embarcadero
Developer Camp

【T3】Delphi/C++Builderテクニカルセッション

「Delphi IDEでTwitter生活！ Open Tools APIで開発環境を簡単カスタマイズ」

株式会社　日本情報システム
筑木真志

自己紹介

- 名前：筑木　真志　(ちくぎ　しんじ)
  - 多分、同姓同名はいません
- 会社：株式会社　日本情報システム
  - URL : http://www.nihon-jyoho.com
  - E-MAIL : nis@nihon-jyoho.com
- 会社概要：
  - 土木・建築・測量系の開発が多い
  - GISとかDBを活用したシステムなんて大好きです
  - でも、会計とか集計処理などOSや言語問わずいろいろやっています
  - GIS関連スクールの講師とかもやることがあります
  - あと、レガシーシステムのお守りとか移行サポートとか
  - ぶっちゃけ、IT系の「大工さん」というか「何でも屋さん」
  - だから、道具にこだわる故に自分はC++Builderが大好き!!

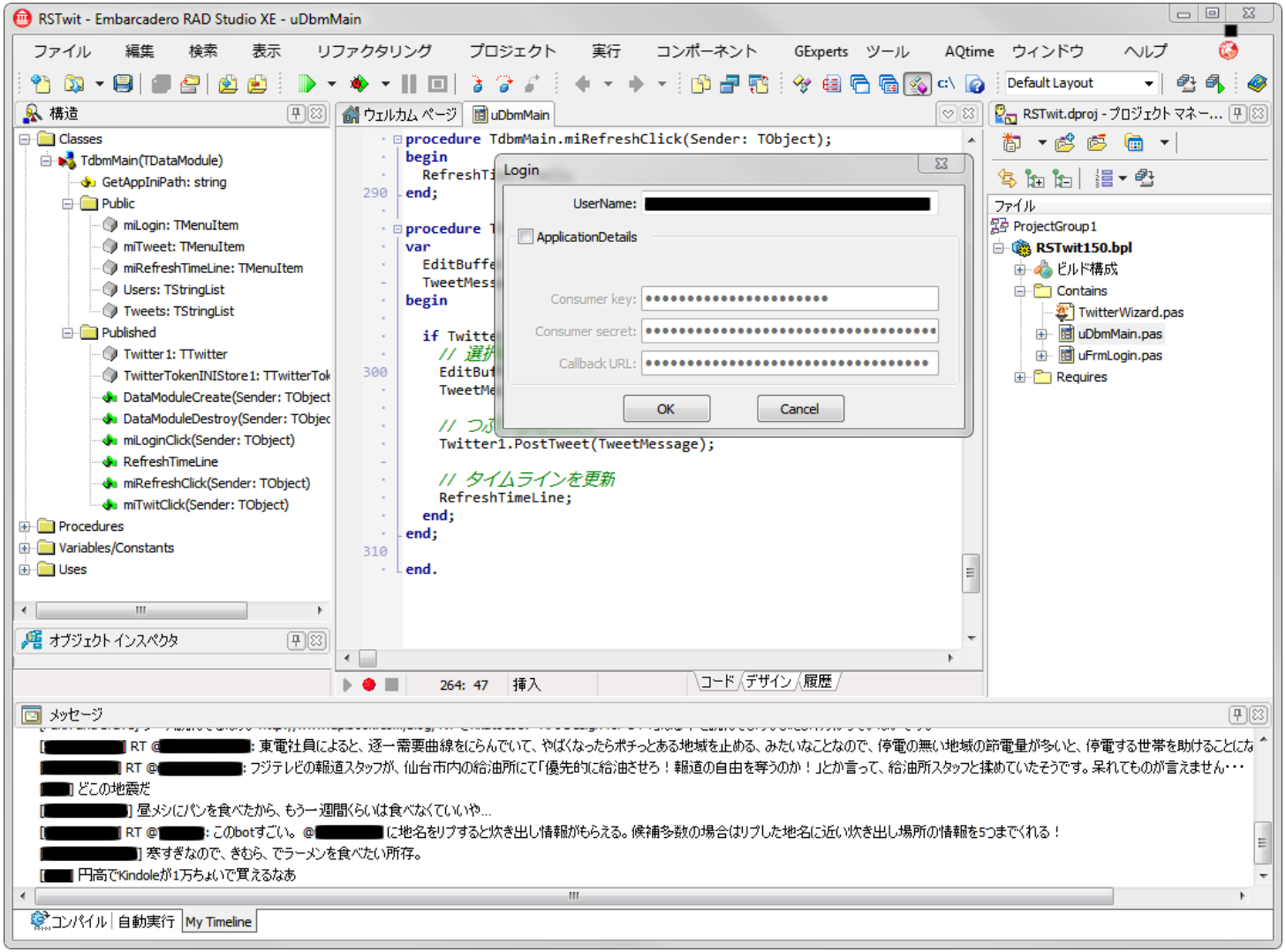
注意点というかエクスキューズ

- Open Tools APIについての資料は少ないです
  - 大半が英語で日本語での情報はマニュアルくらい
- 内容はGExpertsなどのソースコードと試行錯誤の結果
- コードの解説は要点のみ、ほんのさわりの部分
  - 見づらいかもしれないのでレジュメのを見てください
  - オンラインの方は後日サンプルコードを配布します
- コードはDelphiが基本だけど、自分はC++Builder使い
- もしかしたら、間違いがあるかも… <(_ _)>
- 途中でエラーが発生する場合もあったり
  - てか、一回わざとやる

embarcadero
Developer Camp

いきなり
デモンストレーション

こんなのを作ってみました



アジェンダ

- “Open Tools API”って何？
- とりあえず、“Hello, world!”
- Delphi/C++Builder IDEのTwitterクライアント化
  - Twitterクライアントコンポーネント “TTwitter”の紹介
  - タイムラインの表示
    - IDEにメニューの追加
    - メッセージウィンドウへのアクセス
  - つぶやいてみる
    - コードエディタへのアクセス
- まとめ・Q&A

embarcadero
Developer Camp

“Open Tools API” って何？

Open Tools APIとは

- Delphi/C++Builder IDEを拡張するためのAPI群
  - メニュー、ツールバーへのアクセス
  - ソースコードエディタへのアクセス
  - フォームエディタのカスタマイズ
  - オブジェクトインスペクタのカスタマイズ
  - キーバインドやキーボードマクロの定義
  - デバッガへのアクセス
  - メッセージウィンドウへのアクセス
  - コード補完
  - Etc.
- インターフェースは”ToolsAPI.pas”ですべて定義

Native Tools APIとOpen Tools API

- Native Tools API
  - IDEのTMainMenuオブジェクトなど、IDEを構成するVCLオブジェクトに直接アクセス
  - クラス名にNTAを含む
- Open Tools API
  - 抽象的なインターフェースを介してIDEにアクセス
  - クラス名にOTAを含む
- 両者を総称してTools APIと呼ぶ
  - でも、一緒くたに“Open Tools API”と呼ぶ場合もある

Open Tools APIによるIDEの拡張手順

- 設計時パッケージ(*.bpl)かDLLを作成（アドオン）

- 作成したアドオンをIDEに「登録」
  - アドオンに”Register”関数を記述し、エクスポートする
  - プロジェクトマネージャから「インストール」
- アドオンはVCLアプリケーション
  - 通常のアプリケーション作成テクニックがそのまま使える
  - クラスはToolsAPIユニットで定義
    - Delphiの場合：uses節にToolsAPIを含める
    - C++Builderの場合：ToolsAPI.hppをインクルード
  - designide.dcpパッケージを必ずリンクする

Open Tools APIの要素　その1:ウイザード

- アドオンの中核となる、プログラマが記述するサブクラス
  - IOTAMenuWizard
    - [ヘルプ]メニューに項目を追加する
  - IOTAFormWizard
    - [新規作成]ダイアログに項目を追加する
  - IOTAProjectWizard
    - [新規作成]ダイアログの[Delphi/C++Builderプロジェクト]に項目を追加する
  - IOTAWizard
    - 上記以外のアドオンの基本的なクラス

Open Tools APIの要素　その2：サービス

- IDEが提供する機能とアドオンの仲介役
  - 主なサービス

INTAServices	IDEそのもの
IOTAEditorServices	コードエディタ、フォームエディタ
IOTAModuleServices	ファイル・プロジェクトなど
IOTAMessageServices	メッセージウィンドウ
IOTAKeyBindingServices	キーバインド
IOTADebuggerServices	デバッガ
IOTACodeInsightServices	コードインサイト

  - サービスからオブジェクトを取得
    - 基本となる” BorlandIDEServices”からキャスト

```
  // IDEのノーティファイアの登録
IIDEServices := BorlandIDEServices as IOTAServices;
```

Open Tools APIの要素　その3:ノーティファイヤ

- IDE内部でイベントが発生すると、コールバックされるメソッドのグループ（インターフェースクラス）
  - 主なノーティファイヤ

IOTANotifier	ノーティファイヤの抽象基本クラス
IOTAIDENotifier	IDE内でのイベント
IOTADebuggerNotifie	デバッガでのイベント
IOTAEditorNotifier	コードエディタでのイベント
IOTAFormNotifier	フォームエディタでのイベント

  - サブクラスをIDEのオブジェクトに「登録」
  - イベントが発生すると、特定のメソッドがコールされる
  - プログラマはメソッド内部の処理を記述

Open Tools APIの要素　その4：TNotifierObject

- クラスTNotifierObjectはウイザード、ノーティファイヤのインスタンスを定義するユーティリティクラス

- アドオンで記述するオブジェクトは、TNotifierObjectとウイザード、ノーティファイヤのサブクラス

```
type THelloWorldMenuWizard =
   class(TNotifierObject, IOTAWizard, IOTAMenuWizard)

  function GetIDString: string;
  function GetName: string;
  function GetState: TWizardState;
  procedure Execute;
  function GetMenuText: string;
end;
```

embarcadero
Developer Camp

とりあえず、
“Hello, world!”

Open Tools APIを使ったサンプルコード

```
unit OTADelWizard;

interface

procedure Register;

implementation

uses
  ToolsAPI, Dialogs;

type THelloWorldMenuWizard =
   class(TNotifierObject, IOTAWizard, IOTAMenuWizard)

  function GetIDString: string;
  function GetName: string;
  function GetState: TWizardState;
  procedure Execute;
  function GetMenuText: string;
end;

function THelloWorldMenuWizard.GetIDString: string;
begin
  Result := 'DevCamp19DelWizard1.0';
end;
```

```
function THelloWorldMenuWizard.GetName: string;
begin
  Result := 'Open Tools API Delphi Sample Wizard';
end;

// メニューの項目
function THelloWorldMenuWizard.GetMenuText: string;
begin
  Result := 'DelphiでHello,world';
end;

// メニュー項目の状態
function THelloWorldMenuWizard.GetState: TWizardState;
begin
  Result := [wsEnabled];  // メニュー項目は有効
end;

// メニュー項目が選択された
procedure THelloWorldMenuWizard.Execute;
begin
  ShowMessage('Delphiから こんにちは、世界！');
end;

// パッケージを登録
procedure Register;
begin
  RegisterPackageWizard(THelloWorldMenuWizard.Create);
end;
end.
```

IOTAMenuWizardを使ってIDEにメニューを登録

- ユニットToolsAPIの使用

```
implementation

uses
  Dialogs, ToolsAPI;
```

Uses節に”ToolsAPI”を含める

- ウイザードクラスを定義する

```
type THelloWorldMenuWizard =
   class(TNotifierObject, IOTAWizard, IOTAMenuWizard)

  function GetIDString: string;
  function GetName: string;
  function GetState: TWizardState;
  procedure Execute;
  function GetMenuText: string;
end;
```

基底クラスは
ウイザードインターフェース

必要なメソッドの宣言

IOTAMenuWizardを使ってIDEにメニューを登録

- メソッドの実装

```
function THelloWorldMenuWizard.GetIDString: string;
begin
  Result := 'DevCamp19DelWizard1.0';
end;

function THelloWorldMenuWizard.GetName: string;
begin
  Result := 'Open Tools API Delphi Sample Wizard';
end;

// メニューのタイトル
function THelloWorldMenuWizard.GetMenuText: string;
begin
  Result := 'DelphiでHello,world';
end;

// メニュー項目の状態
function THelloWorldMenuWizard.GetState: TWizardState;
begin
  Result := [wsEnabled];  // メニューは有効
end;

// メニュー項目が選択された
procedure THelloWorldMenuWizard.Execute;
begin
  ShowMessage('Delphiから こんにちは、世界！');
end;
```

GetIDString
IDEがウイザードを識別するための
ユニークな名前

GetName
ユーザーが任意につける
ウイザードの名前

GetMenuText
[ヘルプ]メニューに追加する
項目のタイトル

GetState
メニュー項目の状態

Execute
メニュー項目が選択された場合に
呼び出される

IOTAMenuWizardを使ってIDEにメニューを登録

- パッケージをIDEに登録する

```
unit OTADelWizard;

interface

procedure Register;

implementation

{
   中略
}

// パッケージを登録
procedure Register;
begin
  RegisterPackageWizard(THelloWorldMenuWizard.Create);
end;
```

Register関数をinterface節に記述して、
外部から参照可能にする

IDEはパッケージの登録時にパッケージの
Register関数を実行する

RegisterPackageWizard関数はウイザードを
IDEに登録する関数

C++Builderの場合（OTAWizard.h）

```
#include <ToolsAPI.hpp>

class PACKAGE TOTAWizard : public TNotifierObject, public IOTAMenuWizard
{
  typedef TNotifierObject inherited;
public:
  // IOTAWizard
  virtual UnicodeString __fastcall GetIDString();
  virtual UnicodeString __fastcall GetName();
  virtual TWizardState __fastcall GetState();
  virtual void __fastcall Execute();
  // IOTAMenuWizard
  virtual UnicodeString __fastcall GetMenuText();
  void __fastcall AfterSave();
  void __fastcall BeforeSave();
  void __fastcall Destroyed();
  void __fastcall Modified();
protected:
  // IInterface
  virtual HRESULT __stdcall QueryInterface(const GUID&, void**);
  virtual ULONG __stdcall AddRef();
  virtual ULONG __stdcall Release();
};
```

IOTAWizardクラスは、IOTAMenuWizardクラスの基底クラスなので、明示的に継承する必要が無い

メモリ管理関連の関数を明示的に宣言

C++Builderの場合（OTAWizard.cpp）

```
//---------------------------------------------------------------------------
ULONG __stdcall TOTAWizard::AddRef()
{
  return inherited::_AddRef();
}
//---------------------------------------------------------------------------
ULONG __stdcall TOTAWizard::Release()
{
  return inherited::_Release();
}
//---------------------------------------------------------------------------
// ウイザードが各クラスのインターフェースをサポートするか？
HRESULT __stdcall TOTAWizard::QueryInterface(const GUID& iid, void** obj)
{
  if (iid == __uuidof(IOTAMenuWizard)) {
    *obj = static_cast<IOTAMenuWizard*>(this);
    static_cast<IOTAMenuWizard*>(*obj)->AddRef();
    return S_OK;
  }
  if (iid == __uuidof(IOTAWizard)) {
    *obj = static_cast<IOTAWizard*>(this);
    static_cast<IOTAWizard*>(*obj)->AddRef();
    return S_OK;
  }
  return inherited::QueryInterface(iid, obj);
}
```

オブジェクトは参照数カウントで管理するので、その部分は自分で記述

C++Builderの場合（OTAWizard.cpp）

```
namespace Otawizard
{
void __fastcall PACKAGE Register()
{
  RegisterPackageWizard(new TOTAWizard());
}
}
```

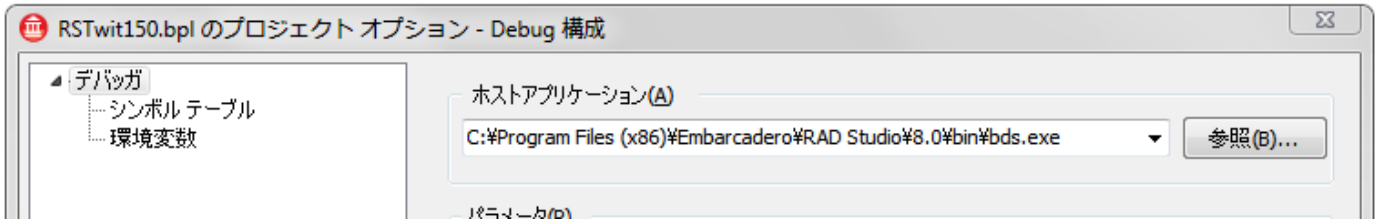
Register関数はnamespaceでくくる
_fastcall PACKAGE 修飾子が必要

- クラス名、ファイル名、名前空間の付け方はルールがある

クラス名：　TOTAWizard
ファイル名：OTAWizard.cpp/ OTAWizard.h
名前空間：Otawizard

実際にデバッグする

- IDEから自分自身(BDS.EXE)をデバッグする
  - ホストアプリケーションにBDS.EXEを指定



- 通常のアプリケーションと同じようにデバッグできる

- ビルド後のアドオンはデバッグ元のIDEに影響を与えるのでIDEがハングする可能性がある
  - ビルド前にプロジェクトの保存
  - タスクマネージャーが必須

embarcadero
Developer Camp

Delphi/C++Builder IDEの Twitterクライアント化

Twitterクライアントコンポーネント “TTwitter”

- TTwitterはTwitter APIをラップするコンポーネント
- 入手元
  - http://www.lakraven.com/delphi-stuff/ttwitter/

- そのままだと日本語が通らないので、修正の必要あり
  - Twitter_OAuth.pasを修正

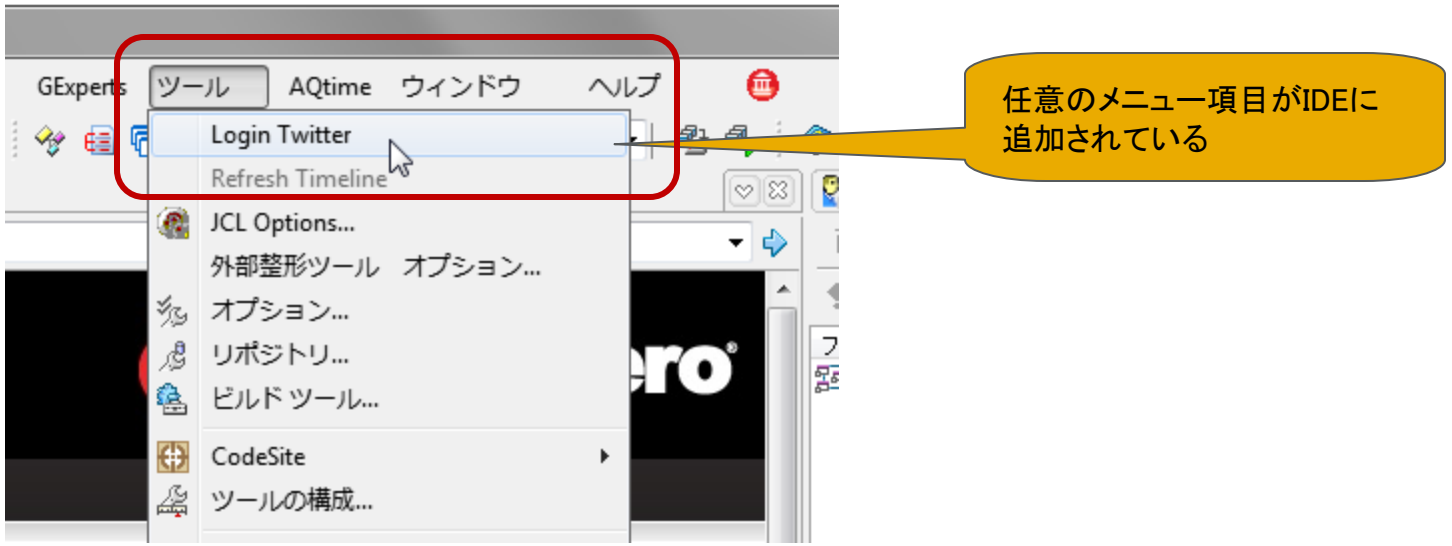
```
//function UrlEncode(const S: String): String;
function UrlEncode(const S: RawByteString): String;
```

関数UrlEncodeの引数をStringから
RawByteStringに修正

Twitterアプリケーションの登録

- https://dev.twitter.com/apps/newにアクセス
  - アプリケーションの種類は「クライアントアプリケーション」
- “OAuth”というプロトコルでユーザー認証
  - TTwitterがラップしてくれるのでフル実装する必要なし
- Twitterに登録すると、認証が必要なキーが発行される
  - **Consumer key**
  - **Consumer secret**

IDEのメニューにメニュー項目を追加する



- ウイザードは最も基本的なIOTAWizardより派生
- ウイザードのコンストラクタでデータモジュールを生成
- データモジュールのOnCreateイベントでIDEにメニューを追加

IDEのメインメニューにメニュー項目を追加する

```
// IDEを拡張するウイザードモジュール
type
  TTwitterWizard = class(TNotifierObject, IOTAWizard)

  public
    procedure AfterSave;
    procedure BeforeSave;
    procedure Destroyed;
    procedure Execute;
    procedure Modified;
    function GetState: TWizardState;
    function GetIDString: string;
    function GetName: string;
    constructor Create;
    destructor Destroy; override;
  private
end;

  {
    中略
  }

constructor TTwitterWizard.Create;
begin
  if fMainForm = nil then
    fMainForm := TdbmMain.Create(nil);
end;

destructor TTwitterWizard.Destroy;
begin
    FreeAndNil(fMainForm);
End;
```

Ttwitterコンポーネントを貼り付けたデータモジュールを生成

この部分は後で説明します

```
 // IDEを拡張するパッケージの登録
procedure Register;
var
  IIDEServices: IOTAServices;
  ModuleServices: IOTAModuleServices;
  I: Integer;
Begin


  // Wizardの登録
  RegisterPackageWizard(TTwitterWizard.Create);


  // IDEのノーティファイアの登録
  IIDEServices := BorlandIDEServices as IOTAServices;
  Assert(Assigned(IIDEServices), 'IOTAServices not
available');
  IDENotifierIndex :=
    IIDEServices.AddNotifier(TTwitterIDENotifier.Create);

  // ソースエディタのノーティファイアの登録
  SourceEditorNotifiers := TList.Create;
  ModuleServices := BorlandIDEServices as
    IOTAModuleServices;
  if ModuleServices.ModuleCount = 0 then
    Exit;
  for I := 0 to ModuleServices.ModuleCount - 1 do
    InstallEditorNotifiers(ModuleServices.Modules[I]);

end;
```

ウイザードをIDEに登録

IDEのメインメニューにメニュー項目を追加する

```
procedure TdbmMain.DataModuleCreate(Sender: TObject);
var
  i: Integer;
  IDE: INTAServices;
  ToolsMenu: TMenuItem;
  AppIniPath: string;
  Ini: TIniFile;
begin

  {
    中略
  }

  // メインメニューの[ツール|オプション]を探す
  IDE := (BorlandIDEServices as INTAServices);

  ToolsMenu := nil;
  with IDE.MainMenu do begin
    for I := 0 to Items.Count - 1 do begin
      if AnsiSameText(Items[I].Name, 'ToolsMenu') then
        ToolsMenu := Items[I];
    end;
  end;
```

INTAServicesがIDEの「本体」

MainMenuプロパティがIDEのメインメニューVCLのTMainMenu型

```
  // メニューアイテムの重複チェック(デバッグ時対策)
  if (ToolsMenu.Find('Login Twitter') <> nil) then
    Exit;
  if (ToolsMenu.Find('Logout Twitter') <> nil) then
    Exit;

  // [ツール]の先頭にメニューアイテムを登録
  if miLogin = nil then begin
    miLogin := TMenuItem.Create(nil);
    with miLogin do begin
      Name    := 'RSTwitLogin';
      Caption := 'Login Twitter';
      OnClick := miLoginClick;
      ToolsMenu.Insert(0, miLogin);
    end;
    if Twitter1.LoggedIn then
      miLogin.Caption := 'Logout Twitter'
    else
      miLogin.Caption := 'Login Twitter';
  end;
  {
    以下略
  }
end;
```

TMenuItemの動的生成

TTwitterでTwitterへログインする

- ApplicationDetails プロパティ：認証キー
  - ApiKey：取得したConsumer key
  - ApiSecret：取得したConsumer secret
- TokenStore プロパティ：認証情報の保存先
  - TTwitterTokenINIStore：INIファイルに保存
  - TTwitterTokenRegistory：レジストリに保存
- UserName / Passwordプロパティ
- Loginメソッド：ログインの実行

Twitterへのログイン

```
IniPath := GetAppIniPath();
Ini := TIniFile.Create(IniPath);

f := TFrmLogin.Create(self);
f.edtUserName.Text := Ini.ReadString('User', 'UserName', '');

f.edtConsumerKey.Text     := Ini.ReadString('ApplicationDetails', 'ApiKey', '');
f.edtConsumerSecret.Text  := Ini.ReadString('ApplicationDetails', 'ApiSecret', '');
f.edtCallbackURL.Text     := Ini.ReadString('ApplicationDetails', 'CallbackURL', '');

// ログインダイアログ
if f.ShowModal = mrOk then begin
  Twitter1.Username := f.edtUserName.Text;

  // 認証済みかどうかはリクエストトークンの有無でチェック
  Token := Ini.ReadString(f.edtUserName.Text, 'Token', '');
  if Length(Token) <> 0 then begin
    // 認証済みならばログインパスワードは不要(空白でOK)
    Twitter1.Password := ''
  end
  else begin
    // 未認証ならば、パスワードとかAPIキーの類いが必要
    if f.chkApplicationDetails.Checked then begin
      Twitter1.Password := f.edtPassword.Text;
      with Twitter1.ApplicationDetails do begin
        ApiKey      := f.edtConsumerKey.Text;
        ApiSecret   := f.edtConsumerSecret.Text;
        CallbackURL := f.edtCallbackURL.Text;
      end;
    end;
  end;
```

アプリが認証済みでかつ、Twitterへのログインが成功していれば、リクエストトークンが発行される

認証済みで無い場合はパスワード、Consumer key 、Consumer secret が必要

Twitterへのログイン（続き）

```
// ログイン
if Twitter1.Login then begin
  // 成功!!
  Ini.WriteString('User', 'UserName', f.edtUserName.Text);

  // キーなども保存
  if f.chkApplicationDetails.Checked then begin
    Ini.WriteString('ApplicationDetails', 'ApiKey', f.edtConsumerKey.Text);
    Ini.WriteString('ApplicationDetails', 'ApiSecret', f.edtConsumerSecret.Text);
    Ini.WriteString('ApplicationDetails', 'CallbackURL', f.edtCallbackURL.Text);
  end;
  miLogin.Caption := 'Logout Twitter';
  miTweet.Enabled := true;
  miRefreshTimeLine.Enabled := true;

  // タイムラインを表示
  RefreshTimeLine;
end;
```

LoginメソッドでTwitterへログイン
アプリが未認証ならば、Twitter側の認証画面が表示されるので、再度、ユーザー名とパスワードを入力

タイムラインの表示

- 表示先はメッセージウィンドウ
  - IMessageServicesを使用する
- タイムラインの取得
  - Timelinesプロパティ：Twitterのタイムライン
  - TTwitterTimelineクラスのGetTimelineメソッド

タイムラインの表示

```
procedure TdbmMain.RefreshTimeLine();
var
  IMessageServices: IOTAMessageServices;
  TimeLineGroup: IOTAMessageGroup;
  ref: pointer;
  I: Integer;
  TwitMessage:  string;
begin
  // メッセージビューに関するサービスを取得する
  IMessageServices := BorlandIDEServices as IOTAMessageServices;

  // メッセージビューにタブを追加し、それを表示
  TimeLineGroup := IMessageServices.AddMessageGroup('My Timeline');
  IMessageServices.ShowMessageView(TimeLineGroup);
  IMessageServices.ClearMessageGroup(TimeLineGroup);

  // タイムラインを取得
  Twitter1.Timelines.Home.GetTimeline;

  // タイムラインを表示(TTwitterのTimelinesプロパティ)
  with Twitter1.Timelines.Home do begin
    for I := 0 to TweetCount - 1 do begin
      // JVCL由来の簡易HTMLパース不等号とかの対応
      TwitMessage := HTMLPrepareText(TweetByIndex[i].Msg);

      // メッセージウィンドウにツイートを表示
      IMessageServices.AddToolMessage('', TwitMessage, TweetByIndex[i].User.Name,
          0, 0, nil, ref, TimeLineGroup);
    end;
  end;
end;
```

メッセージウィンドウの生成

生成したメッセージウィンドウに内容を表示

つぶやいてみる

- 「つぶやき」はコードエディタでセレクションしたのをポストする
- コードエディタに[Tweet]というメニューを追加する

コードエディタのメニューに項目を追加する

1. IDEにノーティファイヤを登録
2. ファイルがロードされると、IDEはコードエディタを生成
3. ノーティファイヤがファイルのロードを検知
4. 生成したコードエディタにノーティファイヤを登録
5. ノーティファイヤがコードエディタのアクティブ化を検知
6. アクティブなコードエディタのメニューに項目を追加
7. 終了時の後始末（ノーティファイヤ）

IDEへのノーティファイヤの登録

- IDEへのノーティファイヤの登録はRegister関数で行う

```
procedure Register;
var
  IIDEServices: IOTAServices;
  ModuleServices: IOTAModuleServices;
  I: Integer;
begin
  // Wizardの登録
  RegisterPackageWizard(TTwitterWizard.Create);

  // IDEのノーティファイヤの登録
  IIDEServices := BorlandIDEServices as IOTAServices;
  Assert(Assigned(IIDEServices), 'IOTAServices not available');
  IDENotifierIndex := IIDEServices.AddNotifier(TTwitterIDENotifier.Create);

  // 以下略
end;
```

IOTAServicesはIDE本体のサービス
AddNotifierメソッドでノーティファイヤを登録

IOTAIDENotifierクラス

- IDEのファイル状態の変化を検知するノーティファイヤ
- 状態が変化するとFileNotification関数が呼ばれる
  - プロジェクトのオープン
  - ファイルのオープン、クローズ
  - パッケージのインストール、アンインストール
  - など

```
// IDEのノーティファイヤ
type
  TTwitterIDENotifier = class(TNotifierObject, IOTANotifier, IOTAIDENotifier)
  private
    // IOTAIDENotifier
    procedure FileNotification(NotifyCode: TOTAFileNotification;
        const FileName: string; var Cancel: Boolean);

    procedure BeforeCompile(const Project: IOTAProject; var Cancel: Boolean); overload;
    procedure AfterCompile(Succeeded: Boolean); overload;
end;
```

IDEにファイルがロードされた場合の検知

- FileNotification関数の実装部分

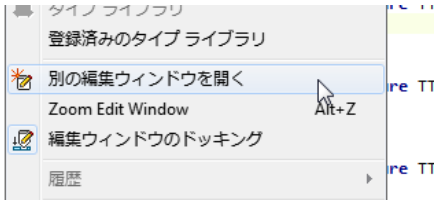
```
// IDE でファイルについての何らかのイベントが発生すると呼び出される
procedure TTwitterIDENotifier.FileNotification(NotifyCode: TOTAFileNotification;
        const FileName: string; var Cancel: Boolean);
var
  ModuleServices: IOTAModuleServices;
  Module: IOTAModule;
begin
  // ソースエディタにファイルが読み込まれたら、ソースエディタにメニューアイテムを登録する
  if NotifyCode =  ofnFileOpened then begin
    ModuleServices := BorlandIDEServices as IOTAModuleServices;
    Module := ModuleServices.FindModule(FileName);
    if Assigned(Module) then begin
      InstallEditorNotifiers(Module);
    end;
  end;
end;
```

1. IOTAModuleServicesでエディタ関連のサービスを取得
2. FindModuleメソッドでファイル名に対応する「モジュール」を取得
3. モジュールにノーティファイヤを登録
   （内部関数：InstallEditorNotifiers）

コードエディタの構成要素

- モジュール（IOTAModule）
  - IDEがアクセスする抽象的なエディタの組み合わせ
  - Delphi/C++Builderのプロジェクトにおけるユニットに相当
  - テキストエディタとフォームエディタなどから構成

- ビュー（IOTAEditView）
  - テキストエディタに対して複数個存在
    - [別の編集ウィンドウを開く]で分割
  - 編集部分の抽象的なレイヤ
  - カーソルの位置、しおりなどの情報を持つ



コードエディタの構成要素

- エディットウィンドウ（INTAEditWindow）
  - エディタの「本体」
  - メニューやフォームにアクセス
- バッファ（ITOAEditBuffer）
  - 編集中ファイルの内部バッファ
  - 文字のセレクション情報を保持
  - 内部的にはUTF-8で処理

エディタへノーティファイヤを登録

- エディタは複数個あるので、それぞれにノーティファイヤを登録しないといけない

```
procedure InstallEditorNotifiers(Module: IOTAModule);
var
  I: Integer;
  SourceEditor: IOTASourceEditor;
begin
  for I := 0 to Module.ModuleFileCount - 1 do
    if Supports(Module.ModuleFileEditors[I], IOTASourceEditor, SourceEditor) then
    begin
      SourceEditorNotifiers.Add(TTwitterEditorNotifier.Create(SourceEditor));
      SourceEditor := nil;
    end;
end;
```

IOTAEditorNotifierクラス

- コードエディタの状態変化を検知するノーティファイヤ

```
// ソースエディタのノーティファイヤ
type
  TTwitterEditorNotifier = class(TNotifierObject, IOTANotifier, IOTAEditorNotifier)
  private
    FEditor: IOTASourceEditor;
    FIndex: Integer;

    // IOTAEditorNotifier
    procedure ViewActivated(const View: IOTAEditView);
    procedure ViewNotification(const View: IOTAEditView; Operation: TOperation);

    // 外部からノーティファイヤを「削除」する
    procedure Destroyed;
  public
    constructor Create(AEditor: IOTASourceEditor);
    destructor Destroy; override;
end;
```

コードエディタのメニューを変更する

```
procedure TTwitterEditorNotifier.ViewActivated(const View: IOTAEditView);
var
  EditWindow: INTAEditWindow;
  EditWindowForm: TCustomForm;
  EditorLocalMenu: TComponent;
  ParentMenu: TMenu;
begin
  EditWindow := View.GetEditWindow;
  if not Assigned(EditWindow) then Exit;

  EditWindowForm := EditWindow.Form;
  if not Assigned(EditWindowForm) then Exit;

  // エディタのメニューを取得
  EditorLocalMenu := EditWindowForm.FindComponent('EditorLocalMenu');
  if not Assigned(EditorLocalMenu) then Exit;

  if (EditorLocalMenu is TMenu) then begin
    with fMainForm do begin
      // アクティブでなくなったコードエディタからメニューアイテムを削除
      ParentMenu := miTweet.GetParentMenu;
      if Assigned(ParentMenu) then
        ParentMenu.Items.Remove(miTweet);
      // アクティブになったコードエディタのメニューにメニューアイテムを登録
      TMenu(EditorLocalMenu).Items.Add(miTweet);
    end;
  end;
end;
```

抽象的なビューからコードエディタを構成するVCLオブジェクトを取得

エディタへのアクセス

```
procedure TdbmMain.miTwitClick(Sender: TObject);
var
  EditBuffer: IOTAEditBuffer;
  TweetMessage: string;
begin

  if Twitter1.LoggedIn then begin
    // 選択範囲の内容を取得
    EditBuffer := (BorlandIDEServices as IOTAEditorServices).TopBuffer;


    TweetMessage := Trim(EditBuffer.EditBlock.Text);

    // つぶやきをポスト
    Twitter1.PostTweet(TweetMessage);

    // タイムラインを更新
    RefreshTimeLine;
  end;
end;
```

エディタサービスから直接バッファを取得することもできる

後始末（アドオンを削除した場合の対応）

- finalization節でノーティファイヤを削除

```
// ソースエディタに登録してあるノーティファイヤを削除
procedure ClearEditorNotifiers;
var
  I: Integer;
begin
  if Assigned(SourceEditorNotifiers) then
    for I := SourceEditorNotifiers.Count - 1 downto 0 do
      TTwitterEditorNotifier(SourceEditorNotifiers[I]).Destroyed;
end;

// IDEに登録してあるノーティファイヤを削除する
procedure ClearIDENotifiers;
var
  Services: IOTAServices;
begin
  if IDENotifierIndex <> -1 then
  begin
    Services := BorlandIDEServices as IOTAServices;
    Assert(Assigned(Services), 'IOTAServices not available');
    Services.RemoveNotifier(IDENotifierIndex);
  end;
end;

finalization
  ClearIDENotifiers;
  ClearSourceEditorNotifiers;
  FreeAndNil(SourceEditorNotifiers);
```

今後の課題

- タイムラインの保存、並び替え
- 一定時間でタイムラインを更新
- ツイートの返信
  - メッセージウィンドウにメニューを追加したいな…

embarcadero
Developer Camp

まとめ

まとめ

- Open Tools APIは「意外と」敷居が低い
- 予想以上に強力
- Try and errorで何とかなる
- ToolsAPI.pasのハックは必須!!

アフターフォローとか

- 今日のアフターフォローは以下で行います
  - 自分のBlog：http://d.hatena.ne.jp/A7M/
  - Twitterハッシュタグ: #dcamp_jp
  - Twitter ID : A7M3J

embarcadero
Developer Camp

Q & A

最後に

ご清聴ありがとうございました!!

<(_ _)>

embarcadero
Developer Camp

付録1
参考になるリソース

Open Tools APIについてのリソース

- オンラインヘルプのIDE の拡張（Tools API）
- C++Builder6開発者ガイド 58章「IDEの拡張」

- Delphi 7のヘルプファイル
- C++Builder 6のヘルプファイル
  - Delphi/C++Builder XEがあれば入手可能

Open Tools APIについてのリソース

- Opening Doors:
  Notes On the Delphi ToolsAPI by its Creator
  - http://edn.embarcadero.com/article/20360

- Opening Doors:
  Notes On the Delphi ToolsAPI by its Creator - Part 2
  - http://edn.embarcadero.com/article/20419

Open Tools APIについてのリソース

- Open Tools API を使用して Delphi IDE をより効果的にする方法を探る (英語)
  - http://edn.embarcadero.com/article/41101

- Delphi IDE を拡張する(英語)
  - http://edn.embarcadero.com/article/41329

Open Tools APIについてのリソース

- GExperts (http://www.gexperts.org/)
  - GExpertsのソースコード
  - Erik's Open Tools API FAQ
    - http://www.gexperts.org/opentools/
- MustangpeakのOpen Tools APIについての情報
  - http://www.mustangpeak.net/
- Tempest SoftwareのOpen Tools APIについての情報
  - http://www.tempest-sw.com/opentools/

Open Tools APIについてのリソース

- Borland C++Builder 6 Developer's Guide
  - (ISBN:0672324806)
- Delphiコンポーネント設計＆開発完全解説
  - (ISBN:4844317466)

Open Tools APIについてのリソース

- 自分のBlogのうち、以下のエントリーがOTAについて
  - Open Tools API その0：概要
    - http://d.hatena.ne.jp/A7M/20101106/1289033279
  - Open Tools API その1："Hello,World!"を表示してみる
    - http://d.hatena.ne.jp/A7M/20101114/1289711588
  - Open Tools API その2：メニューの差し替えとノーティファイアインターフェース
    - http://d.hatena.ne.jp/A7M/20101231/1293763501
  - Open Tools API その3：ソースエディタへのアクセス
    - http://d.hatena.ne.jp/A7M/20110205/1296893189
  - Open Tools API その4：続・ソースエディタへのアクセス　ソースエディタの「中身」をいじる
    - http://d.hatena.ne.jp/A7M/20110212/1297497495

embarcadero
Developer Camp

付録2
C++Builder用サンプル

ヘッダファイル（OTAWizard.h）

```
//---------------------------------------------------------------------------
#ifndef OTAWizardH
#define OTAWizardH
//---------------------------------------------------------------------------

#include <ToolsAPI.hpp>

class PACKAGE TOTAWizard : public TNotifierObject,  public IOTAMenuWizard
{
  typedef TNotifierObject inherited;
public:
  // IOTAWizard
  virtual UnicodeString __fastcall GetIDString();
  virtual UnicodeString __fastcall GetName();
  virtual TWizardState __fastcall GetState();
  virtual void __fastcall Execute();
  // IOTAMenuWizard
  virtual UnicodeString __fastcall GetMenuText();
  void __fastcall AfterSave();
  void __fastcall BeforeSave();
  void __fastcall Destroyed();
  void __fastcall Modified();
protected:
  // IInterface
  virtual HRESULT __stdcall QueryInterface(const GUID&, void**);
  virtual ULONG __stdcall AddRef();
  virtual ULONG __stdcall Release();
};

#endif
```

ソースファイル（OTAWizard.cpp）

```
//---------------------------------------------------------------------------
#include <vcl.h>
#include <ToolsAPI.hpp>
#include <tchar.h>
#pragma hdrstop

#include "OTAWizard.h"
//---------------------------------------------------------------------------
 #pragma package(smart_init)

// パッケージの登録（namespaceはクラス名の先頭のみを大文字にする）
namespace Otawizard
{
void __fastcall PACKAGE Register()
{
  RegisterPackageWizard(new TOTAWizard());
}
}

//---------------------------------------------------------------------------
ULONG __stdcall TOTAWizard::AddRef()
{
  return inherited::_AddRef();
}

//---------------------------------------------------------------------------
ULONG __stdcall TOTAWizard::Release()
{
  return inherited::_Release();
}
```

ソースファイル（OTAWizard.cpp）

```
//---------------------------------------------------------------------------
HRESULT __stdcall TOTAWizard::QueryInterface(const GUID& iid, void** obj)
{
  if (iid == __uuidof(IOTAMenuWizard)) {
    *obj = static_cast<IOTAMenuWizard*>(this);
    static_cast<IOTAMenuWizard*>(*obj)->AddRef();
    return S_OK;
  }
  if (iid == __uuidof(IOTAWizard)) {
    *obj = static_cast<IOTAWizard*>(this);
    static_cast<IOTAWizard*>(*obj)->AddRef();
    return S_OK;
  }
  return inherited::QueryInterface(iid, obj);
}
//---------------------------------------------------------------------------
void __fastcall TOTAWizard::AfterSave()
{
}
//---------------------------------------------------------------------------
void __fastcall TOTAWizard::BeforeSave()
{
}
//---------------------------------------------------------------------------
void __fastcall TOTAWizard::Destroyed()
{
}
//---------------------------------------------------------------------------
void __fastcall TOTAWizard::Modified()
{
}
```

ソースファイル（OTAWizard.cpp）

```
//---------------------------------------------------------------------------
UnicodeString __fastcall TOTAWizard::GetIDString()
{
  return _T("DevCamp19CBWizard1.0");
}
//---------------------------------------------------------------------------
UnicodeString __fastcall TOTAWizard::GetName()
{
  return _T("Open Tools API C++Builder Sample Wizard");
}
//---------------------------------------------------------------------------
TWizardState __fastcall TOTAWizard::GetState()
{
  TWizardState result;
  result << wsEnabled;
  return result;
}
//---------------------------------------------------------------------------
UnicodeString __fastcall TOTAWizard::GetMenuText()
{
  return _T("C++BuilderでHello,world");
}
//---------------------------------------------------------------------------
void __fastcall TOTAWizard::Execute()
{
  ShowMessage(_T("C++Builderから こんにちは、世界！"));
}
```