【B6】 Delphi Prismテクニカルセッション

EMBARCADERO TECHNOLOGIES

DEVELOPER CAMP

.NET/Mono次世代開発ソリューション Delphi Prismの概要

エンバカデロ・テクノロジーズ エヴァンジェリスト
高橋智宏

アジェンダ

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- 製品概要
- Delphi Prism言語
- Mono
- dbExpress
- Q&A

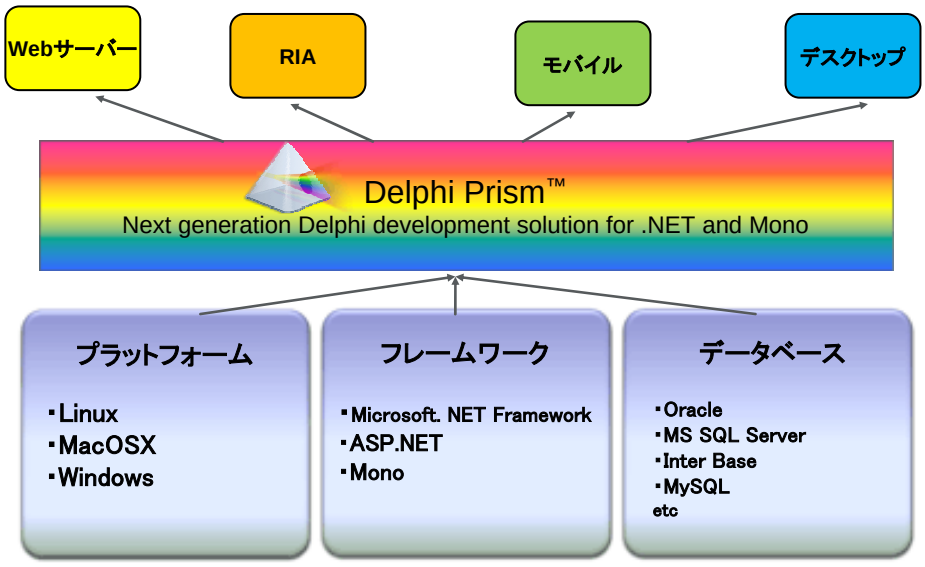
EMBARCADERO TECHNOLOGIES

DEVELOPER CAMP

製品概要

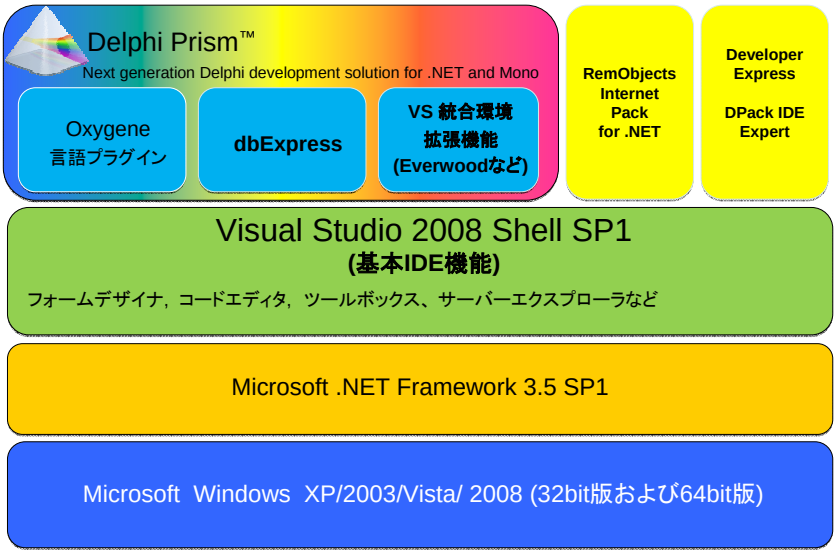
Delphi Prismの製品コンセプト

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP



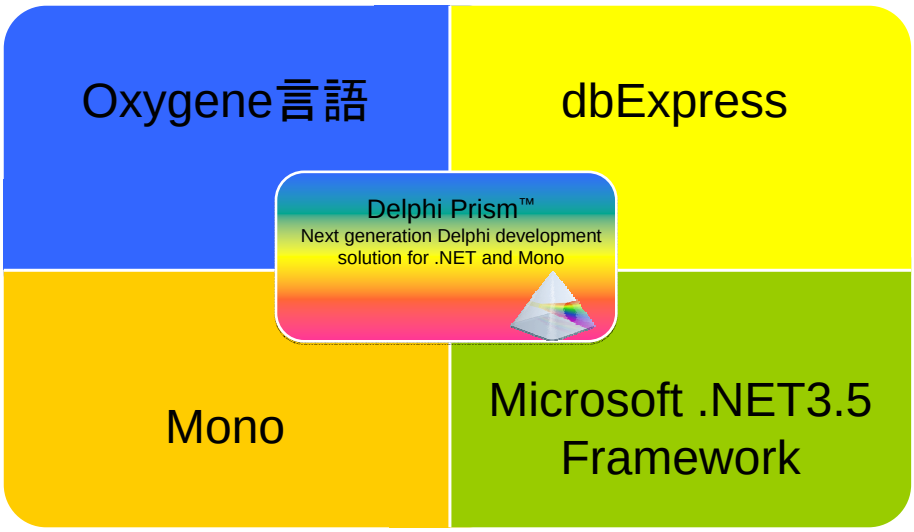
Delphi Prismのインストール環境

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP



Delphi Prismの機能構成

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP



Delphi Prismのエディション

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

Delphi Prism言語

Delphi Prism言語

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- Oxygene言語
- 既存のDelphiとの互換性
- 互換性に関するオプション
- Oxygene言語機能の紹介

Oxygene言語

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- RemObjects Softwareによって開発された言語
  (かつては、Chromeと呼ばれていた言語)
- Oxygeneは、Object Pascal言語をベースとし、
  最新.NETが要求する全ての言語イディオムをサポート
  -ジェネリクス, LINQ, ラムダ式, 無名メソッド, デリゲート
  など
- Oxygeneコンパイラは、.NET1.1 , 2.0 , 3.0, 3.5 に対応
- Oxygeneは、既存の.NET言語で行える事は、ほぼ全て可能。
  さらにこれらの言語より、簡単にコードが記述できる言語機能も
  搭載されている

既存のDelphi との互換性

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- VCL.NETについて

  →フレームワークとして互換性が無いため、
  GUI系のコントロールはDelphi Prismへ移行はできない

  →Delphi Prismでは今後、VCL.NETはサポートされない

  Winforms, WPFといった.NET標準のフレームワークへご移行ください

- 既存のDelphi コードをDelphi Prismへ移行させるには?

  →言語仕様は類似していますが、現行では移行するにはコードの修正が必要

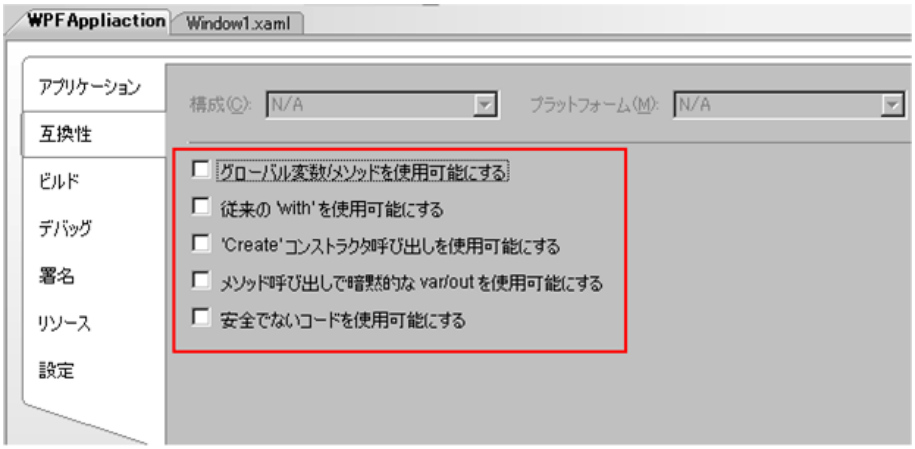
  Delphi Prismのプロジェクトオプション、移行ツール(ShineOn プロジェクト, Oxidizer コードコンバータなど) をご利用ください。

将来的に、Delphi for win32 , Delphi Prism双方の互換性が、より保てるような言語仕様の拡張が行われていく予定です。

互換性に関するオプション

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- Delphi Prismでは、既存のDelphi言語に対するいくつかの言語機能に関して互換性を保つことができるオプションが用意されています。



Oxygene言語機能の紹介

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- ジェネリクス
- 統合言語クエリ(LINQ)
- デリゲート
- シンタックスシュガー
- 拡張されたNullable型
- マルチスレッド

ジェネリクス

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- ジェネリクスというテンプレートを利用することで、特定の型に縛られること無くコードを汎用化、そしてコードを記述する量も減らすことができ、効率化が計れます。

```
type
   MyList<T> = class(System.Object, system.Collections.Generics.IEnumerable<T>)
        where T is class;
   private
     fData: array of  T;
     fCount: Integer;
   public
          ...
```

このクラスを利用するには、型宣言においてジェネリックな型を定義する必要があります

```
var
Data: MyList<String>;
begin
   Data := new MyList<String>;
   Data.Add('Test');
          ...
end;
```

統合言語クエリ(LINQ)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- 統合言語クエリ(LINQ)は、イテレート可能なデータ構造に対してSQLライクなクエリで 検索、操作が行えるプログラミング言語に統合された構文

  Microsoft .NET 3.5 Framework上で利用可能　(Monoでもサポート)

- LINQは様々なデータに対して適用可能

  -配列やコレクションなどのオブジェクト, XML ,リレーショナルデータベースなど

- 利用可能なクエリ構文

  -where, order by, select, from, join, group by, take, skip, reverse, distinct など

使用例:

```
type Customer = public class
...
var  MyCustomers: sequence of Customers …

var  x :=  from  c in  MyCustomers  where  c.City  =  'London'  order by  c.Name select  c.Name;
for each cust  in  x  do
  Console.WriteLine(cust.Name);
```


デリゲート

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- デリゲート(委譲)とは、メソッドを参照するためのオブジェクト
- 同じシグネチャと戻り値で、順序付けられたメソッドのリストを持ち、
  複数のメソッドを同時に参照する事ができる
- イベントハンドラがデリゲート型で宣言されていることから、他のメソッドに処理を委譲させる目的で使用されます

```
type
  MyDelegate = delegate (data:string);

method  MyClass.func(param:String);
begin
 Console.WriteLine('Thread_ID:{0} func: {1}',System.Threading.Thread.CurrentThread.ManagedThreadId, param);
end;
.....
method  MyClass.Proc
var
   del:MyDelegate;
begin
    Console.WriteLine('Thread_ID:{0} MainProc',System.Threading.Thread.CurrentThread.ManagedThreadId);
    del:=new MyDelegate(func);
    del('Hello World');
end;
```

メソッドの呼び出しは、同期的に実行される

シンタックスシュガー

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- Oxygene言語では、C#よりもシンプルなコードが記述できる言語機能が用意されています。 このシンタックスシュガーによって、コーディングを簡略化することができます。

・プロパティ・アクセッサのインライン化
・クラスコントラクト
・プロパティ通知
・フューチャー
・非同期メソッド(async)
など

プロパティ・アクセッサのインライン化

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

Delphi Prism

```
type
    MyClass = public class
 public
   property Data:Int32;
   property Twice: Int32 read Data*2;
end;
```

C# 3.0

```
public class   MyClass
{
     public int Data { get; set; }
     public int Twice {
      get {
         return (this.Data * 2);
       }
      }
}
```

クラスコントラクト(クラス不変条件)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

Delphi Prism

```
type
  TClassWithInvariant = class
  public
    SomeInteger: Integer;
  public invariants
    SomeInteger < 10;
  public
    procedure SetSomeIntegerGreaterThanTen;
  end;

procedure TClassWithInvariant.SetSomeIntegerGreaterThanTen;
begin
  SomeInteger:=10;
end;
```

C#

```
using System.Diagnostics;

class TClassWithInvariant  {
      public int SomeInteger;

       public void SetSomeIntegerGreaterThanTen()
       {
          this.SomeInteger = 10;
            lock (this)
            {
               try {
                Debug.Assert(this.SomeInteger < 10, "public invariant
      (SomeInteger < 10)");
               }finally{
               ...
               }
            }
       }
}
```


クラスコントラクト(Require/Ensure)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

Delphi Prism

```
type
  MyClass = class
  private
    Mylist:List<String>:=new List<string>;
    Count:Integer;
  public
    method Add(Data: String);
  end;

method MyClass.Add(Data: String);
require
   Data.Length <> 0: '文字列が空です';
begin
   MyList.Add(Data);
   Inc(Count);
ensure
   Count = old Count+1;
end;
```

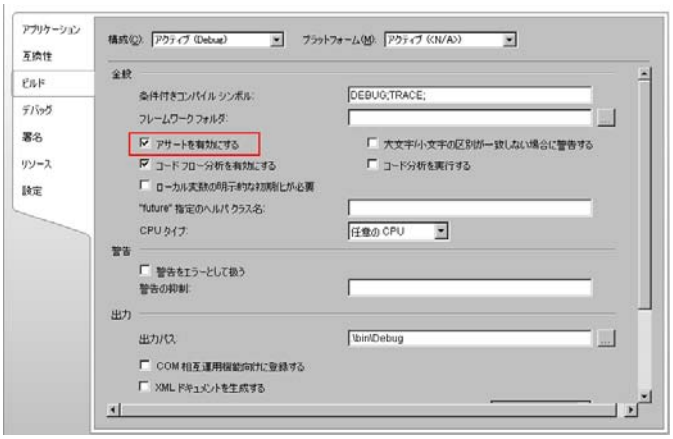
C#

```
class MyClass
{
    private int Count;
    private List<string> Mylist = new List<string>();

    public void Add(string Data)
    {
//require
        Debug.Assert(Data.Length != 0, "WindowsApplication12.MyClass.Add(Data:
        System.String) precondition 文字列が空です");
        int count = this.Count;
        this.Mylist.Add(Data);
        this.Count++;
//ensure
        Debug.Assert(this.Count == (count + 1), "WindowsApplication12.MyClass.Add(Data:
        System.String) postcondition (Count = (old_Count + 1))");
    }
}
```

アサーションを有効にする

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

Delphi Prismでは、アサートを有効にするための専用のオプションが用意されており、
条件の中に"DEBUG"を含める必要はありません。



このオプションのチェック有無によってコンパイラは、クラスコントラクトで定義されている
アサートコードをMSILへ埋め込むか判断しています。

プロパティ通知

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

Delphi Prism

```
type
  ValueClass = class
    public
      property Name: string; notify 'ValueName';
end;
```

C#

```
using System.ComponentModel;

public class ValueClass: INotifyPropertyChanged,INotifyPropertyChanging
{
      public event PropertyChangedEventHandler PropertyChanged;
      public event PropertyChangingEventHandler PropertyChanging;

      public void NotifyPropertyChanged(String PropertyName)
      {
         if (PropertyChanged != null) {
            PropertyChanged(this, new PropertyChangedEventArgs(PropertyName));
         }
      }
      public void NotifyPropertyChanging(String PropertyName)
      {
           ....
      }
      public string Name
      {
           ....
          set{
            if (value != _name)
            {
               NotifyPropertyChanging("ValueName");
               _name = value;
               NotifyPropertyChanged("ValueName");
            }
       }
}
```

Nullable型

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- Nullable 型とは?
  - ・値型は、通常 null値（無効な値）を取れない
  - ・値型の初期値が有効な値なのか、無効な値なのか区別がつかない

この問題を解決するために・・・

.NET 2.0 Frameworkで Nullable型という特殊な型が用意されました

- Nullable 型

-System.Nullable<T>というジェネリクス構造体のことを指します

-型パラメータTには、値型のみ指定できる

-このTに対して指定された値型データとnull値の両方を扱うことができる

Delphi Prism

```
var iData: nullable Integer:=100;
var dData: nullable Double;
iData:=nil;
dData:=1234.56;
```

C#

```
int? iData=100;
Nullable<double> dData;
iData=null;
dData=1234.56;
```


拡張されたNullable型とは?

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- 拡張されたNullable型では、算術式中にnullableな型が含まれている場合式全体がnullableとして扱われます。

C#

```
int? x;
int  y;
var a := x+y;
var b := 5*x;
```

C#では、算術式中にnullableな型を含むため上記のコードはコンパイルできません。

Delphi Prism

```
var x: nullable Int32;
var y: Integer;
var a := x+y;
var b := 5*x;
```

Delphi Prismでは、上記のコードはコンパイル可能です。
これは算術式中にnullableな型が含まれていても、式全体がnullableと解釈されるため、
ローカル変数 aもbも nullable なInt32として扱われます。

Delphi Prismでは、上記のようなNullable式の拡張サポートが施されています。

Quiz

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

ここで、Nullableに関するクイズです!!

以下のDelphi Prismのコードを実行した時、答えは(1)～(3)のうち、どれでしょうか。

```
class method ConsoleApp.Main;
begin
  var x: nullable Int32;
  var y: Integer:=100;

  Console.WriteLine('x+y='+(x+y));

  Console.ReadLine();

end;
```

(1)実行時エラーが発生する

(2)コンソール画面に “x+y=“ と表示される

(3)何も表示されない

.NETのマルチスレッド(1) -非同期処理-

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- **BeginInvokeメソッド / EndInvokeメソッド**

-デリゲートを使用し、メソッドを非同期に実行するための仕組みで、BeginInvokeメソッドを呼び出した時に返されるIAsyncResultのIsCompletedプロパティを参照することで、非同期処理が完了したか判断することができます

- **AsyncCallbackデリゲート**

-非同期処理の完了を待たずに主スレッドの処理を継続させたいケースでは、AsyncCallbackデリゲートさせたコールバック・メソッドを定義しておくことで、完了時にメソッドが呼び出され、通知を受けることができます

```
type
  MyDelegate = delegate (data: string);

method MyClass.func(param: string);
begin
    //非同期で実行される処理
end;

method MyClass.MyCallback(iar: IAsyncResult);   //完了後、コールバックされる処理
begin
    var ar:  System.Runtime.Remoting.Messaging.AsyncResult   :=   System.Runtime.Remoting.Messaging.AsyncResult(iar);
    var del: MyDelegate  := MyDelegate( ar.AsyncDelegate );
    del.EndInvoke( iar );      //コールバックメソッド内で、EndInvokeを呼び出す
end;

method  MyClass.Proc;
var
  del:MyDelegate;
  iar:IAsyncResult;
begin
  del:=new MyDelegate(func);
  iar := del.BeginInvoke('HelloWorld',new AsyncCallback(MyCallback),nil);
  …
end;
```

.NETのマルチスレッド(2) -バックグラウンド処理-

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- **ThreadPoolクラスによるマルチスレッド**

-.このクラスを利用することで、別スレッドで実行させたいメソッドをWaitCallbackにデリゲートさせ、ThreadPool.QueueUserWorkItemメソッドに渡すことで、定義したメソッドをバックグラウンドで処理させることができます

- **スレッドプールのスレッド数**

-スレッドプールには 1 CPUプロセッサごとに最大25個のワーカースレッドを待機させる事ができ、この最大許容範囲内でスレッドが自動的に追加、そして再利用されます

```
uses
  System.Threading.*;

type
  ConsoleApp = class
  private
    class method MyThreadProc(data:Object);
  public
    class method Main;
end;

class method ConsoleApp.Main;
var
  waitCallback:WaitCallback;
begin
  waitCallback:=new WaitCallback(MyThreadProc);
  ThreadPool.QueueUserWorkItem(waitCallback,'Sample');
  .....
end;

class method ConsoleApp.MyThreadProc(data:Object);
begin
    //実際の処理
end;
```


非同期メソッド (async)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

Delphi Prismでは、予約語として asyncを定義することで、非同期処理がより簡単に記述できます

一般的には、asyncブロックは以下のように修飾します:

```
async begin
...
end;
```

また asyncはディレクティブで指定することもでき、さきほどの.NETのマルチスレッド(2)のプログラムをasyncで以下のようなコードに置き換えることもできます

```
uses
  System.Threading.*;

type
  ConsoleApp = class
  private
    class method MyThreadProc(data:Object);  async;
  public
    class method Main;
end;

class method ConsoleApp.Main;
begin
  MyThreadProc('Sample');
  ...
end;

class method ConsoleApp.MyThreadProc(data:Object);
begin
    //実行の処理
end;
```

並列ループ (Parallel Loops)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- Task Parallel Library (TPL)

Parallel FX Libraryの主要コンポーネントであり、マイクロソフト社が提供する
.NET Framework上の言語から利用できる並列実行の向上を目的としたライブラリです。
TPLは、.NETの次期バージョンである「.NET Framework 4.0」のコアライブラリとして
組み込まれる予定です。
並列ループを利用するためには、TPLが必要です。

Delphi Prismでは、このTPLに言語レベルで早期に対応!!

Delphi Prism

```
for parallel  i: Integer := 0  to  10  do
  begin
    // 実際の処理
  end;

for parallel  i: Integer := 1  to  49  step  7 do
  begin
   // 実際の処理
  end;

var  SomeCollection: sequence of  String;

for each parallel elem in SomeCollection do
  begin
    // 実際の処理
  end;
```

C#

```
Parallel.For(0, 11, 1, delegate (int i,  ParallelState state) {
        // 実際の処理
  });

Parallel.For(1, 50, 7, delegate (int i,  ParallelState state) {
       // 実際の処理
 });

Parallel.ForEach<string>(SomeCollection, delegate (string elem, int index,
ParallelState state) {
         // 実際の処理
 });
```


並列ループ (Parallel Loops)　ちょっと実験

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

規定回数ループさせる単純なプログラムでparallelの有無により、どの程度実行速度が異なるか確認してみました

通常ループ(シングル・スレッド)

```
method ConsoleApp.Main;
begin
    var sw:=new Stopwatch;
    sw.Start;
    for  i: integer := 0 to 9 do
         ConsoleApp.Test;
    sw.Stop;
    Console.WriteLine('経過時間={0} ms', sw.ElapsedMilliseconds);
      ....
end;

method ConsoleApp.Test;
begin
     //重い処理
....
  Console.WriteLine('Thread:{0} is done.',
Thread.CurrentThread.ManagedThreadId);
end;
```

```
Thread:1 is done.
Thread:1 is done.
Thread:1 is done.
All 10 loop have been spawned!
経過時間=110407 ms
```

並列ループ(マルチ・スレッド)

```
method ConsoleApp.Main;
begin
    var sw:=new Stopwatch;
    sw.Start;
    for  parallel i: integer := 0 to 9 do
         ConsoleApp.Test;
    sw.Stop;
    Console.WriteLine('経過時間={0} ms', sw.ElapsedMilliseconds);
      ....
end;

method ConsoleApp.Test;
begin
     //重い処理
....
  Console.WriteLine('Thread:{0} is done.',
Thread.CurrentThread.ManagedThreadId);
end;
```

```
Thread:3 is done.
Thread:5 is done.
Thread:7 is done.
All 10 loop have been spawned!
経過時間=33220 ms
```

フューチャー(Futures)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

futureは強く型付けされた変数で、計算が完了しているかもしれない、もしくは完了してないかもしれない変数を表しますが、必要となる時点で利用可能であることが保証されています

```
method  MyClass.Sum:Int32;
begin
 ....
  var i: future Int32:= async MyThreadProc1;
  var j: future Int32:= async MyThreadProc2;
  var k:Int32:=500;
 ....
  Result:=i+j+k;    → (1)
end;
```

上記の断片コードをご覧ください

実際にはMyThreadProc1, MyThreadProc2は非同期で実行されています。そして最終的に計算結果が揃うのは(1)の時点なため、もしこの段階で、非同期メソッドが値を返す値、つまりfuture変数が取得する値の計算が完了していなければ、ここで同期が取られます。

```
method MyClass.MyThreadProc1:Int32;
var
  Count:Integer;
begin
  for i:Integer:=0 to 20 do
  begin
     ...
     Count:=Count+i;
  end;
   Result:=Count;
end;

method MyClass.MyThreadProc2:Int32;
var
  Count:Integer;
begin
  for i:Integer:=0 to 50 do
  begin
     ...
     Count:=Count+i;
  end;
  Result:=Count;
end;
```

Delphi Prismでは、async、futureを記述するだけで、内部的にFutureHelperクラスのインスタンスが作成され、このオブジェクトが非同期処理のデリゲートと結果取得の調停を担います。このヘルパークラスの働きによって、複雑になりがちなコードもシンプルに書くことができます

FutureHelperクラス

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

FutureHelperクラスの抜粋

```
FutureHelper nested in (PrivateImplementationDetails) = assembly sealed class

  [ Fields ]
  private fAsyncResult: IAsyncResult;
  private fMethod: Action;

  [ Methods ]
  private constructor(aMethod: Action; aAsync: Int32);
  private method AsyncStart(SkipResult: Boolean);
  public class method Execute<T>(aFuture: Func<T>; modopt(future$1): Func<T>; modop
  public class method Execute(aMethod: Action);
  public class method ExecuteAsync<T>(aFuture: Func<T>; modopt(future$1), aWantRe
  public class method ExecuteAsync(aMethod: Action; aWantResult: Boolean): Action;
  private method GetValue;
  public class method IsDone<T>(aFuture: Func<T>; modopt(future$1)): Boolean;
  public class method IsDone(aFuture: Action): Boolean;
  public method ThreadPoolInvoke(dummy: Object);
end;
```

```
method (PrivateImplementationDetails).FutureHelper.AsyncStart(SkipResult: Boolean);
begin
   if SkipResult then
      ThreadPool.QueueUserWorkItem(new WaitCallback(self.ThreadPoolInvoke))
   else
      self.fAsyncResult := self.fMethod.BeginInvoke(nil, nil)
end;
```

```
method (PrivateImplementationDetails).FutureHelper.GetValue;
begin
   lock.ing self do begin if (self.fMethod <> nil) then begin
            if (self.fAsyncResult <> nil) then
               self.fMethod.EndInvoke(self.fAsyncResult)
            else
               self.fMethod.Invoke;
            self.fMethod := nil
         end
      end
end;
```

```
class method (PrivateImplementationDetails).FutureHelper.ExecuteAsync(aMethod: Action; aWantResult: Boolean): Action;
begin
   var action: Action;
   var helper: FutureHelper := new FutureHelper(aMethod, iif(not aWantResult, 2, 1));
   if aWantResult then
      action := new ActionHelper.GetValue);
   begin
      result := action;
      exit
   end
end;
```

```
constructor FutureHelper(aMethod: Action; aAsync: Int32);
begin
   inherited constructor;
   self.fMethod := aMethod;
   if (aAsync <> 0) then
      self.AsyncStart((aAsync = 2))
end;
```

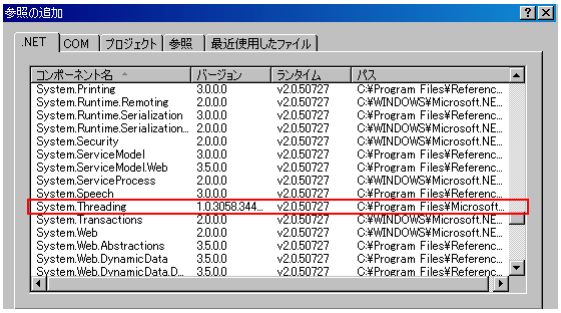
```
class method (PrivateImplementationDetails).FutureHelper.IsDone(aFuture: Action): Boolean;
begin
   var target: FutureHelper := FutureHelper(aFuture.Target);
   if (target = nil) then
      raise new ArgumentException;
   begin
      result := iif((target.fMethod <> nil), iif((target.fAsyncResult <> nil), target.fAsyncResult.IsCompleted, true), true);
      exit
   end
end;
```

.NET 3.5とTPL

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- .NET Framework 3.5上でTask Parallel Library(TPL)の利用

別途、.NET3.5用コミュニティ向けテクノロジープレビュー版(CTP)の入手が必要です。
「Microsoft Parallel Extensions to .NET Framework 3.5, June 2008 Community Technology Preview」
http://www.microsoft.com/downloads/details.aspx?FamilyId=348F73FD-593D-4B3C-B055-694C50D2B0F3&displaylang=en
※今後、新しいCTP版がリリースされる可能性があります。
上記のインストール完了後、GACへSystem.Threadingアセンブリがコピーされるので、
TPLの利用時には、必ずプロジェクトでこのアセンブリの参照を追加してください。



EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

Mono

Monoとは?

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- マイクロソフトの.NET Framework互換の開発フレームワーク/ランタイムを提供するオープンソースのソフトウェア
- 選択できるライセンスは、GPL, LGPL, MITなど
- Monoは、クロスプラットホームで動作可能

- Linux, Mac OSX ,Solaris, Windowsなど

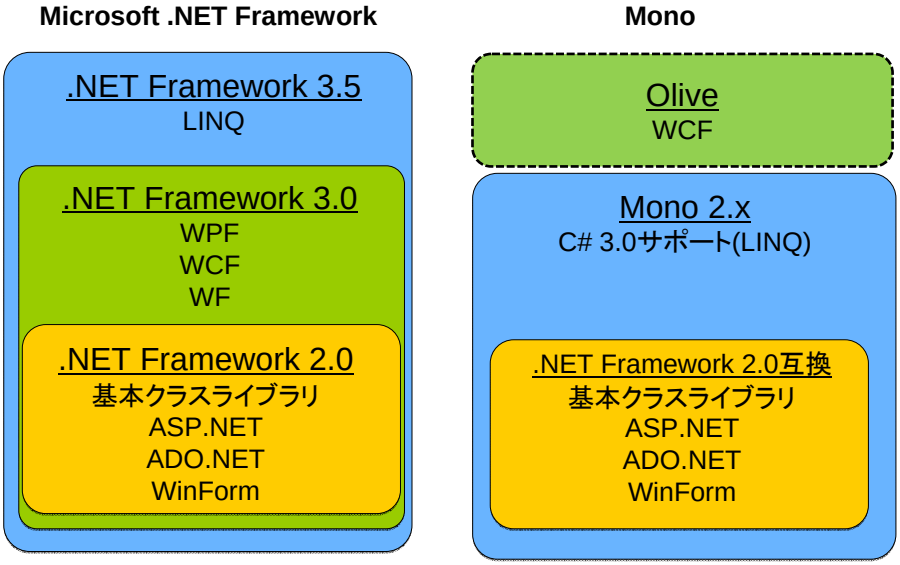
- Mono JITコンパイラが対応しているプロセッサ

-x86,Amd64, IA-64, SPARC, PowerPCなど

- Monoは、2009月2月現在で、最新バージョンは2.2

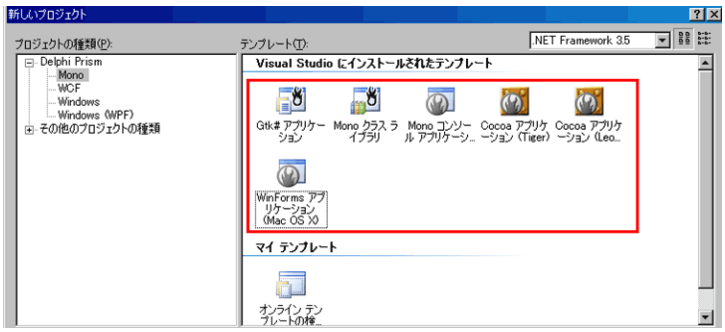
Monoと.NET Framework

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP



Delphi Prismで作成可能なMonoプロジェクト

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP



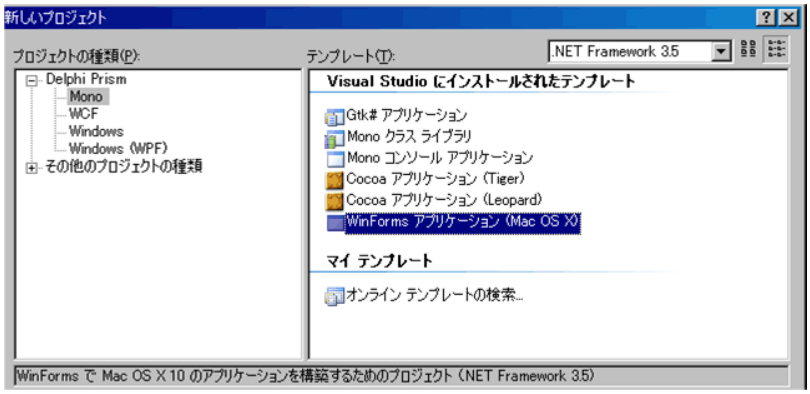
Delphi Prismでは、以下のプロジェクトテンプレートが用意されています

・Gtk# アプリケーション
・Monoコンソールアプリケーション
・Cocoaアプリケーション
・MacOSX 用 WinFormsアプリケーション

MacOSX向けのWinFormsの作成 – ステップ1

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

(1) [ファイル]-[新規作成]-[プロジェクト]から”WinFormsアプリケーション(MacOSX)”を選択

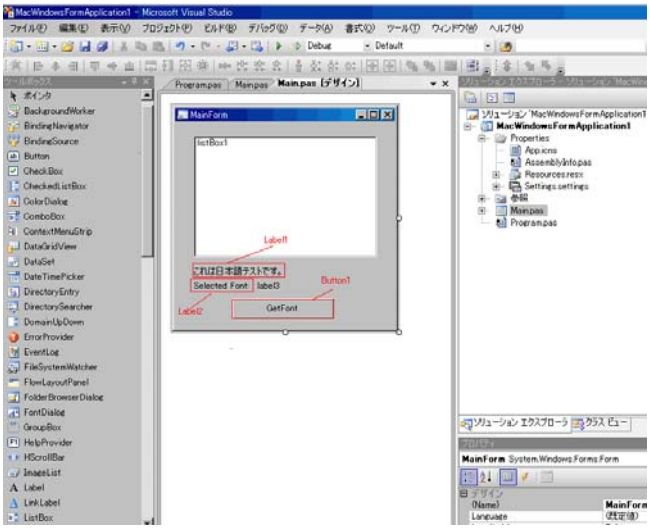


MacOSX用のWinFormsプロジェクトを選択することで、
ビルド時にInfo.plistファイルも含めたMacOSXへ配布可能なモジュール一式が自動生成されます

MacOSX向けのWinFormsの作成 – ステップ2

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

(2) フォーム上に各コントロール Label 3個 , Button 1個, ListBox 1個　を配置



MacOSX向けのWinFormsの作成 – ステップ3

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

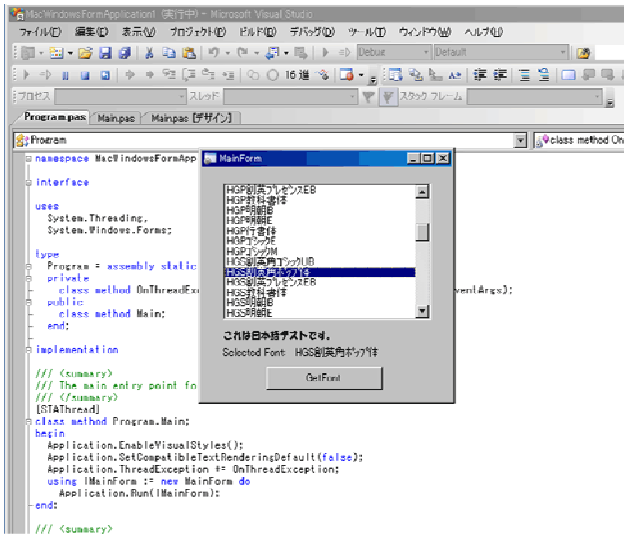
(3) Button1 OnClickイベントでOSにインストールされているフォントの一覧を取得、
ListBox1 onDoubleClickイベントでLabel1を選択されたフォントに変更するコードを記述します。

```
method MainForm.button1_Click(sender: System.Object; e: System.EventArgs);
begin
  var installFonts: System.Drawing.Text.InstalledFontCollection:=  new  System.Drawing.Text.InstalledFontCollection();
  listBox1.Items.Clear;

  for each font:System.Drawing.FontFamily in installFonts.Families do
  begin
    listBox1.Items.Add(font.Name);
  end;
end;

method MainForm.listBox1_DoubleClick(sender: System.Object; e: System.EventArgs);
begin
  Label1.Font:=new Font(listBox1.SelectedItem.ToString,8);
  Label3.Text:=listBox1.SelectedItem.ToString;
end;
```

MacOSX向けのWinFormsの作成 – ステップ4

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

(4) プロジェクトをビルドし、Windows上で実行を確認

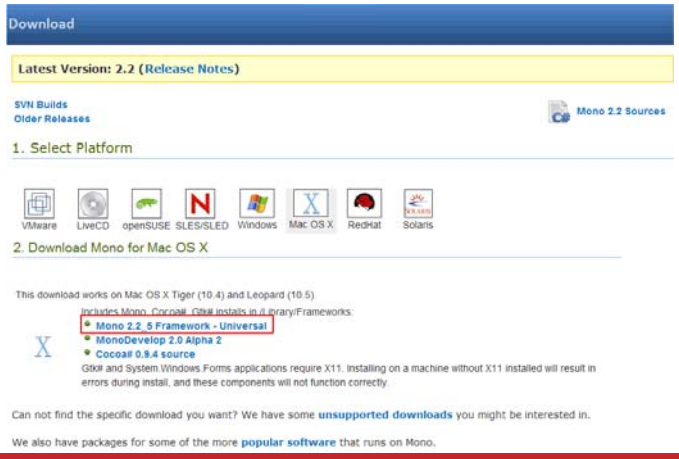


MacOSX向けのWinFormsの作成 – ステップ5

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

(5)MacOSX用のMonoをダウンロードする

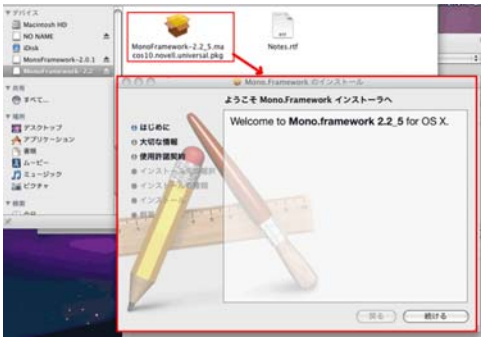
Monoの最新版(2.2)は、以下のURLから入手できます。
http://www.go-mono.com/mono-downloads/download.html



MacOSX向けのWinFormsの作成 – ステップ6

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

(6)MacOSXへMonoをインストールする



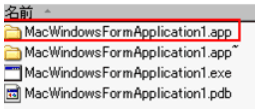
インストール後、
Monoのバージョンの確認

```
Terminal — bash — 80×35
Last login: Wed Feb  4 11:17:37 on ttys000
Support-Macmini:~ support$ mono --version
Mono JIT compiler version 2.2 (tarball Fri Jan  9 10:45:07 MST 2009)
Copyright (C) 2002-2008 Novell, Inc and Contributors. www.mono-project.com
        TLS:           normal
        GC:            Included Boehm (with typed GC)
        SIGSEGV:       normal
        Notification:  Thread + polling
        Architecture:  x86
        Disabled:      none_
```


MacOSX向けのWinFormsの作成 – ステップ7

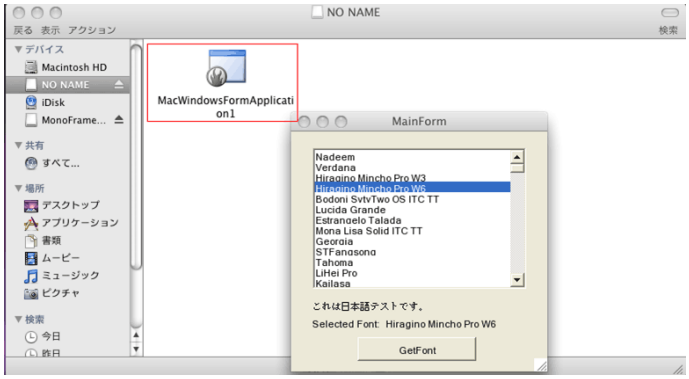
EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

(7) Windows環境でビルドしたアセンブリをMacOSXへデプロイする



MacWindowsFormApplication1.appフォルダの中には、
Info.plistファイルといったMacOSX上で動作させるために
必要な環境ファイルが含まれています。

デプロイ



MacOSX上での注意点

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

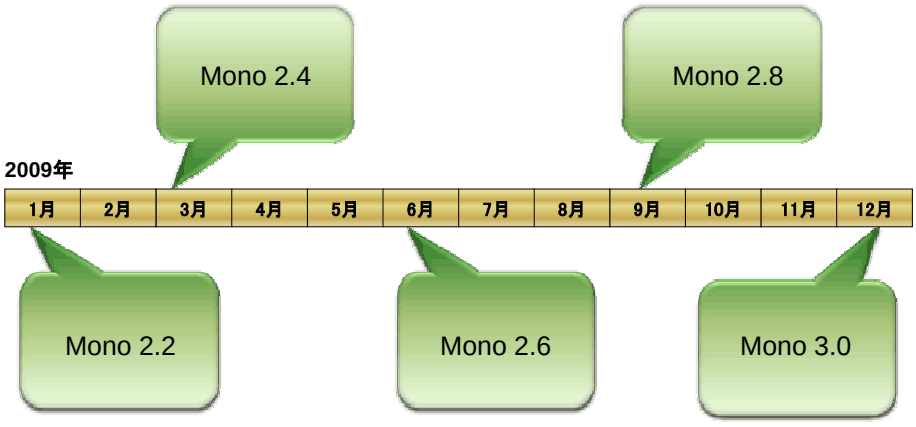
- MacOSXへデプロイしたMonoアプリケーションを実行する場合，
適切なフォントが選択されていないと、文字化けが発生いたします。
-日本語が表示可能なフォントは、ヒラギノ W3/W6，Osakaなど

- 上記はDelphi Prism の問題ではなく、Monoランタイムの仕様と考えます。
アプリケーション側で日本語が表示可能なフォントを明示的にセットしてください。

- 内部的にP/Invokeで呼び出しを行っているライブラリは、
Windowsプラットホームに依存してしまうため、そのライブラリを含む
アセンブリを配布してもMono上では動作しない。

Monoプロジェクトの今後のリリースについて

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP



- Mono Project Roadmap
http://www.mono-project.com/Roadmap

EMBARCADERO TECHNOLOGIES

DEVELOPER CAMP

dbExpress

dbExpressとは?

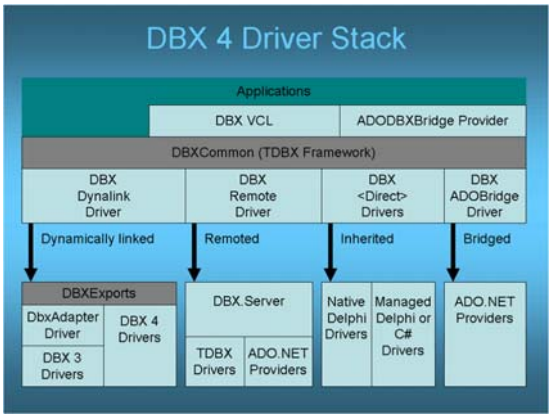
EMBARCADERO TECHNOLOGIES

DEVELOPER CAMP

Delphi Prismは、データベース開発における強力なフレームワークである「dbExpress 4」を搭載しています。

dbExpressを利用することにより、.NET開発者はバックエンドの特定のデータベースに縛られることなく、単一のフレームワークで複数のデータベースにアクセスすることが可能で、**データベースドライバ構築のプロセスを大幅に単純化**することができます。

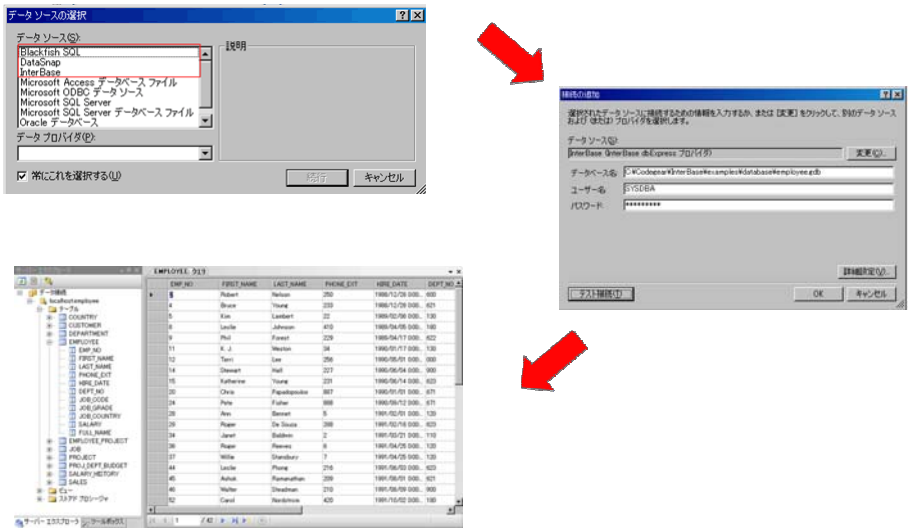
またdbExpressは、Delphi for Win32でも採用されており、**ネイティブとマネージドコードのクロスプラットフォームに対応**しています。Win32環境と同じエンジンを採用しているため、データベース接続に対する高いポータビリティにより、.NET環境へ移行も容易です。



現在サポートされているデータベースは、
-InterBase
-BlackfishSQL

サーバーエクスプローラとの統合

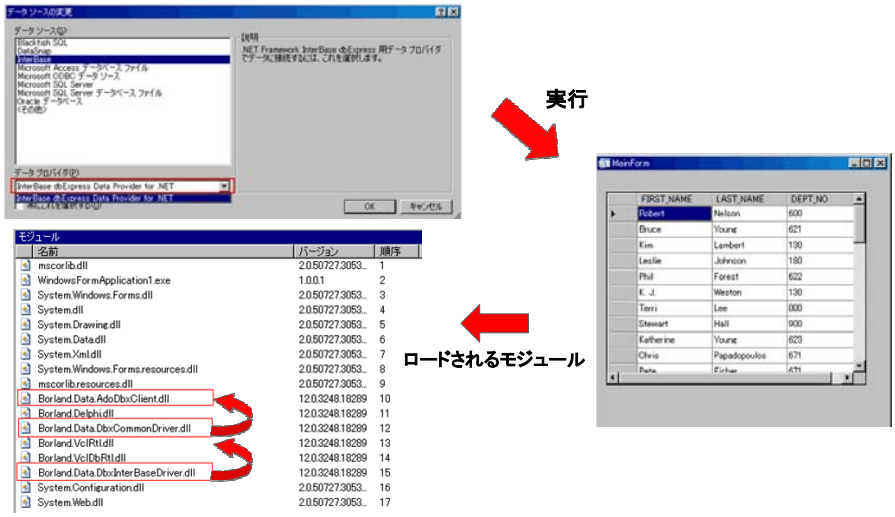
EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP



dbExpressモジュールのロード

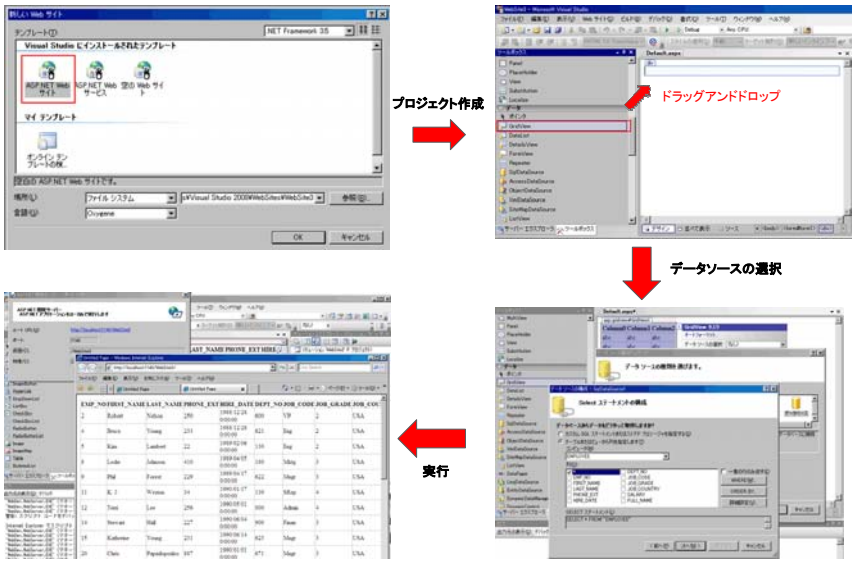
EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

データベースとしてInterBaseへ接続した場合:



ASP.NETアプリケーション作成 例

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP



Delphi Prism製品のロードマップ(予定)

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

Delphi Prism™

Next generation Delphi development solution for .NET and Mono

- .NET4.0とVisual Studio 2010対応に向けた言語仕様の拡張
- Delphi for Win32とDelphi Prismの言語互換性の向上
- dbExpressドライバのアップデート
  -Oracle, DB2 , Sybase, SQL Anywhere, MySQL, Informixのサポート
  -ADO.NET エンティティフレームワークに対応
- Mono開発環境の向上(Cocoa#, MonObjC, Gtk#)
- DataSnapサーバーの開発機能
- CruiseControl.NET，NUnit, ソースコード管理機能の統合
- 新たなプロジェクトテンプレートに対応
  -ASP.NET MVC テンプレート，BlackfishSQLストアドプロシージャテンプレート

製品をご評価ください!!

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

Delphi Prism™

Next generation Delphi development solution for .NET and Mono

- **最新の.NET環境をフルサポート**
- **強力なフル機能のDelphi Prism開発言語**
- **データベースアプリケーション構築のためのdbExpressフレームワーク**
- **Monoプラットフォーム向けの開発サポート**

エンバカデロ・テクノロジーズが提供する
新しい.NET開発ソリューション環境
「**Delphi Prism**」にどうかご期待ください!!

製品のご評価は以下のURLまで
http://cc.codegear.com/free/prism

参考資料

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- Delphi Prism Wiki
http://prismwiki.codegear.com/en/Main_Page

- Delphi Prism 機能評価ガイド
http://dn.codegear.com/article/39118

- Delphi Prismロードマップ
http://dn.codegear.com/article/39276/

- Migrating a Project to Delphi Prism from Delphi.NET
http://jamie.op-i.net/blog/2008/12/migrating-a-project-to-delphi-prism-from-delphinet/

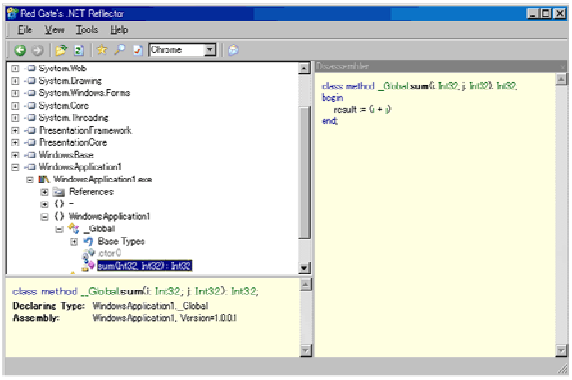
- マルチコア コンピュータ用にマネージ コードを最適化する
http://msdn.microsoft.com/ja-jp/magazine/cc163340.aspx

.NET Reflector™

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

- .NETアセンブリをC#やVB.NETのソース・コードへ変換する
  逆コンパイラツール(フリーウェア)　　Oxygene(Chrome)言語にも対応

  http://www.red-gate.com/products/reflector/



Q&A

EMBARCADERO TECHNOLOGIES
DEVELOPER CAMP

ご清聴ありがとうございました

Q&A