embarcadero
Developer Camp

【B4】C++Builderテクニカルセッション

# 「FireMonkeyクロス開発テクニカルエッセンス - アニメーションデータ作成ツール開発事例から」

株式会社ウェブテクノロジ
R&Dグループ
橋本 孔明(Yoshiaki Hashimoto)

# はじめに

# 自己紹介

橋本 孔明（はしもと よしあき）　hashimoto@webtech.co.jp

- 経歴
  - 勘違いからPC入門と同時にプログラミング入門
  - 中高生時代、VC++、Delphiなどいろいろな環境でオンラインソフトを作っていたことがきっかけで入社
- 現在
  - Windows・Mac・スマートフォンなどなんでも手がける
  - プロダクト開発のほか、各種プラットフォームおよび開発環境の調査や導入決定などにも携わる
  - 基本的にC++屋（STL/Boost/C++11などを多用）
- その他
  - 趣味人間（鉄道、釣り、アクアリウムとか）

# 会社紹介 - 株式会社ウェブテクノロジ

各種製品・サービスの企画・開発・販売・サポート

- 画像最適化ツール『OPTPiX imésta』
  ゲーム業界国内シェア100%
- スプライトアニメーション作成ツール『SpriteStudio』
- 携帯電話向け画像最適化ASP『OPTPicture』
- 絵を描かずにマンガが作成できる『コミPo!』

受託開発

- 大手電機メーカー向け データ放送用画像変換モジュール
- 大手コンテンツプロバイダ向け 高機能画像変換システムの提供
- 大手光学機器メーカー 複合機内蔵PDF生成用減色モジュールの提供

# アジェンダ

1. ソーシャル開発におけるプラットフォームの変遷
2. 「SpriteStudio」の紹介
3. クロスプラットフォーム開発環境の模索
4. C++Builder XE2＆FireMonkeyとの出会い
5. FireMonkey苦労話（今回のメイン）
6. まとめ

embarcadero
Developer Camp


# ソーシャル開発における プラットフォームの変遷

# ソーシャル開発環境の変遷

- ソーシャルゲーム：iOSデバイス、Androidデバイスなどで動作するゲームの呼称
  - 必ずしもネットワーク要素を含まない
- 従来、ゲームプログラム開発はWindows環境が主流
- iOSの台頭により、MacOS X上での開発が必須に
- MacOS X上ではAndroid開発（Eclipse）も行える
  - すべてをMac上で済ませるベンダーが増えた
- 開発ツール類のMac対応要望が増加

embarcadero
Developer Camp


# 「SpriteStudio」の紹介

# SpriteStudioとは（1）

**SpriteStudio（スプライトスタジオ）は、汎用的なアニメーションデータ作成ツールです。**


| for Designers | for Programmers | for All Developers |
|---|---|---|
| •ドラッグ &ドロップなど直感的な操作で、アニメーションデータの作成が可能に<br>•タイムチャートとグラフでアニメーションデータを視覚的に管理 | •動きを付ける作業をデザイナに任せることにより、負担が軽減<br>•SDKを使って、アニメーションデータのコンバータにも簡単に作成可能 | •SpriteStudioが、アニメーションの開発効率を向上させる！ |

http://www.webtech.co.jp/spritestudio/

# SpriteStudioとは(2)

- スプライトアニメーション編集ツール
- 著名コンシューマーゲーム、ソーシャルゲーム等での採用事例多数

# SpriteStudioとは（3）

- リリース中のバージョンはWindows専用
  - VC++、ネイティブC++およびWin32 APIで開発
- MacOS Xにて、仮想環境上のWindowsによる動作も可能だが…
- Mac版の要望を多数頂戴
- 開発決定

embarcadero
Developer Camp

# デモンストレーション

# SpriteStudio
# 開発途上版

embarcadero
Developer Camp


# クロスプラットフォーム開発環境の模索

# 要件定義

- MacOS XとWindowsのGUIの違いを吸収したい
  - メニューバー位置などはOSネイティブに合わせたい
- OSネイティブではなく、ダークグレーベースの外観としたい
  - 他のデザイナー向けプロフェッショナルツールでも導入事例が多い（Adobe CS6、3Dツールなど）
  - 見た目が良いというだけではなく、デザイナーの長時間利用に耐え、目に優しい色遣いとなっている
- プロフェッショナルユースに耐えうるUX設計
  - 社内ツールや簡単なフリーソフトなどとは違います

# クロスプラットフォーム開発環境の模索

いくつかの候補を調査

- C/C++＆ネイティブAPI
- Mono
- Qt
- wxWidgets
- C++Builder XE2

※主観がかなり入っています

# C/C++＆ネイティブAPI

- ○ UIはそれぞれのプラットフォームに合わせて細かく設計可能
- ○ 確実な開発環境(VC++/Xcode)の存在
- × 標準コントロールのみを使用していると「古くさい」UIになる（特にWindows）
- × 標準コントロールのカスタマイズが難しく、なんでもかんでも自作していく羽目に
  - 昔は実際にやっていましたが…
- × MacOS Xネイティブ開発技術の習得
  - ただしiOS学習である程度のObjective-Cの知識有り

# Mono

C#および.NETを用いたクロス対応開発環境
IDEとしてMonoDevelopを使用

- ○ C#でRAD風の開発が可能(Windows Forms)
- ○ .NET(Windows Forms)開発知識が転用可能
- × .NETの最新潮流であるWPFは使用できない
- × UIがOSネイティブに準じてカスタマイズしにくい
- × 中間コードで出力されることによるソフトウェアプロテクトへの不安

# クロス対応GUIフレームワーク

C++で記述可能なGUIフレームワーク
（Qt、wxWidgetsなど）

- ○ C++やネイティブAPIが使用できる
- × 商用サポートを受けるためにはライセンス料金が高額だったり、そもそもサポートが無かったり
- × 他はネイティブと同じデメリット
- × 日本語情報の不足（微妙に不安）
- × 試そうとしたがビルド環境構築(OSX)で挫折

# C++Builder XE2

実は当初候補に入っておらず、他のダメ出しをした後で存在を思い出し、検討を開始

- ○ C++で開発可能
- ○ 昔のVCLの知識が役立つ？
- ○ Windows上で同時開発し、MacOS Xプログラムのデバッグがリモートで可能
- ○ 柔軟なGUI設計が可能
- × Mac開発の実績がいまいち不明

embarcadero
Developer Camp


# C++Builder XE2 & FireMonkey との出会い

# 試用版でFireMonkeyプラットフォームを学習

- Delphi/C++Builderの知識はVer6くらいで止まっていた＆かなり忘れていた
- 久々に起動したC++Builderはすっかり変わって…
- FireMonkeyプラットフォームはVCLの面影がなんとなく残っていた
- スタイル編集により、GUI要素の外観カスタマイズが自在→これはいけそう

# 早速開発に着手

- アプリケーションの性格上、いきなりOpenGLの導入など高度な要素から挑戦
  - 後から「無理」と判明すると困る
- 買って数日でランタイムライブラリの内部を覗いたりする羽目に…
  - C++Builder単体より、ランタイムソースの付いているRAD Studioを持っていた方がよさそう
- ランタイムはDelphi製
  - 忘れかけていたDelphi言語知識をなんとか思い出す

embarcadero
Developer Camp


# FireMonkey
# 苦労話
# （今回のメイン）

# 注意事項

- 基礎知識的なものは省略します
- 独自調査に基づく項目が多く含まれます
- 他開発環境(VC++)からの移行も重視しています
- 弊社の開発スタイルの都合に合わせるための対策も含まれます
  - 実際には別の解決策があったり、公式には他の方法が推奨されている可能性があります
- XE3になると解消する問題もありそうです
- WindowsをWin、MacOS XをOSXと省略

embarcadero
Developer Camp

# C++環境レベルでの違い

# C++環境レベルでの違い(1)

- ビルド分けしたい場合
  - #ifdef __APPLE__ でOSXを識別
  - #ifdef _WIN32(#ifdef _WINDOWS) でWinを識別
  - Winのx64では_WIN64が加わります(_WIN32も存在)

- CRT関数の違い
  - Windowsで使われるwcscpy_s()などがOSXには無い
    - OSXはPOSIX準拠＋α程度のCRT関数しか無い
    - VC++などで書いたコードの移植時に注意
    - 汎用処理はRTLの機能か、STL・Boostに頼る

# C++環境レベルでの違い(2)

- Win32構造化例外はOSXにはありません
  - アクセス違反などはシグナルハンドラで捕まえます
  - 他コンパイラでtry-catchを流用していたコードも注意
- Boostの一部機能が使えない(XE3で解消？)
  - OSXではスタティックライブラリを要するBoostコンポーネントの大半が使用不可能(ヘルプに記載はあったが…)
  - serialization、filesystemなど
- 文字列リテラルの文字コード
  - Windowsはシステム依存、OSXは基本的にUTF-8
  - この問題は深いので各自研究してください

# C++環境レベルでの違い(3)

wchar_tの違い(公式でも解説されている)

- Winでは2bytes、OSXでは4bytes
- UnicodeString(String)型、WideChar型はどちらのOSでも2bytesベース
- このためOSXではwchar_tを使用するCRT関数・std::wstring・Boostなどへのアクセスが不便
- UnicodeStringからstd::wstringへの変換関数を自前で用意しました(ただし重そう)
- テンプレートにWideCharを適用した型を自分で定義して使う方法もある
  - std::basic_string、boost::basic_formatなど

# コードサンプル

```
inline std::wstring wstring_from_UnicodeString(
    const UnicodeString& str)
{
#ifdef __APPLE__
    return
        (wchar_t*)PUCS4Chars(
            UnicodeStringToUCS4String(str));
#else // 手抜き。将来のiOSやAndroidを考えると非推奨
      return str.w_str();
#endif
}
```

# C++環境レベルでの違い(4)

L”hello” と u”hello” と _D(”hello”)

- L”hello”はWinで2バイト文字、OSXでは4バイト文字
  - 厳密にはwchar_tの配列(環境ごとにサイズ可変)
- u”hello”はOSXでも2バイト
  - 厳密にはchar16_tの配列(C++11で導入)
  - char16の名の通り、どんな環境でも「最低」2バイト
  - 大文字のU””にするとchar32_t
- _D(”hello”)にするとどちらでも2バイト
  - UnicodeStringに渡すならとりあえずこれがいいらしい
  - マクロなので何かあっても後で再定義すればいい

embarcadero
Developer Camp

# OSネイティブAPI
# の
# 呼び出し

# OSネイティブAPIの呼び出し

- FireMonkeyだけでも結構なものは作れる
- 細かいユーザーインターフェースの改良にはどうしてもネイティブAPIのサポートが必要となる
- SpriteStudioの場合、アニメーション再生にOpenGLが必要となった
- OSXのネイティブAPIはObjective-Cだが…？

# OSネイティブAPIの呼び出し：Windows

- 大半がただの関数なのであまり難しくない
  （必要なヘッダをincludeして呼ぶだけ）
- ハンドルの取得方法さえわかれば大丈夫

```
#include <FMX.Platform.Win.hpp>

FmxHandleToHWND(this->Handle);    // TForm*からHWND
(HMENU)(pMenu->Handle);           // TMenuItem*からHMENU
```

# OSネイティブAPIの呼び出し：MacOS X

- Cocoa API：Objective-Cの知識が必要
- ランタイムが、Objective-CクラスをC++クラス（インターフェース）として呼び出せる高度なラッパーを提供している
- Delphi向けのためC++での記述方法がかなり特殊
- 私もまだ完全理解には至っていません
- 公式の説明が見あたらない…
- Carbon APIなら簡単です
  - Win32 APIと同じくC関数呼び出し方式
  - 最新のOS機能には非対応

# コードサンプル

```
// 必要となるヘッダ
#include <FMX.Platform.Mac.hpp>
#include <Macapi.Foundation.hpp>
#include <Macapi.AppKit.hpp>

// TForm*からNSWindowを得る
// Cocoaクラス名に接頭辞_di_が付くものとTが付くものがあり役割も分かれる
_di_NSWindow wnd =
    TNSWindow::Wrap(_di_ILocalObject(
        FmxHandleToObjC(pForm->Handle))->GetObjectID());

// 自アプリのNSBundleを得る(クラスメソッド呼び出し)
// 生のObjCクラスポインタはWrapしてラッパクラス(スマートポインタ)化する
Void* pMainBundle = TNSBundle::OCClass->mainBundle();
_di_NSBundle bundle = TNSBundle::Wrap(pMainBundle);

// 得られたNSWindowインスタンスのメソッドを呼んでみる
// NSString生成にはRTL提供のNSSTR関数が利用可能
// [wnd setTitle: @"Hello World"]; ObjCの記法
wnd->setTitle(NSSTR(_D("Hello World"))); // C++同等に呼べる
```

# コードサンプル

```
// TForm*からNSViewを得る
TObject* pObj =
   _interfaceToObjectCast<TObject>(FmxHandleToObjC(this->Handle));

if(pObj)
{
    boost::scoped_ptr<TRttiContext> pRttiContext(new TRttiContext);
    TRttiType* pRttiType = pRttiContext->GetType(pObj->ClassType());
    if(pRttiType)
    {
        TRttiProperty* pRttiProperty = pRttiType->GetProperty("View");
        if(pRttiProperty)
        {
            _di_NSView view = pRttiProperty->GetValue(pObj).AsInterface();
            return view;
        }
    }
}
```

embarcadero
Developer Camp

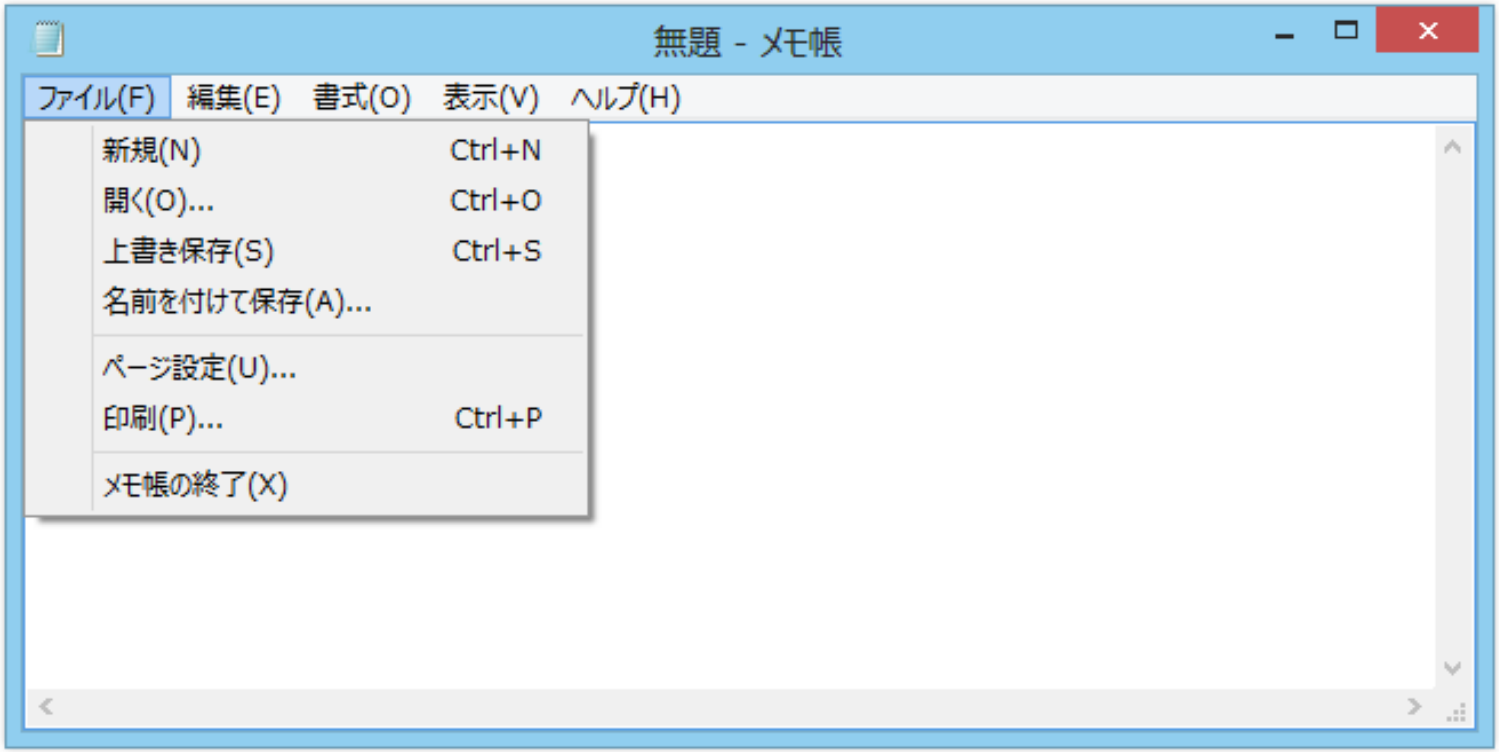
# OSごとの
# メニューの違い

# OSごとのメニューの違い

- 結構マニアックな内容に
- しかし、メニューはアプリケーションGUIの基本中の基本
- 中規模以上のアプリケーションを開発するには、メニューを柔軟に制御する必要はどうしても出てくる
- GUI要素でOSネイティブなのは基本的にメニューだけ（それ以外のコントロールはFireMonkey経由）
- 以上の理由により、複雑になりがちですが、ちょっと詳しくまとめてみました

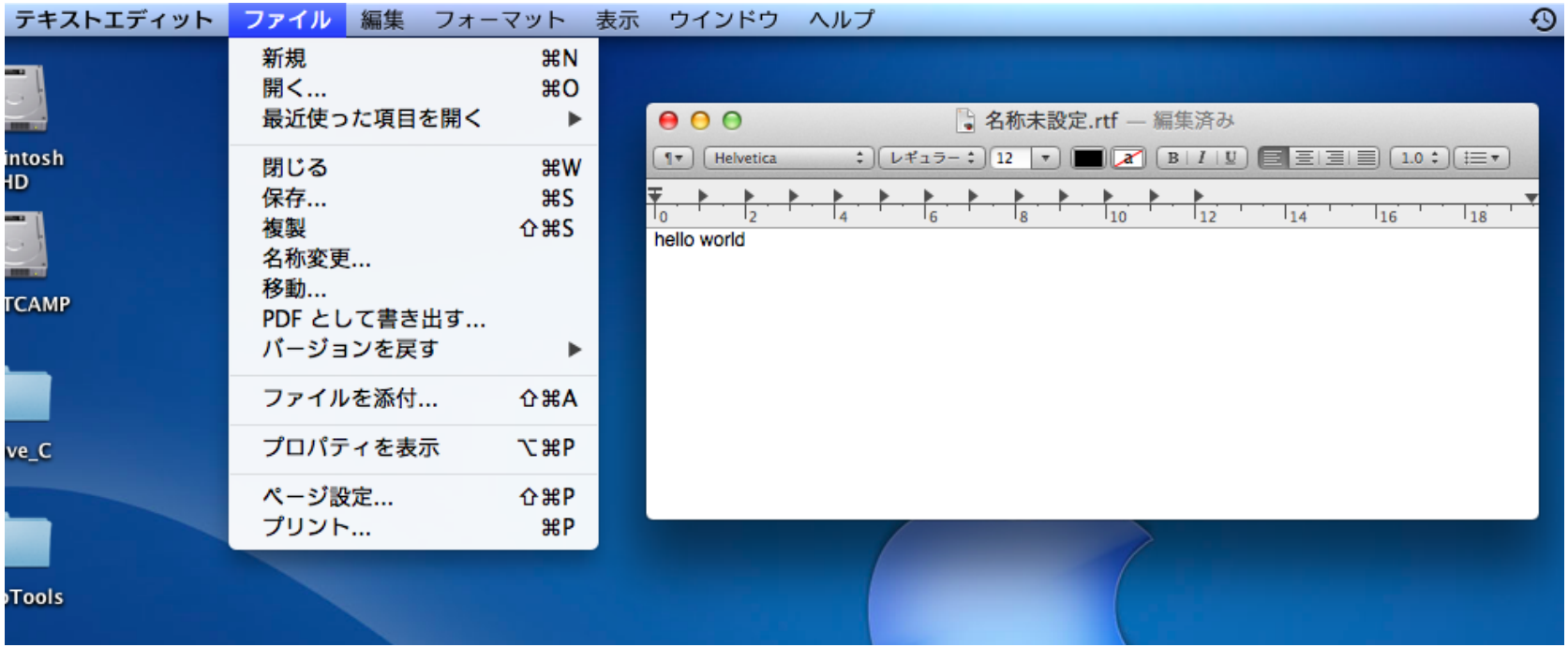
# OSごとのメニューの違い – Windows

ショートカットキーとアクセラレータが別の概念となっている



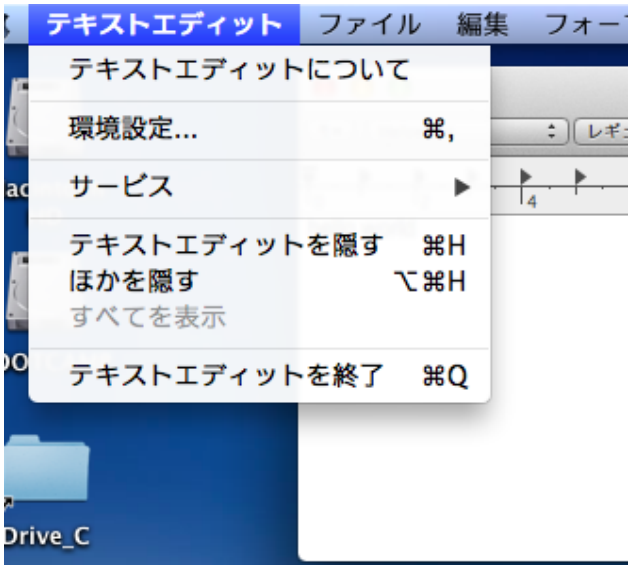
# OSごとのメニューの違い – MacOS X(1/2)

Windowsでいう「ショートカットキー」が無く、
「アクセラレータ」相当を「ショートカット」と呼称



# OSごとのメニューの違い – MacOS X(2/2)

「アプリケーションメニュー」の存在




Windowsでいうところの「システムメニュー」「ツール」「ヘルプ」などの内容に相当

# OSごとのメニューの違い – その他

- メニュー配置の「常識」の違い
  - OSXでは「終了」は「ファイル」の中ではない
  - バージョン情報や環境設定はアプリケーションメニューに入っている
- 同じ内容でも微妙に表現が違う
  - Win：切り取り・コピー・貼り付け
  - OSX：カット・コピー・ペースト
- ショートカット文字列の有無
- Ctrlキー vs. Commandキー

# OSごとのメニューの違い – 解決策

- フォーム上に別々のTMenuBarを作っておき、OSごとに割り付けすることで、ある程度解決可能
- しかし、今後の変更やローカライズを考えるとフォーム上のアイテムでの個別管理はわずらわしい
- そこで、メニュー内容をXML化しリソースに格納
  - 属性としてWindows用・MacOS X用などのフラグを追加
  - ショートカットもOS別に指定可能とした
  - これをRTLのTXMLDocumentで読み込み、TMenuBar/TMenuItemを構築するライブラリを開発

# メニュー定義用XMLサンプル（抜粋）

```
<!-- Commandにコマンド文字列（イベントハンドラ関数に渡される）を指定する -->
<!-- WinShortcutでWindowsショートカットキーのアルファベットを指定する -->
<!-- Accelにアクセラレータキーのキーコンビネーションを文字列で指定する -->
<!-- 一般的な流儀に沿い、WinでCtrl+●、MacでCmd+●をアクセラレータにする（核キーが同じ）場合、
Accelアトリビュートに「Def+」と書いて共通化できる -->
<Item Name="新規作成" Command="New" WinShortcut="N" Accel="Def+N" />

<!-- セパレータを作る場合はこのように指定する -->
<Item Name="-" />

<!-- WinとMacで名称を分けることもできる。Win用にショートカット文字列を入れてしまう
 (WinShortcut省略) ことも可能 -->
<!-- アクセラレータキーをWinとMacで別々に定義（または片方だけ定義）したい場合、WinAccelと
MacAccelを両方定義することで分けられる -->
<!-- OpenDlg=1とすると項目名の後に「...」を付与する -->
<Item WinName="開く(&amp;O)" MacName="開く" Command="Open" WinAccel="Ctrl+O"
    MacAccel="Cmd+O" OpenDlg="1" />
<Item Name="上書き保存" Command="Save" WinShortcut="S" Accel="Def+S" />

<!-- Winのみで表示したい場合はMacHide=1、逆はWinHide=1とする -->
<Item Name="-" MacHide="1" />
<Item Name="Applicationの終了" Command="Exit" WinShortcut="X " MacHide="1" />
```

# OSネイティブメニューへの対応

- 作ったTMenuBarをOSネイティブのメニューに設定する方法
  - 構築完了してから UseOSMenu() メソッドを呼べばOK
  - UseOSMenu()してからアイテム追加しても反映されない

- OSXアプリケーションメニューに設定する方法
  - 別のTMenuBarを構築し、Application->ApplicationMenuItems に代入する

# OSごとのメニューの違い – 実装してみたが

- Winでは特に問題無く動作…した？（後述）
- OSXでトラブル発生
  - XMLのCommandの内容をTMenuItemのTagStringに格納し、共通のOnClickハンドラから読み取って識別しようとした
  - TMenuItemのOnClickに指定したメソッドで、Sender引数をTMenuItem*にキャストして使用したところアクセス違反
  - ランタイムライブラリのソースコードを漁ったり検索したり
  - QCにて既知不具合と判明
    - `http://qc.embarcadero.com/wc/qcmain.aspx?d=105091`

# コードサンプル

```
void __fastcall MenuLoader::MenuItemClick (
    TObject *Sender)
{
    pMenuItem = (TMenuItem*)(Sender);
    ProcessCommand(pMenuItem->TagString); // Error on OSX
}
```

# コードサンプル

```
#include <FMX.Platform.Mac.hpp>
#include <Macapi.Foundation.hpp>
#include <Macapi.AppKit.hpp>

TMenuItem* pMenuItem;
#ifdef __APPLE__

// パラメータクラスが誤って渡されているので、同じメモリモデルのクラスに
// 無理矢理キャストして中身を取り出す（暫定対応）
class TFMXOSMenuItem : public TOCLocal
{
public:
    TMenuItem* FMXMenuItem;
};
TFMXOSMenuItem* pOSXMenuItem = (TFMXOSMenuItem*)Sender;
pMenuItem = pOSXMenuItem->FMXMenuItem;

#else

pMenuItem = static_cast<TMenuItem*>(Sender);

#endif
```

# OSXショートカットキーの対応

「環境設定」に[Cmd+,]を指定したい

- TextToShortcut("Cmd+,"); が動かない
- TextToShortcutはアルファベット程度しか対応していないらしい
- TextToShortcut同等品を自作して差し替え
  （今回のプロジェクトに対応できる程度の簡素なもの）

# メニュー項目の有効無効・チェック対応

- TMenuItemのEnabledやIsCheckedを変更しても正常に反映されない場合がある（Win・OSXとも）
- さらに、はっきり確認できていないが、OSXでIsCheckedを変更すると、メモリ破壊か何かが起こって他のところにトラブルを招く模様？
- これらに対応するにはOSネイティブ処理を呼ぶ必要がある
- このほかツールバーボタンとの共用などもあり、総合的な管理クラスを設計して使っています

# メニュー項目の有効無効・チェック対応：OSX

- チェック対応
  - TMenuItemからCocoaのNSMenuItemオブジェクトを得て、setState(NSOnState または NSOffState)メソッドを呼ぶ
- 無効化対応
  - 次のコードサンプル参照
  - ややこしいです
  - 今覚えなくても「なんとかなる」と記憶しておいてください

# コードサンプル

```
// TMenuItemからNSMenuItemを得る
static _di_NSMenuItem GetNSMenuItem(TMenuItem* pMenuItem)
{
    _di_NSMenuItem item;
    if(pMenuItem->Handle)
    {
        item = TNSMenuItem::Wrap(_di_ILocalObject(
                    FmxHandleToObjC(pMenuItem->Handle))
                        ->GetObjectID());
    }
    return item;
}
```

# コードサンプル（続き）

```
// メニューアイテムを無効にする場合
TMenuItem* pMenuItem = （TMenuItem取得）;
_di_NSMenuItem item = GetNSMenuItem(pMenuItem);
class_addMethod(object_getClass(item->target()),
    sel_getUid("validateMenuItem:"),
    &validateMenuItem, "C@:@");
item->setEnabled(false);

// 無効化対応のためのチェックメソッド用関数
static BYTE validateMenuItem(void* pthis, SEL cmdsel,
    void* sender)
{
    _di_NSMenuItem item = TNSMenuItem::Wrap(sender);
    return item->isEnabled() ? 1 : 0;
}
```

# メニュー項目の有効無効・チェック対応：Win

- チェック対応
  - TMenuItemのIsCheckedを変更するだけで対応可能（ネイティブ対応不要）
- 無効化対応
  - 次のコードサンプル参照
  - Windows APIを直接使用します
  - OSXほどややこしくないです

# コードサンプル

```
// メニューアイテムを無効にする場合
TMenuItem* pMenuItem = （TMenuItem取得）;

MENUITEMINFO mii;

ZeroMemory(&mii, sizeof(mii));
mii.cbSize = sizeof(mii);
mii.fMask = MIIM_STATE;
mii.fState = MFS_GRAYED; //有効化ならMFS_ENABLED

SetMenuItemInfo((HMENU)pMenuItem->Handle,
    (UINT_PTR)pMenuItem, FALSE, &mii);
```

# メニュー:その他

- ツールバー等とも連携できる、アクションリストが欲しい
  - XE3でサポート予定らしい？
  - 現状では自作した
  - 前述の不具合なども治っているといいのですが
- メニュー項目へのビットマップ表示ができない
  - OSXでは使わないがWindowsでは標準的
  - これもネイティブコードを使えば可能
  - XP以前とVista（+AeroGlass）以降で扱いが異なる

embarcadero
Developer Camp

# その他Tips

# その他Tips(1)

GPUアクセラレーションをオフにする

- SpriteStudioのようにフォーム内のコントロールが多いとかえって遅くなるようです
- OpenGLとも動作が衝突するのでオフにします
- **FMXMain()の最初で**
  ```
  GlobalUseDirect2D = false;
  ```
  とすればオフにできます（Winのみ）
- OSXでは方法が無いようです（けっこう重い）

# その他Tips(2)

コントロールフォーカスエフェクトをオフにする

- デフォルトではフォーカスを得たコントロールの周囲にぼかしエフェクトが付きます
- これがかなり重い
- **FMXMain()の最初で**
  ```
  GlobalDisableFocusEffect = true;
  ```
  とすればオフにできます

# その他Tips(3)

OSXでdynamic_castがおかしい

- 複数のコントロールからイベントハンドラを共有している場合など、Senderを当該コントロールの型にdynamic_castして判定したいことがある
- OSXで使ったら落ちた
- 自分でnewしたもの同士だと問題ない
- ランタイムが生成したインスタンスがRTTIに対応できていない？
- static_castし、Tagで識別するなどして対応

# コードサンプル

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
    TForm* pForm = dynamic_cast<TForm*>(Sender);
    if(pForm)
    {
        pForm->Caption = _D("Hello World");
    }
}

void __fastcall TForm1::ItemClick(TObject *Sender)
{
    if(dynamic_cast<TButton*>(Sender))
    {
        // ボタンだった場合の処理
    }
    else if(dynamic_cast<TMenuItem*>(Sender))
    {
        // メニュー項目だった場合の処理
    }
}
```

# その他Tips(4)

## OSXでのアプリケーション終了の挙動

- OSXアプリはウィンドウを閉じただけでは終了しない
- OnCloseイベントでメインウィンドウを隠しておき、OSから通知を受けたら再表示する
- アプリケーションメニューの「終了」で完全終了
- 詳しくは前回のDevCamp資料にあった、ような…

# その他Tips(5)

## その他

- ダイアログボックスでのボタンの順序
  - GUIデザインはオリジナルで作っても、こういうところは慣れが出るので標準からあまり弄らないほうがいい
- IDEおよびエディタの動作
  - 謎のコンパイルエラー・リンクエラーが出たら再起動
  - 外部でソース編集する場合は齟齬に注意

embarcadero
Developer Camp


# まとめ

# まとめ

- 一部愚痴っぽくなってしまった感もありますが、C++Builder XE2のクロスプラットフォーム対応は間違いなくすばらしいものです
- クロス開発で「一番大変な部分」の大半がFireMonkeyによって片付けられます
- プロフェッショナルユースに耐えるアプリケーション開発を行うためには、デベロッパー側での工夫や調整も必要です
- XE3での改良にも期待します

embarcadero
Developer Camp

# Q & A

embarcadero
Developer Camp

# ありがとうございました