ExcelShore®
Operational Excellence

Cybage®
Delivering Value. Scientifically.

File IO and Serialization

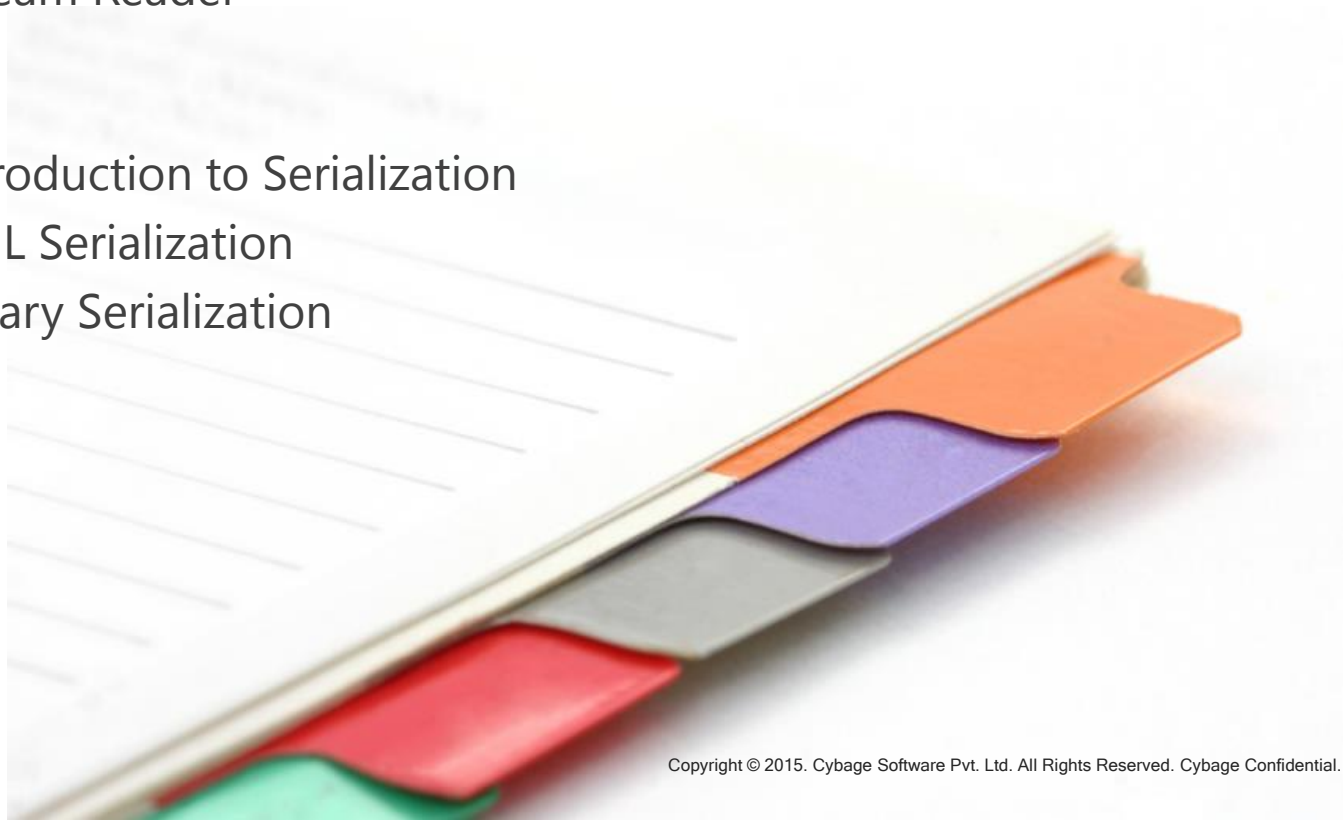
Sushant Banerjee | sushantba@cyabge.com | 7221

Cybage®
Delivering Value. Scientifically.

Agenda

- Introduction to File IO
- File Stream
- Stream Writer
- Stream Reader

- Introduction to Serialization
- XML Serialization
- Binary Serialization



What is File I/O

- File I/O or File Input / Output or File Handling is term used frequently when you work with files and directories system.
- In .NET Framework **System.IO** is the namespace used to work with files and directories.

Files and Streams

- A file is a collection of persistent data stored in the disk using a file name with a directory path.
- When you open a file for reading or writing it becomes a stream.
- A stream is a sequence of bytes traveling from a source to destination over a communication path.
- Streams may be input stream and output stream.
  - The input stream is used for reading data from files.
  - The output stream is used for writing data into files.

FileStream Class

- The FileStream class is used to read from, write to, open and close files on a file system.
- When you create an object of the FileStream class you pass different parameters to the constructor of this class.
  - String path
  - FileMode
  - FileAccess
  - FileShare

FileMode Enum

- **Append** : Opens the file if it exists and seeks to the end of the file or creates a new file.
- **Create** : The OS will create a new file. If the file already exists, it will be overwritten.
- **CreateNew** : The OS will create a new file. If the file already exists a System.IO.IOException is thrown.
- **Open** : The OS should open an existing file. If the file does not exist a System.IO.FileNotFoundException is thrown.
- **OpenOrCreate** : The OS should open a file if it exists, otherwise a new file should be created.
- **Truncate** : The OS should open a file and truncates its size to zero bytes.

FileAccess Enum

- **Read** : Read access to the file. Data can be read from the file.
- **Write** : Write access to the file. Data can be written to the file.
- **ReadWrite** : Read and Write access to the file. Data can be written to and read from the file.

FileShare Enum

- **Read** : Allows subsequent opening of the file for reading.
- **Write** : Allows subsequent opening of the file for writing.
- **ReadWrite** : Allows subsequent opening of the file for reading or writing.
- **None** : Declines sharing of the current file. Any request to open the file will fail until the file is closed.
- **Delete** : Allows subsequent deleting of a file.

Character Data I/O

- The following classes can be used to perform read and write operation
- **StreamWriter** : This class can be used to write character data to a stream.
- **StreamReader** : This class can be used to read character from a byte stream.

Binary Data I/O

- The following classes are used specially for input and output operations on binary data :
- **BinaryReader** : Reads primitive data types as binary values.
- **BinaryWriter** : Writes primitive data types in binary format.

Serialization

What is Serialization

- Serialization is the process of converting an object into a stream of bytes
- This stream of bytes can be stored permanently in database, file
- This stream of bytes not only contains just data but information about object's type, version, culture and assembly name
- The main purpose of serialization is to save the state of an object and recreate it when needed
- The reverse process of recreating object from stream of bytes is known as **deserialization**.

Uses for Serialization

- Using serialization a developer can perform following actions :
- Sending the object to a remote application using Web Service.
- Passing an object from one domain to another.
- Passing an object through firewall as an XML string.
- Maintaining security or user specific information across applications.

Creating Serializable Object

- **System.Runtime.Serialization** provides classes for serializing and deserializing objects.
- You need to apply an attribute **SerializableAttribute** to a type to indicate that instances of this type can be serialized.
- If you do not want a field within your class to be serializable, apply the attribute **NonSerializedAttribute**.

Types of Serialization

- Serialization can be following types :
- **Binary Serialization** : Binary serialization uses binary encoding and saves objects state in binary format.
- **XML Serialization** : XML Serialization serializes the object into an XML stream and saves the objects state in xml format.

Binary Vs. XML Serialization

- **System.Runtime.Serialization.Formatters.Binary** provides all the necessary classes for serializing and deserializing objects in binary format.
- You need to use **BinaryFormatter** class for binary serialization and deserialization.

- **System.Xml.Serialization** contains the classes necessary for serializing and deserializing objects in XML format.
- You need to use **XmlSerializer** class for xml serialization and deserialization.

Bibliography, Important Links

- http://msdn.microsoft.com/en-us/library/hyz69czz(v=vs.110).aspx
- http://msdn.microsoft.com/en-us/library/1c9txz50(v=vs.110).aspx
- http://www.codeproject.com/Articles/274062/Improved-Multi-Threading-Support-in-Net
- http://msdn.microsoft.com/en-us/library/k3352a4t(v=vs.110).aspx
- http://msdn.microsoft.com/en-IN/library/z0w1kczw.aspx
- http://msdn.microsoft.com/en-IN/library/ms233843.aspx

Any Questions?

Thank you!