Serialization

Scott Seely
http://www.pluralsight.com/

Outline

- Introduction to serialization
- Default behaviors for serialization
- Taking control of format
- JSON
- Binary serialization *(XML doesn't have to be bloated!)*

Introduction to Serialization

- **Serialization: The writing an object to some stream**
- **Deserialization: The reading of some stream into an object**
- **Primary use: Exchange data in some neutral mechanism**
- **Artifacts:**
  - Description Language: Describes the format of the data in the stream
  - Description Language Generator: Reads source code and produces the description.
  - Description Language Reader: Reads the description language, produces code.

Artifacts in WCF

- Description Language: XML Schema Documents (XSD)
- Description Language Generator: svcutil.exe, ServiceMetadataBehavior
- Description Language Consumer: svcutil.exe, Add Service Reference, MetadataResolver

Serializer Infrastructure

- **System.Xml.Serialization.XmlSerializer: Close as Microsoft gets to implementing full XSD (Full XSD support: XMLSpy from Altova.com)**
  - Use cases: XML with attributes, unordered XML elements (xsd:group), consuming formats you don't control
  - Benefits: Very flexible, allows dev to really shape XML
  - Cons: Too many decisions to make when defining serialization structure. Only writes to XML 1.0.
- **System.Runtime.Serialization.DataContractSerializer: Default WCF serializer**
  - Use cases: Read/write ISerializable, [Serializable], IXmlSerializable data
  - Benefits: High performance, can write to many formats: XML 1.0, MTOM, Binary XML. Most decisions made for you, only emits XMLNS and elements.
  - Cons: Few decisions you can make, doesn't work with XML Attributes.

Why is element only OK?

```
public class LineItem
{
  public int Line { get; set; }
  public int ItemId { get; set; }
  public double Price { get; set; }
  public int Quantity { get; set; }
  public int PurchaseOrderId { get; set; }
}
```

minOccurs = 0, maxOccurs = 1

```
<xs:complexType name="LineItem">
  <xs:sequence>
    <xs:element minOccurs="0" name="ItemId" type="xs:int" />
    <xs:element minOccurs="0" name="Line" type="xs:int" />
    <xs:element minOccurs="0" name="Price" type="xs:double" />
    <xs:element minOccurs="0" name="PurchaseOrderId" type="xs:int" />
    <xs:element minOccurs="0" name="Quantity" type="xs:int" />
  </xs:sequence>
</xs:complexType>
```

Customization: Namespace level

- **[assembly: ContractNamespace]: Define the default XML Namespace for all types in a CLR Namespace**

```
[assembly: ContractNamespace("http://www.pluralsight.com/service/v1.0.0",
  ClrNamespace = "DefaultSerialization.Contract")]
```

- **[DataContract]: Set the name and XML Namespace for the type**

```
[DataContract(Name = "lineItem",
  Namespace = "http://www.pluralsight.com/service/v1.0.0")]
public class LineItem
```

- **Recommendation: Use ContractNamespace, do not use the [DataContract] Namespace property.**

Customization: Data Elements

- **[IgnoreDataMember]: For default serialization, skips members (property or field)**

```
[IgnoreDataMember]
public int ItemId { get; set; }

[IgnoreDataMember]
public int PurchaseOrderId;
```

EnumMember

```
[DataContract]
 public enum BlockColors
{
  [EnumMember(Value = "green")]
  Green,

  [EnumMember(Value = "red")]
  Red,

  [EnumMember(Value = "blue")]
  Blue
}
```

```
<xs:simpleType name="BlockColors">
  <xs:restriction base="xs:string">
    <xs:enumeration value="green" />
    <xs:enumeration value="red" />
    <xs:enumeration value="blue" />
  </xs:restriction>
</xs:simpleType>
```

DataMember

```
[DataMember(
    IsRequired = false,
    EmitDefaultValue = false,
    Name = "line",
    Order = 2)]
public int Line { get; set; }
```

Use to handle serialization behavior.

Customize names of List items, Dictionary items

```
[CollectionDataContract(ItemName = "lineItem")]
public class LineItemCollection : List<LineItem>{}

[DataMember]
public LineItemCollection LineItems { get; private set; }

[CollectionDataContract(ItemName = "key", ValueName = "value")]
public class MyDictionary : Dictionary<string, LineItem>{}
```

Just an annotation on a type. Useful for fixing names.

Object Oriented, Polymorphism

- **Common use case: Have a base type and many derived types**
- **Want to use polymorphism to handle specializations over one endpoint**
- **[KnownType]:**
  - Type .ctor: Fixed set of types
  - String .ctor: Many types being provided via function

KnownType

```
[KnownType(typeof(DerivedTypeA))]
```

Works great!!! until someone introduces DerivedTypeB ☹

```
[KnownType("GetDerivedTypes")]

static public IEnumerable<Type> GetDerivedTypes()
{
  return from type in typeof (BaseType).Assembly.GetTypes()
         where typeof (BaseType).IsAssignableFrom(type)
         select type;
}
```

Serialization Options

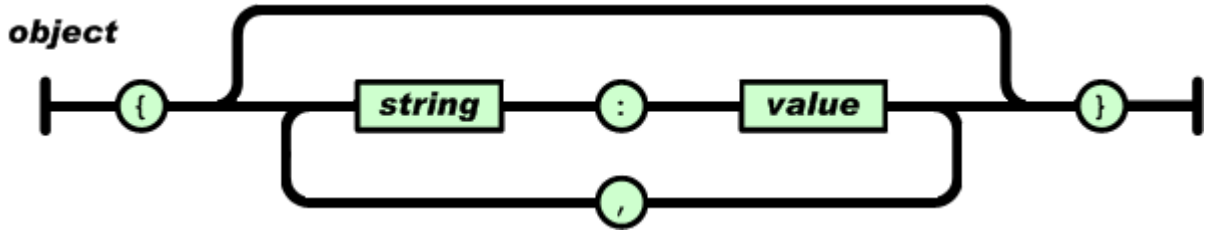
- **XML 1.0 (text)**
  - Use for interop
  - WCF Bindings: HTTP based, default for all except BasicHttpBinding
- **XML 1.0 + MTOM (Message Transmission Optimization Mechanism)**
  - Use for interop (limited), large binary elements
  - Binary elements: file and image data
  - WCF Bindings: HTTP based, default for BasicHttpBinding
- **Binary**
  - Use for WCF to WCF communication
  - Everything is smaller

XML Binary Serialization

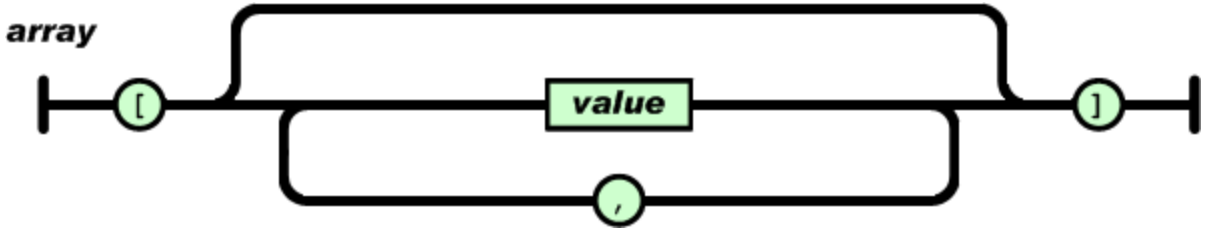
- **What makes it all so small?**
- **XmlDictionary: Map of XmlDictionaryStrings**
- **XmlBinaryWriterSession: Session used to write strings to a stream.**
  - Store key to replace XML String
  - “Name” becomes 5
  - “http://www.pluralsight.com/” becomes 7
- **XmlBinaryReaderSession uses same numbers and values to rehydrate messages**
  - WCF exchanges additions to dictionary as part of binary protocol

JavaScript Object Notation (JSON)

- **Builds on two structures:**
  - Name/value pairs
  - Ordered lists of values, *aka arrays*
- **Object:**

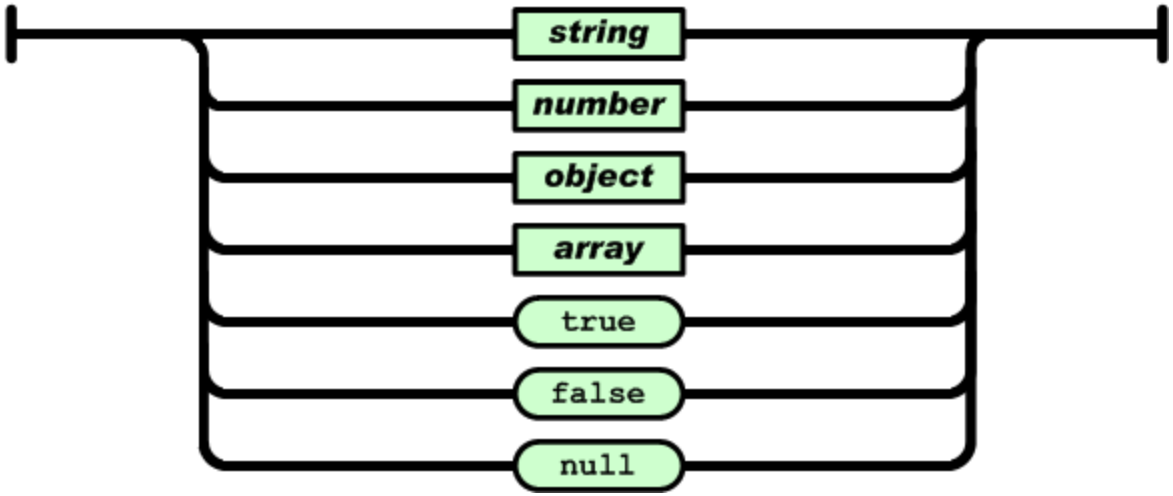


- **Array:**



JavaScript Object Notation (JSON)

- **Value:** *value*



- **String: C-style strings.**
  - Escape character: \
  - \b, \f, \r, \n, \t, \\, \/, \", \uXXXX all have usual meanings
- **Numbers are signed, support digits [0 .. 9], decimal, and exponent**
  - 9, -10, 3.14, 6.022E+23

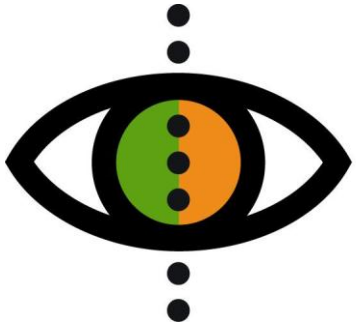
DataContractJsonSerializer

- WebHttpBinding, WebScriptHttpBinding use JSON as an wire format
- Also useful in Silverlight 3.0 and later
- Used by <ScriptManager> in ASP.NET pages
- Preferred interop format for JavaScript, Ruby, Python, and PHP

Summary

- Serialization provides a number of mechanisms to go from Data→Object, Object→Data
- WCF supports 3 XML formats: XML Text, MTOM, and Binary
- XML Text and MTOM available on HTTP, interoperable bindings
- Binary available on .NET bindings
- JSON allows for interop with ASP.NET, JavaScript, and other Web languages (Ruby, Python, PHP)

For more in-depth online developer training visit



pluralsight

see what you can learn

on-demand content from authors you trust