Attribute

Überblick

- Attribute ermöglichen das Hinzufügen von benutzerdefinierbaren Informationen zur Klasse
- analog der Sprache, Bsp.: ein *schönes* Auto
- können zur Laufzeit abgefragt und ausgewertet werden
- Attribute werden bei der Kompilierung in der Assembly durch Metadaten dargestellt
- Verarbeitungsschritte
  - Definition
  - Verwendung (Attribuierung, deklarative Progr.)
  - Auswertung
- Definition von Attributen: Attribute sind Klassen, die von System.Attribute abgeleitet werden
- Namenskonvention: BezeichnerAttribute wobei Bezeichner beliebiger Name ist

Attribute

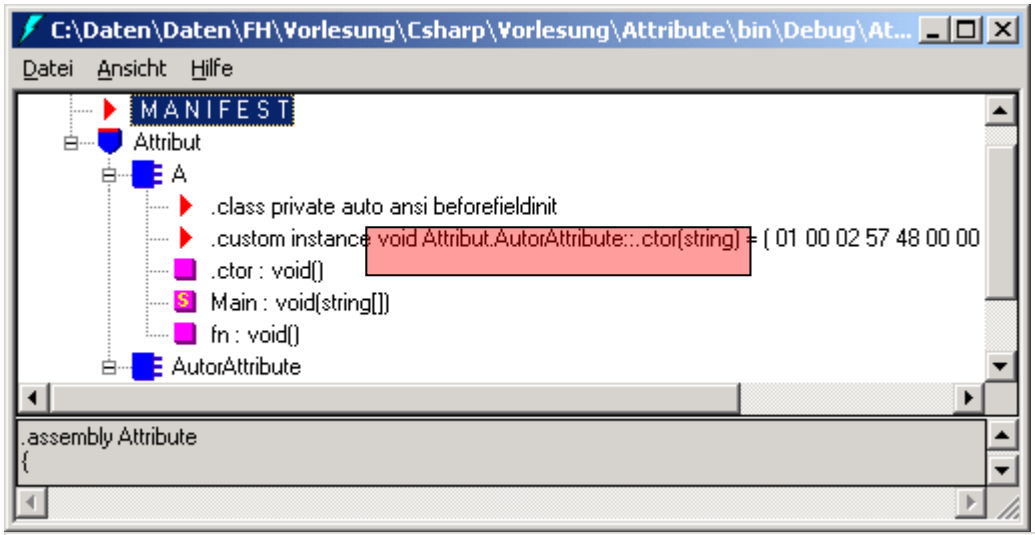
Verwendung

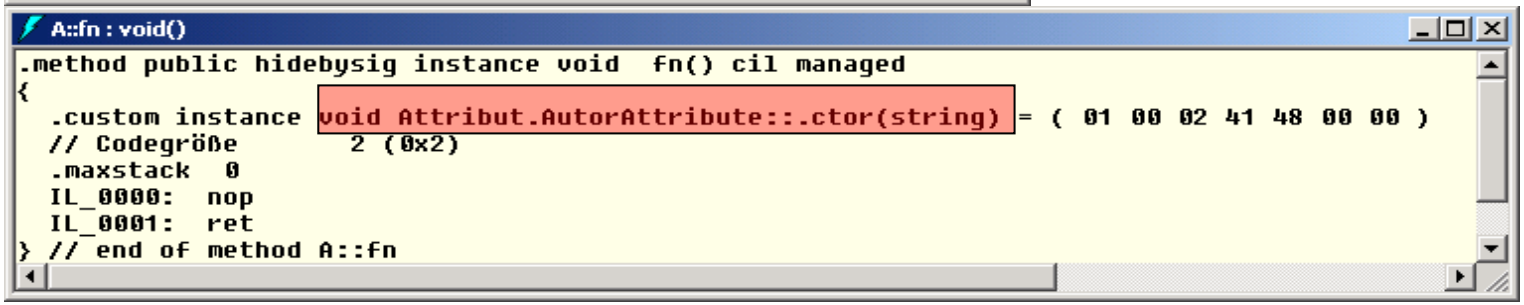
- Attribute werden vor dem jeweiligen Element platziert, in eckigen Klammern
  - Bsp.: [Obsolete()] Element // d.h. Element hat // Attribut ObsoleteAttribute
  - Postfix „Attribute“ des Namens kann entfallen (automatisch vom Compiler ergänzt)
- Syntax der Attributverwendung gleicht Konstruktoraufruf

```
class AutorAttribute : Attribute {
   public AutorAttribute(string name){. . .}
}
[Autor("WH")]                          // Klassenattribut
class A{
   [AutorAttribute("AH")]
   public void fn(){. . .}             // Methodenattribut
}
```

Attribute

Attributinformation in den Metadaten (ildasm)





Attributinformation wird mit Reflection ausgewertet

Attribute

Auswertung der Attribute

- Attributinformation wird mit Reflection ausgewertet (zusätzlicher Code ist somit zwingend erforderlich)

```
static void Main(string[] args){
    Type t = typeof(A);
    foreach(Attribute attr in t.GetCustomAttributes(true))
    {
        AutorAttribute auth = attr as AutorAttribute;
        if (auth!=null)
        Console.WriteLine("Typ: {0} {1}",t.Name,auth._name);
    }
}                    // Ausgabe: Typ A WH
```

- GetCustomAttributes(true) liefert Attribute nur des Typs. (nicht alle in der Klasse gesetzten Attribute) Dies sind Instanzen der Attributklasse

Attribute

Konstruktoraufruf

- Attribut muss mindestens einen Konstruktor haben
- Attribute werden nur instanziiert (Aufruf des Konstruktors), falls sie ausdrücklich gesucht (Reflection) werden d.h. bei der Verwendung

Ziel des Attributs (AttributeUsage)

- teilt dem Compiler mit, wo Attribut verwendbar ist
- AttributeUsage ist wiederum Attribut das auf Attribute angewendet wird (Meta-Attribut)
- Attributziele: assembly, event, field, method, module, param, property, return, type

```
[AttributeUsage(AttributeTargets.Class|AttributeTargets.Field
,Inherited = false,AllowMultiple = false)]
   public class AutorAttribute : Attribute{. . .}
```

Attribute

Parameter der Attributverwendung

- positionelle Parameter
  - Bsp.: [Obsolete("Text")]
  - werden durch Konstruktor übergeben
- benannte Parameter
  - werden nach allen positionellen Parametern aufgeführt
  - der Name des Parameters muss angegeben werden
    Bsp.: [WebService(Namespace="http...")]
    [Autor(ID=4711,"Hans")] <- falsch !!
  - können als Eigenschaft (Property) implementiert werden
  - tatsächlich jedoch kein Konstruktoraufruf mit benannten Parametern

Attribute

Klassifizierung der Attribute

- benutzerdefinierte
  - werden von System.Attribute abgeleitet
  - können mit der Methode GetCustomAttributes(true); der Klasse Type ausgewertet werden

```
Type t = ...; // Attributinformation zum Typ
t.GetCustomAttributes(true);
oder:
MemberInfo info = ...; // zum Member
Info.GetCustomAttributes(true);
```

- intrinsische (vordefinierte)
  - sind Bestandteil der CLR
    Bsp.: Serialisierung über Attribut [Serializable]
  - Auswertung über Reflection ebenfalls möglich

Attribute

Vordefinierte Attribute (Beispiele)

- [AttributeUsageAttribute]
  definiert Verwendungsmöglichkeiten eines Attributs
- [Serializable]
  ermöglicht die Serialisierung einer Instanz, d. h. Instanzdaten werden in einen „flachen Strom“ geschrieben
- [Obsolete]
  Compiler erzeugt Warnung, dass Sprachelement veraltet ist
- [WebMethod]
  Deklariert eine Methode als XML-Web-Service
- [Conditional(„debug“)]
  Ist auf void Methoden anwendbar die ausgeführt werden falls Bedingung („debug“) über #define gesetzt

Attribute

Attribute und Vererbung

- Attribute können vererbt werden
- Vererbung wird über das Metaattribut AttributeUsage gesteuert

```
[AttributeUsage(AttributeTargets.Class|...
,Inherited = true,AllowMultiple = true)]
```

  - mit Vererbung: Inherited = true
  - ohne Vererbung: Inherited = false
- Falls Attribut mehrfach verwendet werden kann (AllowMultiple = true) kann das Attribut sowohl in der Basisklasse als auch in allen abgeleiteten Klassen gesetzt werden (Bsp.: Autor)

Attribute

Serialisierung [Serializable]

Ausgangspunkt: Unterbrechung Arbeit mit unterschiedlichen Instanzen (Pause) mit
Ziel: Fortsetzung zu beliebigem späteren Zeitpunkt

Problem: Speicherung der Statusinformation
Beenden des Programms => Freigabe der Instanzen

Fragen: Welche Daten werden gespeichert?
Wie erfolgt Speicherung?
Welche Formate werden verwendet?
Wie erfolgt Wiederherstellung der Instanzen?
Sind Kontextinformationen erforderlich?

Attribute

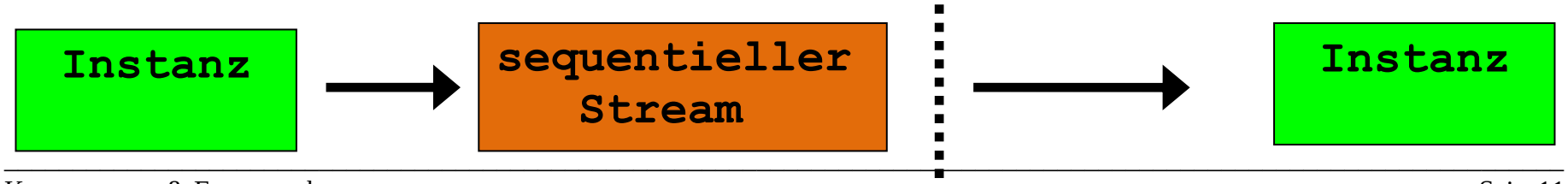
Definition

Serialisierung ist das Umwandeln einer Instanz einer Klasse in einen sequentiellen Stream.

Die Umwandlung dieses Streams in eine Instanz der entsprechenden Klasse nennt man Deserialisierung

Das Attribut [Serializable]

- **kennzeichnet Klassen, die serialisiert werden sollen**
- **ermöglicht automatische Serialisierung**
- **die serialisierte Klasse, d. h. der sequentielle Stream kann an beliebige Stellen versendet bzw. als Datei gespeichert werden.**



Attribute

Beisp.: Serialisierung – SOAP - Deserialisierung

```
[Serializable]
class A                                  // Serialisierung
{   public A(int j){i=j;}
    public int i;
    static void Main(string[] args){
        A a = new A(5);
        FileStream myfile;
          myfile = new FileStream(@"C:\temp\test.dat",
                                  FileMode.Create);
        SoapFormatter soapformat = new SoapFormatter();
        //BinaryFormatter binformat = new BinaryFormatter();
        soapformat.Serialize(myfile,a);
        myfile.Close();
        . . .}
}
```

Attribute

Sequentieller SOAP-Stream

```
<SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/2001/
                   XMLSchema-instance". . .
 <SOAP-ENV:Body>
  <a1:A id="ref-1" xmlns:a1="http://schemas.microsoft.com/
                   clr/nsassem/Serialisierung/
                   Attribute%2C%20Version%3D1.0.39.210C%20
                   Culture%3Dneutral%2C%20
                   PublicKeyToken%3Dnull">
      <i>5</i>
  </a1:A>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Attribute

Deserialisierung

```
[Serializable]
class A                                    // Deserialisierung
{       ...
        myfile = new FileStream(@"C:test.dat",FileMode.Open);
        object obj = soapformat.Deserialize(myfile);
         A aneu = (A)obj;
         Console.WriteLine(aneu.i);
        myfile.Close();
    }
}
```

Attribute

Formatter-Klassen

- BinaryFormatter
  - kompakte binäre Codierung

- SOAP-Formatter
  - basiert auf SOAP-Protokoll
  - gut geeignet für Übertragung über Firewalls bzw. zu unterschiedlichen Systemen

- XML-Formatter
  - basiert auf XML

- Benutzerdefinierte Formatter

Attribute

Interface ISerializable

- Ermöglicht einem Objekt, die eigene Serialisierung und Deserialisierung zu überwachen.

- Methode: GetObjectData (zur Serialisierung)

```
[Serializable()]
class A : ISerializable
 {
   private A(SerializationInfo info, StreamingContext context)
             :this()        // Deserialisierung
             {i = info.GetInt32("i");}
    public void GetObjectData(SerializationInfo info,
                              StreamingContext context)
            {info.AddValue("i",i);}
 }
```

Attribute

Vererbung Aggregation

- Attribut Serializable wird nicht vererbt
  => abgeleitete Klassen mit Serializable kennzeichnen

- Alle Basisklassen serialisierbarer abgeleiteter Klassen müssen als Serializable gekennzeichnet werden (sonst: Exception)

- Klassen aggregierter Instanzen müssen als Serializable gekennzeichnet werden (sonst: Exception)

- lediglich das Setzen des Attributes Serializable reicht aus für korrekte Serialisierung

Attribute

Serialisierung mehrerer Instanzen

- Serialize-Methode für alle Instanzen sequentiell aufrufen

- Deserialize-Methode mit Stream sequentiell aufrufen bis Stream geleert

  Alternative:

  - Alle Instanzen in einer Auflistung verwalten

  - Auflistung serialisieren

  - Auflistung deserialisieren

Attribute

Xml-Serialisierung

- verarbeitet plattformunabhängies XML
- Ausgabe kann über Attribute gut beeinflusst werden
- Vorbedingungen:
  - zu serialisierende Klasse muss public sein
  - Klasse muss einen Standardkonstruktor definieren
  - Attribut [Serializable] muss nicht gesetzt sein (im Gegensatz zum Binary- und SOAPformatter)
- Speichert nur public Fields und Read/Write Properties eines Objekts. Deserialisierte Objekte sind eventuell unvollständig.
- Speichert keine Typinformation (keine Versionierung)
- wird bei XML WebServices in .Net verwendet

Attribute

Beispiel zur XML-Serialisierung

```
public class A{                  // public erforderlich
    public A(int j){i=j;}
    public A(){}                 // erforderlich
    public int i;
    static void Main(string[] args)  {
        A a = new A(5);
        FileStream myfile;
        myfile = new FileStream("C:test.dat", FileMode.Create);
        XmlSerializer xmlser = new XmlSerializer(typeof(A));
        xmlser.Serialize(myfile,a);
        myfile.Close();
        myfile = new FileStream("C:test.dat", FileMode.Open);
        A aneu = (A) xmlser.Deserialize(myfile);
        myfile.Close();
    }
}
```

Attribute

XML serialisierte Instanz

- reines XML keinerlei .Net Informationen
- kann von beliebigem Klienten verarbeitet werden
  natürlich auch von .Net

```
<?xml version="1.0"?>
<A xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <i>5</i>
</A>
```