

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 2 ENTWURF UND IMPLEMENTIERUNG DES EMS V2.0

Inhalt

1	VORÜBERLEGUNGEN	1
2	DATENBANKUPDATE ÜBER DIE MITTELSCHICHT (UC 04.14)	2
3	DIE FREIGABE DER EVENTS ZUR BUCHUNG (UC 04.12)	4
3.1	Neue Datenbindungen in XAML	4
3.2	Das Abfangen einer Freigabe	4
4	DER ABGLEICH MIT DER DATENBANK (UC 04.14)	6
4.1	Eine Erweiterung des Menüs	6
4.2	Die Synchronisation mit der Datenbank	6
5	NEUE EVENTS UND EVENTDATEN (UCS 04.8, 04.9)	8
5.1	Neue Steuerelemente des ManageEventsWindows	8
5.2	Das Hinzufügen neuer Events (UC 04.9)	9
5.3	Das Hinzufügen neuer Eventdaten (UC 04.8)	10
5.4	Das Abbrechen der Bearbeitung	11
5.5	Die Übernahme der Änderungen	12
6	NEUE VERANSTALTER ERSTELLEN (UC 04.11.1)	14
6.1	Das Erstellen des ManageVeranstalterWindows	14
6.1.1	Die Steuerelemente des ManageVeranstalterWindows	14
6.1.2	Die Anzeige der Datensätze	15
6.1.3	Änderungen im ManageEventsWindow	16
6.2	Das Bearbeiten der Veranstalter	17
6.2.1	Die Behandlung der Fehlermeldungen	17
6.2.2	Das Anlegen neuer Veranstalter	18
6.2.3	Das Abbrechen der Bearbeitung	19
6.2.4	Die Übernahme der Daten	19
6.2.5	Das Schließen des Fensters	20
7	NEUE KATEGORIEN ERSTELLEN (UC 04.10.1)	21
7.1	Das Erstellen des ManageKategorieWindows	21
7.1.1	Die Steuerelemente des ManageKategorieWindows	21
7.1.2	Die Anzeige der Datensätze	22
7.1.3	Letzte Änderungen im ManageEventsWindow	23
7.2	Das Bearbeiten der Kategorien	23
7.2.1	Die Behandlung der Fehlermeldungen	23
7.2.2	Das Anlegen neuer Kategorien	23
7.2.3	Das Abbrechen der Bearbeitung	24
7.2.4	Die Übernahme der Daten	24
7.2.5	Das Schließen des Fensters	24

1 Vorüberlegungen

Im zweiten Durchlauf des Spiralmodells soll dem Event Management System EMS die Möglichkeit zum Ändern und Hinzufügen von Events und Eventdaten hinzugefügt werden. Ebenso sind die Verwaltung der Daten zu den Eventveranstaltern sowie den Eventkategorien als Ziel gesetzt. Design und Realisierung des EMS V 2.0 erfolgen wieder mit Hilfe der UML sowie der Microsoft Programmiersprache C#.

Für den zweiten Durchgang der Entwicklung im Spiralmodell werden als Ziel die Eventverwaltung sowie der Abgleich der Eventdaten mit der Datenbank gesetzt. Ebenso sollen Veranstalter und Kategorien bearbeitet und hinzugefügt werden können. Dazu zählen die Use Cases

- UC 04.3 Preise für einen Event bearbeiten,
- UC 04.4 Termin bearbeiten,
- UC 04.5 Veranstaltungsort bearbeiten,
- UC 04.6 Vermittlungssatz bearbeiten,
- UC 04.7 maximale Teilnehmerzahlen für Event bearbeiten,
- UC 04.8 Neue Eventdaten erfassen,
- UC 04.9 neue Events im Buchungssystem aufnehmen,
- UC 04.10 Event in Kategorie einteilen,
- UC 04.10.1 neue Kategorie erstellen,
- UC 04.11 Eventveranstalter zuordnen,
- UC 04.11.1 neuen Eventveranstalter erfassen,
- UC 04.12 Event zur Buchung freigeben sowie
- UC 04.14 Events mit Datenbank abgleichen.

2 Datenbankupdate über die Mittelschicht (UC 04.14)

Auf Geschäftsreisen vereinbart der Geschäftsführer der FunEvents GmbH, Herr Fringsen, neue Events und Eventdaten mit den Veranstaltern. Diese Daten speichert er zunächst lokal auf seinem Laptop. Im Geschäft sind diese Daten dann an die Datenbank zu übertragen.

Zum Abgleich von lokalen Änderungen an den Events und Eventdaten mit der Datenbank muss die Mittelschicht `ManageEvents.dll` um eine Funktionalität erweitert werden.

In der Version 1.0 des EMS sind bereits alle benötigten `TableAdapter` und gespeicherten Prozeduren zum Abgleich der Datenbank mit den Daten des `DataSet` `manageEventsDataSet` implementiert worden. Zum Update der Daten müssen in einer **neuen Methode** der Mittelschichtklasse `ManageEventsModule` nur noch die entsprechenden Methoden der `TableAdapter`-Objekte aufgerufen werden.

Zum Vermeiden von Datenkonflikten soll der Abgleich mit der Datenbank innerhalb einer Transaktion ablaufen. Transaktionen erfolgen allgemein nach dem **ACID** Prinzip:

Atomicity	Alle Aktionen einer Transaktion erfolgen als Einheit und gelingen oder schlagen gemeinsam fehl.
Consistency	Daten werden konsistent von einem Ausgangszustand in einen Endzustand überführt. Es gibt keine sichtbaren Zwischenzustände.
Isolation	Transaktionen laufen für sich gesehen isoliert ab. Jede Transaktion im System arbeitet, als wäre sie die einzig laufende.
Durability	Beim Erfolg einer Transaktion ist die Integrität der Daten gewährleistet, selbst wenn nach dem Commit das System abstürzen sollte.

Unter ADO .NET können lokale Transaktionen über ein Transaktionsobjekt gesteuert werden, welches über eine bestehende Datenbankverbindung mit der `BeginTransaction(...)` initialisiert wird. Die `Transaction Property` aller an der Transaktion beteiligten `Command` Objekte ist daraufhin auf das erzeugte `Transaction` Objekt zu setzen. Wichtige Methoden des Transaktionsobjektes sind:

- **Commit()**: Die Transaktion ist erfolgreich verlaufen und wird abgeschlossen.
- **Rollback()**: Ein Fehler ist aufgetreten, die Transaktion wird abgebrochen. Alle bisherigen Änderungen werden zurückgenommen.

Für die Steuerung des `Isolationlevels` sind folgende Parameter für die Methode `BeginTransaction(...)` vorgegeben:

<code>ReadUncommitted</code> :	Keine <code>Isolation</code> , fehlerhafte Daten können gelesen werden.
<code>ReadCommitted</code> :	(Standard), beim Lesen werden die Daten gesperrt. Andere Transaktionen können die Daten aber zwischenzeitlich ändern.
<code>RepeatableRead</code> :	Alle benutzten Daten der Transaktion sind für Updates gesperrt. Andere Transaktionen können die Daten aber zwischenzeitlich löschen.
<code>Serializable</code> :	Komplette <code>Isolation</code> und sperren der bearbeiteten Daten, sehr langsam, da immer nur eine Transaktion im System ausgeführt wird.

☞ Erstellen Sie im `ManageEventsModule` die neue öffentliche Methode `UpdateDatabase(...)`, welche ein `DataSet` vom Typ `ManageEventsDataSet` als Parameter übernimmt.

Nach dem Öffnen der Datenbankverbindung über die Connection `cnEvents` wird das Transaction Objekt `trans` deklariert und mit der `BeginTransaction(...)` Methode initialisiert. Der `IsolationLevel` ist auf `ReadCommitted` zu setzen.

Durch das Setzen der Transaction Property aller Update und Insert Command Objekte auf das Transaktionsobjekt werden Updates und Inserts in die Transaktion aufgenommen. Dies erfolgt über die `TableAdapter`.

In einem try-Block prüft man mit der Methode `HasChanges()` ob das übergebene Dataset Änderungen enthält.

Ist das der Fall, müssen die `Update(...)` Methoden der verschiedenen `TableAdapter` Objekte entsprechend der referentiellen Integrität aufgerufen werden. Die Elterntabellen sind zuerst abzugleichen.

Sind keine Fehler aufgetreten, so schließt man die Transaktion mit dem Aufruf von `Commit()` ab, die Daten sind an die Datenbank übertragen.

Bei einem Fehler führt man einen Rollback aus und wirft die aufgetretene Exception in den beiden catch-Blöcken mit `throw` an die aufrufende Methode zurück. Der Abgleich mit der Datenbank wird abgebrochen.

Im finally-Block schließt man die Datenbankverbindung.

Zuletzt lädt man über die bereits in der Version 1 erstellte Methode `LoadData(...)` die aktuellsten Daten aus der Datenbank in das Dataset `dsEvents`.

```
public void
    UpdateDatabase(ManageEventsDataset dsEvents)
{
    // Datenbankverbindung oeffnen
    ...

    // Transaktionen anlegen
    SqlTransaction trans =
        cnEvents.BeginTransaction(IsolationLevel.ReadCommitted);

    // Update und Insert Commands zur Transaktion inzufuegen
    taEvents.Adapter.UpdateCommand.Transaction = trans;
    taEvDaten.Adapter.UpdateCommand.Transaction = trans;
    ...
    taEvents.Adapter.InsertCommand.Transaction = trans;
    ...

    try
    {
        // Sind Aenderungen vorhanden?
        if(ds.HasChanges( ))
        {
            // Updates und Inserts in der richtigen Reihenfolge
            taKategorie.Update(dsEvents.tbl_EvKategorie);
            taVeranstalter.Update(dsEvents.tbl_EvVeranstalter);
            ...
        }

        // Hat alles geklappt dann...
        trans.Commit( );
    }

    catch(SqlException ex)
    {
        // Sonst Rollback
        trans.Rollback( );
        throw ex;
    }
    catch(System.Exception ex)
    {
        // Sonst Rollback
        ...
    }
    finally
    {
        // Datenbank schliessen
        if(this.cnEvents.State == ConnectionState.Open)
            this.cnEvents.Close( );
    }

    // Aktuelle Daten aus der DB holen
    LoadData(dsEvents);
}
```

☞ *Kompilieren Sie das `ManageEventsModule` und korrigieren Sie alle aufgetretenen Fehlermeldungen.*

☞ *Betrachten Sie das Klassendiagramm der Klasse `ManageEventsModule` mit der neuen Methode.*

3 Die Freigabe der Events zur Buchung (UC 04.12)

Laut den Ergebnissen der Anforderungsanalyse müssen Events bevor sie gebucht werden können vom Geschäftsführer der Fun Events GmbH zur Buchung freigegeben werden. Ist ein Event einmal freigegeben, so dürfen Preise, Termine sowie die Veranstaltungsorte nicht mehr verändert werden. Ebenso ist die Freigabe unwiderruflich.

3.1 Neue Datenbindungen in XAML

Um das Bearbeiten der Eventdaten von freigegebenen Events zu verhindern, setzt man diese im User Interface auf schreibgeschützt.

Da unter Windows Presentation Foundation nicht nur die Text Property an Datenquellen gebunden werden kann, nutzt man die `IsReadOnly` Properties der Steuerelemente `ed_BeginnTextBox`, `ed_EndedTextBox`, `ed_PreisTextBox`, `ed_StartOrtTextBox` und `ed_ZielOrtTextBox`. Über simple binding ist diese Property an den Wert des Feldes `ed_Freigegeben` aus der Datenquelle `tbl_EventDaten` zu binden.

☞ Öffnen Sie das `ManageEventsWindow` in der XAML Ansicht und navigieren Sie zum Textfeld `ed_BeginnTextFeld`.

☞ Fügen Sie ein Binding für die `IsReadOnly` Property auf das Datenfeld `ed_Freigegeben` im XAML Code hinzu!

☞ Orientieren Sie sich dabei an der Bindung der Text Property!

Das Binding wird im XAML Code für das entsprechende Steuerelement codiert.

```
<TextBox x:Name="ed_BeginnTextBox"...IsReadOnly="{Binding ...}".../>
```

☞ Verfahren Sie mit den Textfeldern für Termine, Orte und Preis analog dazu.

☞ Testen Sie den neuen „Schreibschutz“.

Bei allen freigegebenen Events sind diese Textfelder nun schreibgeschützt.

3.2 Das Abfangen einer Freigabe

Da eine Freigabe eines Events nicht zurückgenommen werden darf, muss auch das Steuerelement `ed_FreigegebenCheckBox` nach einer Freigabe deaktiviert werden. Dies kann mit dem `Checked` bzw. dem `Unchecked` Event des Steuerelements abgefangen werden.

☞ Generieren Sie mit dem Visual Studio die Handler Methoden `ed_FreigegebenCheckBox_Checked(...)` und `ed_FreigegebenCheckBox_Unchecked(...)` für das Steuerelement.

In dieser Methode muss nur noch die `Enabled` Property des Steuerelements `ed_FreigegebenCheckBox` gesetzt werden.

```
private void ed_FreigegebenCheckBox_Checked(...)
{
    // Event ist freigegeben
    ed_FreigegebenCheckBox.IsEnabled = false;
}
```

Die Freigabe eines Events „verriegelt“ sich nun quasi selbst und ist unwiderruflich.

☞ *Implementieren Sie die `ed_FreigegebenCheckBox_Unchecked(...)` analog zum `Checked Event!`*

Die Änderungen bei der Freigabe eines Events sind sofort in das Dataset zu übertragen. Dazu benötigt man die **UpdateSourceTrigger** Eigenschaft der Datenbindung an die Checkbox. Über die UpdateSourceTrigger Einstellung kann der Zeitpunkt der Aktualisierung der Datenquelle bestimmt werden. Bei Textboxen und Checkboxes ist die Standardeinstellung auf „LostFocus“ gesetzt. Über die Einstellung „PropertyChanged“ können Änderungen der Checkbox unmittelbar erfasst und im Dataset gespeichert werden.

Als letzte Aktion ist daher das Binding für die Checkbox `ed_FreigegebenCheckBox` im XAML Code abzuändern.

☞ *Bearbeiten Sie den XAML Code der Checkbox `ed_FreigegebenCheckBox`.*

```
<CheckBox ... IsChecked="{Binding ed_Freigegeben, ... UpdateSourceTrigger=PropertyChanged}.../>
```

Mit der UpdateSourceTrigger Eigenschaft der Bindung überträgt man die Änderungen der Checkbox direkt in das Dataset.

Klickt der Benutzer zur Freigabe eines Events auf die CheckBox, so werden die Eingabefelder für Termine, Orte und Preis sofort zur Bearbeitung gesperrt.

☞ *Betrachten Sie das Klassendiagramm des `ManageEventsWindows` mit den beiden neuen Handler Methoden.*

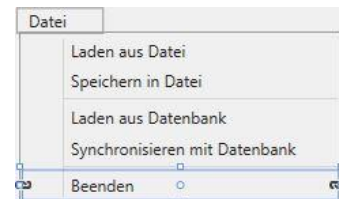
4 Der Abgleich mit der Datenbank (UC 04.14)

Die lokal auf dem Laptop gespeicherten Daten des Datasets `manageEventsDataset` müssen im Geschäft mit der Datenbank abgeglichen werden. Dazu benötigt der Laptop eine Netzwerkverbindung zum Firmennetzwerk der FunEvents GmbH.

4.1 Eine Erweiterung des Menüs

Zum Abgleich der Datenbank mit dem Dataset des EMS besteht bereits die Methode `UpdateDatabase(...)` in der Mittelschicht. Der Benutzer kann das Update mit einem neu zu implementierenden Menüpunkt starten.

- ☛ Erstellen Sie den neuen Menüpunkt `synchronisierenMitDatenbankMenuItem` mit der Text Property `S_ynchronisieren mit Datenbank`.
- ☛ Betrachten Sie auch das Klassendiagramm des `ManageEventsWindows`.



4.2 Die Synchronisation mit der Datenbank

Bevor ein Abgleich mit der Datenbank gefahren werden kann, müssen zunächst alle noch laufenden Bearbeitungsvorgänge am Dataset abgeschlossen sein. Nur auf diese Weise ist gewährleistet, dass auch wirklich alle geänderten Daten der lokalen Applikation zur Datenbank gesendet werden. Eine einfache Möglichkeit alle Bearbeitungen an einer Tabelle eines Datasets zu beenden bietet der Einsatz von `BindingGroup` - Objekten. Mit der Methode `UpdateSources()` wird jeder noch offene Bearbeitungsvorgang einer `BindingGroup` gestoppt und an die Datenquelle übertragen. Später können zusätzlich eine benutzerdefinierte Prüfung der Eingabewerte in der GUI sowie ein Rollback von Änderungen erfolgen.

Eine `Binding Group` definiert mehrere zusammen gehörende Steuerelemente, welche an dieselbe Datenquelle gebunden sind (z.B. an die `tbl_Events`). Da alle Child Steuerelemente die `BindingGroup` des enthaltenden Containers übernehmen, ist nur für das `eventGrid` bzw. das `eventDatenGrid` eine `BindingGroup` zu definieren.

Beispiel `eventGrid` im XAML Code:

Innerhalb des Grids wird die `BindingGroup` definiert. Über das Objekt `eventBindingGroup` kann im Quelltext die Aktualisierung der Quelle (`tbl_Events`) gesteuert werden.

```
<Grid x:Name="eventGrid" ...>
  ...
  <Grid.BindingGroup>
    <BindingGroup x:Name="eventBindingGroup">
      </BindingGroup>
    </Grid.BindingGroup>
  ...
</Grid>
```

- ☛ Erstellen Sie die `BindingGroups` `eventBindingGroup` und `eventDatenBindingGroup` im XAML Code der zugehörigen Grids.

In einer neu zu erstellenden Handler Methode werden die `BindingGroups` sowie die Mittelschichtkomponente `ManageEvents.dll` zum Update mit der Datenbank genutzt.

- ☛ Fangen Sie mit Hilfe des Visual Studios den Click Event auf den Menüpunkt `synchronisierenMitDatenbankMenuItem` in einer Handler Methode.

Im try-Block erzeugt man ein Objekt vom Typ der Mittelschichtklasse `ManageEventsModule`.

Anschließend beendet man über die `Binding-Groups` die aktuelle Bearbeitung der Tabellen.

Die eigentliche Datenübertragung beginnt in der `Auswahlentscheidung`. `HasChanges()` prüft zuerst, ob überhaupt geänderte Daten im Dataset vorliegen.

Ist dies der Fall, meldet eine `Message Box` den Start der Datenübertragung. Der `Cursor` wird zum besseren Verständnis für den User auf `Wait` gesetzt.

Da nur die geänderten Datensätze an die Datenbank gesandt werden sollen, speichert man die Änderungen über die `GetChanges()` Methode im Dataset `dsChanges`. Der Aufruf von `UpdateDatabase(...)` liefert nach dem Abgleich der Daten mit der Datenbank ein aktuelles Dataset mit den neuesten Daten in `dsChanges` zurück.

Mit `Clear()` und `Merge(...)` übernimmt man die aktuellsten Daten in das Dataset `dsEvents`. `AcceptChanges()` sorgt für ein Rücksetzen des Status. Ab jetzt werden wieder alle Änderungen im Dataset mitprotokolliert.

Im letzten Schritt aktualisiert man die Anzeige durch den Aufruf von `RefreshBindings()`.

Sind keine Änderungen im Dataset aufgelaufen, so gibt man dies dem User über eine `Message Box` bekannt.

Die beiden `catch`-Blöcke fangen aufgetretene Fehler auf und melden diese an den Benutzer weiter.

Auch der `Cursor` ist im `finally`-Block wieder auf die Standardanzeige zu setzen.

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

        // Alle aktuellen Bearbeitungen stoppen
        eventBindingGroup.UpdateSources();
        ...

        // Sind Änderungen im Dataset?
        if(dsEvents.HasChanges())
        {
            MessageBox.Show("Die Änderungen werden zur "
                + "Datenbank übertragen.", "Datentransfer",
                MessageBoxButtons.OK, MessageBoxIcon.Information);

            Cursor = Cursors.Wait;

            // Änderungen herausfiltern
            ManageEvents.ManageEventsDataset dsChanges =
                ((ManageEvents.ManageEventsDataset)
                    dsEvents.GetChanges());

            // Update mit der DB fahren
            m.UpdateDatabase(dsChanges);

            // Aktuellste Daten der DB übernehmen
            dsEvents.Clear();
            dsEvents.Merge(dsChanges);
            dsEvents.AcceptChanges();

            // Databindings aktualisieren
            RefreshBindings();
        }
        else
        {
            MessageBox.Show("Im Dataset finden sich zur Zeit "
                + "keine Änderungen zum Abgleich mit der Datenbank.",
                "Kein Datentransfer",
                MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
    }
    catch(Microsoft.Data.SqlClient.SqlException ex)
    {
        MessageBox.Show(...);
    }
    catch(Exception ex)
    {
        ...
    }
    finally
    {
        this.Cursor = Cursors.Arrow;
    }
}
```

Achtung: Da auch beim lokalen Speichern der Daten alle Bearbeitungsvorgänge abgeschlossen sein müssen, ist die Methode `speichernInDateiMenuItem_Click(...)` zu ergänzen.

☞ Überarbeiten Sie die Methode `speichernInDateiMenuItem_Click(...)`.

Zu Beginn dieser Methode beendet man wieder alle aktuellen Bearbeitungsvorgänge über die `BindingSource` Objekte.

```
private void speichernInDateiMenuItem_Click(...)
{
    ...
    // Alle aktuellen Bearbeitungen stoppen und übernehmen
    ...
}
```

5 Neue Events und Eventdaten (UCs 04.8, 04.9)

Auf Geschäftsreisen requiriert der Geschäftsführer der Fun Events GmbH, Herr Fringsen, neue Events und Eventdaten (neue Termine für bestehende Events) bei den Veranstaltern. Diese neuen Daten müssen in das EMS aufgenommen werden können.

5.1 Neue Steuerelemente des ManageEventsWindows

Zur Aufnahme neuer Events und Eventdaten ist die Benutzeroberfläche des ManageEventsWindows um vier neue Buttons in einem Grid und zwei Menüpunkte zu ergänzen.

Über die Buttons **Neuen Event erstellen** und **Eventtermin hinzufügen** können neue Events und Eventdaten eingegeben werden.

Der Button **Änderungen übernehmen** dient zum Speichern, der Button **Abbrechen** zum Verwerfen der neu eingegebenen Daten im Event Management System.

☞ Erstellen Sie die vier Buttons in einem Grid auf der Benutzeroberfläche entsprechend der folgenden Tabelle.

Tabelle Benutzerschnittstelle

Steuerelement	Objektbezeichnung	Text
Grid	buttonGrid	--
Button	btnNeuerEvent	_Neuen Event erstellen
Button	btnNeuerTermin	Eventtermin _hinzufügen
Button	btnUebernehmen	_Änderungen übernehmen
Button	btnAbbrechen	Abb_rechen

☞ Setzen Sie die *IsEnabled* Eigenschaft der Buttons *btnUebernehmen* und *btnAbbrechen* auf *False* und verankern Sie das Grid.

Über ein zugehöriges Menü ist das Erstellen neuer Events und Eventdaten zu steuern.

☞ Fügen Sie dem *ManageEventsWindow* die neuen Menüpunkte gemäß der Tabelle hinzu.

Tabelle Menü

Objekt (Name)	Text
bearbeitenMenuItem	_Bearbeiten
eventMenuItem	_Event
neuerEventMenuItem	_Neuen Event erstellen
eventterminHinzufügenMenuItem	Eventtermin _hinzufügen



☞ Betrachten Sie auch das Klassendiagramm des *ManageEventsWindows*.

5.2 Das Hinzufügen neuer Events (UC 04.9)

Klickt der User auf den Button btnNeuerEvent, so sind dem Dataset dsEvents in zwei Tabellen (tbl_Events, tbl_EventDaten) jeweils ein neuer Datensatz hinzuzufügen.

Zuvor allerdings sollten die Bearbeitung der bestehenden Daten gesperrt und die Buttons **Änderungen übernehmen** sowie **Abbrechen** aktiviert werden. Zu diesem Zweck ist zunächst die neue private Methode SetProtection(...) mit dem boolschen Parameter stat zu implementieren.

☛ Erstellen Sie die Methode SetProtection(...) und betrachten Sie das Klassendiagramm des ManageEventsWindows.

In dieser Methode setzt man die IsEnabled Properties von ed_BeginnComboBox, et_BezeichnungListBox, btnNeuerEvent, btnNeuerTermin, btnVeranstalter, btnKategorie, dateiMenuItem und bearbeitenMenuItem auf den Parameter stat. Die Navigationsleisten eventNavBar und EventDatenNavBar sind ebenfalls mit einzubeziehen. Mit der Übergabe von true oder false beim Methodenaufwurf kann somit das Freigeben oder Sperren der Steuerelemente bewerkstelligt werden. Die Buttons Abbrechen und Übernehmen sind gegenläufig zu den anderen Steuerelementen zu aktivieren.

```
private void SetProtection(bool stat)
{
    // Menues/ Textfelder/ Buttons sperren oder freigeben
    ed_BeginnComboBox.IsEnabled = stat;
    et_BezeichnungListBox.IsEnabled = stat;
    ...
    eventNavBar.IsEnabled = stat;

    btnAbbrechen.IsEnabled = !stat;
    ...
}
```

Zum Abfangen des Click-Events auf den Button btnNeuerEvent lässt man sich vom Visual Studio eine Handler Methode generieren. Hier erfolgt das Anlegen der neuen Datensätze in den beiden Tabellen.

☛ Erzeugen Sie mit Hilfe des Visual Studios die neue Handler Methode btnNeuerEvent_Click(...), welche das Klick-Ereignis abfängt.

In der Handler Methode werden neue Datenreihen im Dataset erzeugt, welche ohne Initialisierung zu Fehlern bei der Überprüfung der Constraints führen könnten. Bis die Daten durch den User übernommen oder verworfen werden, ist die Überprüfung durch das Setzen der

Property EnforceConstraints auf false zu unterbinden.

Mit der NewRow() Methode der Events Tabelle erstellt man eine neue Zeile für die Eventstabelle und fügt sie der Rows Collection der Tabelle durch den Aufruf von Add() hinzu.

```
private void btnNeuerEvent_Click(...)
{
    // Constraints ausschalten
    dsEvents.EnforceConstraints = false;

    // neue Rows erzeugen und hinzufuegen
    ManageEvents.ManageEventsDataset.tbl_EventsRow evRow =
    (ManageEvents.ManageEventsDataset.tbl_EventsRow)dsEvents.tbl_Events.NewRow( );
    dsEvents.tbl_Events.Rows.Add(evRow);
    ...
}
```

Für die Zeile der Eventdaten wählt man zunächst die beiden ersten Einträge der Veranstalter- und Kategorietabellen. `SetParentRow(...)` setzt die Voreinstellung.

Auch in der Eventdaten Tabelle ist eine neue Zeile zu erzeugen. Nach dem Erstellen der Zeile implementiert man zuerst die zugehörige Foreign Key-Beziehung zwischen Event und Eventdaten mit Hilfe der `SetParentRow(...)` Methode, welche ein `DataRow` Objekt als Parameter erwartet.

Die Anzahl der Teilnehmer, die Freigabe, der Preis und das Flag zur Veranstalterbenachrichtigung sind ebenfalls auf Standardwerte zu initialisieren, bevor man die Zeile der Tabelle hinzufügt.

Um die Anzeige synchron zum Dataset zu halten sind die Position der View für die Event Tabelle auf die neue Row zu setzen. Die neu eingefügte Zeile liegt am Ende der View.

Der Aufruf von `SetProtection(...)` sorgt daraufhin für das Freigeben und Sperren der betroffenen Steuerelemente.

```
// ParentRows setzen
evRow.SetParentRow(
    dsEvents.tbl_EvKategorie.Rows[0]);
evRow.SetParentRow(...);

ManageEvents.ManageEventsDataset.tbl_EventDatenRow
evDaRow = ...;

// Parentrow und Standardwerte setzen
evDaRow.SetParentRow(evRow);

evDaRow.ed_AktTeilnehmer = 0;
evDaRow.ed_Freigegeben = false;
evDaRow.ed_Preis = 0.0M;
evDaRow.ed_VeranstalterBenachrichtigt = false;
dsEvents.tbl_EventDaten.Rows.Add(evDaRow);

// Zeile der Tabelle tbl_EventDaten hinzufügen
...


// Auf aktuelle Position springen und Refresh
eventView.MoveCurrentToLast();

// Eingabefelder freigeben
this.SetProtection(false);

// Editiervorgang starten
evRow.BeginEdit();
evDaRow.BeginEdit();
}
```

Die beiden `BeginEdit()` Methoden der `DataRow`s schalten die Zeilen zuletzt für die Eingabe von Daten frei, bis der Benutzer Übernehmen oder Abbrechen wählt.

Auch der Klick auf den Menüpunkt `neuerEventMenuItem` soll das Anlegen eines neuen Events ermöglichen.

 *Verknüpfen Sie den Click-Event auf den Menüpunkt `neuerEventMenuItem` mit der gerade erstellten Handler Methode `btnNeuerEvent_Click(...)`.*

5.3 Das Hinzufügen neuer Eventdaten (UC 04.8)

Ähnlich wie das Hinzufügen neuer Events läuft das Anlegen eines neuen Eventtermins ab. Beim Klick auf den Button `Eventtermin hinzufügen` muss in der Eventdaten-Tabelle eine neue Zeile generiert und zur Bearbeitung frei geschaltet werden, bis der User `Abbrechen` oder `Übernehmen` auswählt.

 *Fangen Sie das Click Event auf den Button `btnNeuerTermin` in der Handler Methode `btnNeuerTermin_Click(...)` ab.*

Um Fehler bei der Bearbeitung zu vermeiden, ist zuerst die Überprüfung der Constraints im Dataset abzuschalten.

```
private void btnNeuerTermin_Click(object sender, RoutedEventArgs e)
{
    // Constraints ausschalten
    ...
}
```

Anschließend erzeugt man wieder eine neue DataRow für die Eventdaten-Tabelle. Dabei setzt man die NewRow() Methode ein, um eine Zeile mit allen Spalten der Eventdaten Tabelle zu erhalten.

Als Elternzeile ist die aktuell gewählte DataRow in der Events-Tabelle über die eventView zu ermitteln und zu setzen. Die Spalten ed_AktTeilnehmer, ed_Freigegeben, ed_Preis und ed_VeranstalterBenachrichtigt erhalten Standardwerte.

Nach dem Hinzufügen der DataRow evDataRow zur Eventdaten-Tabelle springt man zur Synchronisation mit der View eventDatenView auf die aktuelle Zeile und ruft die BeginEdit() Methode zum Start des Editiervorgangs auf.

```
...
// neue DataRow fuer Eventstable erzeugen
DataRow evDataRow = ...

// Parentrow und Standardwerte setzen
evDataRow.SetParentRow(
    ((DataRowView)eventView.CurrentItem).Row);
evDataRow.ed_AktTeilnehmer = 0;
...

// Row zur Eventdaten Tabelle hinzufügen
...

// Auf aktuelle Zeile springen
eventDatenView. ...

// Mit den Aenderungen in evDataRow beginnen
...

// Editfelder freischalten
...
}
```

SetProtection(...) schaltet im letzten Schritt die Buttons Abbrechen und Übernehmen für den User frei.

Wie beim Erstellen eines neuen Events ist auch bei einem neuen Termin der Klick auf den Menüpunkt „Termin hinzufügen“ mit der gerade implementierten Handler Methode zu koppeln.

☞ *Verbinden Sie das Click Event auf den Menüpunkt eventTerminHinzufuegenMenuItem mit der Handler Methode btnNeuerTermin_Click.*

5.4 Das Abbrechen der Bearbeitung

Sollen die Änderungen beim Anlegen eines neuen Events oder eines neuen Eventtermins nicht im Dataset übernommen werden, so hat der Benutzer die Möglichkeit den Abbrechen-Button zu betätigen. Sowohl neu angelegte Events als auch neue Termine werden daraufhin wieder verworfen.

☞ *Erstellen Sie mit dem Visual Studio die neue Handler Methode btnAbbrechen_Click(...), welche das Klick-Ereignis auf den Abbrechen-Button abfängt.*

☞ *Betrachten Sie das Klassendiagramm des ManageEventsWindows.*

In der Handler Methode definiert man zunächst die lokale Variable evRow, welche über die View eventView mit der aktuell bearbeiteten DataRow initialisiert wird.

```
private void btnAbbrechen_Click(object sender, System.EventArgs e)
{
    // lokale DataRow fuer das bearbeitete Event
    DataRow evRow =
        ((DataRowView)eventView.CurrentItem).Row;
```

Im Anschluss daran muss zwischen zwei Möglichkeiten beim Abbrechen der Bearbeitung unterschieden werden:

- es wurde ein neuer Event hinzugefügt, das heißt es gibt zum Eintrag in der Mastertabelle `tbl_Events` nur einen Eintrag in der Detailtabelle `tbl_EventDaten`,
- es wurde ein neuer Termin zu einem bestehenden Event hinzugefügt.

Im ersten Fall ist mit der `Remove(...)` Methode zunächst die abhängige `EventDataRow` aus dem Dataset zu löschen. Diese kann über die `GetChildRows(...)` Methode ermittelt werden.

Ebenfalls mit `Remove(...)` löscht man daraufhin die Zeile mit dem neuen Event.

`EnforceConstraints = true` schaltet die Überprüfung der Constraints im Dataset wieder ein, bevor mit der Methode `MoveCurrentToFirst()` auf eine gültige Zeile in der Eventtabelle navigiert wird.

```
...
// ein neuer Event wurde hinzugefügt
if(((CollectionView)eventDatenView).Count == 1)
{
    dsEvents.tbl_EventDaten.Rows.Remove(
        evRow.GetChildRows("tbl_Events_tbl_EvDaten_FK")[0]);

    dsEvents.tbl_Events.Rows.Remove(evRow);
    dsEvents.EnforceConstraints = true;
    eventView.MoveCurrentToFirst();
}
```

Ist einem bestehenden Event nur ein neuer Eventtermin hinzugefügt worden, so darf beim Betätigen des Abbrechen Buttons auch nur dieser Termin und nicht das ganze Event wieder gelöscht werden. Hierzu kommt ebenfalls die `Remove(...)` Methode zu Einsatz.

Über die `Count` Property der `eventDatenView` ermittelt man den zuletzt hinzugefügten Eventtermin zum Löschen.


Die Constraints des Datasets sind wieder zuzuschalten.

Am Ende der Methode aktiviert man die Steuerelemente zur Bearbeitung der Events mit Hilfe der Methode `SetProtection(...)`.

```
...
// ein neuer Eventtermin wurde hinzugefügt
else
{
    dsEvents.tbl_EventDaten.Rows.Remove(
        evRow.GetChildRows(
            "tbl_Events_tbl_EvDaten_FK")(((CollectionView)eventDatenView).Count - 1));


    // Constraints im Dataset wieder zuschalten
    ...
}

// Schreibschutz wieder herstellen
this.SetProtection(true);
}
```

 **Testen Sie das Hinzufügen neuer Events und Eventdaten sowie das Abbrechen des Bearbeitungsvorgangs.**

5.5 Die Übernahme der Änderungen

Sollen die Änderungen beim Erstellen eines neuen Events oder beim Hinzufügen eines neuen Eventdaten Datensatzes dauerhaft in das Dataset `dsEvents` übernommen werden, so muss der User auf den Button Änderungen übernehmen klicken.

 **Generieren Sie mit dem Visual Studio die Handler Methode `btnUebernehmen_Click(...)`, welche das Click-Event auf den entsprechenden Button abfängt.**

Zuerst definiert man zwei lokale Variablen für die neuen Event- bzw. `EventDataRows`.

```
private void btnUebernehmen_Click(object sender, System.EventArgs e)
{
    // lokale Rows für EventDaten und Event
    DataRow evRow =
        ((DataRowView)eventView.CurrentItem).Row;
    DataRow evDataRow =
        ((DataRowView)eventDatenView.CurrentItem).Row;
    ...
}
```

In einem try-Block werden zuerst die Daten aus den Steuerelementen in das Dataset übertragen. Dazu nutzt man wieder die BindingGroup.

Anschließend prüft man, ob ein neuer Event hinzugefügt wurde. In diesem Fall ist die DataRowState Property des DataRow Objektes auf Added gesetzt.

Beim Hinzufügen des neuen Events müssen dessen Veranstalter und Kategorie gewählt werden.

SetParentRow(...) übernimmt dazu als Parameter die entsprechende DataRow der Elterntabelle sowie das Relations-Objekt.

Die EndEdit() Methoden beenden den Bearbeitungsvorgang und speichern die neuen Daten in den DataRows.

Nach dem Zuschalten der Constraint-Überprüfung im Dataset ist nur noch die Anzeige der Steuerelemente über RefreshBindings() auf den aktuellsten Stand zu bringen. Die Navigationsleisten werden dadurch ebenfalls neu initialisiert.

Aufgetretene Fehler werden im catch-Block abgefangen. Nach der Ausgabe einer Fehlermeldung löscht man die neuen Daten durch einen Aufruf der RaiseEvent(...) Methode des Buttons btnAbbrechen.

Am Ende der Methode aktiviert man im finally-Block die Steuerelemente zur Bearbeitung der Events mit Hilfe der Methode SetProtection(...).

```
try
{
    // Daten aus Steuerelementen übernehmen
    eventBindingGroup.UpdateSources( );
    eventDatenBindingGroup.UpdateSources( );

    // Neuer Event wurde erstellt
    if(evRow.RowState == DataRowState.Added)
    {
        // ParentRows für Event setzen
        // Eventveranstalter zuordnen
        evRow.SetParentRow(
            dsEvents.tbl_EvVeranstalter.Rows[cboVeranstalter.SelectedIndex],
            dsEvents.Relations["tbl_EvVeranstaltertbl_Events_FK"]);


        // Eventkategorie zuordnen
        ...
    }
}

// Änderungen an Event und
// gewählter EventDataRow übernehmen
evDaRow.EndEdit( );
evRow....

// Constraints im Dataset zuschalten
...

// Refresh der Bindungen
RefreshBindings( );
}
catch(Exception ex)
{
    MessageBox.Show(ex.Message, "Fehler beim Editieren",
        MessageBoxButtons.OK, MessageBoxIcon.Error);

    // Vorgang abbrechen
    btnAbbrechen.RaiseEvent(...);
}
finally
{
    // Schreibschutz wiederherstellen
    SetProtection(true);
}
```

 Begutachten Sie das Klassendiagramm des ManageEventsWindows.

6 Neue Veranstalter erstellen (UC 04.11.1)

Bei seinen Geschäftsreisen nimmt der Geschäftsführer der FunEvents GmbH neben neuen Events auch neue Eventveranstalter im System auf. Die Bearbeitung der Eventveranstalter soll im EMS mit Hilfe eines neuen WPF Fensters erfolgen.

6.1 Das Erstellen des ManageVeranstalterWindows

Zur Verwaltung der Veranstalter wird dem Projekt ein neuer Windows Form hinzugefügt.

☛ Erstellen Sie im Projekt *ManageEventsApp* das neue WPF Window *ManageVeranstalterWindow* (im Projekt / Hinzufügen / Fenster (WPF)).

☛ Setzen Sie die Größe des Fensters auf 650 und 460.

☛ Der *ResizeMode* ist auf „NoResize“ zu stellen.

☛ Als *Title Property* kann „Bearbeitung der Eventveranstalter“ eingegeben werden.

6.1.1 Die Steuerelemente des ManageVeranstalterWindows

Nach dem Erstellen des Fensters sollen die Steuerelemente im XAML Editor des Visual Studios angepasst werden. Für die Navigationsleiste kommt erneut die *NavBar* zum Einsatz.

☛ Fügen Sie dem Fenster im XAML Code den Verweis auf den *NavigationBar Namespace* hinzu!

Die Steuerelemente liegen in zwei Grids. Grid 1 (*veranstalterGrid*) umfasst die Daten, darunter liegt die *Navbar*. Grid 2 (*buttonGrid*) beinhaltet die Buttons.

Damit Steuerelemente über mehrere Zeilen oder Spalten eines Grids gelagert werden können, verwendet man die XAML Properties *Grid.RowSpan* und *Grid.ColumnSpan*.

☛ Erstellen Sie die Grids und Steuerelemente entsprechend der Vorgaben aus den folgenden Tabellen im XAML Editor!

☛ Achten Sie auch auf die *DataBindings* an die Spalten der Veranstaltertabelle aus dem *Dataset*.

```
<Grid x:Name="veranstalterGrid" VerticalAlignment="Top" Margin="10,10,10,0" Height="335" >
<Grid.RowDefinitions>
  <RowDefinition Height="1*" />
  ...
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
  ...
</Grid.ColumnDefinitions>
<ListBox x:Name="ev_FirmaListBox" Margin="3" DisplayMemberPath="ev_Firma" ItemsSource="{Binding}"
  Grid.Row="0" Grid.RowSpan="10" IsSynchronizedWithCurrentItem="True" />
<Label Margin="3" Grid.Column="1" Grid.Row="0" Content="Eventveranstalter" Grid.ColumnSpan="3" />
<TextBox x:Name="ev_FirmaTextBox"
  Text="{Binding ev_Firma, Mode=TwoWay, NotifyOnValidationError=true,
  ValidatesOnExceptions=true}" Grid.Row="1" Margin="5" Grid.Column="1" Grid.ColumnSpan="3" />
<Label Margin="3" Grid.Column="1" Grid.Row="2" Content="Straße" />
...
</Grid>
```


Tabelle Benutzerschnittstelle (1), Datenfenster

Steuerelement	Objektbezeichnung	Properties
Grid	veranstalterGrid	zehn Zeilen, vier Spalten Spalte 1 doppelt so breit wie Spalte 2-4
ListBox	ev_FirmaListBox	Zeile 0, Spalte 0, Grid.RowSpan =10 IsSynchronizedWithCurrentItem = true
Label		Eventveranstalter
TextBox	ev_FirmaTextBox	-
Label		Straße
TextBox	ev_StrasseTextBox	-
Label		Hausnummer
TextBox	ev_HNummerTextBox	-
Label		PLZ
TextBox	ev_PLZTextBox	-
Label		Ort
TextBox	ev_OrtTextBox	-
Label		Telefon
TextBox	ev_TelefonTextBox	-
Label		Fax
TextBox	ev_FaxTextBox	-
Label		E-Mailadresse
TextBox	ev_EMailTextBox	-

Tabelle Benutzerschnittstelle (2), Toolbox

Steuerelement	Objektbezeichnung	Text
NavBar	veranstalterNavBar	
Grid	buttonGrid	Zwei Zeilen, drei Spalten
Button	btnNeuerVeranstalter	_Neuer Veranstalter
Button	btnUebernehmen	_Übernehmen
Button	btnAbbrechen	Abb_rechen
Button	btnSchliessen	_Schließen

☞ Setzen Sie die *IsEnabled* Properties der Buttons *btnAbbrechen* und *btnUebernehmen* auf *false*.

☞ Betrachten Sie das Klassendiagramm zur *ManageEventsApp* mit der neuen Klasse *ManageVeranstalterWindow*.

6.1.2 Die Anzeige der Datensätze

Für die Anzeige der Veranstalterdaten benötigt der Form eine Kopie des Datasets und eine View, welche mit der Veranstaltertabelle zusammen arbeitet.

☞ Deklarieren Sie das Dataset *dsVeranstalter* und die *CollectionView veranstalterView* im *ManageVeranstalterWindow*.

Das Dataset und die *CollectionView* sind als private Elemente der Klasse *ManageVeranstalterWindow* zu deklarieren.

```
// Dataset zur Bearbeitung der Veranstalter
private ManageEvents.ManageEventsDataset dsVeranstalter;
private...
```

Die Anzeige der Datensätze soll im Navigationsbereich mit Angabe des aktuellen sowie der enthaltenen Datensätze der Veranstaltertabelle in folgender Form erfolgen:

Datensatz x von y

Dies erledigt die Navigationsleiste VeranstalterNavBar. Auch dieser wird automatisch aktualisiert, wenn deren SourceView Property auf die View VeranstalterView gesetzt ist. Zur Anzeige der Anzahl und Position nutzt die NavBar dann die zugehörige View.

Beim Start der Veranstalterbearbeitung muss ein Verweis auf das Dataset übergeben werden. Ebenso sollte der im ManageEventsWindow gewählte Eventveranstalter zur Bearbeitung selektiert sein. Diese Aufgabe sowie das Binden der Veranstaltertabelle an die Steuerelemente zur Anzeige ist Aufgabe der neuen Methode SetDataSet(...). Die Methode übernimmt dazu als Parameter das ManageEventsDataset sowie den Index des gerade ausgewählten Veranstalterdatensatzes.

☞ Implementieren Sie die Methode `public void SetDataSet(...)` welche als Parameter ein `ManageEventsDataset` sowie einen integer index übernimmt.

☞ Betrachten Sie das Klassendiagramm des `ManageVeranstalterWindows`.

Innerhalb der Methodendefinition initialisiert man zuerst das Dataset dsVeranstalter mit dem übergebenen ManageEventsDataset.

```
// Dataset und Eventveranstalter zum Bearbeiten uebernehmen
public void SetDataSet(ManageEvents.ManageEventsDataset ds,
    int index)
{
    // Dataset initialisieren
    dsVeranstalter = ds;
```

Der DataContext des Grids ist auf das Dataset und die Tabelle tbl_EvVeranstalter zu setzen, damit die Daten in den Steuerelementen angezeigt werden.

```
//Databindings setzen
veranstalterGrid.DataContext = dsVeranstalter.tbl_EvVeranstalter;
```

View und Navigationsleiste initialisiert man im C# Code über die CollectionViewSource Klasse.

```
// View und NavBar initialisieren
veranstalterView =
    (CollectionView)CollectionViewSource.GetDefaultView(
        dsVeranstalter.tbl_EvVeranstalter);
veranstalterNavBar...
```

Durch das Setzen der Position Property der Viwe auf den übergebenen Index springt die Anzeige automatisch auf den im ManageEventsWindow gewählten Veranstalterdatensatz.

```
// aktuell gewaehlten Eventveranstalter setzen
veranstalterView.MoveCurrentToPosition(index);
}
```

Für die Steuerung des Datenflusses definiert man für das veranstalterGrid eine BindingGroup in XAML!

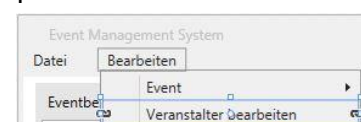
☞ Generieren Sie die BindingGroup veranstalterBindingGroup im XAML Code des Grids.

6.1.3 Änderungen im ManageEventsWindow

Zum Starten der Veranstalterbearbeitung ist auf der grafischen Benutzerschnittstelle des ManageEventsWindows bereits ein Button „Veranstalter bearbeiten“ vorhanden.

Die Bearbeitung der Veranstalter soll zusätzlich über einen Menüpunkt durch den User aufgerufen werden können.

☞ Der zugehörige Menüpunkt `veranstalterBearbeitenMenuItem` ist unter `Bearbeiten` einzufügen.



Für den Start der Veranstalterbearbeitung ist zuletzt nur noch das Click-Event auf den Button `btnVeranstalter` abzufangen.

☞ *Generieren Sie mit Hilfe des Visual Studios die neue Handler Methode `btnVeranstalter_Click(...)`.*

In der Handler Methode erzeugt man zunächst ein Objekt vom Typ `ManageVeranstalterWindow`.

Mit dem Aufruf von `SetDataSet(...)` übergibt man eine Referenz auf das `ManageEventsDataset` sowie den Index des aktuell ausgewählten Veranstalters an das Fenster.

`ShowDialog(...)` zeigt das `ManageVeranstalterWindow` im letzten Schritt als Dialog an.

// Startet das Editieren/ Anlegen von Veranstaltern

```
private void btnVeranstalter_Click(...)
{
    // Fenster erzeugen und anzeigen
    ManageVeranstalterWindow mvw = new ...

    mvw.SetDataSet(dsEvents, cboVeranstalter.SelectedIndex);

    mvw.ShowDialog();
}
```

☞ *Debuggen und testen Sie die Anzeige des `ManageVeranstalterWindows` im Projekt.*

☞ *Verbinden Sie auch das Klick-Ereignis auf den Menüpunkt `veranstalterBearbeitenMenulitem` mit der Handler Methode `btnVeranstalter_Click(...)` und betrachten Sie das Klassendiagramm.*

6.2 Das Bearbeiten der Veranstalter

6.2.1 Die Behandlung der Fehlermeldungen

Da bei der Eingabe der Veranstalterdaten bestimmte Felder, wie z. B. die Veranstalterbezeichnung und dessen Adresse, mit Werten gefüllt sein müssen, damit beim Zuschalten der Constraints im Dataset keine Exceptions auftreten, empfiehlt sich eine einheitliche Behandlung der Fehler. Für das Abfangen von aufgetretenen Fehlern soll daher die Methode `ShowErrorMessages(...)` erstellt werden.

☞ *Implementieren Sie die Methode `void ShowErrorMessages(object sender, RoutedEventArgs e)` im `ManageVeranstalterWindow`.*

☞ *Betrachten Sie die Methode im Klassendiagramm des Windows.*

In dieser Methode wird dem Benutzer eine Warnmeldung in einer `MessageBox` mit den zwei Auswahlbuttons *Ja* und *Nein* angezeigt. Die `MessageBoxResult` Variable `mr` speichert, welchen Button der Benutzer angeklickt hat.

Über diese Rückgabe der `MessageBox` im `MessageBoxResult` überprüft man im Anschluss daran, ob der User die fehlenden Einträge ergänzen oder die Bearbeitung abbrechen will.

```
// Fehlermeldung ausgeben
private void ShowErrorMessages(object sender, RoutedEventArgs e)
{
    MessageBoxResult mr =
        MessageBox.Show(((Exception)sender).Message
            + "\nVeranstalter, Straße, Hausnummer, Postleitzahl und Ort "
            + "müssen Werte enthalten.\n"
            + "Möchten Sie die fehlenden Einträge ergänzen?",
            "Fehler",
            MessageBoxButton.YesNo,
            MessageBoxImage.Error);

    // Abbrechen oder Ueberarbeiten?
    if(mr == MessageResult.No)
        btnAbbrechen.RaiseEvent(...);
    else
        ev_FirmaTextBox.Focus();
}
```

6.2.2 Das Anlegen neuer Veranstalter

Zum Anlegen neuer Veranstalter klickt der Benutzer auf den Button btnNeuerVeranstalter. Mit Abbrechen oder Übernehmen können die Daten nach der Eingabe dann verworfen oder in das Dataset zurück gespeichert werden. Dieses Klick-Ereignis gilt es in einer Handler Methode abzufangen.

☞ Erzeugen Sie die neue Handler Methode *btnNeuerVeranstalter_Click(...)* zur Behandlung des Click-Events.

Da während dem Anlegen eines neuen Datensatzes keine Navigation stattfinden darf, müssen alle Icons zur Navigation gesperrt werden.

Ebenso ist es aus Gründen der Datenkonsistenz untersagt, während des Bearbeitungsvorgangs erneut einen Datensatz anzulegen oder den Dialog zu schließen.

Zum Verwerfen oder zur Übernahme der Daten sind die entsprechenden Buttons gegenläufig zu aktivieren.

Nach dem Sperren bzw. Freigeben der Steuerelemente beginnt die eigentliche Arbeit in einem try-Block.

Zuerst erzeugt man einen neuen Datensatz. Mit dem Abschalten der Constraint Überprüfung verhindert man Fehlermeldungen durch noch nicht ausgefüllte Einzeldaten.

Die Add(...) Methode fügt den neuen Datensatz im nächsten Schritt der Veranstalter Tabelle hinzu bevor mit der veranstalterView die Navigation zum letzten Datensatz erfolgt.

Mit BeginEdit() der DataRow startet die Bearbeitung.

Damit der Benutzer sofort mit der Eingabe der Daten beginnen kann, ist der Focus nur noch auf das Textfeld ev_FirmaTextBox zu setzen.

Eventuell aufgetretene Ausnahmen leitet man an die zuvor erstellte Methode ShowErrorMessage(...) zur Bearbeitung weiter.

```
// Ein neuer Eventveranstalter wird hinzugefügt
private void btnNeuerVeranstalter_Click(...)
{
    // Buttons bei Bearbeitung sperren/ freigeben
    veranstalterNavBar.IsEnabled = false;

    btnNeuerVeranstalter....
    btnSchliessen....

    btnAbbrechen.IsEnabled = true;
    btnUebernehmen....;

    try
    {
        // neue Row erzeugen
        ManageEvents.ManageEventsDataset.tbl_EvVeranstalterRow verRow =

        // Ueberpruefung der Constraints zum Bearbeiten abschalten
        dsVeranstalter.EnforceConstraints = ...;

        // Row hinzufuegen und Anzeige updaten
        dsVeranstalter.tbl_EvVeranstalter.Rows.Add(...);

        veranstalterView.Move...

        // Bearbeitung der Zeile beginnen
        ...

        ev_FirmaTextBox.Focus( );
    }
    catch(Exception ex)
    {
        // Fehler abfangen und ausgeben
        ShowErrorMessage(ex, e);
    }
}
```

6.2.3 Das Abbrechen der Bearbeitung

Zum Abbrechen des aktuellen Bearbeitungsvorganges muss der Benutzer nur den entsprechenden Button betätigen.

☞ *Fangen Sie den Click-Event auf den Button btnAbbrechen in einer neuen Handler Methode ab.*

Beim Verwerfen der Daten ist der eben erzeugte Datensatz mit der Remove(...) Methode aus der Veranstalter Tabelle zu entfernen.

Die Überprüfung der Constraints kann wieder zugeschaltet werden.

Gegenläufig zur vorher erstellten Methode btnNeuerVeranstalter_Click(...) sind die Buttons zur Navigation, zum Schließen des Dialogs und zum Abbrechen bzw. Übernehmen erneut freizugeben oder zu sperren.

```
// Abbrechen der aktuellen Bearbeitung
private void btnAbbrechen_Click(object sender, RoutedEventArgs e)
{
    // neue Row wieder loeschen
    dsVeranstalter.tbl_EvVeranstalter.Rows.Remove(
        ((DataRowView)veranstalterView.CurrentItem).Row);

    // Constraints wieder herstellen
    ...

    // Buttons nach Bearbeitung sperren/ freigeben
    veranstalterNavBar.IsEnabled = true;
    ...
}
```

6.2.4 Die Übernahme der Daten

Die Übernahme der neuen Veranstalterdaten erfolgt in der Handler Methode.

☞ *Erstellen Sie die Handler Methode btnUebernehmen_Click(...) mit dem Visual Studio und betrachten Sie das Klassendiagramm des ManageVeranstalterWindows.*

Zur Übernahme der Änderungen sendet man die Daten von den Steuerelementen in das Dataset.

EndEdit() schließt die Bearbeitung der aktuellen Zeile ab.

Wie beim Abbrechen schaltet man die Constraints wieder zu und sperrt bzw. gibt die entsprechenden Buttons frei.

Zusätzlich muss die Navigationsleiste aktualisiert werden, da jetzt ein Eintrag mehr in der Veranstaltertabelle enthalten ist.

Ausnahmen übergibt man an die Methode ShowErrorMessages(...).

```
// Uebernimmt die aktuellen Aenderungen
private void btnUebernehmen_Click(object sender, RoutedEventArgs e)
{
    try
    {
        // Daten an Dataset senden
        veranstalterBindingGroup.UpdateSources( );

        // aktuelle Bearbeitung beenden
        ((DataRowView)veranstalterView.CurrentItem).Row.EndEdit( );

        // Ueberpruefung der Constraints wieder zuschalten
        ...
        // NavBar neu initialisieren
        veranstalterNavBar.SourceView = (CollectionView)veranstalterView

        // Buttons nach Bearbeitung sperren/ freigeben
        ...
    }
    catch(Exception ex)
    {
        ShowErrorMessages(...);
    }
}
```

6.2.5 Das Schließen des Fensters

Als letzte Aufgabe im ManageVeranstalterWindow sind nur noch das Klick-Ereignis auf den Schließen-Button sowie das Window_Closing Event abzufangen.

☞ *Generieren Sie die Handler Methode btnSchliessen_Click(...).*

In der Handler Methode btnSchliessen_Click(...) ist nur die Close() Methode des Forms aufzurufen.

Beim Schließen sollten aber alle bisher noch nicht gespeicherten Daten im Dataset aktualisiert werden. Dies bewerkstelligt man in der Window_Closing Handler Methode.

☞ *Fangen Sie den Window_Closing Event in einer Handler Methode ab und betrachten Sie das Klassendiagramm.*

Damit alle Daten im Dataset eingetragen werden, ist beim Schließen des Fensters die UpdateSources() Methode der Binding-Group aufzurufen.

Der abschließende catch-Block leitet alle Exceptions an ShowErrorMessages(...) weiter.

```
// Schliessen des Dialogs
private void Window_Closing (...)
{
    try
    {
        // aktuelle Bearbeitung beenden
        veranstalterBindingGroup.UpdateSources();
    }
    catch(Exception ex)
    {
        // Fehler abfangen und ausgeben
        this.ShowErrorMessages(ex, new RoutedEventArgs( ));
    }
}
```

7 Neue Kategorien erstellen (UC 04.10.1)

Der Geschäftsführer der FunEvents GmbH, Herr Fringsen, teilt alle Events zur späteren Auswertung und für gezielte Kundenwerbung in Kategorien ein. Analog zum Verwalten der Veranstalterdaten verläuft das Pflegen und Erstellen neuer Kategorien im EMS.

7.1 Das Erstellen des ManageKategorieWindows

Zur Verwaltung der Kategorien ist dem Projekt wieder ein neues WPF Fenster namens ManageKategorieWindow hinzuzufügen.

☛ Erstellen Sie im Projekt ManageEventsApp das neue WPF Fenster ManageKategorieWindow (Projekt/Fenster hinzufügen...).

☛ Setzen Sie die Size des Windows auf 480, 320 sowie die ResizeMode Property auf „NoResize“.

☛ Als Title Property kann „Bearbeitung der Eventkategorien“ eingegeben werden.



7.1.1 Die Steuerelemente des ManageKategorieWindows

Nach dem Erstellen des Fensters sollen die Steuerelemente im XAML Editor des Visual Studios angepasst werden. Für die Navigationsleiste kommt erneut die NavBar zum Einsatz.

☛ Fügen Sie dem Fenster im XAML Code den Verweis auf den NavigationBar Namespace hinzu!

Die Steuerelemente liegen in zwei Grids. Grid 1 (kategorieGrid) umfasst die Daten, darunter liegt die NavBar. Grid 2 (buttonGrid) beinhaltet die Buttons.

Damit Steuerelemente über mehrere Zeilen oder Spalten eines Grids gelagert werden können, verwendet man die XAML Properties Grid.RowSpan und Grid.ColumnSpan.

☛ Erstellen Sie die Grids und Steuerelemente entsprechend der Vorgaben aus den folgenden Tabellen im XAML Editor!

☛ Achten Sie auch auf die DataBindings an die Spalten der Veranstaltertabelle aus dem Dataset.

```
<Grid x:Name="kategorieGrid" Margin="10,10,10,120" >
  <Grid.RowDefinitions>
    <RowDefinition Height="2*" />
    ...
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    ...
  </Grid.ColumnDefinitions>
  <Label VerticalAlignment="Bottom" Grid.Row="0" Margin="5,0,0,0" Grid.Column="1"
    Content="Kategoriebezeichnung" />
  <TextBox x:Name="ek_KatBezeichnungTextBox" Grid.Row="1" Margin="5" Grid.Column="1" Text=
    "{Binding ek_KatBezeichnung, Mode=TwoWay, NotifyOnValidationError=true, ValidatesOnExceptions=true}" />
  ...
</Grid>
```

Tabelle Benutzerschnittstelle

Steuerelement	Objektbezeichnung	Properties
Grid (Datenquellen)	kategorieGrid	vier Zeilen, zwei Spalten
ListBox (Datenquellen)	ek_KatBezeichnungListBox	Grid.RowSpan="3" IsSynchronizedWithCurrentItem ="True"
Label (Datenquellen)		Kategoriebezeichnung
TextBox (Datenquellen)	ek_KatBezeichnungTextBox	-
Grid	buttonGrid	
Button	btnNeueKategorie	_neue Kategorie
Button	btnUebernehmen	_Übernehmen
Button	btnAbbrechen	Abb_rechen
Button	btnSchliessen	_Schließen
NavBar	kategorieNavBar	

☛ Setzen Sie die *IsEnabled* Properties der Buttons *btnAbbrechen* und *btnUebernehmen* auf *false*.

☛ Fügen Sie dem Klassendiagramm zur *ManageEventsApp* die neue Klasse *ManageKategorieWindow* hinzu.

7.1.2 Die Anzeige der Datensätze

Für die Anzeige der Kategorien benötigt das Fenster wieder eine Kopie des Datasets und eine *CollectionView*, welche mit der Kategorietabelle zusammen arbeitet.

☛ Deklarieren Sie das Dataset *dsKategorie* und die *CollectionView* *kategorieView* im *ManageKategorieWindow*.

Das Dataset und die View sind wie beim *ManageVeranstalterWindow* als private Element der Klasse *ManageKategorieForm* zu deklarieren.

```
// Dataset und View zur Bearbeitung der Kategorien
private ManageEvents.ManageEventsDataset dsKategorie;
private CollectionView ...
```

Auch die Anzeige der Datensätze im Navigationsbereich erfolgt analog zur Veranstalterbearbeitung. Die *NavBar* Control wird mit der *CollectionView* initialisiert, bevor das Fenster gestartet wird.

☛ Implementieren Sie die Methode *public void SetDataSet(...)*, welche als Parameter ein *ManageEventsDataset* sowie einen integer index übernimmt.

Der Verweis auf das Dataset sowie die im *ManageEventsWindow* gewählte Kategorie setzt man im nächsten Schritt durch die Methode *SetDataSet(...)*. Ebenso wird hier die Kategorietabelle an die *kategorieView* gebunden. Die Bindung der *NavBar* an die Datenquelle setzt man über die *kategorieView*. Die Methode übernimmt dazu als Parameter das *ManageEventsDataset* sowie den Index der gerade ausgewählten Eventkategorie.

```
public void SetDataSet(ManageEvents.ManageEventsDataset ds,
                      int index)
{
    // Dataset initialisieren
    dsKategorie = ds;

    //Databindings setzen
    kategorieGrid.DataContext = dsKategorie.tbl_EvKategorie;

    // View und NavBar initialisieren
    kategorieView =
        (CollectionView)CollectionViewSource.GetDefaultView(
            dsKategorie.tbl_EvKategorie);

    kategorieNavBar. ...

    // auf gewaehlte Kategorie aus ManageEventWindow springen
    kategorieView.MoveCurrentToPosition(index);
}
```

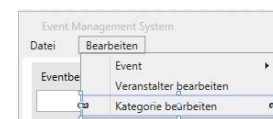

- ☞ Initialisieren Sie das Dataset *dsKategorie* und die *DataView* *kategorieView* in der Methode.
- ☞ Binden Sie den *DataContext* des *kategorieGrids* an die *tbl_Kategorie*.
- ☞ Initialisieren Sie die *NavBar* mit der *kategorieView*.
- ☞ Setzen Sie die *Position Property* der *View* auf den übergebenen *Index*.
- ☞ Betrachten Sie das *Klassendiagramm* des *ManageKategorieWindows*.

7.1.3 Letzte Änderungen im ManageEventsWindow

Zum Starten der Kategoriebearbeitung ist auf der grafischen Benutzerschnittstelle des *ManageEventsWindows* bereits ein Button vorhanden.

Die Bearbeitung der Kategorien soll auch über einen Menüpunkt durch den User aufgerufen werden können. Lediglich ein weiterer Menüpunkt ist zu ergänzen.

- ☞ Der zugehörige Menüpunkt *kategorieBearbeitenMenuItem* ist unter *Bearbeiten* einzufügen.



Für den Start der Kategoriebearbeitung ist in einem letzten Schritt nur noch das Click-Event auf den Button *btnKategorie* abzufangen.

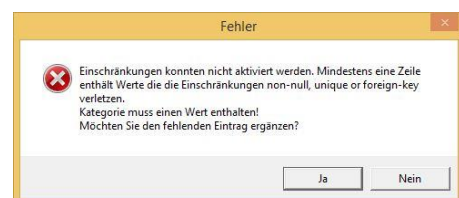
- ☞ Generieren Sie mit Hilfe des *Visual Studios* die neue Handler Methode *btnKategorie_Click(...)*.
- ☞ Implementieren Sie die Methode analog zur Methode *btnVeranstalter_Click(...)* aus dem vorhergehenden Kapitel.
- ☞ Debuggen und testen Sie die Anzeige der Kategorien im EMS.
- ☞ Verbinden Sie auch das Klick-Ereignis auf den Menüpunkt *kategorieBearbeitenMenuItem* mit der Handler Methode *btnKategorie_Click(...)*.

7.2 Das Bearbeiten der Kategorien

7.2.1 Die Behandlung der Fehlermeldungen

Bei der Eingabe der Events darf das Feld *ek_KategorieTextBox* nicht leer bleiben, damit beim Zuschalten der Constraints im Dataset keine Exceptions auftreten. Analog zum *ManageVeranstalterWindow* empfiehlt sich eine einheitliche Behandlung der Fehler über die neue Methode *ShowErrorMessage(...)*.

- ☞ Implementieren Sie die Methode *void ShowErrorMessage(object sender, RoutingEventArgs e)* im *ManageKategorieWindow* analog zur gleichlautenden Methode des *ManageVeranstalterWindows*.



7.2.2 Das Anlegen neuer Kategorien

Zum Anlegen neuer Kategorien hat der Benutzer auf den Button *btnNeueKategorie* zu klicken. Mit Abbrechen oder Übernehmen können die Daten nach der Eingabe dann verworfen oder in das Dataset gespeichert werden.

- ☞ Erzeugen Sie die neue Handler Methode *btnNeueKategorie_Click(...)*.

- ☞ Sperren Sie die Steuerelemente zur Navigation, zum Anlegen einer neuen Kategorie und zum Schließen des Forms. Die Buttons Abbrechen und Übernehmen sind freizugeben.
- ☞ Erstellen Sie in einem try-Block eine neue DataRow und schalten Sie die Überprüfung der Constraints im Dataset ab.
- ☞ Fügen Sie die neue DataRow der Kategorietabelle hinzu und führen Sie ein Reset der Anzeige durch (Sprung auf den letzten Datensatz).
- ☞ Beginnen Sie den Editvorgang der Row. Der Focus ist auf das Textfeld ek_KategorieTextBox zu setzen.
- ☞ Leiten Sie aufgetretene Exceptions an die Methode ShowErrorMessage(...) weiter.
- ☞ Betrachten Sie das Klassendiagramm des ManageKategorieWindows.

7.2.3 Das Abbrechen der Bearbeitung

Zum Abbrechen des aktuellen Bearbeitungsvorganges betätigt der User den entsprechenden Button auf dem ManageKategorieWindow.

- ☞ Fangen Sie den Click-Event auf den Button btnAbbrechen in einer neuen Handler Methode ab.
- ☞ Die Implementierung der Handler Methode erfolgt analog zur gleich lautenden Methode des ManageVeranstalterWindows.

7.2.4 Die Übernahme der Daten

Zur Übernahme der neuen Kategoriedaten klickt der Benutzer auf den Button btnUebernehmen. In einer Handler Methode ist dieses Click-Event abzufangen. Als Anhaltspunkt zur Implementierung dient wieder die gleichnamige Methode des ManageVeranstalterWindows.

- ☞ Erstellen Sie die Handler Methode btnUebernehmen_Click(...) mit dem Visual Studio.
- ☞ Beenden Sie in einem try-Block die Bearbeitung der DataRow und schalten Sie die Überprüfung der Constraints im Dataset wieder ein.
- ☞ Initialisieren Sie die kategorieNavBar erneut.
- ☞ Sperren bzw. geben Sie die entsprechenden Buttons des Windows frei.
- ☞ Im catch-Block ist die Methode ShowErrorMessage(...) zu verwenden.
- ☞ Betrachten Sie auch das Klassendiagramm des ManageKategorieWindows.

7.2.5 Das Schließen des Fensters

Zum Schließen des Forms sind zuletzt noch eine Handler Methode btnSchliessen_Click(...) und Window_Closing(...) zu erstellen.

- ☞ Generieren Sie die Handler Methoden btnSchliessen_Click(...) und Window_Closing(...).
- ☞ Im try – Block ist die Bearbeitung der Kategorien zu beenden.
- ☞ Exceptions reicht man an die ShowErrorMessage(...) Methode weiter.
- ☞ Ergänzen Sie das Klassendiagramm und kopieren Sie die neue Version in das vorhandene Visio Projekt.
- ☞ Kompilieren und Testen Sie die fertige Version 2.0 des Event Management Systems EMS.