



# **.NET** **Architecture** **CAMP**

Jörg Neumann | Acando  
Jörg Krause | [www.IT-Visions.de](http://www.IT-Visions.de)

## **Serverarchitektur in** **Client-Server-Anwendungen mit** **Entity Framework, LINQ, WCF, WPF**

- Principal Consultant bei Acando
- Associate bei Thinktecture
- MVP im Bereich „Client App Dev“
- Beratung, Training, Coaching
- Buchautor, Speaker
- Mail: [Joerg.Neumann@Acando.de](mailto:Joerg.Neumann@Acando.de)
- Blog: [www.HeadWriteLine.BlogSpot.com](http://www.HeadWriteLine.BlogSpot.com)



- Consultant und Trainer seit 1996
  - ASP.NET seit 2001, SharePoint Server seit 2003
  - SQL Server, .NET-Entwicklung, Web allgemein
  - IT-Visions.de Partner
- Autor
  - Carl Hanser, Apress, Pearson, Diverse Zeitschriften
- Projekte (Auswahl)  
*netrixcomponent.net, texxtoor.de*
- Mehr Informationen:  
*www.joergkrause.de*

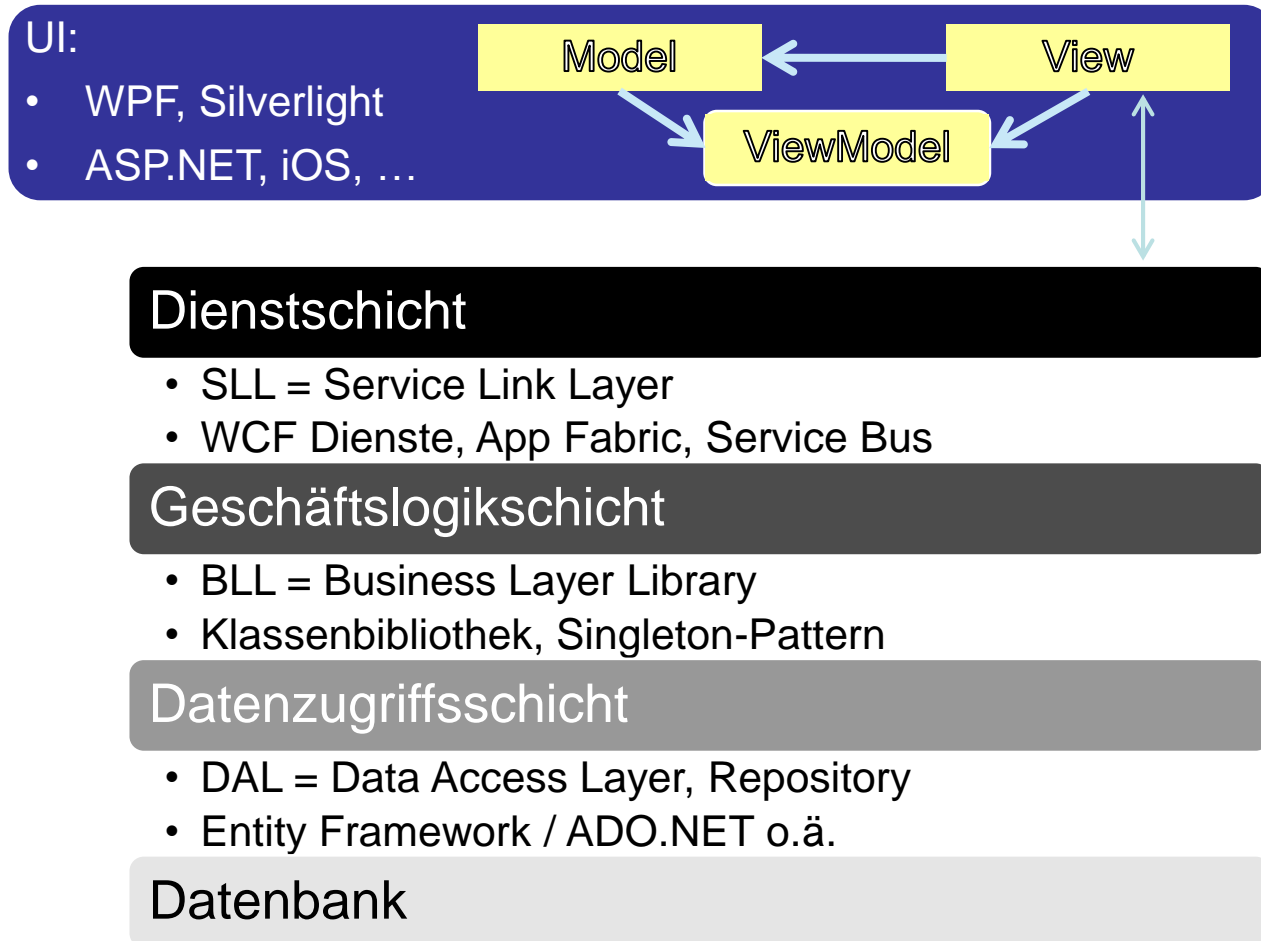
- Architektur
- Technologie:
  - Data Layer: Entity Framework
  - Business Layer: LINQ 2 EF
  - Service Layer: WCF

- Client-Architektur
- Technologie:
  - WPF
  - MVVM
  - Dependency Injection

# Architektur

- Ein komplettes Projekt mit
    - Datenbank (SQL Server)
    - Datenzugriffsschicht / Persistenzschicht (EF)
    - Businessschicht (C#/LINQ)
    - Serviceschicht (WCF)
    - Frontend (WPF)
- ...in nur 2 Tagen

Auch bei kleinen Projekten immer ein Mehrschichtmodell implementieren





- Singleton
- Multiuserfähiger Context
- Context-Factory

- Bessere Integration mit Service Layer
- Erweiterbarkeit
- Lesbarkeit

```
[DebuggerStepThrough]
public class Singleton<T> where T: new() {

    private static volatile object instance;
    private static object syncRoot = new Object();

    public static T Instance {
        get {
            if (instance == null) {
                lock (syncRoot) {
                    if (instance == null)
                        instance = new T();
                }
            }
            return (T)instance;
        }
    }
}
```

- Universelle Basis
- sicherer Web-Context Store für DataContext

```
public abstract class Manager<T> : Singleton<T>, IManager, IDisposable where T : new() {  
  
    private PortalContext _ctx;  
  
    protected PortalContext Ctx {  
        get {  
            return _ctx ?? DataContextFactory.GetWebRequestScopedDataContext<PortalContext>();  
        }  
    }  
  
    public PortalContext Context {  
        get { return Ctx; }  
        set { _ctx = value; }  
    }  
}
```

- Ein DbContext pro Service-Context
- Performance und einfachere Nutzung

```
static object GetWebRequestScopedDataContextInternal(Type type, string key, string connectionString) {
    object context;

    if (HttpContext.Current == null) {
        if (key == null) {
            key = "__CON__";
        }
        if (!ConsoleCache.ContainsKey(key)) {
            context = connectionString == null ? Activator.CreateInstance(type) : Activator.CreateInstance(type, connectionString);
            ConsoleCache.Add(key, context);
        } else {
            context = ConsoleCache[key];
        }
        return context;
    }

    // *** Create a unique Key for the Web Request/Context
    if (key == null)
        key = "__WRSCDC_" + HttpContext.Current.GetHashCode().ToString("x") + Thread.CurrentContext.ContextID;

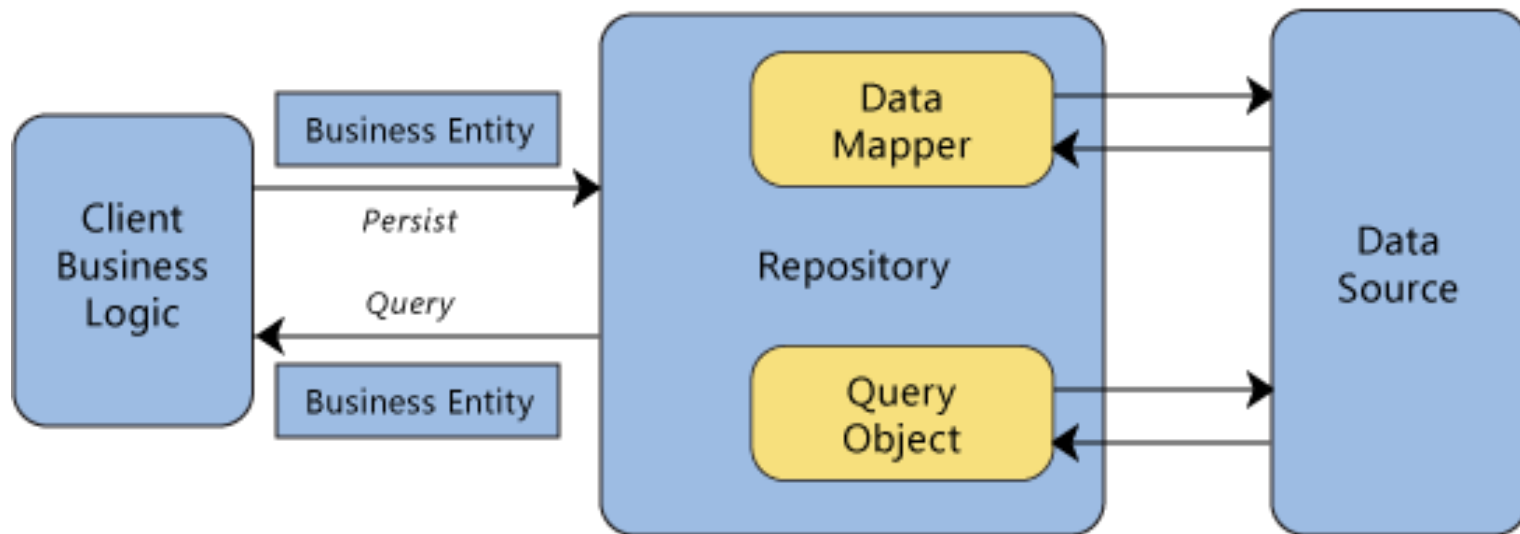
    context = HttpContext.Current.Items[key];
    if (context == null) {
        context = connectionString == null ? Activator.CreateInstance(type) : Activator.CreateInstance(type, connectionString);
        HttpContext.Current.Items[key] = context;
    }

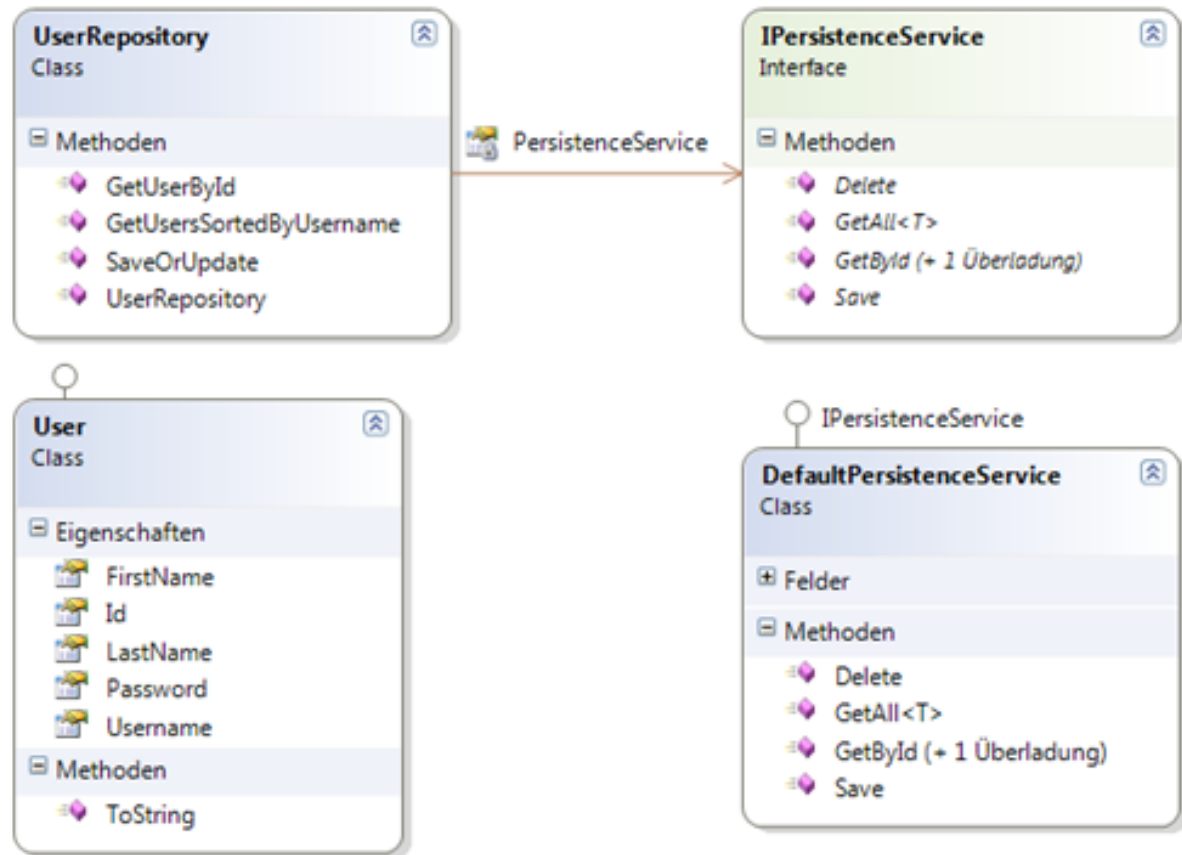
    return context;
}
```

- Direkter Zugriff auf Datenschichten ist falsch, weil
  - Code doppelt
  - Programmierfehler
  - Schwache Typisierung
  - Schwer mit Cache zu arbeiten
  - Business Logik nicht getrennt testbar

- Mehr testbarer Code
- Mehrere Zugriffsebenen auf die Datenschicht
- Zentrale Cache-Strategie
- Lesbarkeit und Wartbarkeit
- Arbeiten mit Domänenmodellen –  
modellgetriebener Ansatz
- Geschäftsregeln auf der Ebene der Modelle

- Repositories sind Brücken zwischen Datenquelle und Domänenmodellen
- Das Repository trennt die Geschäftslogik von der Datenquelle









**DEMO**

**Anlegen der Projektstruktur**

# Entity Framework

Einführung und Übersicht

- ***Viele OR/Ms*** verfügbar, meist auf ähnlichem Niveau
  - LLBLGenpro, NHibernate, EntitySpaces, ...
- Microsoft ***hat(te) zwei ...***
  - LINQ to SQL in (ab) Visual Studio 2008
  - ADO.NET Entity Framework ab Visual Studio 2008 SP1 (V 1.0) und 2010 (V 4.0) , neuste Version VS 2013 (V 6.0)

# Was ist Entity Framework?

- Sammlung von Werkzeugen und Diensten zum *Erzeugen eines Entity Data Model (EDM)*
  - Konzept → Mapping → Speicher
- Werkzeuge und Dienste zum *Benutzen des Entity Data Model*
  - LINQ to Entities, Object Services und Entity SQL
- Zusammen wird der “*object relational impedance mismatch*” adressiert
  - Konzeptionelle und technische Probleme, wenn objektorientierte Programmierstile auf relationale Datenbanken treffen

Objekte in  
Applikationen

Entity Data Model

Konzept

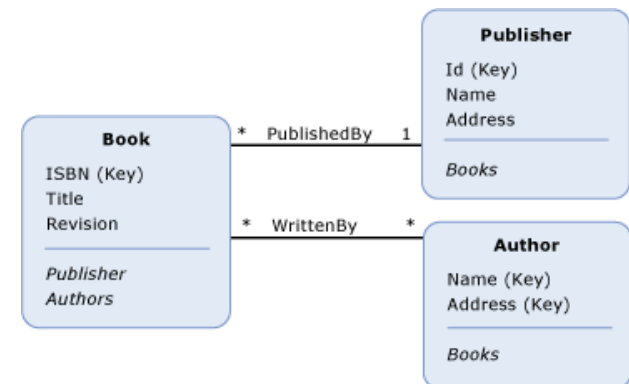
Mapping

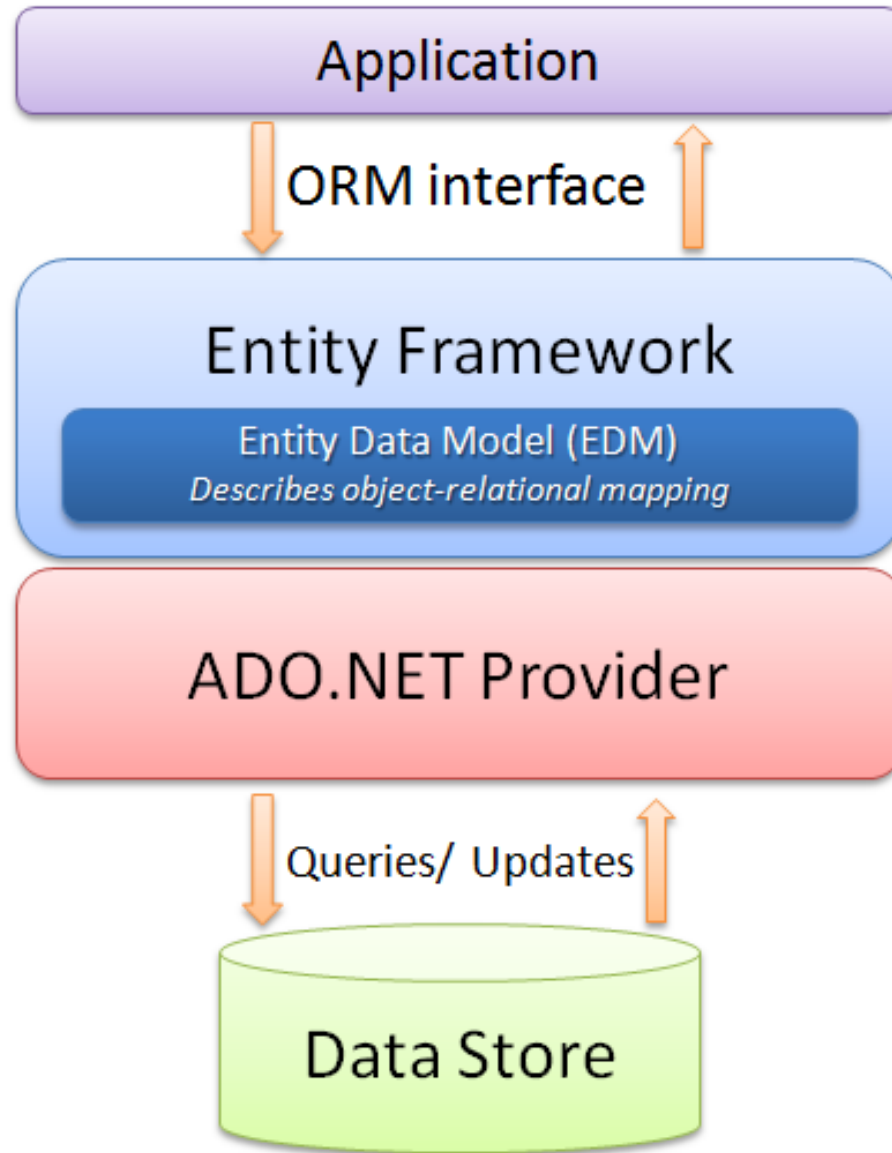
Speicher

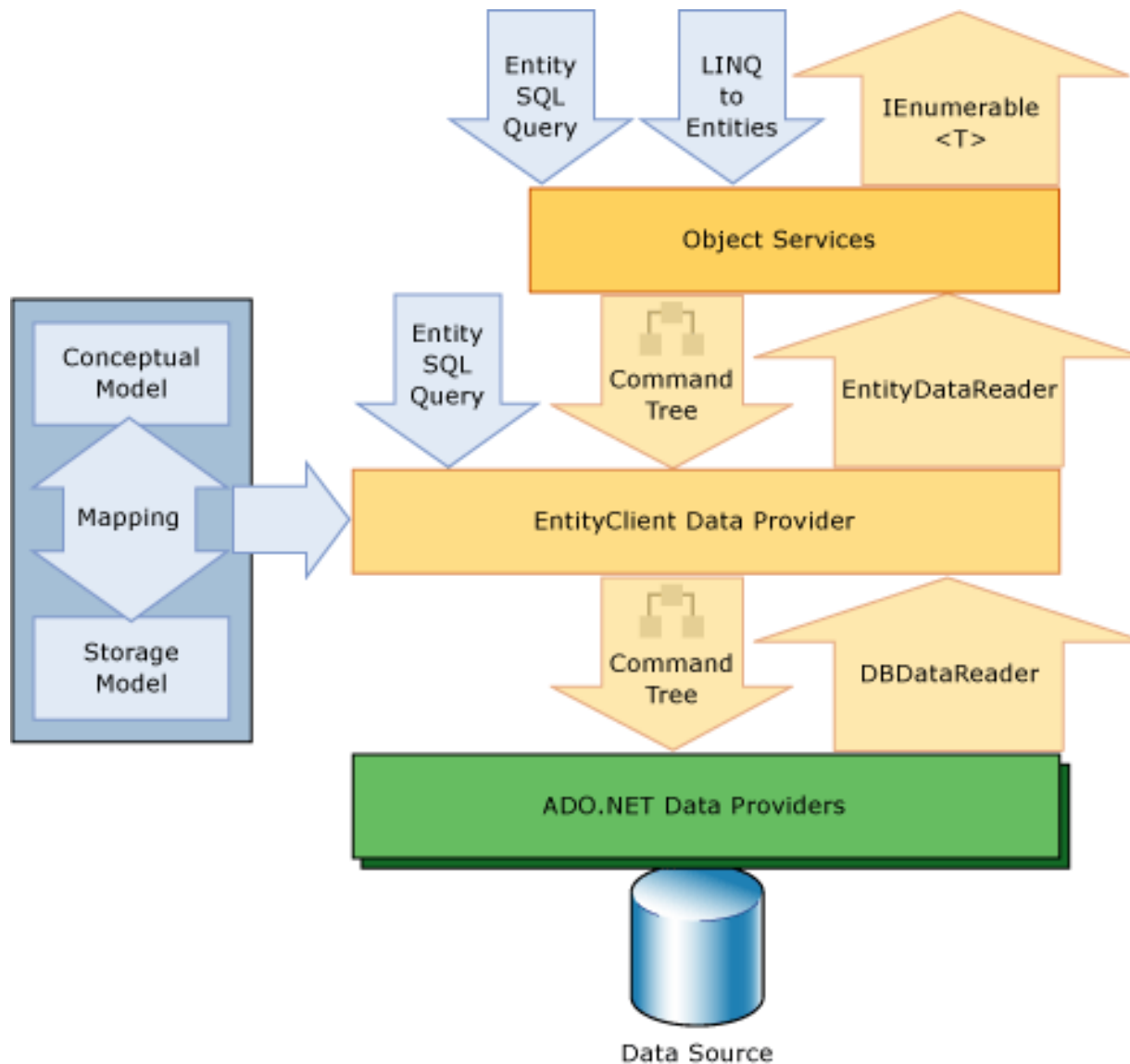
Datenreihen in  
Tabellen

- Entity Data Model
  - The Entity Data Model (EDM) is a set of concepts that describe the structure of data, regardless of its stored form. The EDM borrows from the Entity-Relationship Model described by Peter Chen in 1976, but it also builds on the Entity-Relationship Model and extends its traditional uses.
- Erweiterungen des E-R Modells:
  - Trennung der Entitäten und Relationen vom Speichermedium
    - <http://msdn.microsoft.com/en-us/library/ee382825.aspx>

- Entity Type
  - Beschreibt die Struktur der Daten, mittels
    - Entitätenschlüssel (key), Vererbung, Eigenschaften, Navigationseigenschaften
- Entity Set
  - Logischer Container für Instanzen eines Entity Types
- Association Type
  - Beschreibt Beziehungen im Datenmodell, mittels
    - Fremdschlüssel (foreign key), Multiplizität, Navigationseigenschaften
- Eigenschaften
  - Name, Typ (int, string), und Facet (Details)







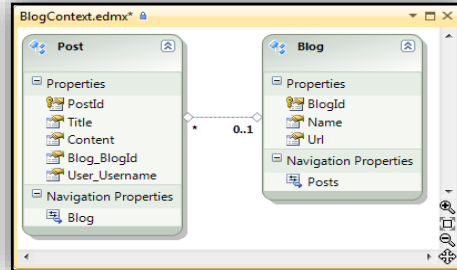


# Konzepte

Praktisches Arbeiten mit dem Entity Framework

- Visual Studio:
  - Entity Data Model Wizard
  - ADO.NET Entity Data Model Designer
  - Update Model Wizard
- Kommandozeile:
  - EDM Generator (EdmGen.exe)

## Designer basierend



## Code basierend

```
using System.Data.Entity;
using ConsoleApplication3.Entities;
using ConsoleApplication3.Mapping;

namespace ConsoleApplication3
{
    public class BlogContext : DbContext
    {
        public DbSet<Blog> Blogs { get; set; }
        public DbSet<Post> Posts { get; set; }
        public DbSet<User> Users { get; set; }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            modelBuilder.Configurations.Add(new BlogMap());
            modelBuilder.Configurations.Add(new PostMap());
            modelBuilder.Configurations.Add(new UserMap());
        }
    }
}
```



Neue  
Database

### Model First

- Erzeuge .edmx Modell im Designer
- Erzeuge Datenbank aus .edmx
- Klassen werden aus .edmx generiert

### Code First

- Klassen und Mapping im Code
- Datenbank zur Laufzeit erzeugen

Vorhandene  
Datenbank

### Database First

- .edmx model aus Datenbank entwickeln
- Klassen werden aus .edmx generiert

### Code First

- Klassen und Mapping im Code  
(Ggf. Reverse Engineering)

# Model First Database First

- Konzeptionelles Modell
- Speichermodell (Storage)
- Daten-Provider

- Schema Definition Language (CSDL)
- Konzeptionelles Modell → Domain Modell
- Dateierweiterung .csdl
- CSDL ist die Implementation des EF für das Entity Data Model

- Store Schema Definition Language (SSDL)
- Speichermodell → Logisches Modell
- Speichermodelle sind providerspezifisch
- Dateierweiterung .ssdl

- Mapping specification language (MSL)
- Mapping von Konzept zu Speichermodell
- Dateierweiterung .msl



- T4 Templates zum Erzeugen von Entity-Klassen aus dem CSDL
- Entitäten basierend auf EntitySet
  - Basiert auf .NET, System.Data
- Plain Old CLR Objects (POCO)
  - POCO-Erweiterungen aus Visual Studio Gallery
- POCO Proxy

- Lazy Loading (Deferred)
- Foreign Keys Abbildung
- Umfassende LINQ-Implementation
- ExecuteStoreQuery
- EntityFunctions und SqlFunctions
- Brauchbares generiertes SQL



**DEMO**

**Entity Framework installieren**

# Code First

- Direkt Datenbanken aus Code erzeugen
  - Kein Modell
  - Diverse Konventionen
  - Extrem produktiv für neue Projekte
- DAL Strategien
  - SoC
  - Manuelle Mapper
  - DTOs

- **Initializer**
  - Erster Zustand
  - Definierter Testzustand
- **ModelBuilder**
  - Verfeinerung des Models
  - Fluent-API
  - Globale Einstellungen
- **Migration**
  - Aktualisieren von Datenbanken
  - Erhalten von Daten

- DAL: Persistenzschicht
- Repository Pattern:
  - Data Transfer Objects
  - Interface basiert
  - Collection basiert
  - Wahlweise IQueryable<T> oder IEnumerable<T>
- Ergänzend:
  - Funktionen, Stored Procedure, Store Commands

- Database.SetInitializer(new
  - DropCreateDatabaseIfModelChanges
  - DropCreateDatabaseAlways
- Im Context:
  - override OnModelCreating
  - dann Fluent-API auf modelBuilder nutzen
- Ansonsten gelten Conventions

<http://msdn.microsoft.com/en-us/library/system.data.entity.modelconfiguration.conventions.aspx>



- Standards ("magic") für interne Einstellungen
- Wenn man was nicht haben will, dann diese entfernen:

```
modelBuilder.Conventions.Remove<ColumnOrderingConvention>();
```

- Oder man fügt sie explizit hinzu:

```
modelBuilder.Conventions.Remove<ColumnOrderingConvention>();
```

- Wenn es nicht reicht, gibt es Custom Conventions (> EF 6):

<http://msdn.microsoft.com/en-us/data/jj819164>

- Database (→ Einstellungen)
- DbContext (→ Datenbank)
- DbSet (→ Tabellen)

**DEMO**

**Datenmodelle anlegen und  
Datenbank erzeugen**

# Attribute

- Themen:
  - Attribute für den modellgetriebenen Ansatz
  - Steuerung über Attribute
  - Was sind Attribute?
  - DataAnnotations
  - Schema Attribute

- Deklarative Informationen zu Typen
- Modellgetriebener Ansatz
- Unterstützt das Entwurfsprinzip "Separation of Concerns"
  1. Beschreibe was bearbeitet werden soll
  2. Teile Sequenzen in kleine Schritte
  3. Organisiere Abhängigkeiten (Infrastruktur)

**Idealerweise beschränken wir uns erstmal auf 1.**

- Selbstbeschreibende Objekte
- Modellgetriebene Softwareentwicklung (MDSD)
  - DRY Prinzip (Don't Repeat Yourself)
- Vorteile:
  - Qualität, Testbarkeit, Klarheit, SoC
- Nachteile:
  - Anfangsaufwand
  - Komplexe Modellvalidierung ist eine Herausforderung

- Die Lösung des SoC-Problems über
  - Objekte beschreiben Daten (=Metadaten)
  - Attribute beschreiben
    - Datenbankdesign
    - Validierung
    - Hilfeinformationen
    - UI-Hinweise
    - Serialisierung



- Dekorieren von
  - Klassen
  - Eigenschaften
  - Methoden
  - Felder
  - ...also alle Typen

```
using System;
[AttributeUsage(AttributeTargets.All)]
public class HelpAttribute : System.Attribute
{
    public readonly string Url;

    public string Topic           // Topic is a named parameter
    {
        get
        {
            return topic;
        }
        set
        {
            topic = value;
        }
    }

    public HelpAttribute(string url) // url is a positional parameter
    {
        this.Url = url;
    }

    private string topic;
}
```

- [AttributeUsage]
  - Wo darf das Attribut eingesetzt werden?
  - Wie oft darf es vorkommen?
- NameAttribute == [Name]
  - verkürzte Schreibweise

- Auslesen der Meta-Daten des Attributes über `GetCustomAttributes`

```
class MainClass
{
    public static void Main()
    {
        System.Reflection.MemberInfo info = typeof(MyClass);
        object[] attributes = info.GetCustomAttributes(true);
        for (int i = 0; i < attributes.Length; i++)
        {
            System.Console.WriteLine(attributes[i]);
        }
    }
}
```

- Linq2Objects ist hilfreich
  - `Cast<AttributeName>()`
  - `Single()`, `SingleOrDefault()`

# Data Annotations

DataAnnotations-Attribute in  
ASP.NET, MVC, DD, EF CF, .NET, WPF, ...

- `System.ComponentModel.DataAnnotations.Schema`
- `System.Web.Mvc`
  - Auch anwendbar, wenn kein MVC!

- Nutzung in diversen Umgebungen:
  - WinForms, ein wenig WPF
  - Silverlight, spez. RIA-Services
  - ASP.NET Dynamic Data
  - ASP.NET WebForms und MVC
  - Entity Framework Code First
- Nutzung immer mit Erweiterungscode

**Nutze die Infrastrukturkonzepte,  
auch wenn nicht alles auskodiert ist!**



- **Key**: Primärschlüssel
- **ConcurrencyCheck**: Update unter Einbeziehung des Originalwertes, sodass Änderungen anderer Benutzer eine Ausnahme auslösen
- **TimeStamp**: Zeitstempel (byte[]) für Concurrency Checks
- **DbFunction**: ab EF 6, Methode → UDF
- **Required**: NOT NULL
- **StringLength**: varchar(Wert)
- **DataType**: Time- oder Date-Anteil von DateTime u.ä.

- Attribute aus

`System.ComponentModel.DataAnnotations.Schema:`

- `Column`: Spaltenname und Typ
- `Table`: Tabellenname und Schema
- `ComplexType`: Abbildung als flaches Spaltenmodell
- `DatabaseGenerated`: Identity-Spalte
- `ForeignKey`: Fremdschlüssel
- `InverseProperty`: Anderes Endes der Relation
- `NotMapped`: Wird nicht in der Datenbank abgebildet

- Compare
- CreditCard\*
- CustomValidation, Remote
- DataType
- EmailAddress, Phone \*
- FileExtension \*, Url \*
- MaxLength \*\*, MinLength \*\*, StringLength \*\*
- Range
- Required \*\*
- RegularExpression
  - \* Erforderliche Erweiterung MetadataProvider und z.T. nur über eigene Templates
  - \*\* Wirkt auch auf den Datenbankmodellierer in EF Code First

- `[MetadataType(typeof(T))]`
  - Liefert Metadaten für die Validierung
  - Ursprünglich für EF Database First, um generierte Modelklassen mit eigenen Annotations zu versehen
  - Auch, wenn Validierung in MVC != EF
  - Model-Erweiterung ohne explizit anderes View-Model oder Verweis auf View-Model
  - Klareres Klassendesign, Übersichtlichkeit

- Display: Feldname
- DisplayFormat: Formatierung der Ausgabe
- DisplayColumn: Name einer Relationsspalte
- ReadOnly: Nur Lesen
- Editable\*: Kann bearbeitet werden
- ScaffoldColumn, ScaffoldTable: Wird beim Generieren von Formularen beachtet
- HiddenInput\*: Erzeugt ein verstecktes Feld
- UIHint, FilterUIHint\*: Anzeige- und Filtervorlagen
- Nicht direkt unterstützt:
  - Browsable, Category, Description, DefaultValue

\* Erforderliche Erweiterung MetadataProvider und z.T. nur über eigene Templates

**DEMO**

**Dekorieren der Datenmodelle  
mit Attributen**

# Umgang mit dem Context

Für Multi-User-Umgebungen (Client/Server, ASP.NET)

- Erzeugen von Proxyobjekten:
  - Standardmäßig werden Proxies benutzt (keine POCOs), auch bei Code First (→ [Proxies](#))
- Lazy Loading
  - Standardmäßig an
  - in POCOs mit `virtual` übergebar
  - kann aber nur eine Stufe laden, keinen kompletten Graphen**
  - `Configuration.LazyLoadingEnabled = true|false`



- Weitere Funktionen der Konfiguration:
  - **AutoDetectChangesEnabled**
  - **UseDatabaseNullSemantics**
  - **ValidateOnSaveEnabled**

- DbContext
  - **Entry()**
  - **GetValidationErrors()** → Collection
  - **SaveChanges()**
- Tipp:
  - Überschreibe **SaveChanges** und rufe dort **GetValidationErrors** ab

- **Add():**
  - Hinzufügen (INSERT)
- **Attach():**
  - Anhängen eines vorher abgehängten Objekts (muss in der Datenbank bereits existieren, hat aber möglicherweise Tracking verloren, wegen Dienst oder so)

- Find():
  - Suche nach Primärschlüssel
- Remove():
  - Entfernen (DELETE)
- Local:
  - Erzeugt eine ObservableCollection<T>, deren Änderungen Teil des Tracking sind und bei SaveChanges beachtet werden → DataSet

[http://msdn.microsoft.com/en-us/library/gg679592\(v=vs.113\).aspx](http://msdn.microsoft.com/en-us/library/gg679592(v=vs.113).aspx)

- **Create**
  - Erzeuge Proxy (siehe Proxies)
  - Füge das Objekt nicht sofort ein (erfordert weiteres Add/Attach)
- **Include**
  - Erweiterung des Objektgraphen
  - unabhängig von virtual / Lazy Loading
- **SqlQuery**
  - Direkte Abfrage mit SQL
  - Ergebnis wird Teil des Tracking

- Added
- Unchanged
- Modified
- Deleted
- Detached

Wirken sich auf SaveChanges aus!

- Der Status steuert das Tracking – Verfolgen von Änderungen an Proxy-Objekten

- Entweder direkt:

```
using (var context = new SessionContext())
{
    var session = new Session { Name = "EF 6" };
    context.Sessions.Add(session);
    context.SaveChanges();
}
```

- Oder über den Status:

```
using (var context = new SessionContext())
{
    var session = new Session { Name = "EF 6" };
    context.Entry(session).State = EntityState.Added;
    context.SaveChanges();
}
```

- Entscheidend ist der Zustand des Primärschlüssels:

```
public void InsertOrUpdate(Session session)
{
    using (var context = new SessionContext())
    {
        context.Entry(session).State = session.Id == 0 ?
                                         EntityState.Added :
                                         EntityState.Modified;

        context.SaveChanges();
    }
}
```



- Standard: Lazy-Loading
  1. Entweder überhaupt kein Laden
  2. Oder Laden nur auf Anfrage, dann mit virtual
  3. Oder Explizites Laden (Nachladen)

- Nachladenverhalten (einfache Navigation Property)

```
context.Entry(sess).Reference(p => p.Speaker).Load();
```

- Nachladeverhalten (Collection)

```
context.Entry(spk).Collection(p => p.Sessions).Load();
```

Geht auch, wenn LazyLoading abgeschaltet ist

- Beschränkung der Abfrage von nachgeladenen Collections
- **Query()**  

```
context.Entry(sess)  
    .Collection(b => b.Speakers)  
    .Query()  
    .Where(p => p.Name.Contains("EF"))  
    .Load();
```
- Andernfalls würde erst die ganze Collection geladen werden und dann gefiltert, mit **Query** wird das Filter Teil der Abfrage
- Auch gut mit Count, z.B. **Query() .Count()**

- Option: Eager-Loading

- Include(x => x.NavProperty)
- Include(x => x.NavProperty.Select(y = y.MoreNavProperty))

```
var sess = context.Sessions
    .Include(b => b.Speakers)
    .where(b => b.Name == "EF 6")
    .FirstOrDefault();
```

- Include("NavProperty")
- Include("NavProperty.MoreNavProperty")

- Zum Verfolgen von Änderungen werden Proxy-Objekte benutzt
- **Configuration.ProxyCreationEnabled** auf dem Context-Objekt

- Erzeugen eines Proxy-Objekts:
  - `context.<SetName>.Create<EntityName>()`
  - Also z.B.  
`context.Sessions.Create<Session>()`
- Erzeugen eines Basisobjekts aus einem Proxy-Objekt:
  - `ObjectContext.GetObjectType(<entity>.GetType())`
  - Also z.B.  
`using System.Data.Entity.Core.Objects;`  
`ObjectContext.GetObjectType(session.GetType())`



**DEMO**

**Businesslayer, erste  
Zugriffsfunktionen**

# Datenzugriffstechnik

# LINQ

LINQ allgemein, LINQ to EF



- Was ist LINQ?
- Versionen
  - LINQ to ...
- LINQ intern
- LINQ Übungen

> C# 3.0

> VB 9.0

...

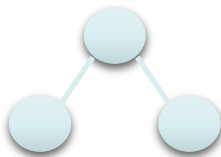
## .NET Language Integrated Query

LINQ to  
Objects

LINQ to  
EF

LINQ to  
SQL

LINQ to  
XML



Objekte



SQL

```
<book>  
<title/>  
<author/>  
<year/>  
<price/>  
</book>
```

XML

```
ICollection<int> result = new List<int>();  
foreach(int n in numbers) {  
    if (n % 2 == 0) {  
        result.Add(n);  
    }  
}
```

ALT

Laufvariable IEnumerable

```
var result = from n in numbers  
              where n % 2 == 0  
              select n;
```

Selektion

Projektion

LINQ

```
var query =  
    from k in kunden  
    where k.Umsatz > 300  
    group k by k.Ort into g  
    orderby g.Key  
    select new {  
        Ort = g.Key,  
        Summe = g.Sum(k => k.Umsatz)  
    };
```

- Implementiert als Erweiterung der Sprachen C# und VB.NET
- Streng typisiert
- Abfragen werden zur Compile-Zeit geprüft, nicht erst zur Laufzeit
- IntelliSense-Unterstützung in Visual Studio
- Keine neue CLR notwendig, basiert auf Erweiterungsmethoden

- LINQ Ausdrücke werden zur Compile-Zeit übersetzt in Erweiterungsmethoden und Lambda Expressions
- Durch die Lambda Expressions steht der LINQ-Ausdruck zur Laufzeit als Expression Tree zur Verfügung
- Die Expression Trees werden je nach LINQ Variante in eine andere Darstellung (wie z.B. SQL) übersetzt

- Deferred Execution
  - Ein LINQ-Ausdruck wird erst ausgewertet, wenn auf die Werte zugegriffen wird
  - Ein LINQ-Ausdruck kann deshalb wiederverwendet werden, auch wenn sich die zugrundeliegenden Werte geändert haben
  - Es kann mit Datenmengen hantieren, bevor diese im Speicher sind

- Basis für LINQ, aber auch sonst gut verwendbar:
  - Lambda-Ausdrücke
  - Extension Methods
  - Expression Trees
  - Type Inference (var) und anonyme Typen



- Lambda-Ausdrücke
  - Anonyme Ausdrücke als Erweiterung anonymer Methode
  - Nur andere Syntax oder steckt doch mehr dahinter?
  - Werden benötigt um die in LINQ benutzte Syntax zu erlauben

Klassischer Weg:

**// Definition Funktion → Delegate → Definition → Aufruf**

```
public static bool IsOdd(int i) {  
    return (i & 1) == 1;  
}  
  
public delegate bool NumTester(int i);  
    public static void ShowOdd(  
        int from, int to, NumTester filter) {  
        for (int i = from; i <= to; ++i) {  
            if (filter(i)) { Console.WriteLine(i);  
            }  
        }  
    }
```

**// Verwendung**

```
ShowOdd(1, 10, new NumberTester(IsOdd));
```

## C# 2.0-Weg:

**// Definition Delegate → Definition → Aufruf**

```
public delegate bool NumTester(int i);  
MatchingNumbers(1, 10,  
    delegate(int i) {  
        return (i & 1) == 1; }));  
}
```

**// Verwendung**

```
ShowOdd(1, 10, new NumberTester(IsOdd));
```

## C# 3.0-Weg:

**// Definition Delegate → Aufruf**

```
public delegate bool NumTester(int i);  
ShowOdd(1, 10, i => (i % d) == 0);
```

**// Verwendung**

```
void ShowOdd(1, 10, NumTester test)  
{  
}
```

## C# 3.0-Weg:

**// Definition Delegate → Aufruf**

```
ShowOdd(1, 10, i => (i % 2) == 0);
```

**// Verwendung**

```
void ShowOdd(int from, int to,  
             Func<int, bool> filter) {  
  
}
```

- Extension Methods
  - Hinzufügen neuer Methoden zu öffentlichen CLR-Typen
  - Kein erneutes Compilieren und keine Ableitungen erforderlich
  - Flexibilität dynamischer Sprachen mit der Sicherheit der typsicheren Sprache
  - Benutzt, um bestehenden Klassen „LINQ-Funktionalität“ zu geben

- Und so geht's
  - Statische Klasse mit den Extension Methods
  - Extension Methods durch Parameter mit Schlüsselwort „this“
  - Weitere Parameter folgen wie üblich
  - Direkter Aufruf möglich

```
public static class MyProjectExtensions
{
    public static bool IsValidEmail(this string s)
    {
        Regex regex = new
            Regex(@"^[\w-\.\.]+\@([\w-]+\.\.)+[\w-]{2,4}$");
        return regex.IsMatch(s);
    }
}
```

- Expression-Trees stellen Abfragestrukturen dar
  - Darstellung der Auflösung von Ausdrücken
  - Baumstruktur mit Darstellung aller Entitäten
  - Zur Laufzeit manipulierbar
  - Nicht „magic“, sondern basierend auf spezifischen Klassen in **System.Linq.Expressions**
- Vorbereitung zur Auswertung von Ausdrücken:

```
Expression<Func<int, bool>> filter =  
    n => !((n * 3) < 5);  
filter.Body.PrefixForm();
```



- Rückgaben bei Projektionen oft nicht definierbar
- Collections von anonymen Typen?
- Drei "flexible" Typen in .NET
  - **var**
    - Compiler legt fest und fixiert → Typsicher, Flexibel, kein Cast erforderlich
  - **object**
    - Compiler akzeptiert alles → Nicht Typsicher, Cast erforderlich
  - **dynamic**
    - Dynamische Bindung zur Laufzeit → Laufzeittyp, kein Cast erforderlich

- Die primären erweiterten Schnittstellen
- **IEnumerable<T>**
  - LINQ to Objects, Abfrage erfolgt sofort und weitere Abfragen erfolgen im Speicher
- **IQueryable<T>**
  - Spätere "Verfeinerungen" der Abfrage werden gesammelt und in den Abfragebaum aufgenommen, diese Abfrage erfolgt komplett in der Datenbank (wenn möglich und vom Provider unterstützt)

- Materialisierung
  - Objekte sind nun physisch entstanden
  - IEnumerable<T> alleine reicht nicht
  - Explizite Materialisierung:
    - ToList()
    - ToArray()
    - ToDictionary()
  - Implizite Materialisierung:
    - GetEnumerator()
    - foreach
    - Datenbindung

- **Operatoren:**

- Einschränkungen:

- `where`            `// .Where()`

- Gruppierung:

- `group by`    `// .GroupBy()`

- Sortierung:

- `orderby`       `// .OrderBy()`

- Projektion:

- `select`        `// .Select()`

- **Einschränkungen**  
schränken die Ergebnismenge ein, z.B.:

`where n > 2`

`where m.Ort == "Berlin"`

`where a.Count > 100 && a.Sum > 1000`

`numbers.Where(number => number == index)`

- **Gruppierungen**  
gruppieren die Ergebnismenge:

```
group k by k.City into g
```

```
group k by k.Name[0] into g
```

```
.GroupBy(k => k.Name)
```

- **Sortierungen**

sortieren die Ergebnismenge, z.B.:

`orderby n descending`

`orderby kunde.Nachname, kunde.Vorname`

`orderby a.Umsatz descending, a.Gewinn`

`.OrderBy(n => n.Name)`

`.OrderByDescending(a => a.Umsatz)`

- **Projektionen**

bestimmen die Struktur der Ergebnismenge:

```
select kunde
```

```
select new { kunde.Name, kunde.Umsatz }
```

```
select artikel.Preis * pos.Menge
```

```
.Select(k => k)
```

```
.Select(k => new { Index = k.Id })
```



- **Partitionierung:**
  - Take, Skip, TakeWhile, SkipWhile
- **Mengenoperationen:**
  - Distinct, Union, Intersect, Except
- **Umwandlung:**
  - ToArray, ToList, ToDictionary, OfType, Cast
- **Elementauswahl:**
  - First\*, Single\*, Last\*, ElementAt\*
- **Erzeugen:**
  - Range, Repeat
- **Vorhandensein in einer Menge:**
  - Any, All
- **Aggregatfunktionen:**
  - Count, Sum, Min, Max, Average, Aggregate, Zip

- As-Funktionen zum Abruf mit erweiterten Eigenschaften
- AsEnumerable
  - kennt IEnumerable: Macht Daten iterierbar
- AsQueryable
  - kennt IQueryable: Macht Daten abfragbar (providerabhängig)
- AsNoTracking
  - IQueryable ohne Tracking der Entities (nur EF)
- AsParallel
  - ParallelQuery: Parallelisierbare Abfrage (siehe PLINQ)
- AsStreaming
  - Bisher war keine Pufferung, es wurde immer gestreamt
  - Neu (ab EF 6) ist Pufferung an, wenn man dies nicht haben will, nutzt man AsStreaming, um wieder Abfragen per Stream zu holen

- The LINQ Project

<http://msdn.microsoft.com/data/ref/linq>

- 101 LINQ Samples

<http://msdn2.microsoft.com/en-us/vcsharp/aa336746.aspx>



**DEMO**

**Businesslayer ausbauen,  
Bedarf des Clients erfüllen**

# WCF

Eine kompakte Einführung  
Entwicklung der Serviceschicht

- Grundlagen (Protokolle etc.)
- SOA
- Einführung (Architektur, Vorgehensweise)
- Tools
- Dienstprogrammierung
- Clientprogrammierung
- Hosting-Varianten

- SOAP
- WSDL
- WS-\*
- REST

- Früher:
  - Simple Object Access Protocol
  - Seit 1.2 nur noch "SOAP" (weil weder einfach noch objektorientiert)
- XML basierte Standard zur Kommunikation
- RPC – Remote Procedure Calls



```
<Envelope>
  <?xml version="1.0"?>
  <soap:Envelope xmlns:soap=" [...]"
    soap:encodingStyle=" [...]">
    <soap:Body>
      <GetFlightsResponse>
        <Flight><Time>1700</Time> [...] </Flight>
        <Flight><Time>1800</Time> [...] </Flight>
        <Flight><Time>1900</Time> [...] </Flight>
      </GetFlightsResponse>
    </soap:Body>
  </soap:Envelope>
</soap:Envelope>
```

**Response**

**Request**

- **RPC-Style**
  - Web-Services werden im Sinne von entfernten Methodenaufrufen genutzt.
  - Eine Web-Service-Methode kann Übergabeparameter und Rückgaben besitzen.
  - Die Übergabewerte müssen umkodiert werden. XML selbst zum Beispiel wird als umkodierter String übertragen.
- **Document-Style**
  - Eine Web-Service-Methode hat nur ein Argument.
  - Das Argument geht als echtes XML-kodiertes Objekt auf den Weg und wird nicht umkodiert.
  - Da sich jedes beliebige XML-Dokument schicken lässt, ist das flexibler bei Anpassungen, denn Änderungen am WSDL für die Argumente sind nicht nötig.

- Literal-Encoding
  - Literal Encoding nutzt ein XML Schema zum Validieren der SOAP-Daten
- SOAP-Encoding
  - auch Section 5 Encoding genannt
  - Schreibt vor, wie Argumente verpackt werden

- Web Service Description Language
  - XML basiert
  - Beschreibung der Operationen eines Dienstes
  - Dient der Erstellung

```
</wsdl:portType>
- <wsdl:binding name="BasicHttpBinding_IFilterService" type="tns:IFilterService">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http"/>
  - <wsdl:operation name="AllFilters">
    <soap:operation style="document" soapAction="http://tempuri.org/IFilterService/AllFilters"/>
    - <wsdl:input>
      <soap:body use="literal"/>
    </wsdl:input>
    - <wsdl:output>
      <soap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
  - <wsdl:operation name="GetFilters">
    <soap:operation style="document" soapAction="http://tempuri.org/IFilterService/GetFilters"/>
    - <wsdl:input>
      <soap:body use="literal"/>
    </wsdl:input>
    - <wsdl:output>
      <soap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
  - <wsdl:operation name="GetFilter">
    <soap:operation style="document" soapAction="http://tempuri.org/IFilterService/GetFilter"/>
    - <wsdl:input>
      <soap:body use="literal"/>
    </wsdl:input>
    - <wsdl:output>
      <soap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
- <wsdl:service name="FilterService">
  - <wsdl:port name="BasicHttpBinding_IFilterService" binding="tns:BasicHttpBinding_IFilterService">
    <soap:address location="http://localhost:23596/FilterService.svc"/>
  </wsdl:port>
</wsdl:service>
</wsdl>
```

- WS-Standards, \* steht für "viele"
- Die wichtigsten:
  - *WS-MetadataExchange*. WS-MetadataExchange definiert SOAP-Nachrichten zum Austausch von Metadaten über Services. Meist WSDL-Dokumente.
  - *WS-Policy*. Beschreibt Eigenschaften, die von einem Service und/oder von einem Client geboten bzw. erwartet werden.
  - *MTOM*. Kommt zum Einsatz, wenn SOAP binäre übertragen soll.
  - *WS-Security, WS-Trust, WS-Federation, WS-SecureConversation*. Diese Standards werden zur sicheren Kommunikation mit Services verwendet.
  - *WS-ReliableMessaging*. Stellt sicher, dass die gesendeten Nachrichten auch beim Service ankommen und richtig verarbeitet werden.
  - *WS-AtomicTransaction*. Legt fest, wie verteilte Transaktionen, die sich über Servicegrenzen erstrecken, stattzufinden haben.

- REST = Representational State Transfer
  - Kein Standard, sondern Architekturstil
  - Basiert aber auf Standards:
    - HTTP, URI, XML/JSON, MIME
  - Merkmale:
    - Skalierbar
    - Allgemeingültig
    - Erweiterbar

- Was genau adressiert REST?
  - Identifizierung von Ressourcen
  - Manipulation von Ressourcen
  - Selbstbeschreibende Nachrichten

- Identifizierung von Ressourcen
  - Eine Ressource ist:
    - Alles, was sich adressieren lässt, oder
    - alles, was ein eindeutigen URI hat



- Manipulation von Ressourcen:
  - Darstellung des Zustands einer Ressource
  - Übertragung zwischen Ressourcen
  - Verbindungen zu anderen Ressourcen

- Selbstbeschreibende Nachrichten
  - Identifizierung der Ressourcen
  - Darstellung des Datenformats

- JSON = JavaScript Object Notation
- Effizient, kompakt
- Alternative

```
{  
  "CreditCard" : "Visa",  
  "Number" : "1234-5678-9012-3456",  
  "Owner" : {  
    "LastName" : "Max",  
    "Firstname" : "Muster",  
    "Sex" : "male",  
    "Preferences" : [  
      "Golf",  
      "Reading",  
      "Badminton"  
    ],  
    "Age" : null,  
    "Deckung" : 1000000,  
    "Währung" : "EURO"  
  }  
}
```

- Abruf von Daten

```
GET /flights/Graz-Frankfurt?date=[...] HTTP/1.1  
Host: www.myserver.com
```

- Schreiben von Daten

```
POST /flights HTTP/1.1  
Host: myserver  
Content-Type: application/xml  
<?xml version="1.0"?>  
<Flight>  
  <Date>2010-01-20</Date>  
  <From>Graz</From>  
  [...]  
</Flight>
```

- Ändern

```
PUT /flights/4711 HTTP/1.1
Host: myserver
Content-Type: application/xml
<?xml version="1.0"?>
<Flight>
<Date>2010-01-20</Date>
<From>Graz</From>
[...]
```

- Löschen

```
DELETE /flight/4711 HTTP/1.1
```

- Plain old XML
- Wie REST mit XML ohne REST "in mind"
- Also eigentlich normale Dienste mit XML als Nachrichtenformat

- REST
  - HTTP als Basis ist extrem akzeptiert
  - Schlankes Format ohne viel Overhead
  - JSON als ernste Alternative zu XML
- SOAP
  - WSDL als anerkannte Beschreibungssprache
  - Automatische Proxy-Generierung
  - Alt aber erfahren 😊
  - Viele Funktionen (Routing, Exception Handling, ...)

# WCF

Das wichtigste zum Einsatz



- Ein Modell für verteilte Systeme, zerlegt in Präsentationsschicht, Logikschicht und Datenschicht
- Präsentation:
  - WPF, ASP.NET/JS, Silverlight
- Daten:
  - LINQ – Bindung an Daten
- Logik:
  - Eigener Entwurf, plain C#
- Und Kommunikation:
  - WCF = Lokal, Netzwerk, Internet

- Explizite Grenzen
  - Kommunikation wird nicht versteckt
- Dienste sind autonom
  - Unabhängig verteilt, verwaltet und versioniert
- Dienste teilen Kontrakte und Schemata, nicht Typen
  - Kontrakte definieren Verhalten, Schematas definieren Daten
- Kompatibilität basiert auf Richtlinien
  - Richtlinien trennen Verhalten von Zugriffsbeschränkungen

- Kontrakt
  - Eine Schnittstelle, die den Dienst beschreibt
  - Service, Daten, Message (Nachricht)
- Endpunkte
  - Address: <http://localhost/Calculator/service>
  - Binding: wsHttpBinding
  - Contract: ICalculator
- Implementierung
  - Klassen, die die Kontrakt-Schnittstelle implementieren
  - Code, der das Hosting liefert oder steuert

## Einheitliches Programmiermodell für dienstorientierte Applikationen auf der Windows Plattform

### Unification

Vereinheitlichung der Modelle – Eine Methode für alle Anwendungen

---

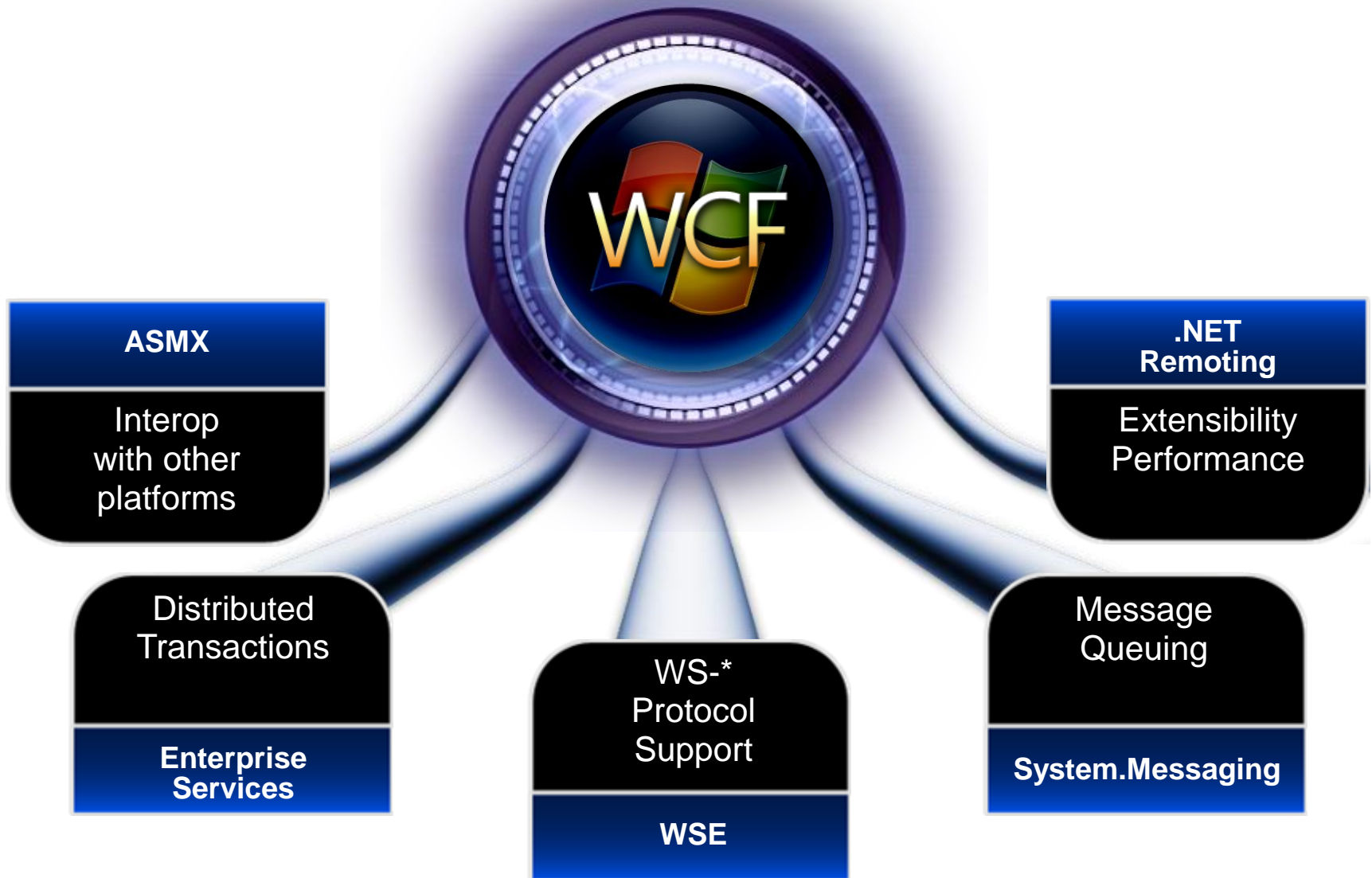
### Service Orientation

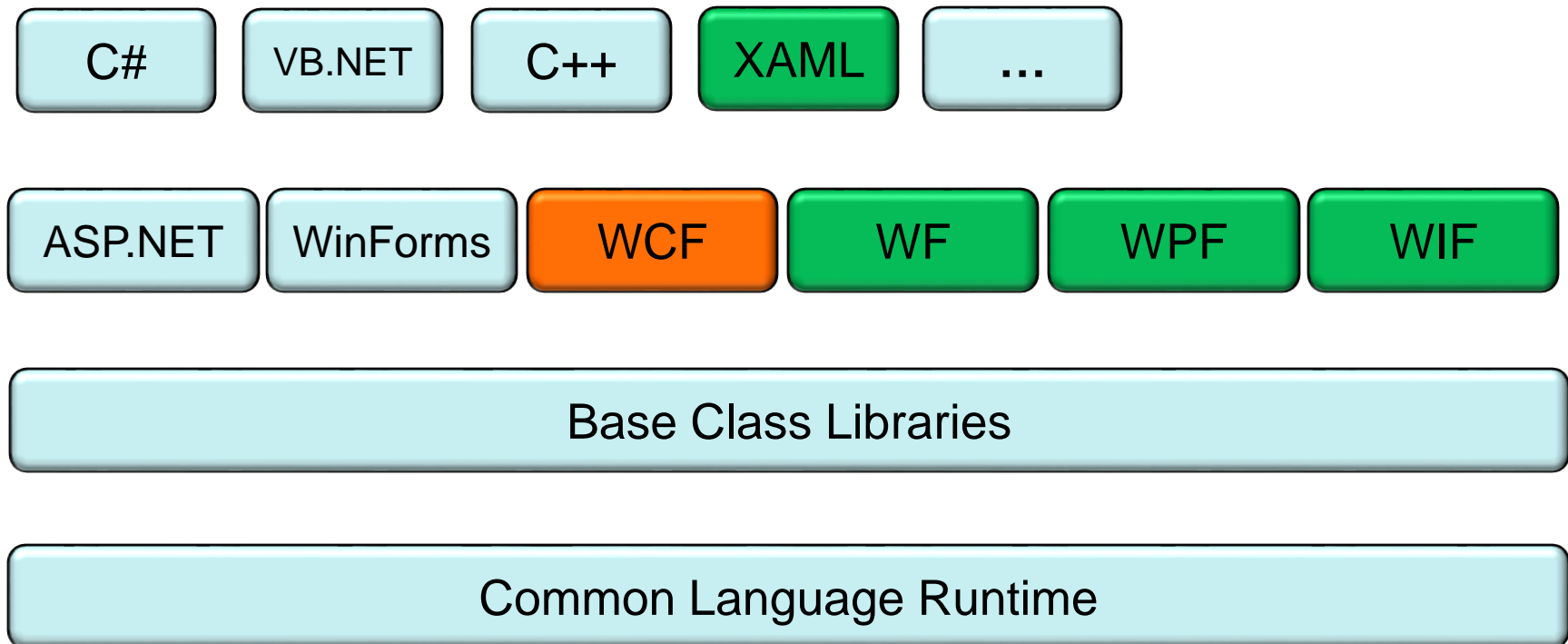
Codebasiertes Modell für dienstbasierte Anwendungen

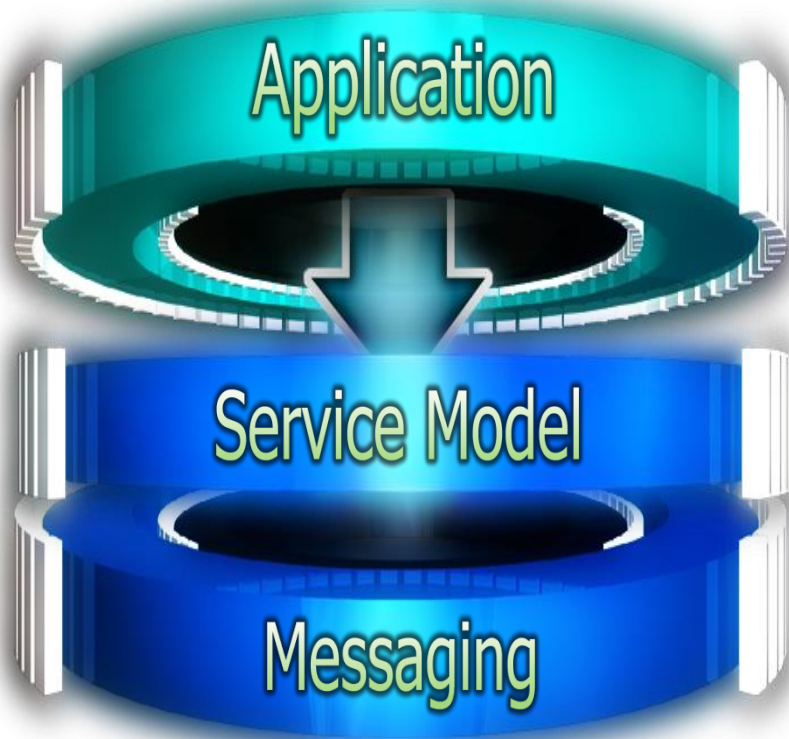
---

### Integration

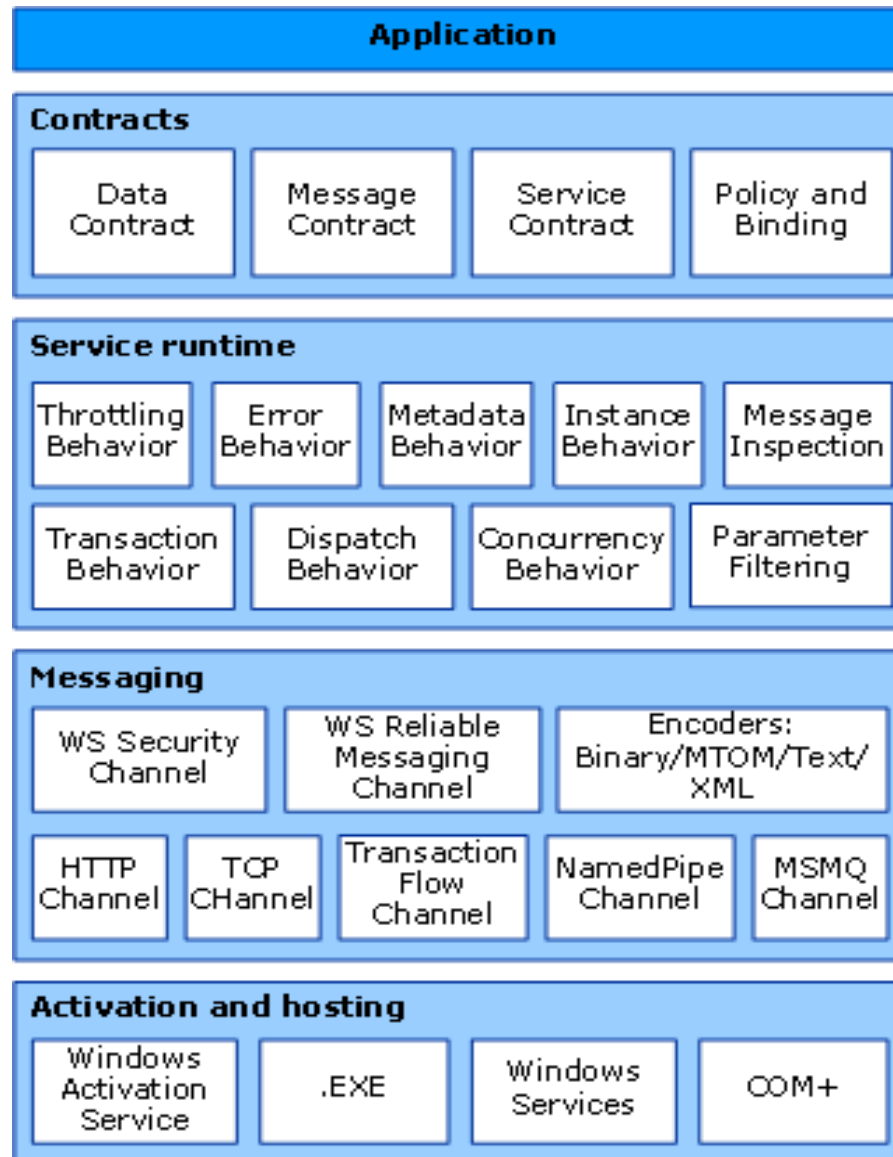
Interoperabel mit anderen Plattformen







- Dienstmodell
  - Dienstkontrakte
  - Datenkontrakte
  - Nachrichtenkontrakte
- Messaging
  - Message Klasse
  - Verteilungskontrolle





- Bindings, Channels, Endpoints, Messages, Serialization
- Activation, Concurrency, Hosting, Security, Sessions
- Queuing, Transactions
- Exceptions

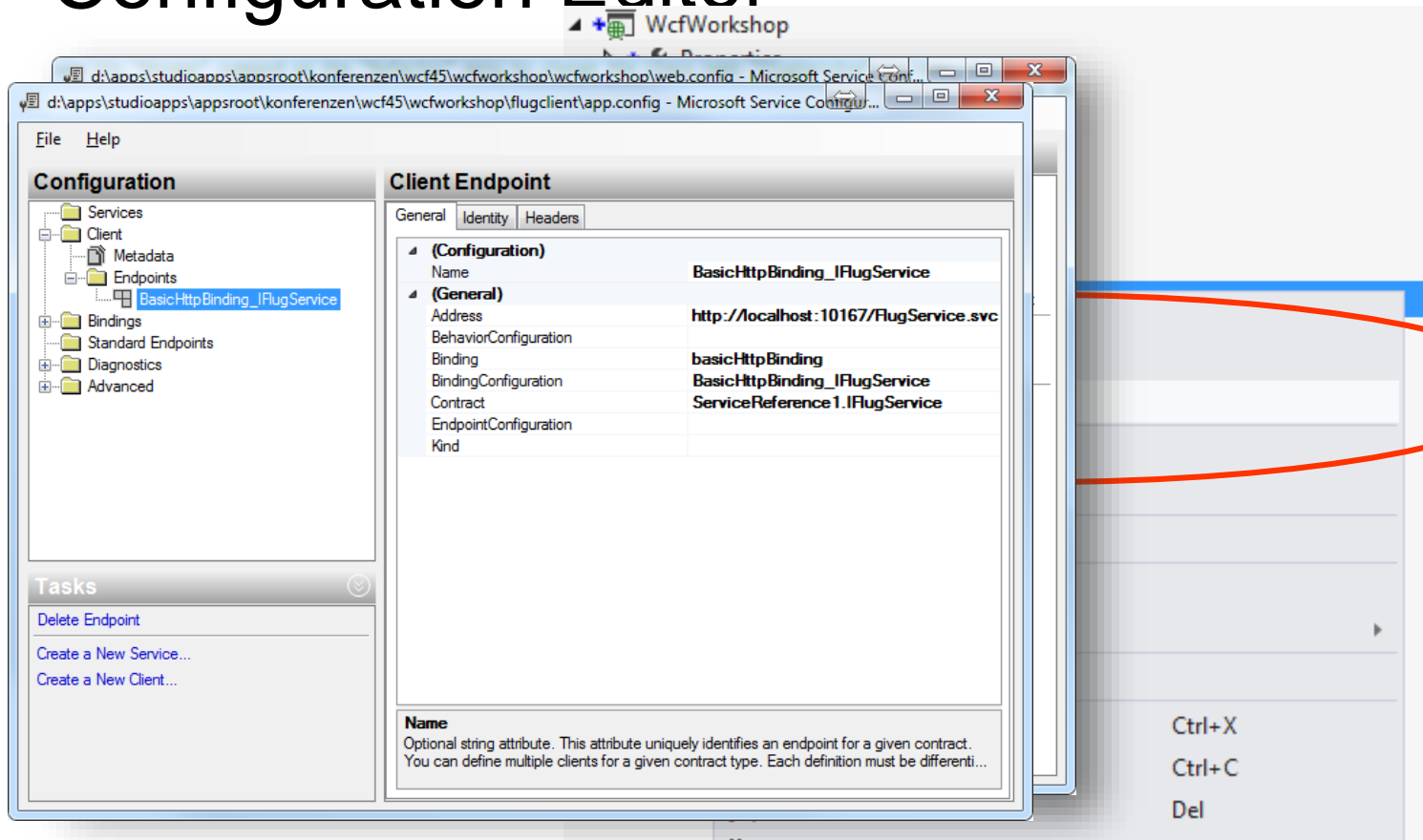
# Werkzeuge

Wie man "WCF" macht

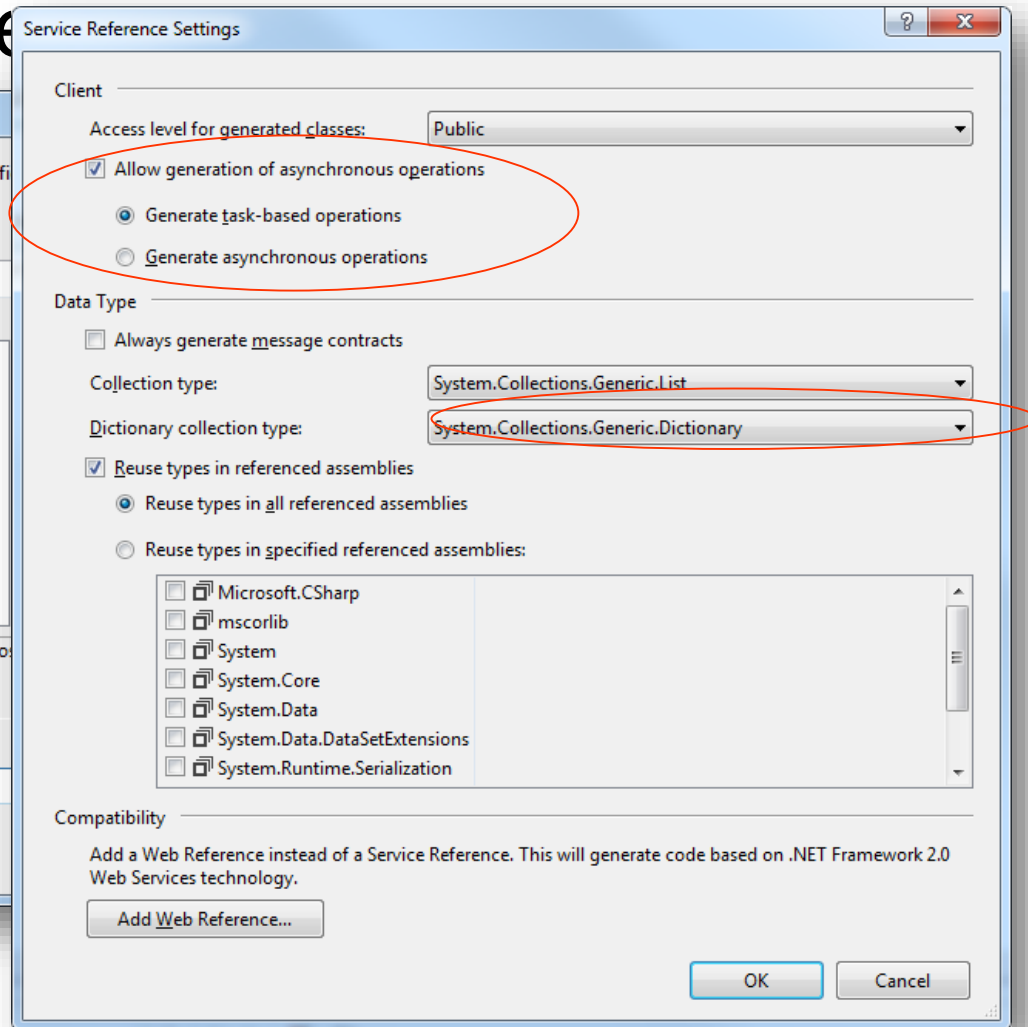
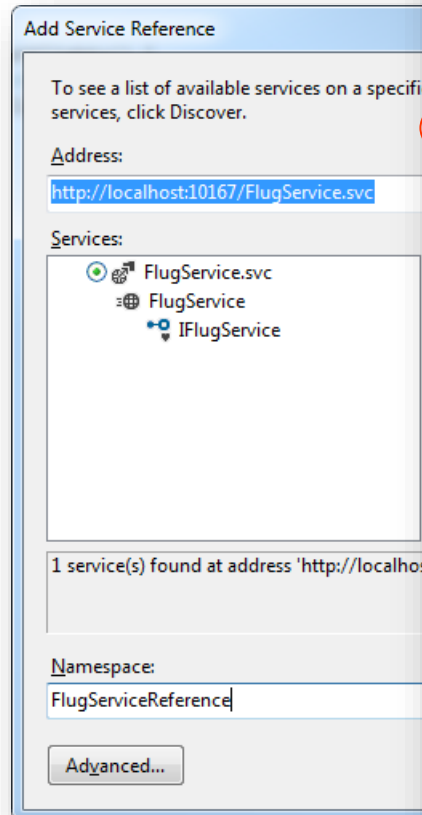
- SvcUtil.exe
  - Code generieren
  - Metadaten extrahieren, importieren, exportieren
  - Konfiguration erzeugen
  - Funktionsprüfung
- [SvcUtil.exe](#)

- SvcConfigEditor
- SvcTraceViewer
- FindPrivateKey
- WcfSvcHost / WcfTestClient
- Makecert
- Certmgr
- MMC für Zertifikate

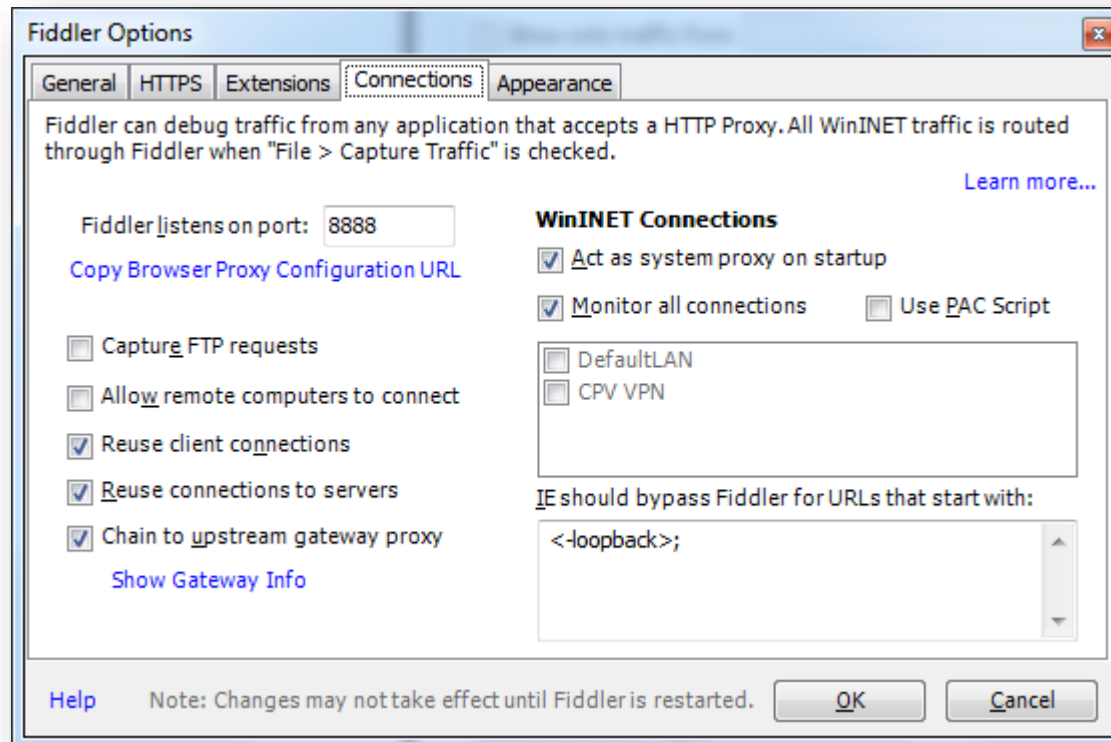
- WCF Configuration Editor




- Proxy erstellen



- Protokoll Sniffer mit Fiddler (wenn HTTP)
  - Fiddler als Reverse Proxy  
(<http://www.fiddler2.com/fiddler/help/reverseproxy.asp>)



- Fiddler als Reverse Proxy → WCF anpassen
  - Server Config unverändert



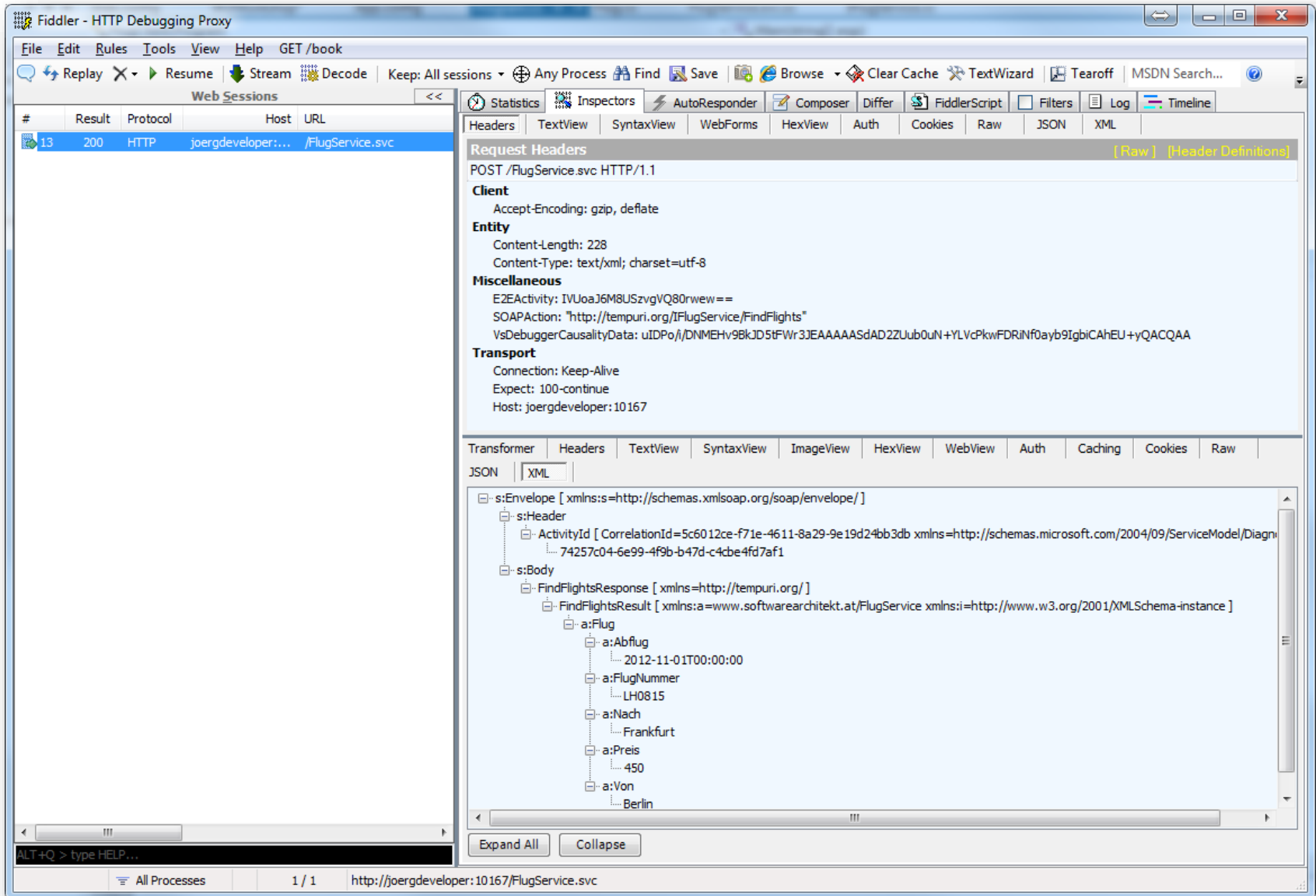
```
<system.net>
  <defaultProxy>
    <proxy bypassonlocal="False" usesystemdefault="True" proxyaddress="http://localhost:8888" />
  </defaultProxy>
</system.net>
```

## – Problem mit localhost

- hosts-Datei: 127.0.0.1 machinenname
- IIS:
  - Web auf Projekt, Pool-Identity auf Admin, Anonym | Edit | Pool User
  - Portnummer, z.B. 10167 (egal, Beispiel), → Browse svc-File
- Fiddler Rule:

```
static function OnBeforeRequest(oSession: Session) {
  if (oSession.host.toLowerCase() == "machine:8888")
    oSession.host = "machine:10167";
}
```





- Tracing

```
<system.diagnostics>
  <sources>
    <source name="System.ServiceModel" switchValue="Information,ActivityTracing"
      propagateActivity="true">
      <listeners>
        <add name="xml" />
      </listeners>
    </source>
    <source name="System.ServiceModel.MessageLogging">
      <listeners>
        <add name="xml" />
      </listeners>
    </source>
  </sources>
  <sharedListeners>
    <add initializeData="C:\logs\TracingAndLogging-service.svclog"
      type="System.Diagnostics.XmlWriterTraceListener" name="xml" />
  </sharedListeners>
  <trace autoflush="true" />
</system.diagnostics>
```

## • Tracing

```
<system.diagnostics>
  <sources>
    <source name="System.ServiceModel"
      propagateActivity="true"
      <listeners>
        <add name="xml" />
      </listeners>
    </source>
    <source name="System.ServiceModel.MessageContract"
      <listeners>
        <add name="xml" />
      </listeners>
    </source>
  </sources>
  <sharedListeners>
    <add initializeData="System.ServiceModel.XmlSerializerTraceListener"
      type="System.ServiceModel.XmlSerializerTraceListener, System.ServiceModel, Version=3.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  </sharedListeners>
  <trace autoflush="true" />
</system.diagnostics>
```

Microsoft Service Trace Viewer - e:\tracingandlogging-service.svclog

File Edit View Activity Help

Look For: Search In: None Level: All Filter Now Clear

Find What: Look In: All Activities Find

Group By: (None) Create Custom Filter Activity - Execute 'WcfWorkshop.IFlugService.FindFlights'.

Activity	# Traces	Description	Level	Thread ID	Process Na...	Time
000000000000	9	From: 7c7a9ae2fd42	Transfer	9	iisexpress	01.11.2012 16:56:39...
Construct Serv...	6	Activity boundary.	Start	9	iisexpress	01.11.2012 16:56:39...
Open Service...	11	To: 7c7a9ae2fd42	Transfer	9	iisexpress	01.11.2012 16:56:39...
Listen at 'http://lo...	9	Activity boundary.	Stop	9	iisexpress	01.11.2012 16:56:39...
Receive bytes on ...	9					
Processing messa...	5					
Process action 'htt...	7					
7c7a9ae2fd42	5					
Execute 'WcfWor...	4					
Close ServiceHost...	11					

Formatted XML

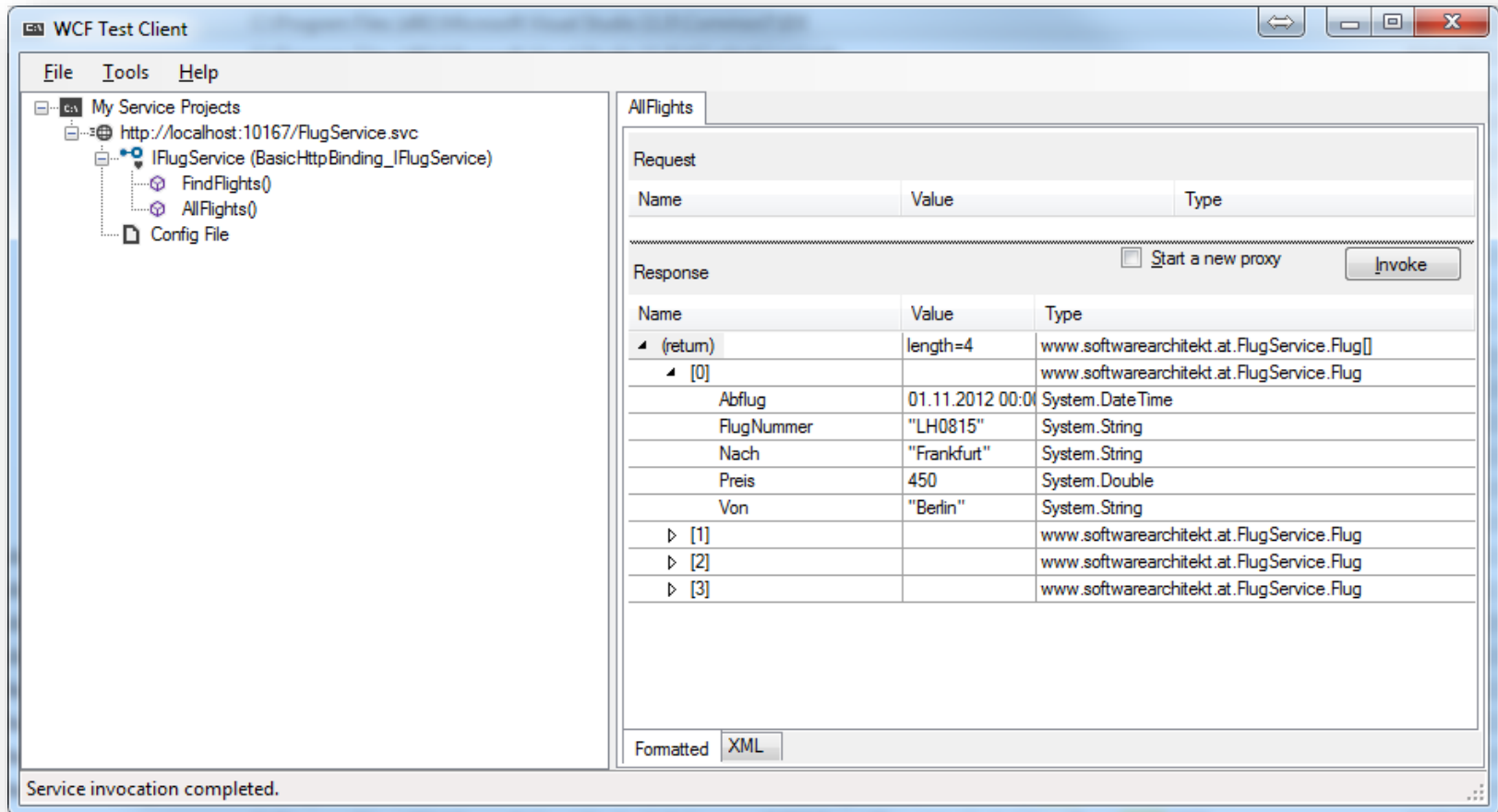
Options

Basic Information

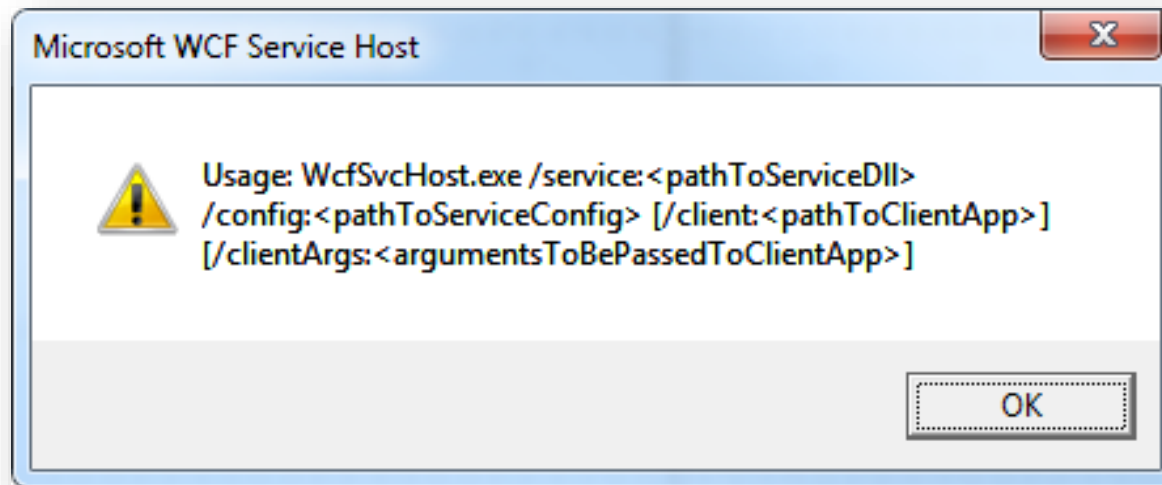
Name	Value
Activity ID	{35623bfa-3e87-4843-9e8b-7c7a9ae2fd42}
Related Activity Name	Execute 'WcfWorkshop.IFlugService.FindFlights'.
Time	2012-11-01 16:56:39.2983
Level	Transfer
Source	System.ServiceModel
Process	iisexpress
Thread	9

Activities: 10 Traces: 60

- %programfiles%\M[...]Studio 11.0\Common7\IDE\WcfTestClient.exe



- %programfiles%\M[...]Studio 11.0\Common7\IDE\WcfSvcHost.exe
- Kommandozeilenhost





**DEMO**

**Servicelayer einrichten,  
Demoservice testen**

# WCF

Programmierung von Diensten



- **IService.cs**
  - Schnittstellen, die Dienst, Daten oder Nachrichtenkontrakte definiert
- **Service.cs**
  - Implementierung der Funktionalität des Dienstes
- **Service.svc**
  - Markup (nur eine Zeile) zum Hosting im IIS
- Konfigurationsdateien zum deklarativen Beschreiben der Attribute, Endpunkte und Richtlinien
  - App.config (self hosted) enthält Markup des Dienstmodells
  - Web.config (im IIS) mit Web Server Richtlinien (Markup) und Markup des Dienstmodells wie in der App.config



# <system.serviceModel>-Markup

```
<system.serviceModel>
  <services>
    <service name="mySvcName" behaviorConfiguration="...">
      <endpoint address="" binding="wsHttpBinding"
        contract="myNamespace.myInterface" />
      <!-- can expose additional endpoints here -->
      <endpoint address="mex" binding="mexHttpBinding"
        contract="IMetadataExchange" />
    </service>
  </services>
  <behaviors>
    <serviceBehaviors>
      <behavior name="myNamespace.mySvcNameBehavior">
        <serviceMetaData httpGetEnabled="true" />
        <serviceDebug includeExceptionDetailInFaults="false" />
      </behavior>
    </serviceBehaviors>
  </behaviors>
</system.serviceModel>
```

- Channels sind eine Methode zum transportieren von Nachrichten. Sie bieten:
  - Transportprotokolle durch Bindungen
    - Http, wsHttp, Tcp, MSMQ, named pipes
  - Encoding und Verschlüsselung
  - Zuverlässigkeitsmaßnahmen (reliable sessions)
  - Kommunikationsmodi
    - Simplex, Duplex, Senden und Warten
  - Sicherheitsmodi

- Channel-Protokolle bestimmen die Interoperabilität mit anderen Plattformen:
  - BasicHttpBinding → universelle Interoperabilität
  - wsHttpBinding → Plattformen mit WS-Erweiterungen
  - netTcpBinding → .NET auf beiden Seiten
  - MSMQ → WCF und ältere WCF Plattformen

- Self Host
  - Service hat einen Haupteintrittspunkt
  - eigener Prozess
- Internet Information Server (IIS)
  - BasicHttpBinding bietet Webdienste.
  - wsHttpBinding bietet Webdienste mit WS\*-Erweiterungen.
- Windows Activation Service (WAS)
  - Wie Webdienste, erweitert Funktionen des IIS auf Non-HTTP

- Instanziierung:
  - **Singleton**: Eine Instanz für alle Clients
  - **Per call**: Eine Instanz pro Dienstaufwurf
  - **Private session**: Eine Instanz pro Client Session
  - **Shared session**: Eine Instanz pro Session, wird zwischen den Clients geteilt
- Concurrency-Modelle für Instanzen:
  - **Single**: Ein Thread pro Instanz
  - **Multiple**: Mehr als ein Thread erlaubt
  - **Reentrant**: Threads machen rekursive Aufrufe ohne Dead-Lock

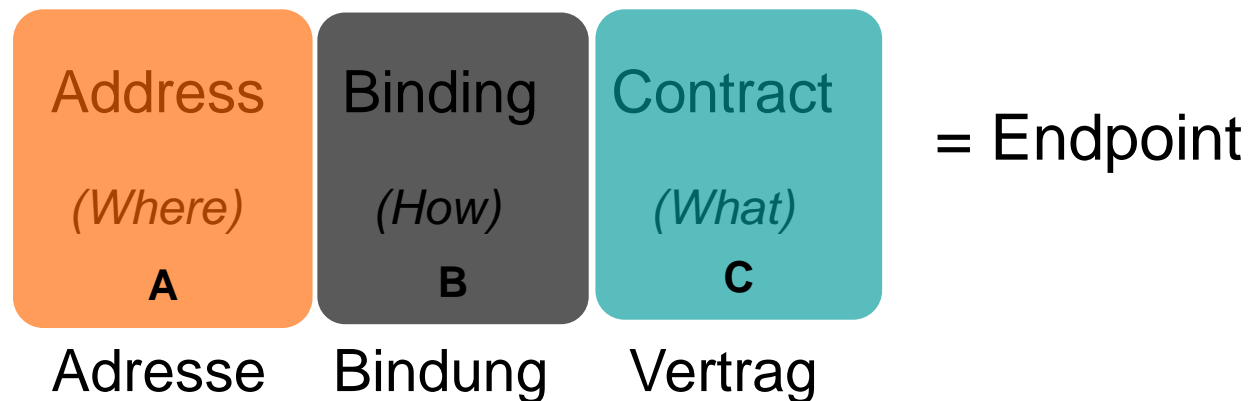
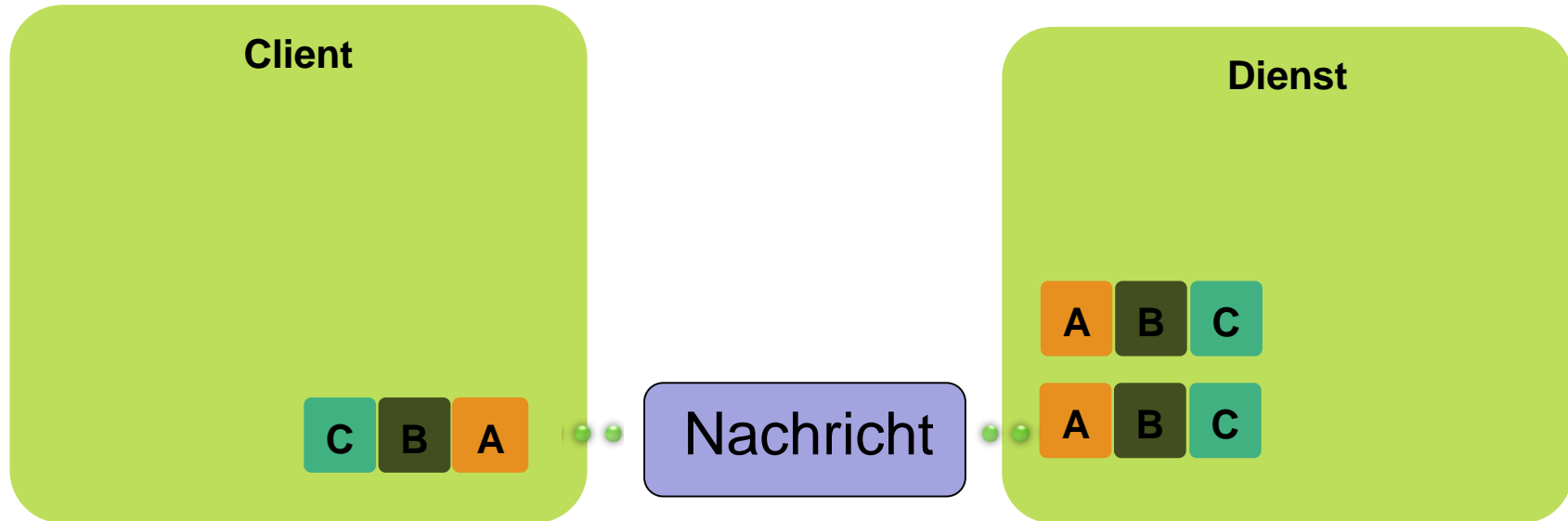
- Throttling:
  - Begrenzt die Anzahl der Nachrichten, Instanzen **oder** Threads, die gleichzeitig verarbeitet werden können
- Error Handling:
  - Optionen zum Behandeln von Fehlern (Framework, Client)
- Metadata:
  - Selbstbeschreibender Endpunkt, MEX-Endpunkte
- Lifetime:
  - Laufzeit einer Session, Dienstoperationen zum Initiieren der Session und andere Beendigungsbedingungen
- Security:
  - Echtheit, Integrität, Authentifizierung, Autorisierung, Überwachung u.ä. für Nachrichten

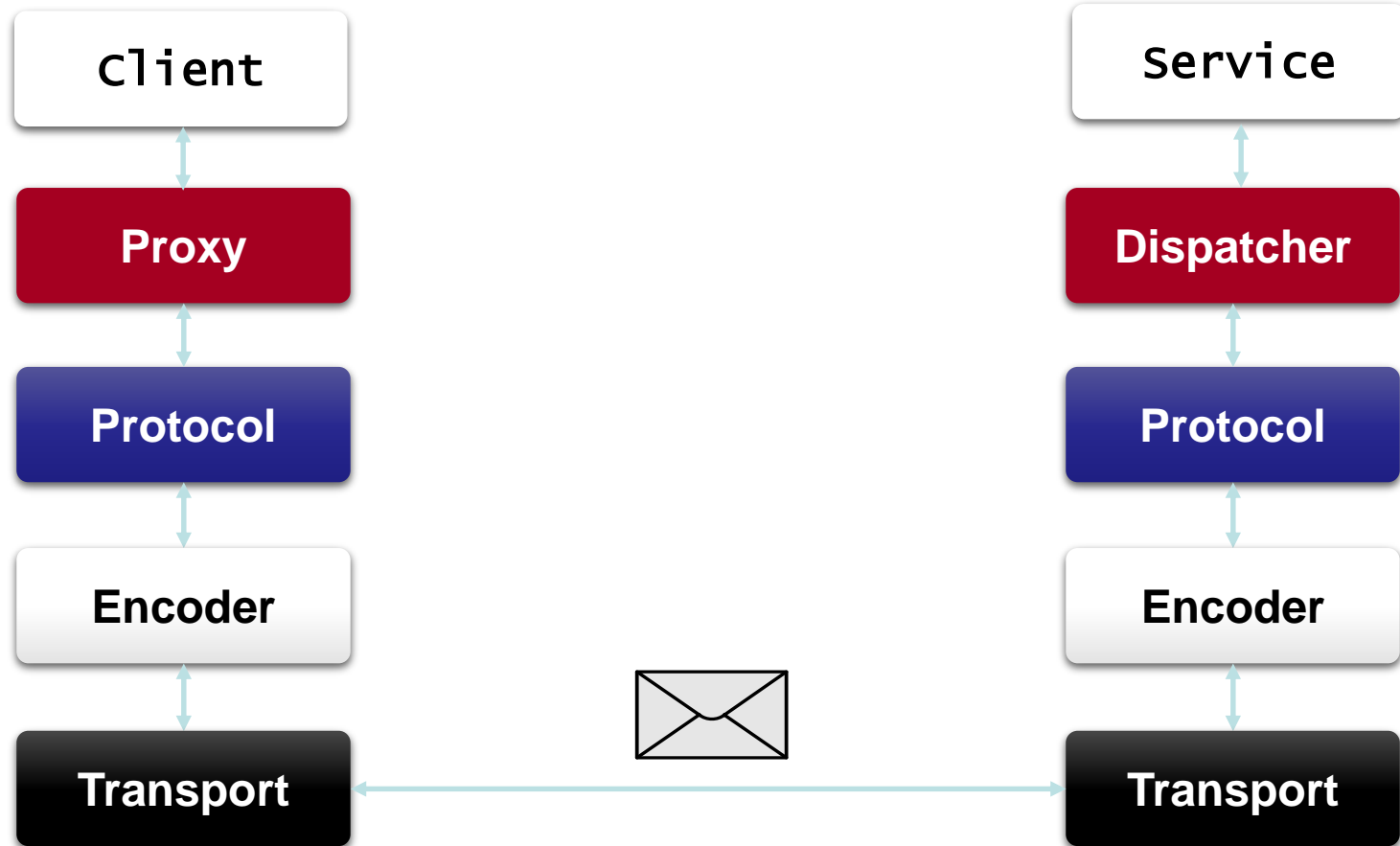
Service	Class	Attribute
Service contract	interface	[ServiceContract]
Service operation	method	[OperationContract]
Implementation	class	[ServiceBehavior] Derive from contract interface
Implementation	method	[OperationBehavior]
Data Contract	class	[DataContract] class [DataMember] member
Message Contract	interface	[MessageContract] interface [MessageHeader] member [MessageBody] member

# Architektur

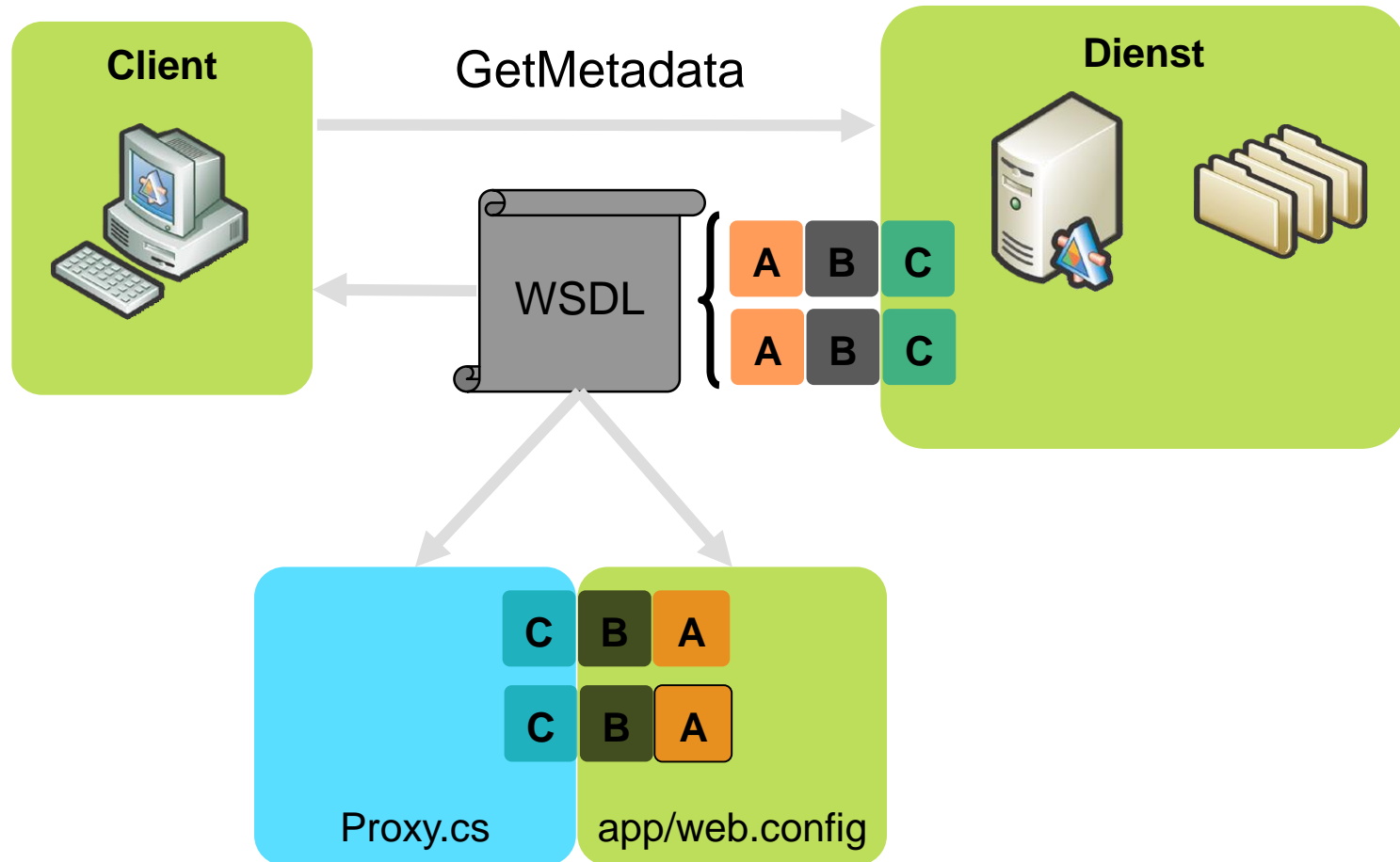
Aufbau einer WCF-basierten Architektur







# Client-Konfiguration



*svcutil.exe http://localhost:port/Dienst/*

```
[DataContract]
public class Order
{
    [DataMember]
    public int orderID;
    [DataMember]
    public int partNumber;
    [DataMember]
    public int price;
    [DataMember]
    private string info;
}
```

```
[DataContract]
public class Query
{
    [DataMember]
    public int orderID;
}
```

```
[ServiceContract]
public interface IOrderService
{
    [OperationContract]
    void Process(Order o);

    [OperationContract]
    Info GetInfo(Query q);
}
```

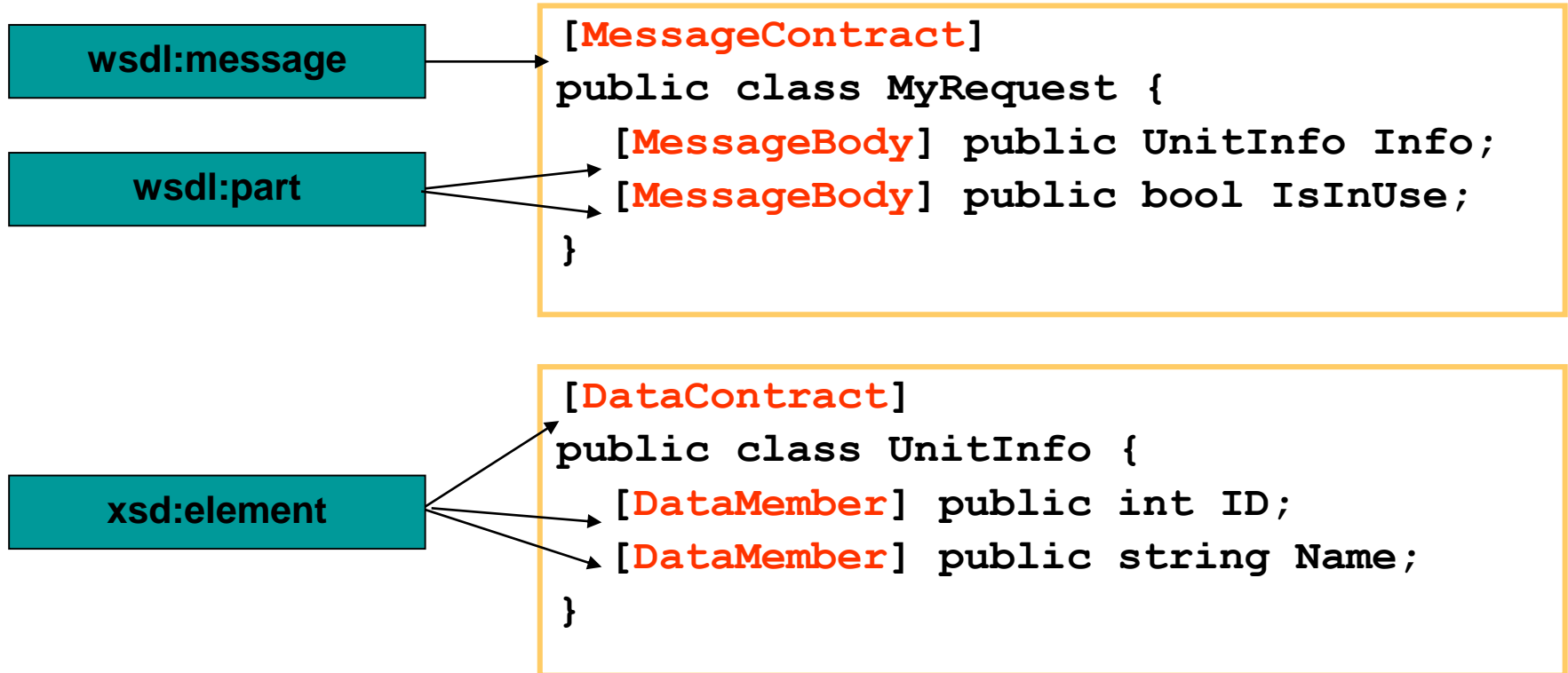
- ServiceContract führt Operationen zusammen

wsdl:portType

wsdl:operation

```
[ServiceContract]
public interface MyContract {
    [OperationContract(
        Action="urn:DoIt",
        ReplyAction="urn:Done")]
    MyReply DoIt(MyRequest request);
}
```

- OperationContract führt Message Contract und Action zusammen



- Action kontrolliert die Zuweisung (Dispatch)

```
[OperationContract(  
    Action = "Foo",  
    ReplyAction = "FooResponse")]  
Message Foo(Message request);
```

- "\*" ist für alle Aktionen zuständig

```
[OperationContract(Action = "*")]  
void Dispatch(Message request);
```

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <services>
      <service type="HelloService"
        <endpoint address="http://localhost/HelloService"
          binding="basicHttpBinding"
          contract="IHello" />
      </service>
    </services>
  </system.serviceModel>
</configuration>
```



# Bindungen (primäre, Auswahl)

	Interop	Security	Session	Transactions	Duplex	Streaming
basicHttpBinding	BP1.1*	T				
wsHttpBinding	WS	T   S	X	X		
wsDualHttpBinding	WS	T   S	X	X	X	
netTcpBinding	.NET	T   S	X	X	X	O
netNamedPipeBinding	.NET	T   S	X	X	X	O
netMsmqBinding	.NET	T   S	X	X		
netPeerTcpBinding	.NET	T   S			X	

T = Transport Security | S = WS-Security | O = One-Way Only

\* Basic Profile 1.1 (WS-I BP 1.1) → <http://www.ws-i.org/profiles/basicprofile-1.1-2004-08-24.html>



**DEMO**

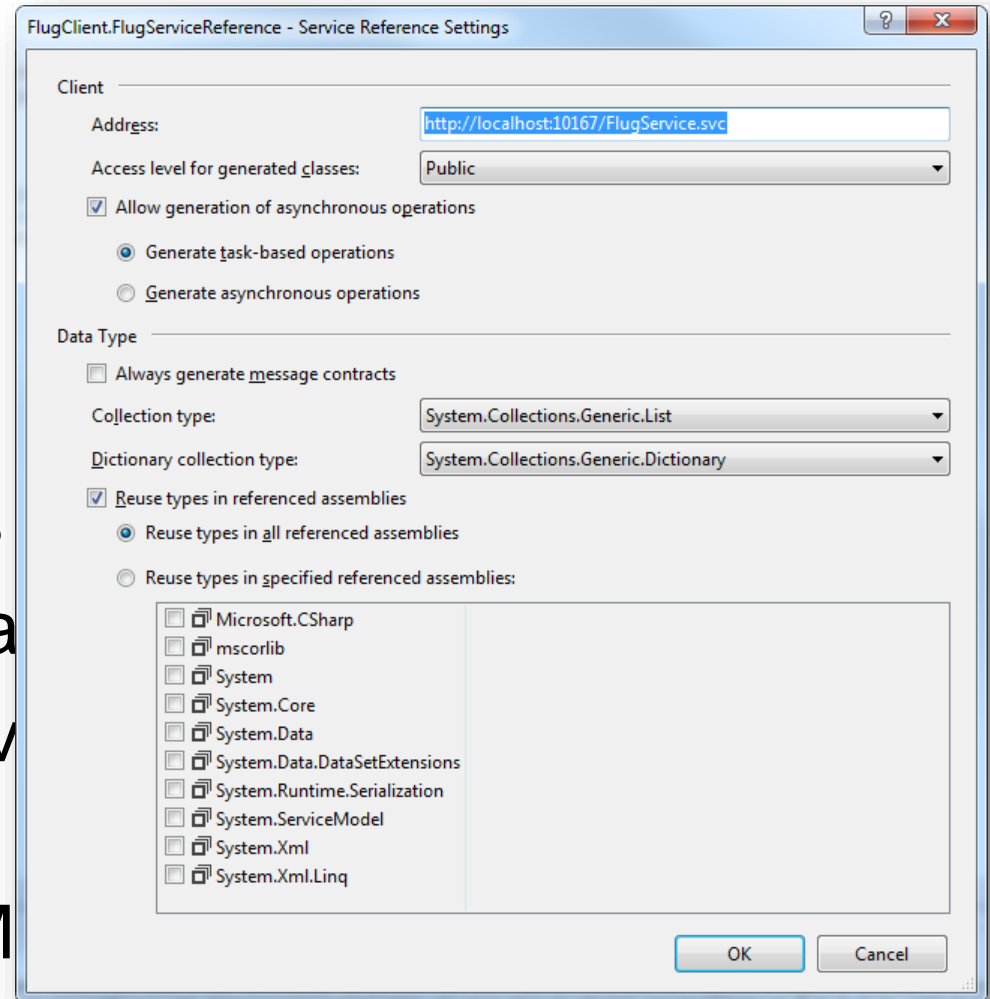
**Dienst für den Client  
entwickeln und testen**

# WCF

# Clientprogrammierung

- Proxy-Generierung:
  - Visual Studio
  - svcutil
  - Laufzeit

- Visual Studio
- Optionen:
  - Access Level
  - Async
  - Message Contracts
  - Collection / Dictionary
  - Wiederverwenden v
  - Aktualisieren
  - Kompatibilität (ASM



- Developer Command Prompt
- Die wichtigsten Optionen:
  - **/out:** file
  - **/config:** file
  - **/language:** CS|VB
  - **/namespace:** <ns>
  - **/async**
  - **/references:** path

Code-Datei  
Config-Datei  
Sprache  
Namensraum  
Begin/End Async-Methoden  
Referenzen

- Andere Funktionen:
  - Metadaten exportieren
  - Typ-Generierung
  - Service-Validierung

- Anwendung

```
svcutil http://server/endpoint.svc /l:CS /o:proxy.cs /config:app.config
```

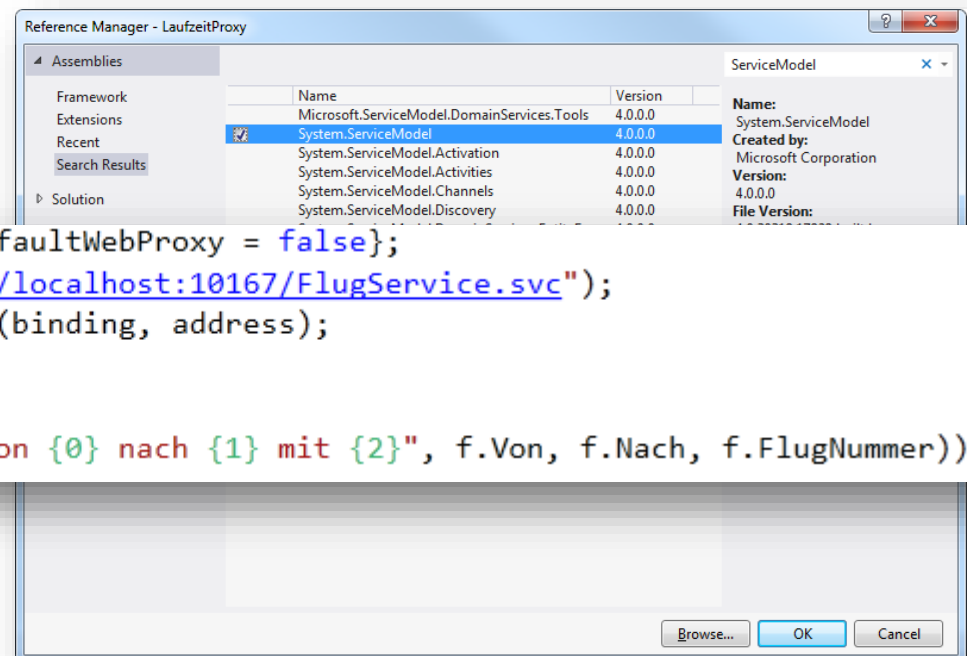
```
//-----  
// <auto-generated>  
//   This code was generated by a tool.  
//   Runtime Version:4.0.30319.17929  
//  
//   Changes to this file may cause incorrect behavior and will be lost if  
//   the code is regenerated.  
// </auto-generated>  
//-----  
  
[assembly: System.Runtime.Serialization.ContractNamespaceAttribute("www.softwarearchitekt.at/FlugService", ClrNamespace="www.softwarearchitekt.at/FlugService")]  
namespace www.softwarearchitekt.at.FlugService  
{  
    using System.Runtime.Serialization;  
  
    [System.Diagnostics.DebuggerStepThroughAttribute()]  
    [System.CodeDom.Compiler.GeneratedCodeAttribute("System.Runtime.Serialization", "4.0.0.0")]  
    [System.Runtime.Serialization.DataContractAttribute(Name="Flug", Namespace="www.softwarearchitekt.at/FlugService")]  
    public partial class Flug : object, System.Runtime.Serialization.IExtensibleDataObject  
    {  
  
        private System.Runtime.Serialization.ExtensionDataObject extensionDataField;  
  
        private System.DateTime AbflugField;  
  
        private string FlugNummerField;  
  
        private string NachField;  
  
        private double PreisField;  
  
        private string VonField;  
    }  
}
```



- Benutzung der ChannelFactory
  - Zugriff auf Typen für ServiceContract (I...) und DataContract
  - Binding
  - Endpoint

Channel

```
var binding = new BasicHttpBinding {UseDefaultWebProxy = false};  
var address = new EndpointAddress("http://localhost:10167/FlugService.svc");  
var cf = new ChannelFactory<IFlugService>(binding, address);  
var proxy = cf.CreateChannel();  
var flights = proxy.AllFlights();  
flights.ForEach(f => Console.WriteLine("Von {0} nach {1} mit {2}", f.Von, f.Nach, f.FlugNummer));
```





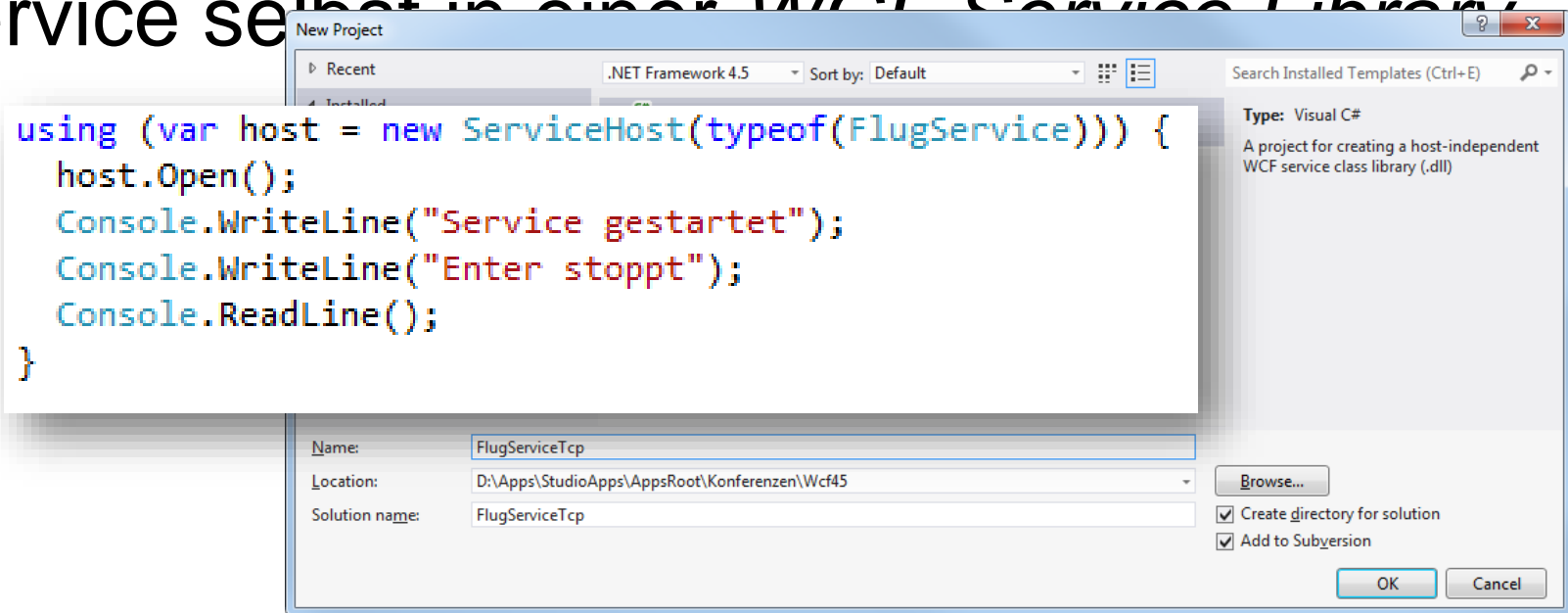
**DEMO**

**Client-Proxy erstellen**

# Hosting

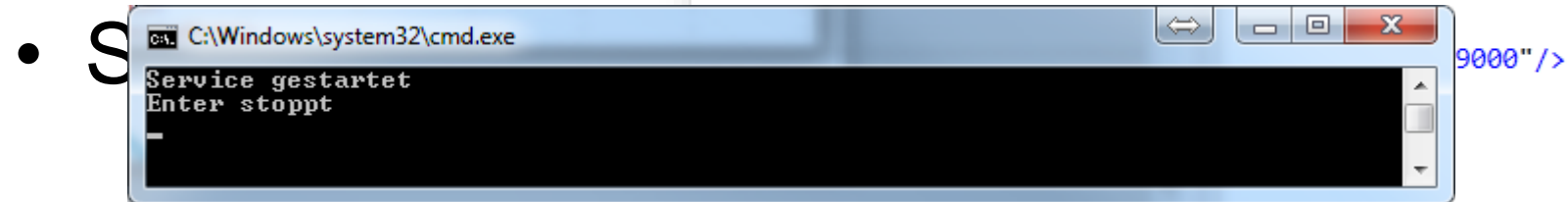
- Zu Testzwecken:
  - Entwicklungswebserver
  - IIS Express
  - WcfSvcHost
  - Self Hosting (Winforms, WPF, Konsole, Windows Systemdienst)
- Produktion:
  - IIS
  - IIS & AppFabric
  - Self Hosting

- z.B. für andere Bindungen, die VS oder IIS Express nicht unterstützen
- *Consolen*-Applikation zum Host
- Service selbst in einer *WCF Service Library*



- Konfiguration Host
  - Kein http
  - Bindung netTcp
  - Mex Endpunkt

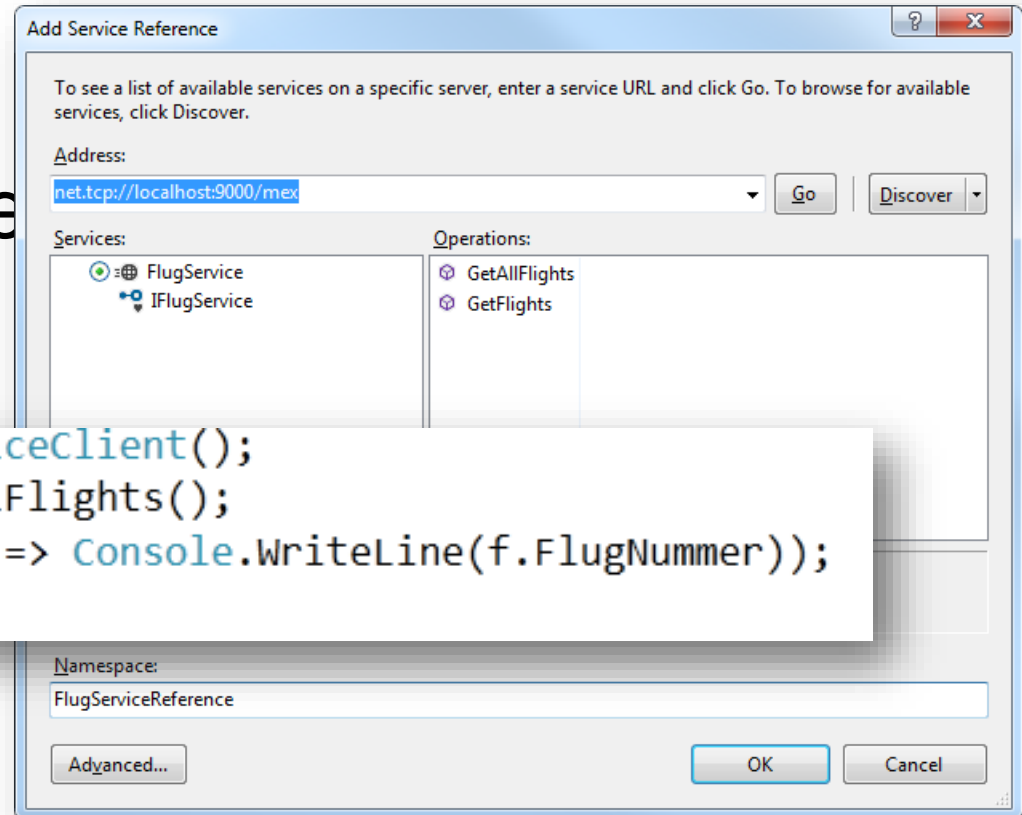
```
<system.serviceModel>
  <services>
    <service name="FlugServiceTcp.FlugService">
      <endpoint
        address=""
        binding="netTcpBinding"
        contract="FlugServiceTcp.IFlugService" />
      <endpoint
        binding="mexTcpBinding"
        address="mex"
        contract="IMetadataExchange" />
    </service>
  </services>
</system.serviceModel>
```



```
</services>
<behaviors>
  <serviceBehaviors>
    <behavior>
      <serviceMetadata httpGetEnabled="false"/>
      <serviceDebug includeExceptionDetailInFaults="true"/>
    </behavior>
  </serviceBehaviors>
</behaviors>
</system.serviceModel>
```

- Client
- Consolen-App
- ServiceReference auf den mex-Endpunkt

```
var client = new FlugServiceClient();  
var result = client.GetAllFlights();  
result.ToList().ForEach(f => Console.WriteLine(f.FlugNummer));  
Console.ReadLine();
```



- Systemdienst erstellen

```
using System.ServiceModel;

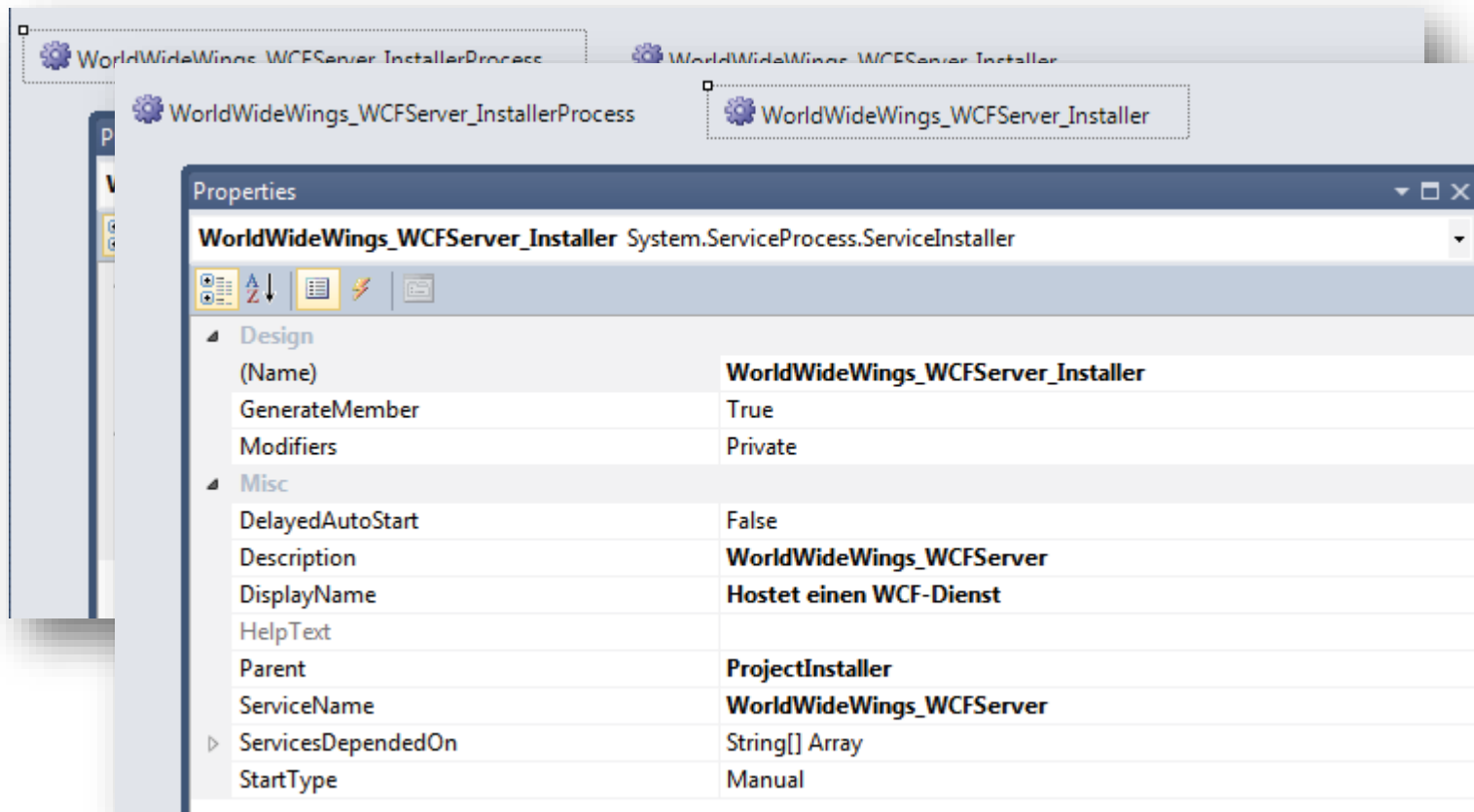
public partial class WorldWideWingsWCFApplicationServer : ServiceBase
{
    public WorldWideWingsWCFApplicationServer()
    {
        InitializeComponent();
    }
    protected override void OnStart(string[] args)
    {
    }
    protected override void OnStop()
    {
    }
}
```



- Systemdienst debuggen

```
static void Main()
{
    ServiceBase[] ServicesToRun;
    ServicesToRun = new ServiceBase[]
    {
        new WorldWideWingsWCFApplicationServer()
    };
    ServiceBase.Run(ServicesToRun);
}
```

- ServiceProcessInstaller



- Installation
  - Installutil.exe
  - Powershell

```
Administrator: Visual Studio Command Prompt (2010)
H:\www\Anwendungsserver\WCF_Server\bin\Debug>installutil WWWings_WCFServer.exe
Microsoft (R) .NET Framework Installation utility Version 4.0.30319.1
Copyright (c) Microsoft Corporation. All rights reserved.

Running
Beginni
See the
The fil
Install
Affecte
logt
logf
asse
Install
Service
Creatin

The Ins
See the
The fil
Committ
Affecte
logtoconsole =
logfile = H:\www\Anwendungsserver\WCF_Server\bin\Debug\WWWings_WCFServer.InstallLog
assemblypath = H:\www\Anwendungsserver\WCF_Server\bin\Debug\WWWings_WCFServer.exe

The Commit phase completed successfully.
The transacted install has completed.
H:\www\Anwendungsserver\WCF_Server\bin\Debug>_
```

Administrator: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe

```
PS P:\> Get-Service *WCF*
```

Status	Name	DisplayName
Stopped	WorldWideWingsW...	World Wide Wings WCF Application Se...

```
PS P:\> Start-Service *WCF*
```

```
PS P:\> _
```