
Continental
The Future in Motion
C1TT
Programmieren mit C#
ASP.NET
C1TT
Continental Institut
für Technologie
und Transformation

ADO.NET

Aufbau von ADO.NET

- Verwaltete Provider
  - Auf bestimmtes DBMS oder Datenzugriffstechnologie spezialisiert
  - Stellen Verbindung zur DB her
  - Lesen und manipulieren Daten über SQL-Anweisungen
- *DataSet*- und *DataTable*-Klassen
  - Unabhängig von DBMS oder Datenzugriffstechnologie
  - Verwalten Daten auf dem Client
  - Speichern Änderungen zunächst lokal
  - Können Daten im XML-Format speichern
- Web- oder Windows-Steuerelemente
  - Binden beliebige Eigenschaften an Datenmenge
  - Datenmenge kann auch Array o.ä. sein

Verwaltete Provider (1/6)

- .NET: Provider für SQL Server (Compact), Oracle, OLE DB, ODBC
- Drittanbieter: z. B. Provider für Firebird oder MySQL
- Normalerweise verwalteter Code
  → Ausnahme: Legacy
- Keine unnötigen Schichten, möglichst nur .NET-Provider benutzen

Verwaltete Provider (2/6)

- Verwaltete Provider bestehen aus
  - `Connection`-Objekt für DB-Verbindung
  - Datenlese-Objekt
  - Befehlsobjekte, repräsentieren SQL-Anweisungen
  - Datenadapter-Objekt zum Verbinden mit DataSet-Objekt

- Jeder verwaltete Provider
  - Benutzt anderes Präfix: Sql, Oledb, Odbc, usw.
  - Implementiert gleiche Schnittstellen: IDbConnection, etc.
  - Trotz anderer Klassennamen gleiche Eigenschaften + Methoden

Verwaltete Provider (3/6)

- Die `SqlConnection`-Klasse
  - `ConnectionString`
  - `Open()` / `Close()`
  - `ConnectionTimeout`, `PacketSize`
  - `BeginTransaction()` → `SqlTransaction`, `Commit()`, `Rollback()`
  - `Close()` / `Dispose()`

Verwaltete Provider (4/6)

- Namespace `System.Data.SqlClient`
- Die `SqlCommand`-Klasse
  - `CommandText`
    - Transact-SQL
    - Tabellenname
    - Gespeicherte Prozedur
  - `CommandType`
  - `CommandTimeout`
  - `Parameters` → evtl. auch benannte
  - `Connection`, `Transaction`
  - `ExecuteNonQuery()` → Anzahl betroffener DS
  - `ExecuteReader()` → SqlDataReader
  - `ExecuteScalar()` → Einzelwert
  - `Dispose()`

Demo

DATEN LESEN

Verwaltete Provider (5/6)

- Die `SqlDataReader`-Klasse
  - Schnell, aber nur lesen + nur vorwärts
  - `GetInt32()`, `GetString()` → `IsDBNull()` nicht vergessen!
  - `Read()` → nächster DS
  - Erzeugung über `SqlCommand.ExecuteReader()`
  - `Close()`, macht `SqlConnection` wieder verfügbar

Verwaltete Provider (6/6)

- Die `SqlDataAdapter`-Klasse
  - Navigierbare Datenmenge, benutzt `SqlDataReader`-Klasse
  - `SelectCommand` = SELECT + `SqlConnection`
  - `Fill()` füllt `DataTable`
  - `Update()` speichert `DataTable`
    - Sendet für jeden DS 1 SQL-Anweisung an DB
      - UpdateBatchSize für SQL Server
    - Braucht dazu `DeleteComand`, `InsertCommand`, `UpdateCommand`
    - `SqlCommandBuilder` machts einfach
      - `SetAllValues`-Eigenschaft
      - `ConflictOption`-Eigenschaft
- `TableAdapter`-Klasse

Datenbankunabhängige Klassen (1/4)

- Namespace `System.Data`
- Die `DataColumn`-Klasse
  - Beschreibt Tabellenspalte:
    - `AllowDBNull`, `AutoIncrement`, `ColumnName`, `DataType`, `ReadOnly`, `Unique`, etc.
  - In `Columns`-Auflistung des `DataTable`-Objekts verwaltet
  - Beim Füllen des `DataTable`-Objekts automatisch erzeugt
  - Programmgesteuertes Erzeugen möglich

Datenbankunabhängige Klassen (2/4)

- Die `DataRow`-Klasse
  - Enthält Felder eines Datensatzes
  - Zugriff auf Feldwerte
    - `Item`, `ItemArray`
    - Datentyp = `System.Object`
    - `DataRowVersion.Current`, `DataRowVersion.Original`
    - `DataRowState.Modified`, `DataRowState.Unchanged`
  - In `Rows`-Auflistung des `DataTable`-Objekts verwaltet
  - Datensatz anlegen: `NewRow()` + `Rows.Add()`
  - Datensatz ändern: Indexer, evtl. `BeginEdit()` + `EndEdit()`
  - Datensatz löschen: `Delete()`, **nicht** `Remove()`
  - Zurückschreiben in DB i. A. mit `Update()` eines Datenadapters

Datenbankunabhängige Klassen (3/4)

- Die `DataTable`-Klasse
  - Repräsentiert Datemenge
  - Füllen mit `Fill()` eines Datenadapters
  - Programmgesteuertes Füllen möglich
  - `Columns`- und `Rows`-Auflistung, `Constraints`
  - Zurückschreiben in DB i. A. mit `Update()` eines Datenadapters
    - Problematisch bei mehreren Tabellen!
    - `AcceptChangesDuringUpdate`-Eigenschaft

Demo

DATEN LESEN UND VERÄNDERN

Datenbankunabhängige Klassen (4/4)

- Die `DataSet`-Klasse
  - Verwaltet mehrere Datemengen
  - `Tables`- und `Relations`-Auflistung
    → Kann ganze relationale DB aufnehmen
  - Speichern im XML-Format mit `WriteXml()`
    - Daten + Schema
    - Daten + Änderungen
    - Nur die Änderungen: `GetChanges()` + `WriteXml()`

Demo

DATASET ALS XML SPEICHERN

ADO.NET-Unterstützung in Visual Studio (1/3)

Continental Institut für Technologie und Transformation

- Rapid Application Development
- Oberfläche generieren lassen
- Features in Visual Studio
  - Datenquellen-Fenster
  - Assistenten zum Konfigurieren von Datenquellen
  - Z. B. Tabelle auf Formular ziehen
    ➔ Benötigte Steuerelemente entstehen automatisch!
  - `DataGridView` ersetzt `DataGrid`
  - `BindingNavigator`
  - Generierte Komponenten auf Registerkarte Datenquellen Komponenten verfügbar
  - `DataSet`-Designer
  - Datenbankdateien werden kopiert

ADO.NET-Unterstützung in Visual Studio (2/3)

Continental Institut für Technologie und Transformation

- Typisiertes DataSet
- Vorher / nachher:

```
// DataSet:
string nachname = dsNamen.Tables["Namen"].Rows[idx]["Nachname"].ToString();


// Typisiertes DataSet:
string nachname = dsNamen.Namen[idx].Nachname;
```

- Vorteile
  - Code ist verständlicher
  - Typsichere Eigenschaften ersparen
    - ständige Typumwandlung
    - Laufzeitfehler durch Tippfehler im Eigenschaftennamen
  - Visual Studio kann IntelliSense und Code-Vervollständigung bieten
- Aufbau
  - `BindingSource`
  - `TableAdapter`
  - ...

ADO.NET-Unterstützung in Visual Studio (3/3)

- Datenbindung der Steuerelemente
  - Steuerelemente zeigen autom. aktuelle Daten
  - Änderungen werden autom. **Lokal (!)** gespeichert
  - Voraussetzung für Datenbindung zur Designzeit
    - Konfigurierte `DataSet`-Komponente im Projekt
  - Einfache Datenbindung
    - Ein Feld im aktuellen Datensatz
    - Nicht nur `Text`, `Checked`, … bindbar
    - Z. B. Vorder- und Hintergrundfarbe aus DB steuern
  - Komplexe Datenbindung
    - Mehrere Felder, mehrere Datensätze
    - Z. B. `GridView`
    - Listen- + Kombinationslistenfeld: einfache + komplexe Bindung

Demo

DATENQUELLE KONFIGURIEREN UND EINSETZEN

Entity Framework

- Objektrelationaler Mapper (ORM)
  - Mapping eines Datenbankmodells in ein Objektmodell
  - Lesende und schreibende Zugriffe
  - Kein SQL mehr im Programmcode
    → Compilerfehler, statt Laufzeitfehler wegen strenger Typisierung
  - Vollständige Intellisense auf **alle** Datenbankobjekte

- Zugriff mittels LINQ to Entities
  - Eingeführt mit dem .NET Framework 3.5

Demo

ENTITY FRAMEWORK MODELL ERSTELLEN UND DATEN AUSLESEN