PV178: Programming for .NET Framework

CLI Libraries, Base Class Library

Vojtěch Forejt, forejt@fi.muni.cz
Martin Osovský, osovsky@ics.muni.cz

Faculty of Informatics and **Institute of Computer Science**
Masaryk University

February 26, 2009

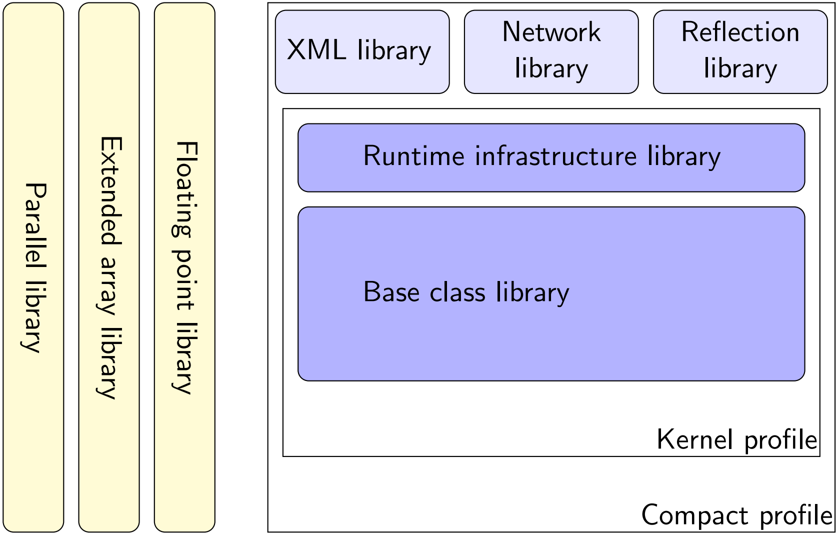
CLI Library – Goals

- Standard library
- Available for all .NET languages
- CLS compliant (almost)
- Consistent design patterns
- Provide features similar to *C Standard library* of 1990
- Support networking, XML, etc.

Taxonomy

- **Libraries** and **Profiles**
- Library is specified by
  - Set of types
  - Set of CLI features
  - Modification to types from other libraries
- CLI defines 7 libraries
- Profile is a set of libraries
- CLI defines 2 profiles: Kernel and Compact

Taxonomy cont.



Base Class Library – Namespaces

- `System`
- `System.CodeDom`
- `System.Collections`
- `System.Diagnostics`
- `System.Globalization`
- `System.IO`
- `System.Resources`
- `System.Text`
- `System.Text.RegularExpressions`

Arrays

- Single-dimensional arrays

```
int[] arrayA = new int[5];
int[] arrayB = { 1, 2, 3, 4, 5, 6 };
```

Arrays

- Single-dimensional arrays

```
int[] arrayA = new int[5];
int[] arrayB = { 1, 2, 3, 4, 5, 6 };
```

- Multi-dimensional

```
int[,] arrayC = new int[2, 3];
int[,] arrayD = { { 1, 2, 3 }, { 4, 5, 6 } };
```

Arrays

- Single-dimensional arrays

```
int[] arrayA = new int[5];
int[] arrayB = { 1, 2, 3, 4, 5, 6 };
```

- Multi-dimensional

```
int[,] arrayC = new int[2, 3];
int[,] arrayD = { { 1, 2, 3 }, { 4, 5, 6 } };
```

- Jagged arrays

```
int[][] arrayE = new int[5][];
arrayE[0] = new int[2];
```

Indexers

- Allow instances of classes to be indexed like arrays
- Like properties with parameters

```
public class MyCharArray
{
 private char[] arr;
 public MyCharArray() {...}
 public char this[int i]
 {
  get { return this.arr[i]; }
  set { this.arr[i] = value; }
 }
}
```

Collections

- Represent data structures for storing multiple objects.
- Two basic types in BCL: Lists, Dictionaries.

Lists	Contains simple objects.
Dictionaries	Contains key-values pairs.

Enumeration and Iteration

- Interface `IEnumerable` implements method `IEnumerator GetEnumerator();`
- Class implementing `IEnumerator` does the enumeration.
  - `Boolean MoveNext()` increments the index and returns `true` if the element exists.
  - read-only property `Object Current` returns the object at the index.
  - `void Reset()` sets the index to -1.

Enumeration and Iteration

- Interface `IEnumerable` implements method `IEnumerator GetEnumerator();`
- Class implementing `IEnumerator` does the enumeration.
  - `Boolean MoveNext()` increments the index and returns `true` if the element exists.
  - read-only property `Object Current` returns the object at the index.
  - `void Reset()` sets the index to -1.
- Iterator is a section of code that returns an ordered sequence of values of the same type
  - Uses `yield return` and `yield break` statements to return elements and stop iterating
  - Must return `IEnumerable` or `IEnumerator`
  - Multiple iterators can be implemented in one class.
  - The `IEnumerator` class is generated by compiler

Enumeration and Iteration cont.

Example `Iteration`

Comparison

- For sorted collections or sets we need comparison support.
- Equality testing – virtual method `Equals`.
- Comparison – `IComparable` interface
  - method `int CompareTo()` returns positive integer if value of this is greater than argument's value, zero if they are equal and negative integer otherwise.

ICollection

- Implements `IEnumerable`
- Property `int Count`
- Method `void CopyTo(Array array, int index)`
- Members used for synchronization

- Implementing classes

`Stack`	Methods `Push`, `Pop`, `Peek`.
`Queue`	Methods `Enqueue` and `Dequeue`.
`BitArray`	Compact array of bit values.

IList

- Implements `ICollection`.
- Important methods:

`Add`	Adds an item
`Clear`	Removes all items
`Contains`	Determines whether a value is contained
`IndexOf`	Returns index of first occurence of a value
`Insert`	Inserts an item at the index
`Remove`	Removes first occurence of the item
`RemoveAt`	Removes the item at the index

- Important properties:

`IsFixedSize`	
`IsReadOnly`	
`Item`	Gets or sets the element at the index

- Implementing class in BCL

  `ArrayList` - Array whose size is dynamically increased as required

ArrayList

Example `CollectionsExample.cs`

IDictionary

- Implements `ICollection`
- Important members

`Item`	Gets or sets element with specified key.
`Keys`	
`Values`	
`Add`	
`Clear`	
`Contains`	
`Remove`	

- Implementing classes in BCL

`HashTable`	Key/value pairs organised based on the hash code of the key.
`SortedList`	Key/value pairs sorted by keys, accessible by key and index.

SortedList

Example `CollectionsExample.cs`

Generic Collections

- Namespace `System.Collections.Generic`
- Defined with type parameters:
  - Interfaces `IEnumerable<T>`, `IComparable<T>`, `IList<T>`, `IDictionary<TKey, TValue>`
  - Classes `Dictionary<T>`, `SortedList<T>`, `Stack<T>`, `Queue<T>`
  - "New" classes: `LinkedList`, `List`, `SortedDictionary`

System.IO Namespace – Overview

- Reading and writing to streams and files
- File and directory support

Streams

- Stream is a potentially infinite sequence of elements of certain type.
- Operate on a resource whose internal structure is hidden.
- Basic operations: reading, writing.
- Types:
    - binary vs. character vs. other
    - read-only vs. write-only vs. read/write
    - synchronous (blocking) vs. asynchronous (non-blocking)
    - sequential vs. random access
    - “low-level” vs. adapter streams

System.IO.Stream Class

- Abstract class
- Basic methods and properties:

`Write`	Write a block (array) of bytes to the stream.
`WriteByte`	Write a single byte to the stream.
`Read`	Read a block of bytes from the stream.
`ReadByte`	Read a single byte from the stream.
`Close`	Close the stream.
`Seek`	Move the internal position pointer.
`Flush`	Flush the internal cache.

`CanRead`
`CanWrite`
`CanSeek`
`Position`

I/O Timeouts

- `Read()`,`Write()` are blocking operations.

`CanTimeout`	Does the stream support timeouts?
`ReadTimeout`	read timeout (usually in ms).
`WriteTimeout`	write timeout (usually in ms).

- May utilize asynchronous streams (not in this lecture).

Inheriting Classes

- `FileStream`
  - Operates on file system file.
  - Allows to set access permissions in a constructor.
  - Support for seeking, no support for timeouts.

Inheriting Classes

- `FileStream`
  - Operates on file system file.
  - Allows to set access permissions in a constructor.
  - Support for seeking, no support for timeouts.
- `BufferedStream`
  - Adapter stream.
  - Implements caching on a nested stream.
  - Buffer size may be set in constructor.

Inheriting Classes

- `FileStream`
  - Operates on file system file.
  - Allows to set access permissions in a constructor.
  - Support for seeking, no support for timeouts.
- `BufferedStream`
  - Adapter stream.
  - Implements caching on a nested stream.
  - Buffer size may be set in constructor.
- `MemoryStream`
  - Useful to compose data in memory.

Stream Example

- Example `StreamsExample`.

System.IO.TextReader

- Reading character data
- Similar to `Stream`
- Abstract class for `StringReader` and `StreamReader`.

Read	Read single character
ReadBlock	Read block (array) of characters
ReadLine	Read single line as string
ReadToEnd	Read to the end of reader and return as string

System.IO.TextWriter

- Writing character data
- Similar to `Stream`
- Abstract class for `StringReader` and `StreamReader`.

Write	write single object (of various types)
WriteLine	like Write, appends newline character

StreamWriter Example

Example `StreamWriterExample`.

Compression

- adapter streams: `DeflateStream`, `GZipStream`
- `DeflateStream` implements deflate algoritm
- `GZipStream` as above, in standard format.

Compression Example

Example `Compress`.

Filesystem classes

- Static `File` and non-static `FileInfo` classes
  - Copying, deleting, moving or creating files
  - Creating of `FileStream`
- static `Directory` and non-static `DirectoryInfo` classes
  - Creating, moving, and enumerating through directories and subdirectories

Filesystem Example

Example `FileSystemExamples`