SQL, LINQ-2-SQL, NuGet

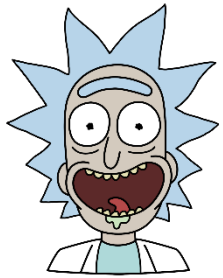
Aufbau von Datenbankverbindungen

Nutzung von LINQ für Datenbankabfragen

Benutzung von Fremdpaketen mittels NuGet

LINQ ist eine Abfragesprache für Objektlisten, die IEnumerable implementieren.
Und SQL ist auch eine Abfragesprache. Ist es möglich, die beiden Konzepte zu vereinen?

Ja. LINQ-2-SQL ist eine Schnittstelle, die es ermöglicht, mittels LINQ-Syntax direkt mit Datenbanken zu arbeiten. Dabei erhält man sehr viele Vorteile.



Datenbanken

- Speichern und Verwalten großer Mengen an Daten
- Datenabfrage erfolgt über Abfragesprache (üblich: SQL)
- Daten werden in Tabellen hinterlegt
- Daten können in Beziehung zueinander gesetzt werden

Beispieldatenbank: Cats and Owners

Tabelle: cats

id	name	owner_id	birth
int, PK	char[50]	int	date

Tabelle: owners

id	firstname	name	address
int, PK	char[150]	char[150]	char[150]

Aufbau einer SQL-Serververbindung in C#

Beispiel: MS SQL Server

```
static void Main(string[] args)
{
    string connectionString = "Data Source=gandalf.xsitepool.tu-freiberg.de;" +
        "Initial Catalog=master;User ID=sa;Password=**********";
    using (SqlConnection conn = new SqlConnection(connectionString))
    {
        try
        {
            conn.Open();
            Console.WriteLine("Verbindung hergestellt.");
            //conn.Close(); // durch using nicht notwendig am Ende
        }
        catch (SqlException e)
        {
            Console.WriteLine(e.ToString());
        }
    }
}
```

zugehörige Namespaces:

```
using System;
using System.Data.SqlClient;
```

Ausführung von SQL-Queries in C#

```
conn.Open();
Console.WriteLine("Verbindung hergestellt.");
using (SqlCommand cmd = new SqlCommand())
{
    SqlDataReader reader;
    cmd.CommandText = "SELECT * FROM cats";
    cmd.Connection = conn;

    reader = cmd.ExecuteReader();

    while(reader.Read())
    {
        Console.WriteLine(reader.GetName(0) + ":\t\t" + reader.GetInt32(0));
        Console.WriteLine(reader.GetName(1) + ":\t\t" + reader.GetString(1));
        Console.WriteLine(reader.GetName(2) + ":\t" + reader.GetInt32(2));
        Console.WriteLine(reader.GetName(3) + ":\t\t" +
            reader.GetDateTime(3).ToShortDateString() + "\n");
    }
    reader.close();
    Console.WriteLine("Verbindung wird geschlossen");
}
```

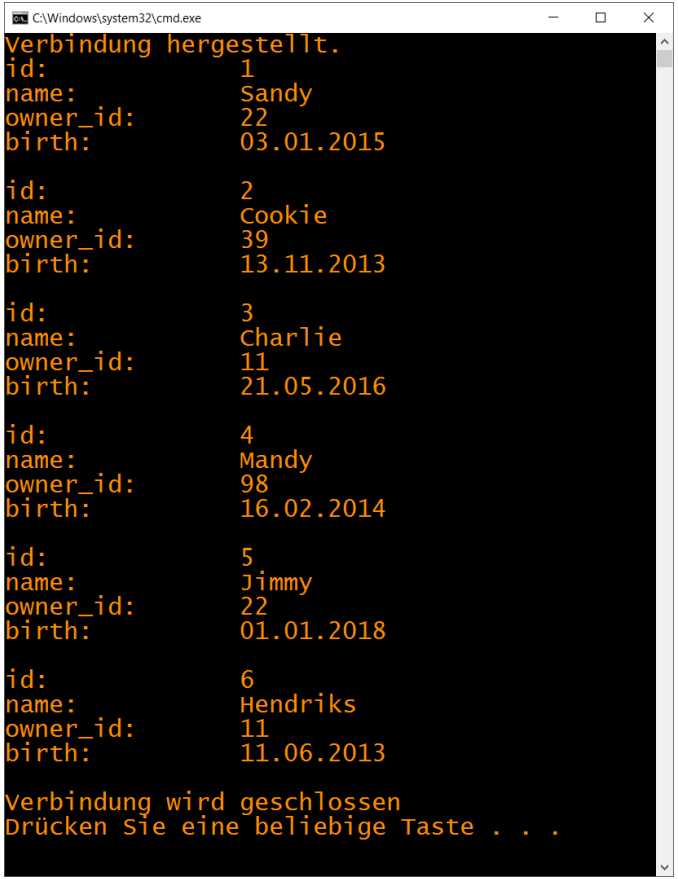
Vorsicht: Datenbankstruktur kann sich verändern

```
reader = cmd.ExecuteReader();

int id_column = reader.GetOrdinal("id");
int name_column = reader.GetOrdinal("name");
int ownerid_column = reader.GetOrdinal("owner_id");
int birth_column = reader.GetOrdinal("birth");

while (reader.Read())
{
    Console.WriteLine(reader.GetName(id_column) + ":\t\t" + reader.GetInt32(id_column));
    Console.WriteLine(reader.GetName(name_column) + ":\t\t" + reader.GetString(name_column));
    Console.WriteLine(reader.GetName(ownerid_column) + ":\t" + reader.GetInt32(ownerid_column));
    Console.WriteLine(reader.GetName(birth_column) + ":\t\t" +
        reader.GetDateTime(birth_column).ToShortDateString() + "\n");
}
reader.Close();
```

Die Anwendung ist auch nach einer Erweiterung oder Umsortierung der Spalten benutzbar.



- Daten können über spezialisierte Methoden genau angezeigt werden
- Wissen über die Struktur der Datenbank ist essentiell
- Alternative Methoden mittels Generics:

```
reader.GetFieldValue<int>(0)
reader.GetFieldValue<string>(1)
reader.GetFieldValue<int>(2)
reader.GetFieldValue<DateTime>(3)
```

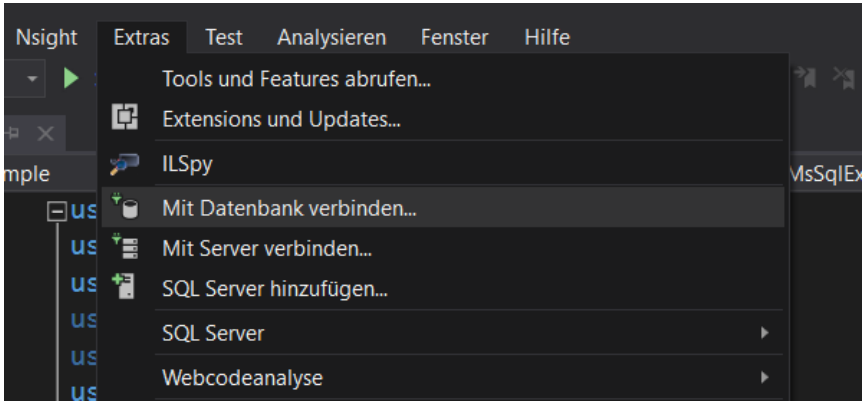
→ noch besser: Zugriff über LINQ

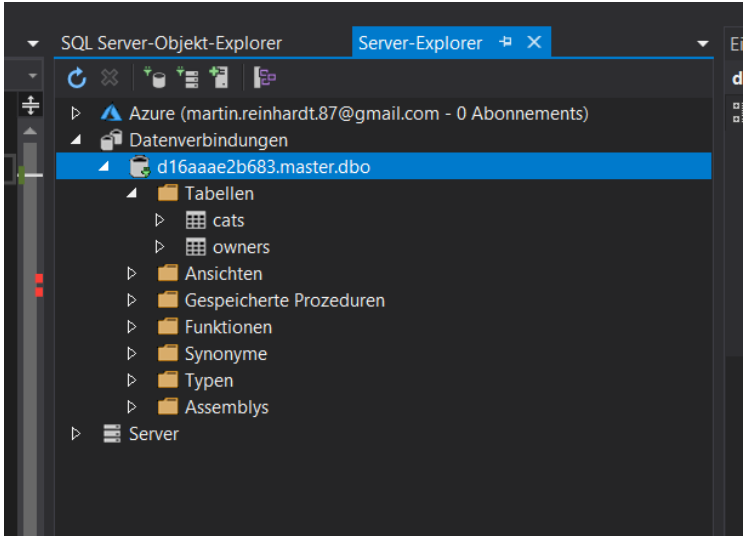
Besserer Ansatz: LINQ

- LINQ benötigt zur Verwendung mit MS SQL eine **DataContext-Klasse**
- Klasse kann in VS Studio und auch über die Kommandozeile automatisch generiert werden
- zuständiges Werkzeug: **sqlmetal** (implizit GUI oder explizit CMD)
- **DataContext-Klasse** erstellt sämtliche Werkzeuge und Datentypen

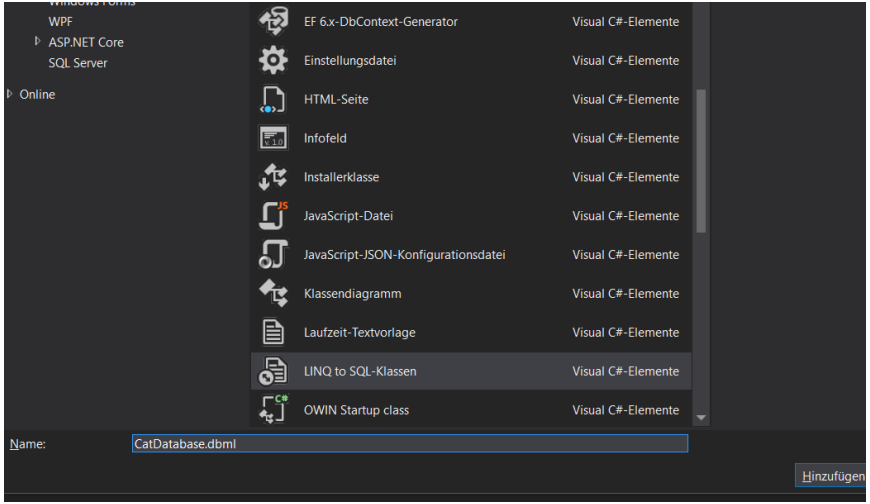
Ansatz 1 : Visual Studio

1. Verbindung mit dem MS SQL Server herstellen

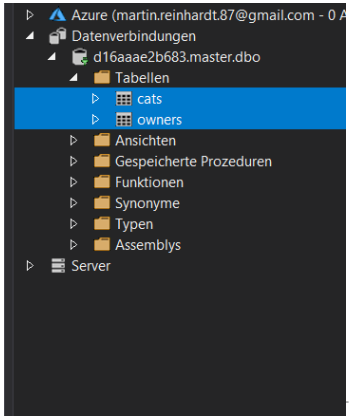




2. Erstellen einer LINQ-2-SQL-Klasse (CatDatabase.dbml)



3. Ziehen der Tabellen aus dem Server-Explorer in die Designer-Ansicht der neuen Klasse



4. Nutzung der generierten Klasse **CatDatabaseDataContext** mit der **using** Direktive zusammen mit der Serververbindung

```
string connectionString = "Data Source=gandalf.xsitepool.tu-freiberg.de;" +
    "Initial Catalog=master;User ID=sa;Password=*********";
using (SqlConnection conn = new SqlConnection(connectionString))
using (CatDatabaseDataContext myData = new CatDatabaseDataContext(conn))
{
    try
    {
        conn.Open();

...
```

Ansatz 2 : sqlmetal (Plattform-unabhängig)

Benutzung mit dem Connection String:

```
sqlmetal --conn="Data Source=gandalf.xsitepool.tu-
freiberg.de;Initial Catalog=master;User ID=sa;Password=********" --code=CatsAndOwnersContext.cs
--database=CatsAndOwnersContext --namespace=MsSqlExample
```

Danach kann die Klasse in das Projekt eingebaut werden.

Beispiel: SQL/LINQ-Abfrage-Vergleich

- Anfrage an die Datenbank:

  Name und Geburtstag aller Katzen, welche seit 2015 geboren wurden, geordnet nach ihrem Alter

- Vergleich: LINQ vs. SQL

Beispiel: SQL-Query

- SQL: **prepared statement**: Sicherheit gegen SQL-Injections

```
conn.Open();
using (SqlCommand sqlCmd = new SqlCommand(
    "SELECT name, birth FROM cats WHERE birth>=@startdate ORDER BY birth",conn))
{
    sqlCmd.Parameters.AddWithValue("@startdate", "01/01/2015");
    using (SqlDataReader reader = sqlCmd.ExecuteReader())
    {
        int birth_column = reader.GetOrdinal("birth");
        int name_column = reader.GetOrdinal("name");
        while (reader.Read())
        {
            Console.WriteLine(reader.GetFieldValue<DateTime>(birth_column).ToShortDateString() +
                "\t " + reader.GetFieldValue<string>(name_column));
        }
    }
}
```

Beispiel: LINQ-Query

- Datentypen für alle Tabellen wurden automatisch generiert (cats, owners)
- sämtliche Tabellen implementieren das **IEnumerable-Interface**
- LINQ ist auf die Datentypen anwendbar:

```
using (CatDatabaseDataContext myData = new CatDatabaseDataContext(conn))
```

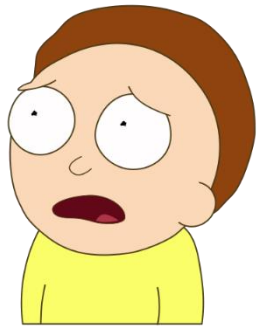
```
conn.Open();
IEnumerable<cats> query = myData.cats
    .Where(c => c.birth >= new DateTime(2015, 1, 1))
    .OrderBy(c => c.birth);
foreach (var element in query)
{
    Console.WriteLine($"{element.birth.ToShortDateString()} \t {element.name}");
}
```

- In beiden Fällen wird das gleiche Ergebnis ausgegeben:

```
C:\Windows\system32\cmd.exe
03.01.2015      Sandy
21.05.2016      Charlie
01.01.2018      Jimmy


Drücken Sie eine beliebige Taste . . .
```

Und was ist jetzt besser?



LINQ-2-SQL vs. pure SQL

Vorteile:

- Unterstützung des Compilers
- Typsicherheit
- Unterstützung der IDE (MS Visual Studio)
- Gleichbleibendes Arbeitsmuster
- Lazy-Loading (Ausführungszeit ist bestimmbar)

Nachteile:

- Einfügen einer weiteren Codeebene

In Bezug auf Leistung kann es Verbesserungen sowie Verschlechterungen geben

vollständige Tabellen

cats:

id	name	owner_id	birth
1	Sandy	22	03. 01. 2015
2	Cookie	39	13. 11. 2013
3	Charlie	11	21. 05. 2016
4	Mandy	98	16. 02. 2014
5	Jimmy	22	01. 01. 2018
6	Hendriks	11	11. 06. 2013

owners:

id	firstname	name	address
22	John	Lennon	251 Menlove Avenue
39	Casey	Adams	Somewhere over the Rainbow
11	River	Goldstein	NULL
13	Rudolph	Catless	John‘s Dog House
98	Max	NULL	Terrier Drive 48

Weitere LINQ-2-SQL Abfragen: Joins

Joins: Katzen und ihre Adressen

```
var query2 = myData.cats
    .Join(myData.owners, c => c.owner_id, o => o.id, (c, o) => new { c.name, o.address })
    .Where(q => q.address != null);

foreach (var element in query2)
{
    Console.WriteLine(element.name +"\t\t" + element.address);
}
```

Joins: Besitzer und die dazugehörigen Katzen

```
var query3 = myData.owners
    .Join(myData.cats, o => o.id, c => c.owner_id,
        (o, c) => new { o.firstname, o.name, catname = c.name })
    .Where(e => e.firstname != null)
    .Where(e => e.name != null);

foreach (var element in query3)
{
    Console.WriteLine(element.firstname + " " + element.name + ":\t\t" + element.catname);
}
```

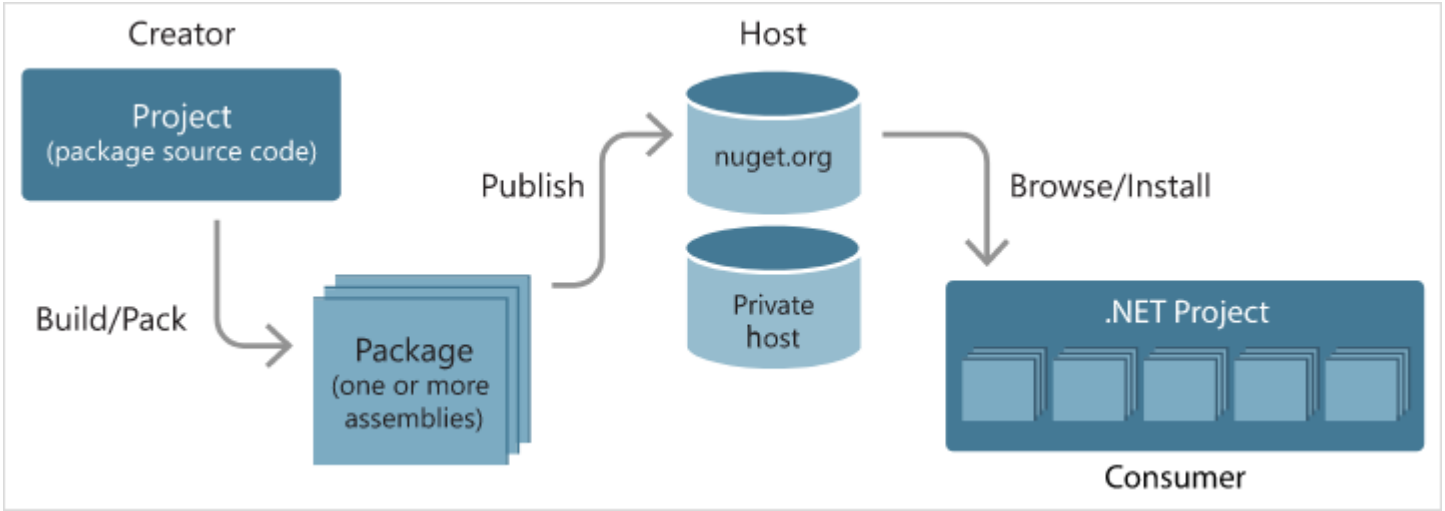
Benutzung von MySQL

- MS SQL ist eher im Enterprise-Segment zu finden
- Es gibt viele verschiedene SQL-Dialekte
- System.Data.SqlClient und .NET unterstützt nur MS SQL
- Für MySQL werden externe Pakete benötigt
- Fremdpakete → NuGet

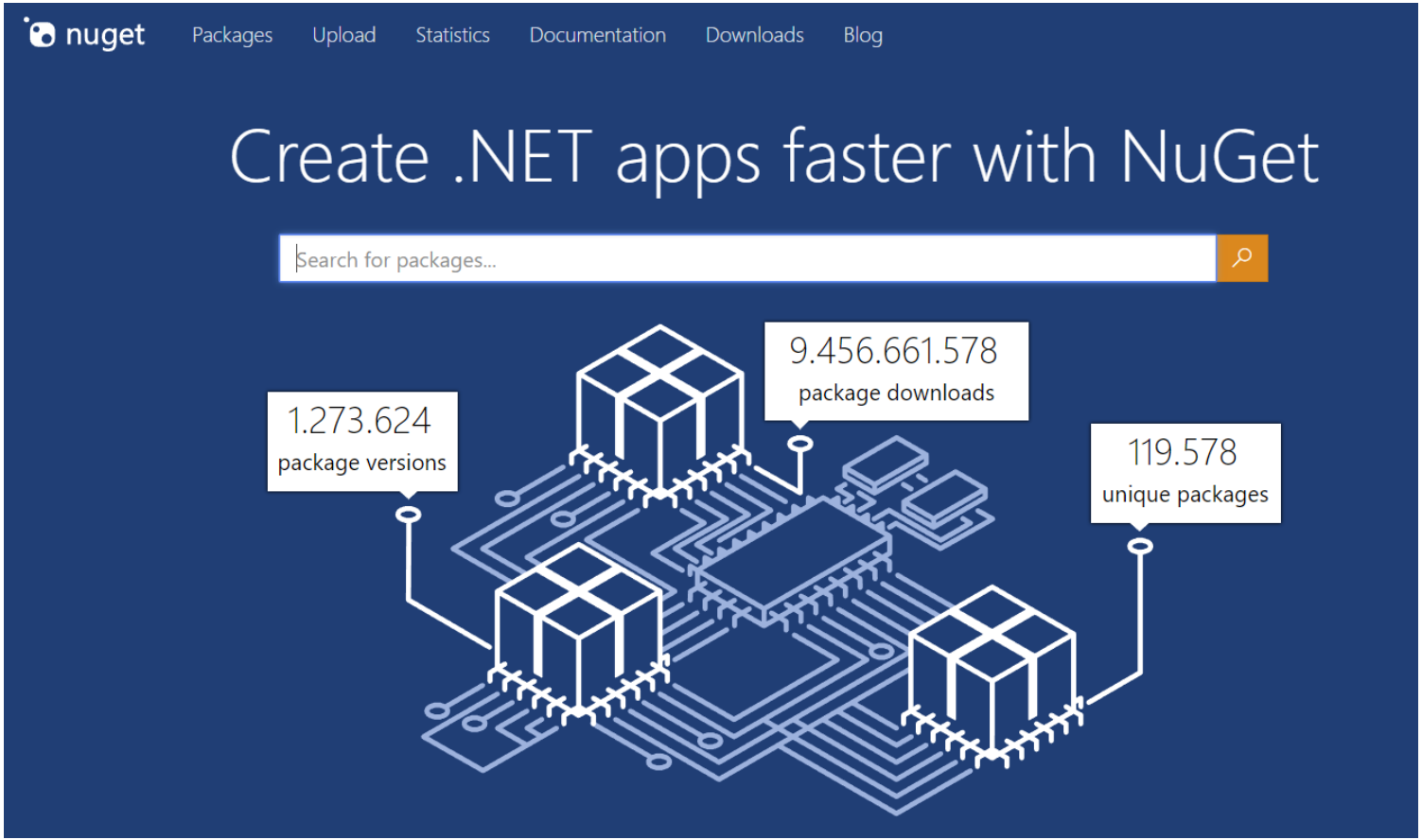
NuGet – der .NET Paketmanager

- Nutzung von Fremdpaketen in .NET und .NET Core

- NuGet-Pakete liegen entweder zentral auf nuget.org oder privaten Hosts

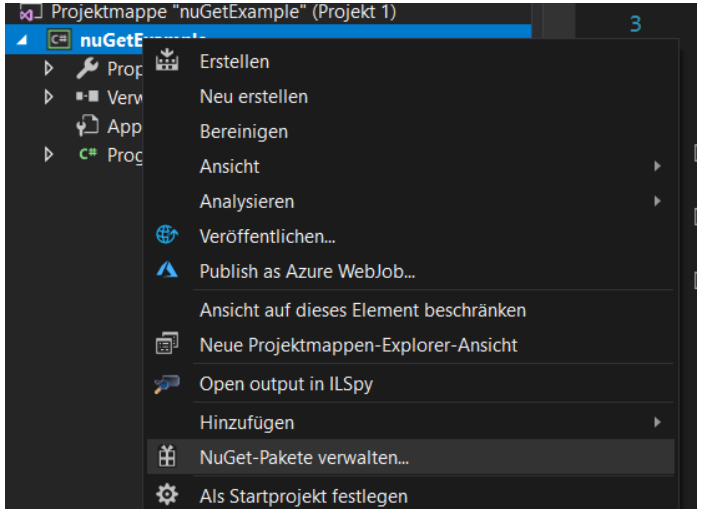


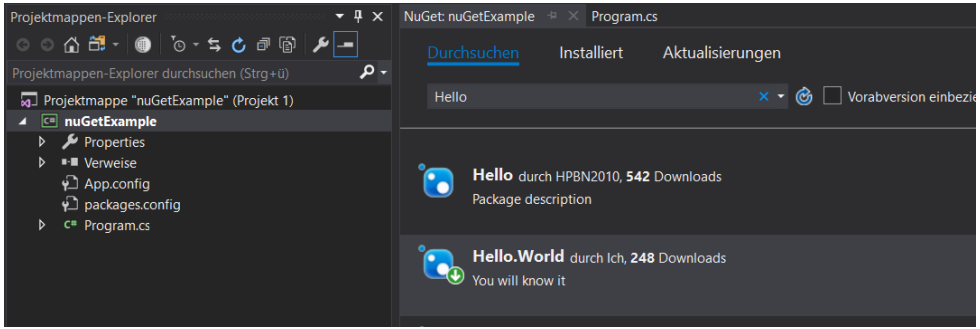
nuget.org


nuget
Packages
Upload
Statistics
Documentation
Downloads
Blog
Create .NET apps faster with NuGet
Search for packages...
9.456.661.578
package downloads
1.273.624
package versions
119.578
unique packages

Installieren neuer Pakete: MS Visual Studio

- Pakete sind Projekt-Gebunden





- spezielle Versionen/Revisionen ebenfalls auswählbar

- Pakete werden automatisch in die **packages.config** eingetragen

```xml
<?xml version="1.0" encoding="utf-8"?>
<packages>
  <package id="Hello.World" version="1.0.2" targetFramework="net47" />
</packages>
```


```csharp
namespace nuGetExample
{
    class Program
    {
        static void Main(string[] args)
        {
            HelloWorld.Hello hello = new HelloWorld.Hello();
            hello.World();
        }
    }
}
```

Installieren neuer Pakete: Kommandozeile:

- Erst Herunterladen von nuget.exe (oder die Mono-Variante)

- Installieren des Paketes mit **nuget.exe install** (lokal)

```
>nuget install Hello.World
Feeds used:
  https://api.nuget.org/v3/index.json
  C:\Program Files (x86)\Microsoft SDKs\NuGetPackages\
```

- Nutzung der Assembly mit **/ref:HelloWorld.dll**

```
>copy Hello.World.1.0.2\lib\netstandard2.0\HelloWorld.dll .
        1 Datei(en) kopiert.

>csc nugetExample.cs /r:HelloWorld.dll
Microsoft (R) Visual C# Compiler Version 2.8.3.62923 (7aafab56)
Copyright (C) Microsoft Corporation. Alle Rechte vorbehalten.
```

- NuGet beinhaltet knapp 120.000 einzelne Pakete
- NuGet bildet eine wichtige Säule in der Entwicklung von C#-Anwendungen
- NuGet bildet eine Quelle für sinnvolle Ergänzungen der Standard Klassenbibliothek
- über NuGet lassen sich externe Abhängigkeiten mit der Datei **packages.config** auch im Team managen