La serializzazione di un oggetto in un file XML

Vedremo ora come avviene la serializzazione di un oggetto non in un file binario ma in un file XML. Ma prima un breve accenno di cosa è l'XML.

XML `e l'acronimo di

eXtensible Markup Language

ed è uno standard ormai acquisito che consente lo scambio di informazioni attraverso varie tipologie di applicazioni. Viene utilizzato per rappresentare dati strutturati di varia natura come, per esempio, records estratti da una base di dati, può essere utilizzato come file di configurazione di una applicazione o di un sito web.

XML consente la descrizione (*markup*) dei dati. I dati sono inclusi nei documenti XML come stringhe di testo racchiuse tra opportune etichette (*markup tag*) che ne descrivono struttura e tipo di contenuto.

Un documento **XML** è un documento di puro testo che può essere letto e modificato con un qualsiasi text editor.
Questo è un esempio generato in automatico da Visual Studio, il file non ha estensione **.xml** ma si chiama

App.config

```
<?xml version ="1.0" encoding = "utf-8" ?>
<configuration>
    <startup>
       <supportedRuntime version ="v4.0" sku =".NETFramework,Version=v4.6.1" />
    </startup>
</configuration>
```

ma nonostante non abbia il suffisso **.xml** è perfettamente compatibile con lo standard. Assomiglia molto all'HTML ma, al contrario, non esistono tag propri e questo significa che è possibile crearne sempre nuovi a discrezione del programmatore e dell'applicazione che deve usare le informazioni in esso contenute.

Nella pagina seguente un altro esempio di file XML editato con Visual Studio:

```
<?xml version="1.0" encoding="utf-8"?>
<Roots >
  <Roots.RootTypes>
    <RootType FullName="Windows.UI.Xaml.Application">
      <RootProperty Name="RequestedTheme" />
    </RootType>
    <RootType FullName="Windows.UI.Xaml.ResourceDictionary" />
    <RootType FullName="Windows.UI.Xaml.Controls.ControlTemplate">
      <RootProperty Name="TargetType" />
      <RootProperty Name="Template" />
    </RootType>
    <RootType FullName="Windows.UI.Xaml.DataTemplate">
      <RootProperty Name="Template" />
    </RootType>
    <RootType FullName="Windows.UI.Xaml.Controls.TextBlock">
      <RootProperty Name="Text" />
      <RootProperty Name="FontFamily" />
      <RootProperty Name="FontWeight" />
      <RootProperty Name="FontSize" />
      <RootProperty Name="Foreground" />
      <RootProperty Name="TextTrimming" />
      <RootProperty Name="TextWrapping" />
    </RootType>
    <RootType FullName="Windows.UI.Xaml.Data.Binding" />
    <RootType FullName="Windows.UI.Xaml.Controls.Border">
      <RootProperty Name="BorderBrush" />
      <RootProperty Name="BorderThickness" />
      <RootProperty Name="Child" />
      <RootProperty Name="Background" />
      <RootProperty Name="Padding" />
    </RootType>
    <RootType FullName="Windows.UI.Xaml.Controls.ContentPresenter" />
  </Roots.RootTypes>
</Roots>
```

Come si vede i tag sono differenti e sono stati generati da Visual Studio per le impostazioni di qualche applicazione grafica.

Pertanto, XML non è un linguaggio di presentazione dei dati come HTML.
XML definisce il significato (*contenuto*) dei dati, ma non lo stile di presentazione. Tale informazione può essere inclusa separatamente in un foglio di stile.
Fogli di stile scritti nel linguaggio

XSL

(**X**ML **S**tylesheet **L**anguage)

possono essere usati per tradurre dati XML in HTML.

Un documento XML `e un testo racchiuso fra tag di testo. I tag di testo sono racchiusi tra parentesi angolari, ad esempio:

```
<person>
   Alan Turing
</person>
```

Il documento contiene un unico elemento denominato `person`. Tale elemento è racchiuso fra il tag iniziale `<person>` e il tag finale `</person>`.
I tag sono una forma di markup del testo. Ciò che viene racchiuso tra i tag iniziale e finale è detto contenuto dell'elemento (nell'esempio, la stringa Alan Turing). Un elemento XML può contenere testo libero (detto *character data*) o altri elementi XML.
Un ultimo esempio:

```
<?xml version="1.0" encoding="utf-8"?>
<person>
      <name>
        <first>
          Alan
        </first>
        <last>
          Turing
        </last>
      </name>
      <profession>
          informatico
      </profession>
      <profession>
          matematico
      </profession>
      <profession>
          statistico
      </profession>
</person>
```

L'elemento `<person>` contiene un sottoelemento `<name>` e tre sottoelementi `<profession>`.
L'elemento `<name>`, a sua volta, contiene due sottoelementi nome proprio `<first>` e cognome `<last>`.
Come si evince utilizzare uno standard XML possiamo strutturarlo come meglio crediamo l'importante che il file sia '*well formatted*' e che sia aderente allo standard dettato dal **W3C**, cioè il

World Wide Web Consortium[1]

Ora passiamo al codice C# che ci consente la persistenza di un oggetto in formato XML.

[1] https://www.w3.org/

Questo è il codice:

```
class Program
    {
      static void Main(string[] args)
        {
           Studente Alunno = new Studente();
           Alunno.Nome = "Michele";
           Alunno.Cognome = "De Benedittis";
           Alunno.Classe = "4C";
           XmlSerializer x = new XmlSerializer(Alunno.GetType());
           TextWriter tw = new StreamWriter("AlunnoXml.xml");
           x.Serialize(tw, Alunno);
           tw.Close();
           Console.ReadKey();
        }
    }
```

e questo è il contenuto del file 'AlunnoXml.xml':

```
<?xml version="1.0" encoding="utf-8"?>
<Studente xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Nome>Michele</Nome>
  <Cognome>De Benedittis</Cognome>
  <Classe>4C</Classe>
</Studente>
```

Spiegazione codice C#:

`XmlSerializer x = new XmlSerializer(Alunno.GetType());`	Creo l'oggetto 'x' che mi permetterà la serializzazione in formato XML della classe che ha istanziato l'oggetto Alunno tramite il metodo **GetType()** che appunto ritorna il tipo dell'oggetto, cioè **Studente** come si vede nel file XML generato
`TextWriter tw = new StreamWriter("AlunnoXml.xml");`	La classe **TextWriter** consente di scrivere una sequenza di caratteri nel file attraverso un flusso di scrittura **StreamWriter**
`x.Serialize(tw, Alunno);`	Serializzo l'oggetto Alunno tramite il serializzatore 'x'
`tw.Close();`	Chiudo il flusso e quindi anche il file XML

Spiegazione codice XML:

`xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`	Viene utilizzato nelle istanze del documento XML per diversi attributi speciali definiti dalla Raccomandazione di schema XML
`xmlns:xsd="http://www.w3.org/2001/XMLSchema"`	Viene utilizzato negli schemi XML (XSD) per gli elementi, gli attributi e i tipi

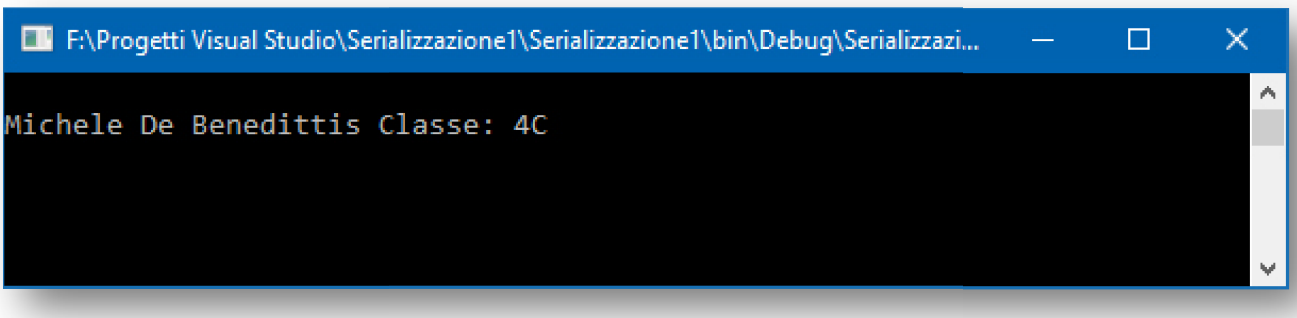
`<Nome>Michele</Nome>`
`<Cognome>De Benedittis</Cognome>`
`<Classe>4C</Classe>`
	Attributi dell'oggetto che sono stati serializzati

Ed ora passiamo alla lettura del file appena creato, questo è il codice:

```
XmlReader xread = XmlReader.Create("AlunnoXml.xml");
while(xread.Read())
  {
    if (xread.NodeType==XmlNodeType.Element)
      {
        if (xread.Name == "Nome")
           Console.Write(xread.ReadElementContentAsString());
        if (xread.Name == "Cognome")
           Console.Write(' '+xread.ReadElementContentAsString());
        if (xread.Name == "Classe")
           Console.WriteLine(" Classe: {0}", xread.ReadElementContentAsString());
      }
 }
```

Spiegazione codice C#:

`XmlReader xread = XmlReader.Create("AlunnoXml.xml");`	Creo un oggetto **XmlReader** per leggere un file XML richiamando il metodo **Create** a cui passo il nome (*path*) del file
`while(xread.Read())`	Con un ciclo while vado a leggere il contenuto
`if (xread.NodeType==XmlNodeType.Element)`	Controllo se ogni nodo (es. `<Nome>Michele</Nome>`) letto è un elemento

`if (xread.Name == "Nome")`
`  Console.Write(xread.ReadElementContentAsString());`
`if (xread.Name == "Cognome")`
`  Console.Write(' '+xread.ReadElementContentAsString());`
`if (xread.Name == "Classe")`
`  Console.WriteLine(" Classe: {0}",read.ReadElementContentAsString());`
	Se il nome dell'elemento è uguale al nome del campo serializzato, lo visualizzo tramite la chiamata al metodo `ReadElementContentAsString()` che è la stringa contenuta nel campo

In figura una prova di esecuzione: