Module 6

Reading and Writing Local Data

Module Overview

- Reading and Writing Files
- Serializing and Deserializing Data
- Performing I/O by Using Streams

Lesson 1: Reading and Writing Files

- Reading and Writing Data by Using the File Class
- Manipulating Files
- Manipulating Directories
- Manipulating File and Directory Paths
- Demonstration: Manipulating Files, Directories, and Paths

Reading and Writing Data by Using the File Class

- The **System.IO namespace** contains classes for manipulating files and directories

- The **File** class contains atomic read methods, including:
  - **ReadAllText(String)**
  - **ReadAllLines(String[])**

- The **File** class contains atomic write methods, including:
  - **WriteAllText(String)**
  - **AppendAllText(String)**

Manipulating Files

- The **File** class provides static members

```
File.Delete(...);
bool exists = File.Exists(...);
DateTime createdOn = File.GetCreationTime(…);
```

- The **FileInfo** class provides instance members

```
FileInfo file = new FileInfo(...);
...
string name = file.DirectoryName;
bool exists = file.Exists;
file.Delete();
```

Manipulating Directories

- The **Directory** class provides static members

```
Directory.Delete(...);
bool exists = Directory.Exists(...);
string[] files = Directory.GetFiles(...);
```

- The **DirectoryInfo** class provides instance members

```
DirectoryInfo directory = new DirectoryInfo(...);
...
string path = directory.FullName;
bool exists = directory.Exists;
FileInfo[] files = directory.GetFiles();
```

Manipulating File and Directory Paths

The **Path** class encapsulates file system utility functions

```
// Put together a full path from parts
string settingsPath =
          Path.Combine("d:\apps\", "\coffee", "settings.json")


// Check to see if path has an extension.
bool hasExtension = Path.HasExtension(settingsPath);


// Get the extension from the path.
string pathExt = Path.GetExtension(settingsPath);


// Get path to temp file.
string tempPath = Path.GetTempFileName();
// Returns C:\Users\LeonidsP\AppData\Local\Temp\ABC.tmp
```

Demonstration: Manipulating Files, Directories, and Paths

In this demonstration, you will use the **File**, **Directory**, and **Path** classes to build a utility that combines multiple files into a single file

Lesson 2: Serializing and Deserializing Data

- Serialization Formats
- Serializing Objects as JSON by Using JSON.Net
- Serializing Objects as JSON with .NET only
- Serializing Objects as Binary
- Serializing Objects as XML
- Demonstration: Serializing Objects as JSON using JSON.Net
- Creating a Custom Serializer

Serialization Formats

- Binary

```
10101010101011111010110101010110101111111010101101100001
```

- XML

```
<SOAP-ENV:Envelope ...>
  <SOAP-ENV:Body>
    ...
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

- JSON

```
{
  "ConfigName":"FourthCoffee_Default",
  "DatabaseHostName":"database209.fourthcoffee.com"
}
```

Serializing Objects as JSON by Using JSON.Net

- JSON.Net – a 3rd party library embraced by MS

- Serialize as JSON

```
// Create the object you want to serialize.
ServiceConfiguration config = ServiceConfiguration.Default;

// Serialize the object to a string
var jsonString = JsonConvert.Serialize(config);
```

- Deserialize from JSON

```
// Deserialize to the desired type
var deserializedConfig =
JsonConvert.DeserializeObject<ServiceConfiguration>(jsonString);
```

Serializing Objects as JSON with .NET Only

- Serialize as JSON

```
ServiceConfiguration config = ServiceConfiguration.Default;
DataContractJsonSerializer jsonSerializer
   = new DataContractJsonSerializer(config.GetType());
FileStream buffer = File.Create(@"C:\fourthcoffee\config.txt");
jsonSerializer.WriteObject(buffer, config);
buffer.Close();
```

- Deserialize from JSON

```
DataContractJsonSerializer jsonSerializer = new
   DataContractJsonSerializer(
      typeof(ServiceConfiguration));
FileStream buffer = File.OpenRead(@"C:\fourthcoffee\config.txt");
ServiceConfiguration config = jsonSerializer.ReadObject(buffer)
   as ServiceConfiguration;
buffer.Close();
```

Creating a Serializable Type

Implement the **ISerializable** interface

```
[Serializable]
public class ServiceConfiguration : ISerializable
{
   ...
   public ServiceConfiguration(
      SerializationInfo info, StreamingContext ctxt)
   {
      ...
   }

   public void GetObjectData(
      SerializationInfo info, StreamingContext context)
   {
      ...
   }
}
```

Serializing Objects as Binary

- Serialize as binary

```
ServiceConfiguration config = ServiceConfiguration.Default;
IFormatter formatter = new BinaryFormatter();
FileStream buffer = File.Create("C:\\fourthcoffee\\config.txt");
formatter.Serialize(buffer, config);
buffer.Close();
```

- Deserialize from binary

```
IFormatter formatter = new BinaryFormatter();
FileStream buffer = File.OpenRead("C:\\fourthcoffee\\config.txt");
ServiceConfiguration config
   = formatter.Deserialize(buffer) as ServiceConfiguration;
buffer.Close();
```

Serializing Objects as XML

- Serialize as XML

```
ServiceConfiguration config = ServiceConfiguration.Default;
IFormatter formatter = new SoapFormatter();
FileStream buffer = File.Create(@"C:\fourthcoffee\config.xml");
formatter.Serialize(buffer, config);
buffer.Close();
```

- Deserialize from XML

```
IFormatter formatter = new SoapFormatter();
FileStream buffer = File.OpenRead(@"C:\fourthcoffee\config.xml");
ServiceConfiguration config
   = formatter.Deserialize(buffer) as ServiceConfiguration;
buffer.Close();
```

Demonstration: Serializing Objects as JSON using JSON.Net

In this demonstration you will see how to serialize and deserialize objects using JSON.NET

Creating a Custom Serializer

Implement the **IFormatter** interface

```
class IniFormatter : IFormatter
{
   public ISurrogateSelector SurrogateSelector { get; set; }
   public SerializationBinder Binder { get; set; }
   public StreamingContext Context { get; set; }

   public object Deserialize(Stream serializationStream)
   {
      ...
   }

   public void Serialize(Stream serializationStream, object graph)
   {
      ...
   }
}
```

Lesson 3: Performing I/O by Using Streams

- What are Streams?
- Types of Streams in the .NET Framework
- Reading and Writing Binary Data by Using Streams
- Reading and Writing Text Data by Using Streams
- Demonstration: Generating the Grades Report Lab

What are Streams?

- The **System.IO namespace** contains a number of stream classes, including:
  - The abstract **Stream** base class
  - The **FileStream** class
  - The **CryptoStream** class
  - The **GZipStream** class
  - Many others
- Typical stream operations include:
  - Reading chunks of data from a stream
  - Writing chunks of data to a stream
  - Querying the position of the stream

Using a Buffer

```
// configurable buffer size
const int BUFFER_SIZE = 1024;

// initialize buffer
byte[] buffer = new byte[BUFFER_SIZE];

// get the next chunk of data
while (stream.Read(buffer, 0, buffer.Length) > 0)
{
    ProcessData(buffer);
}
```

int Read (byte[] buffer, int offset, int count)

- Works with buffer of any size
- Returns the number of bytes actually read

Types of Streams in the .NET Framework

- Classes that enable access to data sources include:

Class	Description
FileStream	Exposes a stream to a file on the file system.
MemoryStream	Exposes a stream to a memory location.
NetworkStream	Exposes a stream to a network location.

- Classes that enable reading and writing to and from data source streams include:

Class	Description
StreamReader	Read textual data from a source stream.
StreamWriter	Write textual data to a source stream.
BinaryReader	Read binary data from a source stream.
BinaryWriter	Write binary data to a source stream.

Reading and Writing Binary Data by Using Streams

You can use the **BinaryReader** and **BinaryWriter** classes to stream binary data

```
string filePath = “C:\\fourthcoffee\\applicationdata\\settings.txt";

// Underlying stream to file on the file system.
FileStream file = new FileStream(filePath);

// BinaryReader object exposes read operations on the underlying
// FileStream object.
BinaryReader reader = new BinaryReader(file);

// BinaryWriter object exposes write operations on the underlying
// FileStream object.
BinaryWriter writer = new BinaryWriter(file);
```

Reading and Writing Text Data by Using Streams

You can use the **StreamReader** and **StreamWriter** classes to stream plain text

```
string filePath = “C:\\fourthcoffee\\applicationdata\\settings.txt";

// Underlying stream to file on the file system.
FileStream file = new FileStream(filePath);

// StreamReader object exposes read operations on the underlying
// FileStream object.
StreamReader reader = new StreamReader(file);

// StreamWriter object exposes write operations on the underlying
// FileStream object.
StreamWriter writer = new StreamWriter(file);
```

Demonstration: Generating the Grades Report Lab

In this demonstration, you will learn about the tasks that you will perform in the lab for this module

Lab: Generating the Grades Report

- Exercise 1: Serializing Data for the Grades Report as XML
- Exercise 2: Previewing the Grades Report
- Exercise 3: Persisting the Serialized Grade Data to a File

Estimated Time: 45 minutes

Lab Scenario

You have been asked to upgrade the Grades Prototype application to enable users to save a student's grades as an XML file on the local disk. The user should be able to click a new button on the StudentProfile view that asks the user where they would like to save the file, displays a preview of the data to the user, and asks the user to confirm that they wish to save the file to disk. If they do, the application should save the grade data in XML format in the location that the user specified.

Module Review and Takeaways

- Review Questions