Advanced Language Features of Delphi

David Intersimone “David I”
Vice President, Developer Relations and Chief Evangelist

EMBARCADERO
TECHNOLOGIES.

Agenda

- Generics – types and methods
- Anonymous Methods
- Inline functions and procedures
- Class declarations
  - Nested classes, Abstract classes, Sealed classes
  - Static class methods, Static properties, and Final methods
  - Scope directives: strict private and strict protected
  - Class constants, Class helpers
- Enhanced RTTI, Custom attributes
- Class Constructors and Class Destructors
- Exit procedure - takes optional Result parameter
- for-in loop
- Delayed – delay loading DLLs
- Operator overloading
- Unicode strings

Generics – types and methods

- <T> - type T is provided later

```
TKeyValue<T> = class
private
    FKey: string;
    FValue: T;
    procedure SetKey(const Value: string);
    procedure SetValue(const Value: T);
public
    property Key: string read FKey write SetKey;
    property Value: T read FValue write SetValue;
end;
```

- unit Generics.Collections
  - TList, TObjectList
  - TQueue, TObjectQueue
  - TStack, TObjectStack
  - TDictionary, TObjectDictionary

Anonymous Methods

- A procedure or function that does not have a name associated with it.

```
// declare anonymous method and save it as the reference
proc := procedure(a: Integer) begin ... end;
Call(proc);
// anonymous method declared and used as a parameter
Call(procedure(a: Integer) begin ... end);
```

- Can be assigned to a variable or used as a parameter to a method.
- Can refer to variables and bind values to the variables in the context in which the method is defined.
- You don’t have to create a class to hold the method(s)

Inline functions and procedures

- **Inline** directive

```
function MaxInline(const X,Y,Z: Integer): Integer;inline;
function MaxNotInline(const X,Y,Z: Integer): Integer;
```

- Inline is a **suggestion** to the compiler
- If the function or procedure meets certain criteria, the compiler will insert code directly – see the next two slides for the criteria

Inline Criteria

- Cannot be inlined
  - Late-bound methods (virtual, dynamic, and message).
  - Routines containing assembly code
  - Constructors and destructors
  - Main program block, unit initialization, and unit finalization
  - Routines that are not defined before use
  - Routines that take open array parameters
  - Circularly dependent units, indirect circular dependencies,
  - Routine defined in the interface section that accesses symbols defined in the implementation section
  - Procedures and functions used in conditional expressions in While-Do and Repeat-Until

Inline Criteria

- Might be inlined
  - The compiler can inline code when a unit is in a circular dependency, as long as the code to be inlined comes from a unit outside the circular relationship.
  - If a routine marked with inline uses external symbols from other units, all of those units must be listed in the uses statement, otherwise the routine cannot be inlined.
  - Within a unit, the body for an inline function should be defined before calls to the function are made. Otherwise, the body of the function, which is not known to the compiler when it reaches the call site, cannot be expanded inline.

Strict Private and Strict Protected

- **Strict private**: creates a true private field, not viewable by any other class, not even classes in the same unit.
- **Strict protected**: creates a true protected member, visible only to the declaring class and its descendents.

```
TMyClass = class
  private
    FPrivate : integer;
  strict private
    FStrictPrivate : integer;
  protected
    FProtected : integer;
  strict protected
    FStrictProtected : integer;
  public
    FPublic : integer;
    procedure CallMe;
  end;
```

Nested Classes

```
type
  TOuterClass = class
  strict private MyField: Integer;
  public
     type
      TInnerClass = class
      public
        MyInnerField: Integer;
        procedure InnerProc;
      end;
     procedure OuterProc;
  end;

procedure TOuterClass.TInnerClass.InnerProc;
begin
  ...
end;
```

Abstract and Sealed Classes, Final Methods

```
TAbstractClass = class abstract
Public
  procedure Bar; virtual;
end;

TSealedClass = class sealed(TAbstractClass)
public
  procedure Bar; override;
end;

TFinalMethodClass = class(TAbstractClass)
public
  procedure Bar; override; final;
end;
```

Static Class Methods and Class Properties

```
type TMyClass = class
strict private
  class var FX: Integer;
strict protected
  // Note: accessors for class properties must be declared
static.
  class function GetX: Integer; static;
  class procedure SetX(val: Integer); static;
public
  class property X: Integer read GetX write SetX;
  class procedure StatProc(s: String); static;
end;

TMyClass.X := 17;
TMyClass.StatProc('Hello');
```

Class Constants

- Class constants
  - Constant value associated with the class itself
  - Not associated with an instance of the class

```
TClassWithConstant = class
public
  const SomeConst = 'This is a class constant';
end;
procedure TForm1.FormCreate(Sender: TObject);
begin
  ShowMessage(TClassWithConstant.SomeConst);
end;
```

Class Helpers

- Class helpers
  - Introduces additional method names and properties
  - A way to extend a class without using inheritance
  - When you declare a class helper, you state the helper name, and the name of the class you are going to extend with the helper.
  - You can use the class helper any place where you can legally use the extended class.

```
TMyClassHelper = class helper for TMyClass
  procedure HelloWorld;
  function MyFunc: Integer;
end;
```

Enhanced RTTI

- Roughly similar to .NET and Java reflection
- New unit: RTTI.pas
- TRttiObject class, TRttiContext record
- Run time information includes all types
  - Classes and all other user defined types
  - Core data types predefined by the compiler
  - Published fields as well as public ones
  - Protected and private elements

```
for aType in package.GetTypes
  for aMethod in aType.GetMethods
    if aMethod.HasAttribute then (...)
```

RTTI Descriptors

- RTTI objects live in a graph
  - TRttiMethod has declaring type and parameters
  - TRttiParameter has parameter type and owner
  - Cycles
- Descriptor <=> RTTI object instance
  - Descriptor not bound to a context
  - Migration of instance from one context to another
  - Caching only interesting RTTI instances for later

Method Invocation

- TRttiMethod.Invoke
  - Invoke instance, class and class static methods
- TRttiConstructor.Invoke
  - Dynamically construct instances without needing virtual constructors and metaclasses
  - Required for general custom attribute support

- Delphi 2010 RTTI Contexts: how they work, and a usage note
  - http://blog.barrkel.com/2010/01/delphi-2010-rtti-contexts-how-they-work.html

TValue – a simple top type

- Is a tuple of raw value data and type info
  - Does not support operators, methods, etc.
  - Not a replacement for Variant
- Conversions in:
  - Has implicit conversions where possible
  - Has explicit generic conversion for other types
- Conversions out:
  - Runtime typed with explicit type (generic)
  - Untyped access to underlying bytes

- "*TValue is the type used to marshal values to and from RTTI-based calls to methods, and reads and writes of fields and properties.*"

Custom Attributes

- Same syntax as Delphi for .NET
  - All attributes descend from TCustomAttribute

```
type
[MyAttribute]
TMyClass = class … end;
MyAttribute = class(TCustomAttribute) … end;
```

- Only simple types allowed in constructors
  - Ordinal types – integers, characters, enums
  - Strings
  - Type references
- RTTI objects have GetAttributes method
  - Attributes are owned by the RTTI object, and therefore implicitly part of the context

Class Constructors and Class Destructors

- Class Constructor - code used to initialize the class itself and executed before any initialization code.
- Class Destructor – code executed after any finalization code.
- Code is linked only if the class is used.
- Better encapsulation, better control over smart linking, compatibility with Delphi Prism.

```
MyException = class(Exception)
private
  class constructor Create;
  class destructor Destroy;
end;
```

Exit procedure

- You can provide an optional Result parameter

```
for I := 0 to sl.Count do
 if Pos (IntToStr (n), sl[I]) > 0 then begin
        Result := sl[I];
        Exit;
 end;
end;

for I := 0 to sl.Count do
 if Pos (IntToStr (n), sl[I]) > 0 then
        Exit (sl[I]);
end;
```

For-in loop

- for index **in** variable do

```
var
  IArray1: array[0..9] of Integer = (1,2,3,4,5,6,7,8,9,10);
  I: Integer;
begin
  for I in IArray1 do begin
    ShowMessage(I) ;
  end;
end;

var
  character : 'a' .. 'z';
begin
  for character in [Low(character) .. High(character)] do
begin
   ShowMessage(character) ;
  end;
end;
```

Delayed directive

- Used for delay loading of DLLs

```
procedure APICall(Param1, Param2: Integer); stdcall;
  external 'kernel32.dll' name 'APICall';
procedure APICall(Param1, Param2: Integer); stdcall;
  external 'kernel32.dll' name 'APICall' delayed;
```

- Instructs the linker to generate the external API reference differently in the executable binary
- Rather than generating the import in the normal “Imports” section of the executable’s PE file, it is generated into the “Delayed Imports” section.
- Delphi RTL will take care of the ugly “late-binding” code you usually have to write.
- RTL has a generic function that does the lookup and binds the API for first call. Subsequent calls are as fast as a normal import.

Operator Overloading

- The name of the operator function maps to a symbolic representation in source code (ie: “Add” operator maps to “+”)
- Compiler generates call to the appropriate implementation based on matching the context to the signature of the operator function.
- Overloaded operator names:
  - Conversion: Implicit, Explicit
  - Unary: Negative, Positive, Inc, Dec, LogicalNot, Trunc, Round
  - Set: In
  - Comparison: Equal, NotEqual, GreaterThan, GreaterThanOrEqual, LessThan, LessThanOrEqual
  - Binary: Add, Subtract, Multiple, Divide, IntDivide, Modulus, LeftShift, RightShift,LogicalAnd, LogicalOr, LogicalXor, BitwiseAnd, BitwiseOr, BitwiseXor

Unicode Strings and Chars

- Since Delphi 2009: string = Unicode string, char = Unicode char
- 1,2,4 bytes per character
- AnsiString, AnsiChar, pAnsiString, pAnsiChar
- AnsiString(codepage)
- I/O streaming impacts – LoadFromFile, SaveToFile, …
- TEncoding class
- RTL modified to do a lot of conversion “under the covers”

Q & A

クイズに答えて賞品を当てよう！

問題

「エンバカデロ、2010年はツール○○○○」。
○○○○に入る4文字のカタカナは？

○ ○ ウ ○

素敵な賞品が当たる！応募はこちらから
http://forms.embarcadero.com/forms/dcamp_quiz


サブディスプレイ機能付
デジタルフォトフレーム


マルチタッチ対応
タッチパッド


Delphiコード入り
エンバカデロTシャツ


エンバカデロ ロゴ入り
光る携帯マウス

Marco Cantù氏の
Delphiハンドブック

応募締切は3月12日15時まで！

EMBARCADERO TECHNOLOGIES.

Thank You ☺

davidi@embarcadero.com