

## Kapitel 32

# ADO.NET-Datendienste (Data Services)

### In diesem Kapitel:

Architektur der Datendienste	694
Anlegen eines Datendienstes	696
Testen eines Datendienstes	697
Abfragesyntax der Datendienste	702
Nutzung eines Datendienstes in .NET-Anwendungen	702
Generieren der Zugriffsklassen	702
Abfrage ohne LINQ	703
LINQ to ADO.NET Data Services	704
Datenänderungen	705
AJAX-Webseiten als WCF-Clients (AJAX Client Library for ADO.NET Data Services)	707
Weitere Möglichkeiten	710

ADO.NET-Datendienste (Codename *Astoria*) sind eine Zusatzbibliothek (eingeführt in .NET 3.5 Service Pack 1, auch verfügbar in Silverlight ab Version 2.0) zum Lesen und Ändern einer Datenmenge (häufig eine Datenbank) über XML-Webservices. Als Webservices werden allerdings nicht SOAP-basierte Webservices, sondern nur REST-basierte Webservices mit HTTP-Standardverben GET, POST, PUT und DELETE unterstützt.

---

**HINWEIS** ADO.NET-Datendienste basieren auf der Windows Communication Foundation (WCF). Da ADO.NET-Datendienste sehr stark von der WCF abstrahieren, ist für die konkrete Nutzung der Datendienste gar kein Wissen über die WCF notwendig. Wenn Sie die Hintergründe der Datendienste verstehen wollen, sollte Sie das Kapitel »Windows Communication Foundation« lesen.

---

ADO.NET Data Services sind unabhängig von dem Datenspeicher. Die Datenmenge wird vorgegeben durch ein Modell im ADO.NET Entity Framework (mit Zugriff auf unterschiedliche relationale Datenbanken) oder jede andere Menge, die durch die `System.Linq.IQueryable`-Schnittstelle abgefragt werden kann.

Die Datenübertragung erfolgt via HTTP in der Representational State Transfer (REST)-Semantik. Serialisierungsformate sind JSON und ATOM. ADO.NET Data Services setzen technisch auf der WCF auf. Ein ADO.NET Data Service ist eine erweiterte Form eines WCF-Dienstes.

Clients sind .NET-Anwendungen (unterstützt durch Klassen im .NET-Namensraum `System.Data.Services`) und AJAX-Anwendungen (unterstützt durch JavaScript-Klassen im Namensraum `Sys.Data`). In .NET-Clients kann LINQ eingesetzt werden (LINQ to DataServices), wobei die Abfrage serverseitig zur Ausführung kommt. LINQ to DataServices bietet einen clientseitigen Proxy eines serverseitigen Entity Framework-Datenkontextes. Eine LINQ-Abfrage wird von dem Proxy in einen REST-URL übersetzt. Das per JSON bzw. ATOM übermittelte Ergebnis wird auf Geschäftsobjekte abgebildet. Auch Datenänderungen, Datenlöschungen und das Anfügen neuer Datensätze sind auf diesem Wege möglich.

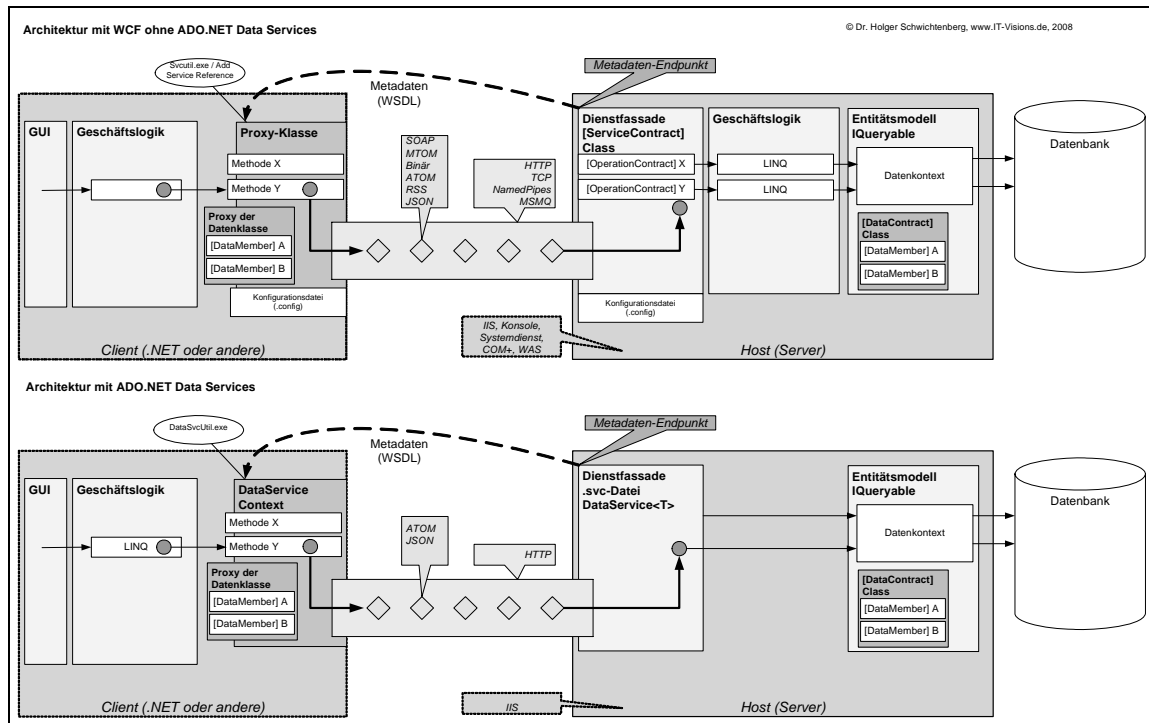
---

**HINWEIS** Über den Namen *ADO.NET Data Services* kann man streiten, da die ADO.NET Data Services nichts mit dem bisherigen Funktionsumfang von ADO.NET gemein haben, weil die Data Services mit Geschäftsobjekten und nicht mit Tabellenstrukturen arbeiten. WCF Data Services wäre ein besserer Name gewesen.

---

## Architektur der Datendienste

Abbildung 32.1 zeigt eine typische Architektur unter Einsatz der ADO.NET Data Services im Vergleich zur direkten Nutzung der WCF. Die anschließende Tabelle listet auf, welche Programmierarbeiten beim Einsatz der ADO.NET Data Services entfallen.



**Abbildung 32.1** Eine typische Architektur unter Einsatz der ADO.NET Data Services im Vergleich zur direkten Nutzung von WCF

Baustein	Bedeutung	WCF direkt	ADO.NET Data Services
Datenbank	Datenspeicher	Selbst erstellt	Selbst erstellt
Datenmodell	Kapselung des Datenspeichers / Objekt-Relationales Mapping	Generiert (z.B. mit dem ADO.NET Entity Framework)	Generiert (z.B. mit dem ADO.NET Entity Framework)
Serverseitige Geschäftslogik	Bereitstellung des Zugriffs auf Datenspeicher in Form von Diensten	Selbst erstellt, Einsatz von LINQ möglich	Entfällt
Dienstfassade	Kapselung der Dienste der Geschäftslogik in WCF-Dienste	Selbst erstellt (hoher Aufwand)	Selbst erstellt (sehr geringer Aufwand)
Metadaten-Endpunkt	Bereitstellung der Metadaten zur Proxygenerierung auf dem Client	Manuelle Konfiguration	Automatische Bereitstellung
Proxy	Proxys für Dienste und Datenklassen	Generiert (mit <i>Add Service Reference</i> in Visual Studio oder <i>svcutil.exe</i> )	Generiert (mit <i>WebData-Gen.exe</i> )
Clientseitige Geschäftslogik	Aufruf der Dienste im Proxy	Selbst erstellt, Einsatz von LINQ möglich	Selbst erstellt
GUI	Darstellung der Daten/Interaktion mit dem Benutzer	Selbst erstellt	Selbst erstellt
Konfigurationsdateien	Festlegung der Kommunikationseigenschaft	Selbst erstellt	Entfällt

**Tabelle 32.1** Vergleich des Implementierungsaufwandes bei den oben dargestellten Architekturen

## Anlegen eines Datendienstes

Zum Anlegen eines ADO.NET Data Service fügt man einem Webprojekt ein Element von der Vorlage *ADO.NET Data Services* hinzu. Daraus entsteht eine WCF-Dienstdatei (Dateinamenserweiterung *.svc*) mit zugehöriger Hintergrundcodedatei.

In der Hintergrundcodedatei ist eine Klasse implementiert, die von der generischen Klasse *DataService* erbt. An diese Stelle muss man die *ObjectContext*-Klasse des mit dem ADO.NET Entity Framework erstellten ORM-Modells als Parameter eintragen:

```
public class WWWingsDataService :  
    System.Data.Services.DataService< WWWingsModel.WWWingsEntities > public
```

Eine Implementierung von Dienstmethoden ist nicht notwendig. Die Dienstmethoden für das Lesen, Anfügen, Ändern und Löschen werden automatisch erzeugt.

Es ist aber notwendig, die Zugriffsrechte zu setzen, denn in der Grundeinstellung sind keine festgelegt. Die Rechte sind in der Methode *InitializeService()* zu vergeben:

```
public static void InitializeService(IDataServiceConfiguration config)  
{  
    config.SetEntitySetAccessRule("*", EntitySetRights.AllRead);  
    config.SetServiceOperationAccessRule("*", ServiceOperationRights.All);  
}
```

**Listing 32.1** Beispiel für die Vergabe voller Zugriffsrechte auf alle Entitäten

```
public static void InitializeService(IDataServiceConfiguration config)  
{  
    config.SetEntitySetAccessRule ("FL_Fluege", ResourceContainerRights.AllRead);  
    config.SetEntitySetAccessRule ("PS_Passagier", ResourceContainerRights.AllRead);  
    config.SetEntitySetAccessRule("GF_GebuchteFluege", ResourceContainerRights.WriteAppend);  
    config.SetServiceOperationAccessRule("*", ServiceOperationRights.All);  
}
```

**Listing 32.2** Beispiel für die Vergabe von vollen Rechten auf der Tabelle *FL\_Fluege*, von Leserechten auf *PS\_Passagier* und dem Recht zum Hinzufügen von Datensätzen für *GF\_GebuchteFluege*

---

**TIPP** Wie bei der WCF üblich, liefert auch ein Data Service standardmäßig keine aufschlussreichen Fehlermeldungen. Wenn man wirklich wissen will, was auf dem Server schief gegangen ist, muss man in *InitializeService()* Folgendes ergänzen: `config.UserVerboseErrors = true`.

---

**TIPP** Man kann einen ADO.NET Data Service auch außerhalb der IIS betreiben, z.B. in einer Konsolenanwendung oder einem Windows-Dienst. Dazu muss man die Klasse `System.ServiceModel.Web.WebServiceHost` unter Angabe des Typs `WWWingsDataService` instanziiieren und in der Konfigurationsdatei einen Endpunkt mit `webHttpBinding` für `System.Data.Service.IRequestHandler` definieren.

**HINWEIS** Im Standardfall liefert der ADO.NET Data Service immer die Daten im ATOM-Format. Die alternative Darreichungsform JSON kann von dem Client beeinflusst werden, indem dieser im HTTP Header im Feld *Accept* den Typ *application/json* anfordert. Das Format ist also keine feste Konfiguration des Dienstes.

## Testen eines Datendienstes

Zum Testen kann man einen ADO.NET Data Service über einen Webbrowser aufrufen (vgl. ASP.NET-basierte Webservices). Beim Aufruf des Dienst-URL ohne weitere Zusätze liefert der Data Service eine Liste aller verfügbaren Entitäten (Abbildung 32.2).

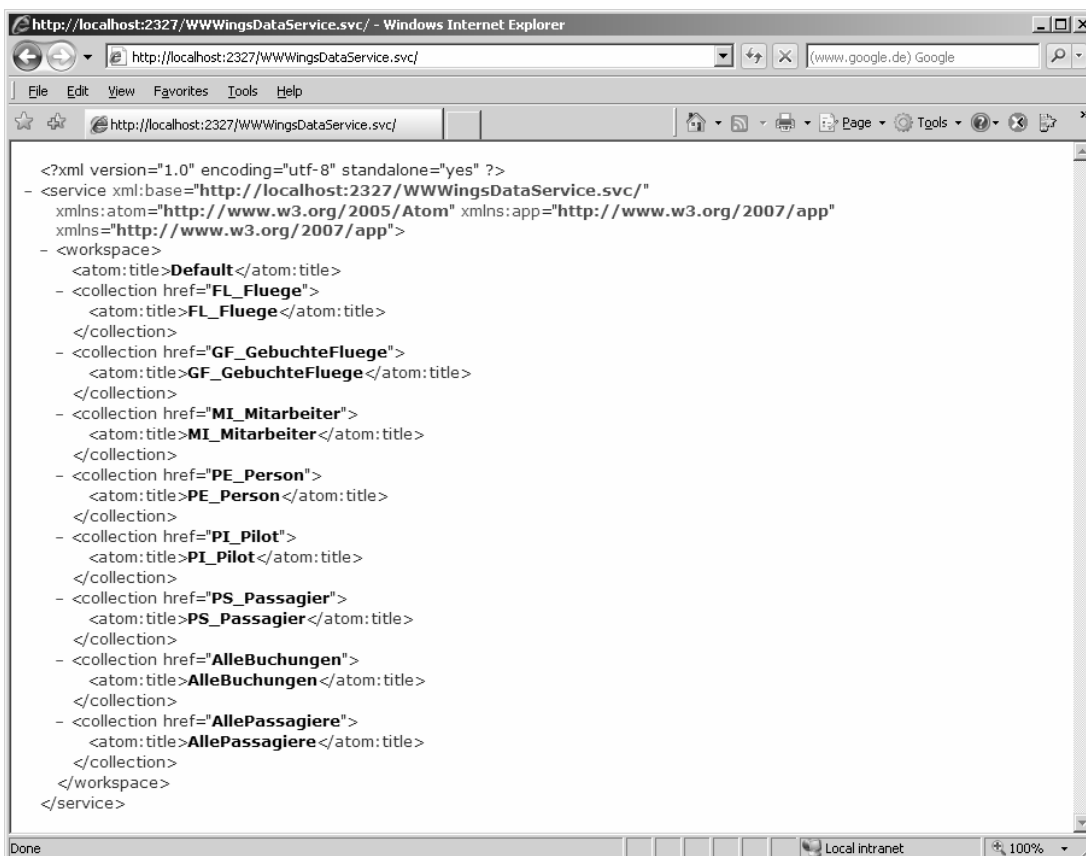


Abbildung 32.2 Aufruf des Dienst-URL

Durch das Anhängen des Namens einer Entität an den URL erhält man alle Instanzen der Entität:

*http://E01/WWWingsDataService.svc/FL\_Fluege*

Da das Standardformat ATOM ist, zeigt der Internet Explorer 7.0 die Daten als *Feed* an. Da dies keine sinnvolle Darstellung ist, muss man die Feedansicht in den Internet Explorer-Eigenschaften deaktivieren (Abbildung 32.3).

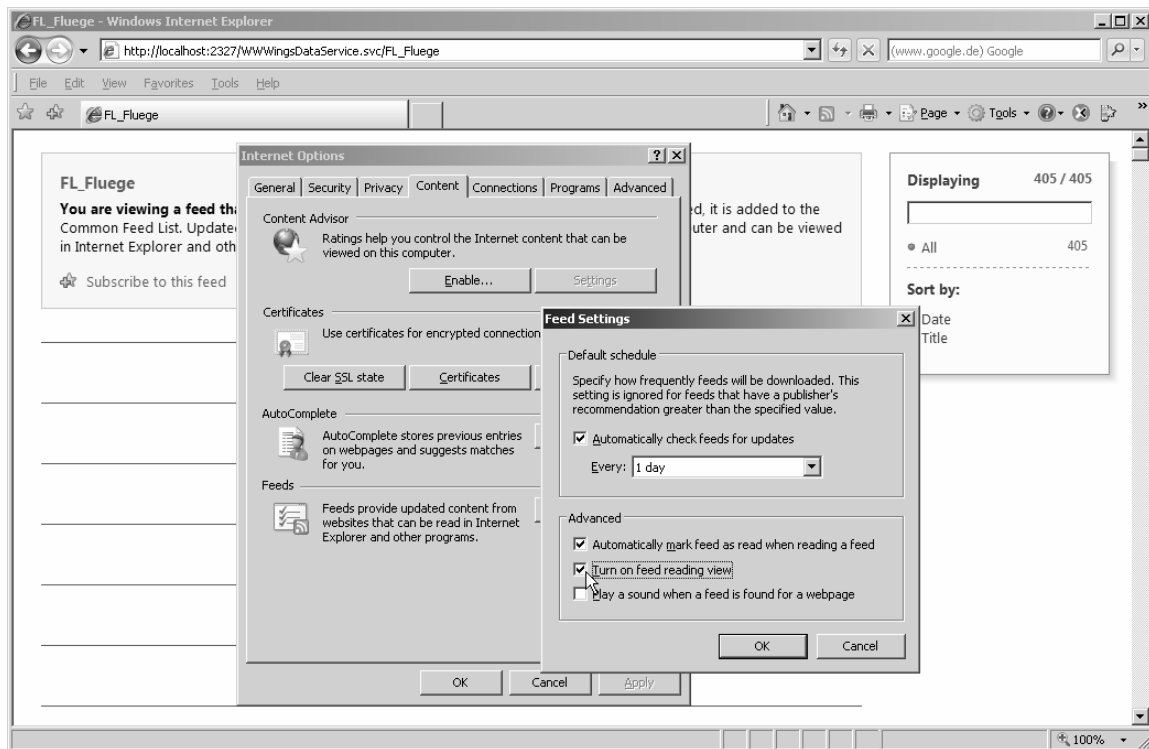


Abbildung 32.3 Deaktivieren der Feedansicht

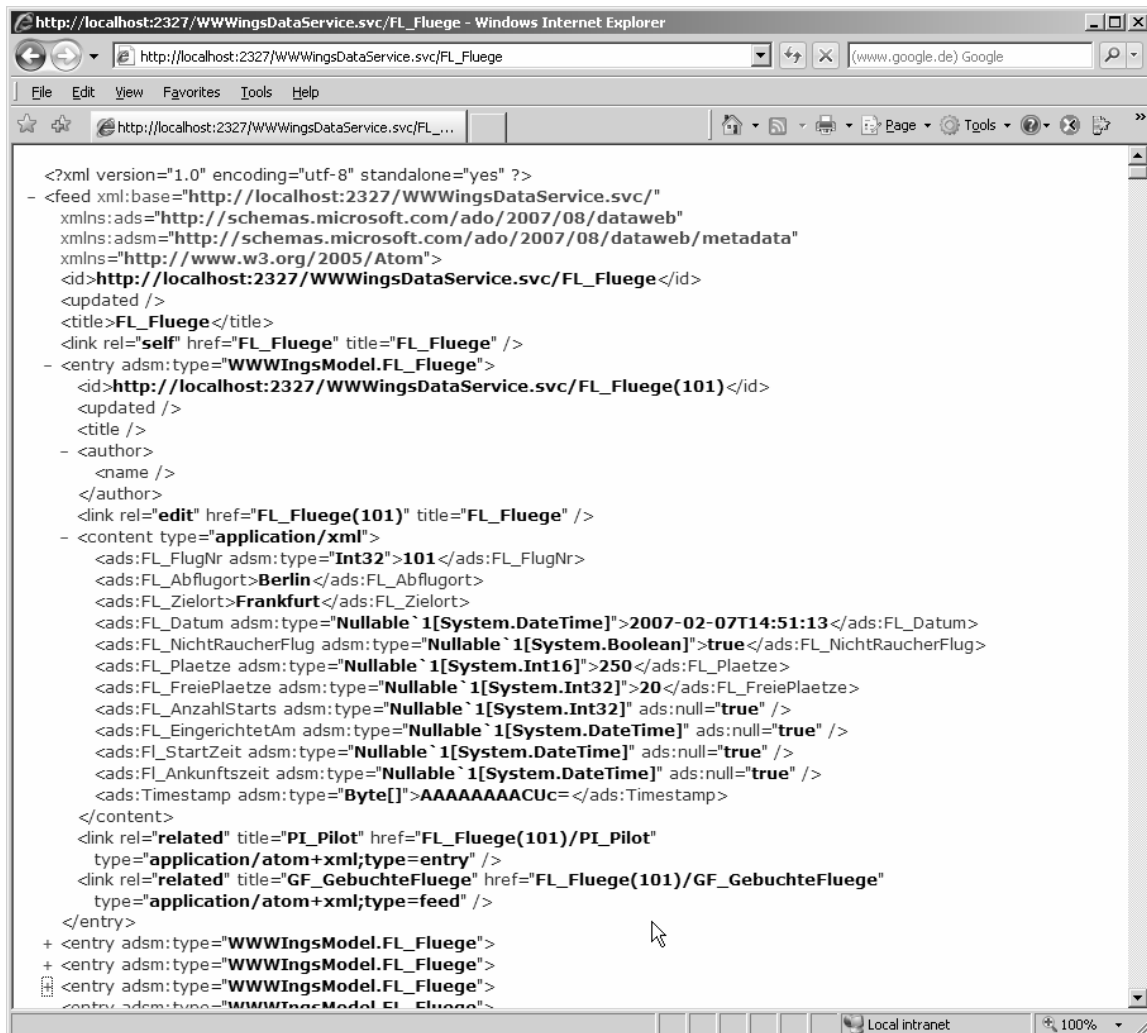


Abbildung 32.4 Anzeige der Instanzen ohne Feedsicht

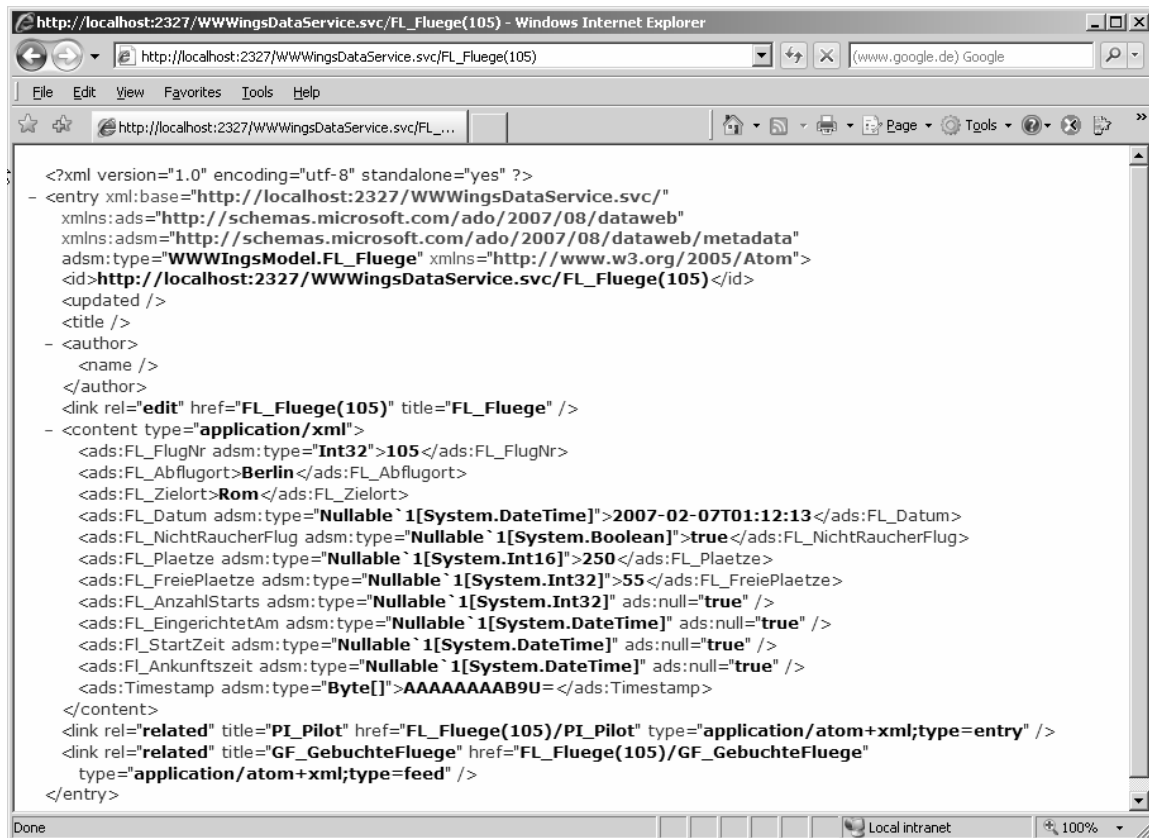


Abbildung 32.5 Anzeige einer ausgewählten Instanz

Die Metadaten, die in der ADO.NET Entity Framework Store Schema Definition Language (SSDL) verfasst sind (nicht zu verwechseln mit der SOAP Service Description Language (SSDL)), kann man abrufen, indem man *\$metadata* an die *.svc*-Datei anhängt:

*http://E01/WWWingsDataService.svc/\$metadata*



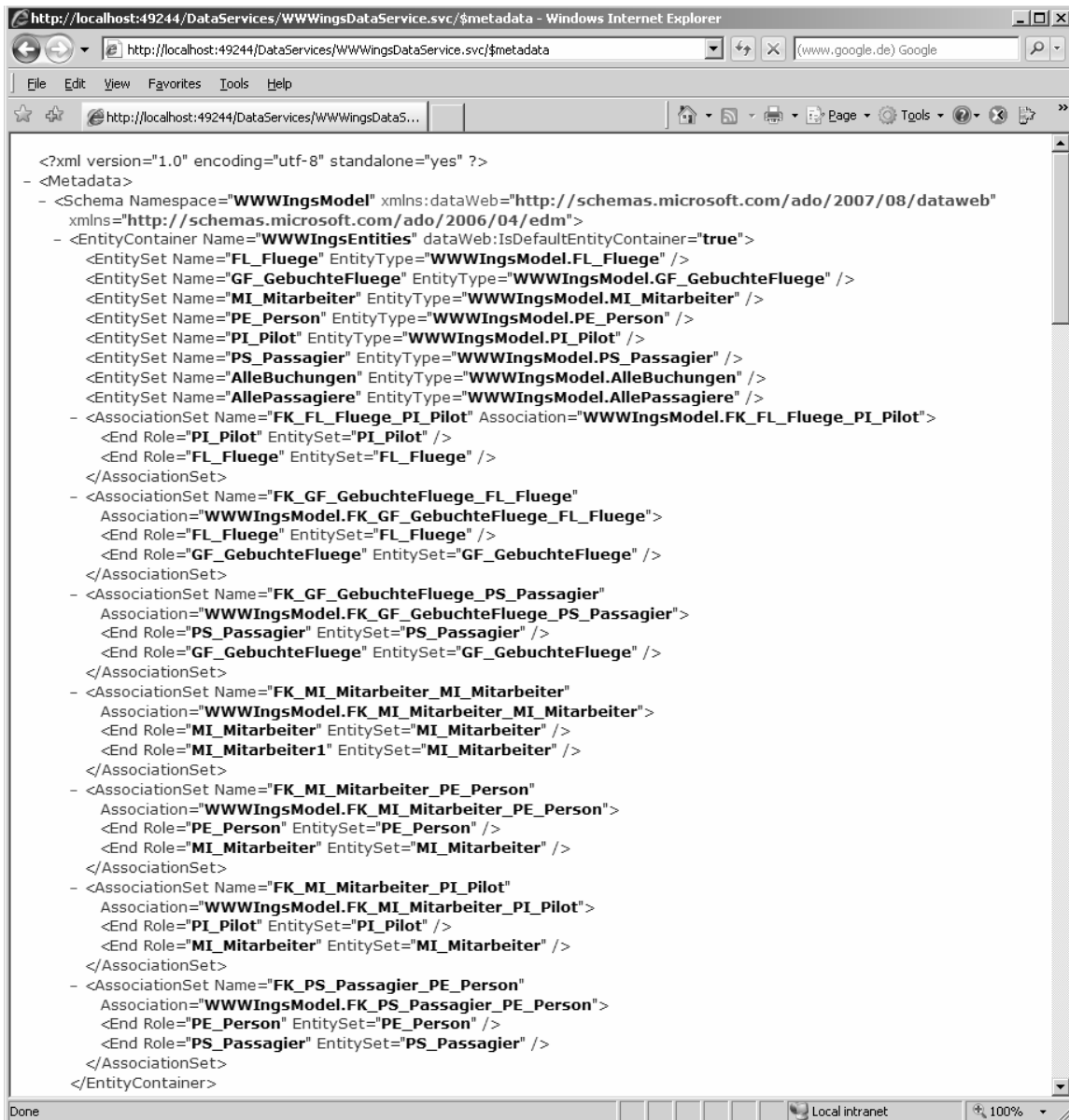


Abbildung 32.6 Anzeige der Metadaten

## Abfragesyntax der Datendienste

Dieser Abschnitt beschreibt die Abfragesyntax der ADO.NET Data Services. Alle Abfragen werden an den URL des Dienstes durch einen Schrägstrich getrennt angehängt.

**ACHTUNG** Zu beachten ist, dass die Abfragesprache zwischen Groß- und Kleinschreibung unterscheidet.

Beschreibung	URL
Auflisten der verfügbaren Entitäten	http://E01/WWWingsDataService.svc/
Abruf der Metadaten	http://E01/WWWingsDataService.svc/\$metadata
Alle Instanzen der Entität FL_Fluege	http://E01/WWWingsDataService.svc/FL_Fluege
Die Instanz der Entität FL_Fluege mit dem Wert 105 im Primärschlüssel	http://E01/WWWingsDataService.svc/FL_Fluege(105)
Nur die Anzahl der freien Plätze in der Instanz 105 der Entität FL_Fluege	http://E01/WWWingsDataService.svc/FL_Fluege(101)/FL_FreiePlaetze
Alle Instanzen der Entität FL_Fluege mit dem Abflugort "Rom"	http://E01/WWWingsDataService.svc/FL_Fluege?\$filter=FL_Abflugort%20eq%20'Rom'
Alle Instanzen der Entität FL_Fluege mit dem Abflugort "Rom" und dem Zielort "Berlin"	http://E01/WWWingsDataService.svc/FL_Fluege?\$filter=FL_Abflugort%20eq%20'Rom'%20and%20FL_Zielort%20eq%20'Berlin'
Alle gebuchten Flüge für Flug 101	http://E01/WWWingsDataService.svc/FL_Fluege(101)/GF_GebuchteFluege
Sortieren: Instanz der Entität FL_Fluege absteigend sortiert nach freien Plätzen	http://E01/WWWingsDataService.svc/FL_Fluege?\$orderby=FL_FreiePlaetze%20desc
Blättern (Paging): Instanz 200 bis 204 der Entität FL_Fluege	http://E01/WWWingsDataService.svc/FL_Fluege?\$skip=100&\$top=5

**Tabelle 32.2** Beispiel für Abfragen mit Datendiensten

## Nutzung eines Datendienstes in .NET-Anwendungen

Ein ADO.NET Data Service kann in jeder beliebigen .NET-Anwendung konsumiert werden, das heißt z. B., eine Konsolenanwendung oder eine Windows Forms-Anwendung können Client für einen Datendienst sein.

## Generieren der Zugriffsklassen

Für den komfortablen Zugriff auf die ADO.NET Data Services sind Proxyklassen notwendig. Diese kann man mit Visual Studio 2008 Service Pack 1 über die Funktion *Dienstverweis hinzufügen* (*Add Service Reference*) anlegen. Dort gibt man den URL des Datendienstes an. Automatisch wird neben den Klassen auch eine Referenz zu *System.Data.Services.Client.dll* angelegt.

Alternativ setzt man das Kommandozeilenwerkzeug *DataSvcUtil.exe* ein, das im Verzeichnis *WINDOWS\Microsoft.NET\Framework\v3.5* liegt.

Der folgende Befehl erstellt die Proxyklassen für den zuvor in diesem Kapitel erstellten Data Service:

```
DataSvcUtil.exe
/language:csharp
/out:c:\temp\WWWingsDSCClient.cs
/uri:http://E01/WWWingsDataService.svc/
```

Der Klassengenerator erzeugt eine Klasse für den Datenkontext, die abgeleitet ist von *System.Data.Service.Client.DataServiceContext* (hier: *WWWingsEntities*) sowie jeweils eine Klasse für jede Entität in dem Ausgangsmodell (ohne Basisklasse).

## Abfrage ohne LINQ

Nach der Generierung der Proxyklassen kann man den Datenkontext instanziiieren und dann von diesem mithilfe der Methode *Execute()* unter Angabe eines URL Daten abfragen:

```
/// <summary>
/// Abfragen von Datensätzen über einen Data Service in URL-Syntax
/// </summary>
private static void DSCClient_Lesen_URI()
{
    // Kontext erstellen
    WorldWideWingsModel.WorldWideWingsEntities DB = new WorldWideWingsModel.WorldWideWingsEntities(new
    Uri("http://E01/WWWings_Web35_SP1/WWWingsDataService.svc/"));

    // Abfrage
    IEnumerable<FL_Fluege> Fluege = DB.Execute<FL_Fluege>(new
    Uri("http://E01/WWWings_Web35_SP1/WWWingsDataService.svc/FL_Fluege?$orderby=FL_Abflugort"));

    // Ausgabe
    foreach (FL_Fluege f in Fluege)
    {
        Console.WriteLine(f.FL_FlugNr + " fliegt von " + f.FL_Abflugort + " nach " + f.FL_Zielort + " und
        hat " + f.FL_FreiePlaetze + "!");
    }
}
```

**Listing 32.3** Abfrage von Daten von einem ADO.NET Data Service

### TIPP

Durch die Methode *BeginExecute()* in der Klasse *WebDataQuery* kann man Abfragen auch asynchron ausführen.

## LINQ to ADO.NET Data Services

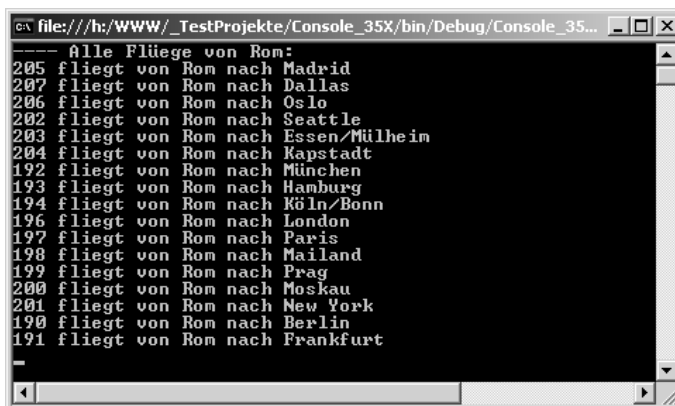
Der Client für die ADO.NET Data Services unterstützt auch LINQ. Diese Variante von LINQ wird *LINQ to ADO.NET Data Services* genannt. LINQ to ADO.NET Data Services bilden einen LINQ-Abfrageausdruck auf die URL-Syntax der ADO.NET Data Services ab:

```
/// <summary>
/// Abfragen von Datensätzen über einen Data Service in LINQ-Syntax
/// </summary>
private static void DSClient_Lesen_LINQ()
{
    // Kontext erstellen
    WorldWideWingsModel.WorldWideWingsEntities DB = new WorldWideWingsModel.WorldWideWingsEntities(new
    Uri("http://E01/WWWings_Web35_SP1/WWWingsDataService.svc/"));

    // LINQ-Abfrage
    var Fluege = from f in DB.FL_Fluege where f.FL_Abflugort == "Rom" select f;

    // Ausgabe
    foreach (FL_Fluege f in Fluege)
    {
        Console.WriteLine(f.FL_FlugNr + " fliegt von " + f.FL_Abflugort + " nach " + f.FL_Zielort + " und
        hat " + f.FL_FreiePlaetze + "!");
    }
}
```

**Listing 32.4** Abfrage von Daten von einem ADO.NET Data Service mithilfe von LINQ



**Abbildung 32.7** Ausgabe des Beispiels aus Listing 32.4

**ACHTUNG** Zumindest in der aktuellen Vorabversion werden noch nicht alle LINQ-Ausdrücke auf dem Client unterstützt.

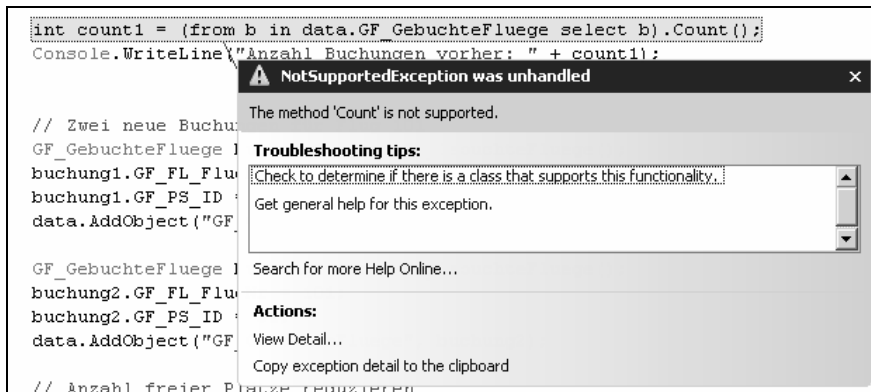


Abbildung 32.8 Count() wird (noch) nicht unterstützt

## Datenänderungen

ADO.NET Data Services unterstützen Datenänderungen:

- Zum Anfügen muss man eine Instanz des Proxydatenobjekts erstellen und mit `AddObject()` hinzufügen. Dabei ist als Parameter auch der Name der Entität (Klassenname) anzugeben.
- Zum Ändern einer Instanz muss man `UpdateObject()` nach der Änderung ausführen.
- `DeleteObject()` dient dem Löschen.

**WICHTIG** Auf jeden Fall muss man die Operation mit `SaveChanges()` abschließen. Asynchron speichern kann man mit `BeginSaveChanges()`. Optional kann angegeben werden, wie der Server mit Änderungskonflikten verfahren soll, z.B. `DB.MergeOption = MergeOption.OverwriteChanges;`

```

/// <summary>
/// Ändern von Datensätzen über einen Data Service
/// </summary>
private static void DSClient_Lesen_Aendern()
{
    WorldWideWingsModel.WorldWideWingsEntities DB = new WorldWideWingsModel.WorldWideWingsEntities(new
    Uri("http://E01/WWWings_Web35_SP1/WWWingsDataService.svc/"));

    var Fluege = (from f in DB.FL_Fluege where f.FL_Abflugort == "Rom" && f.FL_FreiePlaetze < 0 select
    f);

    foreach (FL_Fluege f in Fluege)
    {
        Console.WriteLine(f.FL_FlugNr + " fliegt von " + f.FL_Abflugort + " nach " + f.FL_Zielort + " und
        hat " + f.FL_FreiePlaetze + "!");

        // Objekt ändern
        f.FL_FreiePlaetze = 0;
    }
}

```

```

    // Objekt als verändert markieren
    DB.UpdateObject(f);
}

Console.WriteLine("Speichern aller Änderungen...");
// Alle Änderungen speichern
DB.SaveChanges();
}

/// <summary>
/// Löschen eines Datensatzes über einen Data Service
/// </summary>
private static void DSClient_Lesen_Loeschen()
{
    WorldWideWingsModel.WorldWideWingsEntities DB = new WorldWideWingsModel.WorldWideWingsEntities(new
Uri("http://E01/WWWings_Web35_SP1/WWWingsDataService.svc/"));

    // Zugriff auf einen bestimmten Flug
    var flug = (from f in DB.FL_Fluege where f.FL_FlugNr == 99999 select f).SingleOrDefault();

    // Flug löschen, wenn er gefunden wurde
    if (flug != null)
    {
        Console.WriteLine("Flug wird gelöscht!");
        DB.DeleteObject(flug);
    }
    else
    {
        Console.WriteLine("Flug nicht gefunden!");
    }

    // Alle Änderungen speichern
    DB.SaveChanges();
}

/// <summary>
/// Anfügen eines neuen Datensatzes über einen Data Service
/// </summary>
private static void DSClient_Lesen_Anfuegen()
{
    WorldWideWingsModel.WorldWideWingsEntities DB = new WorldWideWingsModel.WorldWideWingsEntities(new
Uri("http://E01/WWWings_Web35_SP1/WWWingsDataService.svc/"));

    // === Einen Datensatz ergänzen
    WorldWideWingsModel.FL_Fluege n = new WorldWideWingsModel.FL_Fluege();
    n.FL_FlugNr = 99999; // bitte auf Einmaligkeit des Keys achten!
    n.FL_Abflugort = "Peking";
    n.FL_Zielort = "Sydney";
    n.FL_Plaetze = 250;
    n.FL_FreiePlaetze = 250;
    DB.AddObject("FL_Fluege", n);
}

```

```

Console.WriteLine("Flug wird ergänzt...");
// Alle Änderungen speichern
DB.SaveChanges();
}

```

**Listing 32.5** Beispiel für das Anlegen, Ändern und Löschen von Datensätzen über einen Data Service

## AJAX-Webseiten als WCF-Clients (AJAX Client Library for ADO.NET Data Services)

AJAX-Webseiten können als WCF-Clients und Clients für ADO.NET Data Services zum Einsatz kommen. Ursprünglich sollte diese Funktion in .NET 3.5 Service Pack 1 enthalten sein. Leider hat Microsoft sich dazu entschlossen, diese Funktion nur als ein nicht durch den Produktsupport unterstütztes Beispiel auf Codeplex bereitzustellen. Der Name des Projekts dort ist *AJAX Client Library for ADO.NET Data Services* (siehe <http://www.codeplex.com/aspnet>).

Die Nutzung eines ADO.NET Data Service von JavaScript aus funktioniert ähnlich wie die direkte Nutzung eines WCF-Dienstes. Im Detail gibt es jedoch Unterschiede:

- Die Seite benötigt ein ScriptManager-Steuerelement. Der ScriptManager muss jedoch keinen Verweis auf die .svc-Datei enthalten, sondern eine ScriptReference auf *DataService.js*.
- Für den Aufruf ist kein generierter Proxy notwendig. Stattdessen instanziiert man die Klasse *Sys.Data.DataService* (aus der Skriptdatei *MicrosoftAjaxDataService.js*) unter Angabe des Pfades zur .svc-Datei.
- Der eigentliche Aufruf erfolgt mit der Methode *query()* unter Angabe der Abfragezeichenkette. Wie bei den WCF-AJAX-Proxys sind zwei Methoden anzugeben: Rückrufmethode im Erfolgsfall und Rückrufmethode im Fehlerfall.
- LINQ kann man nicht als Abfragesprache einsetzen.

**ACHTUNG** Man kann nur Datendienste auf der gleichen Website aufrufen. Dies ist eine Sicherheitsrestriktion der Webbrowser.

```

<%@ Page Language="..." AutoEventWireup="true" CodeFile="..."
Inherits="Flugdaten AJAX_DatendiensteClient" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title>Flugplan - AJAX mit ADO.NET Data Service</title>

<script src="AJAX_Hilfsroutinen.js" type="text/javascript"></script>
<script type="text/javascript">

function DatenLaden() {
var abflug = document.getElementById('C_Abflugort').options[$get('C_Abflugort').selectedIndex].value;
$get("C_Tabelle").innerHTML = "Daten für " + abflug + " werden geladen...";
var DataService = new Sys.Data.DataService("/WingsDataService.svc");

```

```

    DataService.query("/FL_Fluege?$filter=FL_Abflugort eq '" + abflug + "'", DatenLaden_Callback,
    Fehler_Callback);
}

// ===== Fehlerfall
function Fehler_Callback(error)
{
    ///<summary>Zeigt den Fehler anstelle der Tabelle an</summary>
    ///<param name="error" type="Error Object">Fehlerinformation</param>
    ///<returns>string</returns>
    var text = "Fehler bei der AJAX-Verarbeitung auf dem Server: " + error.get_message();
    $get("C_Tabelle").innerHTML = text;
    return error.get_message();
}

// ===== Erfolgsfall
function DatenLaden_Callback(result, context, operation)
{
    var s = "";
    s = "<table border='0' width='90%' noshade='noshade'>";
    //alert(result.length);
    for (var i=0; i<result.length; i++)
    {
        if (i % 2 == 0) { s = s + "<tr bgcolor='yellow'>"; } else { s = s + "<tr>"; }

        s = s + "<td>" + result[i].FL_FlugNr + "</td>";
        s = s + "<td>" + result[i].FL_Datum.toLocaleString() + "</td>";
        s = s + "<td>" + result[i].FL_Abflugort + "</td>";
        s = s + "<td>" + result[i].FL_Zielort + "</td>";
        s = s + "<td>" + "<a href='F' + result[i].FlugNr + ".aspx'>Buchen</a></td>";
        s = s + "</tr>";
    }
    s = s + "</table>";
    $get("C_Tabelle").innerHTML = s;
}

</script>

<style type="text/css">
    .style1
    {
        font-family: Arial;
        font-size: large;
    }
</style>

</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ScriptManager ID="ScriptManager1" runat="server">
                <Scripts>
                    <asp:ScriptReference Name="MicrosoftAjaxDataService.js" />
                </Scripts>
            </asp:ScriptManager>
        </div>
        <p>
            <span class="style1">Alle Flüge von einem bestimmten
Ort</span></p>

```



```

<select id="C_Abflugort" name="C_Abflugort">
  <option value="Berlin" selected="selected">Berlin</option>
  <option value="Rom">Rom</option>
  <option value="Paris">Paris</option>
  <option value="Dallas">Dallas</option>
</select>
<input id="C_Anfordern" type="button" value="Anfordern" onclick="DatenLaden();" />
<br />
<br />
<div id="C_Tabelle">
</div>
</form>
</body>
</html>

```

Listing 32.6 Nutzung eines Datendienstes in JavaScript

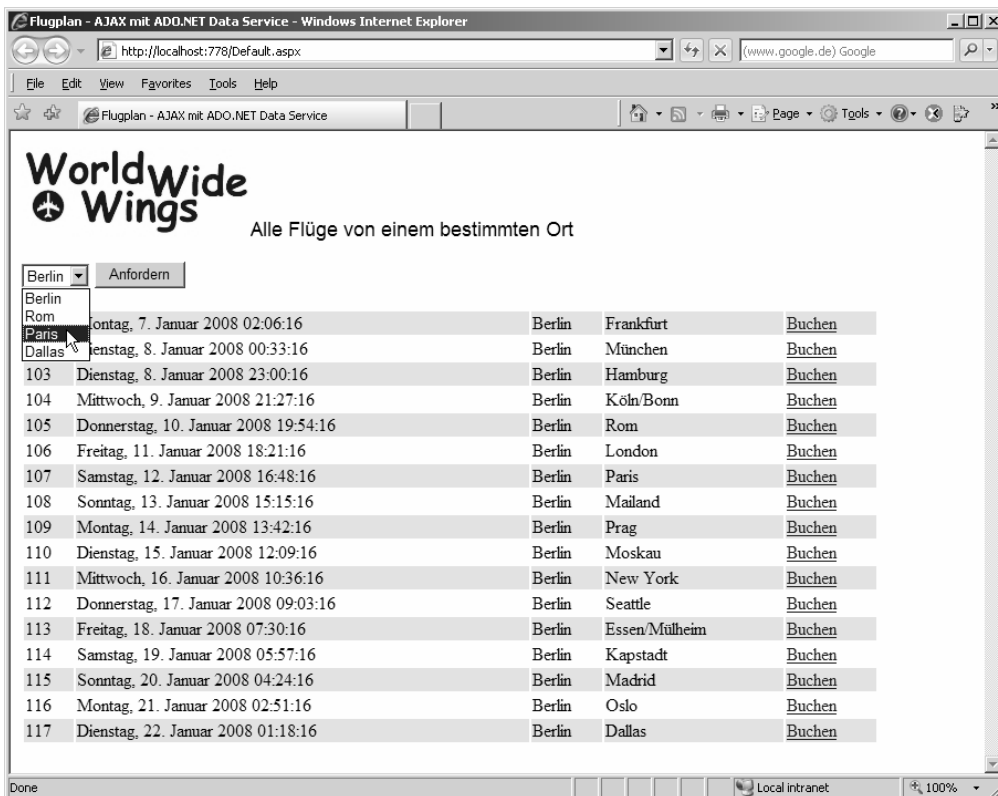


Abbildung 32.9 Ausgabe des Beispiels

**HINWEIS** Die Klasse `Sys.Data.DataService` stellt auch die Methoden `update()`, `remove()` und `insert()` bereit. Diese Aktionen werden sofort übertragen. Möchte man mehrere Aktionen zusammen übertragen, muss man mit `createActionSequence()` zunächst ein Objekt vom Typ `ActionSequence` erstellen und darauf dann `addInsertAction()`, `addUpdateAction()` und `addRemoveAction()` aufrufen. Am Ende ist dann `executeActions()` aufzurufen.

## Weitere Möglichkeiten

Es gibt weitere Möglichkeiten in Datendiensten, die in dieser Auflage des Buchs aus Platzgründen leider nicht dargestellt werden können:

- Neben dem ADO.NET Entity Framework sind auch andere Datenquellen (z.B. LINQ to SQL, XML) möglich.
- Dienstoperationen (Service Operations) sind Methoden in dem Datendienst, die ebenfalls über einen URL aufgerufen werden können (Verwendung der Annotationen `[WebGet]` und `[WebInvoke]`). Zugriffsrechte darauf kann man mit `config.SetServiceOperationAccessRule("OperationsName", ServiceOperationRights.All)` vergeben.
- Mit einem Abfrageeingriff (Query Interceptor) durch die Annotation `[QueryInterceptor]` kann man für Entitäten Bedingungen angeben, welche Daten der Client anfragen darf.
- Mit so genannten Batch Operationen kann man mehrere Operationen zu einem HTTP-Request zusammenfassen.
- Mit HTTP eTags (siehe [WP01]) lassen sich Änderungskonflikte feststellen.
- Datendienste kann man auch in Microsoft Silverlight verwenden.
- Datendienste lassen sich auch in eigenen Anwendungen bereitstellen.