

Daten im Web mit OData veröffentlichen

# Offene Daten für ein offenes Web

Daten im Web nicht nur über HTML verfügbar zu machen, sondern direkt und über Webadressen, die gleichzeitig Abfragen à la SQL enthalten – das Open Data Protocol macht es möglich und schafft dazu ein einheitliches Format für Webadressen.

## Auf einen Blick



**Matthias Jauernig** ist 27 Jahre alt, hat Informatik in Leipzig studiert (M.Sc.) und arbeitet als Senior Expert (.NET) bei SDX in Frankfurt. Das Unternehmen unterstützt Betriebe bei der Entwicklung komplexer Lösungen in den Bereichen Custom Development, Cloud Computing, SOA und BI. Matthias Jauernig ist über seinen Blog unter [www.minddriven.de](http://www.minddriven.de) zu erreichen.

### Inhalt

- OData erlaubt es, Datenabfragen via Webadresse zu formulieren.
- OData basiert auf offenen Webstandards wie HTTP, Atom und JSON.
- OData-Diensten liegt eine REST-Architektur zugrunde.

### dnpCode

A10110data

Das World Wide Web besteht aus einem fast unendlich scheinenden Netz von untereinander verbundenen Webseiten, die ihre Informationen meist über HTML bereitstellen. Oft liegen diesen Webseiten Datenquellen zugrunde, die im Verborgenen arbeiten und in denen die eigentlich interessanten Informationen – die Rohdaten – eingeschlossen sind. Anwendungen, die diese Daten direkt nutzen wollen, müssen daher erst das HTML-Markup einer Webseite verarbeiten, um Informationen herauszuziehen. Dies ist zwar dem ursprünglichen Zweck des World Wide Web geschuldet, genügt heute aber nicht mehr. Das hat Weberfinder Tim Berners-Lee erkannt, der die nächste Entwicklungsstufe des Webs in offen zugänglichen und miteinander verbundenen Daten sieht [1].

Mit dem Open Data Protocol [2] hat Microsoft ein offenes Format geschaffen, das genau dies ermöglicht: Daten direkt im Web über einen einfachen Zugriffsmechanismus, nämlich die Webadresse, bereitzustellen. Microsoft hat OData unter der Open Specification Promise [3] veröffentlicht, sodass jeder das Protokoll nutzen und anpassen kann und dabei keine Angst vor Lizenzforderungen oder Patentklagen seitens Microsoft haben muss. Im Laufe des Artikels werden verschiedene Beispiele gezeigt. Viele davon beziehen sich auf den Demo-Service des OData-Teams [4], welcher ein Modell der Northwind-Datenbank im Internet allgemein zugänglich macht [5].

Die Einheitlichkeit des Protokolls schlägt sich in mehreren Teilen nieder:

- in der einheitlichen Repräsentation strukturierter und unstrukturierter Daten,
- in einheitlichen URL-Konventionen,
- in einheitlichen Datenoperationen (lesend/schreibend, CRUD).

OData erfindet das Rad nicht neu, sondern baut auf vorhandenen und etablierten Webtechnologien wie HTTP, Atom/AtomPub und JSON auf und stellt auf deren Basis reichhaltige Funktionen zum Umgang mit Daten bereit.

Ursprünglich entstammt das Format dem „Project Astoria“ und den ADO.NET Data Services, die in .NET 4.0 als WCF Data Services Ein-

zug gehalten haben. Hier ermöglichen sie das Bereitstellen von Daten aus .NET heraus über Webdienste. Ende 2009 hat sich Microsoft entschlossen, das zugrunde liegende Protokoll von der .NET-Implementierung zu trennen und sprachenunabhängig zur Verfügung zu stellen. OData ist somit losgelöst von Microsoft-Technologien und im Prinzip von jeder Plattform aus zugänglich – von Java, PHP, Silverlight oder Ajax.

### Protokoll-Beziehungen

OData-Dienste sind prinzipiell nach dem REST-Architekturprinzip aufgebaut. Die grundlegenden CRUD-Operationen kann man über die Standardmethoden des HTTP-Protokolls ausführen:

- GET liest Daten vom Server.
- POST fügt neue Datensätze hinzu.
- PUT aktualisiert bestehende Daten.
- DELETE löscht Datensätze auf dem Server.

Durch HTTP als Basisprotokoll lassen sich auch die üblichen HTTP-Funktionen nutzen, zum Beispiel für Authentifizierung, Verschlüsselung und Caching.

Als Datenformat verwendet OData Atom und AtomPub [6]. AtomPub war ursprünglich als Protokoll für Aktionen auf Blogbeiträgen gedacht, wird von vielen modernen Blogengines unterstützt und erlaubt bereits das Ausführen von CRUD-Operationen auf Atom-Daten.

OData erweitert die Möglichkeiten von AtomPub. AtomPub fehlt jegliche Form eines Datenmodells und einer Abfragesprache für Daten. Auch URL-Konventionen sind nicht vorgegeben. Bei OData sind diese Elemente fester Bestandteil. Zudem erlaubt OData, alternative Datenformate wie JSON zu verwenden. OData bietet somit reichhaltigere Möglichkeiten der Datenabfrage und -manipulation.

Atom ist das Standardformat, in dem das Open Data Protocol Daten überträgt. Es fasst mehrere Datenelemente, die sogenannten Entries, in Gruppen zusammen (Feeds). Atom selbst ist ein XML-Dialekt, der einige Vorgaben zum Aufbau des XML-Codes macht. So verfügen Entries über vordefinierte Tags wie *id*, *title* und *author*, die ein Element charakterisieren und unter anderem von Feed-Readern ausgewertet werden können.

Darüber hinaus existiert in jedem Eintrag ein `<content>`-Element, in dem Nutzdaten jeder Art hinterlegt werden können, die jedoch nicht einheitlich strukturiert sein müssen. OData verfügt hingegen über ein Datenmodell, mit dem sich hier strukturierte und unstrukturierte Daten einfügen lassen, die Clients abrufen und auswerten können.

## Das OData-Datenmodell

OData stellt Daten als Feeds typisierter Entities bereit. Die Typisierung erfolgt durch das abstrakte Datenmodell von OData, das Entity Data Model (EDM), das in der Conceptual Schema Definition Language (CSDL) definiert wird; der Kasten *EDM, CSDL und EDMX* auf Seite 97 erläutert die Zusammenhänge zwischen den Formaten.

Die zentralen Teile des EDM sind Entities und Assoziationen. Entities sind strukturierte Datenobjekte, die über einen Schlüssel zu identifizieren sind und sich aus einer Liste von Eigenschaften zusammensetzen. Assoziationen zwischen Entities werden in Navigation Properties definiert, die Links auf andere Entities enthalten. Ein Beispiel für eine in CSDL definierte Entity namens *Product* zeigt Listing 1.

Jeder OData-Service stellt auf der obersten Ebene Metadaten zur Verfügung, die das EDM enthalten, formuliert in CSDL. Das Dokument zum Beispiel, das die Metadaten des Demodienstes zur Northwind-Datenbank enthält, ist unter [http://services.odata.org/Northwind/Northwind.svc/\\$metadata](http://services.odata.org/Northwind/Northwind.svc/$metadata) abrufbar.

## Das URL-Schema

Auf der obersten Ebene stellt ein OData-Service neben den Metadaten vor allem auch das Dokument zur Verfügung, das einen Überblick über die bereitgestellten Daten gibt. Dies geschieht durch Verweise auf die Collections, also durch Verknüpfungen zu den Top-Level-Feeds. Dieses Dokument stellt somit den Einstiegspunkt für die Daten dar, die der Service anbietet.

OData bietet eine Vielzahl von Konventionen für URLs und deren Abfrageoptionen [10]; hier besteht ein wichtiger Unterschied zu AtomPub, das keine Vorgaben zu den URLs macht. Mittels URL-Schema lassen sich alle denkbaren Informationen auf jeder Granularitätsebene via URL abrufen.

Ein Abfrage-URL enthält immer den Pfad der abzufragenden Ressource, gefolgt von den Optionen, welche die Abfrage spezifizieren. Nicht alle von OData definierten URL-Konventionen müssen tatsächlich bei

## Listing 1

### Eine in CSDL definierte Entity namens Product (vereinfacht).

```
...
<EntityType Name="Product">
  <Key><PropertyRef Name="ID" /></Key>
  <Property Name="ID" Type="Edm.Int32" Nullable="false" />
  <Property Name="Name" Type="Edm.String" Nullable="true" />
  <Property Name="ReleaseDate" Type="Edm.DateTime" Nullable="false" />
  <Property Name="Price" Type="Edm.Decimal" Nullable="false" />
  ...
  <NavigationProperty Name="Category"
    Relationship="ODataDemo.Product_Category_Category_Products"
    FromRole="Product_Category" ToRole="Category_Products" />
</EntityType>
...
```

## Listing 2

### Der EF-Kontext NorthwindEntities als einfacher WCF Data Service.

```
public class NorthwindService : DataService<NorthwindEntities>
{
    public static void InitializeService(DataServiceConfiguration config)
    {
        config.SetEntitySetAccessRule("*", EntitySetRights.All);
        config.DataServiceBehavior.MaxProtocolVersion = DataServiceProtocolVersion.V2;
    }
}
```

einem OData-Service implementiert sein. OData ist ein modulares Protokoll, bei dem Anbieter lediglich die Teile implementieren müssen, die sie brauchen, um die gewünschten Funktionen bereitzustellen. Einzige Voraussetzung ist, dass der Dienst Anfragen an ein nicht implementiertes Merkmal nicht akzeptieren darf und eine Fehlermeldung liefert. Eine einfache Abfrage stellt folgender URL dar:

```
http://services.odata.org/Northwind/
Northwind.svc/Products
```

Die Adresse gibt die gesamte *Products*-Collection des Dienstes zurück. Ist nur das Produkt mit dem Schlüssel 1 gefragt, so ist dieses unter folgendem URL verfügbar:

```
http://services.odata.org/Northwind/
Northwind.svc/Products(1)
```

OData definiert eine Vielzahl an Optionen, mit denen sich Abfragen direkt über den URL gestalten lassen. Das ermöglicht unter anderem die Ausführung elementarer Datenoperationen auf dem Server wie Sortieren, Filtern und Paging über URL-Parameter. Ein weiteres Beispiel dazu:

```
http://services.odata.org/Northwind/
Northwind.svc/Suppliers?
$filter=Address/Country eq 'USA'
&&expand=Products
&&format=json
```

Der URL fragt die *Suppliers*-Collection des Dienstes ab. Er gibt nur solche Lieferanten zurück, die dem Filterkriterium entsprechen, deren Adressen also in den USA liegen. Zudem werden in die Antwort die assoziierten Produkte per *\$expand* mit ihren Werten eingebunden und das Resultat im JSON-Format ausgegeben. Die wichtigsten URL-Optionen fasst Tabelle 1 zusammen. Die Tabellen 2 und 3 gehen explizit auf die mächtige *\$filter*-Option ein.

OData bietet neben seinen CRUD-Funktionen noch weitere Möglichkeiten wie Service-Operationen, Transaktionen und Batch-Verarbeitung. Weitere Informationen dazu auf der OData-Webseite.

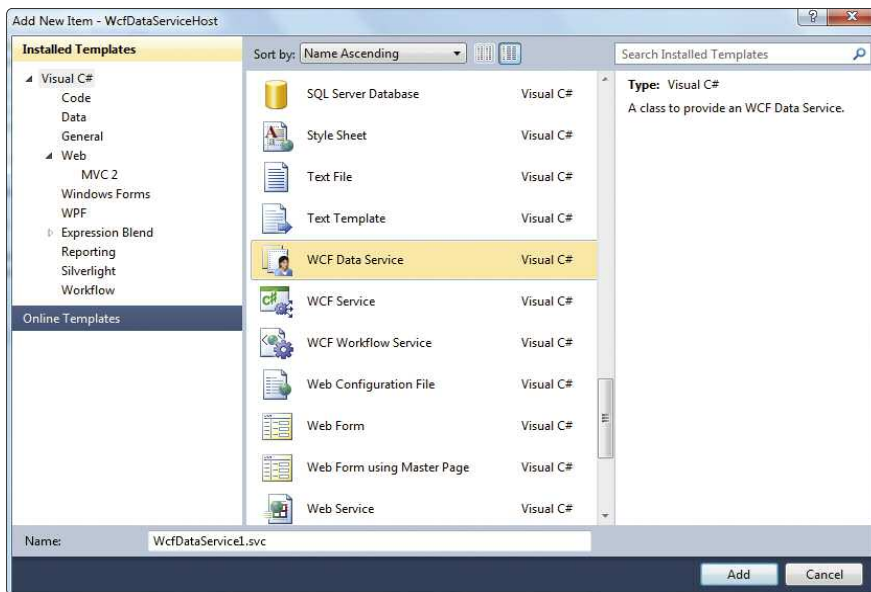
## OData-Dienste erstellen

Das Bereitstellen eigener OData-Dienste ist vor allem aus .NET heraus komfortabel möglich. Die Basis dafür sind die WCF Data Services. Visual Studio 2010 enthält eine

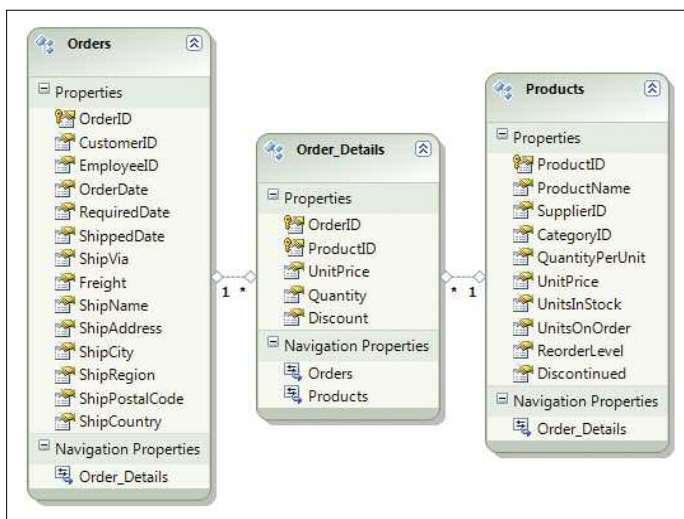
Listing 3

Service-Dokument der Top-Level-Site der SDX-Site-Collection auf SharePoint 2010 (vereinfacht).

```
<service xml:base="http://demo2010/sites/sdx/_vti_bin/listdata.svc/" ...>
  <workspace>
    ...
    <collection href="Kompetenzfelder">
      <atom:title>Kompetenzfelder</atom:title>
    </collection>
    <collection href="Mitarbeiter">
      <atom:title>Mitarbeiter</atom:title>
    </collection>
    <collection href="MitarbeiterRolle">
      <atom:title>MitarbeiterRolle</atom:title>
    </collection>
    ...
  </workspace>
</service>
```



[Abb. 1] Die Vorlage für ein WCF-Data-Service-Projekt in Visual Studio 2010.



[Abb. 2] Einfaches EDM mit dem Entity Framework auf Basis der Northwind-Datenbank.

neue Vorlage *WCF Data Service*, mit dem sich ein solcher Dienst erstellen lässt (Abbildung 1).

Ein WCF Data Service wird mit einer beliebigen Datenquelle parametrisiert und stellt deren Daten nach außen zur Verfügung. Dabei können Zugriffsregeln die Rechte auf die einzelnen Daten definieren. Query und Change Interceptors erlauben darüber hinaus weiteres individuelles Verhalten, um zum Beispiel Ergebnismengen benutzerabhängig zu filtern oder Autorisierungen zu ermöglichen.

Bei der Art der Datenquellen selbst gibt es zunächst keine Einschränkungen, hier lassen sich auch eigene Modelle angeben. Am einfachsten ist das Bereitstellen von EDMs aus dem Entity Framework, die ebenfalls Modelle von Datenbanken oder Datenquellen darstellen. Wird ein WCF-Datendienst derart über *ObjectContext* parametrisiert, so wird das EDM automatisch als OData-Dienst publiziert.

Als Beispiel soll einmal mehr die Northwind-Datenbank herhalten, zu der mit dem Entity Framework für einige Tabellen EDMs erstellt wurden (Abbildung 2). Der *ObjectContext NorthwindEntities* kann nun als Typparameter für einen WCF Data Service verwendet werden, der die Tabellen im EDM nach außen exponiert.

Listing 2 zeigt die einfachste Implementierung für einen solchen Dienst, der OData-Funktionen zur Verfügung stellt. Jede Änderung am EDM zieht automatisch eine Änderung des OData-Dienstes nach sich, es sind keine manuellen Anpassungen an dem Dienst selbst nötig. Im genannten Beispiel hat nun jeder Client Zugriff auf alle Collections des EDMs. In der Praxis sollten die Zugriffsrechte natürlich entsprechend eingeschränkt werden. Außerdem bieten WCF-Datendienste noch eine Vielzahl weiterer Funktionen; diese sollen an dieser Stelle allerdings nicht weiter von Belang sein. Der interessierte Leser sei daher auf [11] verwiesen. Neben den Datendiensten publizieren auch die in .NET 4.0 eingeführten WCF RIA Services automatisch einen OData-Endpoint und können so ihre Daten anbieten [12].

OData-Produkte

Es gibt bereits einige Produkte, die OData-Dienste zur Verfügung stellen. IBMs WebSphere zum Beispiel kann Daten als OData-Feeds anbieten. Ebenso sind SQL Azure-Datenbanken und der Azure Table Storage in Windows Azure dazu in der Lage. Auch Microsofts Datendienst Dallas stellt alle

Daten als OData-Feeds bereit, sodass sich aus ihnen leicht Mashups erzeugen lassen. Zu den aktuellen Anbietern frei verfügbarer Daten auf diesem Onlinemarktplatz zählen unter anderem die NASA, Associated Press und staatliche Institutionen wie die US-Regierung. Letztere sind gerade im Zusammenhang mit der Diskussion um Open Government interessant. Eine Liste aller in Dallas verfügbaren Datenanbieter liefert [13]. Und auch MS SQL Server 2008 R2 bleibt nicht außen vor: Die SQL Server Reporting Services erlauben, Berichtsdaten mittels OData zu veröffentlichen.

Sehr interessant dürfte für Entwickler die Unterstützung von OData durch SharePoint 2010 sein. Die Plattform stellt alle Daten als OData-Feeds zur Verfügung, sodass sie sehr leicht mit CRUD-Operationen zu verarbeiten sind. Dabei publiziert jede Site eines SharePoint-Servers einen eigenen OData-Service, welcher die in der Site enthaltenen Listen und weitere Daten veröffentlicht. Das wird viele Entwickler freuen, die früher bereits Daten aus SharePoint verarbeiten wollten und dabei einen hohen Aufwand betreiben mussten.

Wenn zum Beispiel ein SharePoint-Server namens *demo2010* die Site Collection *SDX* enthält, dann ist ein OData-Service für die Top-Level-Site dieser Collection unter dem folgenden URL ansprechbar:

```
http://demo2010/sites/sdx/_vti_bin/
listdata.svc/
```

Diese Webadresse stellt das Service-Dokument zur Verfügung, das Listing 3 vereinfacht wiedergibt. Auf den enthaltenen Auflistungen lassen sich per URL-Konvention die bekannten OData-Operationen ausführen. Zum Beispiel enthält die Site die Liste *Mitarbeiter*, wie sie in Abbildung 3 dargestellt ist. Diese Liste lässt sich direkt in SharePoint 2010 verwalten, wobei etwa das *Kompetenzfeld* einen Querverweis auf die weitere Liste *Kompetenzfelder* darstellt. Doch auch mittels OData lassen sich alle Daten abrufen und bearbeiten. So liefert folgender URL den Mitarbeiter mit *ID=1*, siehe Listing 4:

```
http://demo2010/sites/sdx/_vti_bin/
listdata.svc/Mitarbeiter(1)
```

Weitere Informationen zu OData und SharePoint 2010 gibt es unter [14].

Microsoft hat bereits ein beachtliches Ökosystem an Datenproduzenten mit OData als Basis aufgebaut. Zudem ist absehbar, dass zukünftig weitere Microsoft-Produkte OData unterstützen werden.

Tabelle 1

### Die wichtigsten Optionen für OData-Abfragen.

QueryString-Option	Beschreibung
\$orderby=x,y	Mit dieser Option lässt sich die Sortierung eines Abfrageresultats bestimmen. Dabei kann kommagetrennt eine Liste von den Eigenschaften <i>x,y</i> einer Entity angegeben werden. Die Sortierreihenfolge ist mit <i>asc</i> und <i>desc</i> festlegbar. Auch die Angabe von Eigenschaften assoziierter Objekte ist möglich, beispielsweise <i>Products/?\$orderby=Rating,Category/Name desc</i> .
\$top=n	Hier werden nur die obersten <i>n</i> Elemente eines Resultats zurückgegeben, etwa die obersten zehn Entities mit <i>\$top=10</i> . Zusammen mit <i>\$skip</i> ermöglicht die Option ein serverseitiges Paging.
\$skip=n	<i>\$skip</i> überspringt die obersten <i>n</i> Elemente eines Resultats, gibt also die Elemente ab der Position <i>n+1</i> zurück. Über <i>\$skip</i> und <i>\$top</i> lässt sich leicht ein serverseitiges Paging erreichen. Zum Beispiel gibt folgende Abfrage bei einer Seitengröße von 10 die dritte Seite der <i>Products</i> -Collection zurück: <i>Products/?\$skip=20&amp;\$top=10</i> .
\$filter=x,y	<i>\$filter</i> erlaubt die Angabe vielfältiger Prädikate zu Eigenschaften und Literalen, welche die Elemente eines Resultats erfüllen müssen. Dabei lassen sich mehrere Prädikate <i>x,y</i> durch Kommas getrennt angeben. Tabelle 2 zeigt die wichtigsten Operatoren und Methoden, die dabei zum Einsatz kommen können, Tabelle 3 gibt einige Beispiele für deren Verwendung an.
\$expand=x,y	Dieser Befehl bindet die Daten der verlinkten Entity <i>x</i> in das Resultat ein. <i>Products?\$expand=Category</i> zum Beispiel liefert neben dem abgefragten Produkt auch gleich die Daten der verlinkten Kategorie zurück. Mehrere Entities <i>x,y</i> lassen sich ebenfalls wieder kommagetrennt angeben.
\$select=x,y	Mit <i>\$select</i> ist eine serverseitige Projektion von Entity-Eigenschaften möglich. Das heißt, <i>\$select</i> erlaubt die Angabe von Eigenschaften <i>x,y</i> , die im Resultat enthalten sein sollen. Der Client bekommt somit nur die Daten, die er benötigt.
\$format=x	Legt das Format <i>x</i> fest, in dem der Server das Resultat an den Client senden soll, zum Beispiel <i>\$format=Atom</i> (das ist der Standardwert) oder <i>\$format=json</i> . Hier lässt sich ein beliebiges Format angeben, das der OData-Service auf dem Server erzeugen kann.

Tabelle 2

### \$filter-Operationen.

Operation	Beschreibung
Logische Operatoren: <i>eq, ne, gt, ge, lt, le, and, or, not</i>	Diese Operatoren erlauben den Vergleich oder die logische Verknüpfung zwischen Termen, Eigenschaften und Literalen. <i>eq</i> (equals) etwa entspricht dem „ist gleich“, <i>lt</i> (less than) dem „kleiner als“ und <i>ge</i> (greater equal) dem „größer gleich“.
Arithmetische Operatoren: <i>add, sub, mul, div, mod</i>	Operationen auf Zahlwerten, um mit deren Ergebnis weiterarbeiten zu können, etwa <i>Price add 5</i> .
String-Funktionen: <i>substringof(...), startswith(...), length(...), replace(...), tolower(...)</i> usw.	Vielfältige String-Operatoren; zum Beispiel prüft <i>substringof('Sony', Name)</i> , ob die Property <i>Name</i> den Text <i>Sony</i> enthält. Eine vollständige Liste liefert [13].
Datumsfunktionen: <i>day(...), year(...), hour(...)</i> usw.	Mit diesen Funktionen lässt sich ein übergebenes Datum in seine Teile zerlegen. Eine vollständige Liste liefert [13].
Mathematische Funktionen: <i>round(...), floor(...), ceiling(...)</i> usw.	Ermöglicht mathematische Operationen, wie etwa <i>ceiling(Price)</i> zum Aufrunden der <i>Price</i> -Property auf die nächste Ganzzahl. Eine vollständige Liste liefert [13].
Typ-Funktionen: <i>isof(...), cast(...)</i>	Prüft oder konvertiert Datentypen.

Tabelle 3

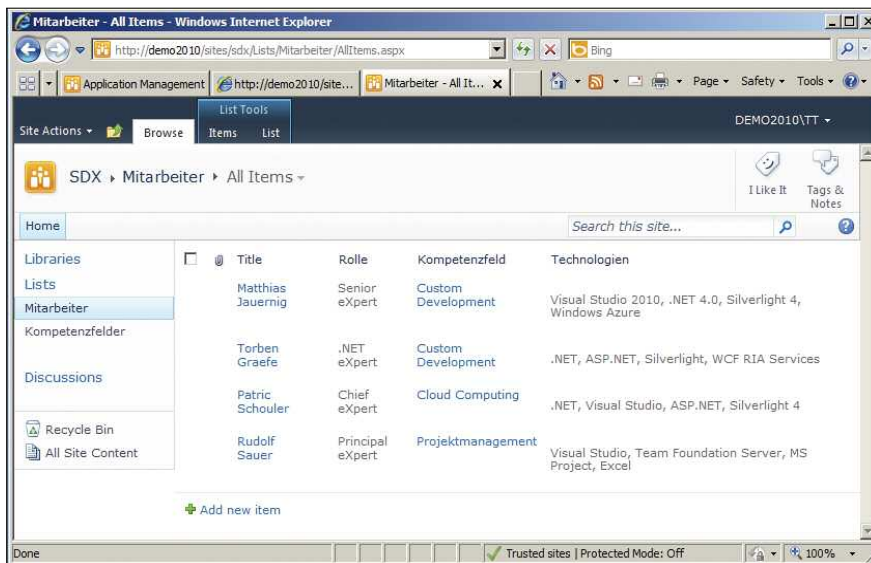
### Beispiele zur Anwendung der \$filter-Operationen.

Beispiel-URL	Beschreibung
<code>http://services.odata.org/Northwind/Northwind.svc/Customers?\$filter=length(CompanyName) gt 10</code>	Gibt alle Kunden zurück, deren Firmenname aus mehr als zehn Zeichen besteht.
<code>http://services.odata.org/Northwind/Northwind.svc/Customers?\$filter=not startswith(CompanyName, 'Alfred') and 'Alfred' beginswith(CompanyName) and (Country eq 'Germany')</code>	Gibt alle Kunden zurück, deren Firmenname nicht mit „Alfred“ beginnt und deren Land „Germany“ entspricht.
<code>http://services.odata.org/Northwind/Northwind.svc/Products?\$filter=substringof('Chef', ProductName) and ((UnitsInStock mod 2) eq 0)</code>	Gibt alle Produkte zurück, deren Name „Chef“ enthält und die eine gerade Anzahl von Einheiten auf Lager haben.
<code>http://services.odata.org/Northwind/Northwind.svc/Products?\$filter=(floor(UnitPrice) le 10) and (Category/CategoryName eq 'Beverages')</code>	Gibt alle Produkte zurück, deren Preis pro Einheit abgerundet kleiner gleich 10 ist und deren Kategorie den Namen „Beverages“ hat.

Listing 4

Mitarbeiter mit der ID=1 aus SharePoint 2010 in Atom (vereinfacht).

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<entry xml:base="http://demo2010/sites/sdx/_vti_bin/listdata.svc/" ... >
  <id>http://demo2010/sites/sdx/_vti_bin/listdata.svc/Mitarbeiter(1)</id>
  <title type="text">Matthias Jauernig</title>
  <updated>2010-06-28T21:55:32+02:00</updated>
  <author><name /></author>
  <content type="application/xml">
    <m:properties>
      <d:Id m:type="Edm.Int32">1</d:Id>
      <d:Title>Matthias Jauernig</d:Title>
      <d:RolleValue>Senior eXpert</d:RolleValue>
      <d:Technologien>
        <div>
          <p>Visual Studio 2010, .NET 4.0, Silverlight 4, Windows Azure</p>
        </div>
      </d:Technologien>
      <d:KompetenzfeldId m:type="Edm.Int32">1</d:KompetenzfeldId>
      ...
    </m:properties>
  </content>
  ...
</entry>
```



[Abb. 3] Die Liste der Mitarbeiter in SharePoint 2010.

OData-Services beziehen

Daten können von OData-Services auf vielfältige Weise abgerufen und verarbeitet werden. Zunächst lassen sich Anfragen an solche Dienste per URL mit jedem gängigen Browser stellen, der die Daten als XML-Dokument und somit auch als Atom-Feed auch darstellen kann.

Geht es um das Auswerten von Daten einer einfachen Geschäftslogik, bietet das Excel-Add-in PowerPivot [15] die Möglichkeit, OData-Dienste anzubinden. Power-

Pivot kann Daten aus OData-Feeds abrufen, auswerten und darstellen.

Aus .NET heraus lassen sich OData-Dienste mittels LINQ abfragen. Eine LINQ-Abfrage wird dabei in einen URL übersetzt und an den Server übermittelt; alle Operationen finden also dort statt, der Netzwerkverkehr wird minimiert.

Auf diese Weise sind auch komplexe Operationen möglich wie zum Beispiel Projektionen, die nur die Daten übermitteln, die der Client benötigt.

In Visual Studio 2010 können OData-Dienste für einen Client mittels *Add Service Reference* hinzugefügt werden. Der Verweis erzeugt einen Proxy vom Typ *DataServiceContext* für die Interaktion mit dem Dienst; zudem werden alle verfügbaren Entities und Datentypen als Klassen erzeugt, was ein einfaches, typisiertes Arbeiten mit den Daten ermöglicht. Der Zugriff auf den Datenkontext und die Daten kann auch hier effizient über LINQ erfolgen.

Ein Beispiel ist in Listing 5 zu sehen, das eine LINQ-Anfrage an SharePoint 2010 implementiert und Mitarbeiter abfragt. Die Abfrage fordert den Namen (*Title*) und die Rolle (*RolleValue*) der Mitarbeiter an, die einem *Development*-Kompetenzfeld zugeordnet sind, und sortiert sie nach dem Namen. Der Namensraum *SharePointService* enthält die generierten Klassen, *SDXDataContext* stellt den Datenkontext dar, der den Zugriff auf die Top-Level-Site der SDX-Site-Collection des SharePoint-Servers erlaubt. Über das *select*-Schlüsselwort projiziert die LINQ-Abfrage nur die Eigenschaften, die für den Client relevant sind. Neben der Abfrage erlaubt der Datenkontext auch Schreiboperationen. Listing 6 zeigt, wie ein neuer Mitarbeiter in SharePoint angelegt werden kann. Alle Aktionen in dem Datenkontext finden zunächst im Client statt. Erst der Aufruf von *SaveChanges()* sorgt für die eigentliche Übernahme, wobei sich der Entwickler keine Gedanken um die Änderungsverfolgung machen muss.

Für das dynamische Erstellen und Testen von LINQ-Abfragen ist das frei verfügbare Tool LINQPad nützlich, das auch zu OData-Diensten Verbindung aufnimmt [16]. Abbildung 4 zeigt die Abfrage aus Listing 5 in LINQPad. Nach der Verbindung mit dem OData-Service zeigt LINQPad zunächst auf der linken Seite die zur Verfügung stehenden Collections. Über das Abfragefenster lässt sich eine LINQ-Abfrage formulieren; LINQPad prüft sie, führt sie aus und zeigt das Ergebnis in der unteren Fensterhälfte an. Über die – etwas irreführende – Option *SQL* gibt LINQPad den URL aus, der nach der Übersetzung der LINQ-Abfrage auf dem Server aufgerufen wird. Im Beispiel aus Abbildung 4 handelt es sich um folgenden komplexen URL, der Filterung, Sortierung und auch die Projektion mittels *\$select* umfasst:

```
http://demo2010/sites/sdx/_vti_bin/
listdata.svc/Mitarbeiter(?)
$filter=substringof(
'Development',Kompetenzfeld/Title)
&$orderby=Title
&$select=Title,RolleValue
```

## EDM, CSDL und EDMX

Die Begriffe EDM, CSDL und EDMX haben ihren Ursprung in Microsofts Entity Framework [7]. Die Konzepte des Entity Data Models (EDM) gehen mittlerweile allerdings über das Entity Framework hinaus, das nur eine mögliche Implementierung des EDM darstellt.

EDM stellt das Modell einer Datenmenge dar. Es wird in der CSDL (Conceptual Schema Definition Language) als XML-Dialekt beschrieben [8]. CSDL gibt bereits primitive Datentypen vor, zum Beispiel *Edm.Int32*, *Edm.Bool*, *Edm.Single*, *Edm.Guid* und so weiter. Eine komplette Liste ist unter [9] zu finden.

Neben der CSDL gibt es im Entity Framework auch die Store Schema Definition Language (SSDL) zur Beschreibung des physischen Modells, und die Mapping Schema Language (MSL), welche die Informationen zum Übersetzen zwischen SSDL und CSDL liefert. CSDL, MSL und SSDL werden im EDMX-Format in einer XML-Datei zusammengefasst.

Mit LINQPad steht somit ein effizientes Werkzeug für die Abfrage von OData-Services zur Verfügung, womit sich LINQ-Abfragen leicht formulieren, analysieren und optimieren lassen.

Da OData auf den Prinzipien von REST beruht, bestehen grundsätzlich keine Einschränkungen hinsichtlich der Zugriffstechnologien. So sind OData-Dienste mit praktisch jeder Programmiersprache ansprechbar. Auch Silverlight kann via Ajax auf OData-Dienste zugreifen und damit beispielsweise auch Windows Phone 7, das Silverlight als Entwicklungsplattform verwendet. Weiterhin gibt es bereits Clientbibliotheken für das iPhone und Mac OS X (Objective C), JavaScript (Ajax, Palm WebOS), PHP und Java (Restlet Extensions).

### Was bringt das Open Data Protocol?

Sollen Daten und nicht Oberflächen über das Web zur Verfügung gestellt werden, so bietet die Verwendung von Microsofts Open Data Protocol einige interessante Perspektiven. Wesentlich ist die Mächtigkeit des Datenzugriffs über REST, womit sich leicht CRUD-Operationen über HTTP realisieren lassen. Die weitreichenden Möglichkeiten der Datenabfrage erlauben eine flexible Verarbeitung von Daten im Dienst, bevor er diese an den Client schickt. Das reduziert die Netzlast und erlaubt sehr effiziente Abfragen, beispielsweise mit LINQ.

Da OData auf eingeführten und weitverbreiteten Webstandards aufbaut, ist die Zusammenarbeit mit den unterschiedlichsten Systemen gewährleistet; zudem ist die Offenheit des Protokolls ein eindeutiger Pluspunkt. OData-Dienste bieten somit genau das, was sich Tim Berners-Lee mit seiner Idee eines Webs der offenen Daten vorstellt.

In diesem Zusammenhang kommt auch der Einheitlichkeit des Protokolls große Bedeutung zu. OData bietet mächtige CRUD-Funktionen und macht sie über einfache URL-Konventionen möglich. Das einheitliche und funktionale API erlaubt es, Tools zur Abfrage und Auswertung von OData-Diensten zu entwickeln. Auch Mashups aus mehreren Diensten lassen sich dadurch leicht konstruieren.

Sollen Daten als OData-Service bereitgestellt und dabei nicht die WCF Data Services in .NET verwendet werden, so ist die Modularität von OData ein wichtiger Aspekt. Entwickler müssen nur die Funktionen implementieren, die sie benötigen, um den Nutzern die gewünschten Abfragen zur Verfügung zu stellen. Somit wird der Implementierungsaufwand von OData-Diensten beschränkt.

## Listing 5

### Abfrage von SharePoint 2010 mit LINQ mit .NET 4.0.

```
var uri = new Uri(@"http://demo2010/sites/sdx/_vti_bin/listdata.svc/");
var context = new SharePointService.SDXDataContext(uri);
context.Credentials = SomeCredentials;

var mitarbeiterListe = from mitarbeiter
    in context.Mitarbeiter
    where mitarbeiter.Kompetenzfeld.Title.Contains("Development")
    orderby mitarbeiter.Title
    select new { Name = mitarbeiter.Title,
                Rolle = mitarbeiter.RolleValue };

foreach (var mitarbeiter in mitarbeiterListe)
    Console.WriteLine(mitarbeiter.Name + " - " + mitarbeiter.Rolle);
```

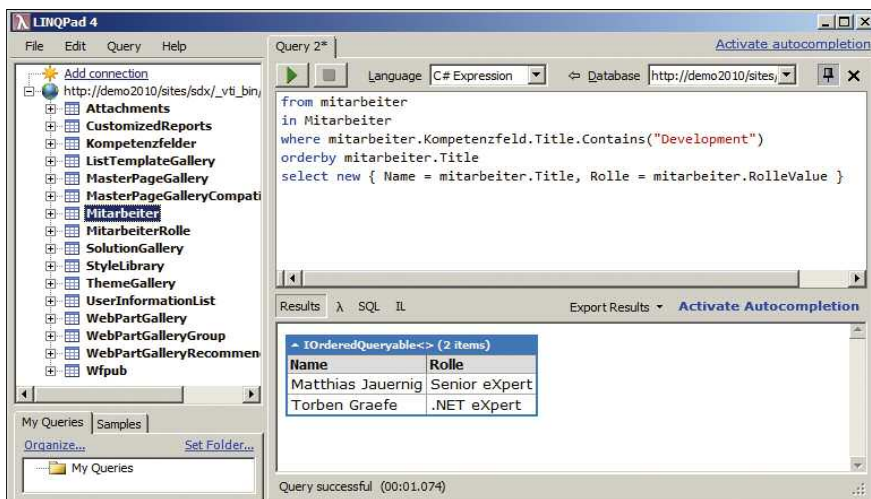
## Listing 6

### Hinzufügen eines Mitarbeiters in SharePoint 2010 mit .NET 4.0.

```
var uri = new Uri(@"http://demo2010/sites/sdx/_vti_bin/listdata.svc/");
var context = new SharePointService.SDXDataContext(uri);
context.Credentials = SomeCredentials;

var mitarbeiter = new MitarbeiterItem
{
    Title = "Neuer eXpert",
    Technologien = "Visual Studio 2010, .NET 4.0, SharePoint 2010"
};

context.AddToMitarbeiter(mitarbeiter);
context.SaveChanges();
```



[Abb. 4] LINQPad mit einer SharePoint-Abfrage.

## Fazit

OData ist als offenes Protokoll für den einheitlichen Zugriff auf Datendienste gedacht. Das Format zeigt, dass sich Microsofts Datenstrategie mittlerweile konkretisiert und Redmond dabei auch an Offenheit, Interoperabilität und an die Welt außerhalb von

.NET denkt. Darüber hinaus ist OData eine weitere Technologie, die auf dem Entity Data Model basiert, mit dem sich Daten standardisiert erzeugen und konsumieren lassen. Und Microsoft hat bereits ein beachtliches Ökosystem aus OData-Produzenten und -Konsumenten geschaffen [17] [18].

Es fragt sich, ob Microsoft mit OData außerhalb seines Kosmos Erfolg haben wird. Denn andere Anbieter verfolgen ähnliche Ansätze, wie Google mit GData [19].

Ein wichtiger Schritt in die richtige Richtung sind die OData-Client-SDKs, die für eine Vielzahl an Plattformen bereitstehen. Außerdem ist zu erkennen, dass die Anzahl an Webseiten und Anbietern, die OData nutzen, steigt [20].

Andererseits ist das Erzeugen von OData-Diensten außerhalb von .NET derzeit nur mit großem Aufwand möglich und hat mit dem EDM und der CSDL ein nicht zu unterschätzendes Maß an Komplexität. Es ist zu wünschen, dass Microsoft auch die Produktion von OData-Diensten aus anderen Programmiersprachen und IDEs heraus vereinfacht – und diese Aufgabe nicht nur der Community überlässt. [ljp]

- [1] Video „Tim Berners-Lee on the next Web“, [www.dotnetpro.de/SL10110data1](http://www.dotnetpro.de/SL10110data1)
- [2] Open Data Protocol, [www.odata.org](http://www.odata.org)
- [3] Microsoft Open Specification Promise, [www.dotnetpro.de/SL10110data2](http://www.dotnetpro.de/SL10110data2)
- [4] WCF Data Services Team Blog, [www.dotnetpro.de/SL10110data3](http://www.dotnetpro.de/SL10110data3)
- [5] OData-Modell der Northwind-Datenbank, [www.dotnetpro.de/SL10110data4](http://www.dotnetpro.de/SL10110data4)
- [6] The Atom Publishing Protocol, [www.dotnetpro.de/SL10110data5](http://www.dotnetpro.de/SL10110data5)
- [7] The ADO.NET Entity Framework, [www.dotnetpro.de/SL10110data6](http://www.dotnetpro.de/SL10110data6)
- [8] Conceptual Schema Definition File Format, [www.dotnetpro.de/SL10110data7](http://www.dotnetpro.de/SL10110data7)
- [9] OData Protocol Overview, [www.dotnetpro.de/SL10110data8](http://www.dotnetpro.de/SL10110data8)
- [10] OData Protocol URI Conventions, [www.dotnetpro.de/SL10110data9](http://www.dotnetpro.de/SL10110data9)
- [11] WCF Data Services, [www.dotnetpro.de/SL10110data10](http://www.dotnetpro.de/SL10110data10)
- [12] Bytes Matters: Enabling OData Endpoint When Creating WCF RIA Services, [www.dotnetpro.de/SL10110data11](http://www.dotnetpro.de/SL10110data11)
- [13] Microsoft, Codename „Dallas“ – Developer Portal, Catalog, [www.dotnetpro.de/SL10110data12](http://www.dotnetpro.de/SL10110data12)
- [14] MSDN: REST and LINQ in SharePoint 2010, [www.dotnetpro.de/SL10110data13](http://www.dotnetpro.de/SL10110data13)
- [15] PowerPivot, [www.powerpivot.com](http://www.powerpivot.com)
- [16] LINQPad, [www.linqpad.net](http://www.linqpad.net)
- [17] Practical OData – Building Rich Internet Apps with the Open Data Protocol, [www.dotnetpro.de/SL10110data14](http://www.dotnetpro.de/SL10110data14)
- [18] OData Protocol by Example, [www.dotnetpro.de/SL10110data15](http://www.dotnetpro.de/SL10110data15)
- [19] Google Data Protocol, [www.dotnetpro.de/SL10110data16](http://www.dotnetpro.de/SL10110data16)
- [20] OData Producers, [www.odata.org/producers/](http://www.odata.org/producers/)