- TRichEditEx のコード

```
procedure TRichEditEx.CreateWnd;
var
  Style: DWORD;
begin
  inherited;

  Style := GetClassLong(Handle, GCL_STYLE);

  if (FAutoScroll) then
    SetClassLong(Handle, GCL_STYLE, Style or ES_AUTOVSCROLL)
  else
    SetClassLong(Handle, GCL_STYLE, Style and not ES_AUTOVSCROLL);

  Perform(EM_GETOLEINTERFACE, 0, LPARAM(@FRichEditOle));

  Perform(
    EM_SETOLEINTERFACE,
    0,
    LPARAM(FRichEditCallback as IRichEditOleCallback));

  Perform(
    EM_SETEVENTMASK,
    0,
    Perform(EM_GETEVENTMASK, 0, 0) or ENM_LINK);

  Perform(
    EM_SETLANGOPTIONS,
    0,
    Perform(EM_GETLANGOPTIONS, 0, 0) and (not IMF_DUALFONT));

  SetURLEnabled(FURLEnabled);
end;
```

太字の赤字部分でメッセージを送信している
Perform は TControl のメソッドで自分自身にメッセージを送る

SendMessage(自分のハンドル, メッセージ, パラメータ１, ２);

とするのと同じ意味だが、Perform のほうが効率が良い。

ここでは自分自身にメッセージを送り RichEdit2.0 に様々な命令を送っている

ちなみに GetClassLong API で CreateParams で指定したスタイルを取得、SetClassLong API でスタイルを設定できる

メッセージを受信する例

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- TCustomForm.WMSysCommand のコード

```
// 宣言部
TCustomForm = class(TScrollingWinControl)
private
  procedure WMSysCommand(var Message: TWMSysCommand); message WM_SYSCOMMAND;
end;



// 実現部
procedure TCustomForm.WMSysCommand(var Message: TWMSysCommand);
begin
  with Message do
  begin
    if (CmdType and $FFF0 = SC_MINIMIZE) and (Application.MainForm = Self) then
      Application.WndProc(TMessage(Message))
    else if (CmdType and $FFF0 <> SC_MOVE) or (csDesigning in ComponentState) or
      (Align = alNone) or (WindowState = wsMinimized) then
      inherited;
    if ((CmdType and $FFF0 = SC_MINIMIZE) or (CmdType and $FFF0 = SC_RESTORE)) and
      not (csDesigning in ComponentState) and (Align <> alNone) then
      RequestAlign;
  end;
end;
```

WM_SYSCOMMAND はシステムメニューが選ばれた時に送られるメッセージ

WParam の値によって、何が選ばれたか分かる

SC_RESTORE 元のサイズに戻す
SC_MOVE 移動
SC_SIZE サイズ変更
SC_MINIMIZE 最小化
SC_MAXIMIZE 最大化
SC_CLOSE 閉じる

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

付録 TForm

TForm の生成 (1)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- TForm の生成
  - 今まで登場した
    - Windows API
    - ウィンドウハンドル
    - メッセージ
  - この3つ全てが使われ、VCL アプリケーションの根幹をなしている
  - 他にも
    - デバイス コンテキスト (DC)
      - TCanvas.Handle で表されるハンドル
      - Windows の描画は DC を介して行われる
        - » ただし、Vista 以降は変わってきている
    - ファイルハンドル
      - ファイルのオープンや読み取りをする際に必要なハンドル
  - などが使われている

TForm の生成 (2)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- TForm の継承を見る

TObject	全てのクラスの基底クラス
↓	
TPersistent	ストリーミング機能を持つオブジェクトの基本クラス
↓	
TComponent	コンポーネントの基本クラス
↓	
TControl	コントロールの基本クラス
↓	
TWinControl	Windows コントロールの基本クラス
↓	
TScrollingWinControl	スクロールをサポートするコントロール用の基本クラス
↓	
TCustomForm	フォームやダイアログボックスなどのウィンドウの派生元となる基本クラス
↓	
TForm	フォーム

TForm の生成 (3)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- TForm は TApplication が生成する
  - 簡単な流れ
    - プログラムが実行される
    - TApplication が生成される
    - TApplication.CreateForm で TForm が生成される
    - TForm に乗っているコントロールが生成される
    - TForm の Visible が True になり、表示されようとする
    - TForm 上のコントロールも表示されようとする
    - TForm は、自身を表示する

TForm の生成 (4)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- TButton とあまり変わらず、ただ高度なコンポーネントなので様々な Windows API, メッセージを用いている
- ここでは、例として TCustomForm.CreateWnd を見てみよう

TForm の生成 (5)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- TCustomForm.CreateWnd のコード

```
procedure TCustomForm.CreateWnd;
var
  I: Integer;
  ClientCreateStruct: TClientCreateStruct;
begin
  inherited CreateWnd;
  if NewStyleControls then
    if BorderStyle <> bsDialog then
      SendMessage(Handle, WM_SETICON, 1, LPARAM(GetIconHandle)) else
      SendMessage(Handle, WM_SETICON, 1, 0);
  if not (csDesigning in ComponentState) then
    case FormStyle of
      fsMDIForm:
        begin
          with ClientCreateStruct do
          begin
            idFirstChild := $FF00;
            hWindowMenu := 0;
            if FWindowMenu <> nil then hWindowMenu := FWindowMenu.Handle;
          end;
          FClientHandle := Windows.CreateWindowEx(WS_EX_CLIENTEDGE, 'MDICLIENT',
            nil, WS_CHILD or WS_VISIBLE or WS_GROUP or WS_TABSTOP or
            WS_CLIPCHILDREN or WS_HSCROLL or WS_VSCROLL or WS_CLIPSIBLINGS or
            MDIS_ALLCHILDSTYLES, 0, 0, ClientWidth, ClientHeight, Handle, 0,
            HInstance, @ClientCreateStruct);
          FClientInstance := Classes.MakeObjectInstance(ClientWndProc);
          FDefClientProc := Pointer(GetWindowLong(FClientHandle, GWL_WNDPROC));
          SetWindowLong(FClientHandle, GWL_WNDPROC, Longint(FClientInstance));
        end;
      fsStayOnTop:
        SetWindowPos(Handle, HWND_TOPMOST, 0, 0, 0, 0, SWP_NOMOVE or
          SWP_NOSIZE or SWP_NOACTIVATE);
    end;
  if Assigned(FRecreateChildren) then
  begin
    for I := 0 to FRecreateChildren.Count - 1 do
      TCustomForm(FRecreateChildren[I]).UpdateControlState;
    FRecreateChildren.Clear;
  end;
  for I := Low(FPopupWnds) to High(FPopupWnds) do
    SendMessage(
      FPopupWnds[I].ControlWnd,
      CM_CREATEPOPUP,
      FPopupWnds[I].ID,
      LPARAM(WindowHandle));

  SetLength(FPopupWnds, 0);

  if not (csLoading in ComponentState) and GlassFrame.FrameExtended then
    UpdateGlassFrame(nil);
end;
```

数多くの API, メッセージを用いているのが分かる

FormStyle が fsMDIForm の場合、CreateWnd 内で CreateWindowEx を呼んでいる