HCI 프로그래밍

14. C# Serialization & XML

HCI
Human Computer Interaction

Overview

- Serialization
- Deserialization
- XML

Serialization

- **Serializaiton**(직렬화)란 객체 상태를 지속시키거나 전송할 수 있는 형식으로 변환하는 과정으로, **Serialization** 반대로 다시 객체로 변환하는 것이 **Deserialization** 임
- **Serialization**을 사용하는 이유
  - 객체의 상태를 저장소에 보존 했다가 나중에 똑같은 복사본을 다시 만들기 위하거나, 한 응용프로그램에서 다른 응용프로그램으로 객체를 전송하기 위해서 임

Serialization

닷넷에서 제공하는 **Serialization** 방식

- **Binary Serialization**
  - ✓ 형식(Type) 정확도를 유지하므로 응용프로그램의 여러 호출간에 객체 상태를 유지하는데 유용. 객체를 스트림, 디스크, 메모리, 네트워크 등으로 serialize 함
- **XML Serialization**
  - ✓ public 속성과 필드만 serialize하며 형식 정확도를 유지하지 않음.

Serialization

Binary Serialization

- Serialize하고자 하는 객체의 public, private 필드와 클래스 이름을 모두 바이트의 스트림(binary)로 변환하는 방식이며, deserialize 되면 원본 객체의 정확한 복제본이 생성
- Binary Serialization 방법
  - ✓ 기본 Serialization
  - ✓ 클래스의 일부 멤버를 Serialize하는 선택적 Serialization
  - ✓ ISerializable 인터페이스를 사용한 사용자 지정 Serialization

XML/SOAP Serialization

- XML Serialization은 객체의 public 필드와 속성 또는 메소드의 매개 변수와 반환 값을 XML 스트림으로 Serialize 하는 방식으로, XML Serialization을 사용하면 저장이나 전송을 위해 직렬형식으로 변환되는 public 속성 및 필드가 있는 강력한 형식 클래스 생성
- XML/SOAP Serialization 방법
  - ✓ XML Serialization
  - ✓ SOAP인코딩된 XML 스트림으로 Serialization

기본
Serialization

클래스를 Serializable 특성으로 표시

```
[Serializable]
public class MyObject {
    public int n1 = 0; public int n2 = 0; public string str = null;
}
```

클래스 객체를 Serialize하여 파일로 저장

```
MyObject obj = new MyObject();                    // 객체 생성
obj.n1 = 1; obj.n2 = 2; obj.str = “test”;
FileStream stream = new FileStream(“TestFile.bin”, FileMode.Create, FileAccess.Write,
FileShare.None);                                  // FileStream 생성
BinaryFormatter bf= new BinaryFormatter();        // BinaryFormatter 생성
bf.Serialze(stream, obj);                         // Formatter에게 stream과 객체를 주고 serialize
stream.Close();                                   // FileStream 닫기
```

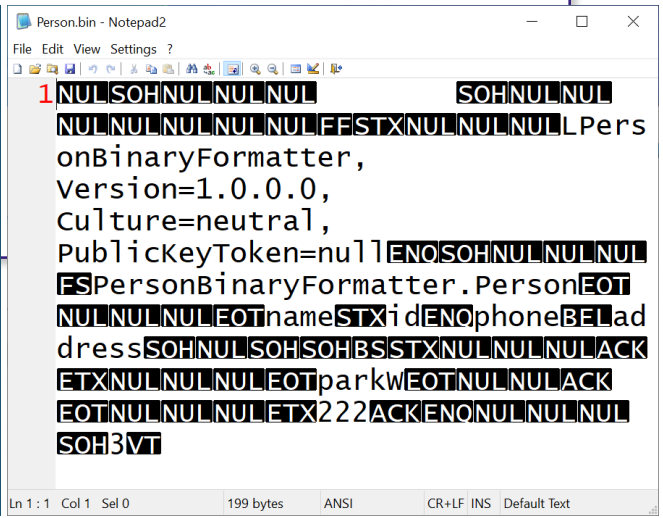
기본
Serialization

Serialize된 객체를 반대로 Deserialize로 복원

```
BinaryFormatter bf= new BinaryFormatter();              // BinaryFormatter 생성
FileStream stream = new FileStream("TestFile.bin", FileMode.Open, FileAccess.Read);
MyObject obj = (MyObject) bf.Deserialize(stream);       // Deserialize
stream.Close();                                         // FileStream 닫기
```

Serialization

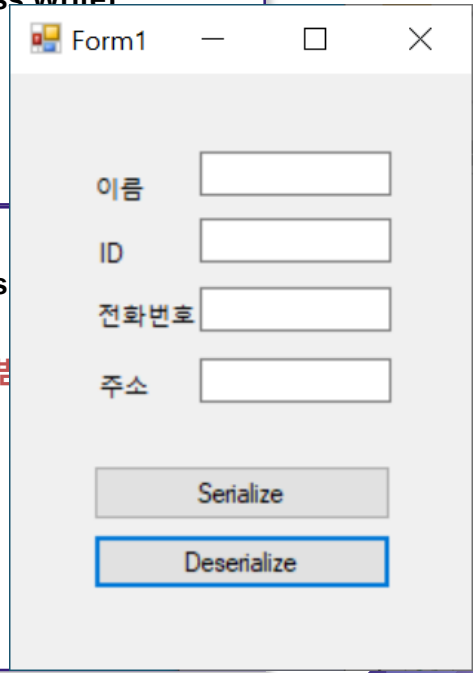
```
[Serializable]
public class Person: IComparable<Person>, IEquatable<Person> {
    protected string name;
    protected int id;
    protected string phone;
    protected string address;
    public string Name {
        get { return name; }
        set { name = value; }
    }
    // 중간 생략.....
}
```



Serialization

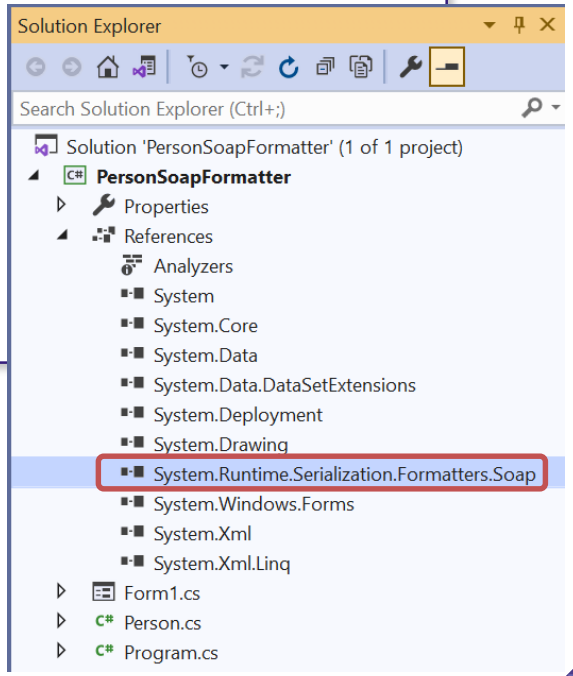
```
using System.Runtime.Serialization.Formatters.Binary;
private void button1_Click(object sender, EventArgs e) {
    Person p = new Person(textBox1.Text, int.Parse(textBox2.Text),
                          textBox3.Text, textBox4.Text);
    FileStream fs = new FileStream("Person.bin", FileMode.Create, FileAccess.Write);
    BinaryFormatter bf= new BinaryFormatter();
    bf.Serialze(fs, p); // Formatter에게 stream과 Person 객체를 주고 serialize
    fs.Close();
}
```

```
private void button2_Click(object sender, EventArgs e) {
    FileStream fs = new FileStream("Person.bin", FileMode.Open, FileAccess
    BinaryFormatter bf= new BinaryFormatter();
    Person p = (Person)bf.Deserialze(fs); // deserialize를 통해 Person 객체를 받
    fs.Close();
    textBox1.Text = p.Name.ToString();
    textBox2.Text = p.ID.ToString();
    textBox3.Text = p.Phone.ToString();
    textBox4.Text = p.Address.ToString();
}
```



Serialization

```
[Serializable]
public class Person: IComparable<Person>, IEquatable<Person> {
    protected string name;
    protected int id;
    protected string phone;
    protected string address;
    public string Name {
        get { return name; }
        set { name = value; }
    }
    // 중간 생략.....
}
```



Serialization

```
using System.Runtime.Serialization.Formatters.Soap;
// SoapFormatter
private void button1_Click(object sender, EventArgs e) {
    Person p = new Person(textBox1.Text, int.Parse(textBox2.Text),
                          textBox3.Text, textBox4.Text);
    FileStream fs = new FileStream("Person.xml", FileMode.Create,
                                   FileAccess.Write);
    SoapFormatter sf= new SoapFormatter();
    sf.Serialze(fs, p); // SoapFormatter에게 stream과 Person 객체를 주고 serialize
    fs.Close();
}
```

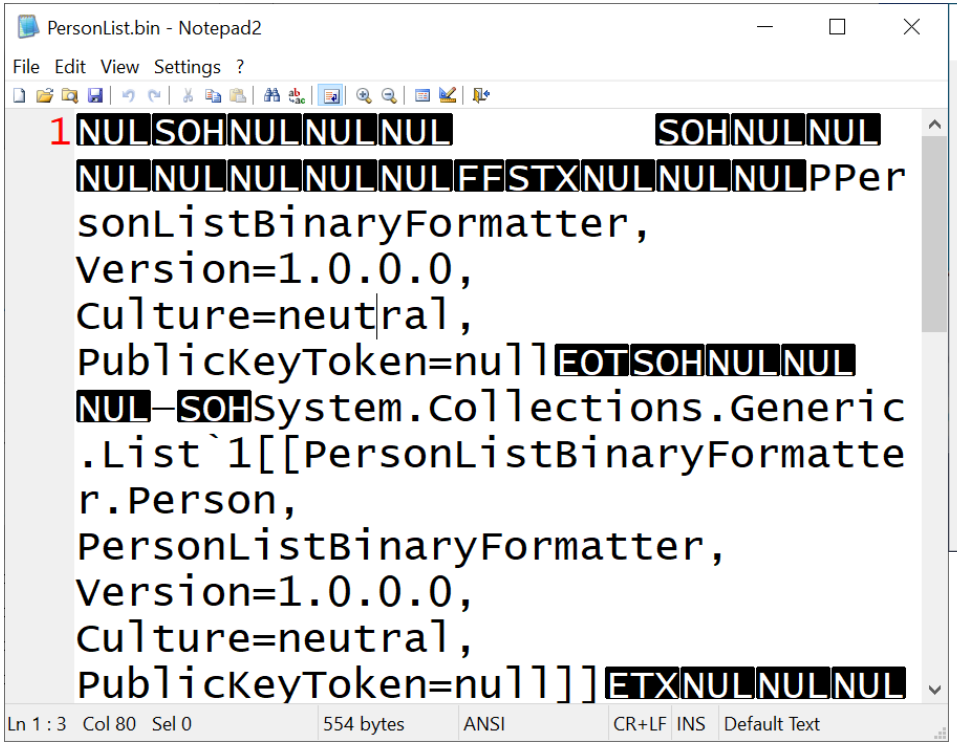
Serialization

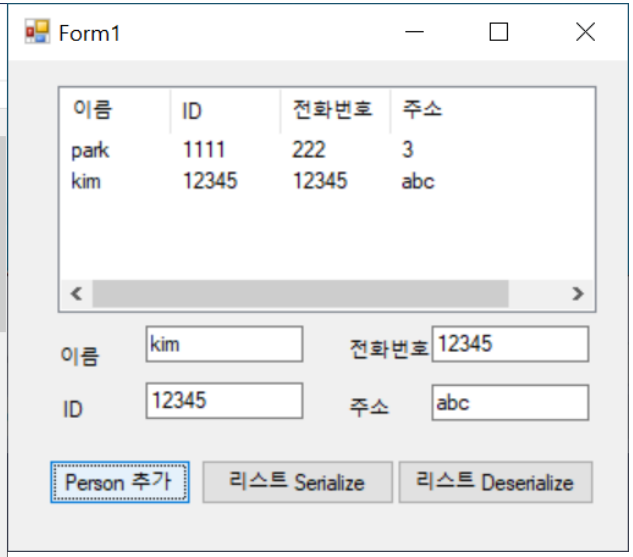

```
<SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:clr="http://schemas.microsoft.com/soap/encoding/clr/1.0" SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">

<SOAP-ENV:Body>

<a1:Person id="ref-1"
xmlns:a1="http://schemas.microsoft.com/clr/nsassem/PersonSoapFormatter/PersonSoapFormatter
%2C%20Version%3D1.0.0.0%2C%20Culture%3Dneutral%2C%20PublicKeyToken%3Dnull">

<name id="ref-3">박경신</name>

<id>1111</id>

<phone id="ref-4">222</phone>

<address id="ref-5">3</address>

</a1:Person>

</SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```


Serialization

```
using System.Runtime.Serialization.Formatters.Binary;
List<Person> pList = new List<Person>();
private void button1_Click(object sender, EventArgs e) {
    FileStream fs = new FileStream("PersonList.bin", FileMode.Create, FileAccess.Write);
    BinaryFormatter bf= new BinaryFormatter();
    bf.Serialze(fs, pList); // BinaryFormatter에게 stream과 Person 리스트 객체를 주고 serialize
    fs.Close();
}
private void button2_Click(object sender, EventArgs e) {
    FileStream fs = new FileStream("PersonList.bin", FileMode.Open, FileAccess.Read);
    BinaryFormatter bf= new BinaryFormatter();
    pList = (List<Person>)bf.Deserialze(fs); // Person 리스트 deserialize
    fs.Close();

    // 리스트뷰에 출력
    listView1.Items.Clear();
    foreach (Person p in pList) {
        listView1.Items.Add(p.ToListViewItem());
    }
}
```

Serialization





선택적 Serialization

- 클래스에서 **serialize**하지 않아야 할 필드를 **NonSerialized** 특성으로 표시함으로써 해당 변수가 **Serialize**되지 않게 함

```
[Serializable]
public class MyObject {
    public int n1 = 0;
    [NonSerialized]
    public int n2 = 0; // n2 멤버는 더 이상 serialize되지 않음
    public string str = null;
}
```

Serialization

```
<SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:clr="http://schemas.microsoft.com/soap/encoding/clr/1.0" SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<a1:Person id="ref-1"
xmlns:a1="http://schemas.microsoft.com/clr/nsassem/PersonSoapFormatter/PersonSoapFormatter
%2C%20Version%3D1.0.0.0%2C%20Culture%3Dneutral%2C%20PublicKeyToken%3Dnull">
<name id="ref-3">park</name>
<phone id="ref-4">222</phone>
<address id="ref-5">3</address>
</a1:Person>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

```
[Serializable]
public class Person {
        protected string name;
        [NonSerialized]
        protected int id;
        protected string phone;
        protected string address;
// 중간생략 … }
```

Custom Serialization

ISerializable 인터페이스를 사용한 사용자 지정 Serialization

```
// GetObjectData 메소드와 deserialize 될 때 사용되는 특수 생성자 구현이 포함되어야 함
[Serializable]
public class MyObject  : ISerializable {
    public int n1=0;     public int n2=0;     public string str=null;
    public MyObject() { }
    protected MyObject(SerializationInfo info, StreamingContext context) {
        n1 = info.GetInt32(“i”);
        n2 = info.GetInt32(“j”);
        str = info.GetString(“k”);
    } // deserialize시 필요
    [SecurityPermissionAttribute(SecurityAction.Demand, SerializationFormatter = true)]
    public virtual void GetObjectData(SerializationInfo info, StreamingContext context){
        info.AddValue(“i”, n1);
        info.AddValue(“j”, n2);
        info.AddValue(“k”, str);
    } // serialize시 필요
}
```

Serialization

```
public class Person: IComparable<Person>, IEquatable<Person>, ISerializable {
    protected string name;
    protected int id;
    protected string phone;
    protected string address;
    // 중간 생략…..
     #region ISerializable
     public Person(SerializationInfo info, StreamingContext context) {
          this.name = info.GetString("Name");
          this.id = info.GetInt32("ID");
          this.phone = info.GetString("Phone");
          this.address = info.GetString("Address");
     }
     public void GetObjectData(SerializationInfo info, StreamingContext context) {
          info.AddValue("Name", this.name);
          info.AddValue("ID", this.id);
          info.AddValue("Phone", this.phone);
          info.AddValue("Address", this.address);
     }
     #endregion
}
```

```
<?xml version="1.0" ?>
<Person xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<Name>park</Name>
<ID>1111</ID>
<Phone>222</Phone>
<Address>3</Address>
</Person>
```

Custom Serialization

사용자가 **Serialization**을 직접 제어하는 방식으로 **ISerializable** 인터페이스를 사용하는 방식 외에, **Serialization** 도중과 이후에 데이터를 수정하는 데 사용되는 메소드에 다음 특성을 적용

- OnDeserializedAttribute
- OnDeserializing Attribute
- OnSerializedAttribute
- OnSerializingAttribute

XML Serialization

XML Serialization 특징

- XML은 공개 표준이기 때문에 XML 스트림은 플랫폼에 관계없이 필요에 따라 모든 응용프로그램에서 처리될 수 있음
- XML serialization은 SOAP 사양과 일치하는 XML 스트림으로 객체를 serialize하는 데 사용할 수 있음
- 객체를 serialize하거나 deserialize 하기 위해서는 XmlSerializer 클래스를 사용
- Serialize된 데이터에는 데이터 자체와 클래스의 구조만 포함
- **Public 속성 및 필드만 serialize될 수 있음**
  – 만약 public 이 아닌 데이터를 serialize 해야 하는 경우 binary serialization을 사용할 것
- 클래스는 XmlSerializer에 의해 serialize될 기본 생성자가 있어야 함
- 메소드는 serialize 될 수 없음

XML Serialization

XmlSerializer를 사용하여 serialize/deserialize하는 방법

```
public class MyObject {
    public int n1=0;     private int n2=0;     public string str=null;
    public MyObject() { n1 = 1; n2 =2; str="XML serialization"; }
} // 결과물인 "obj.xml"에는 private 멤버와 메소드의 정보는 기록되지 않음
class Program {
    static void Main(string[] args) {
        MyObject obj = new MyObject();                          // MyObject 객체 생성
        // XmlSerializer 생성자에 serialize하고자 하는 객체의 타입을 전달하여 생성
        XmlSerializer xs = new XmlSerializer(typeof(MyObject));
        StreamWriter sw = new StreamWriter("obj.xml");          // stream writer 생성
        xs.Serialize(sw, obj);      // XmlSerializer에게 stream과 객체를 전달하여 serialize
        sw.Close();
        StreamReader sr = new StreamReader("obj.xml");          // stream reader 생성
        MyObject obj2 = (MyObject) xs.Deserialize(sr);          // deserialize
        Console.WriteLine("n1=" + obj2.n1 + " str=" + obj2.str);
        sr.Close();
    }
}
```

SOAP 인코딩된 XML Serialization

- SOAP 인코딩된 XML Serialize하기 위해서는,
  새로운 SoapReflectionImporter를 만들고 serialize된 클래스의
  형식으로 ImportTypeMapping 메소드를 호출하여
  XmlTypeMapping을 만든 뒤 XmlSerializer 생성자 매개 변수에 전달

```
class Program {
   static void Main(string[] args) {
      MyObject obj = new MyObject();                    // MyObject 객체 생성
      XmlTypeMapping xtm =
            new SoapReflectionImporter().ImportTypeMapping(typeof(MyObject));
      XmlSerializer xs = new XmlSerializer(xtm);
      StreamWriter sw = new StreamWriter("obj.xml"); // stream writer 생성
      xs.Serialize(sw, obj);  // XmlSerializer에게 stream과 객체를 전달하여 serialize
      sw.Close();
   }
}
```

Serialization

```
[XmlType(Namespace="urn:Person")]
public class Person: IComparable<Person>, IEquatable<Person> {
    protected string name;
    protected int id;
    protected string phone;
    protected string address;
    [XmlAttribute(AttributeName="Name")]
    public string Name {
        get { return name; }
        set { name = value; }
    }
    // 중간 생략.....
}
```

Serialization

```
using System.Xml;
Using System.Xml.Serialization;
// XmlSerializer
private void button1_Click(object sender, EventArgs e) {
    Person p = new Person(textBox1.Text, int.Parse(textBox2.Text),
                          textBox3.Text, textBox4.Text);
    FileStream fs = new FileStream("Person.xml", FileMode.Create, FileAccess.Write);
    XmlSerializer xs= new XmlSerializer(typeof(Person));
    xs.Serialze(fs, p); // XmlSerializer에게 stream과 Person 객체를 주고 serialize
    fs.Close();
}
```

Serialization

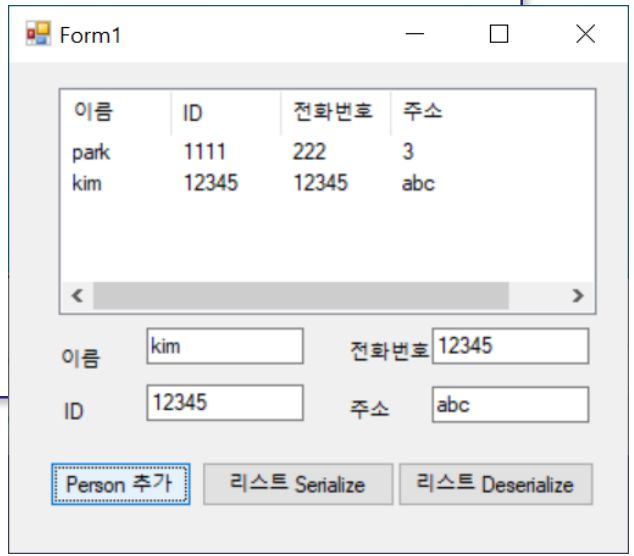
```
using System.Xml;
Using System.Xml.Serialization;
// Soap Encoding XmlSerializer
private void button1_Click(object sender, EventArgs e) {
    Person p = new Person(textBox1.Text, int.Parse(textBox2.Text),
                          textBox3.Text, textBox4.Text);
    FileStream fs = new FileStream("Person.xml", FileMode.Create, FileAccess.Write);
    XmlTypeMapping xtm= new
        SoapReflectionImporter().ImportTypeMapping(typeof(Person));
    XmlSerializer xs= new XmlSerializer(xtm);
    xs.Serialze(fs, p); // XmlSerializer에게 stream과 Person 객체를 주고 serialize
    fs.Close();
}
```

Serialization

```
using System.Xml;
Using System.Xml.Serialization;
private void button1_Click(object sender, EventArgs e) {
    FileStream fs = new FileStream("PersonList.xml", FileMode.Create, FileAccess.Write);
    XmlSerializer xs= new XmlSerializer(typeof(List<Person>));
    xs.Serialze(fs, pList); // XmlSerializer에게 stream과 Person 리스트 객체를 주고 serialize
    fs.Close();
}
```

Serialization

```
<?xml version="1.0" ?>
<ArrayOfPerson
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<Person Name="park" ID="1111" Phone="222" Address="3" />
<Person Name="kim" ID="12345" Phone="12345" Address="abc" />
</ArrayOfPerson>
```



Serialization

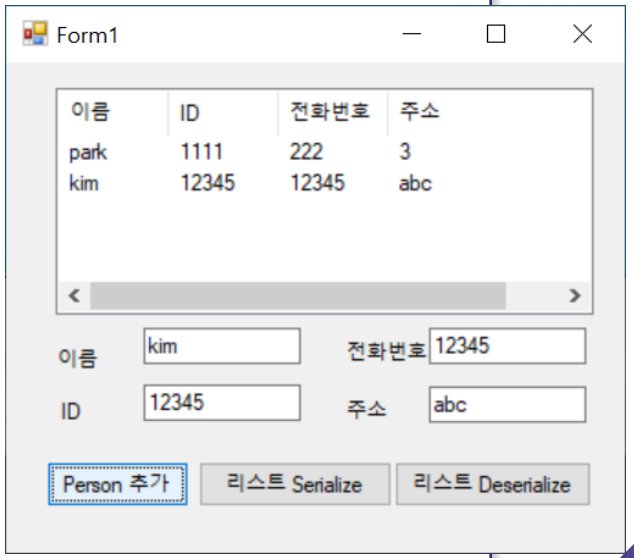
```
[XmlType(Namespace="urn:Person")]
public class Person: IComparable<Person>, IEquatable<Person> {
    protected string name;
    protected int id;
    protected string phone;
    protected string address;
    [XmlElement("Name")]
    public string Name {
        get { return name; }
        set { name = value; }
    }
    // 중간 생략.....
}
```

Serialization

```
using System.Xml;
Using System.Xml.Serialization;
private void button1_Click(object sender, EventArgs e) {
    FileStream fs = new FileStream("PersonList2.xml", FileMode.Create,
FileAccess.Write);
    XmlSerializer xs= new XmlSerializer(typeof(List<Person>));
    xs.Serialze(fs, pList); // XmlSerializer에게 stream과 Person 리스트 객체를 주고 serialize
    fs.Close();
}
```

Serialization

```
<?xml version="1.0" ?>
  <ArrayOfPerson
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Person>
<Name xmlns="urn:Person">park</Name>
<ID xmlns="urn:Person">1111</ID>
<Phone xmlns="urn:Person">222</Phone>
<Address xmlns="urn:Person">3</Address>
</Person>
  <Person>
<Name xmlns="urn:Person">kim</Name>
<ID xmlns="urn:Person">12345</ID>
<Phone xmlns="urn:Person">12345</Phone>
<Address xmlns="urn:Person">abc</Address>
</Person>
</ArrayOfPerson>
```



Serialize multiple objects into the same stream

- 1개의 스트림 안에서 여러 개의 클래스 객체를 serialize

```
private void button1_Click(object sender, EventArgs e) {
        Person person = new Person(textBox1.Text, int.Parse(textBox2.Text), textBox3.Text,
textBox4.Text);
        int x = int.Parse(comboBox1.Items[comboBox1.SelectedIndex].ToString());
        int y = int.Parse(comboBox2.Items[comboBox2.SelectedIndex].ToString());
        Point point = new Point(x,y);
        WeatherInfo weather = new WeatherInfo(double.Parse(textBox5.Text), double.Parse
(textBox6.Text), double.Parse(textBox7.Text));
        FileStream fs = new FileStream("MultipleObjects.bin", FileMode.Create,
FileAccess.Write);
        BinaryFormatter formatter = new BinaryFormatter();
        // serialize multiple objects into the stream
        formatter.Serialize(fs, person);
        formatter.Serialize(fs, point);
        formatter.Serialize(fs, weather);
        fs.Close();
 }
```

Serialize multiple objects into the same stream

- 1개의 스트림 안에서 여러 개의 클래스 객체를 deserialize

```
private void button2_Click(object sender, EventArgs e) {
FileStream fs = new FileStream("MultipleObjects.bin", FileMode.Open, FileAccess.Read);
        BinaryFormatter formatter = new BinaryFormatter();
        Person person = (Person)formatter.Deserialize(fs);
        Point point = (Point)formatter.Deserialize(fs);
        WeatherInfo weather = (WeatherInfo)formatter.Deserialize(fs);
        fs.Close();
        textBox1.Text = person.Name.ToString();
        textBox2.Text = person.ID.ToString();
        textBox3.Text = person.Phone.ToString();
        textBox4.Text = person.Address.ToString();
        comboBox1.Text = point.X.ToString();
        comboBox2.Text = point.Y.ToString();
        textBox5.Text = weather.Temperature.ToString();
        textBox6.Text = weather.Wind.ToString();
        textBox7.Text = weather.Humidity.ToString();
}
```

Serialization

Create a new XML file using XmlDocument

```
FileStream stream = File.Open(FileName, FileMode.OpenOrCreate);
XmlDocument xmlDoc = new XmlDocument();
XmlNode nodePersonList = xmlDoc.CreateNode("element", "PersonList", "");
foreach (Person p in pList)     {
    XmlNode nodePerson = xmlDoc.CreateNode("element", "Person", "");
      XmlNode nodeName = xmlDoc.CreateNode("element", "Name", "");
      nodeName.InnerText = Convert.ToString(p.Name);
      XmlNode nodeID = xmlDoc.CreateNode("element", "ID", "");
      nodeID.InnerText = Convert.ToString(p.ID);
      XmlNode nodePhone = xmlDoc.CreateNode("element", "Phone", "");
      nodePhone.InnerText = Convert.ToString(p.Phone);
      XmlNode nodeAddress = xmlDoc.CreateNode("element", "Address", "");
      nodeAddress.InnerText = Convert.ToString(p.Address);
      nodePerson.AppendChild(nodeName);
      nodePerson.AppendChild(nodeID);
      nodePerson.AppendChild(nodePhone);
      nodePerson.AppendChild(nodeAddress);
    nodePersonList.AppendChild(nodePerson);
}
xmlDoc.AppendChild(nodePersonList); xmlDoc.Save(stream); stream.Close();
```

Serialization

Load a XML file using XmlDocument

```
FileStream stream = File.Open(FileName, FileMode.OpenOrCreate, FileAccess.Read);
XmlDocument xmlDoc = new XmlDocument();
xmlDoc.Load(stream);
XmlNode nodePersonList;
nodePersonList = xmlDoc.FirstChild;
foreach (XmlNode nodePerson in nodePersonList)    {
        Person p = new Person();
        p.Name = nodePerson.FirstChild.InnerText;
        p.ID = Int32.Parse(nodePerson.FirstChild.NextSibling.InnerText);
        p.Phone = nodePerson.FirstChild.NextSibling.NextSibling.InnerText;
        p.Address = nodePerson.LastChild.InnerText;
        pList.Add(p);
}
stream.Close();
```

XML (eXtensible Markup Language)

XML 기본

- Meta Language – 원래 데이터에 대해 추가적인 정보를 표시
- Element와 Contents로 구성
  - ✓ Element는 문서의 구조를 정의하는 요소
    시작과 끝 태그(Tag)를 사용하여 표시
  - ✓ Contents는 실제 데이터
- XML 파서(Parser)가 필요함
  - ✓ www.w3.org/XML에서 정의한 구문규칙을 사용
  - ✓ 대표적인 파서로 ms XML Core Services(MSXML)
- XML 문서는 선택적으로 문서의 구조를 정의하는
  DTD (Document Type Definition) 또는
  스키마 (Schema)를 참조가능

XML 구성요소

- XML 문서의 선언
- Element – 마크업 태그와 그 안에 포함된 내용
- Root Element – 문서 내 모든 Element와 내용을 포함하고 있는 XML 문서의 요소
- Attribute – Element에 포함되어 추가적인 정보를 제공
- Entity – 텍스트, Binary, 비 ASCII 문자를 저장하는 데 사용
- Processing Instruction – 전체 문서나 문서의 일부를 처리하는 응용프로그램과 연결해주는 명령어
- Comment (주석) – XML 프로세서가 해석하지 않는 설명문
- CDATA Section – 모든 문자를 마크업이나 태그로 인식하지 않고 일반 문자로 인식할 수 있는 표기법. 즉, 특수한 문자를 일반 텍스트로 인식하도록 하는 표기법
- DTD 선언

XML 문서의 선언

Parser에게 현재 **XML** 문서가 어떤 문서인지 알려 줄 수 있도록 몇 가지 정보를 제공

- <?xml version="1.0" encoding="EUC-KR"?>
- XML 문서의 선언은 반드시 파일의 맨 처음에 위치
- Version 속성 필수
- Encoding 속성 선택
  - ✓ 한글 encoding은 "EUC-KR"을 주로 사용
- Standalone 속성 선택
  - ✓ "yes" – 다른 파일에 의존하고 있지 않음
    외부 파일의 요소, 그림, 개체 등을 참조하고 있지 않음
  - ✓ "no" – 다른 파일에 의존하고 있을 수 있음 (default)
- Version, encoding, standalone 순서 준수

XML Element

Root Element

- 모든 Element를 포함하는 Element
- 반드시 1개 존재함

Element content

- 시작 태그와 끝 태그 사이의 내용
- PCDATA (Parsed Character DATA)
- 다른 Element

XML Element

Element 작성 규칙

- 모든 시작 태그는 반드시 끝 태그와 짝을 이루어야 하며, 겹쳐 쓸 수 없음
- XML Document는 반드시 하나의 Root Element를 가져야 함
- Element 이름은 반드시 XML의 이름 짓는 규칙에 따라야 함
  문자로 시작 첫 문자 뒤에는 “_”, “.”, 숫자 사용 가능
- XML은 대소문자를 구별함

XML Element

```
public class Person {
  [XmlElement("Name")]
  public string Name   {
    get;   set;
  }
  [XmlElement(“ID")]
  public int ID   {
    get;      set;
  } ….. }
```

```
<Person>
  <Name>Park </Name>
  <ID>1111</ID>
  <Phone> 222</Phone>
  <Address> 3</Address>
</Person>
```

XML Attribute

- **Attribute**는 하나의 요소와 결합된 이름/값의 쌍
- 시작 태그에 추가
- 속성은 반드시 값을 가짐

```
<movie genre="SF">
        <name>2001: Space Odyssey</name>
        <director>Standley Kubrick</director>
        <year>1968</year>
</movie>
```

XML Attribute

```
public class Person {
  [XmlAttribute(AttributeName="Name")]
  public string Name   {
    get;          set;
  }
  [XmlAttribute(AttributeName="ID")]
  public int ID   {
    get;     set;
  } ….. }
```

```
<Person Name="Park" ID="1111" Phone="222" Address="3"/>
```

XML Element & Attr

```
[XmlRoot(“Person”)]
public class Person {
  [XmlElement(“Name”)]
   public string Name {
      get;   set;
   }
  [XmlElement(“ID")]
   public int ID   {
      get;      set;
   }
  [XmlElement(“Phone")]
   public Phone phone = new Phone();
  [XmlElement(“Address")]
   public string Address   {
      get;      set;
  } ….. }
```

```
public class Phone {
  [XmlElement(“WorkPhone”)]
   public string WorkPhone {
      get;   set;
   }
  [XmlElement(“HomePhone")]
   public string HomePhone   {
      get;      set;
   }
  [XmlElement(“CellPhone")]
   public string CellPhone   {
      get;      set;
   }
 }
```

XML Element & Attr

```
<Person>
  <Name>Park </Name>
  <ID>1111</ID>
  <Phone>
        <WorkPhone>123-457-7890</WorkPhone>
        <HomePhone>02</HomePhone>
        <CellPhone>010</CellPhone>
  </Phone>
  <Address>3</Address>
</Person>
```

Blank Element

- **Element**의 내용이 없는 경우 끝 태그를 생략하고 빈 요소를 사용할 수 있음
- 빈 요소는 어떠한 내용도 포함할 수 없으며, 오직 속성 만을 포함

  - <Person/> (O)
  - <Person /> (O)
  - <Person/ > (X)
  - <Person / > (X)
  - <Person Name="Park" ID="1111" Phone="222" Address="3"/> (O)

XML Comment

- **<!- - 와 - ->** 사이에 정의되고
  주석처리 **(Comment)**로 사용
- **<!-- <Person Name="Park" ID="1111"**
  Phone="222" Address="3"/>-->
- 태그 내에 주석을 쓸 수 없음
- 주석 내에 "--"를 쓸 수 없음

Processing Inst

처리명령 (Processing Instruction)

- <? 와 ?> 사이에 표현되고 특정 응용프로그램에 대해 처리할 정보나 명령을 가리키는 역할
- <?xml:stylesheet type="text/xsl" href="메모.xsl"?>
  - ✓ XML 의 스타일 쉬트를 지정하고, 타입은 test/xsl이며, 소스는 메모.xsl에 있다는 의미
- Element나 Attribute 이외에 Application에 전달하고자 하는 내용을 사용
- XML 문서의 주석문은 프로세서에 전달되지 않기 때문에 JavaScript 코드 등을 삽입
- Nameprocessor – Application의 이름 (PITarget)
  - ✓ "SELECT * FROM customer" : 실제 PI

PCDATA

- **Element**에 포함된 문자열
- 유효하지 않은 **PCDATA** 문자들
  - “<“, “&” 등
  - **해결 방법**
    - ✓ Escape 문자
    - ✓ CDATA section
- **Entity Reference**
  - & 문자 - &
  - < 문자 - <
  - > 문자 - >
  - ‘ 문자 - '
  - “ 문자 - "

Entity

- **Entity**의 단위는 글자에서부터 문서 전체
- 외부 참조
  - 포함 시킬 내용이 외부에 존재하는 것을 참조하는 형태
- 내부 참조
  - DTD 내에 정의
- **Entity**에 대한 포인터는 **Entity Reference**
  - & 문자 - &
  - < 문자 - <
  - > 문자 - >
  - ‘ 문자 - '
  - “ 문자 - "

CDATA Section

- **Parser**나 브라우저에서 유효성 검사를 하지 않는 문자열
- **CDATA** 섹션 **(Character DATA Section)**
  - <![CDATA[와]]> 사이에 정의되고 그 안의 내용을 문자열로 사용
  - <![CDATA[<Person Name="Park" ID="1111" Phone="222" Address="3"/>]]>

DTD 선언

- **Document Type Definition**
- **XML** 문서의 구조를 명시적으로 정의한 것
- **Valid Document** (유효한 문서)
  - Well-formed document이면서 DTD에서 정의된 규칙을 따르는 XML 문서

Namespace

- 여러 개의 문서를 병합할 때 같은 이름의 **Element** 나 **Attribute**을 구별하기 위한 방법
  - Schema의 재사용
  - 여러 문서와의 통합 시 이름 충돌 문제 해결
- **Namespace** 선언
  - <접두어: 태그이름 xmlns:접두어="URL">
  - Name은 URI에 의해 구별
  - 선언된 태그와 그 자식 태그에서 접두어를 사용할 수 있음
  - 하나의 태그에서 여러 개의 namespace를 선언할 수 있음
  - 주로 Root Element에서 선언

Namespace

기본 Namespace 선언

- <태그이름 xmlns="URL">
- 기본 Namespace를 선언하면 접두어를 붙이지 않아도
  해당 Element와 모든 자식 Element에 Namespace가 적용