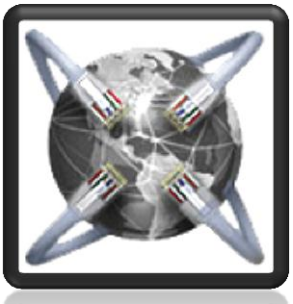
# Chapter 5

## Streaming Classes In .NET

**المواضيع الرئيسية فى هذا الفصل:**

1- مقدمة في Data Streaming Architecture
2- التعامل مع Streaming Classes In .NET
3- Streaming Objects من خلال الـ Serialization In .NET



## 5.1: مقدمة في Data Streaming Architecture:

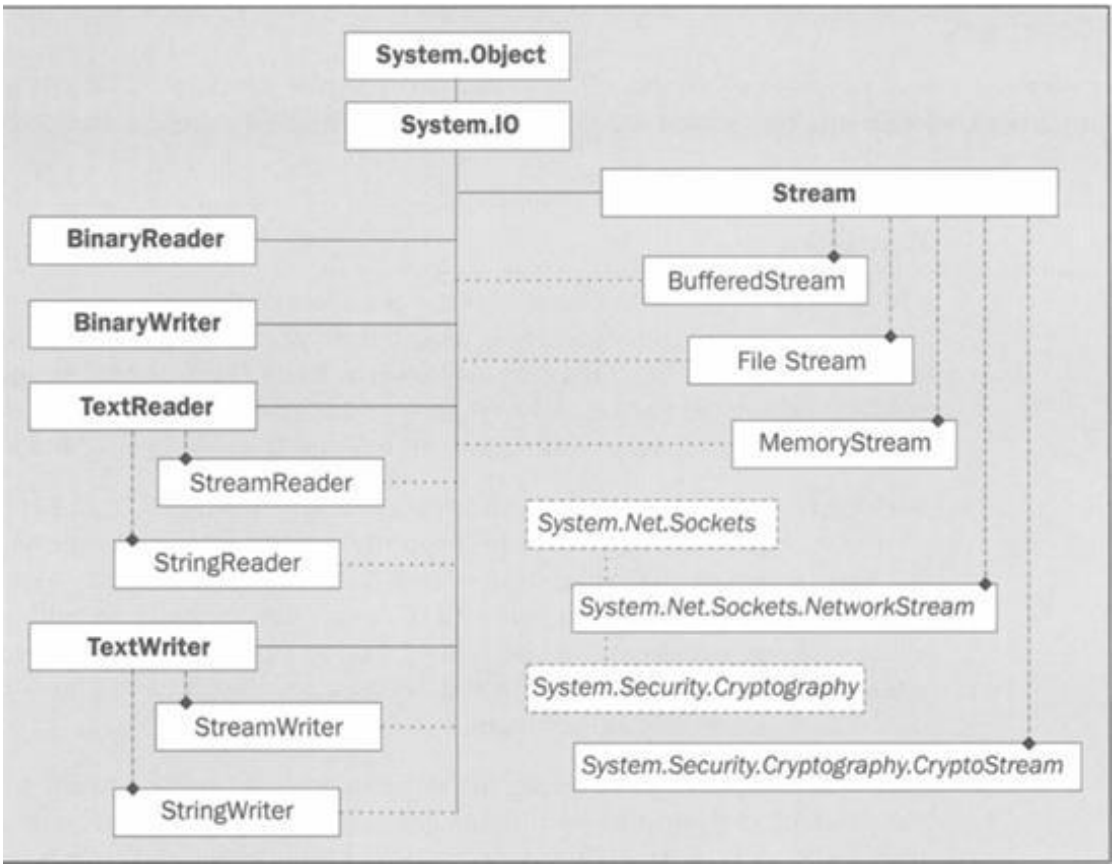
إن الهدف من الـ Streaming تسهيل عملية تحويل البيانات من شكل إلى آخر لإرسالها أو تخزينها أو التعامل معها حيث تسهل علينا عملية تحويلها إلى Bytes وإرسالها وهو ما حل الكثير من المشاكل التي كانت تواجه المبرمجين في التعامل مع Binary Data والملفات الكبيرة وكذلك الـData Multimedia كإرسال الـVideo والصوت وغيره بشكل متتابع.

في الشبكات فإن عملية الإرسال كـStream تعني بأن يتم تحويل البيانات إلى مجموعة من الـBytes تعبأ و تجهز في Buffer وترسل على أجزاء ومن البروتوكولات المعروفة والتي تستخدم هذا الأسلوب في عملية النقل بروتوكول الـTCP حيث يمكن إرسال أي حجم من البيانات من خلاله على أجزاء بشكل Fragments بحيث يكون لكل Fragment رقم فريد Unique ID خاص لكي يتم تجميعها لاحقا (لمزيد من المعلومات أنظر الفصل الأول) إن الفرق بين الـTCP والـUDP في عملية الإرسال هو أن الأخير يقوم بإرسال البيانات دفعة واحدة ولا يقسمها على أجزاء كما في الـTCP ولذلك يعتبر الـTCP كـ Streaming Transport Protocol وبالتالي فإننا لا نستطيع إرسال ملفات كبيرة من خلال الـUDP وذلك لأنه محدود بمحدودية Maximum Size of Ethernet Frame بحيث لا يمكن إرسال ملف يفوق حجمه الـ Bytes1500 إلا إذا تم تقسيم هذا الملف وإرساله على أجزاء بشكل يدوي وعدم إمكانية إرسال High Video Quality بشكل متتابع ومستمر وبزمن الحقيقي ولحل هذه المشكلة قام فريق من الـ IETF بعمل بروتوكول وسيط بين الـUDP والـApplication Layer وهو الـ RTP - Real Time Transport Protocol (http://www.ietf.org/rfc/rfc1889.txt) إن الهدف الرئيسي من هذا البروتوكول هو أن يتم دعم عملية الإرسال كـStreaming من خلال بروتوكول الـUDP بحيث نستفيد من سرعة الـUDP في عملية النقل وإمكانياته في الإرسال كـMulticast و Broadcast لإرسال High Quality Video Streaming وأحجام كبيرة من البيانات بسرعة كبيرة وبزمن الحقيقي مع إمكانية لتصليح الـFrames التي قد تصل مشوهة وذلك من خلال خوارزميات FEC – Forward Error Correction (http://www.packetizer.com/rfc/rfc2733) وبالتالي محاولة الإصلاح بدلا من إعادة طلب الإرسال كما في الـTCP ومن البروتوكولات التي تعتمد مبدأ الـRTP لكن من خلال الـTCP بروتوكول الـ RTSP Real Time Streaming Protocol (http://tools.ietf.org/html/rfc2326) حيث يستخدم لإرسال الـVideo من الـMedia Server إلى الـMedia Player عند الـClient على أجزاء بحيث نتمكن من مشاهدة الـVideo المسجل بدون الحاجة لتحميل الملف كاملا ثم مشاهدته وكذلك يمكن أن يستخدم لإرسال Live Video Streaming من متصل إلى الـMedia Server ثم إلى بقية المتصلين وبشكل مستمر.

في البرمجة وخاصة في النظم الموزعة غالبا ما نحتاج إلى تحويل الـObject إلى بيانات يمكن أن ترسل إلى الطرف الآخر وغالبا ما يتم تحويلها إما إلى XML كما هو الحال في الـWeb Service أو على شكل Binary Data وتسمى عملية التحويل هذه بالـSerialization وهو ما سيتم الحديث عنه لاحقا في هذا الفصل.

## 5.2: التعامل مع Streaming Classes In .NET:

تدعم الـNET. عمليات الـData Streams بمجموعة من الـClasses والمندرجة تحت Namespace System.IO ويحتوي فضاء الأسماء هذا على مجموعة من الـClasses والتي تستخدم في تحويل الـData إلى Bytes Stream والعكس أو في التعامل مع الـInput/Output Devices كتعامل مع الملفات Read/Write على Data Storage Device. ويبين الشكل 5.1 أهم الـ Streaming Classes في الدوت نيت.




الشكل 5.1 ويبين أهم الـStreaming Classes في الدوت نيت

كما هو الحال في الـSocket فإن عملية القراءة والكتابة باستخدام الـIO Streaming Classes يمكن أن تنفذ بطريقة متزامنة Synchronies وغير متزامنة Asynchronous وبشكل افتراضي تعمل معظمها بالأسلوب المتزامن لكن العيب فيه هو تأثيره الشديد على أدائية النظام إذ يقوم بإغلاق الـProcessing Unit في الـThread الرئيسي الذي يعمل عليه بحيث لا يسمح بتنفيذ أي أمر آخر إلا بعد الانتهاء من العملية الجارية ولا ينصح ابدآ باستخدام الأسلوب المتزامن في حالة إذا كنت تتعامل مع أجهزة قراءة وكتابة بطيئة نسبيا مثل الـFloppy Disk أو الـMagnetic Tape ويفضل في هذه الحالة أن تكون عملية القراءة أو الكتابة من خلال Thread منفصل (لمزيد من المعلومات أنظر الفصل الثاني).

### 5.2.1 الدوال والخواص التي تتشارك بها معظم الـ Streaming Classes In .NET:

أولا الدوال التي تتشارك بها معظم الـStreaming Classes في الدوت نيت وهي:

| Method | Description |
|---|---|
| Read , ReadByte | تستخدم Read لقراءة Stream Data وتخزينه في Bytes Buffer ويمكن تحديد عدد البايتات التي سيتم قراءتها من خلال الـLength لتحديد حجمن البيانات المراد قراءته و تمرير الـOffset لتحديد نقطة البدء كما ويمكن باستخدام الـReadByte معرفة نهاية الـStream حيث ترجع قيمة -1 في حالة انتهاء الـStream. |
| Write , WriteByte | تستخدم Write لقراءة Bytes Buffer وتخزينه على الـ Stream Data Object ويمكن تحديد عدد البايتات التي سيتم كتابتها من خلال الـLength لتحديد حجمن البيانات المراد كتابتها و تمرير الـOffset لتحديد نقطة البدء كما ويمكن باستخدام الـWriteByte معرفة نهاية الـStream حيث ترجع قيمة -1 في حالة انتهاء الـStream. |
| Flush | وتستخدم لتفريغ محتويات الـBuffer بعد إتمام العملية المحددة حيث يتم نقل محتويات الـBuffer إلى الـDestination الذي تم تحديده في الـ Stream Object. |
| Close | ستخدم لإغلاق الـStream وتحرير الـResources المحجوزة من قبل الـStream Object وينصح باستخدامها في الجزء الخاص بـ Finally block ولتأكد من أن الـStream سيتم إغلاقه وتحرير كافة الموارد في حالة حدوث أي Exception إثناء التنفيذ ولضمان عدم بقاء هذه الموارد في الذاكرة بعد إغلاق البرنامج. |
| SetLength | وتستخدم لتحديد حجم الـStream والذي نريد إرساله أو استقباله لكن في حالة إذا كان الـStream أقل من المحدد في الـSetLength سوف يؤدي ذالك إلى انقطاع الـStream وعدم وصوله بشكل سليم ، لن تستطيع استخدام هذه الخاصية إلا إذا تأكدت انك تملك الصلاحية لذلك من خلال الخاصية CanWrite و CanSeek لذا ينصح بفحص الصلاحية أولا قبل تحديد حجم الـStream . |
| Seek | وتستخدم لتحديد موقع القراءة أو الكتابة على الـStream وذلك بتمرير الـOffset لتحديد الموقع الجديد للقراءة أو الكتابة. |

ثانيا الخواص التي تتشارك بها معظم الـStreaming Classes في الدوت نيت وهي:

| Prosperity | Description |
|---|---|
| CanRead , CanWrite | وتستخدم لمعرفة إذا كان الـStream المستخدم يقبل عملية القراءة أو الكتابة أم لا حيث ترجع قيمة True في حالة إذا كان يقبل و False في حالة أنه لا يقبل ويستخدم عادة قبل إجراء عملية القراءة أو الكتابة لفحص الصلاحية قبل المحاولة. |
| CanSeek | ترجع هذه الخاصية قيمة True في حالة كان الـStream يدعم الـSeeking و False في حالة أنه لا يدعم الـSeeking وفي العادة تدعم الـClasses التي تستخدم Backing Storage هذه العملية مثل الـFileStream وعندها ترجع قيمة True |

| | |
|---|---|
| CanTimeout | وترجع قيمة True في حالة إذا كان الـstream يحتوي على خاصية الـTimeout والتي تعطي وقت محدد للعملية. |
| Length | و تستخدم لمعرفة حجم الـStream بالـByte ويمكن الاستفادة منها لمعرفة نهاية الـStream أو لتحديد حجم الـArray بناء على حجم الـStream |
| Position | وتأخذ Get أو Set لتحديد الموقع الحالي للـStream إثناء القراءة أو الكتابة. |
| BaseStream | وتستخدم عادة مع الـStreamReader والـStreamWriter والـBinaryReader والـBinaryWriter لإرجاع البيانات أو التعامل معها كـStream Data. |

ثالثا: الدوال غير المتزامنة Asynchronous Methods والتي تتشارك بها معظم الـ Streaming Classes بالدوت نيت:

| Method | Description |
|---|---|
| BeginRead , BeginWrite | وتستخدم لعملية القراءة و الكتابة بالطريقة الغير المتزامنة وتأخذ خمسة باروميترات وهي:<br><br>1-الـBytes Buffer وهو Bytes Array ويستخدم لعملية القراءة أو الكتابة عليه بحسب الدالة.<br><br>2-الـoffset وهو مؤشر لتحديد موقع القراءة أو الكتابة من وعلى الـBytes Buffer.<br><br>3-الـNumByte والذي سوف يتم فيه تحديد الحد الأقصى من البايتات التي سيتم كتابتها أو قراءتها.<br><br>4-الـAsyncCallback وهو Delegate يتم استدعائه لإنهاء عملية القراءة أو الكتابة بدون حجز الـ Thread وحتى لا يبقى الـ Stream في وضعية الانتظار.<br><br>5-الـStateobject وهو User Provided Object ويستخدم لتمرير بيانات إلى دالة الـEnd. |

### 5.2.2 الـ Streaming Classes In .NET:

تقسم الـ Stream Classes في الدوت نيت إلى قسمين فمنها ما يستخدم Backing storage كـBuffer مثل الـ FileStream و الـBufferedStream والـMemoryStream وكذلك فإن بعضها لا يستخدم Backing Storage مثل الـNetworkStream وفي هذه الحالة فإن الـ Backing Storage Classes تحتاج إلى حجز مقدار معين في الذاكرة لإتمام العملية المطلوبة والتي يتم تحديدها عند التعامل مع هذه الـClasses فمثلا في الـFileStream يجب تحديد حجم الملف المطلوب قراءته أو كتابته لإتمام العملية حيث يتم حجز هذا المقدار في الذاكرة لتخزين محتويات الملف.

1- **BufferedStream Class** : ويستخدم بشكل أساسي لحجز مقدار معين من الذاكرة بشكل مؤقت لتنفيذ عملية معينة كما تستخدم بعض البرمجيات الـBuffering لتحسين الأدائية حيث تكون كذاكرة وسيطة بين المعالجة والإرسال أو الاستقبال وكمثال عليها برمجيات الطباعة حيث تستخدم الطابعة ذاكرة وسيطة لتخزين البيانات المراد طباعتها بشكل مؤقت ، و يكمن الهدف الأساسي من استخدام الـBuffering في العمليات التي يكون فيها المعالجة أسرع من عمليات الإدخال و الإخراج حيث يتم معالجة البيانات ووضعها في الـBuffer في انتظار إرسالها وهو ما يساهم في تحسين الأدائية بشكل كبير، وبشكل افتراضي يتم حجز 4096 bytes عند استخدام الـBufferedStream ويمكن زيادتها أو تقليلها حسب الحاجة.، ويستخدم كما يلي كمثال :

**An Example To Use The BufferedStream Class In .NET:**

```
using System;
using System.Text;
using System.IO;

class BufferedStreamExample
{

 static void Main(string[] args)
 {
  ASCIIEncoding asen = new ASCIIEncoding();
  byte[] BytesArr = asen.GetBytes("Hello Buffering");

  MemoryStream ms = new MemoryStream(BytesArr);
  ReadBufStream(ms);
 }

 public static void ReadBufStream(Stream st)
 {
  // Compose BufferedStream
  BufferedStream bf = new BufferedStream(st);
  byte[] inData = new Byte[st.Length];

  // Read and display buffered data
  bf.Read(inData, 0, Convert.ToInt32(st.Length));
  Console.WriteLine(Encoding.ASCII.GetString(inData));
 }

}
```

حيث قمنا بتحويل نص إلى Byte Array باستخدام الـ ASCII Encoding وتحميله في الـMemoryStream ثم إرساله إلى الدالة ReadBufStream لاستقبال الـData Stream وحملناه في ذاكرة مؤقتة باستخدام الـBufferedStream Class ثم قمنا بطباعة محتوياته بعد تحويله إلى نص مرة أخرى باستخدام الـ ASCII Encoding. ومن ثم طباعته على الشاشة.

2- **MemoryStream Class**: ويستخدم بشكل أساسي في عملية تخزين Stream بشكل مؤقت في الذاكرة وتحويله إلى Bytes أو العكس ويمكن أن يستخدم في عملية إرسال صورة أو ملف أو أي شيء آخر بحيث تكون كذاكرة مؤقتة Backing Storage لإجراء عملية الإرسال أو الاستقبال.

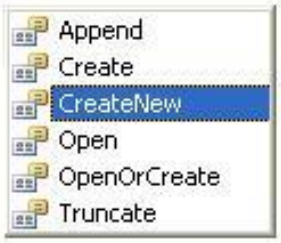
### An Example To Use The MemoryStream Class In .NET:

```
using System;
using System.IO;
using System.Drawing;
using System.Drawing.Imaging;

class MemoryStreamExample
{

 public byte[] ImageToBytes(Image ImageToBeSent)
 {
  MemoryStream ms = new MemoryStream();
  ImageToBeSent.Save(ms,ImageFormat.Jpeg);
  return ms.GetBuffer();
 }

 public Image BytesToImage(byte[] ImageBytes)
 {
  MemoryStream ms = new MemoryStream(ImageBytes);
  Image MyImage = Image.FromStream(ms);
  return MyImage;
 }

}
```

قمنا في المثال السابق باستخدام الـMemoryStream لتحويل صورة ما إلى Bytes ومن Bytes إلى صورة مرة أخرى بحيث نمرر لدالة ImageToBytes الصورة المراد تحويلها إلى Bytes وترجعها لنا كـBytes Array ويمكن أن تستخدم هذه الدالة في عملية إرسال الصورة عبر الـSocket وفي الدالة الثانية BytesToImage نمرر لها الصورة كـBytes Array وترجعها كـObject من نوع Image لعرضه على PictureBox Control على سبيل المثال بحيث تستخدم هذه الدالة عند استقبال الصورة من الـSocket لتحويل الـBytes التي تم استقبالها إلى صورة مرة أخرى.

3- **FileStream** : يستخدم الـFileStream بشكل أساسي لتعامل مع الملفات سواء للكتابة إلى ملف أو القراءة من ملف ويستخدم أيضا لتحويل الملف إلى Byte Array وحتى يسهل نقله عبر الشبكة و ويصنف على أنه Backing Storage Class حيث يستخدم كذاكرة مؤقتة لتحزين محتويات الملف لتعامل معه لاحقا سواء بكتابته على Storage Device أو إرساله عبر الـSocket ولاستخدامه يلزم في البداية تمرير الـFile Path للملف ونوع العملية المطلوبة كالتالي:

```
FileStream(@"C:\MyStream.txt",FileMode.);
```




- **Append:** للإضافة على ملف موجود مسبقا.

- **Create:** لإنشاء ملف جديد ويقوم بعمل Overwriting على الملف القديم في حالة وجوده.

- **CreateNew:** وهو كما في الـCreate إلا انه يعطي Exception في حالة إذا كان الملف موجودا.

- **Open**: لقراءة محتويات ملف محدد ويرجع Exception في حالة عدم وجوده.

- **OpenOrCreate:** في حالة إذا وجد الملف يقوم بقراءته وفي حالة عدم وجوده يقوم بإنشائه.

- **Truncate:** ويستخدم لحذف محتويات الملف وجعله فارغا.

كذلك يمكن تحديد الـFileAccess المسموح به كالقراءة فقط أو الكتابة فقط أو السماح بالقراءة والكتابة.

لمعرفة كافة الأشكال التي يمكن أن تستخدم مع الـFileStream يمكن الرجوع إلى الرابط التالي:

http://msdn.microsoft.com/en-us/library/system.io.filestream.aspx

وكمثال على استخدام الـFileStream في عملية تحويل ملف إلى Bytes ثم من Bytes إلى ملف مرة أخرى:

**An Example To Use The FileStream Class In .NET:**

```
using System;
using System.IO;

public class FileStreamExample
{
    public byte[] FileToBytes(string FilePathToLoad)
    {
        // Read The File and return it as bytes
        FileStream fs = new
             FileStream(FilePathToLoad,FileMode.Open);
```

```
        byte[] buffer = new byte[fs.Length];
        fs.Read(buffer, 0, buffer.Length);
        fs.Close();
        return buffer;
    }
 public void BytesToFile(byte[] ReceivedBytes, string FilePathToSave)
    {
        // Convert The Bytes and save it as a File
        FileStream fs = new FileStream(FilePathToSave,
          FileMode.CreateNew, FileAccess.Write);
        fs.Write(ReceivedBytes, 0, ReceivedBytes.Length);
        fs.Close();
    }
}
```

قمنا في المثال السابق باستخدام الـFileStream لتحويل ملف ما إلى Bytes ومن Bytes إلى ملف مرة أخرى ويمكن أن نستخدم هذه الطريقة في عملية إرسال ملف عبر الـSocket وتحويل الـBytes التي تم استقبالها لملف مرة أخرى.

4- **StreamReader** & **StreamWriter** Classes : ويستخدم في التعامل مع الـText Data بشكل Stream Data وكمثال يمكن أن يستخدما في قراءة بيانات أو جزء من بيانات من ملف نصي وكذلك الكتابة إلى ملف نصي وكمثال على استخدام الـStreamReader لقراءة ملف نصي وتحويله إلى عدة أشكال كتحويله إلى String أو Stream أو Bytes Array:

### An Example To Use The StreamReader Class In .NET:

```
using System;
using System.IO;
public class StreamReaderExample
{
    public string ReadAsStringFromTXTFile(string TXTFilePath)
    {
        // To Read The Full Text File and Return it as string
        StreamReader str = File.OpenText(TXTFilePath);
        return str.ReadToEnd();
    }
    public Stream ReadAsStreamFromTXTFile(string TXTFilePath)
    {
        // To Read The Full Text File and Return it as Stream
        StreamReader str = File.OpenText(TXTFilePath);
        return str.BaseStream;
    }
    public byte[] ReadAsBytesFromTXTFile(string TXTFilePath)
    {
        // To Read The Full Text File and Return it as Bytes
        StreamReader str = File.OpenText(TXTFilePath);
        byte[] buffer = new byte[str.BaseStream.Length];
        str.BaseStream.Read(buffer, 0, buffer.Length);
        return buffer;
    }
}
```

وكذلك يمكن استخدام الـStreamWriter للكتابة إلى ملف نصي وكما يلي كمثال:

**An Example To Use The StreamWriter Class In .NET:**

```
using System;
using System.IO;

public class StreamWriterExample
{
    public void SaveStringToTXTFile(string Text, string SaveFilePath)
    {
        // Save The Passed Text to a Text File
        StreamWriter SW = new StreamWriter(SaveFilePath);
        SW.WriteLine(Text);
    }
}
```

5- Classes **BinaryReader** & **BinaryWriter** و الـClass **NetworkStream**: يمكن من خلال BinaryReader و BinaryWriter قراءة أي نوع من البيانات كالملفات والصور أو النص وكذلك الكتابة إليه وكمثال على استخدام الـBinaryReader مع الـSocket لإرسال أي نوع من البيانات واستقباله قمنا باستخدام وسيط وهو NetworkStream Class ولا يحتوي هذا الـClass على Backing Storage لذا قمنا باستخدامه مع الـBinaryReader & BinaryWriter لإرسال واستقبال البيانات حيث يتم ربط الـNetworkStream Class مع الـTCP Socket لإجراء عمليات الإرسال والاستقبال كـStream Data ثم نقو باستخدام الـBinaryReader والـBinaryWriter لقراءة الـBytes وتخزينه بذاكرة قبل قراءته أو كتابته على الـNetworkStream Object وكمثال على ذلك:

**An Example To Use The BinaryWriter & BinaryReader Classes In .NET:**

```
using System;
using System.IO;
using System.Net.Sockets;

public class BinaryReaderAndWriterExample
{

    public void SendBinaryData(byte[] BinaryBytesData,
                                    string IPAddress,int Port)
    {
        // Declare a Simple TCP Socket
        TcpClient TCP_Socket = new TcpClient(IPAddress, Port);

        // Bind The TCP Socket with the NetworkStream
        NetworkStream myns = TCP_Socket.GetStream();

        // Specific Where to Write The Data
        BinaryWriter mysw = new BinaryWriter(myns);

        // Write The Binary Data To The NetworkStream Socket
        mysw.Write(BinaryBytesData);
    }
```

```
    public byte[] ReceiveBinaryData(int Port)
    {
        // Declare a Simple TCP Listener Socket
        TcpListener mytcpl = new TcpListener(Port);

        // Start The Listening
        mytcpl.Start();

        // Accept The Incoming Sending Request
        Socket mysocket = mytcpl.AcceptSocket();

        // Bind The Socket With The NetworkStream
        NetworkStream myns = new NetworkStream(mysocket);

        // Read The Incoming Stream and Return it as Bytes Array
        BinaryReader br = new BinaryReader(myns);
        byte[] buffer = new byte[br.BaseStream.Length];
        br.Read(buffer, 0, buffer.Length);
        return buffer;
    }
}
```

قمنا في المثال السابق بإنشاء دالتين الأولى للإرسال بيانات من خلال TCP Socket والثانية لاستقبال بيانات من TCP Socket وإرجاعها كـBytes حيث تم في الدالة الأولى تمرير البيانات المراد إرسالها إلى الـBinaryWriter ليتم كتابتها على الـNetworkStream لإرسالها عبر الـTCP Socket، وفي الدالة الثانية قمنا بربط الـNetworkStream مع الـBinaryReader ليتم قراءة البيانات المستقبلة وتخزينها في Buffer مؤقت وإرجاعها.

يفضل إجراء عملية الاستقبال من خلال Thread منفصل عن التطبيق وذلك لضمان عدم توقف البرنامج لحين الانتهاء من عملية الاستقبال ولمعرفة كيفية تطبيق ذلك على الدالة السابقة يمكن الرجوع إلى الفصل الثاني والثالث.

## 5.3: Streaming Objects من خلال الـ Serialization In .NET

في البرمجة عادة ما نحتاج إلى إرسال أو تخزين أو التعامل مع Objects داخل أو خارج النظام الذي نعمل عليه وقد بسطت الدوت نيت هذه العملية بوجود مجموعة من الـManaged Classes والتي تسمح بتحويل الـObjects إلى بيانات قابلة للإرسال/الاستقبال و التخزين/الاسترجاع وذلك بتحويلها إلى XML Data أو Binary Data بناء على طبيعة تلك البيانات والبروتوكولات المستخدمة في عملية نقلها فمثلا في الـ Web Service يتم تحويل الـMethods والبيانات الممررة إليها والبيانات الراجعة منها إلى XML Data من خلال بروتوكول وسيط يسمى SOAP Simple Object Access Protocol (لمزيد من المعلومات انظر الفصل الثامن) وذلك لتسهيل نقلها من خلال الـ HTTP. ونستطيع تلخيص أهم الـ Managed Serialization Classes In .NET ضمن الـ System.Runtime.Serialization إلى التالي:

| Classes | Description |
|---|---|
| Formatters , Formatter And FormatterServices | وتحتوي هذه الـ Classes على كل الوظائف الرئيسية لعمل Serialization لـ Object ما ومنها يستخدم الـ Serialize Method لإرسال Serializing Object عبر الـ Stream |

| | |
|---|---|
| | والـ Deserialize والتي تستخدم للاستقبال الـ Serializing Object من الـStream و تحويله إلى Object مرة أخرى.<br>وتأخذ عملية الإرسال نوعين هما الإرسال عبر بروتوكول الـ Soap ويأتي ضمن الـ Namespace System.Runtime.Serialization.Formatters.Soap حيث يجب إضافته إلى الـ Reference للمشروع .<br>والطريقة الثانية هي بتحويل الـObject إلى Binary Serial Stream ويأتي ضمن الـNamespace :<br>System.Runtime.Serialization.Formatters.Binary |
| FormatterConverter | حيث يحتوي على الـ IFormatter Interface والـ IConvertible Interface لأستخدمهما في عمليات تمثيل الـ Object حتى يمكن إرساله عبر الـ Serializing Stream. |
| ObjectIDGenerator | ويستخدم لتوليد رقم ID عشوائي لأي Object حيث يحتوي هذا الـ Class على Tow Methods الأول يقوم بتوليد رقم تسلسلي للـ Object يسمى GetID والثاني يرجع قيمة الـ ID الخاص بالـ Object ويسمى HasId ويأخذ كل منهما اسم الـ Object الذي نريد توليد رقم تسلسلي له وقيمة True أو False لتحديد فيما إذا كانت هذه هي المرة الأولى التي تم فيها توليد رقم تسلسلي للـ Object أم لا.<br>GetId ( object_name,out bool) → to Set an ID<br>HasId( object_name,out bool) → to Return the ID |
| SerializationInfo | ويحتوي على كل المعلومات التي يمكن أن نحتاج لها لعملية الـ Serialize و الـ Deserialize لأي Object مثلا AssemblyName و الـ FullTypeName والـ MemberCount وتحويلات الـ Data Types للـ Objects وغيرها... |

لسماح بتنفيذ الـSerialization على أي Class يجب أولا أن يتم إضافة الـ [Serializable] Attribute على رأس ذلك الـClass وكما يلي كمثال:

**An Example To Make a Serializable Class In .NET:**

```
using System;
using System.IO;
using System.Net.Sockets;

[Serializable]
public class CustomerDetails
{
    public int CustomerID;
    public string CustomerName= string.Empty;
    public string AcountID= string.Empty;
    public string Address = string.Empty;
    public string Phone = string.Empty;
    public CustomerDetails() { } // Class Constructor
}
```

يوجد طريقتين لعمل الـSerialization الطريقة الأولى كـ XML Serialize والطريقة الثانية كـ Binary Serialize وتعتمد الطريقة الأولى على تحويل الـObject الذي سيتم اشتقاقه من الـ Class السابق إلى XML Data يمكن إرسالها أو تخزينها بشكل Stream Data وكما يلي كمثال باستخدام الـ XmlSerializer Class لتخزين واسترجاع Object من الـ Class السابق كـ XML File:

**An Example To Serialize an object for a .NET Class:**

```
using System;
using System.Xml.Serialization;
using System.Xml;
using System.IO;

public class XMLSerializationExample
{
    static void Main()
    {
   CustomerDetails NewCustomer = new CustomerDetails();
        NewCustomer.CustomerID = 1;
        NewCustomer.AcountID = "800XXXXX";
        NewCustomer.CustomerName = "Fadi Abdelqader";
        NewCustomer.Address = "Amman/Jordan";
        NewCustomer.Phone = "+962XXXXXXX";

   // Serializing And Saving The Customer Information to XML File
   SerializingAndSaving(NewCustomer, @"C:\MyCustomer.XML");

   // Loading and Deserializing The Customer XML File
   CustomerDetails MyCustomer = Deserializing(@"C:\MyCustomer.XML");

   Console.WriteLine(MyCustomer.AcountID + " - " +
                               MyCustomer.CustomerName);
    }

public static void SerializingAndSaving(CustomerDetails CustData,
string XMLSaveFilePath)
    {
FileStream XMLStream = new FileStream(XMLSaveFilePath,
                                          FileMode.Create);
XmlSerializer XMLSer = new XmlSerializer(typeof(CustomerDetails));
   XMLSer.Serialize(XMLStream, CustData);
   XMLStream.Close();
    }

public static CustomerDetails Deserializing(string XMLLoadFilePath)
    {
 FileStream XMLFileStream = new FileStream(XMLLoadFilePath,
                                          FileMode.Open);
 XmlSerializer ser = new XmlSerializer(typeof(CustomerDetails));
 CustomerDetails CustData = (CustomerDetails)
   ser.Deserialize(XMLFileStream);
   XMLFileStream.Close();
   return CustData;
    }
}
```

يحتوي المثال السابق على دالتين الأولى نمرر لها الـClass Object والـXML File Path حيث تم استخدام الـXmlSerializer لتحويل الـClass Object إلى XML Stream ثم باستخدام الـFileStream Class لتخزين الـXML Stream على XML File وفي الدالة الثانية يتم قراءة الـ XML File الذي تم تخزينه باستخدام الـFileStream ثم عمل Deserialize له لتحويله من XML Stream إلى Class Object مرة أخرى حيث يمكن استخدامه في البرنامج.

كما ويمكن تحويل المثال السابق لإرسال الـXML Stream عبر الـSocket وذلك باستبدال الـFileStream Class بالـNetworkStream حيث يتم كتابة الـStream على الـ NetworkStream ثم إرساله عبر الشبكة وكذلك عند الاستقبال حيث يتم استقبال الـXML Stream من الـNetworkStream ثم عمل Deserialize له مرة أخرى وكما يلي كمثال:

**An Example To Send A Serializable Class object Using The Socket:**

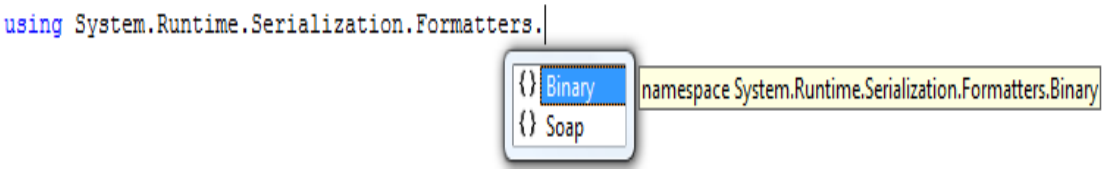
```
using System;
using System.Xml.Serialization;
using System.Xml;
using System.IO;
using System.Net.Sockets;

public class XMLSerializationExample
{
    static void Main()
    {
        CustomerDetails NewCustomer = new CustomerDetails();
        NewCustomer.CustomerID = 1;
        NewCustomer.AcountID = "800XXXXX";
        NewCustomer.CustomerName = "Fadi Abdelqader";
        NewCustomer.Address = "Amman/Jordan";
        NewCustomer.Phone = "+962XXXXXXX";

// Send The Customer Information Using The Socket
SerializeAndSend(NewCustomer,"127.0.0.1",5050);

// Receiving and Deserializing The Customer Stream (it Should execute
on other PC)
CustomerDetails MyCustomer = ReceivingAndDeserializing(5050);
Console.WriteLine(MyCustomer.AcountID + " - " +
MyCustomer.CustomerName);
    }

public static void SerializeAndSend(CustomerDetails CustData,string
IPAddress, int Port)
    {
   // Declare a Simple TCP Socket
   TcpClient TCP_Socket = new TcpClient(IPAddress, Port);
   // Bind The TCP Socket with the NetworkStream
   NetworkStream XMLStream = TCP_Socket.GetStream();
   XmlSerializer XMLSer = new XmlSerializer(typeof(CustomerDetails));
   XMLSer.Serialize(XMLStream, CustData);
   XMLStream.Close();
   TCP_Socket.Close();
    }
```

```
    public static CustomerDetails ReceivingAndDeserializing(int Port)
     {
    // Declare a Simple TCP Listener Socket
    TcpListener mytcpl = new TcpListener(Port);
    // Start The Listening
    mytcpl.Start();
    // Accept The Incoming Sending Request
    Socket mysocket = mytcpl.AcceptSocket();

    // Bind The Socket With The NetworkStream
    NetworkStream XMLNetworkStream = new NetworkStream(mysocket);

    XmlSerializer ser = new XmlSerializer(typeof(CustomerDetails));
    CustomerDetails CustData =
    (CustomerDetails)ser.Deserialize(XMLNetworkStream);
    XMLNetworkStream.Close();
    mytcpl.Stop();
    return CustData;
     }
}
```

في حالة إرسال Serialized Object عبر الـSocket يفضل أن يتم تحويل الملف الذي يحتوي الـClass الرئيسي إلى DLL File وذلك بعمل New Class Library Project وإضافته فيه ثم عمل Build له وإضافة الـCompiled DLL File إلى References المشروع عند المرسل والمستقبل وسبب عمل ذلك هو أنه في حالة عمل Deserialize لـClass Object فإنه يجب أن يكون ذلك الـ Deserialize على نفس الـAssembly Information للـClass الذي تم عمل Serialize له وإلا فإنه عند عمل Deserialize فإنه سيرجع No assembly associated with Xml key Exception.

يعتبر الـXmlSerializer كـSimple XML Serialization Class حيث يمكن من خلاله عمل Serialization لبيانات بسيطة داخل Class ما مثل (integers and strings variables) وكما تم في المثال السابق لكن عند الحاجة إلى عمل Serialization لبيانات أكثر تعقيدا كوجود صورة Image Object مع الـClass أو Serializable Classes أخرى داخل الـSerializable Class الرئيسي فإنه يلزم استخدام إحدى الـFormatters التالية ضمن

System.Runtime.Serialization.Formatters Namespace




يجب التأكد من إضافة System.Runtime.Serialization.Formatters.Soap إلى References المشروع عند الحاجة لاستخدام الـSoap Protocol Formatter حيث لا يضاف بشكل افتراضي مع الـProject.

لا يختلف استخدام الـSoap Formatter عن استخدام الـBinary Formatter بشيء والاختلاف فقط سيكون في الـOutput لعملية الـSterilization حيث أنه في حالة استخدام SoapFormatter Class فإن الـOutput سيكون بشكل XML Data في حين استخدام الـBinaryFormatter Class فإن الـOutput له سيكون على شكل Binary Data لا يمكن قراءتها إلى بعد عمل Deserialize لها وكمثال على استخدام الـBinary Formatter لعمل Serialize لـClass أكثر تعقيدا وتخزينه على Binary File باستخدام الـ FileStream ثم قراءته وتحويله إلى Class مرة أخرى:

## An Example To Make More Complex Serializable Class In .NET:

```
using System;
using System.Drawing;

    [Serializable]
    public class AddressBook
    {
        public string ContractorName;
        public string Address;
        public string Phone;
        public Image ContractorImage;
        public ContractorDetails MoreDetails;
        public AddressBook(string Contractor_Name, string
Contractor_Address, string Contractor_Phone, Image Contractor_Image,
ContractorDetails More_Details)
        {
            ContractorName = Contractor_Name;
            Address = Contractor_Address;
            Phone = Contractor_Phone;
            ContractorImage = Contractor_Image;
            MoreDetails = More_Details;
        }
        public AddressBook(){}
    }

    [Serializable]
    public class ContractorDetails
    {
        public string Email;
        public string WebSite;
        public string Job;
        public DateTime AddedDate;

        public ContractorDetails(string Contractor_Email, string
Contractor_WebSite, string Contractor_Job)
        {
            Email = Contractor_Email;
            WebSite = Contractor_WebSite;
            Job = Contractor_Job;
            AddedDate = DateTime.Now;
        }
        public ContractorDetails(){}
    }
```

## An Example To Serialize The Above Complex Class object:

```
using System;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters;
using System.Runtime.Serialization.Formatters.Binary;
using System.IO;
using System.Drawing;

public class FormattersClassesExample
{
    static void Main()
    {

// Serializing And Saving The Address Book Information as Binary File
AddressBook MyAddBook = new AddressBook("Fadi", "Amman/Jordan",
"+962XXXXXX",Image.FromFile(@"C:\MyImage.JPG"), new
ContractorDetails("myemail@socketcoder.com","www.SocketCoder.Com","Sy
stems Engineer")
         );
  SerializingAndSaving(MyAddBook, @"C:\AddressBook.Dat");

  // Loading and Deserializing The AddressBook Binary File
  AddressBook MyAddedBook = Deserializing(@"C:\AddressBook.Dat");
  Console.WriteLine(MyAddedBook.ContractorName + " - " +
                                    MyAddedBook.MoreDetails.Email);
    }

public static void SerializingAndSaving(AddressBook BookData, string
BinarySaveFilePath)
    {
  FileStream BinaryFileStream = new FileStream(BinarySaveFilePath,
                                                    FileMode.Create);
      BinaryFormatter ser = new BinaryFormatter();
      ser.Serialize(BinaryFileStream, BookData);
      BinaryFileStream.Close();
    }

 public static AddressBook Deserializing(string BinaryLoadFilePath)
    {
  FileStream BinaryFileStream = new FileStream(BinaryLoadFilePath,
                                                  FileMode.Open);
  BinaryFormatter ser = new BinaryFormatter();
  AddressBook adr = (AddressBook)ser.Deserialize(BinaryFileStream);
   return adr;
    }
}
```

**الخلاصة**: بينا في هذا الفصل كيفية استخدام الـStreaming Classes وكذلك التعامل مع الـObjects كإرسالها عبر الشبكة أو تخزينها لاسترجاعها لاحقا باستخدام تقنيات الـSerializationو الـStreaming. سيتم الحديث في الفصل التالي معمارية بناء تطبيقات P2P و Client/Server بالدوت نيت.