
Karl-Peter-Obermaier-Schule

Staatliche Berufsschule 1 Passau

Staatliche Fachschulen für Elektrotechnik und Maschinenbautechnik



UNTERRICHTSPROJEKT FUNEVENTS

TEIL II DESIGN UND REALISIERUNG DES EVENT MANAGEMENT SYSTEMS (EMS)

MODUL 1 ENTWURF UND IMPLEMENTIERUNG DES EMS V1.0

Inhalt

1	VORÜBERLEGUNGEN	1
1.1	Die grundsätzliche Architektur des EMS 1.0	1
1.2	Der Funktionsumfang des EMS 1.0	1
2	DAS ERSTELLEN DER EINZELPROJEKTE	2
2.1	Die Mittelschicht ManageEvents.dll	2
2.1.1	Die Verbindung zur Datenbank	3
2.1.2	Das Generieren eines typed DataSets	3
2.1.3	Das Konfigurieren der TableAdapter - Klassen	5
2.2	Das Userinterface ManageEventsApp.exe	8
3	LESEN DER DATEN ÜBER DIE MITTELSCHICHT (UC 04.14)	10
3.1	Das Erstellen der TableAdapter	10
3.2	Das Füllen des Datasets mit Daten	11
4	DAS GRAFISCHE DESIGN DER BENUTZERSCHNITTSTELLE	12
4.1	Änderungen am ManageEventsWindow	12
4.2	Die Steuerelemente des ManageEventsWindows	12
4.3	Das Menü des ManageEventsWindows	13
4.4	Die Anzeige der Daten im User Interface	13
4.4.1	Datenquellen hinzufügen	13
4.4.2	Der Aufbau der Datenbindung	14
4.4.3	Die Anzeige der Events	15
4.4.4	Die Anzeige der Eventdaten	16
4.4.5	Das Binden der Datenquelle zur Laufzeit	17
4.5	Kategorie und Eventveranstalter	18
4.5.1	Die DataBindings für die Comboboxen	18
4.5.2	Erstellen der Databindings für die Comboboxen	19
5	LADEN DER EVENTS AUS DER DATENBANK (UC 04.14)	20
5.1	Die Methode ladenAusDatenbankMenuItem_Click(...)	20
6	EINE ERWEITERUNG DER ANZEIGE	22
6.1	Die Anzeige einer Navigationsleiste	22
6.2	Die Formatierung des Eventpreises in Euro	24
7	DIE LOKALE VERWALTUNG DER EVENTDATEN (UC 04.13)	26
7.1	Das lokale Speichern des Datasets	26
7.2	Das lokale Laden des Datasets	27

1 Vorüberlegungen

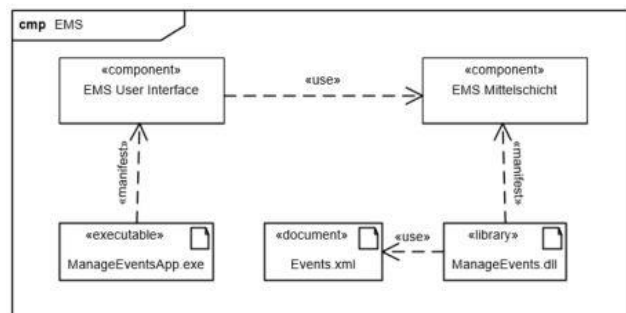
Aus der Analysephase des Projekts ergeben sich zwei unterschiedliche Teilprojekte. Das Kundenbuchungssystem (KBS) zum Vermitteln von Events an die Kunden für die Mitarbeiter der Eventagentur sowie ein Eventmanagementsystem (EMS) zur Verwaltung der Events durch den Geschäftsführer Herrn Fringsen.

Dieser Teil des Skripts beschäftigt sich mit dem Design und der Realisierung des EMS mit Hilfe der UML sowie der Microsoft Programmiersprache C#.

Das EMS wird entsprechend dem Spiralmodell in Teilschritten implementiert, um die Zwischenschritte jederzeit testen und verbessern zu können. Der erste Teil des EMS 1.0 beschäftigt sich dabei mit dem Zugriff auf die Events und Eventdaten über eine Mittelschichtkomponente sowie dem Erstellen eines typed DataSets.

1.1 Die grundsätzliche Architektur des EMS 1.0

Aus den Anforderungen an das EMS, insbesondere der späteren Erweiterbarkeit für Online-Updates der Eventdaten mit der Datenbank durch den Geschäftsführer ergibt sich ein modularer Aufbau des Systems. Modulare Systeme sind besser wieder verwendbar und erweiterbar als monolithische Applikationen. Das EMS soll daher in eine ausführbare Datei (ManageEventsApp.exe) für das Userinterface sowie in eine Mittelschichtkomponente (ManageEvents.dll) zur Kontrolle der Geschäftsregeln und zum Datenbankzugriff unterteilt werden. Die lokale Speicherung der Daten erfolgt im xml – Format in der Datei Events.xml. Spätere Erweiterungen können somit problemlos auf die Funktionalität der Mittelschicht zugreifen, ohne diese neu implementieren zu müssen.



1.2 Der Funktionsumfang des EMS 1.0

Für den ersten Durchgang der Entwicklung im Spiralmodell werden als Ziel die Eventanzeige sowie das Laden der Eventdaten aus der Datenbank gesetzt. Ebenso sollen die Daten lokal auf dem Laptop gespeichert und geladen werden können, wenn der Geschäftsführer nicht in der Geschäftsstelle der FunEvents GmbH anwesend ist und somit keinen Zugang zur Datenbank besitzt. Dazu zählen die Use Cases

- UC 04.2 Eventdaten anzeigen,
- UC 04.13 Events lokal verwalten sowie
- UC 04.14 Events mit Datenbank abgleichen.

2 Das Erstellen der Einzelprojekte

Das Projekt EMS umfasst eine Applikation sowie eine weitere Programmbibliothek. Zur besseren Verwaltbarkeit sollen alle Module in einer Projektmappe zusammengefasst werden.

☞ *Erstellen Sie mit dem Visual Studio eine neue, leere Projektmappe mit der Bezeichnung EMS_2024.*

2.1 Die Mittelschicht ManageEvents.dll

Als Mittelschicht zum Datenbankzugriff und zum Ausführen der Geschäftsregeln dient eine Dynamic Link Library. In der Library werden anschließend alle benötigten Klassen implementiert.

☞ *Erstellen Sie im Visual Studio.NET eine neue Klassenbibliothek mit der Bezeichnung ManageEvents.*

☞ *Verwenden Sie als Zielframework .NET 5.0.*



Die Hauptklasse der Bibliothek ManageEventsModule soll den Zugriff auf die Datenbank beinhalten.

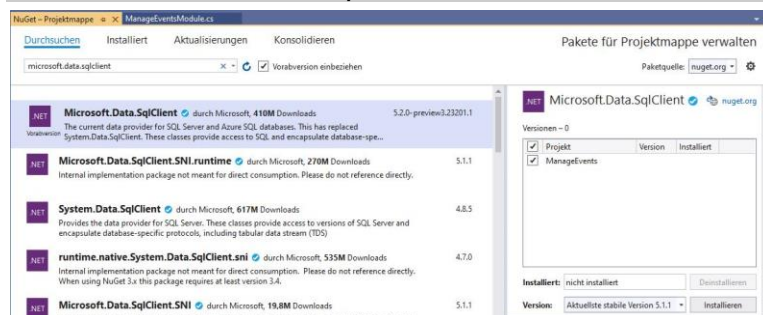
☞ *Ändern Sie den Namen der Klasse Class1 im Projektmappenexplorer auf ManageEventsModule um und bestätigen Sie ggf. die Änderungen!*

In der Klasse ManageEventsModule benötigt man einen öffentlichen Konstruktor für spätere Initialisierungen.

☞ *Erstellen Sie den öffentlichen Konstruktor der Klasse ManageEventsModule.*

Zur Unterstützung des Datenzugriffs auf die Datenbank KundenBuchungsSystem sind im nächsten Schritt zunächst ein .NET NuGet Paket zu installieren sowie die entsprechenden Namensräume einzubinden.

☞ *Öffnen Sie den NuGet Paket-Manager (Menü Extras/ NuGet Paket-Manager/ NuGet Pakete für Projektmappe verwalten...) und installieren die aktuellste stabile Version des Pakets Microsoft.Data.SqlClient!*



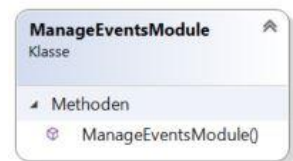
☞ *Binden Sie die Namensräume System.Data sowie Microsoft.Data.SqlClient im Quelltext des ManageEventsModules ein.*

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

☞ *Erstellen Sie das Projekt (Erstellen/ManageEvents erstellen).*

Microsoft bietet ab dem VS 2005 einen integrierten Klassendesigner an. Dieser ist zwar nicht zu 100% kompatibel mit der UML Normung, ermöglicht aber Design und Quellcodebearbeitung in einem Schritt, weshalb dieser bei der Implementierung der Mittelschicht bei allen folgenden Elementen zum Einsatz kommt.

- ☛ Fügen Sie dem Visual Studio-Projekt ein Klassendiagramm hinzu (Rechtsklick auf ManageEvents, Hinzufügen, Neues Element, Klassendiagramm).
- ☛ Benennen Sie das Klassendiagramm in EMS_Mittelschicht.cd um.
- ☛ Ziehen Sie das ManageEventsModule auf das Klassendiagramm.



2.1.1 Die Verbindung zur Datenbank

Über das EventManagementModul sollen alle Inserts, Updates und Select-Befehle an die Datenbank abgewickelt werden. Dazu benötigt man bei der Arbeit mit ADO.NET und der .NET-Plattform zunächst ein Connection Objekt, das eine Verbindung zur Datenbank herstellt.

- ☛ Öffnen Sie die Quelltextdatei ManageEventsModule.cs.
- ☛ Fügen Sie als private Attribut eine SqlConnection cnEvents im Quelltext ein.
- ☛ Erzeugen Sie das Objekt im Konstruktor

Im Konstruktor wird das SqlConnection Objekt erzeugt und im Anschluss daran der ConnectionString gesetzt.

```
public ManageEventsModule( )
{
    cnEvents = new SqlConnection( );
    ...
}
```

Für die korrekte Verbindung zur Datenbank muss die ConnectionString Property des Connection-Objektes gesetzt werden. In ihr sind der Servername, die Datenbankbezeichnung sowie die Anmeldeinformationen an die Datenbank gespeichert.

```
cnEvnets.ConnectionString =
@"Data Source=ITSW16SQL2\SQL_A; Initial Catalog=FunEvents21ITA0xx; Integrated Security=True; Encrypt = False";
```

- ☛ Setzen Sie den ConnectionString mit Servername, ihrer Datenbank sowie den Sicherheitseinstellungen im Konstruktor!
- ☛ Tipp: Als Hilfe zum Connection String kann eine *.udl Datei erzeugt werden.
- ☛ Begutachten Sie auch das Klassendiagramm von ManageEventsModule.

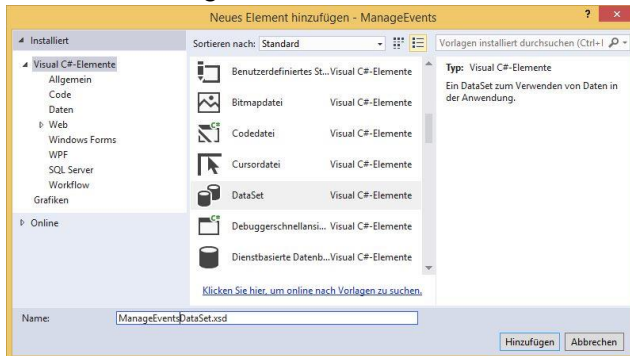
2.1.2 Das Generieren eines typed DataSets

Entsprechend den Ergebnissen aus der Analyse der Anforderungen übernimmt der Geschäftsführer der FunEvents GmbH die Verwaltung und Rekrutierung neuer Events bei den Eventveranstaltern. Dazu benötigt er Zugriff auf bestimmte Daten der Datenbank.

- ☛ Überlegen Sie, welche Tabellen der Datenbank zur Verwaltung der Events benötigt werden.

Da der Geschäftsführer die Daten neuer Events auf seinem Laptop zwischenspeichert und diese erst am Abend im Geschäft mit der Datenbank abgleicht, soll als lokaler Puffer der Daten auf dem Laptop ein DataSet zum Einsatz kommen. Das Update mit der Datenbank erfolgt in ADO.NET deshalb über DataAdapter Objekte mit den zugehörigen Command Objekten. Genauer gesagt verwendet das Visual Studio ab der Version 2008 sogenannte TableAdapter Klassen, welche von DataAdapter Klassen abgeleitet sind.

Nach dem Erstellen der Connection kann nun ein Dataset zum Speichern der Daten in der ManageEvents.dll generiert werden. Zum Einsatz kommt ein so genanntes typed Dataset, das neben den Daten auch eine Beschreibung des Datenaufbaus enthält. Dieses Schema wird in der .NET-Plattform als XSD-Datei gespeichert und umfasst die Beschreibung der Tabellen, Spalten mit Datentypen sowie die Relationen und Constraints. Aufbauend auf dieses XSD-Schema erzeugt das Visual Studio ein Dataset Objekt.



☞ *Fügen Sie dem Projekt ein neues Dataset hinzu (Projekt, Neues Element hinzufügen, Dataset) und benennen Sie dieses in ManageEventsDataset um.*

☞ *Ziehen Sie per Drag and Drop die Tabellen tbl_Events, tbl_EventDaten, tbl_EvKategorie und tbl_EvVeranstalter aus dem Server-Explorer auf den Editor.*

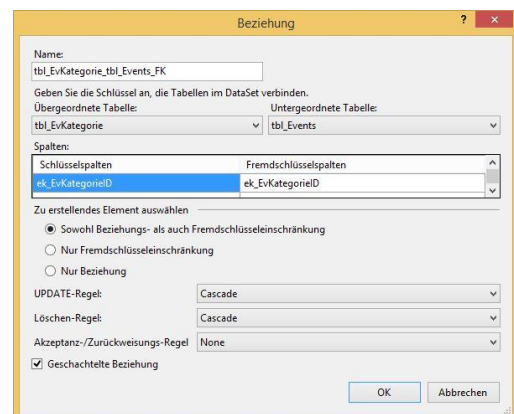
Zusätzlich zu den Tabellen im Dataset sind die Primary Keys und Relationen erstellt worden. Die Foreign Key Constraints sind allerdings noch manuell nachzuarbeiten.

☞ *Öffnen Sie den Grafikeditor der Relationen durch einen Doppelklick auf die Relation zwischen Kategorie und Eventstabelle.*

☞ *Ändern Sie den Namen in tbl_EvKategorie_tbl_Events_FK um.*

☞ *Setzen Sie die Option „Sowohl Beziehungs- als auch Fremdschlüsseleinschränkung“.*

☞ *Für die Aktualisierungs- und Löschregel kann Cascade gesetzt werden.*



Dadurch entsteht eine Teilabbildung der Datenbank als lokaler Datenpuffer zum Zwischenspeichern der Daten im Dataset. Das Dataset ist für die referentielle Integrität verantwortlich.

Wie in einer relationalen Datenbank besteht nun eine 1 zu n Relation zwischen beiden Tabellen, welche zur Einhaltung der referentiellen Integrität sowie zur Navigation zwischen Eltern- und Kind-Datensätzen genutzt werden kann.

☞ *Wiederholen Sie die Bearbeitung der Beziehungen mit den Relationen tblEvVeranstalter_tbl_Events_FK und tbl_Events_tbl_EvDaten_FK entsprechend dem Aufbau der Datenbank.*

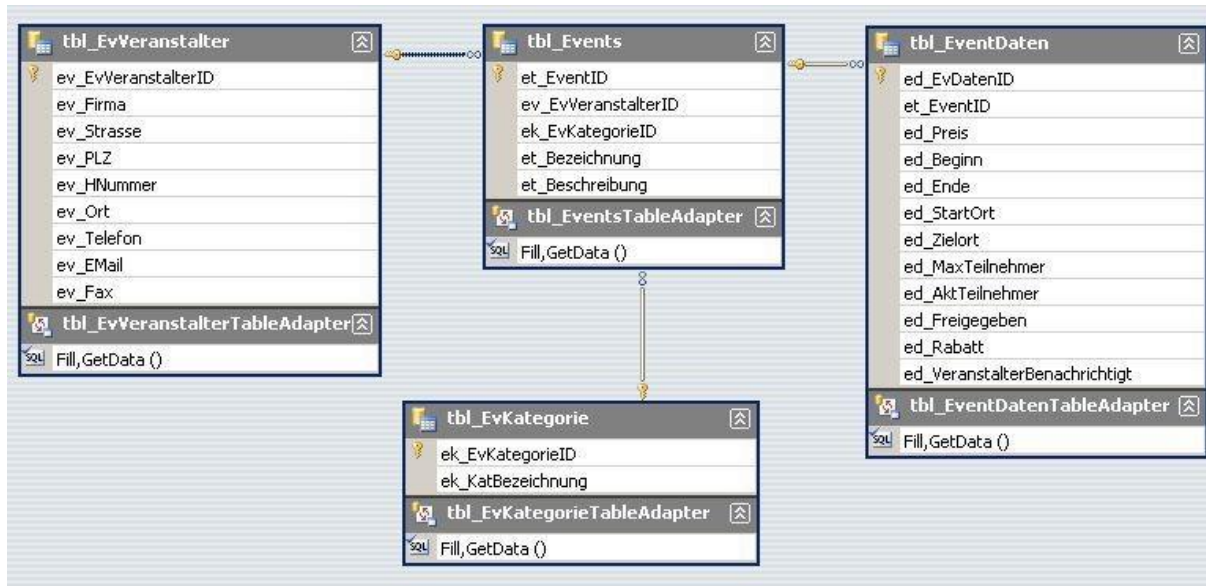
☞ *Speichern Sie das Dataset im Visual Studio und erstellen Sie das Projekt neu.*

Beim Erstellen sollte eine ganze Reihe an **Fehlermeldungen** geworfen werden. Dies liegt an einem Fehler der Implementierung von Typed Datasets unter .NET Core. Anstatt den Microsoft Namensraum des NuGet Pakets zu verwenden, trägt der Dataset Designer den alten System.Data.SqlClient Namensraum bei den Datasetobjekten ein.

☞ *Ändern Sie in der Datei ManageEventsDataset.Designer.cs alle Verweise von System.Data.SqlClient auf Microsoft.Data.SqlClient um!*

☞ *Nutzen Sie dazu „Bearbeiten, Suchen und Ersetzen“!*

Nach dem Generieren der Fremdschlüsseinschränkungen und der Relationen existiert ein Abbild der Datenbank KundenBuchungssystem als XSD-Schema. Anhand dieses Schemas erzeugt das Visual Studio die Dataset Klasse ManageEventsDataset mit allen zugehörigen Table und Relation Objekten sowie den TableAdaptern.



2.1.3 Das Konfigurieren der TableAdapter - Klassen

Neben dem Dataset generiert das Visual Studio für jede Tabelle einen TableAdapter. Dieser verwaltet die Abfragen auf die zugehörige Dataset Tabelle. Seit dem Visual Studio 2008 ist es daher problemlos möglich mehrere (z.B. auch gefilterte) Abfragen für die Dataset Tabellen zu erstellen, solange die zurück gelieferten Daten dem Schema der Tabelle entsprechen.

Genau wie beim DataAdapter sind die wichtigsten Methoden des TableAdapters

- Fill(...) zum Laden der Daten in die Dataset Tabelle sowie
- Update(...) zum Übertragen der Änderungen an die Datenquelle.
- GetData() liefert ein DataTable Objekt zurück.

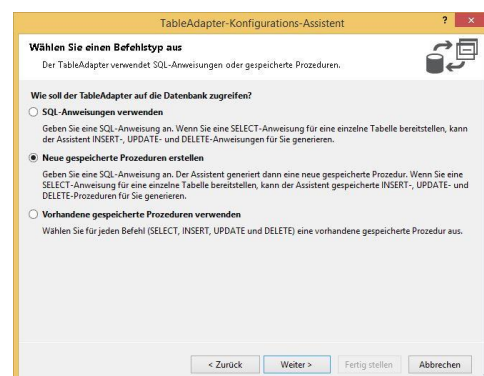
Da die TableAdapter die stored procedures in der Datenbank nutzen sollen müssen diese konfiguriert werden. Die Entwicklungsumgebung des Visual Studios ermöglicht es auf einfache Weise diese TableAdapter mit den zugehörigen Commands erstellen zu lassen.

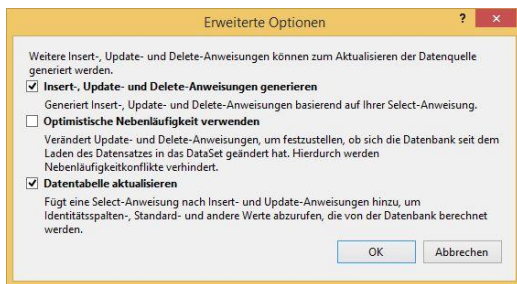
☛ Öffnen Sie das ManageEventsDataset in der Grafiksicht.

☛ Wählen Sie nach einem Rechtsklick auf die Tabelle **tbl_Events** im Kontextmenü den Unterpunkt „Konfigurieren...“. Der Assistent wird gestartet.

☛ Zum Zugriff auf die Datenbank sollen neue gespeicherte Prozeduren verwendet werden. Klicken Sie auf den Button „Zurück“.

☛ Setzen Sie im nächsten Schritt das entsprechende Optionsfeld und klicken Sie auf Weiter.





☞ Wählen Sie über den Abfragegenerator die Tabelle `tbl_Events` und alle Spalten der Tabelle aus. Die `SELECT` Anweisung wird angezeigt.

☞ Klicken Sie auf den Button **Erweiterte Optionen**. Im Dialog deaktivieren Sie die Option **Vollständige Parallelität** verwenden, da Änderungen an den Eventdaten nur durch den Geschäftsführer der FunEvents Herrn Fringsen durchgeführt werden können.

Anhand der ausgewählten Daten können im Anschluss daran die `INSERT`, `UPDATE` und `DELETE` Commands generiert werden.



☞ Benennen Sie die gespeicherten Prozeduren in `spSelectEvents`, `spUpdtEvents`, `spInsEvent` sowie `spDelEvents` um.

☞ Klicken Sie auf **Weiter** und lassen Sie die Command Objekte und die gespeicherten Prozeduren in der Datenbank fertig stellen.

Die Anzeige der Events soll später alphabetisch sortiert erfolgen.

☞ Ändern Sie die gespeicherte Prozedur `spSelectEvents` im Server Explorer so ab, dass die Daten sortiert nach der Bezeichnung zurückgeliefert werden (Aktualisieren im Editor)!

Mit Hilfe des Assistenten sind die benötigten gespeicherten Prozeduren in der Datenbank sowie alle ADO.NET-Objekte zum Zugriff auf die Events-Tabelle der Datenbank generiert worden. Für die weiteren benötigten Tabellen erfolgt die Erstellung der stored procedures und Objekte auf die gleiche Weise.

☞ Generieren und konfigurieren Sie die TableAdapter für die Eventdaten-, Veranstalter- und Kategorietabelle mit Hilfe des Assistenten.

☞ Nutzen Sie dazu die folgende Tabelle:

Konfiguration der TableAdapter:

<code>tbl_EventDatenTableAdapter</code>	<code>tbl_EvVeranstalterTableAdapter</code>	<code>tbl_EvKategorieTableAdapter</code>
<code>spSelectEvDaten</code>	<code>spSelectVeranstalter</code>	<code>spSelectKategorie</code>
<code>spInsEvDaten</code>	<code>spInsVeranstalter</code>	<code>spInsKategorie</code>
<code>spUpdtEvDaten</code>	<code>spUpdtVeranstalter</code>	<code>spUpdtKategorie</code>
<code>spDelEvDaten</code>	<code>spDelVeranstalter</code>	<code>spDelKategorie</code>

Ein Problem beim Update der Daten in der Eventdaten-Tabelle ist bis jetzt allerdings noch nicht behoben. Der Geschäftsführer der FunEvents ist nur für die Verwaltung der Events, nicht jedoch für die Buchungen durch die Kunden zuständig. Buchen die Mitarbeiter der FunEvents Kunden für Events ein, so ändert sich die aktuelle Teilnehmerzahl in der Datenbank. Bei einem Update der Datenbank durch das EMS würde dieser Wert mit den alten Teilnehmerzahlen überschrieben werden. Das Datenfeld `ed_AktTeilnehmer` der Tabelle `tbl_EventDaten` darf deshalb vom EMS nur gelesen werden.

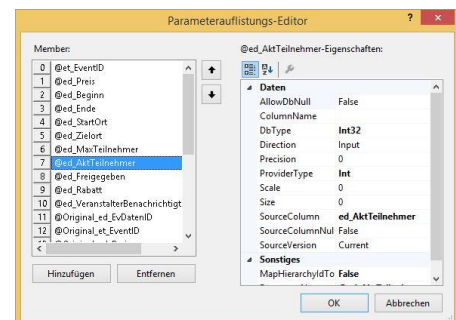
Um dieses Problem zu beheben sind manuelle Änderungen im generierten Quelltext sowie in der gespeicherten Prozedur `spUpdtEvDaten` vorzunehmen.

☞ Öffnen Sie die gespeicherte Prozedur `spUpdtEvDaten` durch einen Doppelklick im Server-explorer im Visual Studio.

☞ Löschen Sie den Parameter `@ed_AktTeilnehmer` sowie die Update Anweisung `ed_AktTeilnehmer = @ed_AktTeilnehmer` im SET Block der stored procedure.

☞ Öffnen Sie den Parameter-Auflistungs-Editor des UpdateCommands zum TableAdapter EventDaten (Eigenschafteneditor, Parameters...).

☞ Entfernen Sie den Parameter `@ed_AktTeilnehmer` aus der Member Liste.



Der Wert für die aktuellen Teilnehmerzahlen kann jetzt nicht mehr fälschlicherweise durch das EMS überschrieben werden.

☞ Betrachten Sie das Klassendiagramm des EventManagementModules.

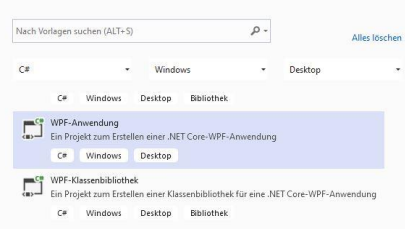
2.2 Das Userinterface ManageEventsApp.exe

Zur Verwaltung der Eventdaten durch den Geschäftsführer der FunEvents GmbH, Herrn Fringsen, soll ein WPF Projekt namens ManageEventsApp.exe dienen. Dieses Userinterface arbeitet später mit der zuvor erstellten Mittelschicht ManageEvents.dll zusammen.

Neues Projekt hinzufügen

Zuletzt verwendete Projektvorlagen

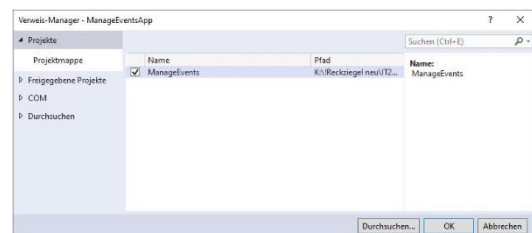
-  Klassenbibliothek C#
-  Konsolen-App C++
-  WPF-App (.NET Framework) C#
-  WPF-Anwendung C#



- ☛ Fügen Sie der Projektmappe eine neue .NET Core WPF-Anwendung namens ManageEventsApp hinzu (Rechtsklick auf die Projektmappe/Hinzufügen/neues Projekt).
- ☛ Wählen Sie als Zielplattform .NET 5.0.
- ☛ Benennen Sie das erstellte Fenster im Projektexplorer in ManageEventsWindow.xaml und im Quelltext um (rechte Maustaste -> Umbenennen) und bestätigen Sie die Änderungen.
- ☛ In der App.xaml ist die StartupUri="ManageEventsWindow.xaml" zu setzen!
- ☛ Ändern Sie die Title Property des Windows auf Event Management System.

Das Userinterface wird im Laufe des Projekts mit der Mittelschichtkomponente ManageEvents.dll zusammenarbeiten. Eine Referenz auf die Library ist dem Projekt deshalb hinzuzufügen.

- ☛ Fügen sie dem Projekt ManageEventsApp einen Verweis auf das Projekt ManageEvents hinzu (Projekt/Verweis hinzufügen).
- ☛ Im Reiter Projekte ist ManageEvents auszuwählen.



Im nächsten Schritt erhält der ManageEventsForm ein Menü, um das Programm zu beenden. Das Menü hat folgenden Aufbau:

Objekt (Name)	Text (Header)
hauptMenue	-
dateiMenuitem	_Datei
beendenMenuitem	_Beenden



- ☛ Ziehen Sie ein Menu von der Toolbox des Visual Studios auf das Fenster. Das Menu erscheint in der Dokumentgliederung. Ändern Sie den Objektnamen in hauptMenue.

Jedes Menü besitzt eine Items Collection mit Menüitems. Ebenso können jedem Menüitem wieder weitere Unterpunkte in einer eigenen Items Collection hinzugefügt werden.

- ☛ Erstellen und bearbeiten Sie die Menüpunkte entsprechend der Tabelle. Das „_“-Zeichen bei der Textproperty erstellt automatisch einen Short Cut für den folgenden Buchstaben.
- ☛ Betrachten Sie den erzeugten XAML Code.
- ☛ Erstellen Sie im Projekt ein Klassendiagramm „EMS_GUI“ analog zum Bibliotheksprojekt.

Arbeiten mit der WPF

Beim Arbeiten mit der Windows Presentation Foundation (WPF) entstehen analog zur Arbeit mit Windows Forms für jede grafische Ressource entsprechende Objekte. Zusätzlich werden die Ressourcen in XAML-Code (ähnlich HTML) protokolliert. Dies gestaltet das Design der GUI zunächst etwas zeitaufwendiger und schwieriger, ermöglicht aber wesentlich größere Gestaltungsspielräume und die Wiederverwendbarkeit von Designs.

Grundsätzlich gilt:

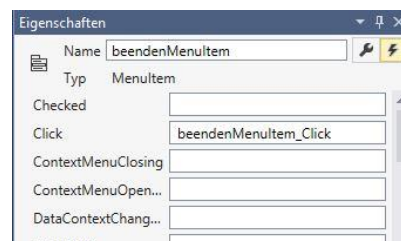
- Die GUI gliedert sich in ein Fenster (Window). Dieses enthält ein Grid. In diesem Grid können weitere Grids, Steuerelemente etc. enthalten sein.
- Bearbeiten Sie WPF GUIs **nur in Ausnahmefällen** im Grafikeditor! Die Bearbeitung im XAML Code ist deutlich schneller und effektiver.
- Navigieren Sie in der GUI über das Fenster „Dokumentgliederung“!

Zum Schließen des ManageEventsForms muss für das Klick-Ereignis des Menüpunktes Beenden eine Handler Methode erstellt werden, die diesen Event abfängt. Die Handler Methode sowie die Verknüpfung der Methode mit dem Event über ein Delegateobjekt kann mit Hilfe des Visual Studios automatisch generiert werden.

☞ *Klicken Sie in der Dokumentgliederung auf den Menüpunkt Beenden.*

☞ *Schalten Sie auf der Eigenschaftenanzeige auf Ereignisse um (der Button mit dem Blitz).*

☞ *Doppelklicken Sie auf das Click- Ereignis.*



Die Handler Methode `private void beendenMenuItem_Click(object sender, RoutedEventArgs e)` wird automatisch generiert. Betrachten Sie dazu auch den erzeugten Quelltext Abschnitt im XAML Code.

Alle Handlermethoden unter WPF in C# besitzen per Konvention zwei Parameter:

- `object sender`: der Absender des Events,
- `RoutedEventArgs e`: ein Objekt, in dem zusätzliche Informationen zum Event gespeichert sind.

In der Handler Methode ist als einzige Aktion die `Close()` Methode des Forms aufzurufen, welche das Fenster schließt und damit die Applikation beendet.

```
// Beendet die Applikation
private void beendenMenuItem_Click(object sender,
                                   RoutedEventArgs e)
{
    Close();
}
```

☞ *Erstellen, starten und testen Sie die Applikation.*

☞ *Vergessen Sie nicht, das Klassendiagramm des ManageEventsWindows zu betrachten.*

3 Lesen der Daten über die Mittelschicht (UC 04.14)

Damit die Daten der Events auf der Benutzeroberfläche angezeigt werden können, sind diese zuerst in das Dataset der Mittelschichtkomponente ManageEvents.dll zu laden. Von dort kann die Applikation über das Dataset auf die einzelnen Events zugreifen.



3.1 Das Erstellen der TableAdapter

Zum Füllen des Datasets mit den Daten der Datenbank nutzt man die bereits erstellten TableAdapter Objekte des ManageEventsDataset in einer neuen public Methode LoadData() der Klasse ManageEventsModule in der Mittelschicht.

Die TableAdapter Objekte müssen zuerst zum ManageEventsModule hinzugefügt und konfiguriert werden.

☛ Binden Sie den Namensraum ManageEvents.ManageEventsDatasetTableAdapters per using Direktive ein.

☛ Deklarieren Sie die vier TableAdapter Objekte taVeranstalter, taKategorie, taEvents und taEvDaten als private Attribute der Klasse ManageEventsModule!

☛ Initialisieren Sie die Objekte im Konstruktor!

Die vier TableAdapter werden anhand des Datasetschemas automatisch generiert.

Die vordefinierten Objekte müssen unter .NET Core nur noch initialisiert werden und stehen danach im Programm zur Verfügung.

```

...
// TableAdapter deklarieren
private tbl_EventsTableAdapter taEvents;
...
/// <summary>
/// Konstruktor mit Initialisierungen
/// </summary>
public ManageEventsModule()
{
    ...
    taEvents = new tbl_EventsTableAdapter();
    ...
}
  
```

Zur besseren Kontrolle der Datenbankverbindungen sollen die TableAdapter Objekte die Connection cnEvents nutzen.

☛ Setzen Sie die Connection Property der vier TableAdapter Objekte am Ende des Konstruktors auf die zuvor erstellte cnEvents (taEvents.Connection = cnEvents;...).

3.2 Das Füllen des Datasets mit Daten

In der Methode LoadData(...) werden in zwei Schritten zuerst ein temporäres Dataset mit den Daten gefüllt und anschließend die Daten in das Dataset dsEvents übertragen.

☞ *Erstellen Sie die öffentliche Methode void LoadData(ManageEventsDataset dsEvents) in der ManageEventsModule Klasse der Mittelschicht.*

Nach dem Erzeugen eines temporären Datasets ist eine Verbindung zur Datenbank herzustellen. Da Arbeiten mit Datenbankzugriff fehleranfällig sind, setzt man alle folgenden Aktionen in einen try-Block.

Mit der Open() Methode des Connection Objektes öffnet man die Verbindung zur Datenbank. Anschließend müssen alle Tabellen des temporären Datasets entsprechend der referentiellen Integrität der Reihe nach mit Daten gefüllt werden. Die Fill(...) Methode übernimmt als Parameter dazu die entsprechende Tabelle des Datasets. Durch den Aufruf von AcceptChanges() werden alle Änderungen im Dataset übernommen.

```
// Laedt die aktuellen Daten aus der Datenbank
// in das Dataset dsEvents.
public void LoadData(ManageEventsDataset dsEvents)
{
    // tempDataset erstellen
    ManageEventsDataset tempEventsDs =
        new ManageEventsDataset();

    try
    {
        // Datenbankverbindung oeffnen
        this.cnEvents.Open();

        // Füllen der Tabellen in der richtigen Reihenfolge
        this.taKategorie.Fill(tempEventsDs.tbl_EvKategorie);
        this.taVeranstalter.Fill(tempEventsDs.tbl_EvVeranstalter);
        ...

        // Aenderungen in Dataset uebernehmen
        tempEventsDs.AcceptChanges();
        ...
    }
```

Damit ist der erste Teil der Methode LoadData() fertiggestellt. Die aktuellsten Daten der Datenbank befinden sich im temporären Dataset.

Im zweiten Schritt kopiert man das temporäre Dataset in das Dataset dsEvents.

Mit der Clear() Methode werden zunächst alle bestehenden Datensätze aus dem Dataset dsEvents gelöscht. Die Methode Merge(...) kopiert anschließend die neuesten Daten aus dem temporären Dataset in dsEvents.

Auch hier müssen die Änderungen mit AcceptChanges() im dsEvents übernommen werden.

Das Abfangen von Fehlern erfolgt später in der Benutzerschnittstelle. Aufgetretene Exceptions werden deshalb mit throw an die aufrufende Methode weitergereicht.

Zuletzt muss in jedem Fall die Verbindung zur Datenbank wieder geschlossen werden. Über den ConnectionState der Connection prüft man zuerst, ob die Verbindung zur Datenbank wirklich hergestellt worden ist, bevor diese mit Close() geschlossen wird.

```
// Alte Datensätze aus dem DataSet entfernen.
dsEvents.Clear();
// Datensätze im Haupt-DataSet zusammenführen.
dsEvents.Merge(tempEventsDs);

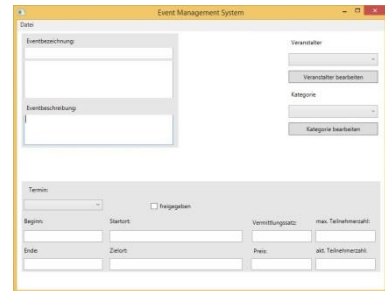
// Aenderungen akzeptieren
...
}

catch(Microsoft.Data.SqlClient.SqlException sqlEx)
{
    // Exceptions weiter reichen
    throw sqlEx;
}
catch(System.Exception ex)
{
    // Exceptions weiter reichen
    throw ex;
}
finally
{
    // Datenbankverbindung schliessen
    if(this.cnEvents.State == ConnectionState.Open)
        this.cnEvents.Close();
}
...}
```

☞ *Kompilieren Sie das Projekt ManageEvents und korrigieren Sie alle eventuell aufgetretenen Fehler.*

4 Das grafische Design der Benutzerschnittstelle

Version 1.0 des EMS beschäftigt sich mit der Umsetzung der Eventverwaltung. Hierzu müssen Events gesucht, Daten abgeglichen bzw. neue Events und Eventdaten angelegt werden können. Alle aus dem Datenbankentwurf vorgegebenen Daten eines Events sind daher auch auf der Benutzerschnittstelle zu implementieren.



4.1 Änderungen am ManageEventsWindow

Das ManageEventsWindow soll in der Applikation auch bei einer Änderung der Größe alle Steuerelemente sinnvoll darstellen. Feste Einstellungen für minimale und maximale Ausmaße des Windows müssen deshalb gesetzt werden.

☞ Ändern Sie dazu folgende Eigenschaften des ManageEventsWindows:

MinimumSize: 700;600

Size: 800;600

4.2 Die Steuerelemente des ManageEventsWindows

Zuerst sollen die Steuerelemente für Veranstalter und Kategorie des ManageEventsWindows im Grafikeditor des Visual Studios erzeugt werden. Zur Anzeige eignet sich ein Grid mit einer Spalte und sechs Zeilen. Die restlichen Steuerelemente zur Anzeige der Daten lassen sich später im XAML Code implementieren.

☞ Ziehen Sie ein Grid von der Toolbox auf das Fenster.

Das Visual Studio erzeugt für jedes grafisch erzeugte Steuerelement automatisch ein Objekt mit den angegebenen Eigenschaften und beschreibt dieses in XAML. Über diese Objekte können später Ereignisse abgefangen oder entsprechende Methoden aufgerufen werden.

☞ Bearbeiten Sie den Objektnamen (`x:Name = "..."`) und die Properties des Grids entsprechend der folgenden Tabelle im Eigenschaftenfenster oder der XAML-Ansicht. Der XAML Code sollte bei Ihren HTML Kenntnissen leicht zu verstehen sein!

Tabelle der Benutzerschnittstelle

Steuerelement	Objektbezeichnung	Properties
Grid (1 * 6)	masterTablesGrid	Height="220" Width="180"
	6 Zeilen <Grid.RowDefinitions> ...	<RowDefinition Height="1*" /> ...

Auf dem Grid fügen Sie nun im XAML Code zwei Mal jeweils ein Label, eine ComboBox sowie einen Button ein.

```
<Label Content="Veranstalter:" Grid.Row="0" Margin="5"/>
<ComboBox x:Name="cboVeranstalter" Grid.Row="1" Margin="5"/>
<Button x:Name="btnVeranstalter" Content="_Veranstalter bearbeiten" Grid.Row="2" Margin="5"/>
```

Grid.Row setzt die Zeile, x:Name steht für den Objektnamen, mit Margin="5" setzen sie die Ränder des Steuerelements auf 5 Pixel Abstand zum enthaltenden Grid.

☞ Bearbeiten Sie die GUI entsprechend der folgenden Tabelle in der XAML-Ansicht. Der XAML Code sollte bei Ihren HTML Kenntnissen leicht zu verstehen sein!

Tabelle der Benutzerschnittstelle

Steuerelement	Objektbezeichnung	Properties
Label		Veranstalter:
Combobox	cboVeranstalter	
Button	btnVeranstalter	_Veranstalter bearbeiten
Label		Kategorie:
Combobox	cboKategorie	
Button	btnKategorie	_Kategorie bearbeiten

☞ Betrachten Sie das Klassendiagramm des ManageEventsWindows im Visual Studio.

4.3 Das Menü des ManageEventsWindows

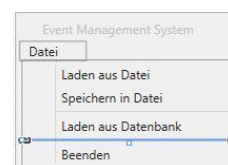
Passend zur Funktionalität des EMS V1.0 soll die Benutzerverwaltung natürlich auch über ein zugehöriges Menü gesteuert werden können.

Der Geschäftsführer der FunEvents GmbH muss dazu die Daten lokal aus einer Datei auf seinem Laptop laden und speichern sowie die Daten auch aus der Datenbank importieren können.

☞ Fügen Sie dem ManageEventsWindow die neuen Menüpunkte hinzu.

☞ Bearbeiten Sie die Menüpunkte gemäß folgender Tabelle.

Objekt (Name)	Text
ladenAusDateiMenuItem	_Laden aus Datei
speichernInDateiMenuItem	_Speichern in Datei
Seperator	
ladenAusDatenbankMenuItem	L_aden aus Datenbank
Seperator	



Das Underscore-Zeichen („_“) sorgt wieder dafür, dass jeder Menüpunkt auch über einen Shortcut von der Tastatur aus aufgerufen werden kann.

☞ Kompilieren und testen Sie das Projekt.

4.4 Die Anzeige der Daten im User Interface

Beim Binden der Daten des Datasets an die Steuerelemente zur Anzeige der enthaltenen Werte müssen die Relationen zwischen den Tabellen beachtet werden. Anzuzeigen sind der Event (Auswahl über die Listbox) und alle zugehörigen Eventdaten (Auswahl über die Combobox mit dem Starttermin) sowie die entsprechenden Veranstalter bzw. Eventkategorien, welche jeweils in Parent-Tabellen des Events gespeichert sind.

Um die miteinander verknüpften Daten aus mehreren Tabellen des Datasets richtig darzustellen, sind mehrere Methoden und Objekte nötig.

4.4.1 Datenquellen hinzufügen

Die Einzeldaten zu den Events können automatisch in Steuerelementen abgelegt werden. Dazu benötigt die Applikation eine Datenquelle, das zuvor bereits erstellte Dataset ManageEventsDataset.

- ☛ Öffnen Sie im Projekt *ManageEventsApp* die XAML Ansicht.
- ☛ Fügen Sie dem Fenster (Window) einen neuen XML Namensraum mit dem Assembly *ManageEvents* hinzu.

```
xmlns:ManageEvents="clr-namespace:ManageEvents;assembly=ManageEvents"
```

Im Assembly *ManageEvents* liegt das Dataset *ManageEventsDataset*.

4.4.2 Der Aufbau der Datenbindung

Im .NET-Framework können alle Properties eines Steuerelements an eine Datenquelle gebunden werden. Dabei unterscheidet man zwischen **simple** und **complex binding**. Beim simple binding bindet man nur eine Eigenschaft an einen Datensatz (z. B. die Text-Property eines Textfeldes an die Bezeichnung eines Events in der Eventstabelle des Datasets). Verwendet man das complex binding, so werden mehrere Datensätze an ein Steuerelement gebunden (z. B. alle Daten mehrerer Datensätze an ein DataGrid oder alle Starttermine der Eventdaten an eine ComboBox).

Beide Arten der Bindung können grafisch, im Quelltext oder im XAML implementiert werden.

In der WPF übernimmt die Datenbindung ein **Binding Objekt**.



Für die Richtung des Datenflusses einer Bindung zwischen Bindungsziel und Bindungsquelle gibt es drei Möglichkeiten, die in der Mode Property gesetzt werden können:

- **TwoWay**: Datenaustausch in beide Richtungen
- **OneWay**: Datenaustausch nur von der Quelle zum Ziel
- **OneWayToSource**: Datenaustausch nur vom Ziel zur Quelle

Beispiele für Simple und Complex Bindings:

Simple Binding: Ein Datenfeld wird an eine Property eines Steuerelements gebunden.

Beim simple binding der Daten an die Textboxen erzeugt man eine Bindung mit der `SetBinding(...)` Methode. Als Parameter übergibt man die Steuerelementproperty, an die gebunden werden soll sowie das Binding Objekt.

An die Textproperty des Textfeldes wird ein Eintrag aus der Tabellenspalte `et_Bezeichnung` der Datasettabelle `tbl_Events` gebunden.

```
Binding myBinding = new Binding();
myBinding.Source = dsEvents.tbl_Events;
myBinding.Path = new PropertyPath("et_Bezeichnung");
txtBeispiel.SetBinding(TextBox.TextProperty, myBinding);
```

Der Path kann aber auch in XAML gesetzt werden, das Übergeordnete Steuerelement (z.B. ein Grid) erhält den `DataContext` dann im Quelltext oder erneut in XAML über ein `CollectionViewSource` Objekt.

```
<TextBox Text="{Binding Path=et_Bezeichnung}" />
...
eventGrid.DataContext = dsEvents.tbl_Events;
...
<Grid x:Name="eventGrid" ...
DataContext="{StaticResource tbl_EventsViewSource}" >
```

Complex Binding: Mehrere Daten einer ganzen Tabellenspalte werden an ein Steuerelement (z.B. eine Combobox) gebunden.

Die Combobox zeigt alle Einträge der Tabellenspalte ed_Beginn der Eventdatentabelle an.

```
cboBeispiel.ItemsSource = dsEvents.tbl_EventDaten;
cboBeispiel.DisplayMemberPath = "ed_Beginn";
```

```
<ComboBox DisplayMemberPath="{Binding Path=et_Bezeichnung}" />
```

In XAML ist für das Grid zusätzlich wieder der DataContext zu setzen.

Als Verbindung zwischen dem C# Code und der XAML GUI dient unter .NET Core die Wrapperklasse `CollectionViewSource`. Sie umhüllt ein `CollectionView`-Objekt, mit dem die Ansicht vom Code aus gesteuert werden kann. Dem XAML Code sind daher als Ressourcen das Dataset und zwei `CollectionViewSource` Objekte zu definieren.

☛ Öffnen Sie im Projekt `ManageEventsApp` die XAML Ansicht.

☛ Fügen Sie unter dem `Window` Tag die folgenden Ressourcen ein!

```
<Window ... >
<Window.Resources>
  <ManageEvents:ManageEventsDataset x:Key="manageEventsDataset"/>
  <CollectionViewSource x:Key="tbl_EventsViewSource"
    Source="{Binding tbl_Events, Source={StaticResource manageEventsDataset}}"/>
  <CollectionViewSource x:Key="tbl_Eventstbl_EventDatenViewSource"
    Source="{Binding tbl_Events.tbl_EvDaten_FK, Source={StaticResource tbl_EventsViewSource}}"/>
</Window.Resources>
```

`tbl_EventsViewSource` bindet an die events Tabelle des Datasets, `tbl_Eventstbl_EventDatenViewSource` bindet nur an die zugehörigen Einträge der Detailtabelle mit den Eventdaten.

4.4.3 Die Anzeige der Events

Im nächsten Schritt sollen die Steuerelemente zur Anzeige auf das `ManageEventsWindow` gelegt werden. Die Databindings werden dabei in XAML Code implementiert. Folgende Grafik zeigt den geplanten Aufbau.

The screenshot shows a window titled "Event Management System". Inside, there's a form with several sections:

- Datei** (File): Contains two text boxes labeled "Eventbezeichnung:" and "Eventbeschreibung:".
- Veranstalter:** (Organizer): A dropdown menu and a button labeled "Veranstalter bearbeiten".
- Kategorie:** (Category): A dropdown menu and a button labeled "Kategorie bearbeiten".
- Termin:** (Date/Time): A dropdown menu and a checkbox labeled "freigegeben".
- Beginn:** (Start): A text box.
- Startort:** (Start Location): A text box.
- Vermittlungssatz:** (Settlement): A text box.
- max. Teilnehmerzahl:** (Maximum number of participants): A text box.
- Ende:** (End): A text box.
- Zielort:** (Destination): A text box.
- Preis:** (Price): A text box.
- akt. Teilnehmerzahl:** (Current number of participants): A text box.

☛ Fügen Sie dem Hauptgrid ein weiteres Grid mit der Bezeichnung *eventGrid* hinzu (*x:Name="eventGrid"*).

Margin legt die Position fest, Background die Hintergrundfarbe.

Als DataContext kann die in den Ressourcen definierte Collection-ViewSource verwendet werden.

Für die fünf Zeilen legt man mit 1* oder 3* die verhältnismäßige Höhe der Zeilen fest.

```
<Grid x:Name="eventGrid" Margin="10,50,450,270"
      Background="#FFE6E6">
  <DataContext="{StaticResource tbl_EventsViewSource}">
  <Grid.RowDefinitions>
    <RowDefinition Height="1*" />
    <RowDefinition Height="1*" />
    <RowDefinition Height="3*" />
    ...
  </Grid.RowDefinitions>
  ...
</Grid>
```

Das eventGrid wird mit zwei Labels, zwei TextBoxes sowie einer ListBox gefüllt.

Die Textfelder sind dabei mit simple binding, die Listbox mit complex binding an die entsprechenden Einträge der Tabelle *tbl_Events* des Datasets gebunden.

Folgender XAML Code zeigt das Binding:

```
<Label Content="Eventbezeichnung:" Grid.Row="0" Margin="2"/>
<TextBox x:Name="et_BezeichnungTextBox" Margin="5" Grid.Row="1"
  Text="{Binding et_Bezeichnung, Mode=TwoWay, NotifyOnValidationError=true, ValidatesOnExceptions=true,
  UpdateSourceTrigger=LostFocus}"/>
<ListBox x:Name="et_BezeichnungListBox" DisplayMemberPath="et_Bezeichnung" ItemsSource="{Binding}"
  Margin="5" Grid.Row="2" IsSynchronizedWithCurrentItem="True"/>
<Label Content="..." />
<TextBox x:Name="et_BeschreibungTextBox" ... />
```

☛ Ergänzen Sie den fehlenden XAML Code.

☛ Setzen Sie die *TextWrapping* Property des Textfeldes *et_Beschreibung* auf "Wrap" (Mehrzeilig).

4.4.4 Die Anzeige der Eventdaten

Analog zu den Events sollen die zugehörigen Eventdaten passend zum ausgewählten Event angezeigt werden. Dazu erstellt man ein weiteres Grid namens *eventDatenGrid* in XAML.

Für eine bessere Auswahl der Eventtermine wird das Feld *ed_Beginn* sowohl als Textbox als auch als Combobox zur Anzeige des Startdatums erzeugt. Die Steuerelemente sind dabei nicht auf die Tabelle, sondern auf die Fremdschlüsselbeziehung zwischen der Events und Eventdaten-Tabelle zu setzen. Das entsprechende *CollectionViewSource* Objekt für den data-Context ist in den Ressourcen bereits definiert.

- ☛ Fügen Sie dem Hauptgrid das neue Grid `eventDatenGrid` hinzu.
- ☛ Bearbeiten Sie die grafische Benutzeroberfläche in XAML mit Hilfe der Abbildung analog zur folgenden Tabelle.

Tabelle: Teil 2 der Benutzerschnittstelle

Steuerelement	Zeile/ Spalte	Objektbezeichnung	Property
Grid		<code>eventDatenGrid</code>	Sechs Zeilen, vier Spalten, grau
Label	0/0		Termin:
ComboBox	1/0	<code>Ed_BeginnComboBox</code>	
CheckBox	1/1	<code>ed_FreigegebenCheckBox</code>	Content : freigegeben IsChecked : gebunden an <code>ed_Freigegeben</code>
Label	2/0		Beginn:
TextBox	3/0	<code>ed_BeginnTextBox</code>	
Label	4/0		Ende:
TextBox	5/0	<code>ed_EndedTextBox</code>	
Label	2/1		Startort:
TextBox	3/1	<code>ed_StartOrtTextBox</code>	
Label	4/1		Zielort:
TextBox	5/1	<code>ed_ZielortTextBox</code>	-
Label	2/2		Vermittlungssatz:
TextBox	3/2	<code>ed_RabattTextBox</code>	-
Label	4/2		Preis:
TextBox	5/2	<code>ed_PreisTextBox</code>	-
Label	2/3		Max. Teilnehmerzahl
TextBox	3/3	<code>ed_MaxTeilnehmerTextBox</code>	-
Label	4/3		Akt. Teilnehmerzahl
TextBox	5/3	<code>ed_AktTeilnehmerTextBox</code>	-

- ☛ Setzen Sie die `IsReadOnly` Eigenschaft der Textbox `ed_AktTeilnehmerTextBox` auf `true`.

Die Steuerelemente zu den Eventdaten sollen am unteren Rand des Forms verankert werden. Bei einer Vergrößerung der Breite müssen sich die Textfelder mit dem Fenster strecken.

- ☛ Setzen Sie `Margin` des Grids auf „10,0,10,44“ und das `VerticalAlignment` auf „Bottom“.
- ☛ Kompilieren und Testen Sie die Applikation.

4.4.5 Das Binden der Datenquelle zur Laufzeit

Bis jetzt sind alle Steuerelemente statisch, das heißt zur Kompilierzeit an die entsprechenden Datenquellen gebunden. Das `ManageEventsDataset` wird aber erst zur Laufzeit (dynamisch) mit Daten aus der Datenbank gefüllt. Die Datenbindung ist also jedes Mal „aufzufrischen“, wenn die Daten erneut in das Dataset eingelesen werden. Dies erledigt ein Datasetattribut und die neu zu erstellende Methode `RefreshBindings()`.

- ☛ Erstellen Sie ein private Attribut `dsEvents` vom Typ `ManageEvents.ManageEventsDataset` in der `ManageEventsWindow` Klasse.
- ☛ Holen Sie sich das mit der GUI – Ressource verbundene Dataset im Konstruktor.

```
public ManageEventsWindow()
{
    ...
    dsEvents = (ManageEvents.ManageEventsDataset)(FindResource("manageEventsDataset"));
}
```

 Implementieren Sie die Methode `RefreshBindings()` in der Klasse `ManageEventsWindow`.

Da die Methode `RefreshBindings()` nur innerhalb der Klasse `ManageEventsWindow` aufgerufen werden soll, setzt man den Zugriffsschutz auf `private`.

Da die Bindings bereits im XAML Code definiert sind, muss bei einem Refresh nur auf den ersten Datensatz der Event tabelle gesprungen werden. Dies kann über die Listbox geschehen.

```
// Erstellt die aktuellen Datenbindings zur Anzeige
private void RefreshBindings()
{
    et_BezeichnungListBox.SelectedIndex = 0;
    ...
}
```

4.5 Kategorie und Eventveranstalter

Kategorie und Eventveranstalter eines Events werden bisher nicht angezeigt, obwohl die Comboboxen bereits modelliert sind.

4.5.1 Die DataBindings für die Comboboxen

Etwas komplexer fallen die Datenbindungen der beiden Comboboxen aus. Als Auswahlliste sollen alle Veranstalter bzw. Kategorien zur Verfügung stehen. Gleichzeitig müssen der Veranstalter bzw. die Kategorie des aktuellen gewählten Events als Wert in den Comboboxen angezeigt sein.

Da auf Grund der Foreign Key Beziehung in der Tabelle `tbl_Events` nur die IDs der Veranstalter und Kategorien gespeichert sind, können die in den Comboboxen angezeigten Werte nicht direkt in das Dataset übertragen werden.

Eine Abhilfe dieses Problems bietet das complex binding mit den Properties einer Combobox. Diese verfügt dazu über drei wichtige Properties:

- **ItemsSource**: die Datenquelle für die anzuzeigenden Werte.
- **DisplayMemberPath**: der anzuzeigende Wert aus der Datenquelle.
- **ValueMemberPath**: der Wert, der in Wirklichkeit hinter der Auswahl steht.

Beispiel `cboVeranstalter`:

Im Beispiel des Veranstalters ist die `ItemSource` das Dataset `dsEvents`. Angezeigt werden soll der Wert des Feldes `tbl_EvVeranstalter.ev_Firma`, die Bezeichnung des Veranstalters (`DisplayMemberPath`). In der Event-Tabelle muss als Wert allerdings die ID des Veranstalters gespeichert werden. Die `ValueMemberPath` Property der Combobox ist deshalb auf `tbl_EvVeranstalter.ev_EvVeranstalterID` zu setzen. Als `SelectedValue` liefert die Combobox in diesem Fall nicht die angezeigte Veranstalterbezeichnung, sondern die ID des Veranstalters zurück.

Bindet man jetzt die `SelectedValue` Property der Combobox über ein Bindingobjekt auf das Datenfeld `ev_EvVeranstalterID` der Events Tabelle, so wird die ID des gewählten Veranstalters in der Events Tabelle eingetragen.



4.5.2 Erstellen der Databindings für die Comboboxen

Für die Verbindung der XAML View mit dem C# Code sind bereits zwei Wrapper Objekte in den Ressourcen der Window Definition erzeugt worden.

Analog zum Dataset erstellt man zum Gebrauch im C# Code zuerst zwei private Attribute vom Typ ICollectionView. Diese werden im Konstruktor initialisiert.

☞ *Definieren Sie zwei private Attribute vom Typ ICollectionView in der ManageEvents-Window Klasse.*

☞ *Initialisieren Sie diese im Konstruktor.*

Die ICollectionView Objekte werden im Konstruktor über die ICollectionViewSource Objekte des XAML Codes initialisiert.

```
private ICollectionView eventView;
private ICollectionView eventDatenView;
...
public ManageEventsWindow()
{
    ...
    eventView = ((CollectionViewSource)this.FindResource("tbl_EventsViewSource")).View;
    eventDatenView = ...
}
```

☞ *Erstellen Sie die DataBindings für die Comboboxen cboVeranstalter und cboKategorie in der bestehenden Methode RefreshBindings() des ManageEventsWindows.*

Wie im obigen Beispiel beschrieben, setzt man in der Methode RefreshBindings() die Properties ItemsSource, DisplayMemberPath und ValueMemberPath auf die entsprechenden Werte.

Über die SetBinding(...) Methode der ComboBox wird daraufhin die SelectedValue Property mit dem Binding Objekt verbunden.

Analog dazu initialisiert man die Bindings der Combobox cboKategorie.

```
private void RefreshBindings()
{
    ...
    // Binden der Elterntabellen
    Binding vBinding = new Binding();
    vBinding.Source = eventView;
    vBinding.Path = new PropertyPath("ev_EvVeranstalterID");

    cboVeranstalter.ItemsSource = dsEvents.tbl_EvVeranstalter;
    cboVeranstalter.DisplayMemberPath = "ev_Firma";
    cboVeranstalter.SelectedValuePath = "ev_EvVeranstalterID";
    cboVeranstalter.SetBinding(ComboBox.SelectedValueProperty, vBinding);

    cboVeranstalter.IsSynchronizedWithCurrentItem = true;
    ...
}
```

☞ *Implementieren Sie die Bindings für cboVeranstalter und cboKategorie.*

☞ *Kompilieren Sie die Anzeige und Auswahl von Veranstalter und Kategorie.*

5 Laden der Events aus der Datenbank (UC 04.14)

Kapitel 5 zum EMS 1.0 beschäftigt sich mit den Use Cases 04.2 und 04.14, dem Laden der Eventdaten aus der Datenbank in ein Dataset der Applikation sowie der Anzeige der Daten auf der Benutzeroberfläche. Die Funktionalität zum Laden der Daten in die Mittelschichtkomponente ist bereits in den vorangegangenen Kapiteln erstellt worden.

5.1 Die Methode ladenAusDatenbankMenuItem_Click(...)

Bevor die Eventdaten in das Userinterface geladen werden können, benötigt man einen lokalen Puffer zum Zwischenspeichern in der Applikation. Diesen Puffer stellt das Dataset Objekt dsEvents vom Typ ManageEvents.ManageEventsDataset dar.

Das Dataset Objekt ist bereits als Attribut deklariert und steht im ManageEventsWindow zur Verwendung bereit.

Klickt der User auf den Menüpunkt **Laden aus Datenbank**, so sollen die Eventdaten über die Mittelschichtkomponente ManageEvents.dll aus der Datenbank in das vorher deklarierte Dataset dsEvents übertragen werden. Das Klick Ereignis ist deshalb abzufangen und mit einer Handler Methode zu verknüpfen.

- ☞ Markieren Sie den Menüpunkt ladenAusDatenbankMenuItem in der Entwurfsansicht.
- ☞ Schalten Sie auf der Eigenschaftenanzeige auf Ereignisse um (der Button mit dem Blitz).
- ☞ Doppelklicken Sie auf das Click- Ereignis.

Das Visual Studio hat die Handler Methode void ladenAusDatenbankMenuItem_Click(object sender, RoutingEventArgs e) automatisch generiert und mit einem Delegateobjekt an das Klick Ereignis des Menüpunktes gebunden.

- ☞ Bearbeiten Sie die Methode ladenAusDatenbankMenuItem_Click(...).

Zuerst benötigt man ein Objekt vom Typ ManageEventsModule, um darüber auf die Datenbank zugreifen zu können.

```
private void ladenAusDatenbankMenuItem_Click ( ... )  
{  
    try  
    {  
        // Objekt für Zugriff auf die Mittelschicht erzeugen  
        ManageEvents.ManageEventsModule m =  
            new ManageEvents.ManageEventsModule( );  
        ...  
    }  
}
```

Da alle Fehler im Userinterface abgefangen und als MessageBox an den Benutzer weitergeleitet werden sollen, setzt man die Anweisungen in einen try-Block.

Beim direkten Laden der Daten aus der Datenbank ist es möglich, dass lokale Daten und Änderungen auf dem Laptop des Geschäftsführers mit älteren Daten aus der Datenbank überschrieben werden. Das Überschreiben sollte daher nicht ohne Warnung an den User möglich sein. Dies bewerkstelligt man mit Hilfe einer MessageBox mit den Auswahlbuttons Ja bzw. Nein.



Nach dem Setzen der MessageBoxButton Property auf YesNo fängt man die Rückgabe der Show(...) Methode ab und überprüft das zurück gegebene MessageBoxResult.

Hat der Benutzer mit Ja bestätigt, so setzt am den Cursor während der Zeit des Datenladens zum besseren Verständnis auf „Warten“.

Anschließend initialisiert man das Dataset mit der bereits erstellten Methode LoadData(...) der Mittelschicht. Ein Aufruf der Methode RefreshBindings() aktualisiert die GUI.

Sollte eine Sql-Ausnahme geworfen werden, so fängt man diese in einem catch-Block ab und gibt den Fehlertext über die ToString() Methode des SQLException Objektes als MessageBox aus.

Genauso werden allgemeine Ausnahmen im zweiten catch-Block abgefangen und behandelt. Als Überschrift der MessageBox wählt man hier jedoch „Fehler“ anstelle von „SQL-Fehler“.

Zuletzt muss im finally-Block der Standardcursor wieder hergestellt werden. Der finally-Block wird unabhängig von einer aufgetretenen Ausnahme immer ausgeführt.

```
...
if(MessageBox.Show("Achtung. Die Änderungen ...",
    "Warnung. Möglicher Datenverlust...",
    MessageBoxButton.YesNo,
    MessageBoxImage.Warning)
    == MessageBoxResult.Yes)
{
    // Wait Cursor setzen
    this.Cursor = Cursors.Wait;


    // Daten aus Datenbank laden und akzeptieren
    m.LoadData(dsEvents);
}

}

catch(System.Data.SqlClient.SqlException ex)
{
    MessageBox.Show(ex.ToString(),
        "SQL Fehler",
        MessageBoxButton.OK,
        MessageBoxImage.Error);
}

catch(Exception ex)
{
    MessageBox.Show(ex.ToString(), "Fehler", MessageBoxButton.OK,
        MessageBoxImage.Error);
}

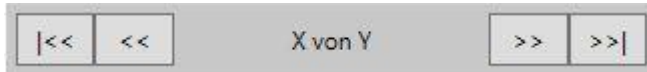
finally
{
    this.Cursor = Cursors.Arrow;
}
}
```

 *Kompilieren und testen Sie das Projekt.*

6 Eine Erweiterung der Anzeige

6.1 Die Anzeige einer Navigationsleiste

Etwas aufwendiger gestaltet sich die Anzeige einer Navigationsleiste. Die Daten der Events werden bereits über die Events und Eventdaten-Tabelle an die Steuerelemente der Benutzerschnittstelle gebunden. Die Navigation zwischen den Datensätzen erfolgt aber nur über eine Listbox bzw. eine Combobox. Mehr Komfort bietet eine Navigationsleiste, welche unter WPF,



anders als unter Windows Forms, nicht automatisch zur Verfügung steht. Aus Gründen der Wiederverwendbarkeit soll

die Navigationsleiste als Steuerelementbibliothek programmiert werden.

Um innerhalb gebundener Daten navigieren zu können oder diese manipulieren zu können wird unter WPF eine Standardview für gebundene Daten, die sogenannte *CollectionView* genutzt. Die *CollectionView* Objekte sind bereits für das Databinding im C# Code definiert.

Wichtige Properties/ Methoden eines *CollectionView* Objekts:

- **Count** liefert die Anzahl der gebundenen Elemente (z. B. die Anzahl der Events in der Events-Tabelle).
- **CurrentPosition** gibt die aktuelle Position des gebundenen Elements zurück (z.B. den Index des gerade angezeigten Events in der Events-Tabelle). Mit Hilfe der *MoveCurrentTo...* Methode kann so auf jeden beliebigen Event in der Events-Tabelle gesprungen werden.
- **MoveCurrentToFirst() / MoveCurrentToLast() / MoveCurrentToNext() / MoveCurrentToPrevious()**, **IsCurrentBeforeFirst/ IsCurrentAfterLast** setzen die aktuelle Position bzw. fragen die Endpunkte ab.
- **CurrentItem** liefert ein *DataRowView* Objekt der gerade ausgewählten Tabellenzeile zurück.

Ändert der User die Eventauswahl, so ändert sich auch die *CurrentPosition* Property der *CollectionView* zur Tabelle *tbl_Events*. Gleichzeitig wird das Event *CurrentChanged* der *CollectionView* ausgeworfen. Dieser Event muss bei der Navigationsleiste in einer Handler Methode abfangen werden.

Die Navigationsleiste wird als Benutzersteuerelement in einem neuen Projekt angelegt.

☛ Fügen Sie der Projektmappe *EMS_2024 das neue WPF Benutzersteuerungselementbibliothek Projekt* *NavigationBar* hinzu. *.NET Core, Zielplattform 5.0.*



☛ Benennen Sie die Steuerelementklasse im Projektmappenexplorer und im erzeugten C# Code in *NavBar* um.

Die Benutzeroberfläche der Navigationsleiste wird in WPF definiert.

☞ Bauen Sie das Benutzersteuerelement entsprechend folgender Tabelle auf!

☞ Erstellen Sie die Handler Methoden für die Klick Ereignisse auf die Buttons.

Tabelle: Navigationsleiste

Steuerelement	Zeile/ Spalte	Objektbezeichnung	Property
Grid		MasterGrid	eine Zeile, fünf Spalten, grau
Button	0/0	First	Content: <
Button	0/1	Previous	Content: <<
Label	0/2	Number	Content: x von y
Button	0/3	Next	Content: >>
Button	0/4	Last	Content: >

Zur Navigation durch beliebige Datensätze verwendet man nun ein `CollectionView` Objekt. Die Anzahl der gebundenen Elemente speichert man zur späteren Anzeige in einem private Attribut `count`.

☞ Definieren Sie die beiden private Attribute `sourceView` und `count` in der `NavBar` Klasse!

Über das `CollectionView` Objekt `sourceView` lässt sich die aktuelle Position der Auswahl steuern. Dies wird in den vier Navigationsmethoden genutzt.

```
...
private CollectionView sourceView;
private int count = 0;
...
private void First_Click(object sender, RoutedEventArgs e)
{
    sourceView.MoveCurrentToFirst( );
}
```

☞ Implementieren Sie die vier Handler Methoden zur Navigation mit der `sourceView`!

☞ Achten Sie auch darauf, dass nicht über den Anfang oder das Ende der Tabelle hinaus navigiert wird (siehe vorherige Seite).

Zum Setzen der `CollectionView` von außen benötigt man noch eine öffentliche Property.

☞ Implementieren Sie die public Property `NavSourceView` mit `get` und `set` Teil.

Neben der `sourceView` ist im `set`-Teil auch das `count` Attribut zu setzen. Anschließend navigiert man auf den ersten Datensatz und setzt die Anzeige des Labels.

Immer wenn die aktuelle Auswahl geändert wird, muss die Anzeige im Label aktualisiert werden. Dies erledigt man mit dem `CurrentChanged` Event. Tipp: Benutzen Sie die TAB Taste!

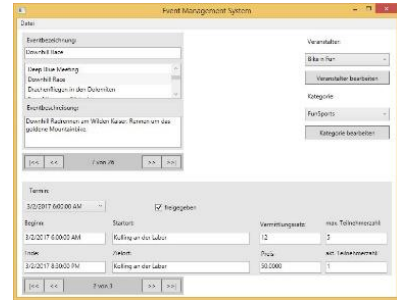
```
public CollectionView NavSourceView
{
    get
    ...
    set
    {
        sourceView = value;
        // Anzahl Elemente
        count = sourceView.Count;
        sourceView.MoveCurrentToFirst( );
        Number.Content = sourceView.CurrentPosition + 1 + " von " + count;
        sourceView.CurrentChanged += SourceView_CurrentChanged;
    }
}
```

☞ Implementieren Sie die Handler Methode `CurrentChanged(...)` der `sourceView` mit der Aktualisierung des Labels (TAB Taste nutzen)!

☞ Erstellen Sie das Projekt neu zum Generieren der *.dll!

Dem Einfügen der Navigationsleiste in der Benutzeroberfläche der Applikation steht nun nichts mehr im Weg.

- ☞ *Fügen Sie dem Projekt `ManageEventsApp` unter `Abhängigkeiten` einen Verweis auf das Projekt `NavigationBar` hinzu!*
- Die `NavBar` sollte nach dem Neuerstellen der Projektmappe in der Toolbox sichtbar sein.*
- ☞ *Ziehen Sie zwei `NavBar` Objekte auf die GUI und benennen Sie diese in `eventNavBar` und `eventDatenNavBar` um!*



Die Verbindung der `CollectionView`s aus dem C# Code des `ManageEventsWindows` mit den Navigationsleisten erfolgt am Ende der bereits implementierten `RefreshBindings()` Methode.

```
private void RefreshBindings(...)
{
    ...
    eventNavBar.NavSourceView = eventView;
    eventDatenView = ((CollectionViewSource)this.FindResource("tbl_Eventstbl_EventDatenViewSource")).View;
    eventDatenNavBar.NavSourceView = (CollectionView)eventDatenView;
    eventView.CurrentChanged += ...
}
```

Über die `CollectionView` Klasse setzt man die `CollectionView`s zur Eventtabelle (`eventView`) und den zugehörigen Einträgen in der Eventdatentabelle (`eventDatenView`). Für die Untereinträge wählt man die `FindResource(...)` Methode des Fensters.

Weil sich die Unterauswahl bei jedem Wechsel eines Events erneut ändert, muss die `eventDatenView` und die `NavBar` immer wieder neu aktualisiert werden. Dies erledigt man am einfachsten mit dem `CurrentChanged` Event der `eventView`.

```
private void EventView_CurrentChanged(...)
{
    eventDatenView = ((CollectionViewSource)FindResource(...))...
    if(eventDatenView != null)
    {
        eventDatenNavBar.NavSourceView = ...
    }
}
```

- ☞ *Programmieren Sie die Änderungen in der Methode `RefreshBindings()`.*
- ☞ *Implementieren Sie auch die `CurrentChanged` Handler Methode der `eventView`, in welcher die `eventDatenView` neu initialisiert und die `NavSourceView` der Navigationsleiste gesetzt werden! Tipp: Nutzen Sie die TAB Taste!*
- ☞ *Kompilieren und testen Sie die Applikation.*

6.2 Die Formatierung des Eventpreises in Euro

Eine Unzulänglichkeit bei der Anzeige der Eventdaten muss noch behoben werden: Der Preis eines Events wird als `Decimal` dargestellt und sollte in die Währung Euro umgewandelt werden. Die Umwandlung muss dabei in zwei Richtungen erfolgen:

- von `Decimal` zu EUR, wenn der Preis angezeigt wird;
- von EUR nach `Decimal`, wenn der Preis bearbeitet worden ist und wieder im Dataset gespeichert werden soll.

C# bietet dazu die Möglichkeit eine Converter Klasse an die Datenbindung zu hängen und zwei Methoden zur Konvertierung zu erstellen.



Werden Daten aus einer Datenquelle an ein gebundenes Steuerelement übertragen, so wird die `Convert()` Methode aufgerufen. Sind Daten des Steuerelements wieder in der Datenquelle zu speichern, so löst `ConvertBack()` aus. Mit Hilfe des übergebenen Parameterobjekts vom Typ `object` kann der Programmierer entsprechende Umwandlungen der Daten vornehmen.

Zuerst ist aber eine Converter Klasse mit dem Interface `IValueConverter` zu implementieren.

☞ *Fügen Sie dem Projekt `ManageEventsApp` eine neue C# Klasse namens `MoneyConverter` hinzu und ändern Sie die Namensraumdeklaration in `CustomConverter` um!*

☞ *Binden Sie `System.Windows`, `System.Windows.Data` und `System.Globalization` ein.*

☞ *Leiten Sie die Klasse von `IValueConverter` ab und lassen Sie sich die Konvertierungsmethoden durch das Visual Studio generieren!*

Das Runtime Attribute `ValueConversion` beschreibt dem .NET Framework beim Laden der Klasse die umzuwandelnden Datentypen.

Im value parameter der `Convert()` Methode wird der Wert als `object` übergeben. Nach einem Cast in ein decimal formatiert die `ToString(...)` Methode einen Text daraus. Das Formatierungszeichen „c“ (currency) sorgt für eine Darstellung des Wertes im Währungsformat.

`ConvertBack()` nutzt die `TryParse()` Methode zur Rückkonvertierung des in Euro formatierten Währungsstrings in einen decimal.

Bei einem Fehler wird der alte Wert im Textfeld beibehalten (`DependencyProperty.UnsetValue`).

```
[ValueConversion(typeof(Decimal), typeof(String))]
public class MoneyConverter : IValueConverter
{
    public object Convert(object value, Type targetType,
        object parameter, CultureInfo culture)
    {
        decimal money = (decimal)value;
        return money.ToString("c");
    }

    public object ConvertBack(object value, Type targetType,
        object parameter, CultureInfo culture)
    {
        string strValue = (string)value;
        decimal result;
        if (decimal.TryParse(strValue, NumberStyles.Currency,
            System.Globalization.CultureInfo.CurrentCulture, out result))
        {
            return result;
        }
        return DependencyProperty.UnsetValue;
    }
}
```

☞ *Implementieren und kompilieren Sie die `MoneyConverter` Klasse!*

Für den Einsatz ist die `MoneyConverter` Klasse nur noch als static Ressource im XAML Code zu definieren und mit dem Binding der Textbox `ed_PreisTextBox` zu verknüpfen.

☞ *Erstellen Sie die statische Ressource in der XAML Datei des `ManageEventsWindows`!*

Im Window Tag des XAML Files ist der Namensraum `CustomConverter` als Attribute anzugeben.

Im Resources Knoten registriert man anschließend die `MoneyConverter` Klasse.

```
<Window
...
xmlns:CustomConverter="clr-namespace:CustomConverter
...>
<Window.Resources>
...
<CustomConverter:MoneyConverter x:Key="moneyConverter"/>
</Window.Resources>
...
```

☞ *Verknüpfen sie das Binding der `ed_PreisTextBox` in der XAML Datei des `ManageEventsWindows` mit der `MoneyConverter` Klasse!*

```
<TextBox x:Name="ed_PreisTextBox" Text="{Binding ed_Preis, Converter={StaticResource moneyConverter}, ... }" .../>
```

Die Text Property verwendet beim Binding jetzt die angegebene Converter Klasse.

☞ *Kompilieren und testen Sie das Projekt. Die Preisanzeige sollte nun in der auf dem Testrechner installierten Währung erfolgen.*

7 Die lokale Verwaltung der Eventdaten (UC 04.13)

Der Geschäftsführer der FunEvents GmbH, Herr Fringsen, nutzt das EMS laut der Analyse der Geschäftsprozesse auf seinen Besuchen bei den Eventveranstaltern. Aus diesem Grund hat er keinen direkten Zugriff auf die Datenbank um die neuen Events und Eventdaten zu aktualisieren. Die Daten müssen daher lokal auf dem Laptop zwischengespeichert werden, bis der Geschäftsführer am Abend im Geschäft einen Abgleich mit der Datenbank vornehmen kann.

Daten eines Datasets lassen sich im .NET-Framework auf einfache Art und Weise in eine Datei schreiben oder wieder aus einer Datei auslesen. Die DataSet Klasse implementiert dazu die Methoden ReadXml(...) und WriteXml(...), welche die Daten im XML-Format lesen bzw. schreiben können. Als Parameter erwarten beide Methoden die Dateibezeichnung sowie einen XmlReadMode bzw. XmlWriteMode Parameter, welcher angibt, wie die Daten in das DataSet einzulesen oder vom DataSet aus abzuspeichern sind. Folgende Tabelle gibt einen Überblick über die gängigsten Modi:

XmlReadMode	XmlWriteMode
IgnoreSchema Ignoriert Inlineschemas und liest Daten in das vorhandene DataSet-Schema.	IgnoreSchema Schreibt den aktuellen Inhalt des DataSet als XML-Daten ohne ein XSD-Schema.
ReadSchema Liest jedes Inlineschema und lädt die Daten. Wenn das DataSet bereits ein Schema enthält, können diesem neue Tabellen hinzugefügt werden.	WriteSchema Schreibt den aktuellen Inhalt des DataSet als XML-Daten mit der relationalen Struktur als XSD-Inlineschema.
DiffGram: Liest ein DiffGram, wobei die Änderungen aus dem DiffGram auf das DataSet angewendet werden. Dabei werden sowohl aktueller als auch geänderter Wert eines Datensatzes gespeichert.	DiffGram Schreibt das gesamte DataSet als DiffGram, einschließlich der ursprünglichen und aktuellen Werte.
InferSchema Ignoriert alle Inlineschemas und leitet das Schema von den Daten her ab.	

Im EMS sollen gezielt die Änderungen des Geschäftsführers am DataSet gespeichert werden. Als Read- und Writemode kommt also ein DiffGram zum Einsatz.

7.1 Das lokale Speichern des Datasets

Für das lokale Speichern des Datasets ist bereits der Menüpunkt *Speichern in Datei* erstellt worden. Klickt der Benutzer auf diesen Menüpunkt, so soll das DataSet als DiffGram abgespeichert werden.

☛ Generieren Sie die Handler Methode *speichernInDateiMenuItem_Click(...)*, welche das Klick-Ereignis auf den Menüpunkt abfängt.

In der Handler Methode ist in einem try-Block die WriteXml(...) Methode des Datasets `manageEventsDataset` aufzurufen. Als Parameter gibt man den Namen der XML Datei sowie den `XmlWriteMode` an.

```
// Speichert das DataSet offline in das lokale XML-File.
private void speichernInDateiMenuItem_Click (...)
{
    try
    {
        dsEvents.WriteXml("Buchungen.xml", XmlWriteMode.DiffGram);
    }
    ...
}
```

Eventuell aufgetretene Exceptions werden im catch-Block gefangen und als MessageBox an den User gemeldet.

```
catch(Exception ex)
{
    MessageBox.Show(...);
}
```

☞ *Kompilieren und testen Sie das Programm.*

☞ *Betrachten Sie das generierte XML-File im bin\Debug-Ordner der Applikation.*

Schließt der User den Dialog, so sollten automatisch alle Daten des Datasets gespeichert werden, um ein versehentliches Vergessen der Sicherung zu verhindern.

☞ *Fangen Sie den WindowClosing Event des ManageEventsWindows ab und generieren Sie eine Handler Methode.*

In der Handler Methode ist nur noch ein Klick auf das Menüitem speichernInDateiMenuItem zu simulieren. Dies erledigt die RaiseEvent() Methode, der ein neuer RoutedEventArgs Parameter für das Klick Ereignis übergeben wird.

```
// Der Dialog wird geschlossen. Aenderungen werden gespeichert.
private void WindowClosing(object sender, CancelEventArgs e)
{
    // lokales Speichern der Daten
    speichernInDateiMenuItem.RaiseEvent(
        new RoutedEventArgs(MenuItem.ClickEvent));
}
```

7.2 Das lokale Laden des Datasets

Analog zum Speichern der Daten im XML-File müssen die Daten auch wieder in das Dataset geladen werden können.

☞ *Implementieren Sie die Handler Methode ladenAusDateiMenuItem_Click(...) für das Klick-Ereignis auf den Menüpunkt Laden aus Datei.*

In der Methode muss das Dataset zunächst mit Clear() zurückgesetzt und im Anschluss daran neu initialisiert werden.

Eingeschlossen in einen try-Block liest man die Daten daraufhin mit der ReadXml(...) Methode ein. Als Parameter sind wieder der Dateiname sowie der XmlReadMode DiffGram anzugeben.

RefreshBindings() sorgt zum Schluss für die korrekte Anzeige der Daten in den Steuerelementen.

```
// Laedt die Daten offline aus dem lokal gespeicherten XML-File.
private void ladenAusDateiMenuItem_Click(...)
{
    try
    {
        dsEvents.Clear();
        dsEvents.ReadXml(...);

        RefreshBindings();
    }
    catch(Exception ex)
    {
        MessageBox.Show(...);
    }
}
```

Das Exception Handling erfolgt analog zum Speichervorgang.

Weil das EMS die Eventdaten auch ohne Verbindung zur Datenbank darstellen soll, wenn der Geschäftsführer mit dem Laptop auf Geschäftsreise ist, sind die anzuzeigenden Daten automatisch beim Start der Applikation aus dem XML-File zu laden.

Diese Funktionalität lässt sich am einfachsten im Konstruktor des ManageEventsWindows implementieren.

☞ *Überarbeiten Sie den Konstruktor des ManageEventsWindows so, dass die Daten beim Start lokal aus dem XML-File geladen werden.*

Im Konstruktor ist zum lokalen Laden der Daten die zuvor erstellte Methode ladenAusDateiMenuItem_Click(...) über RaiseEvent() aufzurufen.

```
// Konstruktor mit Initialisierungen
public ManageEventsWindow()
{
    InitializeComponent();
    ...
    // lokale XML Daten laden
    ladenAusDateiMenuItem.RaiseEvent(...);
}
```

☞ *Kompilieren und testen Sie die Version 1.0 des EMS.*

☞ *Fügen Sie dem Visio Projekt die neuen Arbeitsblätter „Klassendiagramm zur GUI“ und „Klassendiagramm zur Mittelschicht“ hinzu.*

☞ *Kopieren Sie dazu die Klassendiagramme aus dem Visual Studio in das Visio Projekt.*