|  |
| --- |
| DevJet.net |
| Spring Framework for Delphi |
| Style Guide (Draft) |

|  |
| --- |
| DevJet.net  5/22/2010 |

Contents

1. **Preample**
2. **Project Management**
3. **Files Organization**

|--Bin

|----Debug

|----Release

|--Lib

|----Debug

|----Release

|--Source

|----Base

|----Core

|--Tests

|----Base

|----Core

|--Documents

|--Examples

|--Third Party

|--Code Templates

1. **Revision management**

The following types of files should never be committed to the project svn server.

**\*.bak, \*.dcu, \*.exe, \*.~\*, \*.ddp, \*.dsk, \*.local, \*.identcache, \*.tvsconfig, \_\_history**

1. **Language Style Guide**
2. **License**

This open source project is licensed under the [**Apache License Version 2.0**](http://www.apache.org/licenses/LICENSE-2.0.html). The following copyright declaration must be included in all units.

{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}

{ }

{ Spring Framework for Delphi }

{ }

{ Copyright (C) 2009-2010 DevJet }

**{ }**

{ http://www.DevJet.net }

{ }

{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}

{ }

{ Licensed under the Apache License, Version 2.0 (the "License"); }

{ you may not use this file except in compliance with the License. }

{ You may obtain a copy of the License at }

{ }

{ http://www.apache.org/licenses/LICENSE-2.0 }

{ }

{ Unless required by applicable law or agreed to in writing, software }

{ distributed under the License is distributed on an "AS IS" BASIS, }

{ WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. }

{ See the License for the specific language governing permissions and }

{ limitations under the License. }

{ }

{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}

1. **Naming Conventions**

* Use namespaces to name source files. (e.g. Spring.Collections.pas)
* Structural Types (Classes, Records and Enumerate Types) should prefix with the letter ‘T’. e.g. TSimpleClass, TDriveType
* Interfaces should start with the letter“I”. e.g. IEnumerable，IList<T>
* Exception classes should start with the letter“E”. e.g. EArgumentException, EFileNotFoundException
* Resource strings starts with the letter “S”. e.g. SFileNotFound
* Fields of classes, local variables and parameters should use **cameCase**. e.g. fOperatingSystem, i, value, targetStream
* Properties and Methods of classes should use **PascalCase**. e.g. Name，IsReadOnly，IsValid，IsEmpty，GetNextID
* Consider naming abstract classes suffix with Base, e.g. TCollectionBase，TStreamBase
* Custom Attribute classes should end with the “Attribute”. The prefix “T” should be ignored.

DisplayNameAttribute = class(TCustomAttribute)

private

fName: string;

public

constructor Create(const name: string);

end;

Tip: The suffix “Attribute” can be ignored when applying attributes.

[DisplayName(‘Paul’)]

TSomeClass = class

end;

* All reserved keywords should be lowercase，e.g. **procedure**，**string**，**begin**，**end**.

1. **White Space Usage**

Use two spaces to instead of a tab.

for i := 0 to list.Count - 1 do

begin

if condition then

begin

DoSomething;

end

else

begin

DoSomethingElse;

end;

case driveType of

dtNetwork:

begin

//...

end;

else

begin

//...

end;

end;

end;

1. **Comments**

Apply the xml style comments in the project.

/// <summary>   
  /// Provides access to information on a drive.   
  /// </summary>   
  /// <remarks>   
  /// Use TDriveInfo.GetDrives method to retrieve all drives of the computer.   
  /// Caller must check IsReady property before using TDriveInfo.   
  /// </remarks>   
  TDriveInfo = record   
 //...  
  end;

*/// <summary>*   
  */// Determines whether a specified file exists. An EFileNotFoundException*   
  */// exception will be raised when not found.*   
  */// </summary>*

**procedure** CheckFileExists(**const** fileName: **string**);

1. **Classes**

/// <summary>   
  /// Represents a complex class declaration.   
  /// </summary>   
  TExampleClass = class(TObject)   
  private   
    type   
      TInnerClass = class(TObject)   
      //...   
      end;   
  strict private  
 const   
      fCMinCount: Integer = 10;   
      fCMaxCount: Integer = 20;   
    class var   
 fSharedObject: TObject;  
    class constructor Create;   
    class destructor Destroy;   
  private   
    fName: string;   
    fAge: Integer;  
  protected   
    procedure DoSomething; virtual;   
  public   
    constructor Create(const name: string);   
    destructor Destroy; override;   
    procedure Test;   
    property Name: string read fName;   
  published   
    // published members   
  end;

1. **Interfaces**

/// <summary>   
/// Provides limited LINQ-like enumerable extension methods for IEnumerable<T>.   
/// </summary>   
IEnumerableEx<T> = interface(IEnumerable<T>)   
  {$REGION 'Property Getters & Setters'}   
    function GetCount: Integer;   
    function GetIsEmpty: Boolean;   
  {$ENDREGION}   
  function First: T; overload;   
  function First(const predicate: TPredicate<T>): T; overload;   
  function FirstOrDefault: T; overload;   
  function FirstOrDefault(const predicate: TPredicate<T>): T; overload;   
  function Last: T; overload;   
  function Last(const predicate: TPredicate<T>): T; overload;   
  function LastOrDefault: T; overload;   
  function LastOrDefault(const predicate: TPredicate<T>): T; overload;   
  function Where(const predicate: TPredicate<T>): IEnumerableEx<T>;   
  function Contains(const item: T): Boolean; overload;   
  function Contains(const item: T; const comparer: IEqualityComparer<T>): Boolean; overload;   
  function ToArray: TArray<T>;   
  function ToList: IList<T>;   
  property Count: Integer read GetCount;   
  property IsEmpty: Boolean read GetIsEmpty;   
end;

**Note:** Generic Interfaces must not have a guid identifier.

1. **Statements**
2. **Anonymous Methods**
3. **Unicode**

Unicode support is a must for the Delphi Spring Framework.

**Note：**The following RTL functions do not support Unicode：

|  |  |
| --- | --- |
| SysUtils.UpperCase | They only work on the Standard Ansi characters (‘a’-‘z’, ‘A’-‘Z’). Use **Character.TCharacter**.ToUpper (or ToLower)instead. |
| SysUtils.LowerCase |
| SysUtils.BytesOf | Use WideBytesOf function instead. |
| SysUtils.StringOf | Use WideStringOf function instead. |

1. **Thread Safety**

It is recommended that global routines and static class methods are ensured to be thread-safety.

1. **Exceptions**

All arguments of global routines and public instance or class methods should be checked.

Note. Use the TArgument class in the Spring namespace to validate arguments.

1. **Additions**

* Use Exception.CreateResXXX overloads to raise an exception when using resource strings.
* Whenever possible, defines parameter as const, var or out.
* Applying the **abstract** keyword to mark a class as abstract.
* Use **deprecated**, **experimental** to keep compatibility**.**
* Prefer using records to design value objects. (automatically lifecycle management, operator overloads)
* Write testable code and write test cases with DUnit.

1. **Design Guidelines**

See [Design Guidelines for Developing Class Libraries](http://msdn.microsoft.com/en-us/library/ms229042.aspx)

1. **References**

* [Design Guidelines for Developing Class Libraries](http://msdn.microsoft.com/en-us/library/ms229042.aspx) （MSDN）
* JEDI Style Guide