Business App Leitfaden für Anpassungen

202103 — Letzte Änderung: 1 June 2022

GEDYS IntraWare GmbH

Inhaltsverzeichnis

1. Einleitung zur Version 2.0 (202103)

Programmatische Anpassungen werden in BusinessApp über eine Solution und ein oder mehrere Projekten innerhalb von Microsoft Visual Studio ab Version 2017 ermöglicht. Voraussetzungen dafür sind

- Gültige Lizenz und entsprechende Lizenzdatei
- Zugang zu dem GEDYS IntraWare Nuget Server

Die Module von Business App und Business App CRM stehen als Nuget-Pakete in den neusten Versionen über einen entsprechenden Server zur Verfügung. Mit Hilfe von C#, .Net, TypeScript und den Technologien von DevExpress wie den MVC Controls und den eXpress Persistent Objects (XPO), kann die gewünschte Business Logik implementiert werden. Im Folgenden werden diese Möglichkeiten beschrieben.

Version 1.0 – 202011

1.1. Konfiguration vs. Programmierung

Die Anlage von Übersetzungen und Auswahllisten, sowie die Definitionen von Datentabellen und Relationen sind sowohl konfigurativ als auch programmatisch möglich. Programmatisch ist es auch möglich mit den konfigurativ angelegten Elementen umzugehen. Daher ist es für Anpassungen nicht relevant, wie diese Teile angelegt werden.

Basiert man programmatisch auf Konfigurationen, sollte man sicherstellen, dass diese in das Modul (Projekt) eingebunden sind.

Ist für die Anwendung die Existenz von bestimmten Datensätzen notwendig, kann dies beim Anwendungsstart sichergestellt werden. Insbesondere trifft dies bei Rollen zu, die zur Ausführung von Funkationalitäten notwendig sind, falls diese fest codiert worden sind.

> Ist zu Beginn eines Projektes schon klar, dass Programmierungen vorgenommen werden, wird Empfohlen die Elemente programmatisch anzulegen.

1.2. Voraussetzungen

Know How

- BA Konfigurations Know How (Fortgeschritten)
- C# .Net (Fortgeschritten)
- LINQ (Grundkenntnisse)
- TypeScript (Grundkenntnisse)
- MVC Webentwicklung (Fortgeschritten)
- IIS (Grundkenntnisse)
- SQL (Grundkentnisse)

Technisch / Lizenzen

- MS Visual Studio (C# .Net Enwticklungsumgebung) (Empfohlen 2017 oder 2019)
- .Net 4.8 (Developer Pack)
- DevExpress ASP.MVC und XPO (Produkt ASP.Net) Version 20.1.8
- MS SQL Server (Empfohlen 2016 oder 2019)
- IIS (Lokal oder Server). Bei Server Installation:
  - Web Deploy
  - Visual Studio Remote Debugger

1.3. Business App Module

Dieser Abschnitt beschreibt den Inhalt der verschiedenen Business App Module und deren Abhängigkeiten untereinander. Die Module stehen als Nuget Pakete zur Verfügung und können beliebig kombiniert werden, die Abhängigkeiten müssen dabei beachtet werden. Grundsätzlich kann man immer weitere Module hinzufügen. Das Entfernen von Modulen aus einer Anwendung ist aber nicht möglich!

BA Module

Die verschiedenen BA Module stellen Funktionalitäten bereit, welche in Summe das Bussines App Produkt ergeben.

BA.Core

Das Basismodul wird grundsätzlich zur Ausführung einer BA Anwendung benötigt und enthält das eigentliche Business App Framework.

Datentabellen

- Basis.Rolle
- Benutzerprofil
- Verzeichnisrolle
- Basis.Protokoll
- Anwendungsprotokoll
- Ordner

Teildatentabellen

- Anschriften
- E-Mail-Adressen von Adressen
- E-Mail-Adressen von E-Mails

BA.Designer

Dieses Modul beinhaltet den Designer zur Konfiguration der Anwendung. Es wird benötigt, wenn in der Anwendung Konfigurationen von Datentabellen, Masken, Ansichten, etc. vorgenommen werden. Falls die Konfiguration durch eine ZIP Datei eingespielt wird, ist dieses Modul nicht notwendig.

Voraussetzung

- BA.Core

BA.Contact

Beinhaltet

- Firmen- und Kontakttabelle

- Erweiterung des Benutzerprofils
- Quelldatensätze (firstFoundAddress, firstFoundContact und firstFoundCompany) für die Platzhalterersetzung
- Berechnete Eigenschaften zum Aufbau der Anschrift und der Anreden
- Standardsteuerelemente zur Anzeige und Bearbeitung von Anschriften und Namensblöcken

Voraussetzung

- BA.Core

BA.Activity

Beinhaltet

- Vorgänge
- Einzel-E-Mail
- Serien-E-Mail
- E-Mail-Vorlagen
- Benachrichtigung

Vorraussetzung

- BA.Core

BA.Correspondence

Beinhaltet

- Einzelbriefe
- Serienbriefe
- Brief- und Tabellenkalkulationsvorlage

Vorraussetzung

- BA.Core
- BA.Activity

BA.Businessmail

Dieses Modul beinhaltet die Funktionalität, um in Outlook E-Mails nach Business App zu dokumentieren.

Voraussetzung

- BA.Core
- BA.Contact
- BA.Activity

BA.Report

Dieses Module beinhaltet die Elemente, um Reports zu erstellen und auszuführen.

Vorraussetzung

- BA.Core

BA.Dashboard

Dieses Module beinhaltet die Elemente, um Dashboards zu erstellen und auszuführen.

Vorraussetzung

- BA.Core

BA.Appointment

Dieses Modul beinhaltet die Komponenten für Termine-. und deren Synchronisation mit einem Exchange Server.

Voraussetzung

- BA.Core

BA.FollowUp

Dieses Modul beinhaltet die Komponenten für Wiedervorlagen.

Voraussetzung

- BA.Core
- BA.Contact
- BA.Activity

BA.Task

Dieses Modul beinhaltet die rudimentären Komponenten für Aufgaben. Es wird als einziges Modul nicht in den Produkten ausgeliefert und wird zukünftig komplett tüberarbeitet. Die Verwendung wird nicht empfohlen.

Voraussetzung

- BA.Core

BA.CRM Module

Die BA.CRM Module erweitern die Core Funktionalitäten, um spezielle CRM spezifische Funktionen und

fassen alle Module zu einer Anwendung inkl. Konfiguration zusammen.

BA.CRM.Contact

Beinhaltet

- Aufteilung der Firmen- und Kontakttabellen in Basis.Firma / Firma / Firmenprofil und Basis.Kontakt / Kontakt
- Funktionalität der Geschäftsbeziehungen
- Betreuer-Relationen und Funktionalitäten
- Massenaktionen zum Setzen von Tags, Verteilern und des Betreuers
- Rollen zum Erstellen, Bearbeiten und Löschen

Voraussetzung

- BA.Core
- BA.Contact

BA.CRM.Activity

Beinhaltet

- Aktion um den Betreuer als Empfänger einzutragen
- Erweiterung der Serien-E-Mail, um die Möglichkeit den Betreuer als Absender zu verwenden.
- Automatisches setzen des Vorgangseigentümers
- Funktionalität „Weitere Beteiligte“
- Automatisches Setzen von Werten (Bspw. Ersteller, Vorgangseigentümer, E-Mail-Typen, u.v.m.)
- Rollen zum Erstellen, Bearbeiten und Löschen

Voraussetzung

- BA.Core
- BA.Contact
- BA.Activity
- BA.CRM.Contact

BA.CRM.Correspondence

Beinhaltet

- Weitere Sortieroptionen für den Serienbrief (Firma / Nachname, Nachname, Betreuer, Betreuer / Land)
- Automatisches Setzen von Werten (Bspw. Ersteller, Vorgangseigentümer, u.v.m.)

Voraussetzung

- BA.Core
- BA.Contact

- BA.Activity
- BA.Correspondence
- BA.CRM.Contact
- BA.CRM.Activity

BA.CRM.Opportunity

Beinhaltet

- Vollständige Funktionalität zum Anlegen und Verwalten von Verkaufschancen

Voraussetzung

- BA.Core
- BA.Contact
- BA.Activity
- BA.Correspondence
- BA.CRM.Contact

BA.CRM.FollowUp

Beinhaltet

- Setzen des Betreffs
- Setzen eines Verantwortlichen (inkl. Autorenrechte).
- Funktionalität zum Setzen des Betreuers als Aktion in der Maske

Voraussetzung

- BA.Core
- BA.Contact
- BA.Activity
- BA.FollowUp
- BA.CRM.Contact

BA.CRM

Beinhaltet

- Die Konfiguration der BA.CRM Anwendung

Voraussetzung

- BA.Core
- BA.Contact
- BA.Activity
- BA.Designer
- BA.Report

- BA.BusinessMail
- BA.Correspondence
- BA.CRM.Contact
- BA.CRM.Activity
- BA.CRM.Correspondence
- BA.CRM.Opportunity

1.4. Änderungen zur vorherigen Version

Vorherige Version 1.0 – 202011

Erweiterung der Dokumentation

Dynamische Konfiguration

In dem neuen Kapitel Dynamische Konfigurationen werden die Möglichkeiten der dynamischen Konfiguration erläutert. In diesem Zusammenhang wurde ein Hinweis in Maskensteuerelemente für Dialoge bei der Ansicht im Dialog eingefügt und in dem Kapitel Eigener Datenprovider für Ansichten wurde erläutert, wie dieser verwendet wird.

Datenprovider Modifiers

Bei den Datenprovidern für generiete Masken wurde ein neues Kapitel zur Erläuterung der Modifiers hinzugefügt.

Relationskategorie bei Leser-Relation

Der Relationstyp für die Leser hat jetzt analog zur Autorenrelation eine Relationskategorie (siehe). Diese kann bei Bedarf in Projekten erweitert werden.
Alle bestehenden Relationen wurden auf die Kategorie `Default` umgestellt. Auch die Relationsdefinition im Core ist entsprechend umgestellt, und damit auch alle Steuerlemente, die sie verwenden.

Wenn über Programmcode auf die Leserelationen zugeriffen werden soll, ist zu entscheiden, ob man alle Relationskategorien benötigt oder nur Standard. Bei Schreibzugriffen ist eine Relationskategorie jetzt obligatorisch.

Beispiel:

Vorher

```
List<Guid> readers = orm.GetSourcesOids(EnumRelationType.Reader, null);
orm.AddSource(newReader, EnumRelationType.Reader, null);
```

Jetzt

```
List<Guid> readers = orm.GetSourcesOids(EnumRelationType.Reader, EnumReaderRel
ationSubTypes.DefaultGuid);
orm.AddSource(newReader, EnumRelationType.Reader, EnumReaderRelationSubTypes.D
efaultGuid);
```

2. Neues Projekt

2.1. Vorlage

Um ein neues Projekt zu erstellen, kann diese Vorlage verwendet werden.

Download Vorlage Version 2.0

Sie sollte unter “\Documents\Visual Studio 2019\Templates\ProjectTemplates” abgelegt werden. Anschließend erstellt man ein Neues Projekt und sucht nach der Vorlage “BA.Template”. Stellen Sie sicher das vorher die Verbindung zum Nuget-Server eingerichtet ist und Benutzername und Passwort abgespeichert (siehe unten) sind und platzieren Sie die Projektmappe und das Projekt nicht in das gleiche Verzeichnis.

Die Vorlage beinhaltet BA.Core. Für das eigene Projekt kann man nun alle weiteren notwendigen Business App Module über den Nuget-Server

https://app.business-app.com:8443/nuget

installieren. Der Nuget-Server ist Passwort geschützt. Falls Sie noch keinen Zugang haben, fragen Sie Ihren zuständigen Service bzw. Vertriebsmitarbeiter von GEDYS IntraWare.

Stellen Sie sicher, dass die Projektmappenplattform auf “Any CPU” steht.

2.2. Veröffentlichen

Richten Sie ein entsprechendes Veröffentlichungsprofil für Ihre Umgebung ein. Wenn der App-Pool die notwendigen Rechte hat, wird die Datenbank automatisch erstellt.

Startet die Anwendung zum ersten Mal existiert noch kein Benutzer. Dieser wird über den Aufruf des Controllers “/Account/CreateDefaultUser” im Browser erzeugt werden. Der erzeugte Benutzer sollte unbedingt anschließend auf einen sinnvollen Anwendernamen und -passwort umgestellt werden.

Beispiel für ein Veröffentlichungsprofil mit Web-Deploy. Folgende Stellen müssen angepasst werden

- SiteUrlToLaunchAfterPublish
- MSDeployServiceURL
- DeployIisAppPath
- PublishDatabaseSettings\ObjectGroup\Destination
- PublishDatabaseSettings\ObjectGroup\Object\PreSource
- PublishDatabaseSettings\ObjectGroup\UpdateFrom\Source
- ItemGroup\MSDeployParameterValue\ParameterValue

```
<?xml version="1.0" encoding="utf-8"?>
<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbu
ild/2003">
  <PropertyGroup>
    <WebPublishMethod>MSDeploy</WebPublishMethod>
    <ADUsesOwinOrOpenIdConnect>False</ADUsesOwinOrOpenIdConnect>
    <LastUsedBuildConfiguration>Debug</LastUsedBuildConfiguration>
    <LastUsedPlatform>Any CPU</LastUsedPlatform>
    <SiteUrlToLaunchAfterPublish>http://server.company.de/ba_training</SiteUrl
ToLaunchAfterPublish>
    <LaunchSiteAfterPublish>True</LaunchSiteAfterPublish>
    <ExcludeApp_Data>False</ExcludeApp_Data>
    <MSDeployServiceURL>http://server.company.de</MSDeployServiceURL>
    <DeployIisAppPath>Default Web Site/ba_training</DeployIisAppPath>
    <RemoteSitePhysicalPath />
    <SkipExtraFilesOnServer>False</SkipExtraFilesOnServer>
    <MSDeployPublishMethod>RemoteAgent</MSDeployPublishMethod>
    <EnableMSDeployBackup>True</EnableMSDeployBackup>
    <UserName></UserName>
    <PublishDatabaseSettings>
      <Objects xmlns="">
        <ObjectGroup Name="DefaultConnection" Order="1" Enabled="False">
          <Destination Path="Data Source=SQLServer;Initial Catalog=ba_trainin
g;Integrated Security=True" Name="data source=SQLServer;integrated security=SS
PI;initial catalog=ba_training" />
          <Object Type="DbDacFx">
            <PreSource Path="Data Source=.\;Initial Catalog=ba_training;Integr
```

```
ated Security=True" includeData="False" />
            <Source Path="$(IntermediateOutputPath)AutoScripts\DefaultConnecti
on_IncrementalSchemaOnly.dacpac" dacpacAction="Deploy" />
          </Object>
          <UpdateFrom Type="Web.Config">
            <Source MatchValue="data source=.\;integrated security=SSPI;initia
l catalog=ba_training" MatchAttributes="$(UpdateFromConnectionStringAttribute
s)" />
          </UpdateFrom>
        </ObjectGroup>
      </Objects>
    </PublishDatabaseSettings>
  </PropertyGroup>
  <PropertyGroup>
    <UseMsDeployExe>true</UseMsDeployExe>
  </PropertyGroup>
  <Target Name="SetupCustomAcls" AfterTargets="AddIisSettingAndFileContentsToS
ourceManifest" Condition="'$(ExcludeApp_Data)' == 'False'">
    <Message Text="Setting ACL on App_Data" />
    <ItemGroup>
      <MsDeploySourceManifest Include="setAcl">
        <Path>$(_MSDeployDirPath_FullPath)\App_Data</Path>
        <setAclAccess>Read,Write,Modify</setAclAccess>
        <setAclResourceType>Directory</setAclResourceType>
        <AdditionalProviderSettings>setAclResourceType;setAclAccess</Additiona
lProviderSettings>
      </MsDeploySourceManifest>
    </ItemGroup>
  </Target>
  <Target Name="SetupCustomSkipRulesForAppData" AfterTargets="AddIisSettingAnd
FileContentsToSourceManifest">
    <Message Text="Setting Skiprules for App_Data" />
    <ItemGroup>
      <MsDeploySkipRules Include="SkipFilesInAppDataFolder">
        <SkipAction>Delete</SkipAction>
        <ObjectName>filePath</ObjectName>
        <AbsolutePath>$(_DestinationContentPath)\\App_Data\\.*</AbsolutePath>
      </MsDeploySkipRules>
      <MsDeploySkipRules Include="SkipFoldersInAppDataFolders">
        <SkipAction>Delete</SkipAction>
        <ObjectName>dirPath</ObjectName>
        <AbsolutePath>$(_DestinationContentPath)\\App_Data\\.*\\*</AbsolutePat
h>
      </MsDeploySkipRules>
    </ItemGroup>
  </Target>
  <ItemGroup>
```

```
    <MSDeployParameterValue Include="DefaultConnection-Web.config Connection S
tring">
      <ParameterValue>data source=SQLServer;integrated security=SSPI;initial c
atalog=ba_training</ParameterValue>
    </MSDeployParameterValue>
    <MsDeploySkipRules Include="CustomSkipFile">
      <ObjectName>filePath</ObjectName>
      <AbsolutePath>Customer.config</AbsolutePath>
    </MsDeploySkipRules>
  </ItemGroup>
</Project>
```

3. Mehrsprachigkeit

Die Anwendung ist grundsätzlich mehrsprachig. Bei der Implementierung eigener Funktionalitäten sollte diese Mehrsprachigkeit ebenfalls berücksichtigt werden, damit auch diese Teile in verschiedenen Sprachen genutzt werden können. Dazu stehen sowohl auf der Server- als auch auf der Clientseite entsprechende APis zur Verfügung.

Sie können eigene Texte definieren und vorhandene überschreiben.

3.1. Übersetzungen definieren

Übersetzungen werden in einem Entwicklungsprojekt in einer CSV Datei abgelegt. Diese kann sowohl neue Texte definieren als auch bestehende überladen. Dazu wird eine Textdatei namens “Translations.csv” in den Ordner “Content” gelegt. Diese Datei muss in den dateieigenschaften als “Eingebettete Ressource” definiert sein.

Der Header `Oid;Language_de;Language_en;Description;Category;Type;UseInJS;TranslationStatus` muss zuoberst in der CSV Datei vorhanden sein. Die Angabe eines Trennzeichens in der ersten Zeile (noch vor dem Header) ist optional, das Standardtrennzeichen ist das Semikolon.

Spalte	Beschreibung
Oid	Eine eindeutige Guid. Für eine neue Übersetzung sollte eine Guid generiert werden. Falls eine bestehende Übersetzung geändert werden soll, muss die Guid dieses Textes eingetragen werden. Groß-/Kleinschreibung ist in dieser Spalte irrelevant.
Language_de Language_en Language_XX	Mehrere Spalten mit Übersetzungen beliebiger Länge. Dabei ist der Sprachcode mit einem Unterstrich von dem Begriff “Language” getrennt.
Description	Eine kurze Beschreibung von maximal 100 Zeichen. Beliebig zu belegen.
Category	Eine Kategorie von maximal 100 Zeichen. Beliebig zu belegen.
Type	Eine Typangabe von maximal 100 Zeichen. Beliebig zu belegen.
UseInJS	Flag („True“ oder „False“) für die Optimierung der Zugriffe im Frontend. Siehe Frontend API
TranslationStatus	Muss auf “0” gesetzt werden.

Beispiel: Translation.csv

```
sep=;
Oid;Language_de;Language_en;Description;Category;Type;UseInJS;TranslationStatus
"4401ff39-77b9-41c9-8b9f-978304f52f95";"Neue Übersetzung";"New translation";"";"Projekt XYX";"Label";"False";"0"
"e5b09032-d8e2-4242-b6d1-9c9130a01c6d";"Passwörter stimmen überhaupt nicht überein.";"Passwords do not match at all.";"";"";"";"False";"0"
```

3.2. Backend-API

Die Umwandlung von Guids in anzeigbaren Text erfolgt über verschiedene Methoden.

String.Translate()

Diese Erweiterungsmethode auf der String Klasse ermöglicht in kurzer Schreibweise eine schnelle Übersetzung in die Benutzersprache des aktuellen Anwenders.

```
"8ce542e4-9eb1-43bf-a227-60aba99916d9".Translate()
```

Bei der Angabe von Argumenten wird wie bei `Format(String format, params object[] args)` werden Teile der Übersetzung mit variablen Bestandteilen gefüllt.

```
"8ce542e4-9eb1-43bf-a227-60aba99916d9".Translate(123, "Text")
```

Api.Text

Mit dieser Methode ist es möglich eine Guid zu übersetzen. Mit dem optionalen Parameter Language ist die Übersetzung in eine beliebige vorhandene Sprache möglich.

```
Guid guid = "8ce542e4-9eb1-43bf-a227-60aba99916d9".ToGuid();
Api.Text.Translate(guid, "de");
```

Mit diesen beiden Methoden kann ein String der Guids und andere Textbestandteile hat übersetzt werden

```
Api.Text.Format(String format, params Object[] args)
Api.Text.FormatInLanguage(String format, String languageCode, params Object[]
args)

Api.Text.Format("Dieser Text enthält 8ce542e4-9eb1-43bf-a227-60aba99916d9 Best
andteile", teil1, teil2)
```

3.3. Frontend-API

Für das Benutzen von Übersetzungen im Frontend steht die TypeScript Klasse `BA.Ui.TranslationsTranslationTools` zur Verfügung. Jede Übersetzung wird nur einmal vom Server abgerufen. Jede Übersetzung welche in der Translations.csv mit UseInJS="True" definiert ist, wird schon beim Laden der Anwendung mit geladen. Alle anderen Übersetzungen können ebenfalls genutzt werden, sie werden dann bei der ersten Verwendung geladen.

Die wichtigsten Frontend-Methoden

Holen von einer oder von mehreren Übersetzungen

```
public static GetTranslation(translationGuid: string, callback)
public static GetTranslations(translationGuids: string[], callback)
```

- translationGuid oder translationGuids: Eine oder meherere Guids der Übersetzungen
- callback: Callback Funktion. Sie bekommt entweder einen einzelnen String mit der Übersetzung oder eine Map mit der Guid als Key übergeben.

Beispiele

```
BA.Ui.Translations.TranslationTools.GetTranslation("965B169D-2C8A-409E-AF6B-350F454769D0", BA.Ui.Toast.Error);
```

```
BA.Ui.Translations.TranslationTools.GetTranslation("5A419213-8EB5-486C-9B77-3545DB6746D0", function (result) {
    BA.Ui.Toast.Error(result);
});
```

```
var texts = [];
var titleGuid = "1E7E83B0-78A3-45E2-A0BD-6241A8038624";
var messageGuid = "EE33B1F1-16D4-438D-A1A0-495BF761FF7D";
texts.push(titleGuid);
texts.push(messageGuid);
BA.Ui.Translations.TranslationTools.GetTranslations(texts, function (results) {
    BA.Ui.MessageBox.ShowYesNo(results[titleGuid], results[messageGuid], 0, function (result) {
        // Do something
    });
}, false);
```

4. Auswahllisten

Auswahllisten sind eine zentrale Technologie, die häufig auch zur technischen Parametrisierung verwendet wird. Es ist möglich neue Auswahllisten zu erstellen und vorhandene zu erweitern. Auswahllisten können weitere Eigenschaften haben. Beispielsweise hat die Liste “Countries” für jedes Land auch den ISO Code hinterlegt.

Die gewählten Werte können entweder per Referenz oder als kommaseparierter String in der Datentabelle abgelegt werden. Vor- und Nachteile sind hier beschrieben.

Werden die Werte als Referenz gespeichert, so werden die Referenzen redundant gespeichert, um für verschiedene Situationen einen optimalen Zugriff zu ermöglichen. Diese redundante Speicherung der Referenzen erfolgt bei allen Auswahlwertfeldern, da zurzeit bei der Datendefinition kein Unterschied zwischen bei Einzelauswahl und Mehrfachauswahl gemacht wird.

In der Regel wird der Umgang mit diesen Redundanzen durch die API Methoden in den unterschiedlichen Teilen gekapselt. Aber als Entwickler sollte man sich dieser Situation bewusst sein. Im Folgenden werden die drei redundanten Speichervarianten der Referenz erläutert.

Einzelwert

In der Datenspalte mit dem definierten Feldwert wird die Oid des Auswahlwertes abgespeichert. In der Programmierung hat man direkten Zugriff auf das Objekt des Wertes, wenn der Wert abgerufen wird.

Werden mehrere Werte ausgewählt, steht in diesem Feld ein zufälliger Wert aus der gewählten Liste und sollte daher nicht verwendet werden.

Mehrfachwert Spalte “_AllValues”

In der Datenspalte FeldName_AllValues wird eine kommaseparierte Liste der Oids aller gewählten Werte als String abgelegt. Diese wird beispielsweise genutzt, um in Ansichten eine Spalte zu erhalten, die alle ausgewählten Werte in einer Zelle anzeigt.

Mehrfachwerte als Relationen

Zusätzlich werden alle gewählten Werte über `OrmRelation` als Relation angelegt. Diese BA Relationen werden beispielsweise für die Mehrfachgruppierung in Ansichten genutzt. Dabei wird die Relationstabelle wie folgt belegt.

- `OrmRelation.Source` Oid des Datensatzes
- `OrmRelation.SourceType` Guid der Datentabelle des Datensatzes
- `OrmRelation.Target` Oid des Auswahlwertes
- `OrmRelation.RelationType` Oid der Auswahlliste
- `OrmRelation.RelationCategory` Feldname der Auswahllistein der Datentabelle

Alle weiteren Belegungen der Tabelle sind nicht relevant.

Diese Relationen werden für Teildatensätze nicht angelegt

4.1. Eigene erstellen

Definition

Mit Hilfe des Attributs `[EnumDefinition]` wird eine Klasse als Auswahliliste gekennzeichnet. Folgende Parameter sind notwendig:

- String name: Name der Auswahlliste
- Bool isLocked: Ist die Auswahlliste für die Bearbeitung gesperrt?

Folgende Parameter sind Optional (Mit „false“ vorbelegt):

- Bool copyRelation: Werden Relationen bei neuen Kindern Kopiert?
- Bool sortByText: Sollen Auswahllistenwerte nach deren Inhalt sortiert werden?
- Bool allowAddingValues: Dürfen neue Werte hinzugefügt werden?
- Bool visibleInManagament: Soll die Auswahlliste in der Verwaltung sichtbar sein?
- Bool visibleInOrmDesigner: Soll die Auswahlliste im Designer auswählbar sein?

Deklaration

Um eine Auswahlliste zu erzeugen muss die Klasse die abstrakte Klasse `ValueEnum<E>` erweitern. E ist hierbei die Referenz auf die neu eurzeugte Auswahlliste.

```
public class EnumYourName : ValueEnum<EnumYourName>
```

Definition von Werten

Bei der Definition von Werten müssen folgende Eigenschaften festgelegt werden

- String valueGuid: Eindeutige Guid
- int sortOrder: Eindeutige Sortierungsnummer
- String translationGuid: Guid der Übersetzung

Optional sind folgende Werte

- String image = null: Der Name eines Icons (Siehe)
- String color = null: Die Farbe des Wertes
- Boolean isActive = true: Ist das Aktiv-Flag gesetzt
- Boolean isLocked = false: Ist der Wert vor Änderungen in der Anwendung geschützt
- Boolean isDeletable = true: Kann der Wert gelöscht werden.

Erweiterte Properties

Den Werten können optional weitere Eigenschaften hinzugefügt werden. Mit Hilfe von Attributen kann die Darstellung in der UI beeinflusst werden. Siehe auch Generierte Masken.
In diesem Beispiel wird die Eigenschaft in der UI nicht dargestellt und ist damit nicht modifizierbar.

```
[Browsable(false)]
public string Property { get; set; }
```

Beispiel einfache Auswahlliste

```
[EnumDefinition("My Enum", false, false, true, true, true, true)]
public class EnumMyEnum : ValueEnum<EnumMyEnum>
{
    public const string Guid = "[INSERT ENUM GUID]";

    public const string FirstValueGuid = "[INSERT VALUE 1 GUID]";
    public const string SecondValueGuid = "[INSERT VALUE 2 GUID]";

    public static readonly EnumMyEnum FirstValue = new EnumMyEnum(FirstValueGu
id, 0, "[INSERT TRANSLATION 1 GUID]");
    public static readonly EnumMyEnum SecondValue = new EnumMyEnum(SecondValue
Guid, 1, "[INSERT TRANSLATION 2 GUID]");

    public EnumMyEnum(String valueGuid, int sortOrder, String translationGui
d) : base(valueGuid, sortOrder, translationGuid)
    { }

    public EnumMyEnum() { }
}
```

Beispiel 2

Diese Beispielliste implementiert zusätzlich Folgendes

- Eine eigene Eigenschaft
- Die Definition eines Icons pro Wert

```
[EnumDefinition("My Enum", false, false, true, true, true, true)]
public class EnumMyEnum : ValueEnum<EnumMyEnum>
{
    public const string Guid = "[INSERT ENUM GUID]";

    public const string FirstValueGuid = "[INSERT VALUE 1 GUID]";
    public const string SecondValueGuid = "[INSERT VALUE 2 GUID]";

    public static readonly EnumMyEnum FirstValue = new EnumMyEnum(FirstValueGu
id, 0, "[INSERT TRANSLATION 1 GUID]", "my value 1", "telephone2");
    public static readonly EnumMyEnum SecondValue = new EnumMyEnum(SecondValue
Guid, 1, "[INSERT TRANSLATION 2 GUID]", "my value 2", "information");
```

```
    [Browsable(false)]
    public string OwnProperty { get; set; }

    public EnumMyEnum(String valueGuid, int sortOrder, String translationGui
d, String ownProperty, String image = null) : base(valueGuid, sortOrder, trans
lationGuid, image)
    {
        OwnProperty = ownProperty;
    }

    public EnumMyEnum() { }
}
```

4.2. Bestehende erweitern

Um eine bestehende Auswahlliste zu erweitern benötigt man lediglich das Attribut `EnumExtension` über der erweiternden Klasse.

```
[EnumExtension(typeof(EnumDataSource))]
```

Danach können Values wie gewohnt hinzugefügt werden.

Beispiel

```
[EnumExtension(typeof(EnumDataSource))]
public static class EnumDataSourceExtension
{
    public const string DataSource1Guid = "[INSERT VALUE 1 GUID]";
    public const string DataSource2Guid = "[INSERT VALUE 2 GUID]";
    public const string DataSource3Guid = "[INSERT VALUE 3 GUID]";
    public static readonly EnumDataSource DataSource1 = new EnumDataSource(Dat
aSource1Guid, 200, "[INSERT TRANSLATION 1 GUID]");
    public static readonly EnumDataSource DataSource2 = new EnumDataSource(Dat
aSource2Guid, 200, "[INSERT TRANSLATION 2 GUID]");
    public static readonly EnumDataSource DataSource3 = new EnumDataSource(Dat
aSource3Guid, 200, "[INSERT TRANSLATION 3 GUID]");
}
```

4.3. Referenz vs. Werte

Es gibt zwei Arten wie die Werte einer Auswahlliste abgespeichert werden. Entweder über eine Referenz oder die Werte werden selbst abgespeichert.

Werte

Werden nur die Werte abgespeichert wird eine Komma separierte Liste der gewählten Werte in eine Spalte der SQL Tabelle abgelegt. Jede Refrenz auf den eigentlichen Wert geht verloren. Das bedeutet im Einzelnen:

- Die ausgewählten Werte werden in Ansichten, Formeln, etc. nur noch als ein Text behandelt
- Die ausgewählten Werte werden nicht in die Sprache des Anwenders übersetzt
- Änderungen an der Auswahlliste haben keinen Einfluss auf schon ausgewählte Werte
- Eine Mehrfachgruppierung in Ansichten ist nicht möglich.
- Filter in Ansichten arbeiten nur eingeschränkt.
- Wählen Anwender mit unterschiedlichen Sprachen Werte aus. So stehen diese in der Sprache des jeweiligen Anwenders in der Liste.

Referenz

Werden die ausgewählten Werte als Referenzen auf den eigentlichen Auswahlwert gespeichert, erreicht man folgendes:

- Die Werte werden den Anwendern in der Benutzersprache angezeigt.
- Eine Mehrfachgruppierung in Ansichten ist möglich.
- Filter in Ansichten sind sinnvoll möglich
- Änderungen an der Auswahlliste haben Auswirkungen auf die gewählten Werte

Empfohlen: Referenz

Es wird empfohlen die Möglichkeit der Referenz zu nutzen. Die Möglichkeit die Werte selbst zu speichern bietet den Vorteil, dass einmal ausgewählte Werte nicht mehr veränderbar sind. Dies ist mit einem koordinierten Umgang der Auswahllisten ebenfalls zu erreichen.

Welche Möglichkeiten stehen dazu zur Verfügung?

1. Auswahllisten können in der Programmierung vor dem Ändern in der UI geschützt werden
2. Auswahllistenwerte können in der Programmierung vor dem Ändern/Löschen in der UI geschützt werden
3. Auswahllistenwerte können Inaktiv gesetzt werden (Anstatt zu Löschen). Dies bedeutet sie werden noch angezeigt aber können nicht mehr ausgewählt werden.

4.4. API

Für den Umgang mit Auswahllisten steht eine API zu Verfügung. Die programmierten Auswahllisten haben den konkreten Typ `EnumMyEnum`. Die konfigurierten Enums sind immer vom Typ `ValueEnum` welcher auch der Basistyp der programmierten Auswahlilisten ist.

```
Api.Enum
```

Werte laden

Man kann einzelne Werte einer Auswahlliste laden.

```
EnumMyEnum firstValue = Api.Enum.GetEnumValue<EnumMyEnum>(EnumMyEnum.FirstValu
e.ValueGuid);
ValueEnum secondValueBase = Api.Enum.GetEnumValue(EnumMyEnum.SecondValue.Value
Guid);
EnumMyEnum secondValue = (EnumMyEnum)secondValueBase;
```

Und man kann alle Werte einer Auswahlliste laden

```
IEnumerable<EnumMyEnum> allValues = Api.Enum.GetEnumValues<EnumMyEnum>();
IEnumerable<ValueEnum> allValuesBase = Api.Enum.GetEnumValues(EnumMyEnum.Maste
rGuid);
```

Auswahlliste laden

Man kann einen einzelne oder alle Auswahlllsiten laden.

```
MasterEnum master = Api.Enum.GetMasterEnum(EnumMyEnum.MasterGuid);
IReadOnlyCollection<MasterEnum> masters = Api.Enum.GetAllMasterEnums();
```

Wenn man den Typ der Auswahlliste hat, kann man auch die Guid laden.

```
Guid masterGuid = Api.Enum.GetMasterGuidByType(typeof(EnumMyEnum));
```

Werte auf Basis ihrer Übersetzung erhalten

Wenn einem nur die Übersetzung bekannt ist, kann man in Ausnahmesituationen auch den Wert auf Basis der Übersetzung laden. Dies ist aber nicht zu empfehlen, da die Übersetzungen sich zu Jederzeit ändern können.

```
IEnumerable<ValueEnum> valuesByTranslation = Api.Enum.GetEnumValuesByTranslati
on(master, "First value", caseInsensitiv: true);
```

Mit dieser Methode ist es möglich auf Basis eines Textes einen neuen Wert anzulegen falls er noch nicht existiert. Insbesondere beim Import macht dies Sinn.

```
(Guid valueGuid, bool isNew) thirdValue = Api.Enum.GetOrCreateEnumValueByTrans
lation(master, "Third value", caseInsensitiv: true);
```

4.5. Übung 1

Installieren Sie das NuGet-Paket “BA.Designer”.

Erstellen Sie einen Ordner “Enums” und dort eine eigene Auswahlliste.

- Es sollen Typen von Maschinen (Engine Types) aufgelistet werden.
- Mindestens 3 Auswahlwerte mit den entsprechenden Übersetzungen.
- Ein zusätzliches Feld Description, welches im Konstruktor befüllt wird.

Lösung

5. Datentabellen

Die Datentabellen in Business App basieren auf dem XPO Framework von Dev Express, welches ein ORM Model abbildet.

Es ist möglich neue Datentabellen zu definieren und vorhandene zu erweitern. Mit Hilfe der Basisdatentabellen ist es möglich verschiede Untertypen zu erstellen. Die programmatische Definition ist dann sinnvoll, wenn Funktionalitäten mit konkreten Datentabellen und Feldern umgehen können müssen. Eine programmatische Anlage stellt dies sicher und kann vom Konfigurator auch nicht mehr verändert werden. Aber auch mit Feldern die konfigurativ angelegt wurden, kann in der Programmierung umgegangen werden.

Als Basis für eine eigene Datentabelle dient `OrmBABase`. Diese Datentabelle ist der Standard aller Datentabellen in Business App. Folgendes bietet diese Basis

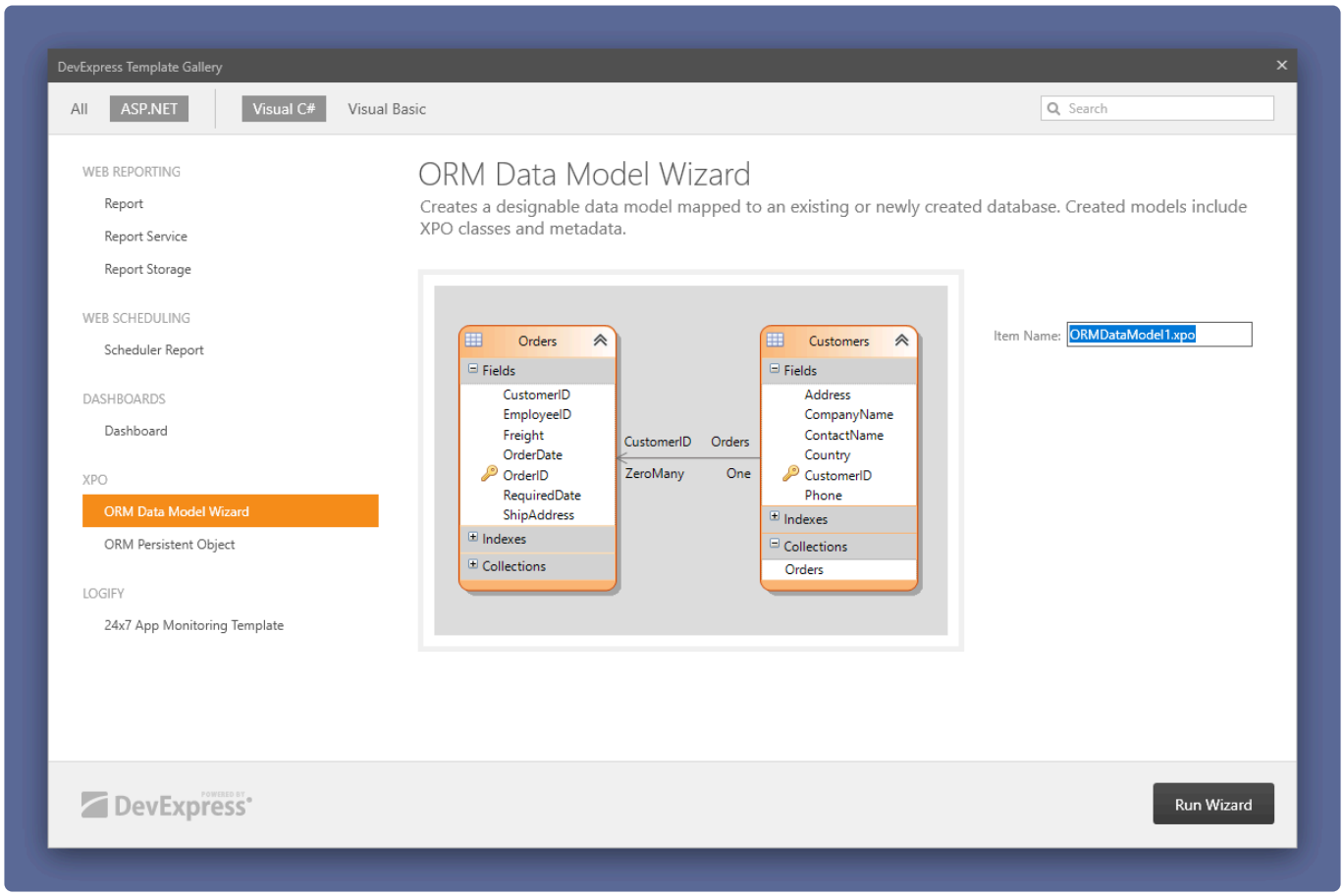
- Es gibt eine zugehörige Konfiguration.
- Validierung
- Created / CreatedBy
- Modified / ModifiedBy
- Änderungsverfolgung
- API zum Umgehen mit Auswahllisten
- Unterstützung von Events
- API für Relationen
- API für Berechtigungsprüfungen

Weiterhin kann als Basis auch eine spezielle Basis-Datentabelle mit weiteren Funktionalitäten dienen. Beispielsweise Basis.Vorgang (`OrmActivityBase`) mit einem Betreff, einem Datumswert, Status, Tags. Oder Basis.Anwendungsprotokoll (`OrmLogBase`), welche für ein eigenes Anwendungsprotokoll genutzt werden kann. Welche Basis Datentabelle zur Verfügung stehen ist abhängig davon, welche Module konkret installiert sind.

Es ist ebenfalls möglich eine bestehende Datentabelle zu erweitern. Beispielsweise Anruf (`OrmPhoneCall`), welche selbst auf Basis.Vorgang beruht und weitere Felder beinhaltet. Tut man dies wird keine neue Datentabelle definiert, sondern die bestehende Tabelle Anruf erhält weitere Felder.

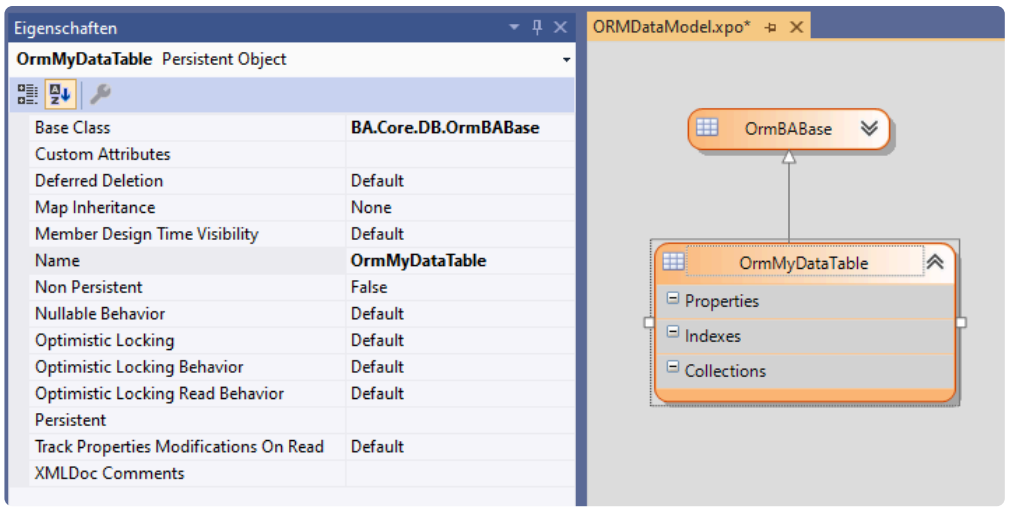
5.1. Definition

Die Definition einer Datentabelle erfolgt am einfachsten über das XPO ORM Data Model Wizard von DevExpress. Rechter Mausklick auf dem Projekt “Add devExpress Item\New Item”



Der Wizard wird an keine Datenbank angebunden. Über die Toolbox kann nun ein “Persistent Object” hinzugefügt werden. In Business App werden alle Datentabellen mit dem Präfix “Orm” benannt.

Um die “Base Class” der Datentabelle setzen zu können, muss sie über “Add Assembly”, Auswahl der korrekten DLL aus dem “bin” Ordner und Auswahl des gewünschten Typs (Datentabelle) hinzugefügt werden. Eine Referenz auf die DLL muss nicht hinzugefügt werden.



Für die Definition einer neuen Datentabelle in Business App wird noch eine Erweiterung der `EnumDataSource` benötigt. Beispiel:

```
[EnumExtension(typeof(EnumDataSource))]
public static class EnumDataSourceExtension
{
    public const string MyDataTableGuid = "[INSERT DATA TABLE GUID]";
    public static readonly EnumDataSource MyDataTable = new EnumDataSource(MyDataTableGuid, 1000, "[INSERT TRANSLATION GUID]", "data_table");
}
```

Eine Schnittstelle wird ebenfalls benötigt, welche per Attribute an den Auswahllistenwert gebunden wird. Beispiel:

```
[OrmDataSource(EnumDataSourceExtension.MyDataTableGuid)]
public interface IOrmMyDataTable { }
```

Die neue Datentabelle (Persistent Object) muss das Interface implementieren

```
 public partial class OrmMyDataTable : IOrmMyDataTable
```

Mit dem Attribut `DesignTimeVisible` mit dem Parameter `true` steht die Datentabelle im Designer auch zur Auswahl und mit `Description` kann man der Datetabelle eine Beschreibung geben.

```
[DesignTimeVisible(true)]
[Description("Meine eigene Datentabelle")]
public partial class OrmMyDataTable : IOrmMyDataTable
```

Als Letztes wird die neue Datentabelle per Dependency Injection bekannt gemacht.

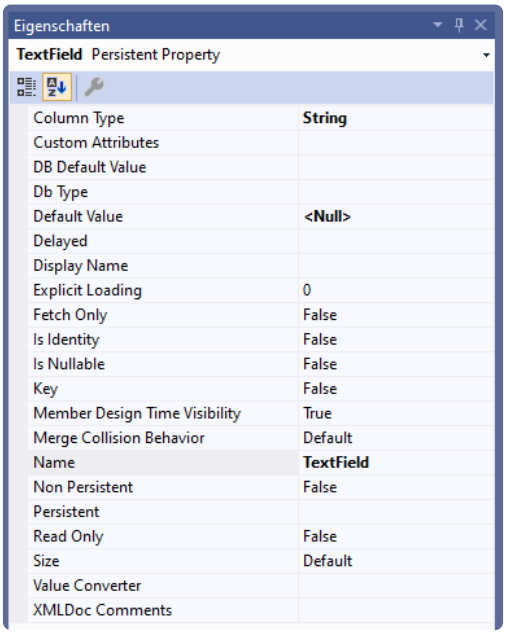
```
Bind<IOrmMyDataTable>().To<OrmMyDataTable>();
```

Wird eine Datentabelle erweitert, ist sowohl der Auswahlwert als auch die Schnittstelle schon vorhanden. In diesem Fall muss lediglich ein Rebind durchgeführt werden.

```
Rebind<IOrmPhoneCall>().To<OrmPhoneCall>();
```

5.2. Feldtypen definieren

Es stehen verschiedene Feldtypen zur Verfügung, die hier kurz beschrieben werden. Im Orm Wizard wird auf der Datentabelle ein “Persistent Property” hinzugefügt. In den Eigenschaften können dann verschiedene Einstellungen vorgenommen werden.



> **!** Die Namen von Feldern sollten mit bedacht gewählt werden. Namen wie “Number” oder “String” sollten vermieden werden. Konfigurierte Feldnamen werden vom System automatisch mit dem Präfix “Custom_” versehen. Einen eigenen Präfix auch für programmierter Feldnamen vermeidet Fehler.

Textfeld / HTML Felder

Eigenschaften

- Column Type: String
- Name: [Beliebiger Feldname]
- Size: Default / Unlimited / Konkrete Länge

Beim Textfeld muss eine Größe festgelegt werden. Die Standardgröße sind 100 Zeichen. Dabei sollte man beachten, das die Größe sinnvoll gewählt ist. Die Wahl von “Unlimited” für jeden Wert sollte vermieden werden.

Für das Speichern von HTML Inhalten werden zurzeit Textfelder mit der Größe “Unlimited” verwendet. HTML Felder können in Teil-Datentabellen nicht verwendet werden.

> **!** HTML Felder sind in Teil-Datentabellen nicht möglich.

Boolesches Feld

Eigenschaften

- Column Type: Boolean
- Name: [Beliebiger Feldname]

Datum-Zeit Feld

Eigenschaften

- Column Type: DateTime
- Custom Attributes: “ValueConverter(typeof(DevExpress.Xpo.Metadata.UtcDateTimeConverter))”
- Is Nullable: True / False
- Name: [Beliebiger Feldname]

Datum-Zeit-Felder werden in Business App grundsätzlich in UTC in die Datenbank geschrieben. Durch den definierten Value Converter arbeitet man im Source Code grundsätzlich mit dem Local Format. Dies muss dringend berücksichtigt werden, ansonsten tauchen entsprechende Zeitverschiebungen auf.

Wiederholendes Datum-Zeit Feld

Um ein Datumsfeld zu erlauben Wiederholungsinformationen zu beinhalten, muss ein weiteres Attribut gesetzt werden.

Zusätzliche Custom Attributes

- BA.Core.CustomAttributes.RecurrentDateAttribute()

> **!** Wiederholendes Datum-Zeit Feld ist in Teil-Datentabellen nicht möglich.

In diesem Fall werden vom System automatisch weitere Spalten angelegt. Das sind `[Feldname]_RecurrenceType` mit dem Wert 0 für nicht wiederholende und 1 für wiederholende Werte. Und `[Feldname]_RecurrenceInfo` welches die Wiederholungsinformationen in einem XML Format beinhaltet.

Ausnahmen von Wiederholungen

Im Benutzerinterface von Business App existiert zurzeit keine Möglichkeit Ausnahmen von Wiederholungen anzulegen oder zu verwalten. Es existieren aber Möglichkeiten diese Ausnahmen technisch zu erstellen. Der Kalender und die Mehrfachgruppierung in Ansichten können diese dann darstellen.

Ob und wie man diese Funktionalität schon in Projekten nutzen kann, ist individuell zu erfragen.

Zahlenfelder

Eigenschaften

- Column Type: [Zahltyp]
- Is Nullable: True / False
- Name: [Beliebiger Feldname]

Auswahllistenfeld

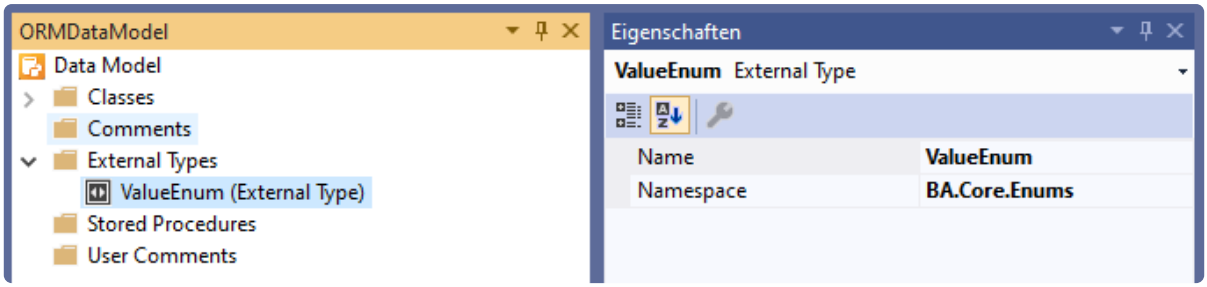
Es gibt zwei Arten von Auswahllisten. In der Ersten wird lediglich eine Referenz auf den eigentlichen Wert gespeichert und in der Anderen werden die Werte abgespeichert. Primär wird die Referenz Variante empfohlen.

Referenz

Eigenschaften

- Column Type: BA.Core.Enums.ValueEnum
- Custom Attributes: “BA.Core.CustomAttributes.OrmEnumFieldAttribute(BA.Training.Enums.EnumMyEnum.Guid, BA.Core.CustomAttributes.TypeOfOrmEnumField.Normal)”
- Is Nullable: False
- Name: [Beliebiger Feldname]
- Value Converter: BA.Core.Converters.EnumValueConverter

Der Column Type muss beim Ersten Mal dem Datenmodel hinzugefügt werden.



In diesem Fall wird von BA beim Starten automatisch eine weitere Datenspalte mit dem Namen `[Feldname]_AllValues`.angelegt.

Werte

Um eine Auswahllistenfeld zu definieren, indem nur die Werte gespeichert werden, wird ein Property benötigt.

Eigenschaften

- Column Type: String
- Custom Attributes:
  “BA.Core.CustomAttributes.OrmEnumFieldAttribute(BA.Training.Enums.EnumMyEnum.Guid, BA.Core.CustomAttributes.TypeOfOrmEnumField.Simple)”
- Name: [Beliebiger Feldname]
- Size: Möglichst konkret (im Zweifelsfall oder Multi-Value unlimited)

5.3. Teil-Datentabelle

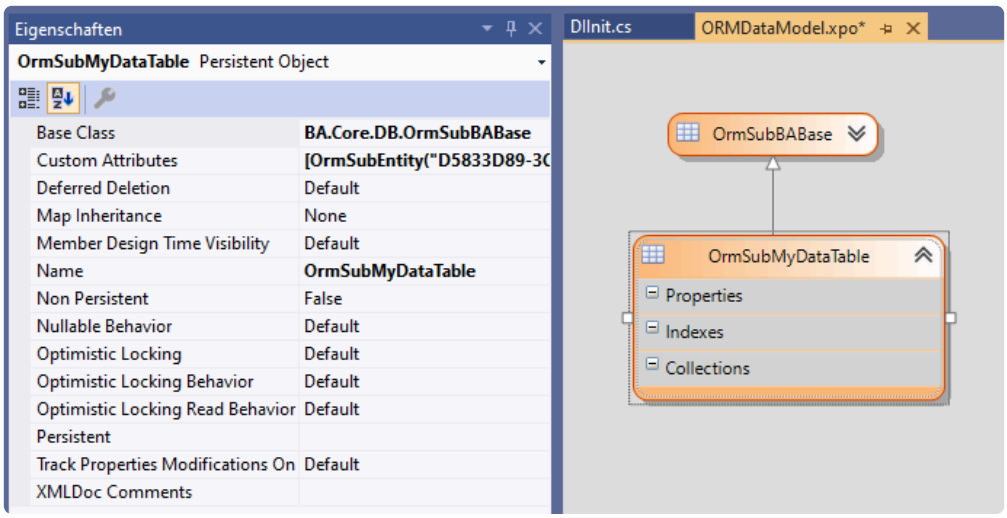
Es ist möglich neue Teil-Datentabellen zu definieren. Das Erweitern vorhandener Teil-Datentabellen ist zur Zeit nur konfigurativ möglich. Die programmatische Definition ist dann sinnvoll, wenn Funktionalitäten mit konkreten Teil-Datentabellen und Feldern umgehen können müssen. Eine programmatische Anlage stellt dies sicher und kann vom Konfigurator auch nicht mehr verändert werden. Aber auch mit Feldern die konfigurativ angelegt wurden, kann in der Programmierung umgegangen werden.

Als Basis für eine eigene Teil-Datentabelle dient `OrmSubBABase`. Folgendes bietet diese Basis

- Es gibt eine zugehörige Konfiguration.
- Validierung
- Feste Reihenfolge der Teil-Datensätze (SortOrder)
- API zum Umgehen mit Auswahllisten

5.3.1. Definieren

Um eine Teil-Datentabelle programmatisch zu definieren muss ein Persistent Object angelegt werden und als Base Class wird `OrmSubBABase` eingetragen.



Bis auf wenige Ausnahmen können dieselben Feldtypen wie bei den Haupt-Datentabellen verwendet werden.

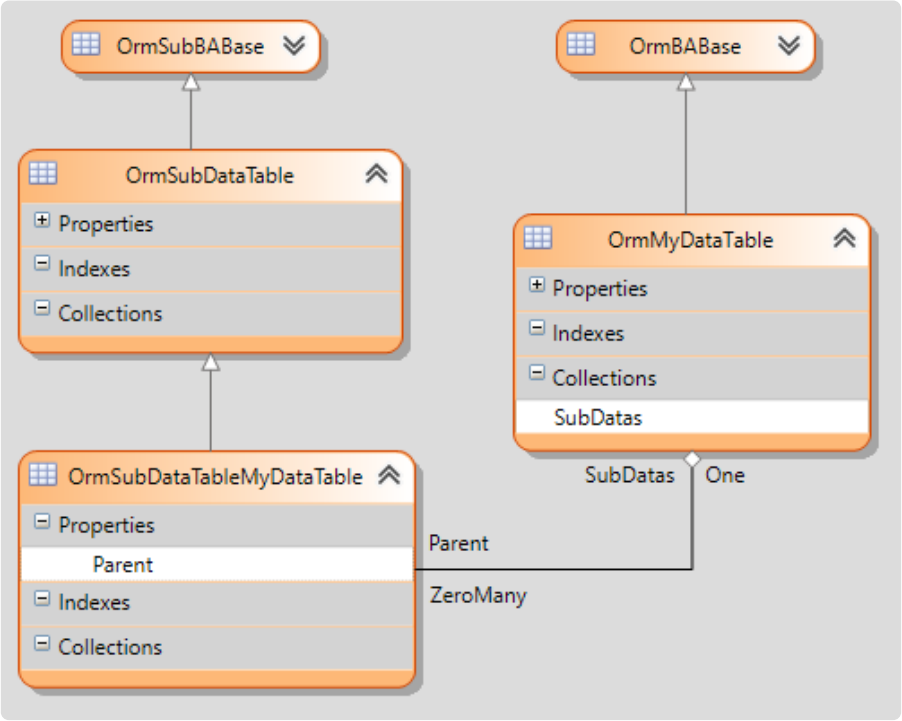
Um die neue Teil-Datentabelle dem System bekannt zu machen muss auf dieser ein Attribut gesetzt werden.

```
[OrmSubEntity("[INSERT SUB TABLE GUID]", "[INSERT TRANSLATION GUID]")]
```

Damit steht die Teil-Datentabelle zur Verfügung und kann konfigurativ in Datentabellen verwendet werden.

5.3.2. In einer Datentabelle definieren

Um eine Teil-Datentabelle in einer Haupt-Datentabelle zu verwenden, muss ein “Persistent Object” erstellt werden, welches als Basis-Klasse die Teil-Datentabelle hat.



Eigenschaften des Persisten Object

- Base Class: [Teil-Datentabelle]
- Map Inheritance: ParentTable
- Name: OrmSub[Beliebiger Name]

Diesem Object wird ein “Persistent Property” hinzugefügt. Eigenschaften:

- Column Type: [Haupt-Datentabelle]
- Custom Attributes: “NoForeignKey”
- Name: Parent

Anschließend wird eine “Aggregated One-To-Many Relationship” zwischen der Haupt-Datentabelle und dieser speziellen Teil-Datentabelle erstellt. Eigenschaften:

- Source Collection Name: [Beliebiger Feldname]
- Target Field Name: Parent

5.4. Gemeinsame Felder

Um ein gemeinsames Feld anzulegen, muss lediglich eine Auswahlliste erweitert werden.

```
[EnumExtension(typeof(EnumCommonFields))]
public static class EnumCommonFieldsExtension
{
    public const string MyCommonFieldGuid = "[INSERT UNIQUE GUID]";
    public static readonly EnumCommonFields MyCommonField = new EnumCommonFiel
ds(MyCommonFieldGuid, 305, "[INSERT TRANSLATION GUID]", EnumDataTypes.String,
"Common_MyCommonField");
}
```

Möchte man für eine Datentabelle eine Formel für ein gemeinsames Feld vorbelegen, wird dies mit dem Attribut `CommonDefinition` an der Datentabelle getan.

```
[CommonDefinition(EnumCommonFields.MyCommonFieldGuid, formulaString)]
```

5.5. Dateien

Zu Datentabellen können Dateien hinzugefügt werden. Diese werden technisch aufgeteilt in einen Speicher und einem beschreibenden Objekt. Das beschreibende Objekt wird zum Datensatz abgelegt und kann pro Datei mehrfach existieren. Die Speicherung der Datei erfolgt aber nur einmalig, so daß kein zusätzlicher Speicher benötigt wird, wenn die identische Datei mehrfach zu verschiedenen Datensätzen abgelegt wird. Dieser Mechanismus wird von BA vollkommen Transparent durchgeführt, so dass dafür keine besonderen Implementierungen notwendig sind.

Für den Umgang mit Dateien gibt es Hilfsmethoden in `Api.Attachments`.

Dateien auf einem Datensatz

Um auf einen Datensatz Methoden für den Umgang mit Dateien zur Verfügung zu haben, muss die Erweiterung über ein `using` eingebunden werden.

```
using BA.Core.Extensions.BA.Core.DB;
```

Hinzufügen

```
byte[] data = new byte[];
OrmAttachment attachment = Api.Attachments.CreateAttachment("Dateinname.txt",
data, null, session: orm.Session);
orm.AddAttachment(attachment, false);
orm.Save();
```

Abfragen

Einzelne Datei

```
OrmAttachment attachment = orm.GetAttachmentById(attachmentGuid);
```

Dateiliste

```
IEnumerable<OrmAttachment> attachments = orm.GetAttachments();
```

Eingebette Bilder in HTML Feldern werden ebenfalls als Dateien abgelegt und werden daher ebenfalls nur einmalig in den Speicher abgelegt. Um diese zu Laden muss man den Parameter `inline` setzen.

```
IEnumerable<OrmAttachment> attachments = orm.GetAttachments(inline: Attachment
InlineType.Inline);
```

Entfernen

```
orm.RemoveAttachment(attachment);
```

5.6. Basis-Datentabellen definieren

Die Basis-Datentabellen sind abstrakte Datentabellen von denen selbst kein Datensatz erstellt werden kann. Sie bilden eine gemeinsame Basis für verschiedene konkrete Datentabellen. Vorteile:

- Ansichten können auf Basis-Datentabellen basieren und damit eine gemeinsame Darstellung ermöglichen.
- Gemeinsame Felder
- Gemeinsame Teil-Datentabellen
- Einfacher programmatischer Umgang
- Gemeinsame Abfragen

Um eine Basis-Datentabelle zu definieren, definiert man zuerst eine normale Haupt-Datentabelle. Es wird ein zusätzlicher Auswahllistenwert benötigt.

```
[EnumExtension(typeof(EnumInheritanceTypes))]
public static class EnumInheritanceTypesExtension
{
    public const string MyBaseDataTableGuid = "[INSERT DATA TABLE GUID]";
    public static readonly EnumInheritanceTypes MyBaseDataTable = new EnumInhe
ritanceTypes(MyBaseDataTableGuid, 1000, "[INSERT TRANSLATION GUID]");
}
```

Sinnvollerweise wird die Klasse `abstract` definiert, damit auch nicht zufällig Datensätze davon angelegt werden können

```
public abstract partial class OrmMyBaseDataTable : IOrmMyBaseDataTable
```

Es wird ein Attribut an der Klasse benötigt, um sie dem System als Basis-Datentabelle bekannt zu machen.

```
BA.Core.CustomAttributes.OrmInheritanceNode(EnumInheritanceTypesExtension.MyBa
seDataTableGuid)
```

Mit diesem Attribute legt man fest ob die Basis-Datentabelle in der Konfiguration zur Auswahl steht.

```
[System.ComponentModel.DesignTimeVisible(true)]
```

Existierende Basis-Datentabellen können programmatisch nicht erweitert werden.

5.7. Validatoren

Mit Hilfe eines Validators können Feldwerte auf bestimmte Kriterien überprüft werden. Um ein Feld mit einem Validator zu versehen, muss ein weiteres Attribute in den “Custom Attributes” angegeben werden. Es dürfen mehrere Validatoren angegeben werden aber jeder Typ maximal einmal.

Man kann vorhandene Validatoren setzen und auch eigene Erstellen. Selbsterstellte Validatoren können entweder nur programmatisch genutzt werden oder sie können im Designer angezeigt, bzw. selbst gesetzt werden.

5.7.1. Standard Validatoren

Zwingend benötigt

Attribute

```
BA.Core.CustomAttributes.Validations.BARequiredAttribute()
```

Länge eines Textes

Attribute

```
BA.Core.CustomAttributes.Validations.LengthValidatorAttribute(minimum, maximum)
BA.Core.CustomAttributes.Validations.LengthValidatorAttribute(maximum)
```

Zahlbereich

Attribute

```
BA.Core.CustomAttributes.Validations.NumberRangeValidatorAttribute(minimum, maximum)
```

Datumsbereich

Attribute

```
BA.Core.CustomAttributes.Validations.DateRangeValidatorAttribute(minYear, minMonth, minDay, minHour, minMinute, minSecond, maxYear, maxMonth, maxDay, maxHour, maxMinute, maxSecond)
```

Spaltenvergleich

Attribute

```
BA.Core.CustomAttributes.Validations.CompareValidatorAttribute(BA.Core.Enums.EnumColumnValueCompareOperators.[Operator]Guid, "[FieldNameToCompare]")
```

Eindeutiger Spaltenwert

Attribute

```
BA.Core.CustomAttributes.Validations.UniqueValidatorAttribute()
```

Bedingung

Attribute

```
BA.Core.CustomAttributes.Validations.FreeValidatorAttribute(conditionFormula, BA.Core.Enums.EnumDataSource.[DataSource]Guid)
```

Regulärer Ausdruck

Attribute

```
BA.Core.CustomAttributes.Validations.RegexValidatorAttribute(regularExpressio
n, System.Text.RegularExpressions.RegexOptions)
```

IBAN Validator

Attribute

```
BA.Core.CustomAttributes.Validations.IBANValidatorAttribute()
```

BIC Validator

Attribute

```
BA.Core.CustomAttributes.Validations.BICValidatorAttribute()
```

5.7.2. Eigene Validatoren

Programmatischer Validator

Für einen eigenen Validator muss ein Attribut implementiert werden, welches in den “Custom Attributes” des Feldes hinterlegt wird. Dort werden die Parameter und die Validierungsmethode definiert.

```
public class MyValidatorAttribute : ValidationAttribute
{
    public readonly string StartsWith;

    public MyValidatorAttribute(string startsWith) : base()
    {
        StartsWith = startsWith;
    }

    protected override ValidationResult IsValid(Object value, ValidationContex
t validationContext)
    {
        if (value == null)
            return ValidationResult.Success;

        ErrorMessage = "";
        if (value is string valueString)
        {
            if (!valueString.StartsWith(StartsWith))
            {
                ErrorMessage = "[INSERT TRANSLATION GUID]".Translate(StartsWit
h);
                return new ValidationResult(ErrorMessage, new String[] { valid
ationContext.MemberName });
            }
        }

        return ValidationResult.Success;
    }
}
```

Steuerelement für den Validator

Auf Basis des Attributs kann man den Validator in der Konfiguration setzen. Dazu wird eine entsprechende Klasse implementiert. Die Eigenschaften der Klasse werden automatisiert als Felder im Designer angezeigt. Mit `ShowInToolbox` legt man fest ob der Konfigurator den Validator selbst auf beliebige Felder setzen kann.

```
[Serializable]
[Toolbox(EnumConfigurationType.OrmEntityConfigurationGuid, EnumConfigurationTy
pe.OrmSubEntityConfigurationGuid, ShowInToolbox = true)]
public class MyValidator : FieldValidatorBase
{
    // Kann in der Konfiguration gesetzt werden
    [DisplayName("Beginnt mit")]
    [HelpText("Erklärung von 'Beginnt mit'")]
    [PropertiesGroup("ba62093b-048c-4842-b9e1-9400ece745b1", 1)] // Allgemein
e Einstellungen
    [PropertiesForm(5)]
    [BARequired] // Muss gefüllt werden. Ist selbst ein Validator Attribut.
    public string StartsWith { get; set; }

    public MyValidator()
    {
        Id = new Guid("[INSERT VALIDATOR GUID]");
        ToolboxName = "Beginnt mit";
        ControlInitName = "StartsWithValidator";
        ToolboxGroupName = Constants.Designer.ToolboxGroups.ORM.Validators;
        Icon = "spellcheck";
        DesignerHintText = "Erklärung des Validators";
    }

    // Definiert das dazugehörige Attribut
    public override ValidationAttribute GetValidationAttribute()
    {
        return new MyValidatorAttribute(StartsWith);
    }

    // Validiert das Konfigurationsobjekt
    public override List<ValidationModel> IsValid(ConfigurationBase configurat
ion = null, EnumDesignerValidationState validationState = null)
    {
        List<ValidationModel> result = base.IsValid(configuration, validationS
tate);

        if (!StartsWith.Length > 3)
            result.Add(new ValidationModel(nameof(StartsWith), this, "Feld mus
s mindestens vier Zeichen beinhalten."));

        return result;
    }
}
```

Attribut mit Steuerelement verknüpfen

Mit `GetValidationAttribute()` verknüpft man das Steuerelment mit dem Attribut. Möchte man das Attribut schon in der programmíerten Datentablle setzen und in diesem Fall das Steuerelement anzeigen, muss das Interface `IDesignableValidator` und die Methode `CreateValidatorControl()`* beim Attribut implementiert werden.

```
public class MyValidatorAttribute : ValidationAttribute, IDesignableValidator
```

```
public FieldValidatorBase CreateValidatorControl()
{
    return new MyValidatorAttribute()
    {
        StartsWith = StartsWith,
    };
}
```

Drag & Drop

Man sollte die Drag & Drop Regeln beachten. Mit den beiden Klassen `TextValidatorBase` und `GenericValidatorBase` als Basisklasse des Validators, kann dies vereinfacht werden.

```
public class MyValidator : TextValidatorBase { }
```

Alternativ müssen die Regeln manuell gesetzt werden.

```
[assembly: DnDRule(typeof(OrmTextField), typeof(MyValidator), EnumConfigurationType.OrmEntityConfigurationGuid)]
```

5.8. Datensätze API (ORMs)

Business App stellt eine API zur Verfügung, um mit Datensätzen arbeiten zu können. Es können Datensätze erstellt, geladen und verändert werden. Mit Hilfe von Abfragen in LINQ können beispielsweise eine Menge von Datensätze ermittelt werden.

```
Api.ORM
```

XPO Session

Man bekommt eine neue Session oder UnitOfWork.

```
Session session;
session = Api.ORM.GetDefaultSession();
session = Api.ORM.GetNewSession();
```

XPO Unit Of Work

Mit der UnitOfWork hat man eine Session, die Transaktionen ermöglicht. Daher muss neben dem `Save()` der Orm Objekte noch ein `CommitChanges()` ausgeführt werden.

```
UnitOfWork uow = Api.ORM.GetNewUnitOfWork();
session = uow;
uow.CommitChanges();
```

5.8.1. Erstellen und Bearbeiten

ORM Datensatz erstellen

Um einen neuen Datensatz zur erstellen, hat man zwei Möglichkeiten. In der ersten Zeile wird ein neuer Datensatz mit der Angabe der Guid der Datentabelle erstellt. Dies ist zum Beispiel dann notwendig, wenn man programmatisch mit konfigurierten Datentabellen umgehen muss. Dies sollte aber nur in Ausnahmefällen notwendig sein. Die zweite Zeile erstellt einen neuen Datensatz mit Angabe des Typs der Datentablle.

```
OrmBABase newOrm = Api.ORM.GetNewORM(EnumDataSourceExtension.MyDataTable.Value
Guid, session);
OrmMyDataTable newMyData = Api.ORM.GetNewORM<OrmMyDataTable>(session);
```

Umgang mit Feldern

Programmierte Standardfelder stehen als normale Eigenschaften zur Verfügung

```
newMyData.TextField = "Inhalt";
if (!string.IsNullOrEmpty(newMyData.TextField))
    newMyData.TextField += " Mehr Inhalt";
if (newMyData.BooleanField)
    newMyData.BooleanField = false;
```

Konfigurierte Standardfelder können über die Member Methoden genutzt werden.

```
newMyData.SetMemberValue("TextField", "Inhalt");
if (!string.IsNullOrEmpty((string)newMyData.GetMemberValue("TextField")))
    newMyData.SetMemberValue("TextField", newMyData.GetMemberValue("TextFiel
d") + " Mehr Inhalt");
if ((bool)newMyData.GetMemberValue("BooleanField"))
    newMyData.SetMemberValue("BooleanField", false);
```

Umgang mit Auswahllisten

Die spezialisierten Methoden für Auswahllistenwerte benötigen den Feldnamen als string. Falls die Datentabelle programmiert ist, sollte man in jedem Fall `nameof()` verwenden.

```
// Wert setzen (Entfernt alle anderen)
newMyData.SetEnumValue(nameof(OrmMyDataTable.EnumField), EnumMyEnum.FirstValu
e);
```

```
// Beinhaltet  einen Wert
if (newMyData.EnumContains(nameof(OrmMyDataTable.EnumField), EnumMyEnum.FirstV
alue))
    // Löscht alle Werte
    newMyData.ClearEnumValues(nameof(OrmMyDataTable.EnumField));

// Fügt einen weiteren Wert hinzu
newMyData.AddEnumValue(nameof(OrmMyDataTable.EnumField), EnumMyEnum.FirstValu
e);

// Entfernt einen Wert
newMyData.RemoveEnumValue(nameof(OrmMyDataTable.EnumField), EnumMyEnum.FirstVa
lue);

// Holt alle Werte (Änderungen an der Liste haben keine Auswirkungen)
IEnumerable<EnumMyEnum> enumValues = newMyData.GetEnumValues<EnumMyEnum>(nameo
f(OrmMyDataTable.EnumField));
```

Umgang mit Teil-Datentabellen

Teil-Datensätze können nicht eigenständig existieren, sondern nur in Zusammenhang eines Hauptdatensatzes. Felder können dann identisch gesetzt werden.

> Achtung SortOrder muss manuell und konistent gesetzt werden.

Neuen Teil-Datensatz erstellen

```
OrmSubDataTableMyDataTable subData = newMyData.SubDatas.AddNewObject();
subData.SortOrder = newMyData.SubDatas.Count() - 1;
```

Teil-Datensatz Abfragen

Position 0

```
OrmSubDataTableMyDataTable firstSubData = newMyData.SubDatas.Where(ff => ff.So
rtOrder == 0).FirstOrDefault();
```

Mit einem bestimmten unique Key

```
OrmSubDataTableMyDataTable keySubData = newMyData.SubDatas.Where(ff => ff.Enum
Contains(nameof(OrmSubDataTable.EnumField), EnumMyEnum.FirstValue)).FirstOrDef
ault();
```

Teil-Datensatz löschen

```
newMyData.SubDatas.Remove(keySubData);
int sortOrder = 0;
foreach(OrmSubDataTableMyDataTable subData in newMyData.SubDatas)
{
    subData.SortOrder = sortOrder;
    sortOrder++;
}
```

Teil-Datensätze speichern

Es muss immer der Hauptdatensatz gespeichert werden! Nur dadurch ist sichergestellt, das Beispielsweise das Datensatzprotokoll und das Änderungsdatum korrekt sind.

```
newMyData.Save();
```

5.8.2. Laden, Abfragen

Einzelnes ORM laden

Man kann anhand der Guid der Datentabelle ein bestimmtes ORM laden und erhält als Rückgabe ein Objekt vom Typ `OrmBABase`. Dieses könnte man dann anschließend in die konkrete Orm Klasse casten. Oder man arbeitet anschließend mit dem Basis Objekt weiter. Dies könnte dann notwendig sein, wenn man auf eine konfigurierte Datentabelle zugreifen möchte.
Wenn man die Datentabelle programmiert hat, kann man auch direkt das konkrete Orm Objekt laden.

```
OrmBABase ormBase = Api.ORM.GetOrm(EnumDataSourceExtension.MyDataTable.ValueGu
id, "[INSERT ORM GUID]".ToGuid(), session);
OrmMyDataTable myData = Api.ORM.GetOrm<OrmMyDataTable>("[INSERT ORM GUID]".ToG
uid(), session);
```

Abfragen

Bei Abfragen muss man zwischen Abfragen entscheiden, die die Leserechte berücksichtigen und welche die sie nicht berücksichtigen. Mehr zum Thema Rechte findet sich im entsprechenden Kapitel.

```
IQueryable<OrmMyDataTable> query = Api.ORM.GetQuery<OrmMyDataTable>(session);
IQueryable<OrmMyDataTable> queryWith = Api.ORM.GetQueryWithReadPermissions<Orm
MyDataTable>(session);
```

Mit LINQ to XPO kann man nun diese Abfragen gestalten.

```
queryWith = queryWith.Where(ff => ff.TextField != null && ff.TextField.Contain
s("Inhalt"));
int count = queryWith.Count();
```

Abfragen auf Basis-Datensätze

Abfragen können auch auf Basis-Datensätze erstellt werden. In diesem Fall erhält man alle Datensätze, welche von diesem Basis-Datentyp erben.

```
IQueryable<OrmActivityBase> activities = Api.ORM.GetQuery<OrmActivityBase>(ses
sion);
IQueryable<OrmActivityBase> activitiesWith = Api.ORM.GetQueryWithReadPermissio
ns<OrmActivityBase>(session);
```

Auch Abfragen auf alle Datensätze sind möglich. Dies könnte dann notwendig sein, wenn man verschiedene Datensätze haben möchte, die keine gemeinsame Basis-Datentabelle haben.

```
IQueryable<OrmBABase> orms = Api.ORM.GetQuery<OrmBABase>(session);
IQueryable<OrmBABase> persons = orms.Where(ff => ff.IsType(typeof(OrmContac
t)) || ff.IsType(typeof(OrmUserProfile)));
```

Abfragen auf alle Datensätze, inkl. Leserechte sind möglich, aber man sollte unbedingt beim Erstellen der Abfrage auf die Typen der gewünschten Datensätze einschränken, da die Berücksichtigung aller Leserechte aller möglichen Typen extrem komplex ist und sich die Laufzeit der Datenbankanfrage dadurch sehr stark erhöhen kann.

```
IQueryable<OrmBABase> ormsWith = Api.ORM.GetQueryWithReadPermissions<OrmBABas
e>(session, new[] { typeof(OrmContact), typeof(OrmUserProfile) });
```

Performance Hinweise

Werden bei den Abfragen nicht die ORMs benötigt sollte man mit `Select()` die Felder konkret angeben. Dadurch wird die Größe der Abfrage beschränkt, und es muss auch kein ORM Objekt erstellt werden. Das Erstellen des ORM Objektes selbst, kann wiederum weitere Abfragen auslösen.

```
IQueryable<OrmMyDataTable> query = Api.ORM.GetQuery<OrmMyDataTable>(session);
var anonymousQuery = query.Select(ff => new { oid = ff.Oid, text = ff.TextFiel
d, boolean = ff.BooleanField });
```

Beim Formulieren der LINQ Abfragen sollte man darauf achten, dass die verwendeten Methoden in einen SQL Query umgewandelt werden können. Dadurch wird sichergestellt, das die Abfrage in Ihrer Gesamtheit dem SQL Server übergeben wird. Beispielsweise sind String-Operationen, Datum-Operationen und Zahlen-Operationen Problemlos einsetzbar. Komplexe .Net Methoden sollten aber vermieden werden.

5.8.3. Löschen

Das Löschen von Datensätzen sollte mit Vorsicht ausgeführt werden. Zur Zeit gibt es kein weiches Löschen in dem Sinne, das die Datensätze einfach wiederhergestellt werden können. Daher muss davon ausgegangen werden, das einmal gelöschte Daten auch verloren sind. Teilweise kann in Notsituationen versucht werden, Teile der Daten zu retten, dies ist aber aufwendig und nicht vollständig. In einer der kommenden Versionen wird ein entsprechender weicher Mechanismus implementiert werden.

```
if (!newMyData.IsDeleted)
    newMyData.Delete();
```

5.8.4. .Net Typ der Datensätze

Der .Net Typ des Datensatzes ist nicht unbedingt bekannt. Beispielsweise kann die Datentabelle in einem anderen Modul erweitert worden sein oder die Datentabelle ist selbst nur konfiguriert worden. In `Api.ORM` befinden sich Methoden, mit denen man den entsprechenden Typen erhalten kann.

```
OrmDataSourceCacheModel cachedType = Api.ORM.GetOrmTypeCacheValue(EnumDataSour
ceExtension.MyDataTable.ValueGuid);
IQueryable<OrmBABase> ormsWithGuid = Api.ORM.GetQueryWithReadPermissions(cache
dType.Guid, session);

IReadOnlyList<OrmDataSourceCacheModel> cachedTypes = Api.ORM.GetOrmTypeCacheVa
luesByBase(BA.Activity.Enums.Extensions.EnumDataSourceExtension.ActivityBase.V
alueGuid);
Type[] types = cachedTypes.Select(ff => ff.Type).ToArray();
IQueryable<OrmBABase> ormsWithType = Api.ORM.GetQueryWithReadPermissions<OrmBA
Base>(session, types);
```

5.9. Events für Datensätze und Teil-Datensätze

Bei der Verarbeitung von Datensätzen werden Events ausgelöst. Dort ist es z.B. möglich Feldwerte zu berechnen und weitere Funktionen aufzurufen. Diese Events werden für einen Datensatz ausgeführt und beeinflussen daher das Laufzeitverhalten von Datensätzen stark, insbesondere bei der Massenverarbeitung selbiger. Man muss daher darauf achten, das diese Funktionalitäten performant implementiert sind.

Um Events für Datensätze zu implementieren, wird eine Klasse benötigt die `BAOrmEventsBase` erweitert. An dieser Klasse, muss das Attribute `BAOrmEvents` gesetzt werden.

```
[BAOrmEvents(typeof(OrmMyDataTable))]
public class OrmEventModifyMyDataTable : BAOrmEventsBase { }
```

Für konfigurierte Datentabellen, ist es auch möglich die Guid der Datentabelle zu verwenden.

```
[BAOrmEvents(EnumDataSourceExtension.MyDataTableGuid)]
public class OrmEventModifyMyDataTable : BAOrmEventsBase { }
```

Es können auch Events für Basis-Datentabellen erstellt werden.

```
[BAOrmEvents(typeof(OrmActivityBase))]
public class OrmEventModifyMyDataTable : BAOrmEventsBase { }
```

Oder auch für mehrere Datentabellen.

```
[BAOrmEvents(typeof(OrmPhoneCall), typeof(OrmMiscellaneous))]
public class OrmEventModifyMyDataTable : BAOrmEventsBase { }
```

Es ist sinnvoll das eine Event Klasse pro Aufgabe implementiert wird. Dies hat den Vorteil, dass dann diese aus einem anderen Teil der Anwendung deaktiviert werden kann.

5.9.1. Life Cycle

In der Implementierung ist man nicht eingeschränkt und kann beliebige Funktionalitäten implementieren. Man sollte sich aber bewusst sein, dass die Events häufig ausgeführt werden und damit ist man aus Gründen der Performance eingeschränkt.

Beispielsweise, könnte man Felder programmatisch vorbelegen

```
public override void OnCreated(OrmBABase orm)
{
    OrmMyDataTable myDataTable = (OrmMyDataTable)orm;
    myDataTable.TextField = "Inhalt";
}
```

Folgende Events stehen im Life Cycle zur Verfügung.

- OnCreated(OrmBABase orm)
- OnSaving(OrmBABase orm)
- OnSaved(OrmBABase orm)
- OnRelationsCreated(OrmBABase orm)
- OnLoaded(OrmBABase orm)
- OnDeleting(OrmBABase orm)
- OnDeleted(OrmBABase orm)

5.9.2. Validierung

Drei verschiedene Events stehen während der Validierung zur Verfügung.

- `OnBeforeValidation` Wird vor der Validierung ausgeführt. An dieser Stelle kann man beispielsweise noch Felder ändern, bevor diese validiert werden. Das Event wird nur aufgerufen, wenn der Datensatz von einer Maske verändert wird.
- `OnOrmValidateModel` Dieses Event wird ausgeführt, wenn bei der Validierung ein Ausnahme auftritt.
- `OnAfterOrmValidation` Wird nach der Ausführung der Validatoren ausgrführt. Zugriff auf den `ModelState` erhält man über den Datensatz selbst (`orm.ModelState`). Es ist auch möglich an dieser Stelle den ModelState abzuändern. Beispielsweise einen Fehler zu erstellen, also eine weitere Validierung durchzuführen oder den Datensatz doch als Valide zu kennzeichnen.

```
public override void OnBeforeValidation(OrmBABase orm)
{
    OrmMyDataTable myDataTable = (OrmMyDataTable)orm;
    myDataTable.TextField = "Inhalt: " + myDataTable.TextField;
}
```

```
public override void OnOrmValidateModel(OrmBABase orm, ref Exception e)
{
    OrmMyDataTable myDataTable = (OrmMyDataTable)orm;
    myDataTable.LastValidatenException = e.Message;
}
```

```
public override void OnAfterOrmValidation(OrmBABase orm, ref bool baseValidati
onResult)
{
    OrmMyDataTable myDataTable = (OrmMyDataTable)orm;
    if (myDataTable.BooleanField && (myDataTable.TextField == null || !myDataT
able.TextField.StartsWith("Inhalt")))
    {
        myDataTable.ModelState.AddModelError(nameof(OrmMyDataTable.TextFiel
d), "[INSERT TRANSLATION GUID]".Translate());
        baseValidationResult = false;
    }

    if (baseValidationResult)
        myDataTable.BooleanField = false;
}
```

5.9.3. Teil-Datensätze

Werden Teil-Datensätze hinzugefügt oder entfernt werden ebenfalls Events ausgelöst.

```
public override void OnSubRecordAdded(OrmBABase orm, string propertyName, OrmS
ubRecordBase subRecord)
{
    OrmMyDataTable myDataTable = (OrmMyDataTable)orm;
    if (propertyName == nameof(OrmMyDataTable.SubDatas))
    {
        OrmSubDataTable subData = (OrmSubDataTable)subRecord;
    }
}

public override void OnSubRecordRemoved(OrmBABase orm, string propertyName, Or
mSubRecordBase subRecord)
{
    OrmMyDataTable myDataTable = (OrmMyDataTable)orm;
    if (propertyName == nameof(OrmMyDataTable.SubDatas))
    {
        OrmSubDataTable subData = (OrmSubDataTable)subRecord;
    }
}
```

5.9.4. Feldänderungen

Bei der Änderung einzelner Feldwerte wird ebenfalls ein Event pro Feld gesendet. Dies wird natürlich nicht bei Änderungen in UI gesendet, sondern nur wenn der Datensatz zum Server gesendet wird.

```
public override void OnFieldChanged(OrmBABase orm, string propertyName, objec
t oldValue, object newValue)
{
    OrmMyDataTable myDataTable = (OrmMyDataTable)orm;
    if (propertyName == nameof(OrmMyDataTable.TextField))
        myDataTable.TextField = "Inhalt: " + myDataTable.TextField;
}
```

Auch für Änderungen von Feldwerten in Teil-Datensätzen werden Events gesendet.

```
public override void OnSubRecordFieldChanged(OrmBABase orm, OrmSubRecordBase s
ubTable, string propertyName, string subTablePropertyName, int entryIndex, obj
ect oldValue, object newValue)
{
    OrmMyDataTable myDataTable = (OrmMyDataTable)orm;
    if (propertyName == nameof(OrmMyDataTable.SubDatas))
    {
        OrmSubDataTable subData = (OrmSubDataTable)subTable;
        if (subTablePropertyName == nameof(OrmSubDataTable.TextField))
            subData.TextField = "Inhalt: " + subData.TextField;
    }
}
```

5.9.5. Import Life Cycle

Sollen Life Cycle Events auch beim Import ausgeführt werden, müssen explizit die entsprechenden Events implementiert werden. Bzw. sollten spezielle Events implementiert werden, um den Import möglichst Performance optimiert durchführen zu können.

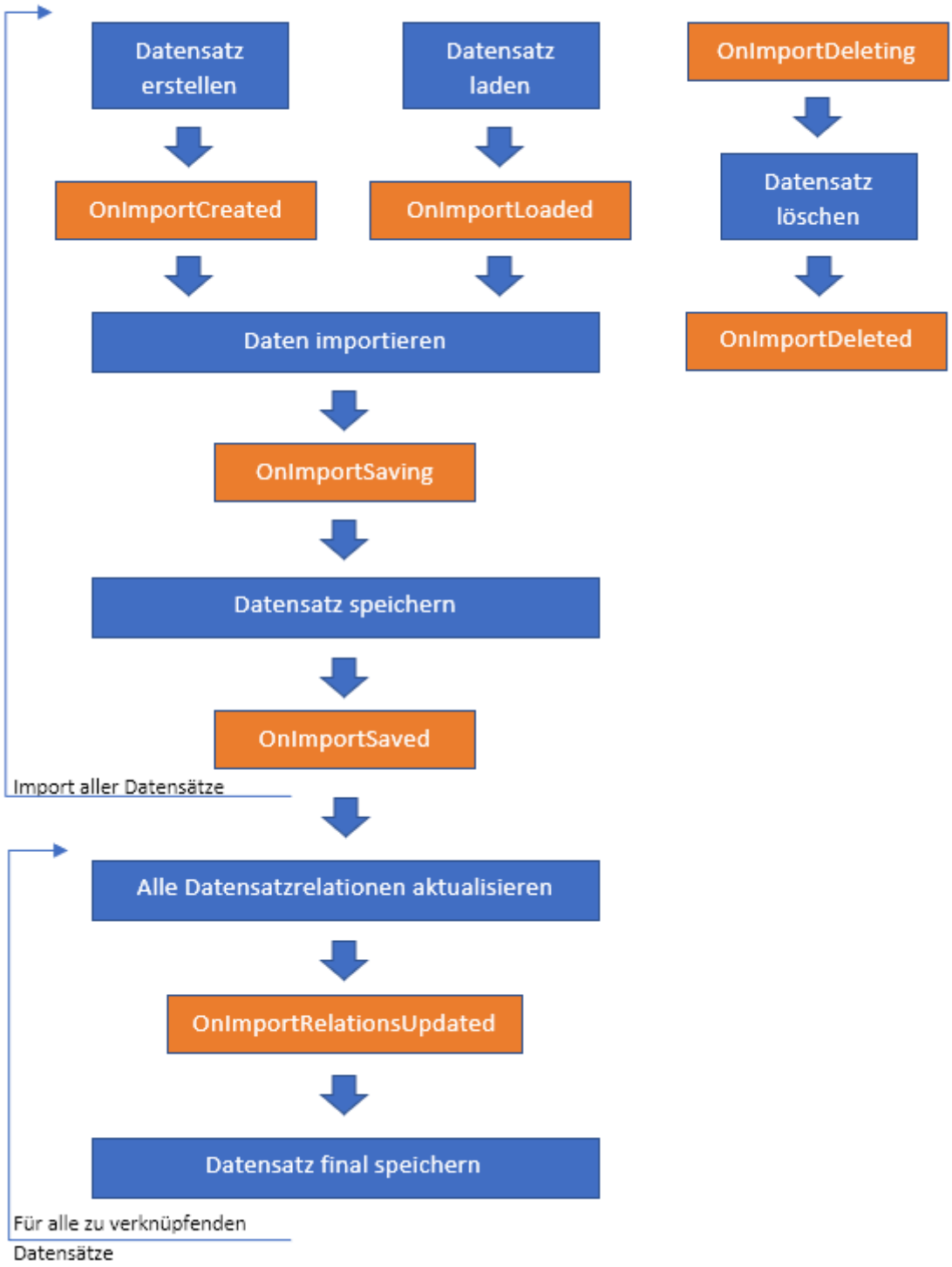
```
public override void OnImportCreated(OrmBABase orm)
{
    OnCreated(orm);
}
```

Folgende Events stehen zur Verfügung

- OnImportCreated(OrmBABase orm)
  Wird ausgeführt, wenn der Importprozess einen neuen Datensatz erstellt hat, um Daten in diesen zu importieren.
- OnImportLoaded(OrmBABase orm)
  Wird ausgeführt, wenn der Importprozess einen existierenden Datensatz aus der Datenbank geladen hat, um Daten in diesen zu importieren.
- OnImportRelationsUpdated(OrmBABase orm)
  Wird ausgeführt, wenn der Importprozess den Import von Daten erfolgreich abgeschlossen und abschließend Änderungen an den Relationen eines Datensatzes durchgeführt hat. Der Aufruf geschieht direkt vor dem abschließenden Speichern des Datensatzes. Der Aufruf erfolgt nicht, wenn keine Relationen geändert wurden.
  Hinweis: Die Relationen des Datensatzes sind hier aktuell, Änderungen am Datensatz werden im nächsten Schritt auch mit gespeichert, RelatedData ist allerdings hier NOCH NICHT aktuell, da in einigen Fällen UpdateRelatedData() auch das Speichern übernimmt und der Event daher vor dem entsprechenden Aufruf gefeuert werden muss.
- OnImportSaving(OrmBABase orm)
  Wird ausgeführt, wenn der Importprozess den Import von Daten abgeschlossen hat und den Datensatz als nächstes speichern möchte.
- OnImportSaved(OrmBABase orm)
  Wird ausgeführt, wenn der Importprozess den Import von Daten abgeschlossen hat und den Datensatz erfolgreich gespeichert hat.
- OnImportDeleting(OrmBABase orm)
  Wird ausgeführt, wenn der Importprozess einen Datensatz auf Basis einer „Record_Deletions.csv“ Datei als nächstes löschen möchte.
- OnImportDeleted(OrmBABase orm)
  Wird ausgeführt, wenn der Importprozess einen Datensatz auf Basis einer „Record_Deletions.csv“ Datei erfolgreich gelöscht hat.

Der Import ist so gestaltet, dass das zuerst alle Datensätze erstellt werden und am Ende die Relationen zwischen den Datensätzen.

Ablauf eines Imports mit Aufruf der entsprechenden Events



5.9.6. Deaktivierung von Events

Manchmal ist es notwendig Events zu deaktivieren, da man beispielsweise das Event neu implementieren möchte. Dazu kann man das Attribute um eine Blacklist erweitern. In diesem Beispiel wird verhindert, dass der aktuelle Anwender in die Autoren-Relation eingefügt wird. Auch dann, wenn dies Konfiguriert ist.

```
[BAOrmEvents(typeof(OrmMyDataTable), Blacklist = new[] { typeof(CreateAuthorRe
lationOfOrmBABase), typeof(CreateAuthorRelationOfOrmBABaseOnImport) })]
public class OrmEventModifyMyDataTable : BAOrmEventsBase {}
```

Die im Standard implementierten Events befinden sich pro Modul in den jeweiligen Namespaces.

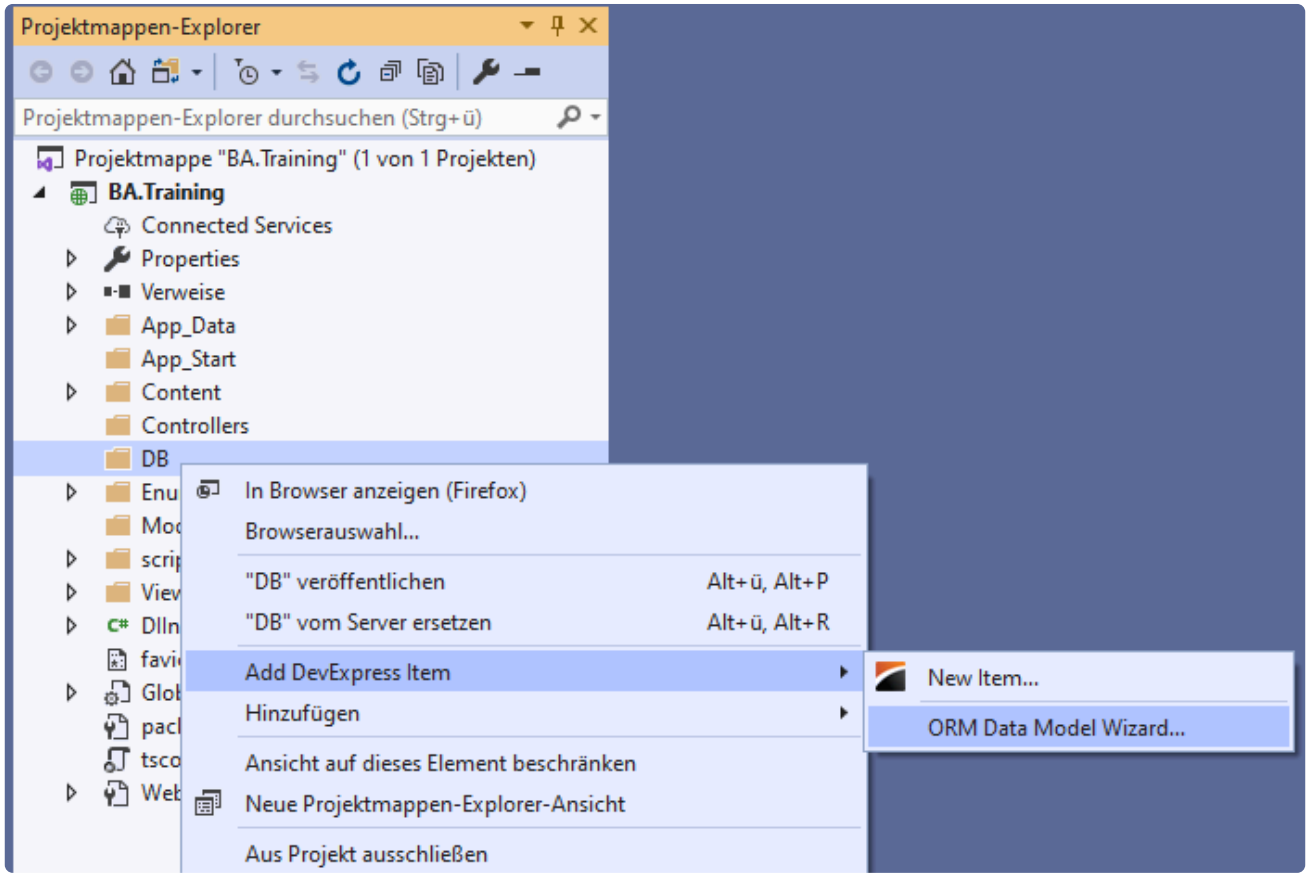
- BA.Core.OrmEvents
- BA.Activity.OrmEvents
- BA.Contact.OrmEvents
- etc.

5.10. Übung 2

In dieser Übung wird eine Datentabelle und eine Teil-Datentabelle erstellt, die Anwendung wird Konfiguriert und es wird ein Event und ein Validator implementiert

XPO Wizard einrichten

Erstellen sie einen neuen Ordner “DB” und fügen Sie einen XPO Wizard hinzu.



Es wird keine Verbindung zu einer Datenbank hergestellt.

Modifizieren Sie den “Namespace” (Rechter Klick / Eigenschaften auf die Fläche), beispielsweise auf “BA.Training.DB”.

Fügen sie die Basis-Klasse `OrmBABase` hinzu. Dazu wählen Sie “Add Assembly”, wechseln Sie in den “bin” Ordner (Anwendung muss mindestens einmal erstellt worden sein) und wählen dort die BA.dll aus.

Neue Datentabelle

Erstellen Sie eine neue Datentabelle (Toolbox/Persistent Object), die die `OrmBABase` erweitert. Als Name setzen Sie “OrmEngine”.

Fügen Sie folgende Felder (Persistent Property) hinzu.

- `Name` String Länge 100 (default).
  Setzen sie das Validator Attribut “BA.Core.CustomAttributes.Validations.BARequired”
- `Type` Auswahlliste “Engine Types”
- `Responsible` String Länge 200
- `DepreciatedCost` Boolean
- `ImplementingDate` Datumswert
- `MaintenanceInterval` Wiederholendes Datum

Neue Teil-Datentabelle

Erstellen Sie eine Neue Teil-Datentabelle “OrmSubService”.

- Fügen Sie die BA Tabelle `OrmSubBABase` dem Wizard hinzu
- Erstellen Sie die Teil-Datentabelle und erweitern Sie `OrmSubBABase`
- Fügen Sie zwei Felder hinzu
  - `ServiceDate` als Datumsfeld
  - `Remark` als String mit Länge 1.000.

Fügen Sie die neue Teil-Datentabelle als Feld “Services” zur Maschine hinzu. Die verwendete Teil-Datentabelle (Eigenschaft) erhält den Namen “OrmSubServices”.

Grundkonfiguration

Konfigurieren Sie die Anwendung

- Belegen Sie das gemeinsame Feld “EntityTitle” mit “[Name]”
- Legen Sie eine Ribbon bar Navigation für die “Engine” Ansicht an.
  - Aktion Ansicht aktualisieren
- Legen Sie eine Ribbon bar Navigation für die “Engine” Maske an.
  - Aktion Bearbeiten
  - Aktion Speichern
  - Aktion Speichern und Schließen
  - Aktion Lesen
  - Aktion Datensatzprotokoll
- Legen Sie eine Maske für “Engine” an
  - Legen Sie die Felder an
  - Legen Sie die Maske in der Datentabelle als Vorgabemaske fest.
- Legen Sie eine Ansicht für “Engine” an
  - Legen Sie die Felder an
- Legen Sie eine Applikationsnavigation an.
  - Mit der “Engine” Ansicht
  - Tragen Sie die Navigation in den Anwendungseinstellungen ein.
- Fügen sie der Ansicht- und der Maskennavigation eine Aktion Erstellen Maschine (Icon: “industrial_machine”) hinzu
- Starten Sie die Anwendung neu

Feldvorbelegung durch Event

Erstellen Sie einen neuen Ordner “OrmEvents” und erstellen Sie eine neue Event Klasse `SetDefaultEngineType`

Belegen Sie das Feld `Type` mit einem Wert als Vorgabe.

Stellen Sie in der Event Klasse sicher das `Type` einen Wert hat und setzen Sie im Fehler Fall einen Validierungsfehler

Programmatischer Validator

Erstellen Sie einen Ordner “CustomAttributes” und dort drunter einen weiteren Ordner “Validations”.

Dort erstellen Sie ein eigenes Validierungsattribut, um zu prüfen ob `ImplementingDate` im Jahr 2021 oder Später liegt.

Werfen Sie einen Validierungsfehler, wenn dies nicht der Fall ist.

Setzen sie eine Drag & Drop Regel zwischen `OrmDateTimeField` und `YearAndLaterValidator` für `EnumConfigurationType.OrmEntityConfigurationGuid`

Konfigurieren Sie im Designer den Validator an dem Feld `MaintenanceInterval`

Validator Steuerelement

Der Validator ist schon aktiv, wird aber in der Konfiguration nicht angezeigt. Dafür wird optional ein Steuerelement benötigt.

Implementieren Sie ein Steuerelement für den Validator und lassen sie das Jahr konfigurieren. Setzen sie für das Jahr selbst einen Validator, so dass man nur Zahlen zwischen 2021 und 2100 eintragen kann.

Erstellen Sie einen Ordner “Configuration” und dort einen weiteren Ordner “Validators”. Erstellen Sie dort das Steuerelement.

Fügen Sie im Validator-Attribut die Methode `CreateValidatorControl()` hinzu, um das in der Datentabelle gesetzt Attribut im Designer anzuzeigen.

Lösung

6. Relationen

Relationen zwischen Datensätzen bauen Abhängigkeiten zwischen diesen Datensätzen auf. Dabei wird zwischen flachen und hierarchischen Relationen unterschieden. In der Regel sind die Relationen flach. Beispiel: “Der Betreuer einer Firma” oder “Die Vorlage eines Datensatzes”. Das System kennt zurzeit zwei hierarchische Relationen. Die “Eltern” Relation und die “Rollenmitgliedschaft”.

Die Relationstypen sind die technische Basis der Relationen. Alle gespeicherten Relationen haben einen entsprechenden Typen. Im Gegensatz dazu sind die Relationsdefinitionen nur eine virtuelle Ausprägung eines Typen, in einer Datentabelle. Damit sind die Relationsdefinitionen beliebig änderbar und in den Datentabellen können auch mehrere Definitionen pro Typ angelegt werden.

Beispiel:
Relationstyp: “Betreuer”
Relationsdefinition in der Firma; “Betreuer der Firma”
Relationsdefinition in der Verkaufschance; “Betreuer der Verkaufschance”

In einer Relation wird zwischen Quellen und Zielen unterschieden. Als Eselsbrücke, was Quellen und was Ziele sind, dient die Vorstellung, dass man in der Maske des Ziels die Quelle wählt und in den Masken der Quelle die Ziele in Detailansichten dargestellt werden. Beispielsweise könnte man in einer Firma (Ziel) den Betreuer (Quelle) wählen und in dem Betreuer könnte man eine Detailansicht mit allen betreuten Firmen anzeigen.

6.1. Relationstypen

Um Relationstypen programmatisch anzulegen muss zuerst eine Erweiterung der Auswahlliste `EnumRelationType` vorgenommen werden.

```
[EnumExtension(typeof(EnumRelationType))]
public static class EnumRelationTypeExtension
{
    public const string MyRelationGuid = "[INSERT RELATION TYPE GUID]";
    public static readonly EnumRelationType MyRelation = new EnumRelationTyp
e(MyRelationGuid, 1000, "[INSERT TRANSLATION GUID]", nameof(MyRelation));
}
```

Die Übersetzung des Auswahlwertes sollte in allen Sprachen den englischen Text beinhalten, um in der Konfigurationsauswahl jederzeit konsistent zu sein.

Anschließend legt man eine Klasse mit dem Interface `IRelationCreation` im Ordner “Configuration” an. Beim Start des Systems werden diese Relationen automatisch angelegt. Die neuen Relationstypen müssen gut geplant werden, da diese nicht mehr verändert werden können.

```
public class RelationCreation : IRelationCreation
{
    public void CreateRelationConfigurations(KeyedDictionary<Guid, RelationTyp
eConfiguration> relationConfigurations)
    {
        relationConfigurations.Add(MyRelation());
    }

    private RelationTypeConfiguration MyRelation()
    {
        RelationTypeConfiguration relationConfiguration = new RelationTypeConf
iguration()
        {
            Id = EnumRelationTypeExtension.MyRelation,
            Type = EnumRelationCardinality.AnythingToAnythingGuid,
        };

        relationConfiguration.Initialize();

        relationConfiguration.ConfigurationName = "My Relation";
        relationConfiguration.Description = "My Relation";
        relationConfiguration.ShowInDesigner = true;
```

```
        return relationConfiguration;
    }
}
```

EnumRelationCardinality

Der `RelationTypeConfiguration.Type` definiert, zwischen welchen Datentabellen eine Relation dieses Typs erstellt werden kann.

AnythingToAnything

Bedeutet das sowohl alle Datentabellen als Quellen als auch als Ziele verwendet werden können.

SomethingToAnything

Bedeutet, dass als Quelle nur bestimmte Datentabellen genutzt werden können aber als Ziele alle. In diesem Fall muss für die in Frage kommenden Quellen noch folgende Zeilen der Relation nach der Initialisierung hinzugefügt werden. Falls als Betreuer nur Benutzer in Frage kommen, fügt man dies hinzu.

```
relationConfiguration.SourceTypes.Add(EnumDataSource.UserProfileGuid);
```

Können neben Betreuern auch Rollen gewählt werden (Die Benutzer selbst basieren ebenfalls auf der Basis-Datentabelle Basis.Rollen), kann die entsprechende Basis-Datentabelle gesetzt werden.

```
relationConfiguration.SourceTypes.Add(EnumDataSource.RoleBaseGuid);
```

Können neben Benutzern auch Kontakte als Betreuer gewählt werden, fügt man zwei Quellen der Relation hinzu.

```
relationConfiguration.SourceTypes.Add(EnumDataSource.UserProfileGuid);
relationConfiguration.SourceTypes.Add(EnumDataSourceExtension.ContactGuid);
```

AnythingToSomething

Bedeutet das als Quelle alle Datentabellen genutzt werden können aber als Ziele nur bestimmte. In diesem Fall muss für die in Frage kommenden Ziele noch folgende Zeile der Relation nach der Initialisierung hinzugefügt werden. Falls die Betreuer nur Firmen betreuen können, fügt man dies hinzu.

```
relationConfiguration.TargetTypes.Add(EnumDataSourceExtension.CompanyGuid);
```

Wie im vorherigen Abschnitt können auch Basis-Datentabellen oder mehrere Datentabellen gesetzt werden.

SomethingToSomething

Wenn sowohl die Quell-Datentabellen als auch die Ziel-Datentabellen bestimmte sein sollen, werden sowohl die Quellen als auch die Ziele konkret gesetzt. Beispielsweise der Betreuer einer Firma.

```
relationConfiguration.SourceTypes.Add(EnumDataSource.UserProfileGuid);
relationConfiguration.TargetTypes.Add(EnumDataSourceExtension.CompanyGuid);
```

Auch in diesem Fall können Basis-Datentabellen oder mehrere Datentabellen sowohl als Quelle als auch als Ziel gesetzt werden.

Relationskategorie

Um Relationskategorien in der Relation zu nutzen, muss eine Auswahlliste angelegt werden und der Relation zugeordnet werden.

```
relationConfiguration.RelationCategoryEnum = EnumMyRelationCategory.Guid
```

ShowInDesigner

Sollen die Quellen und Ziele der Relationen in der Konfiguration geändert werden können, dann sollte man `ShowInDesigner` auf `true` setzen, in anderen Fällen auf `false`.

TrackHierarchy

Falls die eigene Relation eine hierarchische Relation sein soll, muss `TrackHierarchy` auf `true` gesetzt werden. Die Anlage einer eigenen hierarchischen Relation sollte genau überlegt werden, da dafür ein erhöhter Verwaltungsaufwand notwendig ist. Eine solche Relation legt man dann an, wenn über mehrere Ebenen über diese Relation auf Quellen zugegriffen werden soll.

Die “Eltern” Relation aus dem System ist beispielsweise eine solche Relation. Damit ist es möglich auf allen Ebenen beispielsweise auf die Firma oder den Kontakt zuzugreifen.

Firma -> Kontakt -> Vorgang 1 -> Vorgang 2

Sowohl im Kontakt als auch in beiden Vorgängen, kann nun in Masken und Ansichten der Firmenname angezeigt werden. In den beiden Vorgängen auch der Kontaktname.

6.2. Relationsdefinitionen

Relationsdefinitionen basieren auf den Relationstypen und stellen die Sicht aus einer Datentabelle dar. Da technisch nur die Relationstypen abgespeichert werden, sind die Definitionen nur eine flexible Sicht darauf und können auch nachträglich geändert werden, wobei in dem Fall alle Stellen der Konfigurationen oder Formeln, an denen die Definitionen genutzt wurden umgestellt werden müssen. Es sind auch mehrere Definitionen auf Basis des gleichen Typs in einer Datentabelle möglich.

Relationsdefinitionen können per Konfiguration in Zieltabellen festgelegt werden. Diese Konfiguration verweist auf die möglichen Quelltabellen.
Technisch werden aus diesen Konfigurationen beim Neustarten des Systems Eigenschaften der Datentabellen, die beispielsweise in Reports direkt genutzt werden können.
Diese Eigenschaften können in der Programmierung auch direkt hinterlegt werden. Dies ist beispielsweise sinnvoll, wenn bestimmte Dinge unveränderlich sein sollen.
Eine Relationsdefinition hat immer ein Ziel und optional mehrere Quellen. Einer implementierten Eigenschaft in einem Ziel, können deshalb N Implementierungen in den Quellen zugeordnet sein. Die Namen der Eigenschaften in den Quellen müssen aber übereinstimmen.

Programmierte Relationsdefinitionen können ferner eine Basis-Datentabelle als Ziel haben, konfigurierte können das derzeit nicht.

Implementierung der Eigenschaft im Ziel

Man kann zwei Arten der Implementierung unterscheiden: Anzahl der maximalen Quelldatensätze gleich Eins und größer Eins. Der Unterschied liegt darin, dass die Eigenschaft entweder einen oder keinen Datensatz zurückliefert, bzw. wenn mehr als einer möglich ist, wird grundsätzlich eine Liste an Datensätze zurückgegeben.

Beispiel 1

Diese Eigenschaft wird in der Orm Klasse abgelegt. Beispielseweise in #OrmMyDataTable#
Maximale Anzahl == 1, Keine Relationskategorie

```
[Newtonsoft.Json.JsonIgnore]
[DontShowFieldInDesigner]
[RelationDefinitionTarget("[INSERT RELATION DEFINITION GUID EXAMPLE 1]", EnumR
elationTypeExtension.MyRelationGuid, MinCount = 0, MaxCount = 1)]
public OrmRoleBase RelatedResponsible
{
    get {
        return GetSourcePrimary<OrmRoleBase>(EnumRelationTypeExtension.MyRelat
ion);
    }
}
```

Beispiel 2

Diese Eigenschaft wird in der Orm Klasse abgelegt. Beispielseweise in #OrmMyDataTable#
Maximale Anzahl > 1, mit Relationskategorie

```
[Newtonsoft.Json.JsonIgnore]
[DontShowFieldInDesigner]
[RelationDefinitionTarget("[INSERT RELATION DEFINITION GUID EXAMPLE 2]", EnumR
elationTypeExtension.MyRelationGuid, EnumMyRelationCategory.FirstValueGuid, Mi
nCount = 0, MaxCount = 0)]
public IQueryable<OrmContact> RelatedContacts
{
    get {
        return GetSources<OrmContact>(EnumRelationTypeExtension.MyRelation, En
umMyRelationCategory.FirstValueGuid);
    }
}
```

Die erste Guid definiert einen eindeutigen Schlüssel zur Identifizierung der Relationsdefinition. Die Kombination des Attributes [RelationDefinitionTarget] und einer Guid, darf in dem System nur einmal vorkommen!

Als Objekttyp für die Rückgabe sollte man immer die Datentabelle nehmen, die die möglichen Quellen am besten gemeinsam beschreibt. Man sollte also nur dann OrmBABase nehmen, wenn die Datentabellen keinen anderen gemeinsamen Objekttyp besitzen.

Implementierung der Eigenschaften an den Quellen

Die Implementierung in einem Ziel kann pro Guid nur einmal erfolgen. Die Implementierung im Ziel, welche hier beschrieben wurde, kann in mehreren Quellen stattfinden, sollte aber mindestens einmal vorhanden sein. Über die angegebene Guid werden die Implementierungen entsprechend verknüpft. Die beiden folgenden Beispiele korrespondieren zu den beiden Beispielen im vorherigen Kapitel.

Beispiel 1

Diese Eigenschaft wird in der Orm Klasse abgelegt. Beispielseweise in #OrmUserProfile#.

```
[Newtonsoft.Json.JsonIgnore]
[DontShowFieldInDesigner]
[RelationDefinitionSource("[INSERT RELATION DEFINITION GUID EXAMPLE 1]")]
public IQueryable<OrmCompany> RelatedResponsible
```

```
{
    get {
        return GetTargets<OrmCompany>(EnumRelationTypeExtension.MyRelation);
    }
}
```

Beispiel 2

```
[Newtonsoft.Json.JsonIgnore]
[DontShowFieldInDesigner]
[RelationDefinitionSource("[INSERT RELATION DEFINITION GUID EXAMPLE 2]")]
public IQueryable<OrmCompany> RelatedCompanies
{
    get {
        return GetTargets<OrmCompany>(EnumRelationTypeExtension.MyRelation, En
umMyRelationCategory.FirstValueGuid);
    }
}
```

Die Quellen können immer mehrere Ziele zurückliefern, daher werden grundsätzlich Listen zurückgegeben.

Implementierung über Inject-Attribute

Neben der direkten Implementierung der Eigenschaften, kann man mit diesen Attributen auch Relationsdefinitionen in Datentabellen, die nicht im eigenen Modul liegen, hinzufügen. Das kann Quelle und/oder Ziel einer Relationsdefinition betreffen.
Die Attribute `RelationDefinitionSourceInject` und `RelationDefinitionTargetInject` verhalten sich genauso wie `RelationDefinitionSource` bzw. `RelationDefinitionTarget`, nur dass sie auf Assembly-Ebene definiert werden und zusätzlich die Angabe der Klasse und des Namens für die zu erzeugende Eigenschaft enthalten.
Es sind auch Basis-Datentabellen als Quelle oder Ziele zulässig.

Beispiel 1

Nur die Target-Seite wird in die Datentabelle des Kontaktes eingefügt.

```
[assembly: RelationDefinitionTargetInject(typeof(OrmContact), "RelatedConforma
tion", "[INSERT RELATION DEFINITION GUID]", EnumRelationTypeExtension.MyRelati
on, MinCount = 0, MaxCount = 1, Validate = false)]
```

Beispiel 2

Nur die Quelle wird in der Datentabelle der Telefonnotiz und der Email eingefügt.

```
[assembly: RelationDefinitionSourceInject(typeof(OrmPhoneCall), "RelatedConfir
medContacts", "[INSERT RELATION DEFINITION GUID]")]
[assembly: RelationDefinitionSourceInject(typeof(OrmEmail), "RelatedConfirmedC
ontacts", "[INSERT RELATION DEFINITION GUID]")]
```

Beispiel 3

Beide Attribute werden in einem Vorgang eingefügt, ähnlich in Beispiel 1 und Beispiel 2, nur diesmal in alle Aktivitäten.

```
[assembly: RelationDefinitionTargetInject(typeof(OrmContact), "RelatedConforma
tion", "[INSERT RELATION DEFINITION GUID]", EnumRelationType.ConformationGui
d, MinCount = 0, MaxCount = 1, Validate = false)]
[assembly: RelationDefinitionSourceInject(typeof(OrmActivityBase), "RelatedCon
firmedContacts", "[INSERT RELATION DEFINITION GUID]")]
```

6.3. Erstellen und Entfernen

Relationsdaten

Um Relationen erstellen zu können werden die Relationsdaten der Quelle benötigt. Dazu stehen eine Reihe von Methoden in `Api.ORM` zur Verfügung. Es gibt Methoden für einzelne Datensätze.

```
Guid otherOrmGuid = "[INSERT A RECORD GUID]".ToGuid();
RelationData relationDataGuid = Api.ORM.GetRelationDataModel(session, otherOrm
Guid);
// Falls man das Orm zur Verfügung hat. Ansonsten ist dies langsamer
OrmMyDataTable otherOrm = Api.ORM.GetOrm<OrmMyDataTable>(otherOrmGuid, sessio
n);
RelationData relationDataOrm = Api.ORM.GetRelationDataModel(otherOrm);
```

Es gibt Methoden für Abfragen und Listen von Guids.

```
IQueryable<OrmMyDataTable> queryWith = Api.ORM.GetQueryWithReadPermissions<Orm
MyDataTable>(session);
queryWith = queryWith.Where(ff => ff.TextField != null && ff.TextField.Contain
s("Inhalt"));
IQueryable<RelationData> relationDatasQuery = Api.ORM.GetRelationDataModels(qu
eryWith);
IQueryable<RelationData> relationDataGuids = Api.ORM.GetRelationDataModels<Orm
ActivityBase>(session, new[] { guid1, guid2 });
```

Erstellen

Relationen werden auf dem Ziel-Datensatz erstellt. In diesem Beispiel wird keine Relationskategorie angegeben.

```
newMyData.AddSource(relationDataOrm, EnumRelationTypeExtension.MyRelation, nul
l);
newMyData.Save();
```

Falls der Ziel-Datensatz erst geladen werden müsste, kann alternativ eine `Api.Relation` Methode genutzt werden.

```
OrmRelation relation = Api.Relation.CreateRelation(session, relationDataOrm, r
elationDataNewMyData, EnumRelationTypeExtension.MyRelation, null);
relation.Save();
```

Falls möglich sollte diese Methode in der Massenverarbeitung eingesetzt werden.

Entfernen

Das Entfernen funktioniert identisch, außer dass dafür keine Relations-Daten benötigt werden. Es genügt die Guid des Quell-Datensatzes

```
newMyData.RemoveSource(otherOrmGuid, EnumRelationTypeExtension.MyRelation, nul
l);
newMyData.Save();
```

Auch zum Entfernen gibt es eine Methode in der `Api.Relation`, die für die Massenverarbeitung optimal ist.

```
int deletedCount = Api.Relation.DeleteAllRelations(session, otherOrmGuid, newM
yData.Oid, EnumRelationTypeExtension.MyRelation);
```

6.4. Abfragen

Zum Abfragen von in Relation stehenden Datensätzen nutzt man sinnvollerweise die Relationsdefinitionen.

```
IQueryable<OrmBABase> parents = newMyData.RelatedParents;
IQueryable<OrmBABase> children = newMyData.RelatedChildren;
OrmRoleBase createdBy = newMyData.RelatedCreatedBy;
```

Auf einem Datensatz kann man die Quellen und die Ziele auch unabhängig von der Definition abfragen. Mit der Angabe einer Datentabelle oder Basis-Datentabelle schränkt man die Ergebnismenge auf diesen Typen ein. Falls man keine Kenntnisse über die beteiligten Datentabellen hat, wird. `OrmBABase` angegeben.

```
OrmCompany company = newMyData.GetSourcePrimary<OrmCompany>(EnumRelationTypeEx
tension.MyRelation);
Guid? companyOid = newMyData.GetSourcePrimaryOid(EnumRelationTypeExtension.MyR
elation);
IQueryable<OrmBABase> sources = newMyData.GetSources<OrmBABase>(EnumRelationTy
peExtension.MyRelation);
List<Guid> sourcesOids = newMyData.GetSourcesOids(EnumRelationTypeExtension.My
Relation);
```

Bei der Abfrage von Zielen wird immer eine Liste von Datensätzen zurückgegeben.

```
IQueryable<OrmMyDataTable> targets = company.GetTargets<OrmMyDataTable>(EnumRe
lationTypeExtension.MyRelation);
```

Unabhängig vom geladenen Datensatz kann man die in Relation stehenden Datensätze auch über die `Api.Relation` abfragen.

```
OrmCompany company = Api.Relation.GetSourcePrimary<OrmCompany>(newMyData.Oid,
session, EnumRelationTypeExtension.MyRelation);
IQueryable<OrmBABase> sources = Api.Relation.GetSources<OrmBABase>(newMyData.O
id, session, EnumRelationTypeExtension.MyRelation);
IQueryable<OrmMyDataTable> targets = Api.Relation.GetTargets<OrmMyDataTable>(c
ompany.Oid, session, EnumRelationTypeExtension.MyRelation);
```

Bei den Abfragen für die Quellen und Ziele können zusätzlich die Relationskategorie(n), Aktivstatus und ob die Leserechte nicht berücksichtigt werden angeben werden.

6.5. Hierarchieabfragen

Hierarchische Relationen erlauben performante Abfragen über die Hierarchie. In der `Api.Relation` gibt es Methoden um Quelldatensätze über die gesamte Hierarchiezu finden.

Der erste Quelldatensatz bestimmter Typen

```
OrmBABase firstCompany = Api.Relation.GetFirstSourceOfType(EnumRelationType.Pa
rent, BA.Contact.Enums.Extensions.EnumDataSourceExtension.Company, null, sessi
on, newMyData.Oid);
OrmBABase firstCompanyOrContact = Api.Relation.GetFirstSourceOfType(EnumRelati
onType.Parent, new Guid[] { BA.Contact.Enums.Extensions.EnumDataSourceExtensio
n.Company, BA.Contact.Enums.Extensions.EnumDataSourceExtension.Contact }, nul
l, session, newMyData.Oid);
```

Der oberste Quelldatensatz bestimmter Typen

```
OrmBABase topCompany = Api.Relation.GetTopSourceOfType(EnumRelationType.Paren
t, BA.Contact.Enums.Extensions.EnumDataSourceExtension.Company, null, sessio
n, newMyData.Oid);
OrmBABase topCompanyOrContact = Api.Relation.GetTopSourceOfType(EnumRelationTy
pe.Parent, new Guid[] { BA.Contact.Enums.Extensions.EnumDataSourceExtension.Co
mpany, BA.Contact.Enums.Extensions.EnumDataSourceExtension.Contact }, null, se
ssion, newMyData.Oid);
```

Für komplexere Hierarchieabfragen kann man eine Abfrage auf die Hierarchietabelle selbst erhalten.

```
IQueryable<OrmHierarchyRelationBase> hierarchyRelationQuery = RelationTools.Ge
tHierarchyRelationsQuery(EnumRelationType.Parent, session);
```

Ein Beispiel wie man alle Anrufe erhält, die als Kinder und Kindeskinder zu dem aktuellen Datensatz in der gesamten Hierarchie zu finden sind.

```
IQueryable<Guid> phonecallsOids = hierarchyRelationQuery.Where(ff => ff.Sourc
e == newMyData.Oid && ff.TargetType == EnumDataSource.PhoneCall).Select(ff =>
ff.Target);
IQueryable<OrmPhoneCall> phonecallsWithRead = Api.ORM.GetQueryWithReadPermissi
ons<OrmPhoneCall>(session);
IQueryable<OrmPhoneCall> phonecalls = phonecallsWithRead.Where(ff => phonecall
sOids.Contains(ff.Oid));
```

6.6. Übung 3

In dieser Übung definieren wir einen eigenen Relationstyp “Responsible” und nutzen die Parent (Eltern) Relation, um eine Hierarchie der Maschinen aufzubauen. Dafür erweitern wir die Relationskategorie um einen eigenen Untertypen.

Relationstyp “Responsible”

Legen Sie einen Relationstyp “Responsible” an. Als Quelle werden die Benutzerprofile (`OrmUserProfile`) und als Ziel beliebige Datentypen festgelegt.

Relationsdefinition “Responsible”

Programmieren Sie eine Relationsdefinition “Responsible” an der Engine (Ziel), die die Auswahl eines Benutzers (Quelle) zulässt.

Über die Inject-Möglichkeit fügen Sie diese Relationsdefinition dem Benutzerprofil hinzu.

Relationsdefinition “EnginePart”

Erweitern sie die Auswahlliste “Parent Relation Subtype” `EnumParentRelationSubtypes`, um eine Kategorie “Engine Part”.

Programmieren Sie eine Relationsdefinition “EnginePart” an der Engine (Ziel), die die Auswahl mehrerer Engines (Quelle) zulässt, wobei beide Teile der Relationsdefinition in der Engine hinterlegt werden.

Konfigurieren der Maske

Erweitern Sie die Konfiguration der Maske, um die Auswahl eines Betreuers und die Auswahl mehrerer übergeordneter Engines. Zusätzlich wird eine Detailansicht hinzugefügt, die alle untergeordneten Engines anzeigt.

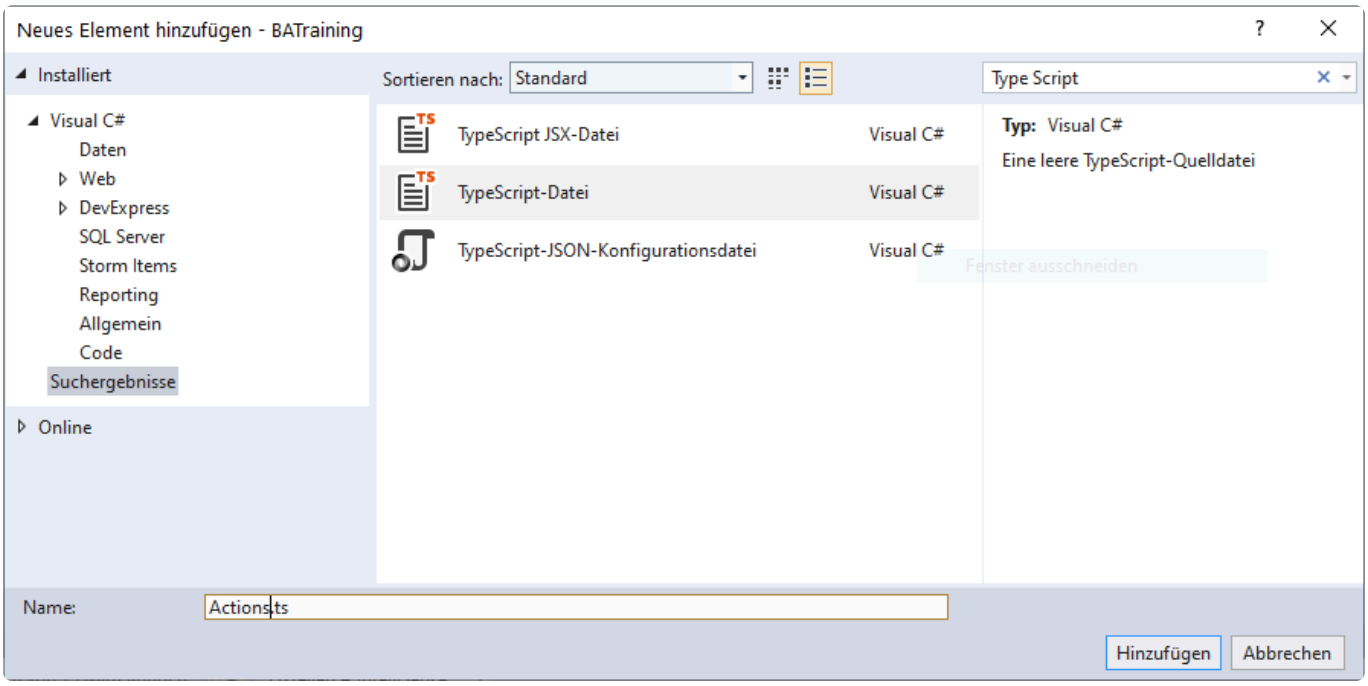
Lösung

7. Type Script

Alle Client-Funktionalitäten innerhalb von BA sind in Type Script implementiert. Type Script bietet eine deutlich verbesserte Typisierung als Java Script und die BA-NuGet-Pakete beinhalten die *d.ts der BA-Bibliotheken. Daher wird empfohlen in den Projekten ebenfalls Type Script zu verwenden. Es ist aber möglich stattdessen Java Script zu verwenden. Die Beispiele und die Erklärungen werden in diesem Handbuch in Type Script verfasst.

7.1. Eigene Bibliothek

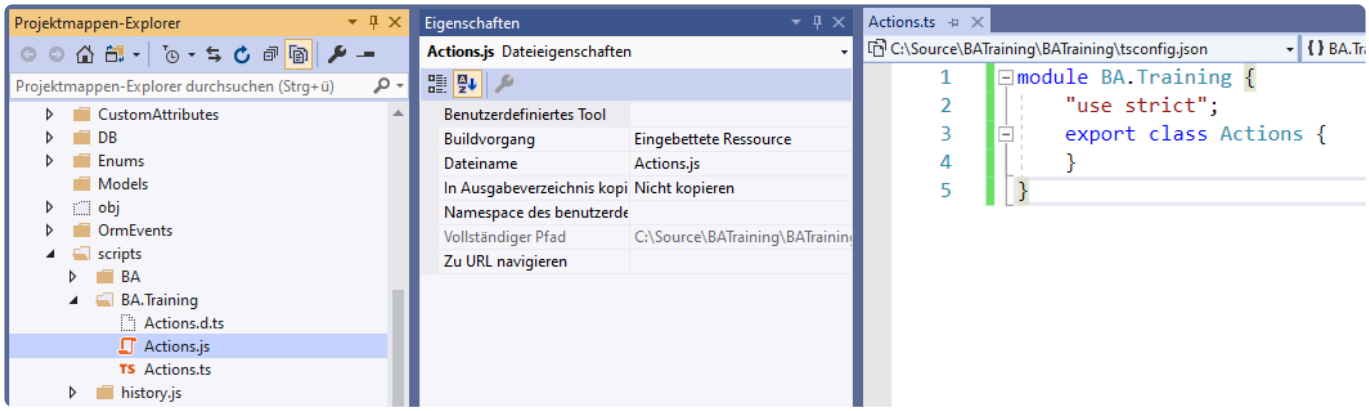
Zur Anlage und Integration der ersten Bibliothek sollte ein eigener Ordner unterhalb des “Scripts” Ordner angelegt werden. In diesem Ordner wird die neue Type Script Bibliothek erstellt.



In der Datei wird das `module` und die erste Klasse definiert.

```
module BA.Training {
    "use strict";
    export class Actions {
    }
}
```

Nun wird das Projekt kompiliert, damit die entsprechenden *.js und *.d.ts Dateien erzeugt werden. Es sollte der Modus “Alle Dateien anzeigen” aktiv sein. Falls die erzeugten Dateien nicht sichtbar sind, “Aktualisieren” sie die Anzeige im “Projektmappen-Explorer”. Die erzeugte Java Script Datei wird dem Projekt hinzugefügt, und in den Eigenschaften wird bei “Build Vorgang” “Embedded Resource” eingestellt.



Nun wird eine Klasse `BundleConfig` im Ordner “App_Start” erstellt. Die Klasse muss das Interface `IBundleConfig` implementieren und im Standard-Namesspace des Projektes liegen. Das wesentliche Bundle in BA ist “~/bundles/scripts”, dort wird die Java Script Datei hinzugefügt.

```
public void RegisterBundles(BundleCollection bundles)
{
    string assemblyName = GetType().Assembly.GetName().Name;
    Bundle bundle = bundles.FirstOrDefault(ff => ff.Path == "~/bundles/script
s");
    if (bundle == null)
    {
        bundle = new ScriptBundle("~/bundles/scripts");
        bundles.Add(bundle);
    }
    bundle.Include(
        $"~/scripts/BA.Training/Actions.js?{assemblyName}"
    );
}
```

Damit steht nun alles in der UI zur Verfügung, was in diesen Dateien definiert wird.

Achten sie bei *bundle.iclude()* auf die Groß- und Kleinschreibung der Ordner.

Sollte eine JavaScript-Funktionalität, die Sie entwickelt haben, in der UI nicht zur Verfügung stehen, prüfen Sie zunächst die entsprechende Integration in das Script-Bundle und dass die entsprechende *.js Datei in das Projekt integriert und als eingebettete Resource deklariert ist. Dies sind in diesem Zusammenhang sehr häufig gemachte Fehler.

7.2. Funktionen für Aktionen

Um eine Aktion auszuführen, die beim Drücken der Schaltfläche in der Ribbon bar aufgerufen wird, wird eine Type Script Funktion benötigt.

```
public static ClientActionMyAction(event: any, customData: CustomData) {
    alert("Hallo Welt");
}
```

Alle Funktionen, die als Aktionen dienen sollen, müssen am Aktionsframework registriert werden. Dabei wird ein eindeutiger String als ID definiert. Um die Eindeutigkeit sicherzustellen, sollte auch dort mit Projekt spezifischen Präfixen gearbeitet werden.

```
window.setTimeout(
    function () {
        BA.Ui.Actions.ActionHandler.RegisterAction("BA.Training.ClientActionMy
Action", Actions.ClientActionMyAction);
    },
    10
);
```

7.3. BA-Bibliotheken

An dieser Stelle wird ein kurzer Überblick über die wichtigsten verwendbaren BA Funktionen gegeben.

Dialoge, Messageboxen und Toaster werden in dieser Stelle beschrieben. Wie Sie Übersetzungen erhalten hier.

BA.Ui.Actions.FormActions

In dieser Bibliothek befinden sich die Ribbon Bar Aktionen für Masken. Wie besipielsweise

- `ClientActionFormSave(event?: any, customData?: CustomData)`
- `ClientActionFormSaveAndClose(event: any, customData: CustomData)`
- `ClientActionNewRelatedRecord(event: any, customData: CustomData)`
- `ClientActionNewRelatedRecordWithAdditionalRelation(event: any, customData: CustomData)`
- `ClientActionSendEMail(event: any, customData: CustomData)`
- `ClientActionShowRecordHistory(event: any, customData: CustomData)`

BA.Ui.Actions.GridActions

In dieser Bibliothek befinden sich die Ribbon Bar Aktionen für Ansichten. Wie besipielsweise

- `ClientActionGridCreate(event: any, customData: CustomData)`
- `ClientActionGridRefresh(event: any, customData: CustomData)`
- `StartMassOperation(event: any, customData: CustomData)`
- `ClientActionGridShowHideFilterRow()`

BA.Ui.Actions.CommonActions

In dieser Bibliothek befinden sich allgemeine Aktionen. Wie besipielsweise

- `ClientActionSignOut(event: any, customData: CustomData)`
- `OpenFormNewRecord(event: any, customData: CustomData)`
- `ClientActionCloseTab(event: any, customData: CustomData)`
- `ClientActionCancelTab(event: any, customData: CustomData)`
- `OpenRecordCollectionsList(event: any, customData: CustomData)`
- `ClientActionUserSettings(event: any, customData: CustomData)`

8. Ribbon bar Aktionen

Mit Hilfe von Steuerelementen in der Ribbon bar können Aktionen ausgelöst werden. Die Aktionen können sowohl auf dem Client als auch auf dem Server ausgeführt werden. Die massenhafte Verarbeitung von Datensätze findet auf dem Server statt und sollte mit Hilfe von Hintergrundprozessen implementiert werden, um eine Entkopplung vom Benutzerinterface zu schaffen.

8.1. Client Aktionen

Es wird eine neue C# Klasse benötigt. Typischerweise wird diese im Ordner „Configuration\Navigation\ClientAction“ abgelegt. Die Basisklasse ist `ClientActionBase`, und es müssen drei Attribute gesetzt werden. Wie alle anderen Konfigurationen muss auch diese serialisier bar sein, mit `Toolbox` wird die Aktion in der Toolbox für die Navigationen registriert und mit `ControlFilter` wird die Aktion nur für die Ribbon bar sichtbar gemacht.

```
[Serializable]
[Toolbox(EnumConfigurationType.NavigationConfigurationGuid, true)]
[ControlFilter("NavigationConfigurationType", ExpressionType.Equal, EnumNaviga
tionConfigurationType.RibbonNavigationGuid, EnumControlFilterApplyState.IfPosi
tive)]
public class ClientActionMyAction : ClientActionBase { }
```

Im Konstruktor werden die wichtigsten Eigenschaften einer Aktion festgelegt (Siehe Beispiel). Folgende Punkte müssen beachtet werden.

- Anstatt Klartext sollte bei den Eigenschaften „ToolboxName“, „Caption“, „ToolboxGroupName“ und „DesignerHintText“ immer eine Translation Guid benutzt werden.
- Der „ControlInitName“ sollte Eindeutig sein. Also immer mit eigenen Präfixen arbeiten.
- Für „Id“ muss immer eine eigene Guid erzeugt werden.

```
public ClientActionMyAction() : base()
{
    ToolboxName = "Meine Aktion";
    Caption = "Meine Aktion";
    ControlInitName = "TrainingMyAction";
    ToolboxGroupName = "Training Aktionen"; // Translation Guid verwendbar
    Id = new Guid("[INSERT ACTION GUID]");
    Icon = "projector";
    IconName = Icon;
    DesignerHintText = "Meine Aktion tut genau das, was gewünscht wird."; // T
ranslation Guid verwendbar
    VisibilityForParentTypes.Add(EnumActionVisibleForParentType.Form);
}
```

Mit dem Objekt `VisibilityForParentTypes` kann man steuern für welche Objekte (Masken / Ansichten) die eigene Aktion prinzipiell verwendet werden kann.
Damit ist schon eine eigene Aktion so implementiert, dass sie im Designer konfiguriert werden kann.

Die registrierte Aktion kann nun in dem Konstruktor der C# Klasse hinterlegt werden.

```
AdditionalClientData.AddOrUpdate("ActionMethodId", "BA.Training.ClientActionMy
Action");
```

Damit ist das Grundgerüst einer konfigurierbaren Aktion erstellt.

Drag & Drop Regel

Bei jedem Steuerelement, welches im Designer zur Konfiguration genutzt werden soll, muss man darauf achten, dass die Drag & Drop Regeln korrekt gesetzt sind. Näheres dazu findet man in dem speziellen Kapitel.

Meistens werden für Aktionen auf Basis von `ClientActionBase` keine weiteren Regeln benötigt.

Wichtige Klassen zur Definition von Regeln

- `NavigationConfiguration` Hauptknoten
- `NavigationGroupControl` Navigationsgruppe
- `ExtendedNavigationGroupControl` Erweiterte Navigationsgruppe
- `ClientActionBase` Basis aller Aktionen
- `DropDownAction` Aufklappelement

Eigenschaften

Den Aktionen kann man Eigenschaften geben, die in der Konfiguration angepasst werden. Beispiel:

```
[PropertiesGroup("Kundengruppe")] // Übersetzungs GUID verwendbar
[DisplayName("Toastermeldung")] // Übersetzungs GUID verwendbar
[Translate("Training")]
[HelpText("Das ist die Nachricht, die im Toaster ausgegeben wird.")] // Überse
tzungs GUID verwendbar
public string Message { get; set; }
```

Übertragung von Werten an die Type Script Funktion

In der Methode `AdditionalRibbonButtonAssignment` können Daten in das `AdditionalClientDa ta` Objekt geschrieben werden, welche anschließend in der Type Script Funktion verwendet werden können. Das Beispiel übersetzt den Inhalt einer Eigenschaft in die Sprache des Anwenders und überträgt den Wert.

```
public override void AdditionalRibbonButtonAssignment(DevExpress.Web.RibbonBut
tonItem ribbonItem, EnumActionVisibleForParentType parentType, DevExUIModelBas
e uiModel = null)
{
    base.AdditionalRibbonButtonAssignment(ribbonItem, parentType, uiModel);
    AdditionalClientData.AddOrUpdate("Message", Api.Text.Format(Message));
}
```

Der Aufruf der Base Methode sollte unbedingt vorhanden sein, damit auch die Basis Klassen Werte an den Client übertragen können. AdditionalRibbonButtonAssignment wird nur für Aktionen in der Ribbon-Bar aufgerufen, so dass diese Möglichkeit Daten zum Client zu transportieren nicht bei Aktionen funktioniert, die in der Applikationsnavigation oder den Applikationsaktionen (Menü oben rechts) konfiguriert sind. Sollten diese Aktionen serverseitige Werte benötigen, müssen diese entweder schon im Konstruktor (statische Werte) oder bei Abhängigkeit einer bestimmten Eigenschaft im Setter dieser Eigenschaft gesetzt werden.

Der Wert kann nun in der Type Script Funktion verwendet werden.

```
public static ClientActionMyAction(event: any, customData: CustomData) {
    alert(customData.Message as string);
}
```

Aktiv / Inaktiv der Aktion

Wenn Aktionen in Masken verwendet werden, kann es Situationen geben, in denen die Aktionen nicht ausgeführt werden dürfen. Dazu verwendet man das Objekt `DynamicClientVisibility` und fügt diesem bestimmte Verhalten hinzu.

Fehlende Rechte

Falls der Benutzer keine Bearbeitungsrechte auf den aktuellen Datensatz hat, sollen bestimmte Aktionen nicht möglich sein.
Falls dies bei der eigenen Aktion der Fall ist, muss im Konstruktor folgende Zeile hinzugefügt werden:

```
DynamicClientVisibility.Add(EnumActionVisibility.IfUserHasRole);
```

Benötigte Client Daten

```
AdditionalClientData.AddOrUpdate("UserHasRole", true/false);
```

Des Weiteren muss das Interface `IOrmSecurityHandler` implementiert und die Methode `CanHandleOrm` überschrieben werden. Mit dem Setzen des Wertes auf `true` wird signalisiert, dass der aktuelle Benutzer den Datensatz bearbeiten darf.

```
public bool CanHandleOrm(object DataObject, OrmBABase orm) { .... }
```

Beispiel

```
public bool CanHandleOrm(object DataObject, OrmBABase orm)
```

```
{
    bool canHandle;
    if (orm != null)
        canHandle = orm.IsAllowed(EnumTableOperations.Edit);
    else
    {
        OrmEntityConfiguration entityConfig = Api.Config.OrmEntity(EnumDataSou
rceExtension.MyDataTable.ValueGuid);
        canHandle = entityConfig.IsAllowed(EnumTableOperations.Edit) != EnumTa
bleOperations.Denied;
    }

    AdditionalClientData.AddOrUpdate("UserHasRole", canHandle);
    return canHandle;
}
```

Weitere Möglichkeiten

Die Auswahlliste `EnumActionVisibility` beinhaltet eine Reihe weiterer Möglichkeiten. Die prinzipielle Verwendung ist immer identisch. Im Konstruktor wird das Verhalten definiert und die dazu benötigten Daten werden in das `AdditionalClientData` Objekt geschrieben. Ein paar Beispiele:

Datensatz muss gespeichert sein

```
DynamicClientVisibility.Add(EnumActionVisibility.OnlyWhenSaved);
```

Maskenwert beinhaltet einen definierten Wert

```
DynamicClientVisibility.Add(EnumActionVisibility.IfFormValueContainsValue);
```

```
AdditionalClientData.AddOrUpdate("TestValueContains", "Hallo");
AdditionalClientData.AddOrUpdate("TestFormContains", "Subject");
```

Maskenwert beinhaltet einen definierten Wert nicht

```
DynamicClientVisibility.Add(EnumActionVisibility.IfFormValueNotContainsValue);
```

```
AdditionalClientData.Add("TestValueNotContains", "Hallo");
AdditionalClientData.Add("TestFormNotContains", "Subject");
```

Clientside Funktion liefert True zurück

```
DynamicClientVisibility.Add(EnumActionVisibility.IfMethodReturnsTrue);
```

```
AdditionalClientData.AddOrUpdate("IfMethodReturnsTrueMethod", "BA.Customer.Pro
ject.Ui.Utils.CheckVisibility");
```

Sichtbar in Masken und/oder Ansichten

```
DynamicClientVisibility.Add(EnumActionVisibility.OnlyForFormMode);
DynamicClientVisibility.Add(EnumActionVisibility.OnlyForGridLocation);
```

8.2. Verarbeitung von selektierten Datensätzen

Damit eine Aktion alle selektieren Datensätze auf dem Server verarbeiten kann, wird ein Igniter benötigt. Man kann steuern, ob alle Datensätze verarbeitet werden sollen, wenn nichts selektiert ist (`AllRecordsIfNothingIsSelectedIndicator`), ob die Oids der Datensätze in eine Tabelle abgelegt werden (`CreateTemporaryRecordsIndicator`) und ob die Oids nur einmal in der Liste vorkommen dürfen (`UniqueRecordListIndicator`).

```
public class MyFirstIgniter : OperationOverSelectedRecordsIgniterBase
{
    public MyFirstIgniter(IgnitionModel ignitionModel) : base(ignitionModel)
    {
        AllRecordsIfNothingIsSelectedIndicator = false;
        CreateTemporaryRecordsIndicator = false;
        UniqueRecordListIndicator = true;
    }
}
```

In der Methode `Ignite()` werden die selektierten Datensätze verarbeitet.

```
public override ActionResult Ignite()
{
    if (SomethingSelected)
        foreach (RelationData record in Records)
        { }
    return null;
}
```

Wenn man `CreateTemporaryRecordsIndicator = true;` gesetzt hat wurden die Oids in einer Tabelle abgelegt. In `TaskExecutionId` steht eine Id über die man die Oids wieder auslesen kann. Anschließend muss man die Sätze in der temporäre Tabelle löschen.

```
public override ActionResult Ignite()
{
    if (SomethingSelected)
    {
        Session session = Api.ORM.GetNewSession();
        IQueryable<OrmTempSelectedRecords> query = Api.ORM.GetQuery<OrmTempSel
ectedRecords>(session);
        query = query.Where(ff => ff.TaskExecutionId == TaskExecutionId);
        // Laden der Guids
        IQueryable<Guid> guids = query.Select(ff => ff.SelectedRecordOid);
        // Laden der Rlationsdaten
```

```
        IQueryable<RelationData> relationData = query.Select(ff => new Relatio
nData() { Oid = ff.SelectedRecordOid, OrmType = ff.SelectedRecordType });
        // Laden von beliebigen Datensätze
        IQueryable<OrmBABase> orms = Api.ORM.GetQuery<OrmBABase>(session).Wher
e(ff => query.Select(q => q.SelectedRecordOid).Contains(ff.Oid));
        // Laden von konkreten Datensätze
        IQueryable<OrmMyDataTable> myDataTable = Api.ORM.GetQuery<OrmMyDataTab
le>(session).Where(ff => query.Select(q => q.SelectedRecordOid).Contains(ff.Oi
d));
        // Verarbeitung
        ....
        // Löschen der temporären Sätze
        foreach (OrmTempSelectedRecords record in query)
        {
            record.Delete();
        }
    }
    return null;
}
```

Über `IgnitionModel.Parameters` kann man Parameter auslesen, die von der UI gesetzt wurden.

```
public override ActionResult Ignite()
{
    if (IgnitionModel.Parameters.TryGetValue("Start", out object startObj) &&
startObj is bool start && start)
    { }
    return null;
}
```

Es wird ein `ActionResult` zurückgegeben. Dies kann `null` sein oder `JsonResult` mit `JsonFormResult` als Objekt. In dem Beispiel wird die aktuelle Ansicht aktualisiert.

```
return new JsonResult() { Data = new JsonFormResult() { RefreshGrid = true }
};
```

Die Aktion implementiert nicht `ClientActionBase` sondern `ClientActionGridMassOperationBase`. Sinnvolle Einstellungen für die Sichtbarkeit der Aktion sind.

```
DynamicClientVisibility.Add(EnumActionVisibility.ManySelected);
DynamicClientVisibility.Add(EnumActionVisibility.NothingSelected);
DynamicClientVisibility.Add(EnumActionVisibility.OneOrNothingSelected);
DynamicClientVisibility.Add(EnumActionVisibility.OneSelected);
```

Massenoperationen laufen auf alle Datensätze, wenn nichts selektiert ist. Wenn dies nicht der Fall sein

sollte, muss dies gesetzt werden

```
SomethingMustBeSelected = true;
```

Der Anwender kann über den Start der Massenoperation per Toaster-Meldung informiert werden

```
OperationStartedMessage = "Meine Massenoperation startet";
```

Als nächstes muss der Igniter gesetzt werden. Es muss darauf geachtet werden, das der `AssemblyQualifiedName` gesetzt wird.

```
MassOperationIgniter = typeof(MyFirstIgniter).AssemblyQualifiedName;
```

Die Massenoperation wird in Type Script ausgelöst. In einer Type Script Aktion, sind alle notwendigen Informationen zum Starten des Igniters in den `customData` vorhanden. Es müssen lediglich notwendige Parameter des Igniters selbst gesetzt werden.

```
public static ClientActionAddService(event: any, customData: CustomData) {
    let igniterParams: CustomData = {};
    igniterParams['myParameter'] = true;
    customData["MassOperationIgniterParameters"] = JSON.stringify(igniterParam
s);
    BA.Ui.Actions.GridActions.StartMassOperation(event, customData);
}
```

Möchte man einen Igniter außerhalb einer Aktion starten, so tut man dies in dem man die Informationen selbst setzt.

```
let igniterParams: CustomData = {};
igniterParams['myParameter'] = false;
customData["MassOperationIgniter"] = "BA.Training.Igniters.MyFirstIgniter, B
A.Training";
customData["MassOperationIgniterParameters"] = JSON.stringify(igniterParams);
BA.Ui.Actions.GridActions.StartMassOperation(null, customData);
```

Für die Massenverarbeitungen mit Hintergrundprozessen gibt es noch ein vereinfachtes Verfahren.

8.3. Übung 4

In dieser Übung bauen wir eine Aktion, die allen selektierten Maschinen einen Service Eintrag hinzufügt.

Type Script Aktion

Erstellen Sie im “Script” Ordner ein eigenes Unterverzeichnis “BA.Training” und dort eine eigene Script Datei “Actions.ts”. Dort legen Sie eine Aktion an und registrieren sie am BA Framework.

Ribbon bar Aktion

Erstellen Sie im Ordner “Configuration” einen weiteren Unterordner “Navigation” und dort eine Ribbon bar Aktion für Ansichten. Die Aktion soll die Type Script Aktion aufrufen.

Legen Sie eine `string` Eigenschaft an, die später in den Teilmaskendatensatz geschrieben wird. Dazu übertragen Sie den Wert an die UI.

Massenaktion

Erstellen Sie einen Ordner “Igniters” und dort einen Igniter welcher allen Maschinen einen Service Teil-Datensatz mit dem aktuellen Datum und dem übergebenen Text hinzufügt.

Bauen Sie die Aktion so um, dass der Igniter ausgeführt wird und der Text in die Igniter Parameter geschrieben wird.

Lösung

9. Dialoge

Die Interaktion zwischen der Funktionalität und dem Anwender ist ein wesentlicher Baustein jeder Anwendung. BA bietet dafür verschiedene Technologien.

- Toaster für einfache Meldungen
- Messageboxen für einfache Dialoge, ohne eine Kommunikation mit dem Server
- Server generierte Dialoge für komplexe Interaktionen (Im nachfolgendem Dialoge genannt)
  - Wiederverwendbare Standarddialoge
  - Dialog zur Darstellung eines Datensatzes in einer Maske
  - Freie Dialoge, die programmatisch selbst gestaltet werden können.

9.1. Toaster

Meldungen an den Benutzer erfolgen in der Regel über sogenannte Toaster. Diese können sowohl auf dem Client als auch auf dem Server initiiert werden. Folgende Typen von Toaster stehen zur Verfügung

- Fehler
- Warnung
- Information
- Erfolg

Der Typ schlägt sich jeweils in einer anderen farblichen Gebung der Toaster-Meldung nieder.

Serverseitig

Toaster vom Server werden immer einem bestimmten Anwender angezeigt. Die Anzeige erfolgt dabei unmittelbar, falls der Anwender angemeldet ist. Falls dies nicht der Fall ist, wird sie beim nächsten Login angezeigt. Das Verhalten eines Toasters kann über folgende Parameter der Methoden aus `Api.ClientCommunication` beeinflusst werden.

- `userOid` Guid des Benutzers, der die Nachricht erhalten soll.
- `message` Die Nachricht
- `title` Optional: Titel der Nachricht
- `url` Optional: Url für einen zusätzlichen Link (Beispiel: `formGuid + "/index/" + OrmOid`)
- `linkText` Optional: Text des Hyperlinks auf obige Url
- `unlimtedTimeToDisplay` Optional: Ob die Nachrichten mehr als 5 Minuten aufgehoben werden sollen. Default: `false`
- `evalString` Optional: Text mit Java Script der über `eval()` ausgeführt wird, wenn die Nachricht angezeigt wird.
- `displayTimeout` Optional: Zeit wie lange der Toaster sichtbar ist. 0 -> stay sticky
- `oid` Optional: Oid der Nachricht, die verändert werden soll.

API Methoden

```
public Guid Create[Error|Warning|Info|Success]Extended(Guid userOid, String me
ssage, String title = "", String url = "", String linkText = "", Boolean unlim
tedTimeToDisplay = false, String evalString = null, uint displayTimeout = Erro
rTime, Guid? oid = null)
```

Es gibt zwei vereinfachte Varianten, die ein BA definiertes Verhalten haben. Zum einen das Standardverhalten über

```
public Guid Create[Error|Warning|Info|Success](Guid userOid, String message, S
tring title = "", String url = "", String linkText = "", Guid? oid = null)
```

Als weiteres für das Verhalten von Nachrichten von Hintergrundprozessen.

```
public Guid CreateWorker[Error|Warning|Info|Success](OrmUser user, String mess
age, String title = "", String url = "", String linkText = "", Guid? oid = nul
l)
```

Beispiele

```
Api.ClientCommunication.CreateSuccess(Api.User.CurrentUserGuid(), message);
Api.ClientCommunication.CreateWorkerError(Api.User.CurrentUserGuid(), messag
e, Title = title);
```

Falls man anstatt der Oid den `OrmUser` angibt.

```
OrmUser user = Session.GetObjectByKey<OrmUser>(UserContextGuid);
Api.ClientCommunication.CreateSuccess(user, message, url: link, linkText: link
Text);
```

Modifizieren oder Ausblenden von Nachrichten

Dafür wird sich die zurückgegebene `Guid` gemerkt und beim Aufruf der Methoden mitgegeben.

```
Guid toastGuid = Api.ClientCommunication.CreateSuccess(Api.User.CurrentUserGui
d(), message1);
Api.ClientCommunication.CreateSuccess(Api.User.CurrentUserGuid(), message2, oi
d: toastGuid);
Api.ClientCommunication.HideToast(toastGuid);
```

Clientseitig

Zur Anzeige von Toasts wird das Framework Toastr verwendet. Die Type Script Klasse `BA.Ui.Toast` stellt eine vereinfachte Schnittstelle zur Verfügung, um das bekannte Standardverhalten zu ermöglichen. Mit den Optionen in `ToastrOptions` kann man das Verhalten entsprechend modifizieren.

Methoden

```
public static Error(message: string, title?: string, overrides?: ToastrOption
s, oid?: BA.Classes.Guid): BA.Classes.Guid
public static Warning(message: string, title?: string, overrides?: ToastrOptio
ns, oid?: BA.Classes.Guid): BA.Classes.Guid
public static Info(message: string, title?: string, overrides?: ToastrOption
s, oid?: BA.Classes.Guid): BA.Classes.Guid
public static Success(message: string, title?: string, overrides?: ToastrOptio
ns, oid?: BA.Classes.Guid): BA.Classes.Guid
```

Es gibt jeweils noch eine Sticky Methode

Beispiele

```
BA.Ui.Toast.Info(message, title);
BA.Ui.Toast.Warning(message, title);
```

```
var toastGuid: BA.Classes.Guid = BA.Ui.Toast.InfoSticky(message, title);
BA.Ui.Toast.HideToast(toastGuid);
```

9.2. Messageboxen

Messageboxen unterscheiden sich von Dialogen dadurch, dass sie ein rein Browser basiertes Framework sind. Daher können dort weder Daten noch konfigurierte Auswahllisten genutzt werden. Dafür ist beim Aufruf auch keine Serverinteraktion notwendig, und Sie können damit schneller geöffnet werden.

Beispiel

```
BA.Ui.MessageBox.ShowYesNo("Titel", "Frage?", 0,  yesFunction, noFunction);
```

Es können auch einfache Eingaben bzw. Auswahlen durch Messageboxen realisiert werden.

```
BA.Ui.MessageBox.ShowSelection(
     "Selection", // Titel
     "Bitte was auswählen...", // Text
     0, // Breite. 0 = Default
     false, // Einzelauswahl (ja/nein)
     { Key1: 'Was', Key2: 'soll ich', Key3: 'auswählen?' }, // Werte
     function (result) { console.log("Ok"); console.log(result) }, // Ergebnis
funktion für OK
     function (result) { console.log("Cancel"); console.log(result) } // Ergeb
nisfunktion für Cancel
);
```

Der Parameter der Ergebnisfunktion:

- Ist immer ein Stringarray
- Meistens mit nur einem Wert: “OK”, “CANCEL”, “YES”, “NO”
- Oder der im Inputfeld eingegebene Wert
- Oder in der Einzelauswahl der ausgewählte Schlüssel
- Oder im Falle einer Mehrfachauswahl alle ausgewählten Schlüssel
- Die Übergabe der Werte erfolgt auch an die Cancel-Funktion

BA.Ui.MessageBox

```
public static ShowOk(title: string, message: string, width: number = 400, okBt
nCallback?: Function)
```

```
public static ShowWarn(title: string, message: string, width: number = 400, ok
BtnCallback?: Function)
```

```
public static ShowError(title: string, message: string, width: number = 400, o
```

```
kBtnCallback?: Function)
```

```
public static ShowYesNo(title: string, message: string, width: number = 400, y
esBtnCallback?: Function, noBtnCallback?: Function)
```

```
public static ShowYesNoCancel(title: string, message: string, width: number, y
esBtnCallback?: Function, noBtnCallback?: Function, cancelBtnCallback?: Functi
on)
```

```
public static ShowOkCancel(title: string, message: string, width: number = 40
0, okBtnCallback?: Function, cancelBtnCallback?: Function)
```

```
public static ShowPrompt(title: string, message: string, width: number = 400,
okBtnCallback?: Function, cancelBtnCallback?: Function)
```

```
public static ShowSelection(title: string, message: string, width: number, sin
gle: boolean, items: { [key: string]: string }, okBtnCallback?: Function, canc
elBtnCallback?: Function)
```

9.3. Umgang mit Dialogen

Clientseitig

Alle Methoden einen Dialog zu kontrollieren befinden sich im `BA.Ui.Dialog.DialogManager`

Diese Methode öffnet einen definierten Dialog.

```
public static OpenDialog(dialogIdentifier: string, dialogParameter: CustomDat
a, customData: CustomData, onClose: Function)
```

- `dialogIdentifier` ID des Dialogs
- `dialogParameter` Parameter des Dialogs
- `customData` Daten die unverändert der `onClose` Funktion übergeben wird.
- `onClose` Funktion die aufgerufen wird, wenn der Dialog geschlossen wird.
  - Parameter 1 ist das `DialogResult`
  - Parameter 2 ist `customData`

Beispiel

```
let paramter: CustomData = {};
paramter.RemarkMessage = customData.RemarkMessage;
paramter.ServiceDate = customData.ServiceDate;
BA.Ui.Dialog.DialogManager.OpenDialog("BA.Training.AddServiceDialog", paramte
r, null,
    function (result: BA.Ui.Dialog.DialogResult) {
        if (result.ButtonId == 'okButton') {
             BA.Ui.Toast.Info(result.Data.MyValue);
        }
    }
);
```

Weitere Methoden

`GetPopup` liefert den aktuellen Dialog zurück. Wird kein Parameter übergeben, wird der oberste offene Dialog zurückgegeben. Übergibt man ein Control welches sich innerhalb eines Dialoges befindet, wird dieser Dialog zurückgegeben.

```
public static GetPopup(control: ASPxClientControl): BADialogPopup
```

`DialogDefaultCancel` bricht den Dialog ab (Parameter sind optional)

```
public static DialogDefaultCancel(button: BADialogButton, evt: ASPxClientButto
```

```
nClickEventArgs)
```

`SetDialogResult` setzt das Ergebnis des Dialoges

```
public static SetDialogResult(popup: BADialogPopup, result: DialogResult)
```

`DialogClose` schließt einen Dialog (Optionale Übergabe des Popups oder ein Dialogbutton, zur Dialogidentifizierung)

```
public static DialogClose(context: ASPxClientControl)
```

`HandleButtonServerside` ruft ein Dialog Button eine Type Script Methode, kann mit Hilfe dieser Methode die Serverkomponente des Buttons ausgeführt werden.

```
public static HandleButtonServerside(button: BADialogButton, evt: ASPxClientBu
ttonClickEventArgs)
```

Um auf ein Control im Type Script zugreifen zu können, muss man den korrekten Namen haben. Im Dialog erhalten die Namen eine Präfix. Mit dieser Methode erhält man den korrekten Namen eines Controls.

```
public static GetDialogControlName(control: ASPxClientControl, name: string)
```

`IsDialogVisible` gibt an ob ein Dialog sichtbar ist.

```
public static IsDialogVisible(control: ASPxClientControl): boolean
```

Zugriff auf die `FormHiddenData`

```
public static GetFormHiddenData(control: ASPxClientControl): BA.Ui.Models.Form
HiddenDataModel
```

Serverseitig

Es ist möglich bei dem aktuelle Benutzer einen Dialog zu öffnen. Dazu nutzt man die Methode `Api.Cli
entCommunication.OpenDialog`

```
public void OpenDialog(String dialogIdentifier, object dialogParameter, objec
t additionalData = null, String csCallbackFunction = "")
```

9.4. Standard Dialoge

Für Auswahllisten stehen die Standarddialoge zur Verfügung.

- Checkbox-Liste (“BA.Core.Dialogs.CheckboxList”)
- Radiobutton-Liste (“BA.Core.Dialogs.RadioList”)
- Combobox (“BA.Core.Dialogs.Combobox”)
  Für Auswahllisten und für eigenen Daten Provider
- Listbox (“BA.Core.Dialogs.Listbox”)

Parameter

Folgende Parameter gibt es für das Aufrufen von diesen Dialogen:

- `DialogTitle` Der Titel den der Dialog haben soll.
- `Caption` Optional: Das Label welches vor dem Masken-Steuerelement gerendert werden soll.
  Wenn keine Caption angegeben wird, nutzt das Control den ganzen Platz aus.
- `ModelType` Name des Models / Auswahlliste den dieser Dialog verwenden soll. Oder die Guid der Auswahlliste. Dazu später mehr.
  Hinweis: Guid der Auswahlliste geht nicht bei der Combobox mit Data Provider (da diese ja keine Auswahlliste anzeigt).
- `IsEnumDialog` Optional: Soll die Combobox eine Auswahlliste darstellen oder eine selbst definierte? (Default: True)
  Nur bei Combobox-Dialog
- `DataProvider` Die Guid des Data Providers welcher verwendet werden soll.
  Nur bei Combobox notwendig, wenn `IsEnumDialog: false` ist.
- `RepeatColumns` Optional: Über wie viele Spalten sollen sich die Elemente aufteilen (Default: 2)
  Nur bei Checkbox-Liste und der Radiobutton-Liste
- `RepeatDirection` Optional: Soll die Aufteilung Horizontal (Default) oder Vertikal erfolgen.
  Nur bei Checkbox-Liste und der Radiobutton-Liste
- `AddButtons` Optional: Zusätzliche Buttons zum Dialog hinzufügen
  Siehe Hinweis für zusätzliche Buttons
- `Preselected` Optional: Liste mit Elementen (Guids der Auswahllistenwerte) die beim Öffnen des Dialoges bereits vorselektiert sein sollen.
  Nur bei Auswahlisten möglich, nicht bei Combobox mit Data Provider.

Wenn mit `AddButtons` zusätzliche Buttons hinzugefügt werden sollen gibt es hierfür zwei weitere Parameter:

- `Title` Der Button-Titel
- `Method` Der vollständige Name der JavaScript Methode die aufgerufen werden soll.
  Name mit Pfad, z. B. `BA.Ui.Dialog.DialogManager.DialogDefaultCancel`

Die zusätzlichen Buttons werden dabei als Array übergeben:

```
AddButtons: [{Title: "MyButton1", Method: "BA.Ui.Dialog.DialogManager.DialogDe
faultCancel"}, {Title: "My other Button", Method:  "BA.Ui.Dialog.DialogTools.D
efaultComboboxControlCallback"}]
```

Beispiele

Hier werden Beispielaufrufe für jeden Dialog mit den entsprechenden Pflichtparametern gezeigt:

Radiobutton-Liste

```
BA.Ui.Dialog.DialogManager.OpenDialog(
     "BA.Core.Dialogs.RadioList",
     { DialogTitle: "Enum RadioList", Caption: "Bitte wählen", ModelType:"Enum
EmailAddressType", Preselected: ["12FE59CA-196D-42D6-A9B1-3D31DD16F5E7"], AddB
uttons: [{Title: "MyButton1", Method: "BA.Ui.Dialog.DialogManager.DialogDefaul
tCancel"}, {Title: "My other Button", Method: "BA.Ui.Dialog.DialogTools.Defaul
tComboboxControlCallback"}] },
     {custom: „data"},
     function(result, addData) { console.log(result); console.log(addData); }
);
```

Alternativ mit Master Guid der Auswahlliste anstelle des Namens

```
BA.Ui.Dialog.DialogManager.OpenDialog(
     "BA.Core.Dialogs.RadioList",
     { DialogTitle: "Enum RadioList", Caption: "Bitte wählen", ModelType:" 6B1
CFDAF-C222-45FE-932F-CEE5B7E89FA4"},
     {custom: „data"},
     function(result, addData) { console.log(result); console.log(addData); }
);
```

Checkbox-Liste

```
BA.Ui.Dialog.DialogManager.OpenDialog(
     "BA.Core.Dialogs.CheckboxList",
     { DialogTitle: "Enum CheckboxList", Caption: "Bitte wählen", ModelType:
"EnumEmailAddressType" },
     {custom: „data"},
     function(result, addData) { console.log(result); console.log(addData); }
);
```

Listbox

```
BA.Ui.Dialog.DialogManager.OpenDialog(
```

```
     "BA.Core.Dialogs.Listbox",
     { DialogTitle: "Enum Listbox", Caption: "Bitte wählen", ModelType: "EnumE
mailAddressType", Preselected: ["12FE59CA-196D-42D6-A9B1-3D31DD16F5E7"] },
     {custom: „data"},
     function(result, addData) { console.log(result); console.log(addData); }
);
```

Combobox

```
BA.Ui.Dialog.DialogManager.OpenDialog(
     "BA.Core.Dialogs.Combobox",
     { DialogTitle: "Enum Combobox", Caption: "Bitte wählen", ModelType: "Enum
EmailAddressType", Preselected: ["12FE59CA-196D-42D6-A9B1-3D31DD16F5E7"] },
     {custom: "data"},
     function(result, addData) { console.log(result); console.log(addData); }
);
```

Combobox (Mit eigenem Daten Provider)

```
BA.Ui.Dialog.DialogManager.OpenDialog(
     "BA.Core.Dialogs.Combobox",
     { DialogTitle: "Combobox", Caption: "Bitte wählen", ModelType: "DPStandar
dUserProfileList", IsEnumDialog: false, "DataProvider": "76739A3A-67C5-44AD-A3
CF-B1F573E8F3D1"},
     {custom: "Data"},
     function(result, addData) { console.log(result); console.log(addData); }
);
```

Rückgabewert

Der Rückgabewert, also der oder die selektierten Einträge (genauer gesagt die Guids davon), werden als `Data` im `DialogResultModel` zurückgegeben.

In der Browserkonsole sieht das dann z. B. wie folgt aus:

```
Object {
     ButtonId: "okButton",
     Data: (2) […]
          0: "12fe59ca-196d-42d6-a9b1-3d31dd16f5e7"
          1: "54028ce3-294d-4b41-825e-c0ecdee0ff8a"
          length: 2
     InvalidFields: [],
     FieldMessages: [],
     Message: "",
```

```
    MessageType: 0,
    Action: "BA.Ui.Dialog.DialogManager.DialogClose"
}
```

Hier sind die Guids der selektierten Einträge also `12fe59ca-196d-42d6-a9b1-3d31dd16f5e7` und `54028ce3-294d-4b41-825e-c0ecdee0ff8a`.

Und auch im Fehlerfall werden hier nützliche Informationen zurückgegeben, die dann ausgewertet werden können.

9.5. Auswahl von Datensätzen

Eine weitere Art von Standarddialog ist der Auswahldialog oder auch Picklist.
Dieser Dialog dient dazu, Daten möglichst komfortabel aus einer oder mehreren Ansichten anzuzeigen und wählen zu lassen. Die gewählten Daten sind weiterhin optional und situationsabhängig sortierbar.
Dialogtitel und Dialognachricht (über der Auswahl der aktuellen Ansicht) sind optional konfigurierbar; die Nachricht verschwindet, wenn der entsprechende Parameter leer ist, der Dialogtitel zeigt standardmäßig den Text „Auswahl“ an. Beide Werte sind übersetzbar und können eine Guid beinhalten.
Existiert nur eine einzige mögliche Ansicht, entfällt in der UI die Auswahl der aktuellen Ansicht.
Der Dialog hat zwei grundsätzliche Ausprägungen, und zwar die Einzelwert- und die Mehrfachwertauswahl.

Mehrfachwertauswahl

Die Mehrfachwertauswahl bietet die Möglichkeit einen oder mehrere Einträge der gerade angezeigten Ansicht zu selektieren und durch Betätigen der Schaltfläche „Gewählte übertragen“ in den Bereich für gewählte Datensätze zu übernehmen. Alternativ können einzelne Einträge auch durch einen Doppelklick oder per Drag & Drop übernommen werden. Es kann zwischen den zur Verfügung stehenden Ansichten hin und her geschaltet werden, ohne dass die bereits gewählten Einträge verloren gehen. Schon hinzugefügte Einträge werden nicht erneut hinzugefügt, das heißt es ist nicht möglich, doppelte Auswahlen zu treffen. Je nach Konfiguration des Dialogs ist es dem Benutzer möglich, die unter „Gewählte Datensätze“ angezeigten Einträge („Tokens“) mit Hilfe von Drag & Drop umzusortieren.

Einzelwertauswahl

Bei der Einzelwertauswahl ist es im Gegensatz zur Mehrfachauswahl tatsächlich in der angezeigten Ansicht nur möglich einen einzelnen Eintrag auszuwählen. Auch die Anzeige der gewählten Datensätze entfällt. Eine Auswahl des gewählten Datensatzes ist sowohl durch Selektion und anschließendes Betätigen der OK-Schaltfläche als auch durch einen Doppelklick auf den gewünschten Eintrag möglich.

Token

Der Auswahldialog arbeitet sowohl zur Vorauswahl bestimmter Elemente als auch in der Rückgabe mit Objekten des TypeScript-Typs `BA.Ui.Controls.Token`.
Dieser ist wie folgt definiert:

```
export class Token {
    constructor(public Oid: string, public Title: string, public Type: strin
g) { }
}
```

Objekte dieses Typs beinhalten drei essenzielle Eigenschaften:

- `Oid` Die Guid des gewählten Elements
- `Title` Der `EntityTitle` des gewählten Elements

- `Type` Der `OrmType` des gewählten Elements (falls ermittelbar)

Aufruf eines Auswahldialogs und Auswertung des Ergebnisses

Der programmatische Aufruf eines Auswahldialogs erfolgt über die oben beschriebene Funktion `BA.Ui.Dialog.DialogManager.OpenDialog`, die zugehörige Dialog ID lautet **BA.Core.Dialogs.PickListDialog**. Die Steuerung des Dialogs erfolgt über die mitgegebenen Dialogparameter.

- `DialogTitle` Optional: Der Titel den der Dialog haben soll.
  Guid für Übersetzungen möglich, wenn nicht angegeben, Standardwert „Auswahl“
- `DialogDescription` Optional: Nachricht, die oben als erstes im Dialog über der Ansichten Auswahl angezeigt wird.
  Guid für Übersetzungen möglich, wenn nicht angegeben, wird diese Zeile ausgeblendet.
- `DialogSelectionGrids` Zur Auswahl stehende Ansichten-Konfigurationen, zwischen denen in der Picklist umgeschaltet werden kann.
  String oder String[] mit Guids von Ansichten-Konfigurationen
- `SelectedTokens` Optional: Bereits gewählte Elemente, die im Bereich „Gewählte Datensätze“ beim Öffnen des Dialogs bereits in der gegebenen Reihenfolge angezeigt werden.
  JSON-String, serialisiertes Array von Objekten des Typs `BA.Ui.Controls.Token`
- `MultiSelect` Optional: True, wenn mehr als ein Element aus der GridView ausgewählt werden können soll.
  `Boolean`, Standardwert: `true`.
- `Sortable` Optional: True, wenn die ggf. angezeigten Auswahlwerte um sortierbar sein sollen. Das ist abhängig von anderen Konfigurationsparametern, die den Bereich, in dem sortiert werden kann, eventuell ausblenden.
  `Boolean`, Standardwert: `true`
- `MinSelect` Optional: Minimale Anzahl zu selektierender Einträge, muss erfüllt sein, bevor der Dialog mit OK geschlossen werden kann.
  Ganzzahl, Standardwert: 0 (keine Beschränkung)
- `MaxSelect` Optional: Maximale Anzahl selektierbarer Einträge, muss erfüllt sein, bevor der Dialog mit OK geschlossen werden kann.
  Ganzzahl, Standardwert: 0 (keine Beschränkung)
  Ein Wert von 1 hat den gleichen Effekt wie `MultiSelect = false`
- `RawResult` Optional: True, wenn der Dialog das resultierende Array von Token-Objekten als serialisierten JSON-String zurückgeben soll.
  `Boolean`, Standardwert: `false`
  Token haben den Typen `BA.Ui.Controls.Token`

Beispiel

```
BA.Ui.Dialog.DialogManager.OpenDialog(
     "BA.Core.Dialogs.PickListDialog",
```

```
     {
          DialogTitle: "Meine Auswahl",
          DialogDescription: "Bitte wählen",
          DialogSelectionGrids: ["12AF9DED-D118-41F5-A050-11D32D879DCB"],
     MultiSelect: false
     },
     {custom: "data"},
     function(result, customData) { console.log(result); console.log(customDat
a); }
);
```

Das Beispiel öffnet in BA.CRM einen Auswahldialog zur Abfrage einer einzelnen E-Mail-Vorlage.
Das Datenfeld des Dialogergebnisses enthält im Normalfall immer ein Array von Objekten des TypeScript-Typs `BA.Ui.Controls.Token`, auch, wenn es sich um eine Einzelauswahl gehandelt hat.
Nur, wenn der Dialogparameter `RawResult` auf `true` mitgegeben wird, liegt im Datenfeld ein String, der das entsprechende Array bereits JSON-serialisiert beinhaltet.
Die Reihenfolge der Objekte im Ergebnis entspricht der Sortierung der Tokens im Dialog.

9.6. Datensätze im Dialog

Es ist möglich beliebige Datensätze in Dialogen mit Hilfe von Masken zu bearbeiten. Beispielsweise wird diese Möglichkeit in den Detailansichten der Masken genutzt, wenn dies entsprechend konfiguriert ist. Prinzipiell ist es möglich jede konfigurierte Maske in einem Dialog anzeigen zu lassen. Entweder über die Konfiguration oder es wird ein entsprechender Aufruf implementiert.

```
BA.Ui.Dialog.OrmDialogManager.OpenDialog
BA.Ui.Dialog.OrmDialogManager.OpenDialog(formModel, customData, returnFunc);
```

Diese Methode öffnet einen Datensatz oder erstellt einen neuen Datensatz in einem Dialog. Das Verhalten wird vom ersten Parameter `formModel` definiert. Dieser ist vom Typ `BA.Ui.Models.FormDialogModel` und beinhaltet alle notwendigen Informationen. In einem späteren Abschnitt wird das Objekt genauer beschrieben.
Der zweite Parameter `customData` ist vom Typ `CustomData` und wird vom Dialogsystem nicht beachtet. Er wird unverändert der Callback Funktion übergeben.
Der dritte Parameter `returnFunc` ist die Callback Funktion und wird nach dem Beenden des Dialoges aufgerufen. Signatur der Callback Funktion

```
function (result: BA.Ui.Dialog.DialogResult, customData: CustomData) { }
```

Beispiel 1: Selektierter Datensatz in Ansicht im Dialog öffnen

Es wird aus einer Ansicht der selektierte Datensatz ausgelesen. Dieser Datensatz wird mit der Standardmaske geöffnet. Hier eine Funktion, die von einer Ribbon bar Aktion aufgerufen wird.

```
public static ClientActionMyAction(event: any, customData: CustomData) {
    // Hauptansicht vorhanden?
    if (window["GridView"] != null) {
        // Selektionen holen
        let selected: Object[] = window.GridView.GetSelectedKeysOnPage();
        if (selected.length == 1) {
            // Dialog form model belegen
            var formModel: BA.Ui.Models.FormDialogModel = new BA.Ui.Models.For
mDialogModel();
            formModel.RecordId = selected[0] as string;
            formModel.RefreshGridOnSuccess = "GridView";

            // Dialogaufruf
            BA.Ui.Dialog.OrmDialogManager.OpenDialog(formModel, customData,
                function (result: BA.Ui.Dialog.DialogResult, customData: Custo
mData) {
```

```
                    // Wenn der Benutzer gespeichert hat, soll das Grid  aktua
lisiert werden.
                    if (result.ButtonId == "okButton") {
                        alert("Do something");
                    }
                }
            );
        }
    }
}
```

Beispiel 2: DialogFormModel auf den Server vorbereiten

Es kann sinnvoll sein das `FormDialogModel` auf dem Server schon mit den wesentlichen Daten vorzubelegen. Dazu wird das Model erstellt, belegt und serialisiert. Mit Hilfe der `AdditionalClientData` steht es im Browser zur Verfügung. Dort wird es deserialisiert, evtl. mit weiteren Daten belegt und dem Dialogsystem übergeben.
Beispiel aus einer Ribbon bar Aktion (Server)

```
public override void AdditionalRibbonButtonAssignment(RibbonButtonItem ribbonI
tem, EnumActionVisibleForParentTypeValue parentType, DevExUIModelBase uiModel
= null)
{
    FormDialogModel dialogModel = new FormDialogModel
    {
        Form = "135dc9db-fd91-407b-93a6-62fc360109b7".ToGuid(),
        ButtonPreset = DialogButtonsOptions.Ok
    };

    AdditionalClientData.AddOrUpdate("JsonFormModel", Api.JsonHelper.Serializ
e(dialogModel));
    base.AdditionalRibbonButtonAssignment(ribbonItem, parentType, uiModel);
}
```

Beispiel aus einer Ribbon bar Aktion (Browser)

```
// Die aktuellen Formdaten holen
let formHiddenData: BA.Ui.Models.FormHiddenDataModel =  BA.Ui.TabController.Ta
bTools.GetCurrentFormHiddenData(null);
// Die AdditionalClientData stehen in Aktionen als customData zur Verfügung
let jsonFormModel: string = customData.JsonFormModel as string;
// JSON string parsen
let formModel: BA.Ui.Models.FormDialogModel = <BA.Ui.Models.FormDialogModel>JS
ON.parse(jsonFormModel);
```

```
// Model um die ID des aktuell offenen Dokumentes erweitern
formModel.RecordId = formHiddenData.RecordId.toString();

// Dialogaufruf
BA.Ui.Dialog.OrmDialogManager.OpenDialog(formModel, null, null);
```

9.7. Eigene Dialoge implementieren

Die Idee des Dialogsystems unterscheidet den Dialogentwickler von dem Entwickler, der den Dialog nutzt. Der Entwickler implementiert einen Dialog, der über Parameter gesteuert eine Aufgabe erfüllt. Über ein Objekt wird das Ergebnis des Dialoges zur Verfügung gestellt.
Der Nutzer des Dialoges ruft den Dialog auf und setzt die Parameter entsprechend seinen Anforderungen. In einer Callbackfunktion wird ihm das Ergebnis des Dialoges zurückgeliefert. Damit wird sichergestellt, dass der Dialog beliebig technisch umgestellt werden kann, da der Nutzer keinen Zugriff auf den Dialog hat.

Dialogklasse

Ein Dialog besteht aus einer Dialogklasse mit zwei Methoden.

```
[DialogImplementation("BA.Customer.Project.MyDialog")]
public class MyDialog : DialogImplementationBase
{
    public override void CreateDialogContent(
        DevExFormModel formModel, // Das Formmodel zur Darstellung des Dialoge
s
        HttpRequestBase request, // Der HttpRequest
        ModelStateDictionary modelState, // Der Status des verwendeten Models
        Dictionary<String, Object> parameter, // Die Parameter des Dialoges
        object bindObject = null // Das Model mit den Daten
    )
    { .... }
    public override DialogResultModel HandleAction(
        HttpRequestBase request, // Der HttpRequest
        ModelStateDictionary modelState, // Der Status des verwendeten Models
        Dictionary<String, Object> parameter, // Die Paramter des Dialoges
        String buttonId, // Die ID des gedrückten Buttons
        object bindObject, // Das Model mit den Daten
        string propertyPrefix = "" // Der mögliche Prefix von Werten
    )
    {
        DialogResultModel result = base.HandleAction(request, modelState, para
meter, buttonId,
                                                     bindObject, propertyPrefi
x);
        return result;
    }
}
```

Die Klasse muss zwei Voraussetzungen erfüllen

1. Sie muss die Klasse `DialogImplementationBase` erweitern.

2. Über das Attribute `DialogImplementation` wird der Dialogidentifier festgelegt

Die Methode `HandleAction` muss nicht überschrieben werden.

CreateDialogContent

In dieser Methode wird der Inhalt des Dialoges implementiert. Diese Methode wird bei allen Aktualisierungen aufgerufen und kann jeweils einen anderen Inhalt generieren.

Einfacher Dialog mit einer Textbox

```
public override void CreateDialogContent(DevExFormModel formModel, HttpRequest
Base request, ModelStateDictionary modelState, Dictionary<String, Object> para
meter, object bindObject = null)
{
        formModel.Title = "Dialogtitel";
        formModel.FormGuid = Guid.NewGuid();
        formModel.DataSource = null;

        TextEditControl textControl = new TextEditControl()
        {
            Id = "4095021E-6762-4894-8B42-9164EBC1281F".ToGuid(),
            Caption = "Text"
        };
        Controls.Add(textControl);

        formModel.AddButton(DialogButtonIds.OkButton, "OK", true);
        formModel.AddButton(DialogButtonIds.CancelButton, "Cancel", false, "B
A.Ui.Dialog.DialogManager.DialogDefaultCancel");

        MVCxFormLayoutItemCollection formControls = DevExFormLayoutTranslato
r.TranslateControls(Controls, formModel.DataSource, formModel, null, requestCo
ntext: request);

        DevExFormPartModel partModel = new DevExFormPartModel();
        partModel.Controls = formControls;
        partModel.FormName = "dialogPart";
        formModel.LayoutName = EnumFormLayout.SingleColumn.LayoutName;
        formModel.LayoutPanels.Add(partModel);

        request.ConfigurationGuid(formModel.FormGuid);
}
```

Mit Hilfe eines Datenmodels werden die Maskensteuerelemente gebunden

```
public override void CreateDialogContent(DevExFormModel formModel, HttpRequest
Base request, ModelStateDictionary modelState, Dictionary<String, Object> para
```

```
meter, object bindObject = null)
{
    formModel.Title = "Dialogtitel";
    formModel.FormGuid = Guid.NewGuid();
    MyDialogModel datenModel;
    if (bindObject == null)
    {
        datenModel = new MyDialogModel();
        datenModel.TextEdit = "Vorgabewert";
    }
    else
        datenModel = (MyDialogModel)bindObject;

    formModel.DataSource = datenModel;

    TextEditControl textControl = new TextEditControl()
    {
        Id = "4095021E-6762-4894-8B42-9164EBC1281F".ToGuid(),
        Caption = "Text",
        OrmFieldName = "TextEdit"
    };
    Controls.Add(textControl);

    formModel.AddButton(DialogButtonIds.OkButton, "OK", true);
    formModel.AddButton(DialogButtonIds.CancelButton, "Cancel", false,
            "BA.Ui.Dialog.DialogManager.DialogDefaultCancel");

    MVCxFormLayoutItemCollection formControls = DevExFormLayoutTranslator.Tran
slateControls(
            Controls, formModel.DataSource, formModel, null, requestContext: r
equest);

    DevExFormPartModel partModel = new DevExFormPartModel();
    partModel.Controls = formControls;
    partModel.FormName = "dialogPart";
    formModel.LayoutName = EnumFormLayout.Values.SingleColumn.LayoutName;
    formModel.LayoutPanels.Add(partModel);

    request.ConfigurationGuid(formModel.FormGuid);
}

public class MyDialogModel
{
    public String TextEdit { get; set; }
}
```

In einem späteren Abschnitt werden alle verwendbaren Steuerelemente aufgelistet, inkl. deren Model Property Definitionen.

Es wird empfohlen, zur Auswahl von Auswahllistenwerten grundsätzlich auch die zugehörigen Steuerelemente (EnumComboboxControl, EnumTokenboxControl, etc) anstelle eines generischen Steuerelements mit eigenem Data-Provider zu verwenden.

Buttons

Mit `AddButton()` werden Buttons zu Dialogen hinzugefügt. Entweder gibt man die 4 wichtigsten Parameter direkt an:

```
formModel.AddButton(DialogButtonIds.CancelButton, "37857EB7-EDBC-4419-9823-6FC
107794621".Translate(), false, "BA.Ui.Dialog.DialogManager.DialogDefaultCance
l");
```

Oder man erstellt ein `DialogButtonDefinition` Model mit allen Eigenschaften und übergibt dieses als Parameter:

```
DialogButtonDefinition okButton = new DialogButtonDefinition() {
     ButtonId = DialogButtonIds.OkButton,
     Caption = Api.Text.Format("68B063D1-9626-4015-BCC8-8D41C76B8596"),
     ServerCallback = true,
     IsEnabled = true,
     HintText = "Jetzt Speichern"
};
formModel.AddButton(okButton);
```

Im Folgenden eine Übersicht möglicher Parameter.

- `ButtonId` Die ID des Buttons.
- `Caption` Die Beschriftung des Buttons.
- `ServerCallback` Gibt es ein Server-Callback
- `FunctionName` Der Name (mit Pfad) der Client-Action.
  Nur wenn es kein Server-Callback ist.
- `IsEnabled`Optional: Soll der Button aktiv sein (Vorgabe: `true`)
- `HintText` Optional: Hinweistext / Tooltip wenn die Maus über dem Button ist.

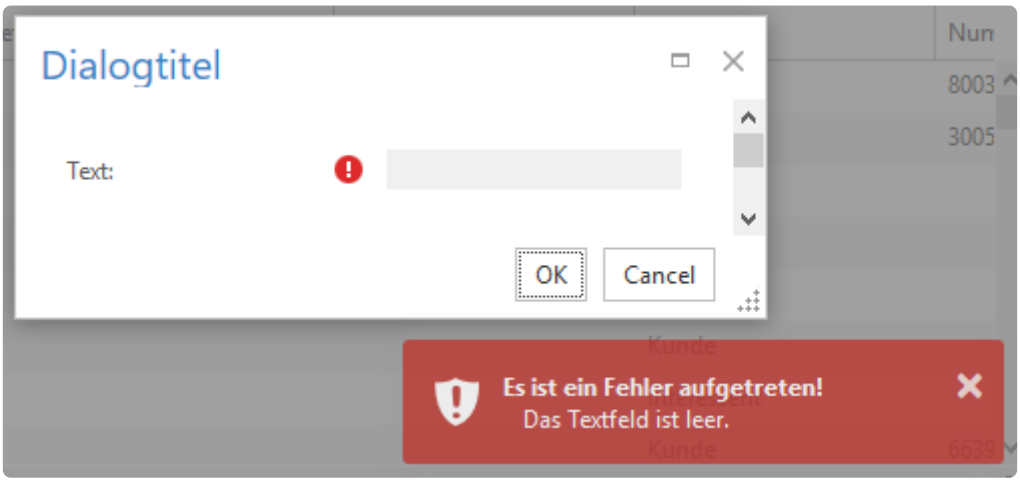
HandleAction

In `HandleAction` werden die Buttons verarbeitet, die auf `serverCallback=true` stehen. Per Default

wird der Dialog geschlossen. Ein anderes Verhalten wird an dieser Stelle implementiert. In dem Beispiel wird das Feld `TextEdit` validiert und eine entsprechende Fehlermeldung ausgegeben.

```
public override DialogResultModel HandleAction(HttpRequestBase request, ModelS
tateDictionary modelState, Dictionary<String, Object> parameter, String button
Id, object bindObject, string propertyPrefix = "")
{
    DialogResultModel result = base.HandleAction(request, modelState, paramete
r, buttonId, bindObject, propertyPrefix);
    if (buttonId == DialogButtonIds.OkButton)
    {
        MyDialogModel dlgData = (MyDialogModel)bindObject;
        if (String.IsNullOrWhiteSpace(dlgData.TextEdit))
        {
            result.Message = "Es ist ein Fehler aufgetreten!";
            result.MessageType = EnumToastNotificationType.Error;
            result.InvalidFields.Add("TextEdit");
            result.FieldMessages.Add("Das Textfeld ist leer.");
            result.Action = "";
        }
    }
    return result;
}
```

Ergebnis für den Anwender, wenn die Validierung fehlschlägt.



Wenn man die Validatoren Attribute im Model setzt, wird die Validierung automatisiert durchgeführt.

Browser Funktionalitäten

Werden im Browser Funktionalitäten implementiert, muss beachtet werden, dass die

Maskensteuerelemente in den Dialogen mit einem Präfix versehen sind. Um den korrekten Namen eines Steuerelementes zu erhalten, steht eine entsprechende Methode im DialogManager zur Verfügung. Beispiel:

```
public static MyDialogButton(button: BADialogButton, evt: ASPxClientButtonClic
kEventArgs) {
    let textBoxName: string = BA.Ui.Dialog.DialogManager.GetDialogControlNam
e(button, "TextEdit");
    if (window[textBoxName]) {
        let box: BAClientTextBox = <BAClientTextBox>window[textBoxName];
        box.SetValue("Neuer Wert");
    }
}
```

Höhe und Breite / Adaptives Layout

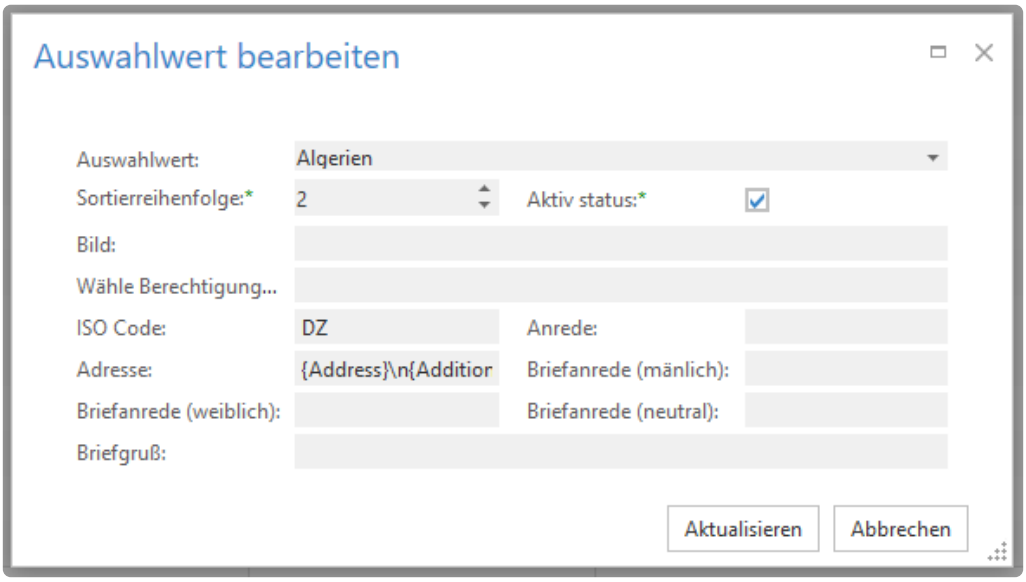
Das Prinzip in der Darstellung der Dialoge basiert auf einer vorgegebenen Breite und einer automatisch berechneten Höhe. Die Standardbreite eines Dialoges ist 600 Pixel. Bei der Implementierung sollte man für seinen Dialog die entsprechende Breite angeben.

```
formModel.Width = 600;
```

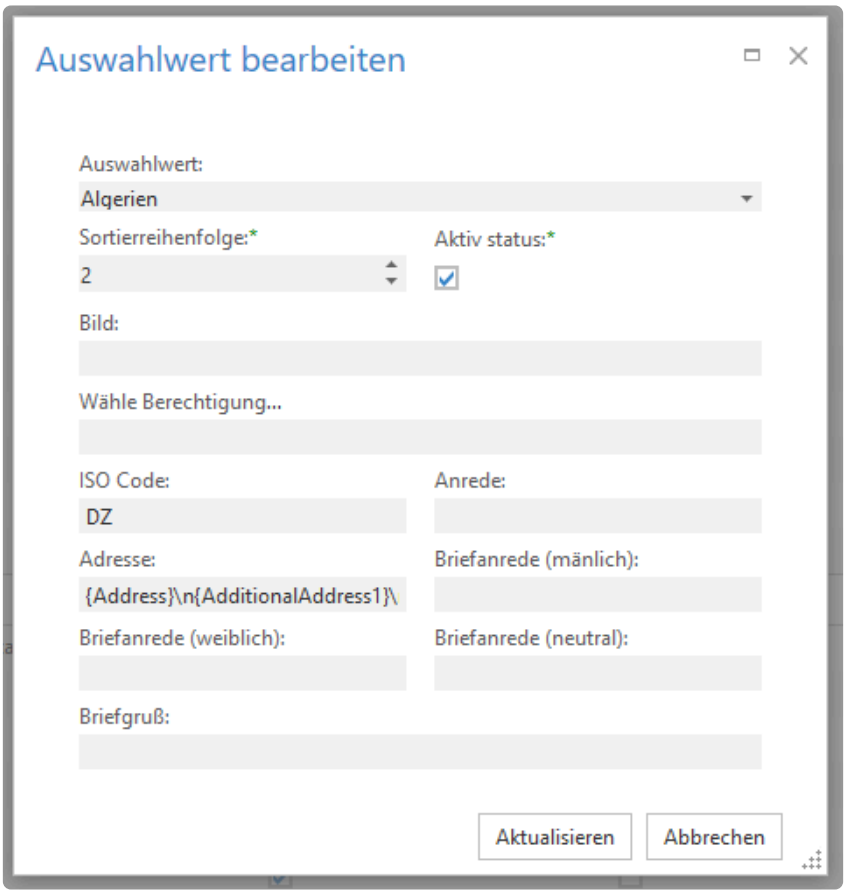
Die Höhe wird anschließend von dem System automatisch berechnet. Alternativ kann man die Höhe ebenfalls definieren.

Das Framework der Maskensteuerelemente unterstützt das Adaptive Layout von DevExpress. Siehe dazu auch die DevExpress Online Dokumentation. Folgende Screenshots desselben Dialoges mit unterschiedlichen Breiten verdeutlichen die Möglichkeiten.

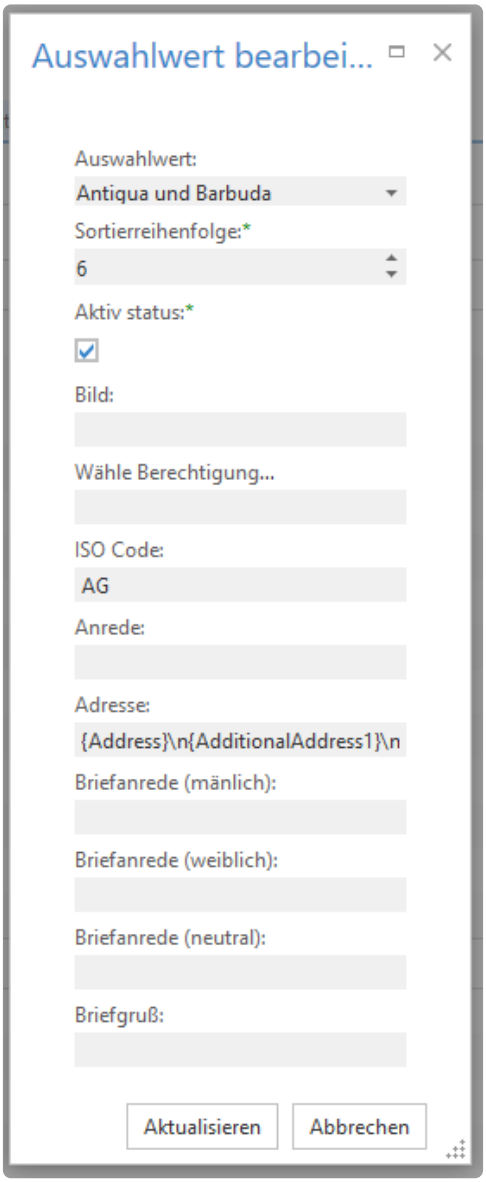
Breiter Dialog (Zweispaltig / Labels vor dem Feld)



Mittlerer Dialog (Zweispaltig / Labels über dem Feld)



Schmaler Dialog (Einspaltig / Labels über dem Feld)



Basis ist ein `LayoutPanelControl`. Beispiel

```
LayoutPanelControl layout = new LayoutPanelControl()
{
    Id = "DEB77CD6-1823-445F-A70D-A0E6001E4351".ToGuid(),
    ColumnCount = 2,
    WrapContentAtWidth = 500,
    StretchLastItem = true,
};
layout.AddBreakpoint(new BreakpointRule() { Name = "S", ColumnCount = 1, MaxWi
dth = 700 });
layout.AddBreakpoint(new BreakpointRule() { Name = "M", ColumnCount = 2, MaxWi
dth = 1000 });
```

Bei den Steuerelementen wird dann jeweils deren Verhalten definiert.

```
TextEditControl textControl = new TextEditControl()
```

```
{
    Id = "4095021E-6762-4894-8B42-9164EBC1281F".ToGuid(),
    Caption = "Text",
    OrmFieldName = "TextEdit",
    ColSpan = 2,
};
textControl.AddSpanRule((new SpanRule() { BreakpointName = "S", ColumnSpan =
1 });
textControl.AddSpanRule((new SpanRule() { BreakpointName = "M", ColumnSpan =
2 });
```

Bei Gruppenelementen legt man sowohl die Breakpoints für die Gruppe als auch die eigenen Spanrules fest.

```
var group = new GroupControl
{
    Id = "E398BFE6-202D-4F14-9B09-FE059B314E2A".ToGuid(),
    Caption = "",
    WrapContentAtWidth = 500,
    StretchLastItem = true,
    ColumnCount = 2
};
group.AddBreakpoint(new BreakpointRule() { Name = "S", ColumnCount = 1, MaxWid
th = 300 });
group.AddBreakpoint(new BreakpointRule() { Name = "M", ColumnCount = 2, MaxWid
th = 500 });
group.AddSpanRule(new SpanRule() { BreakpointName = "S", ColumnSpan = 1 });
group.AddSpanRule(new SpanRule() { BreakpointName = "M", ColumnSpan = 2 });
```

Maskensteuerelemente für Dialoge

In eigenen Dialogen können alle Maskensteuerelemente genutzt werden, die das Attribut `[EnabledForFreeDialogs]` haben. Es können in der Regel die Steuerelemente nicht genutzt werden, die zwingend einen Datensatz erfordern. Beispielsweise sind das Relationsauswahlelemente.

Werden eigene Maskensteuerelemente implementiert und sollen diese in eigenen Dialogen genutzt werden, müssen sie mit dem Attribut `[EnabledForFreeDialogs]` versehen werden.

Aufruf

Der Aufruf eines Dialoges erfolgt über den DialogManager

```
BA.Ui.Dialog.DialogManager.OpenDialog(
     "BA.Customer.Project.MyDialog ", // Dialog Identifier
```

```
    { "para1": "Wert" }, // Parameterobjekt
    customData, // CustomData Objekt. Wird 1 zu 1 an die resultFunc übergeben
    resultFunc // Function die beim Beenden des Dialoges aufgerufen wird
);
```

Signatur der Ergebnisfunktion

```
function (result: BA.Ui.Dialog.DialogResult, customData: CustomData) { }
```

Standarddialoge modifizieren

Unter Umständen ist es notwendig das Verhalten eines Standarddialoges zu modifizieren. Dies birgt immense Risiken und sollte nur sehr bedacht getan werden. Auch sollte nach jedem Update diese Anpassung kontrolliert werden, da Änderungen an den Standarddialogen nicht kommuniziert werden.

Um einen vorhanden Dialog zu ändern, muss lediglich der identische Dialog-Identifier vergeben werden. Sinnvoll ist es, dass die eigene Klasse die ursprüngliche Klasse erweitert und man die Basemethoden aufruft.

9.8. Maskensteuerelemente für Dialoge

In diesem Abschnitt werden die verwendbaren Controls mit Beispielen gezeigt. Die Beispiele teilen sich – wenn notwendig – auf in den Teil, der das Steuerelement definiert und den Teil, der das Property im Model zeigt.

Gruppe

```
// Control
GroupControl group = new GroupControl
{
    Id = "[Insert Unique ID]".ToGuid(),
    Caption = "",
};
```

Statischer Text

```
// Control
StaticLabelControl staticLabelControl = new StaticLabelControl()
{
    Id = "[Insert Unique ID]".ToGuid(),
    Caption = "Static Control",
    LabelText = "Label Text",
};
```

Leeres Element

```
// Control
EmptyControl empty = new EmptyControl()
{
    Id = "[Insert Unique ID]".ToGuid()
};
```

Text- und Zahleingabe

```
// Control
TextEditControl text = new TextEditControl()
{
    Id = "[Insert Unique ID]".ToGuid(),
    Caption = "Text",
    OrmFieldName = "Text"
};
```

```
// Model Property
public String Text { get; set; }
```

Passworteingabe

```
// Control
PasswordControl password = new PasswordControl()
{
    Id = "[Insert Unique ID]".ToGuid(),
    Caption = "Password",
    OrmFieldName = "Password"
};
```

```
// Model Property
public String Password { get; set; }
```

Zahleingabe mit Spin-Buttons

```
// Control
SpinEditControl spin = new SpinEditControl()
{
    Id = "[Insert Unique ID]".ToGuid(),
    Caption = "Spin",
    OrmFieldName = "Spin",
    MinValue = 5,
    MaxValue = 30
};
```

```
// Model Property
public Int16 Spin { get; set; }
```

Zahleingabe als Schieberegler

```
// Control
TrackbarControl trackbar = new TrackbarControl()
{
    Id = "[Insert Unique ID]".ToGuid(),
    Caption = "Trackbar",
    OrmFieldName = "Trackbar",
    MinValue = 0,
    MaxValue = 100,
    SmallTickFrequency = 5,
```

```
    LargeTickInterval = 25
};
```

```
// Model Property
public Int32 Trackbar { get; set; }
```

Texteingabe mit Link im Lesemodus

```
// Control
HyperlinkEditControl hyperLink = new HyperlinkEditControl()
{
    Id = "[Insert Unique ID]".ToGuid(),
    Caption = "Hyper link",
    OrmFieldName = "HyperLink"
};
```

```
// Model Property
public String HyperLink { get; set; }
```

Eingabe Telefonnummer mit Wählfunktion im Lesemodus

```
// Control
CTILinkEditControl phone = new CTILinkEditControl()
{
    Id = "[Insert Unique ID]".ToGuid(),
    Caption = "Phone",
    OrmFieldName = "Phone"
};
```

```
// Model Property
public String Phone { get; set; }
```

IBAN Eingabe

```
// Control
IBANControl iban = new IBANControl()
{
    Id = "[Insert Unique ID]".ToGuid(),
    Caption = "IBAN",
    OrmFieldName = "IBAN"
};
```

```
// Model Property
public String IBAN { get; set; }
```

BIC Eingabe

```
// Control
BICControl bic = new BICControl()
{
    Id = "[Insert Unique ID]".ToGuid(),
    Caption = "BIC",
    OrmFieldName = "BIC"
};
```

```
// Model Property
public String BIC { get; set; }
```

Dezimaleingabe mit Währungssymbol

```
// Control
CurrencyControl currency = new CurrencyControl()
{
    Id = "[Insert Unique ID]".ToGuid(),
    Caption = "Currency",
    OrmFieldName = "Currency"
};
```

```
// Model Property
public Decimal Currency { get; set; }
```

Mehrzeilige Texteingabe

```
// Control
MemoControl memo = new MemoControl()
{
    Id = "[Insert Unique ID]".ToGuid(),
    Caption = "Memo",
    Height = 75,
    OrmFieldName = "Memo"
};
```

```
// Model Property
```

```
public String Memo { get; set; }
```

Datums- und Zeiteingabe

```
// Control
DateEditControl dateTime = new DateEditControl()
{
    Id = "[Insert Unique ID]".ToGuid(),
    Caption = "Date time",
    ShowTimeSection = true,
    OrmFieldName = "DateTime"
};
```

```
// Model Property
public DateTime DateTime { get; set; }
```

Zeiteingabe

```
// Control
TimeEditControl time = new TimeEditControl()
{
    Id = "[Insert Unique ID]".ToGuid(),
    Caption = "Time",
    OrmFieldName = "Time"
};
```

```
// Model Property
public DateTime Time { get; set; }
```

Einzelne Checkbox (Boolean)

```
// Control
CheckEditControl checkBox = new CheckEditControl()
{
    Id = "[Insert Unique ID]".ToGuid(),
    Caption = "Check box",
    OrmFieldName = "CheckBox"
};
```

```
// Model Property
public Boolean CheckBox { get; set; }
```

Auswahlwerteliste über Checkboxen

```
// Control
EnumCheckboxListControl cblist = new EnumCheckboxListControl()
{
    Id = "[Insert Unique ID]".ToGuid(),
    Caption = "Enum check box list",
    OrmFieldName = "EnumSalutation",
    RepeatColumns = 3
};
```

```
// Model Property
public List<EnumSalutationsValue> EnumSalutation { get; set; }
```

Auswahlwerteliste über Radio-Buttons

```
// Control
EnumRadioButtonListControl rblist = new EnumRadioButtonListControl()
{
    Id = "[Insert Unique ID]".ToGuid(),
    Caption = "Enum radio button list",
    OrmFieldName = "EnumAddressType",
    RepeatColumns = 2,
    RepeatDirection = EnumChoiceListRepeatDirection.Values.Vertical
};
```

```
// Model Property
public EnumAddressTypesValue EnumAddressType { get; set; }
```

Auswahlwerteliste über eine Listbox

```
// Control
EnumListboxControl lblist = new EnumListboxControl()
{
    Id = "[Insert Unique ID]".ToGuid(),
    Caption = "Enum list box",
    OrmFieldName = "EnumTitles"
};
```

```
// Model Property
public List<EnumTitlesValue> EnumTitles { get; set; }
```

Auswahlwerteliste über eine Tokenauswahl

```
// Control
EnumTokenboxControl lblist = new EnumTokenboxControl()
{
    Id = "[Insert Unique ID]".ToGuid(),
    Caption = "Enum list box",
    OrmFieldName = "EnumTitles"
};
```

```
// Model Property
public List<EnumTitlesValue> EnumTitles { get; set; }
```

Auswahlwerteliste über eine Combobox

```
// Control
EnumComboboxControl colist = new EnumComboboxControl()
{
    Id = "[Insert Unique ID]".ToGuid(),
    Caption = "Enum combobox",
    OrmFieldName = "EnumCTIProtocol"
};
```

```
// Model Property
public EnumCTIProtocolsValue EnumCTIProtocol { get; set; }
```

Auswahl eines Wertes mit einem eigenen Datenprovider

Wie Datenprovider implementiert werden finden sie hier

```
// Control mit eigenem Datenprovider
ComboBoxControl comboBox = new ComboBoxControl()
{
    Id = "[Insert Unique ID]".ToGuid(),
    Caption = "Combo box",
    NameOverride = "ComboBox",
    DataProviderProperties = new CDPEnumValuesProperties(masterGuid: EnumCount
ries.Guid)
};
```

```
// Model Property
public String ComboBox { get; set; }
```

Auswahl und Erfassung einer Übersetzung

```
// Control
TranslationControl translation = new TranslationControl()
{
    Id = "[Insert Unique ID]".ToGuid(),
    Caption = "Translation",
    NameOverride = "Translation"
};
```

```
// Model Property
public String Translation { get; set; }
```

Eingabe formatierter Text

```
// Control
HTMLControl htmlControl = new HTMLControl()
{
    Id = "[Insert Unique ID]".ToGuid(),
    Caption = "HTML Content",
    OrmFieldName = "HTMLContent"
};
```

```
// Model Property
public String HTMLContent { get; set; }
```

Darstellung einer Ansicht

```
// Control
GridInDialogControl grid = new GridInDialogControl()
{
    Id = "[Insert Unique ID]".ToGuid(),
    Caption = "Grid",
    ControlName = "Grid",
    GridConfigurationGuid = "[INSERT GRID CONFIGURATION ID]"
};
```

Möchte man eine Ansicht haben, welche programmatisch definiert ist, kann eine dynamische Konfiguration angelegt werden. In diesem Fall kann auch ein eigener Datenprovider für die Ansicht erstellt werden.

Erfassung einer Anschrift

```
// Control
AddressControl address = new AddressControl()
{
    Id = "[Insert Unique ID]".ToGuid(),
    Caption = "Address",
};
address.GetControls();
address.AddressLine1Control.OrmFieldName = "AddressLine1";
address.AddressLine2Control.OrmFieldName = "AddressLine2";
address.AddressLine2Control.Visible = true;
address.PostalCodeControl.OrmFieldName = "PostalCode";
address.CityControl.OrmFieldName = "City";
address.StateControl.OrmFieldName = "State";
address.CountryControl.OrmFieldName = "Country";
```

```
// Model Property
public String AddressLine1 { get; set; }
public String AddressLine2 { get; set; }
public String PostalCode { get; set; }
public String City { get; set; }
public String State { get; set; }
public EnumCountriesValue Country { get; set; }
```

Erfassung eines vollständigen Personennamens

```
// Control
NameBlockControl name = new NameBlockControl()
{
    Id = "[Insert Unique ID]".ToGuid(),
    Caption = "Name"
};
name.GetControls();
name.SalutationControl.OrmFieldName = "Salutation";
name.SalutationTitleControl.OrmFieldName = "SalutationTitle";
name.TitleControl.OrmFieldName = "Title";
name.FirstNameControl.OrmFieldName = "FirstName";
name.MiddleNameControl.OrmFieldName = "MiddleName";
name.LastNameControl.OrmFieldName = "LastName";
name.SuffixControl.OrmFieldName = "Suffix";
```

```
// Model Property
public EnumSalutationsValue Salutation { get; set; }
```

```
public EnumTitlesValue SalutationTitle { get; set; }
public List<EnumTitlesValue> Title { get; set; }
public String FirstName { get; set; }
public String MiddleName { get; set; }
public String LastName { get; set; }
public EnumSuffixesValue Suffix { get; set; }
```

Container für Tabs

```
// Control
TabContainerControl tabContainer = new TabContainerControl
{
    Id = "[Insert Unique ID]".ToGuid(),
    Caption = "",
    WrapContentAtWidth = 500,
    StretchLastItem = true,
    ColSpan = 1,
    ColumnCount = 1
};
```

Einzelner Tab

```
// Control
TabControl tab1 = new TabControl()
{
    Id = "[Insert Unique ID]".ToGuid(),
    Caption = "Tab 1",
    ColSpan = 1,
    ColumnCount = 3, // biggest size
};
```

Button Gruppe

```
// Control
ButtonGroupControl buttonGroup = new ButtonGroupControl()
{
    Id = "[Insert Unique ID]".ToGuid(),
    NumberOfColumns = 2,
};
```

Button

```
// Control
ButtonControl button = new ButtonControl()
{
    Id = "[Insert Unique ID]".ToGuid(),
    Caption = "Button",
    MethodIdToExecuteOnButtonClick = "BA.Training.ClientActionMyAction",
};
```

9.9. Übung 5

Toastermeldungen

Überprüfen Sie in der TypeScipt Aktion, ob eine Bemerkung vorhanden ist, wenn nicht zeigen Sie einen entsprechenden Fehler-Toast. Benutzen Sie dafür eine Übersetzung,

Gleiches, wenn Sie im Igniter die Funktion nicht ausführen können und zusätzlich informieren Sie den Anwender wie viele Datensätze erfolgreich geändert wurden.

Freier Dialog

Implementieren Sie einen Dialog im Verzeichnis “Dialogs” in dem Sie den Anwender nach Datum und Bemerkung des Service Abfragen. Diese Werte übergeben Sie dem Igniter zum Setzen der Werte. Beachten Sie die Regel das der Nutzer des Dialoges, nichts von der Struktur des Dialoges wissen muss. Also geben sie die Vorgabewerte in den Dialog hinein und erhalten die Dialogwerte über das Dialogergebnis, um sie an den Igniter weiterzugeben.

Der Dialog sollte einen OK und einen Cancel Button haben.

Layout

Fügen Sie dem Dialog eine Gruppe hinzu und verschieben Sie die beiden Eingabefelder in die Gruppe.

Geben Sie dem Dialog eine Breite und fügen Sie der Gruppe und den Feldern ein adaptives Layout hinzu.

Fügen Sie weitere Eingabefelder und Gruppe(n) hinzu und testen Sie verschiedene Einstellungen beim Layout.

Button

Fügen Sie einen weiteren Button hinzu und öffnen Sie eine Messagebox zur Texteingabe. Diesen Text schreiben Sie dann in das Bemerkungsfeld.

Lösung

10. Berechtigungen

Der aktuelle Benutzer bestimmt die Berechtigungen. Diese werden im Folgenden beschrieben. Neben den normalen Benutzern gibt es den “System User”, dieser hat prinzipiell alle Rechte zum Lesen, Erstellen, Bearbeiten und Löschen.

Bei der Implementierung eigener Funktionalitäten, muss man überlegen, mit welchen Berechtigungen die Funktionalität ausgeführt wird. Beispielsweise kann man sie im Kontext des System User ausführen.

```
using (Api.User.NewSystemUserContext())
{ }
```

Es ist ebenfalls möglich die Funktionalität mit den Berechtigungen eines anderen Benutzers auszuführen.

```
using (Api.User.NewLocalUserContext(currentUserGuid))
{ }
```

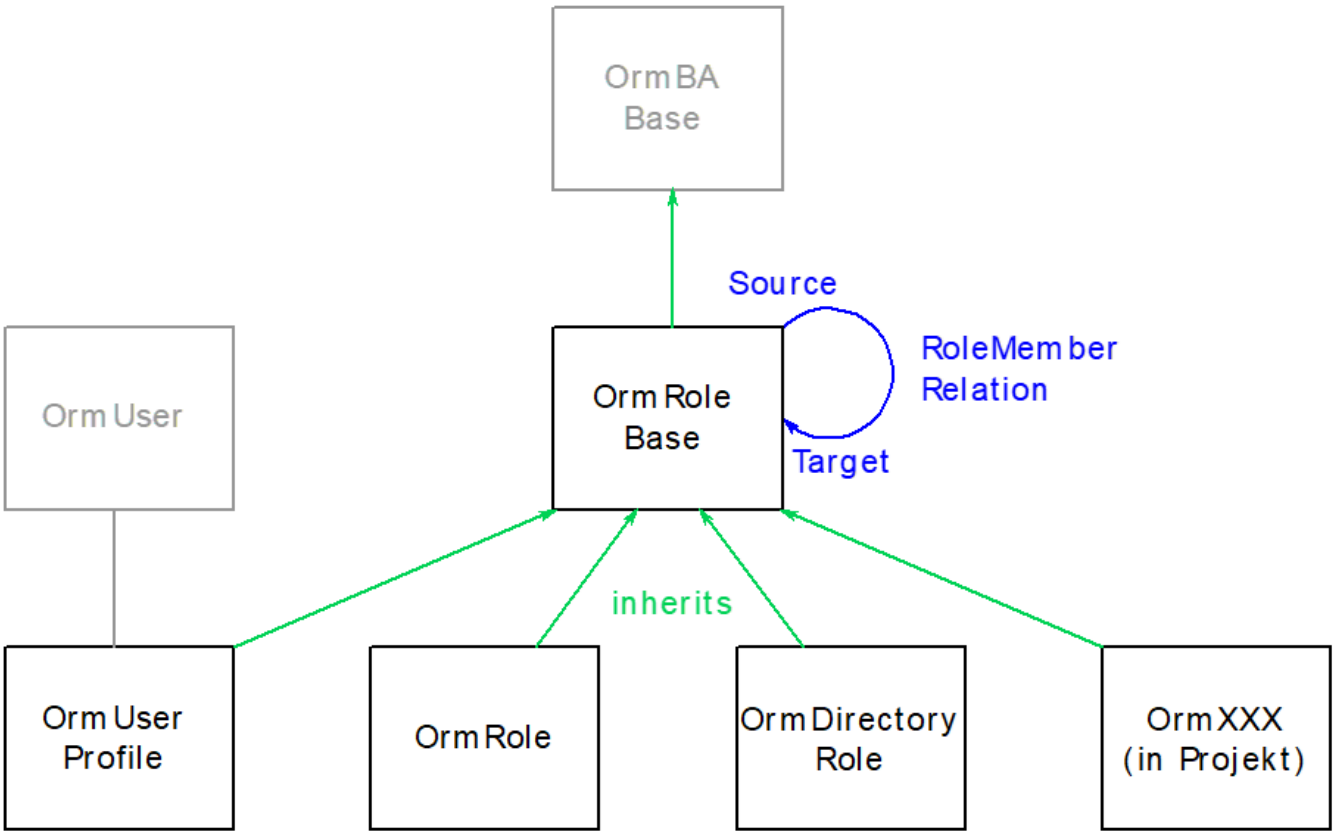
Wenn man einen Thread startet, kennt dieser den aktuellen Benutzer nicht. Einen Thread mit den Berechtigungen erhält man mit

```
Task async = Api.User.RunWithContext(() => { ...; });
```

Wird eine eigene Library implementiert, in der man keine Informationen darüber hat, ob sie grade im Vordergrund oder im Hintergrund ausgeführt wird, kann man einen Thread davon abhängig starten, ob die Ausführung gerade im Vordergrund stattfindet oder nicht.

```
Api.User.RunWithContextIfForeground(() => { ...; })
```

10.1. Rollen und Benutzer



- Alle Berechtigungsobjekte erben von Basis.Rolle (OrmRoleBase). Das sind allen voran Benutzer und Rollen. Die Datentabelle Basis.Rolle ist nur eine logische Gruppierung und enthält weder eigene Felder noch existiert eine SQL Tabelle mit Daten.
- Nur Datensätze auf Basis.Rolle können Berechtigungen erteilt werden.
- Alle Berechtigungen können beliebigen Basis.Rolle Datensätze erteilt werden. Das schließt auch eigene Berechtigungsklassen in Projekten ein.
- Rollen können andere Rollen enthalten. Das bewirkt im Ergebnis dasselbe, als wenn der Benutzer rekursiv alle Rollen direkt zugeordnet hätte. Dabei wird der “Rollenmitglieder” Relation immer nur in der Richtung Quelle -> Ziel gefolgt, also der Benutzer hat alle Rollen in denen er Quelle ist oder in denen eine seiner Rollen selbst Quelle ist.
- Berechtigungsobjekte können allgemein andere Berechtigungsobjekte enthalten. Also nicht nur Rollen andere Rollen. (siehe Beispiele)
- Die “Rollenmitglieder” ist eine hierarchische Relation. Dabei darf es nicht zu Schleifen kommen. Also Rolle A ist in Rolle B und Rolle B ist in Rolle A ist verboten.
- Änderungen an Rollenmitgliedschaften wirken sich erst nach dem nächsten Login der betroffenen User aus.
- Die Datentabellen zu den Berechtigungsobjekten (Benutzerprofil, Rolle …) können genau wie andere Datentabellen vom Konfigurator oder in Projekten erweitert werden.
- In Projekten können auch komplett neue Berechtigungsobjekte erstellt werden.

10.2. Berechtigungen für Datensätze

Lesen

Die Berechtigung Lesen erlaubt es Datensätze in Ansichten zu sehen, sie in Masken zu öffnen und sie in Relationscontrols auszuwählen.
Die Berechtigung verhindert nicht, dass man Teile des Datensatzes über existierende Relationen sieht. Ferner kann man eine bestehende Relation immer sehen, auch wenn man sie nicht erstellen könnte. Man kann sie auch unverändert speichern oder entfernen, nur nicht neu hinzufügen.

Ändern

Die Berechtigung zum Ändern erlaubt zusätzlich zum Lesen das Speichern existierender Datensätze, sowie das Öffnen derselben im Bearbeiten-Modus einer Maske.
Ein Objekt, das man nicht lesen kann, kann man auch nicht verändern.

Löschen

Die Berechtigung zum Löschen erlaubt zusätzlich zum Ändern das Löschen von existierenden Datensätzen.

Erstellen

Die Berechtigung zum Erstellen erlaubt es, neue Datensätze einer bestimmten Datentabelle zu erzeugen. Diese Berechtigung ist unabhängig von allen anderen Rechten. Man kann also auch Sätze erstellen, die man danach selbst nicht mehr lesen darf.

10.3. Vergabe von Berechtigungen

Objektberechtigungen können auf folgenden Ebenen erteilt werden:

1. Pro Datentabelle können hinreichende Rechte für alle Datensätze der Datentabelle erteilt werden. Dieses Recht übersteuert alles andere.
   Die Vergabe erfolgt in der Konfiguration der Datentabelle.
2. Pro Datentabelle können Basis-Rechte für alle Datensätze erteilt werden. Diese Berechtigung ist erforderlich, um mit irgendeinem Datensatz einer Datentabelle etwas machen zu können.
   Die Vergabe erfolgt in der Konfiguration der Datentabelle.
3. Pro Datensatz können Lese- und Bearbeiter-Rechte erteilt werden.
   Diese Rechte manifestieren sich in der Leser- und Autoren-Relation. Das Berechtigungsobjekt ist dabei Quelle und der Datensatz Ziel (genau wie bei der Rollenmitgliedschaft). Die Pflegbarkeit dieser Rechte hängt an den jeweiligen Maskenkonfigurationen.
   Wenn zu einem Datensatz keine einzige Leser- bzw. Autoren-Relation existiert, gelten alle Benutzer als berechtigt, vorausgesetzt, dass sie die Basis-Rechte aus Punkt 2 haben. Sobald mindestens eine Relation des jeweiligen Typs existiert, gilt die Schnittmenge der Basis-Rechte mit den zugewiesenen Benutzern und Rollen.

> In der Konfiguration der Datentabelle kann festgelegt werden, das der aktueller Benutzer automatisch in die Autoren-Relation eingetragen wird (Standardverhalten in BA.CRM), damit hat automatisch nur der aktuelle Benutzer (Falls er die Berechtigung aus Punkt 2 hat) und die Anwender mit der Berechtigung aus Punkt 1 Bearbeitungsrechte auf diesen Datensatz.

Die Autorenrelation ist `EnumRelationType.Author` und verwendet zur Aufteilung die Relationskategorie `EnumAuthorRelationSubTypes`, wobei `CreatedBy` verwendet wird, um den aktuellen Anwender als Autor einzutragen, falls dies in der Datentabellenkonfiguration eingestellt ist. Alle weiteren Autoren werden in `Default` eingetragen.Über die Erweiterung der Auswahlliste `EnumAuthorRelationSubTypes`, kann man sich eigene Gruppierungen erzeugen.

Die Leserrelation `EnumRelationType.Reader` und verwendet zur Aufteilung die Relationskategorie `EnumReaderRelationSubTypes`, wobei `CreatedBy` im Gegensatz zur Autorenrelation nicht automatisch belegt wird. Alle weiteren Leser werden in `Default` eingetragen.Über die Erweiterung der Auswahlliste `EnumReaderRelationSubTypes`, kann man sich eigene Gruppierungen erzeugen.

10.4. Berechtigungsprüfungen

Ebenen der Berechtigungsprüfung

1. Benutzerinterface (UI)
   In der obersten Ebene prüft das Benutzerinterface Rechte und deaktiviert beispielsweise Aktionen, die man ohnehin nicht ausführen darf.
   Diese Art der Berechtigungsprüfung gilt als unsicher, weil sie vom Endanwender durch Manipulation im Browser umgangen werden kann. Projekte, die eigene Steuerelemente implementieren, müssen diese Prüfungen im Allgemeinen auch überprüfen.
2. Implementierung von Funktionen (serverseitig)
   Wenn eine Aktion ausgeführt wird, prüft ihre Implementierung ihrerseits ein zweites Mal die Berechtigungen.
   Diese Prüfung kann vom Endanwender nicht umgangen werden. Programmierungen in Projekten können und müssen diese Prüfungen aber selbst durchführen.
3. Prüfungen in BA
   Diese Prüfungen können auch in Projekten durch Programmierung nicht umgangen werden.
   Dazu gehört zum Beispiel das Prüfen der Schreib bzw. Löschrechte beim Speichern von Datensätzen oder auch der Leserechte beim Öffnen einer Maske.

Automatische Berechtigungsprüfungen

Schreibrechte

Alle modifizierenden Berechtigungen auf Datensätze werden vom Core automatisch geprüft.
Das bedeutet, dass man sich in Projekten bei der Implementierung von Funktionen (Ebene 2) im Allgemeinen nicht unbedingt um die Prüfung der Schreibrechte kümmern muss es aber im Sinne einer sinnvollen Meldung an den Benutzer tun sollte.
Allerdings ist es oft angebracht, bei den zugehörigen Aktionen vorab (Ebene 1) eine Prüfung durchzuführen, um dem Benutzer im Benutzerinterface bereits zu signalisieren, dass er etwas nicht darf.
Die Steuerelemente in BA implementieren alle sinnvollen Berechtigungsprüfungen in Ebene 1 und 2. Nur bei Erweiterungen muss man sich damit befassen.

Leserechte

BA prüft Leserechte bei API-Aufrufen im Allgemeinen nicht (im Gegensatz zu den Schreibrechten).
Ausnahmen:

1. Platzhalter (PropertyResolver)
   Bei der Berechnung von Werten für Platzhalter in Vorlagen erfolgt eine Prüfung der Leserechte, wenn dabei über Relationen auf andere Datensätze zugegriffen wird. Fehlen dabei irgendwelche Leserechte, so wird der Platzhalter immer leer. Dies ist erforderlich, weil sich Anwender mit Bearbeiterrechten für Vorlagen sonst Zugriff auf beliebige Felder unlesbarer Datensätze verschaffen könnten.
2. Ansichten
   Die Ansichten prüfen die Leserechte auf die angezeigten Zeilen. Sie prüfen nicht, wenn man über

Relations-Spalten auf fremde Datensätze zugreift. Es obliegt dem Konfigurator auf diese Weise keine sensiblen Daten zu exponieren.

3. API-Funktionen für Leserechte
   Natürlich gibt es API-Funktionen, die nur oder auch für Leserechte vorgesehen sind. Aber nur bei Funktionen, wo das bereits aus dem Funktions- oder Parameternamen hervor geht, findet eine solche Prüfung statt.

Sonstige Rechte

Alle sonstigen Rechte werden von BA in Ebene 2, also der Implementierung der jeweiligen Funktionen, geprüft. Wenn man eigene Funktionen in Projekten implementiert, muss man sich auch um diese Prüfungen kümmern.

10.5. Eigene Berechtigungsprüfungen

Rechte auf Datentabelle prüfen

Für die Entwicklung eigener Steuerelemente kann es vor allem für Ebene-1-Prüfungen nötig sein, zu prüfen, ob der Benutzer Datensätze einer Tabelle grundsätzlich benutzen darf. Dazu dient die Funktion

```
bool Api.User.CurrentUserIsAllowed(Guid dataSourceId, EnumTableOperations acti
on).
```

Der Parameter `dataSourceId` ist die Datentabelle, für die die Berechtigung geprüft werden soll. Wenn hier eine Basis-Datentabelle angegeben wird, ist die Antwort immer „ja“.
Der Parameter `action` definiert, welche Berechtigung abgefragt werden soll (Create, Read, Edit, Delete). Implizite Berechtigungen werden dabei immer automatisch mit geprüft. Also wenn man Delete abfragt, bekommt man auch dann „nein“ als Antwort, wenn der User zwar Datensätze dieses Typs löschen dürfte, diese aber nicht bearbeiten darf (Edit), was zum Löschen erforderlich ist.

Beispiele:

```
bool userCanCreate = Api.User.CurrentUserIsAllowed(EnumDataSource.Email, EnumT
ableOperations.Create);
```

Es können auch mehrere Berechtigungen auf einmal geprüft werden:

```
bool userCanCreateAndEdit = Api.User.CurrentUserIsAllowed(EnumDataSource.Compa
ny, EnumTableOperations.Create | EnumTableOperations.Edit);
```

Das liefert nur dann „ja“, wenn der Benutzer Firmen anlegen und danach auch wieder editieren darf.

Datensatz-Rechte prüfen

Beim Implementieren von Aktionen oder Hintergrundprozessen (Ebene 2) muss man evtl. die Berechtigungen für einzelne Datensätze explizit prüfen. Dazu dient die Funktion

```
bool OrmBABase.IsAllowed(EnumTableOperations action).
```

Der Parameter `action` definiert, welche Berechtigung abgefragt werden soll (Read, Edit, Delete). Create ergibt an dieser Stelle keinen Sinn, siehe unten.
Die Berechtigungsprüfung ist sehr umfänglich und berücksichtigt alle direkten oder indirekten Abhängigkeiten. Diese sind

1. Hat der Benutzer überhaupt Rechte auf die Datentabelle? (Basis-Rechte)
2. Darf der Benutzer die jeweilige Aktion (Lesen oder Bearbeiten) für genau diesen Datensatz? (Falls er nicht bereits hinreichende Rechte für den Datentyp hat.) Es wird also ggf. geprüft, ob er unter den

Lesern oder Autor/Bearbeitern ist.
3. Benötigt der Benutzer für das angeforderte Recht weitere Rechte? Also z.B. Delete impliziert Edit und Read, und die beiden letzteren müssen ggf. auch für diesen Datensatz vorhanden sein.
4. Handelt es sich um einen neuen, noch nicht gespeicherten Datensatz? In diesem Fall wird die Create-Berechtigung benötigt, wenn man Edit abfragt. Ein neuer Datensatz ist ein Datensatz, der nie gespeichert wurde.

Beispiele:
Leserechte auf einen Datensatz explizit prüfen: (Das macht BA nicht automatisch.)

```
OrmBABase orm = …;
if (!orm.IsAllowed(EnumTableOperations.Read))
    // Zugriff verweigert
Darf ich diesen Datensatz bearbeiten?
OrmBABase orm = …;
if (!orm.IsAllowed(EnumTableOperations.Edit))
    // Zugriff verweigert
```

Diese Prüfung funktioniert auch für neue Datensätze, die gerade angelegt werden. In diesem Fall wird nur die Create-Berechtigung für den Datentyp geprüft.

Leserechte in Datenprovider / Abfragen

Bei GetQuery()

Die API-Funktion `Api.ORM.GetQuery` liefert immer alle Datensätze unabhängig von den Leserechten. Wenn man die Leserechte berücksichtigen möchte, muss man das explizit tun. Dazu gibt es die Funktionen

```
IQueryable<T> Api.ORM.GetQueryWithReadPermissions<T>(Session session, IEnumera
ble<Type> dataSources = null) where T : OrmBABase
IQueryable<OrmBABase> Api.ORM.GetQueryWithReadPermissions(Guid type, Session s
ession, IEnumerable<Type> dataSources = null)
IQueryable<OrmBABase> Api.ORM.GetQueryWithReadPermissions(Type ormType, Sessio
n session, IEnumerable<Type> dataSources = null)
```

Der Parameter `T`, type bzw. `ormType` gibt den gewünschten Datentyp an.
Der optionale Parameter `dataSources` kann (und sollte) benutzt werden, wenn eine Abfrage auf einen Basis-Datentabelle wie Standard (OrmBABase) oder Basis.Vorgang erfolgt, aber nicht alle potentiellen Datentypen gewünscht sind. Das kann beispielsweise vorkommen, wenn in einer kombinierten Ansicht einige einzelne Datentypen explizit ausgewählt sind.
Wenn der Parameter angegeben wird, wird die Ergebnismenge automatisch auf die entsprechenden Datentabellen eingeschränkt. Gleichzeitig kann die Prüfung der Leserechte für alle anderen Datentabellen entfallen. Das ist performanter.

Beispiele:

Im einfachsten Fall braucht man nur die Datensätze einer Datentabelle unter Berücksichtigung der Leserechte. Das kann auch eine Basis-Datentabelle sein.

```
IQueryable<OrmActivityBase> emailAddresses = Api.ORM.GetQueryWithReadPermissio
ns<OrmActivityBase>(session);
```

Wenn der gewünschte Datentyp dynamisch aus der Konfiguration kommt:

```
IQueryable<OrmBABase> records = Api.ORM.GetQueryWithReadPermissions(wantedData
SourceGuid, session);
```

Wenn mehrere Datentypen dynamisch aus der Konfiguration kommen:

```
IEnumerable<Guid> configuredTypeGuids = …;
IEnumerable<Type> types = configuredTypeGuids.Select(ff => Api.ORM.GetOrmTypeC
acheValue(ff).Type);
IQueryable<OrmBABase> records = Api.ORM.GetQueryWithReadPermissions<OrmBABas
e>(session, types)
```

Bei IQueryable<>

Falls man bereits eine fertige Query hat und die Prüfung der Leserechte nur nachträglich hinzufügen möchte:

```
IQueryable<T> Api.ORM.ApplyReadPermissions<T>(IQueryable<T> query, Session ses
sion, IEnumerable<Type> dataSources = null) where T : OrmBABase
```

Die Funktion verhält sich identisch wie GetQueryWithReadPermissions, nur dass sie auf einer bestehenden Query aufsetzt und auf lesbare Sätze sowie optional gewünschte Datentypen einschränkt.

Beispiel:
Um die Leserechte optional hinzuzufügen geht man so vor:

```
IQueryable<T> query = Api.ORM.GetQuery<T>(session).Where(ff => …);
if (readPermissions)
    query = Api.ORM.ApplyReadPermissions<T>(query, session);
```

Bei Formeln (CriteriaOperator)

In manchen Fällen benötigt man die Einschränkung für die Leserechte als Criteria Operator und nicht als LINQ-Expression. Dafür gibt es die Funktion

```
CriteriaOperator Api.ORM.ApplyReadPermissions(CriteriaOperator criteria, Type
queryOrmType, GuidSet sourceTypes = null)
```

Diese Verhält sich analog zu `ApplyReadPermissions<T>` nur, dass hier ein bestehender Criteria Operator, der vom Typ Boolean sein muss, so modifiziert wird, dass die Leserechte berücksichtigt werden.
Der Parameter `queryOrmType` ist der Typ der Datensätze, auf die der Criteria Operator angewendet werden soll. Das kann eine Basis-Datentabelle sein, aber dann müssen die Rechte für alle davon erbenden Typen geprüft werden, falls dies nicht durch den Parameter `sourceTypes` weiter eingeschränkt wird.

Beispiel:

```
filterCriteria = Api.ORM.ApplyReadPermissions(filterCriteria, dataSourceType);
```

Benutzerrechte explizit prüfen

Um zu prüfen, ob ein Benutzer Mitglied einer oder mehrerer Rollen ist, gibt es die Funktionen

```
bool Api.User.CurrentUserIsInRole(Guid role, bool defaultAllowed = false)
bool Api.User.CurrentUserIsInRole(IReadOnlyCollection<Guid> roles, bool defaul
tAllowed = false)
```

Erstere prüft, ob der aktuelle Benutzer genau eine Rolle hat, zweitere prüft, ob der User mindestens eine Rolle aus einer Menge hat.
Der Parameter `defaultAllowed` gibt an, wie sich die Funktionen verhalten sollen, wenn `Guid.Empty` bzw. gar keine Rolle oder null übergeben wird. Mit `true` kann man erreichen, dass in diesem Fall jeder Benutzer berechtigt sein soll.

Beispiele:

```
bool currentUserCanEdit = Api.User.CurrentUserIsInRole(EditorRoles, true);
```

In diesem Fall darf jeder bearbeiten, wenn `EditorRoles` leer ist.
Falls man die Rollen nur als String mit GUIDs zur Verfügung hat, kann man `RoleSet.Parse` verwenden.

```
bool currentUserCanEdit = Api.User.CurrentUserIsInRole(RoleSet.Parse(EditorRol
esString), true);
```

Die Funktion `RoleSet.Parse` kann auch mit leeren Strings und null umgehen. Ferner ist die Groß-Kleinschreibung egal, ebenso wie das Trennzeichen.

Eigene Rolle

Projekte können auch komplett eigene Rollen-Datentabellen anlegen. Dazu muss eine Datentabelle angelegt werden (wahlweise mit XPO-Designer), die von OrmRoleBase (Basis.Rolle) erbt.

Sobald Objekte dieser Klasse existieren, erscheinen diese Objekte auch in der Rollenverwaltung. Beim Doppelklick auf eine Zeile dieses Typs wird die Standardmaske laut Konfiguration geöffnet.
Des Weiteren können die Objekte auch an allen Stellen der Anwendung, wo Berechtigungen vergeben werden können, zugewiesen werden. Das sind

- die Rollen eines Benutzerzugangs,
- die Mitglieder von Rollen,
- Berechtigungen in Datentypkonfigurationen,
- Berechtigungen für Masken- und Ansichtensteuerelemente,
- Leser und Bearbeiter von Datensätzen und
- Berechtigungen für Auswahllistenwerte.

Bei Berechtigungen, die in Masken gepflegt werden, ist zu beachten, dass diese üblicherweise eine Liste von Ansichten hinterlegt haben, die für die Berechtigungsauswahl zur Verfügung stehen. Darin ist nicht unbedingt die projektspezifische Datentabelle enthalten. Die Maskenkonfigurationen müssen daher angepasst werden, um auch die neue Tabelle zu erlauben.

Berechtigungsprüfung für eigenen Datentyp überschreiben

In Einzelfällen kann es erforderlich sein, für einen Datensatz eine abweichende Berechtigungsprüfung zu implementieren. Zu diesem Zweck kann die Methode `IsAllowed` für eigene Datentabellen oder erweiterte Datentabellen überschrieben werden.
Eine solche Überschreibung ersetzt alle Berechtigungsprüfungen des Core auf Einzeldatensatzebene und nur diese. Die Maßnahme greift sehr tief in den Core ein und sollte daher mit Vorsicht eingesetzt werden. Wenn man die Methode überschreibt, muss man sich auch um die Berechtigungsprüfung auf Datentabellenebene kümmern.

> Dies hat keine Auswirkung auf die Leserechteprüfung in Abfragen!

Beispiel:

```
public override bool IsAllowed(EnumTableOperations action)
{
    Guid userGuid = Api.User.GetUserGuidByProfile(this);
    // System user profile should not be deleted
    if ((action & EnumTableOperations.Delete) != 0 &&
       (userGuid == Constants.SystemUser || userGuid == Api.User.CurrentUserGu
id()))
        return false;
    if (userGuid == Api.User.CurrentUserGuid())
        action &= ~EnumTableOperations.Edit;
    return base.IsAllowed(action);
}
```

Im obigem Beispiel wird dem aktuellen Benutzer ein unbedingtes Bearbeiten-Recht auf das eigene

Benutzerprofil eingeräumt. Darüber hinaus wird das Löschen des System-Users und des eigenen Benutzerprofils unterbunden.
Letzteres wird durch das erste `if` erreicht. Letzteres dadurch, dass der Prüfgrund `Edit` aus `action` entfernt wird, wenn es das eigene Benutzerprofil ist.
Action ist ein sogenanntes Flags-Enum, dass mehrere Werte gleichzeitig annehmen kann (Siehe).

10.6. Performance-Aspekte

- Berechtigungsprüfungen auf bereits im Speicher befindlichen Datensätzen sind immer Constant-Time, also sehr schnell.
- **Massenprüfungen** von Datensätzen auf Berechtigungen sind **immer langsam**. Diese werden derzeit nur für Leserechte angeboten.

Leserechte

Für Leserechte wurden **besondere Optimierungen vorgenommen**, um auch Massenprüfungen einigermaßen schnell zu absolvieren.

Allerdings führt kein Weg daran vorbei, dass die Datenbank bereits, um nur die Anzahl der (lesbaren) Datensätze in einer Ansicht zu ermitteln, alle Datensätze einzeln prüfen muss. Die Geschwindigkeit der Leserechte skaliert also immer mit der Gesamtzahl der zu einer Ansicht gehörenden Datensätze. DevExpress ermittelt die Anzahl immer, bevor ein Grid angezeigt wird.

Wenn die **Leserechte für einzelne Datentabellen deaktiviert werden** (falls nicht benötigt), kann die Geschwindigkeit für diese massiv erhöht werden. Dazu muss in der Datentabellen-Konfiguration unter „Alles Lesen“ die Rolle „Jeder“ zugewiesen werden.

In Programmierungen sollte auch immer im Einzelfall gut überlegt werden, ob die Leserechte bei einer Query geprüft werden müssen. In manchen Fällen führen weitere Bedingungen bei der Query oder den nachfolgenden Joins dazu, dass ohnehin nur lesbare Sätze geliefert werden. **Doppelprüfungen sollten vermieden werden**, sofern der Benutzer nicht die Möglichkeit hat das Zwischenergebnis zu manipulieren (z.B. im Browser).

10.7. Übung 6

Konfiguration

Legen Sie Rollen an

- “Read All Engines”
- “Read Engines”
- “Edit All Engines”
- “Edit Engines”

Tragen Sie diese Rollen in die Datentabelle ein.

Legen Sie neue Benutzer an.

- Benutzer 1 mit “Read Engines”
- Benutzer 2 mit “Edit Engines”
- Benutzer 3 mit “Read Engines“und “Edit Engines”
- Benutzer 4 mit “Read All Engines” und “Edit Engines”
- Benutzer 5 mit “Read All Engines” und “Edit All Engines”

Fügen Sie in der Maske zwei Relationsauswahlelemente vom Typ Tokenbox hinzu

- Relationsdefinition: RelatedAdditionalAuthors
- Relationsdefinition: RelatedReaderRight

Überprüfen Sie wann welcher Benutzer einen Datensatz Bearbeiten bzw. Lesen kann.

Level 2 Prüfung

Bauen Sie in den Igniter eine Level 2 Prüfung ein.

Level 1 Prüfung

Bauen Sie in der Aktion eine Level 1 Prüfung ein.

Dialog nur für bestimmte Anwender

Erstellen Sie eine neue Rolle “Service Dialog” und vergeben Sie die Rechte “Read All Engines” und “Edit All Engines” an diese Rolle.

Erweitern Sie die Aktion, um eine Eigenschaft zur Auswahl von mehreren Rollen. Nur den gewählten Rollen wird erlaubt den Dialog zur Eingabe zu öffnen. Für alle weiteren Benutzer, wird der Igniter sofort ausgeführt.

Ändern Sie die Auswahl so ab, dass es nur möglich ist Rollen zu wählen, also keine Benutzerprofile oder Verzeichnisrollen.

Lösung

11. Hintergrundprozesse

Funktionalitäten mit erhöhtem Zeitbedarf oder zeitgesteuerte Funktionen sollten als Hintergrundprozesse implementiert werden. Über entsprechende Protokolle kann z.B. die Fehlersuche deutlich vereinfacht werden. Hintergrundprozesse können mit Benutzer- oder Systemrechten ausgeführt werden.

11.1. Allgemeines

Der Work-Manager

Der Work-Manager ist Teil der BA Core Infrastruktur. Er verwaltet alle geplanten, laufenden oder beendeten Hintergrundprozesse. Dabei achtet er auf Priorisierung und maximalen Ressourcenverbrauch.

Ablauf des Work-Managers

Start

- Beim Start der Anwendung werden alle Work-Items aus der Datenbank ausgelesen, die nicht abgeschlossen sind.
  1. Bei angeblich laufenden, also abgebrochenen Elementen wird der Status nun auf `Aborted` gesetzt und der Work-Manager führt asynchron die `WorkItemFinished` Methode aus.
  2. Bei wartenden Work-Items wird geprüft, ob sie bereit zur Ausführung sind, und der Status ggf. auf `Queued` geändert.
  3. Items, die sofort bereit sind, bleiben unverändert.
- Danach werden die verfügbaren Prozesse nach Priorität der Items an die Items im Status “bereit” verteilt.
- Zuletzt wird ein Timer für den Startzeitpunkt des nächsten wartenden Items gesetzt, falls eines existiert.

Abarbeitung

Der Work-Manager ist komplett Ereignisgetrieben. Er tut nur Etwas wenn

1. sich ein Item beendet. Dann prüft er ob ein anderes den freiwerdenden Prozess nutzen kann.
2. ein neues Item eingeplant oder aktualisiert wird. Dann prüft er, ob es sofort bereit zu Ausführung ist und ob dafür ein Prozess frei ist.
3. ein Item abgebrochen werden soll. Dann signalisiert er dem Item, den Wunsch abzubrechen. Dieses setzt daraufhin asynchron seinen Status auf `CancellingByUser`.
4. der Timer für die nächste Startzeit angesprochen hat. Dann setzt er den Status aller Items, deren Startzeitpunkt verstrichen ist auf `Queued` und prüft, ob dafür sofort ein Prozess frei ist. Anschließend wird der Timer auf das nächste Item neu programmiert.

Work-Items

Work-Items machen die eigentliche Arbeit. Work-Items sind Instanzen der serialisierbaren Worker-Klassen und Zeilen in Tabelle `OrmWorkItem`. Die Worker Klassen sind immer individuell für den Anwendungfall.

- Jede Art einer Aufgabe hat eine implementierende Klasse.
- Jede zu erledigende Aufgabe hat genau eine `InstanceGuid`. Die Aufgabe wird in der API immer über diese identifiziert.

- Zu einer Aufgabenart kann es beliebig viele Aufgaben geben.
- Jede Instanz eines Work-Items hat eine Oid. Diese ist nur intern.
- Eine Aufgabe kann in mehreren Teilschritten (Instanzen) erledigt werden, z.B. wegen Unterbrechung durch Anwendungsneustart.
- Instanzen entstehen, wenn sie eingeplant werden, und sind am Ende ihrer Lebensdauer, wenn sie Abgeschlossen oder aus irgendeinem Grund abgebrochen sind.
- Die Priorität eines Work-Items ist `SchedulingPriority` sowie die geplante Startzeit.

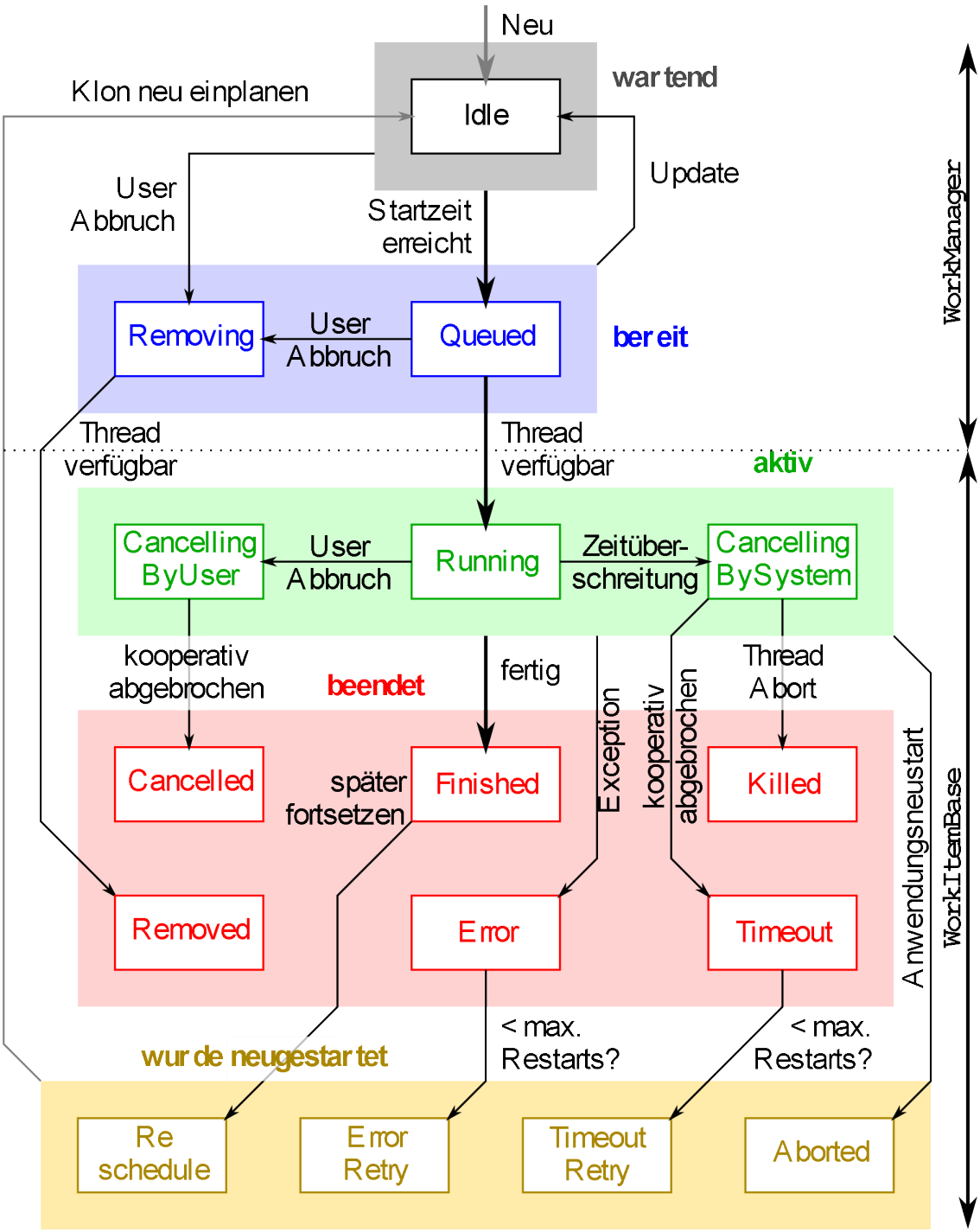
Mögliche Zustände

• Bei allen Zuständen, bei denen das WorkItem noch nicht (=nie) ausgeführt wurde, gehört die Instanz dem Work-Manager. Nur er darf Veränderungen vornehmen.
• Bei allen Zuständen ab dem Start der Ausführung gehört die Instanz dem WorkItem selbst. Nur dieses darf noch Veränderungen vornehmen.
• Ab dem Start der Ausführung ist der Zustand von Instanzen von Work-Items ein gerichteter Graph.
• Die Zustände gehören zu Kategorien, die sich bezüglich vieler Eigenschaften gleich verhalten.

Kategorie	Zustand	Beschreibung
Warten auf Startbedingung	Idle	Das Work-Item liegt in der Datenbank und wartet bis der Zeitpunkt für seine Ausführung gekommen ist.
Warten auf Ausführung	Queued	Das Work-Item soll ausgeführt werden und es ist bereits in der Warteschlange.
Warten auf Ausführung	Removing	Das Work-Item wurde von Benutzer (bzw. über die API) abgebrochen, bevor es gestartet wurde. Die Abbruchfunktion wird ebenfalls asynchron über die Warteschlange ausgeführt.
Laufend	Running	Das Work-Item wird gerade ausgeführt.
Laufend	CancellingByUser	Das Work-Item hat den Befehl zum Abbrechen bekommen und sollte sich in Kürze beenden.
Laufend	CancellingBySystem	Das Work-Item hat seine max. Ausführungszeit erreicht und wird in Kürze gekillt. Spätestens jetzt sollte es seine Arbeit geordnet beenden und ggf. einen nächsten Start einplanen.
Beendet	Finished	Das Work-Item hat seine Arbeit erfolgreich abgeschlossen.

Beendet	Removed	Das Work-Item wurde vor dem normalen Start abgebrochen und hatte Gelegenheit, um darauf zu reagieren.
Beendet	Cancelled	Das Work-Item wurde vom Benutzer (bzw. über die API) abgebrochen und hat sich selbst beendet.
Beendet	Error	Eine unbehandelte Ausnahme ist aufgetreten. Das WorkItem wird in Kürze neu gestartet (wenige male).
Beendet	Timeout	Das Work-Item hat auf den CancellingBySystem Status reagiert und sich selbst beendet.
Beendet	Killed	Das Work-Item hat seine Arbeit nicht binnen der gesetzten Zeit beendet und wurde vom Work-Manager gekillt – gefährlich!
Neustart	Reschedule	Das Work-Item wird (wegen eines Timeouts oder sonstiger Limitierungen) seine Arbeit bald wieder fortsetzen.
Neustart	ErrorRetry	Das Work-Item wurde nach einer Exception neu eingeplant und wird seine Arbeit bald in einer neuen Instanz fortsetzen.
Neustart	TimeoutRetry	Das Work-Item wurde nach einem Timeout neu eingeplant und wird seine Arbeit bald in einer neuen Instanz fortsetzen.
Neustart	Aborted	Das Work-Item wurde nach einem Abbruch durch einen Anwendungsneustart wieder eingeplant.

Statusübergänge



Scheduling-Prioritäten

Die Priorität eines Work-Items ergibt sich aus dem Wert von `ExpectedRuntime`. Es gibt folgende vordefinierten Prioritäten:

1. `Urgent` Work-Items mit dieser Priorität werden immer sofort ausgeführt, ohne Rücksicht auf irgendwelche Ressourcen. Die Maximale Laufzeit beträgt 1 Minute.
2. `Short` Diese Work-Items werden mit normaler CPU-Priorität in der Queue für normale Work-Prozesse

ausgeführt. Die Maximale Laufzeit beträgt standardmäßig 1 Minute.
3. `Normal` Work-Items mit “Normal” werden mit reduzierter CPU-Priorität in der Queue für normale Work-Prozesse ausgeführt. Die Maximale Laufzeit beträgt standardmäßig 15 Minuten.
4. `Long` Diese Work-Items werden mit reduzierter CPU-Priorität in einer separaten Queue für Langläufer ausgeführt. Die Maximale Laufzeit beträgt standardmäßig 5 Stunden.
   Per Konfigurationsdatei *Customer.config* auf dem Webserver können die maximalen Ausführungszeiten für die Prioritätsstufen angepasst werden.

Per programatischer Erweiterung von `EnumExpectedRuntime` können in Projekten zusätzliche Prioritätsstufen geschaffen werden. In den Eigenschaften des Auswahllistenwertes werden die Details angegeben.

Queues und Parallelität

Die Anzahl der gleichzeitig ausgeführten Work-Items (mit Ausnahme Priorität `Urgent`) ist begrenzt. Es wird eine Feste Anzahl von Work-Prozessen in 2 Queues vorgehalten:

1. Normal
   In dieser Queue werden nur Work-Items der Prioritäten Short und Normal ausgeführt.
   Standardmäßig stehen dafür ¼ der auf dem Webserver verfügbaren CPU-Kerne zur Verfügung.
2. Langläufer
   Die Queue für Langläufer führt alle Arten von Work-Items (außer Urgent) aus.
   Standardmäßig stehen dafür die Hälfte der auf dem Webserver verfügbaren CPU-Kerne zur Verfügung.

Der Start der Work-Items erfolgt immer nach Priorität. Wenn also ein Prozess frei ist, wird immer das nächste Short-Item zuerst ausgeführt, auch wenn es ein Platz in der Langläufer-Queue ist. Ein bereits laufendes Work-Item wird aber nicht unterbrochen, um einem höher priorisiertem Work-Item Platz zu machen. Nur Durch die Reservierung von Prozessen in der Normal-Queue ist sichergestellt, dass Langläufer nicht alle höher priorisierten Items aufhalten können.
Per Konfigurationsdatei *Customer.config* auf dem Webserver kann die Anzahl der parallelen Prozesse pro Queue verändert werden.

Lebenszyklus von Work-Items

- Eine Instanz entsteht nur, wenn sie über die Work-Manager-API eingeplant wird.
- Eine noch nicht gestartete Instanz kann über eine erneute Einplanung mit derselben `InstanceGuid` beliebig verändert werden.
- Für jede Instanz (gleiche `Oid`) wird genau einmal die Methode `WorkItemFinished` aufgerufen, auch bei einem Abbruch vor dem Start.
- Wenn eine Instanz vor dem Start aktualisiert wird, wird das nicht als neue Instanz betrachtet. Deshalb wird für die alte, überschriebene Version nicht `WorkItemFinished` aufgerufen. Das ist die einzige Konstellation, in der `WorkItemFinished` nicht für jede technische Instanz des Work-Items aufgerufen wird.
- Jede Instanz endet immer in einem finalen Status oder mit einem Neustart.

Informationen über Work-Items

Neben dem vollen Work-Item gibt es eine abgespeckte Version von Work-Item-Instanzen: `WorkItemProgress`. Diese wird verwendet um Informationen über ein Work-Item zu liefern. Diese Informationen entstammen vollständig den nicht serialisierten Spalten aus `OrmWorkItem` und sind von außen nur lesbar. Nur das Work-Item selbst darf einige der Informationen verändern.

Laufzeitüberwachung

Sobald laufende Work-Items die maximale Laufzeit überschreiten bekommen sie das Signal `CancellingBySystem`. Sie können auf dieses Signal mit einer Frist von standardmäßig 5 Minuten reagieren, indem sie sich selbst beenden. Sie gehen dann in den Status `TimeOut` über. Standardmäßig wird die Implementierung von `WorkItemFinished` sie daraufhin `MaxNumberOfRestarts` mal neu einplanen.

Reagieren sie nicht in der gesetzten Frist, wird der Worker Thread hart gekillt, indem eine `ThreadAbortException` injiziert wird. Der Status des Work-Items wird danach auf `Killed` gesetzt. Dieser Zustand sollte auf jeden Fall vermieden werden, da er den Datenbestand unter Umständen in einem inkonsistenten Zustand zurücklassen kann.

Die Methode `WorkItemFinished` wird aber dennoch noch aufgerufen. Aber auch diese Ausführungszeit ist auf eine Minute begrenzt.

11.2. Die Work-Manager API

Der Work-Manager ist über `Api.Worker.` erreichbar.

Work-Item erzeugen oder aktualisieren

```
bool CreateOrUpdate(WorkItemBase item, Session session = null)
```

Die Methode erstellt ein neues Work-Item oder verändert es. Das Work-Item muss vorher mit `new` erzeugt worden sein. Beispiel:

```
Api.Worker.CreateOrUpdate(new UpdateRecordCollection(collection, recordGuids,
UpdateRecordCollection.WorkType.Add));
```

Optional kann einen XPO-Session angegeben werden, unter der der Datenbankzugriff erfolgen soll. Wenn dieses eine `UnitOfWork` ist, dann erfolgt die Einplanung bzw. Änderung erst, wenn für diese `CommitChanges` aufgerufen wird. Auf diese Weise können auch mehrere Work-Items atomar verändert werden. Das ist im Besonderen beim Einplanen von Nachfolgern wichtig, damit die Kette niemals abreißen kann.
Dabei ist allerdings zu beachten, dass dadurch verzögerte Exceptions auftreten können, beispielsweise eine `LockingException`, wenn ein zu aktualisierendes Work-Item mittlerweile gestartet oder abgebrochen wurde.

Der Rückgabewert sagt, ob die Erzeugung erfolgreich war.

- `true` Work-Item angelegt oder aktualisiert.
- `false` Das Work-Item mit dieser `InstanceGuid` läuft bereits und konnte nicht mehr aktualisiert werden.
  Falls es sich um ein wiederkehrendes Work-Item handelt, ist es dessen Aufgabe bei der Neueinplanung in `WorkItemFinished` ggf. aktualisierte Parameter zu verwenden.

Nach einer erfolgreichen Erstellung hat der Work-Manager im übergebenen Work-Item das Feld `Oid` ausgefüllt. Entweder mit einer neuen `Oid` (Erstellung) oder mit der des aktualisierten Workers.

Es ist zulässig, beim Aktualisieren eines Work-Items wirklich alle Eigenschaften zu ändern. Dazu zählt auch, dass die Implementierende Klasse jetzt eine komplett andere sein kann. Entscheidend ist nur die Instance Guid. Die `Oid` bleibt auch bei diesem Vorgang erhalten.

Laufendes oder geplantes Work-Item abbrechen

```
bool StopExecutionOfWorkItem(Guid instanceGuid, bool cancelRunning = true)
```

Mit dieser Methode kann ein bereits eingeplantes oder laufendes Work-Item abgebrochen werden. Wenn `cancelRunning = false` ist, wird der Abbruch nur ausgeführt, wenn das Work-Item noch nicht läuft.

- Wenn das Work-Item schon läuft, wird der Status auf `CancellingByUser` gesetzt und es bekommt ein Signal (`WorkItemShouldFinish`). Darauf sollte es reagieren und sich beenden (Run-Methode kehrt zurück). Danach wechselt der Status auf `Cancelled`.
- Wenn das Work-Item noch nicht gestartet wurde, wird die Methode `WorkItemFinished` mit Status `Removed` aufgerufen.

Als Rückgabewert erhält man `true`, wenn der Abbruch in die Wege geleitet wurde und `false`, wenn kein aktives Work-Item mit dieser `InstanceGuid` (mehr) gefunden wurde.
Der Abbruch selbst erfolgt immer asynchron. Das bedeutet, wenn die Methode zurückkehrt, ist der Abbruch i.a. noch nicht vollzogen. Allerdings ist der Status schon geändert und ggf. sind die Fortschrittsanzeigen informiert.
Normalerweise sollte die Methode nur aufgerufen werden, wenn das Work-Item `IsCancellable = true` hat. Dies wird aber nicht überprüft. Es obliegt dem Work-Item, was es in diesem Fall macht. Den Abbruch bevor es gestartet wurde, kann es aber nicht verhindern. Es könnte sich allenfalls in `WorkItemFinished` neu einplanen.

Informationen über Work-Items

```
List<WorkItemProgress> GetWorkItems(Guid user, params EnumWorkItemStateCategory[] categories)
```

Mit obiger Methode können Informationen zu allen im System noch verfügbaren Work-Items abgerufen werden. Das umfasst auch historische (beendete) Work-Items, sofern diese noch nicht vom `CleanUpWorkItems` Task aufgeräumt wurden.
Der Parameter `user` gibt an, für welchen User die Work-Items abgerufen werden sollen. Wenn dieser `Guid.Empty` ist, werden die Work-Items aller User abgerufen. Items mit `IsVisibleForAllUsers` werden unabhängig von diesem Parameter zurückgeliefert.

Die folgende Liste gibt an, welche Statuskategorien abgefragt werden sollen. Zur Auswahl stehen:

1. `EnumWorkItemStateCategory.Waiting` Alle Work-Items, die noch nicht bereit zur Ausführung sind – derzeit nur, weil der Startzeitpunkt noch nicht erreicht ist.
2. `EnumWorkItemStateCategory.Ready` Alle Work-Items, die bereit zur Ausführung sind, aber noch nicht zum Zuge kamen, weil nicht genug Prozess-Ressourcen zur Verfügung stehen. Das schließt auf Work-Items ein, die vor dem Start abgebrochen wurden, aber noch keine Gelegenheit hatten, darauf zu reagieren.
3. `EnumWorkItemStateCategory.Active` Alle laufenden Work-Items einschließlich derer, die ein Abbruch-Signal bekommen, sich daraufhin aber noch nicht beendet haben.
4. `EnumWorkItemStateCategory.Final` Alle Work-Items, die abgeschlossen sind, einschließlich abgebrochener, fehlerhafter etc.

Beispiel:

```
Api.Worker.GetWorkItems(UserHelper.CurrentUserGuid(), EnumWorkItemStateCategory.Waiting, EnumWorkItemStateCategory.Ready, EnumWorkItemStateCategory.Active)
```

Das Ergebnis der Funktion ist ein Snapshot der passenden Work-Items. Die Einträge in der Liste sind read-only, aber transient. Bei allen nicht-finalen Work-Items können sich die Werte also jederzeit ändern. Mit technischen Exceptions (Data Race) muss man beim Lesen der Eigenschaften aber nicht rechnen.

```
WorkItemProgress GetWorkItem(Guid instanceGuid)
```

Mit dieser Methode kann ein einzelnes Work-Item abgefragt werden. Da die `InstanceGuid` nur für aktuelle Work-Items eindeutig ist, können damit keine abgeschlossenen Work-Items abgefragt werden, sondern nur laufende oder zukünftige. Bei abgeschlossenen (ohne Nachfolger) kommt `null`.

11.3. Implementierung eigener Work-Items

Um ein neues Work-Item zu erstellen, erstellt man zunächst eine neue Klasse, die von `WorkItemBase` erbt. Im einfachsten Fall überschreibt man nur die Run-Methode und führt darin die Arbeit aus.

```
public class MyWorkItem : WorkItemBase
{
    protected override void Run()
    {
        // Do some work
    }
}
```

Wenn ein Work-Item keinen Standardkonstruktor haben soll (kann sinnvoll sein), dann muss für die JSON-Deserialisierung ein privater Standard-Konstruktor definiert werden.

```
public class MassOperationRelationCleanupTask : WorkItemBase
{
    /// <summary>Default constructor for deserialization only</summary>
    [JsonConstructor]
    private MassOperationRelationCleanupTask() { }
    /// <summary>Own contructor</summary>
    public MassOperationRelationCleanupTask(Guid holderRecordOid, Guid? ...
```

Am Wahrscheinlichsten muss man neben Run noch die Methode `WorkItemFinished` überschreiben, siehe Dokumentation unten.

11.3.1. Allgemeine, öffentliche Eigenschaften

Allgemein gilt: alle Eigenschaften von `WorkItemBase` sind von außen (`public`) read-only (`protected set`). Nur die Workerklasse selbst darf sie zuweisen.
Die Eigenschaften in diesem Kapitel sollten im allgemeinen vor der Einplanung eines Work-Items im Konstruktor oder spätestens in `WorkItemCreating` zugewiesen werden – natürlich nur die erforderlichen.

Guid InstanceGuid

Logischer Primärschlüssel der Arbeitsaufgabe. Dieser muss systemweit eindeutig sein. Der Schlüssel wird in allen Workmanager-API Methoden zum Zugriff auf existierende Work-Items verwendet.

Wird eine noch nicht laufende Aufgabe mit demselben Schlüssel erneut angelegt, so ersetzt diese die vorherige Version.
Eine bereits laufende Aufgabe kann nicht mehr überschrieben werden (Fehlercode). Die Aufgabe kann sich nur selbst erneut einplanen.

Systemweite Aufgaben, die nur einmal existieren dürfen, sollten im Konstruktor eine beliebige, konstante GUID zuweisen.

Wenn immer eine neue Aufgabe erzeugt werden soll, kann die Zuweisung dieses Feldes entfallen. Dann vergibt das System eine GUID.

Wenn die Existenz eines Work-Items an ein anderes Objekt gebunden werden soll, dann kann auch dessen Guid verwendet werden, beispielsweise bei einem an eine Konfiguration gebundenen Worker. Es muss allerdings darauf geachtet werden, dass nicht zwei verschiedene und gleichzeitig erwünschte Worker-Klassen dieselbe GUID verwenden.

Dieses Property darf pro Instanz nur einmal zugewiesen werden.

String Name

Name des Workers. Wird in Administrativen Ansichten von Work-items verwendet sowie als Standard-Titel beim Anlegen des zugehörigen Worker-Protokolls.
Wenn kein Name angegeben wird, wird der Name der implementierenden Worker-Klasse verwendet.

Guid UserContextGuid

Der User-Kontext in dem dieses Work-Item arbeiten soll, z. B. `Constants.SystemUser`.

Standardmäßig wird der aktuelle Benutzer verwendet, unter dem das Work-Item erzeugt wurde.

EnumExpectedRunTime ExpectedRunTime

Priorität des Work-Items (siehe Scheduling/Prioritäten), standardmäßig Normal.

DateTime ScheduledStartTime

Wann soll das Work-Item starten (UTC!), standardmäßig: sofort.

bool IsSystemWorkItem

Ist dies ein System- oder ein User-WorkItem? System-Work-Items tauchen nicht in der Progress Bar auf, auch dann nicht, wenn sie unter der dem angemeldeten User laufen.

bool IsVisibleForAllUsers

Soll dieses Work-Item allen Benutzern im Progress Bar angezeigt werden?

bool IsCancellable

Darf dieses Work-Item vom Benutzer abgebrochen werden?

Die Eigenschaft kann auch davon abhängen, in welchem Zustand sich das WorkItem befindet. Es kann beispielsweise beim Beginn der eigentlichen Arbeit wieder auf `false` gesetzt werden.

int NumberOfStarts, MaxNumberOfStarts

Wie oft wurde diese Instanz des Work-Items bereits gestartet und wie oft darf das maximal passieren.

Diese Parameter werden verwendet, um zu entscheiden, ob ein automatischer Neustart nach einem Fehler oder einem Timeout noch angemessen ist. Standardmäßig werden maximal 3 Startversuche durchgeführt.

WorkerLoggingMode LoggingMode

Soll automatisch ein Worker-Protokoll mit dem Namen der Work-Items für dieses Work-Item angelegt werden?

- `Auto` Beim Schreiben des ersten Ereignisses wird ein Applikationsprotokoll erstellt. (Standard)
- `Always` Vor dem Start des Work-Items eine Applikationsprotokoll angelegt.
- `Never` Es wird kein Applikationsprotokoll erstellt. Die Worker-Implementierung kann dies aber per `InitLogger()` explizit tun.

Dieses Property darf nur im Konstruktor zugewiesen werden.

EnumLogProcesses LoggingProcess

Das Log soll mit diesem Wert für die Eigenschaft `OrmLogBase.Process` erstellt werden.

Dieses Property sollte nur im Konstruktor zugewiesen werden.
Wenn eine dynamische Berechnung gefordert ist, kann diese Alternativ in einer Überladung von `InitLogger()` erfolgen, und zwar vor dem Aufruf von `base.InitLogger()`.

Guid LoggerGuid

GUID des zugehörigen Worker-Protokolls.

Wird hier eine GUID eines existierenden Protokolls verwendet, so wird kein neues angelegt, sondern dieses fortgeschrieben.

Ist das Feld leer und `LoggingMode` aktiviert, wird ein neues Protokoll angelegt.

Dieses Property wird beim Erstellen eines Protokolls automatisch aktualisiert.
Wenn man eine abweichende GUID zuweist, wird das Logging beendet bis wieder `InitLogger()` aufgerufen wird.

EnumWorkItemState State (read-only)

Aktueller Zustand des Work-Items. Siehe Work-Items / Mögliche Zustände.

11.3.2. Eigenschaften für Fortschrittsanzeige

string Title, Caption

HTML-kodierter Titel und Untertitel für die Fortschrittsanzeige. Diese dürfen auch Links o.ä. enthalten.

int CurrentProgress, MaxProgress

Aktueller Fortschritt des Workers, wenn verfügbar.

Der relative Fortschritt ergibt sich aus `CurrentProgress` / `MaxProgress`. Wenn dieser Quotient nicht endlich positiv ist, wird der Fortschritt als „unbekannt“ betrachtet. Das kann man z.B. erreichen, indem man `CurrentProgress` auf -1 setzt.

Standardmäßig sitzen beide Werte auf 0, was „unbekannt“ entspricht.

In der Regel empfiehlt es sich `MaxProgress` einmalig beim ersten Start eines Work-Items auf den gewünschten Arbeitsvorrat zu initialisieren. Von einer früheren, synchronen Initialisierung sollte im Besonderen dann abgesehen werden, wenn diese potentiell zeitaufwändig ist.

11.3.3. Geschützte Eigenschaften

WorkItemLogger Logger

Falls das Logging aktiviert ist, ist das die zu verwendende Instanz des Loggers für das Anwendungslog.

Man sollte damit rechnen, dass dieses Feld null sein kann. Daher sollten Zugriffe darauf entsprechend geschützt werden:

```
Logger?.AddEntry("C10E4BE4-582B-4894-B5AC-4C1413361F42");
```

Session UnitOfWork (read-only)

XPO Transaktion, die für alle Speichervorgänge des Work-Items sowie für das Hinzufügen von Nachfolgern genutzt wird.

Diese `UnitOfWork` darf auch dafür verwendet werden, um die eigentliche Arbeit des Workers zu erledigen. Das ist die einzige Methode, um den Status des Workers, z.B. `CurrentProgress` oder auch interne Eigenschaften, die ihm signalisieren, was schon erledigt ist, zusammen mit der Arbeit transaktional sicher zu speichern. Statt `CommitChanges` muss aber `this.Save()` aufgerufen werden, was ersteres impliziert.

ExceptionDuringRun

Die Methode Run ist mit dieser unbehandelten Ausnahme abgebrochen.

Das ist nur für die Implementierung von `WorkItemFinished` relevant.

11.3.4. Geschützte Methoden

void Save()

Das speichert den Zustand des Work-Items zusammen mit allen ggf. per `AddSuccessor` hinzugefügten Workern und allen anderen Änderungen in der `UnitOfWork`. Ebenfalls werden damit die Fortschrittsanzeigen in den Clients aktualisiert.

void UpdateClient()

Die Methode aktualisiert nur eventuelle Fortschrittsanzeigen in den Clients. Der Zustand des Work-Items wird dabei nicht persistiert. Normalerweise sollte Save bevorzugt werden, dass auch den Zustand des Work-Items persistiert. Nur wenn sich außer dem Fortschritt nichts geändert hat und sich der Erledigungsgrad des Workers ohnehin in anderen Datenbankobjekten manifestiert, ist diese Methode zu empfehlen.

void AddSuccessor(…)

Die Methode fügt einen Nachfolger für den aktuellen Worker hinzu. Das bedeutet, es wird ein Klon des aktuellen Workers angefertigt und anschließend zur `UnitOfWork` hinzugefügt. Beim nächsten Aufruf von Save wird dieser Worker dann eingeplant. Die Vorgehensweise ist essentiell, da der Nachfolger normalerweise dieselbe `InstanceGuid` hat, und diese erst dann eingeplant werden darf, wenn der aktuelle Worker sich beendet hat.

Der Klon wird mit `CreateSuccessor` angefertigt. Siehe dort bezüglich seiner Eigenschaften.

Es gibt mehrere Überladungen der Methode. Diese Ändern den geklonten Worker, bevor er eingeplant wird. Die allgemeinste Überladung erlaubt beliebige Änderungen an dem Worker.

Beispiel:

```
AddSuccessor((FollowUpReminderEmailTask next) =>
{
    next.LastRunTime = ScheduledStartTime;
    next.ScheduledStartTime = NextRun().ToUniversalTime();
});
```

```
void HasSuccessor()
```

Prüft, ob es in der aktuellen `UnitOfWork` bereits einen Nachfolger (mit der gleichen `InstanceGuid`) gibt.

Das kann vor allem dazu verwendet werden, um in der Überschreibung von `WorkItemFinished` zu prüfen, ob das Work-Item schon neu automatisch neu gestartet wurde (z.B. wegen eines automatischen

Neustarts).

void RemoveSuccessor()

Prüft, ob es in der aktuellen `UnitOfWork` bereits einen Nachfolger (mit der gleichen `InstanceGuid`) gibt und entfernt diesen gegebenenfalls. Dies funktioniert nur, wenn die `UnitOfWork` noch nicht gespeichert wurde. Danach kann die Ausführung nur noch mit `StopExecutionOfWorkItem` aufgehalten werden.

void LogError(…), LogWarning(…), LogInfo(…), LogDebug(…)

Meldungen über Ereignisse während der Verarbeitung ins Logfile und ins Anwendungsprotokoll schreiben sofern konfiguriert. Die Methoden berücksichtigen dabei die ggf. unterschiedlichen Sprachen für die beiden Log-Ziele.

Wenn nur ins Anwendungslog geschrieben werden soll, sollte `Logger?.Add…` verwendet werden.

void LogCriticalError(Exception ex, string message = null)

Damit wird eine kritische, also nicht sinnvoll behandelbare Ausnahme protokolliert. Die Protokollierung erfolgt an 3 Stellen:

1. Im Logfile auf dem Webserver.
   Dort wird immer Name und Oid des Workers, sowie die vollständige Exception incl. Callstack geschrieben.
2. Im Anwendungsprotokoll des Workers, falls verfügbar.
   Dort wird Standardmäßig nur eine Fehlermeldung mit der Exception-Message ohne Callstack geschrieben.
3. An die UI. Wenn es sich um ein Work-Item eines Benutzers handelt, wird dieser per Server-Toast auf den Fehler hingewiesen. Auch hier ist die Fehlermeldung ohne Callstack.

Optional kann ein abweichender Meldungstext angegeben werden. Dieser kann per Platzhalter auf Name, Oid, und Exception-Message zugreifen, Siehe XmlDoc.

11.3.5. Überschreibbare Methoden

void WorkItemCreating()

Bevor ein (noch nicht laufendes) Work-Item vom Work-Manager eingeplant oder aktualisiert wird, wird die Methode aufgerufen. Darin kann das Work-Item eigene Eigenschaften berechnen, die mehr Informationen brauchen als dem Konstruktor zur Verfügung stehen.

Die Standardimplementierung ist leer.

void WorkItemShouldFinish()

Diese Methode wird aufgerufen, wenn ein laufendes Work-Item abgebrochen werden soll. Das gibt dem Work-Item die Möglichkeit, asynchron auf ein solches Event zu reagieren, und beispielsweise ein `ManualResetEvent` zu setzen.

> Die Implementierung dieser Methode muss thread-safe sein.

Die Standardimplementierung schreibt eine Warnung über dieses Ereignis ins Applikationsprotokoll, falls dieses aktiv ist.

void WorkItemFinished()

Diese Methode wird für jedes Work-Item genau einmal ausgeführt, das der Work-Manager einmal zu Gesicht bekommen hat. Was dann tatsächlich passiert, ist der Eigenschaft “State” zu entnehmen:

- `Finished` Die Methode Run ist ordnungsgemäß zurückgekehrt oder das Work-Item wurde zwar abgebrochen, aber es war sowieso gerade fertig (`CurrentProgress == MaxProgress`)
- `Removed` Das Work-Item war schon einmal eingeplant und wurde, bevor es starten konnte abgebrochen.
- `Cancelled` Das Work-Item hat auf eine Abbruchanforderung durch den Benutzer reagiert und sich selbst beendet.
- `Timeout` Das Work-Item hat die maximale Verarbeitungsdauer überschritten und sich daraufhin beendet.
  Die Standardimplementierung würde hier eine automatische Neueinplanung versuchen, solange `MaxNumberOfRestarts` noch nicht erreicht ist.
- `Killed` Das Work-Item hat die maximale Verarbeitungsdauer überschritten und sich daraufhin binnen der gesetzten Nachfrist nicht beendet.
- `Error` Die Run-Methode ist mit einer unbehandelten Ausnahmen abgebrochen. Diese steht in `ExceptionDuringRun`.
  Die Standardimplementierung protokolliert die Ausnahme mit `LogCriticalError` und versucht einen Neueinplanung, solange `MaxNumberOfRestarts` noch nicht erreicht ist.
- `Aborted` Die Ausführung wurde durch einen Anwendungsneustart unerwartet unterbrochen. Dieser

Aufruf erfolgt natürlich erst nach dem Neustart der Anwendung.
Die Standardimplementierung versucht hier immer einen Neustart, unabhängig von `MaxNumberOfRestarts`.

Wenn sich das Work-Item nach der Ausführung der Methode neu eingeplant hat, wechselt der Status von `Finished` auf `Reschedule`, von `Timeout` auf `TimeoutRetry` und von `Error` auf `ErrorRetry`.

Überschriebene Implementierungen sollten im allgemeinen `base.WorkItemFinished()` in den Fällen aufrufen, die sie nicht selbst explizit behandelt haben. Nur dadurch ist sichergestellt, dass Worker z.B. nach einem Anwendungsneustart auch wirklich wieder starten.

Ferner sollte die Methode nicht selbst Save aufrufen. Das würde die transaktionale Integrität brechen. Das Work-Item wird danach immer automatisch gespeichert.

Ebenfalls sollten keine aufwändige Berechnungen in der Methode erfolgen. Für die Ausführung steht nur ein enges Zeitkontingent von einer Minute zur Verfügung. Danach wird der Worker unwiderruflich abgebrochen. Auch ein wiederholt eingeplanter Worker würde nach einem solchen Abbruch nie wieder laufen.

Beispiel, um nach einem (bestätigten) Timeout beliebig oft und ohne Karenzzeit dazwischen neu zu starten – das entspricht logisch `Thread.Yield()`:

```
protected override void WorkItemFinished()
{
    if (State == EnumWorkItemState.Timeout)
        AddSuccessor(TimeSpan.Zero);
    else
        base.WorkItemFinished();
}
```

InitLogger()

Diese Methode wird vor der eigentlichen Ausführung eines Work-Items aufgerufen, wenn `IsLoggingEnabled` aktiviert ist. Alternativ kann die Methode auch selbst aufgerufen werden, wenn eine verzögerte bzw. bedingte Anlage des Protokolls gewünscht ist. Dann muss `LoggerGuid` leer sein.

Die Standardimplementierung legt ein neues Applikationsprotokoll an, wenn `LoggerGuid` leer ist oder öffnet ein bestehendes Protokoll wenn einer `LoggerGuid` angegeben wurde. Nachher ist `LoggerGuid` immer gefüllt. Dadurch ist sichergestellt, dass nach Neustarts im selben Protokoll fortgefahren wird.

Die Methode kann überschrieben werden, um abweichende Logging-Verfahren zu implementieren. beispielsweise täglich rollierende Logs.

CreateSuccsessor()

Das ist faktisch eine Klon-Methode. Sie erstellt eine Kopie des Work-Items. Dabei werden alle

persistierten Eigenschaften des Work-Items übernommen, mit folgenden Ausnahmen:

- `State` ist immer `Idle` (Standard für neue Items)
- `ScheduledStartTime` wird auf die aktuelle Uhrzeit gesetzt, also sofort einplanen.
- `NumberOfStarts` wird nicht übernommen. Das muss explizit selbst gemacht werden, wenn erwünscht ist, dass bei dieser Neueinplanung der Countdown für `MaxNumberOfRestarts` läuft.

Eigene Worker können diese Methode überschreiben, um ein abweichendes Verhalten zu implementieren. Das gilt dann für alle per `AddSuccessor` erzeugten Nachfolger einschließlich automatischer Neustarts nach Fehlern.

Beim Überschreiben sollte immer zuerst die Standardimplementierung `base.CreateSuccessor()` aufgerufen werden.

11.3.6. Eigene, persistierte Eigenschaften

Der Work-Manager persistiert automatisch alle Felder und Properties, die vom JSON-Serializer erfasst werden. Man muss also ggf. nur die entsprechenden Attribute setzen, um dem Work-Item erweiterte Eigenschaften zu verpassen, die ebenfalls in der Datenbank gesichert werden. Näheres siehe NewtonSoft-Dokumentation.

Dieses Property wird automatisch serialisiert, weil es `public` ist:

```
public WorkType Type { get; set; }
```

Diese Feld wird explizit mit serialisiert, obwohl es nicht `public` ist:

```
[JsonProperty]
private List<Guid> RecordGuids = new List<Guid>();
```

11.3.7. Wiederkehrende Work-Items

Der Work-Manager bietet Standardmäßig keine solche Funktionalität. Allerdings kann man diese sehr leicht erreichen, indem die Methode `WorkItemFinished` überschieben wird.

```
protected override void WorkItemFinished
{
    base.WorkItemFinished();
    if (!HasSuccessor())
        AddSuccessor(PeriodicCheckInterval);
}
```

Diese Implementierung probiert zuerst die Standard-Restarts (über `base.WorkItemFinished`), nur wenn auf diese Weise nicht schon Neueinplanung erfolgte (`!HasSuccessor()`), plant sich das Work-Item nach dem Zeitintervall `PeriodicCheckInterval` wieder ein.

In diesem Beispiel ist die Periodizität eine konstante Pause. Mit anderen Überladungen von `AddSuccessor` können auch bestimmte Zeitpunkte ausgewählt oder aber das neu einzuplanende Work-Item vorher noch verändert werden.

Wiederkehrenden Worker bei Anwendungsstart initialisieren

Damit ein Wiederkehrender Prozess seine Arbeit aufnimmt, muss er einmalig explizit gestartet werden. Das kann z.B. beim Speichern einer zugehörigen Konfiguration erfolgen. Diese könnte sich aber durch einen Konfigurationsimport ändern.

Es empfiehlt sich, den Status von wiederkehrenden Workern beim Anwendungsstart einmalig zu aktualisieren. Dazu ist das Interface `IInitializationEvent` zu implementieren. Beispiel:

```
public class DeleteOldLogsWorkerTask : WorkItemBase, IInitializationEvent
{
    public void AfterStart()
    {
        Api.Worker.CreateOrUpdate(new DeleteOldLogsWorkerTask());
    }
```

Dem Beispiel liegt die Hypothese zugrunde, dass der Worker im Standard-Konstruktor einen geeigneten Ausführungszeitpunkt wählt.

> Es sollten nicht alle Prozesse direkt beim Anwendungsstart starten.

11.3.8. Nachfolge-Worker starten

Wenn ein Work-Item, nachdem es fertig ist, oder zum Aufräumen, ein weiteres, anderes Work-Item starten will, sollte die Methode `Api.Worker.CreateOrUpdate` mit der `UnitOfWork` des aktuellen Work-Items verwendet werden. Beispiel:

```
Api.Worker.CreateOrUpdate(new MassEMailSendWorkerTask(_ControlOrm, LoggerGuid)
{
    MessageOnSuccess = MessageOnSuccess,
    MessageOnError = MessageOnError
}, UnitOfWork);
```

Dadurch ist sichergestellt, dass das neue Item nur startet, wenn auch die begleitenden Änderungen am aktuellen Item gespeichert wurden. So ist nach einem unerwarteten Anwendungsneustart klar, ob der Nachfolger noch gestartet werden muss oder nicht.

Falls der Nachfolger nicht aus `WorkItemFinished` heraus gestartet wird, sollte sich das Work-Item einer internen, persistierten Eigenschaft merken, dass dies erledigt ist, um doppelte Starts zu vermeiden.

Ein Nachfolge-Worker unterscheidet sich von einer Neueinplanung dadurch, dass er eine andere `InstanceGuid` und im allgemeinen auch eine andere Worker-Klasse verwendet.

Es können auch mehrere Nachfolge-Worker in einem Schritt gestartet werden.

11.3.9. XPO Exceptions selbstfangen

Wenn man in Workern eine Exception selbst behandeln will, muss man aufpassen, danach auch richtig aufzuräumen. In der `UnitOfWork` des Work-Items könnten z.B. noch ungespeicherte Datenreste liegen, die beim nächsten Save dann doch den Weg ins Ziel finden oder aber eine erneute Exception auslösen.

Heißt: diese müssen im Allgemeinen vernichtet werden (Rollback). Zusätzlich muss auch der Cache der UOW gelöscht werden, damit ggf. modifizierte XPO-Objekte neu aus der DB geladen werden. Das beides erledigt `UnitOfWork.DropIdentityMap()`. Darüber hinaus muss man aber auch aufpassen, nicht etwa selbst noch Referenzen auf möglicherweise veraltete XPO-Objekte zu haben. Diese können notfalls auch per `Reload` aktualisiert werden.

```
try
{
    // was immer schief gehen kann
    Save(); // Arbeitsschritt komplett
} catch (Exception ex) when (!ex.IsThreadAbort())
{
    // Fehler protokollieren oder was auch immer
    // UOW aufräumen!
    UnitOfWork.DropIdentityMap();
}
```

Das Aufräumen der UOW verhindert allerdings auch das Speichern eventueller ungespeicherter Änderungen am Work-Item selbst. Wenn da ein Status erhalten bleiben soll, beispielsweise, dass der Arbeitsschritt schon erledigt ist, wenngleich erfolglos, dann muss das nach `DropIdentityMap` in das Workitem geschrieben und ggf. sofort gespeichert werden.

11.4. Massenverarbeitung von Datensätzen

Für die Massenverarbeitung von ORM-Objekten wurde `MassWorkItem` von `WorkItemBase` abgeleitet und kann als Basis eines eigene Work-Items benutzt werden.

11.4.1. Ablauf

Mass-Work-Items basieren darauf, dass

- eine zu implementierende Query den Arbeitsvorrat bereit stellt,
- der Arbeitsvorrat nach Oid sortiert bearbeitet wird,
- der Arbeitsvorrat in Paketen aus der Datenbank geladen wird und
- und jedes ORM einzeln in einer zu implementierenden Funktion bearbeitet wird.

Darüber hinaus werden folgende Sonderfälle behandelt:

- Wenn die Anwendung unerwartet neu gestartet wird, wird die Arbeit exakt bei dem ORM fortgesetzt, wo sie unterbrochen wurde.
- Wenn der Worker die maximale Laufzeit überschreitet, unterbricht er seine Arbeit und plant sich für den Rest neu ein. Dies erfolgt maximal `MaxNumberOfRestarts` mal.

Die Aktualisierung der Daten für die Fortschrittsanzeige (`MaxProgress`, `CurrentProgress`) obliegt der Worker-Implementierung.

Nach jedem ORM wird der Zustand des Workers automatisch gesichert und die UI aktualisiert.

11.4.2. Abstrakte Methoden

GetQueryable

```
protected abstract IQueryable<OrmBABase> GetQueryable();
```

Die wird benutzt, um den Arbeitsvorrat zu ermitteln, der von `ProcessSingleOrm` bearbeitet werden soll. Der zurückgelieferten Query werden weitere Where-Bedingungen hinzugefügt, um die Paketierung und das Wiederaufsetzen zu steuern.

Die Implementierung sollte für die Query die `UnitOfWork` des Work-Items benutzen, wenn die später gemachten Änderungen mit dem Status des Workers in einer Transaktion gespeichert werden sollen. Dies ist der einfachste Weg, um sicher zu stellen, dass Änderungen niemals doppelt ausgeführt werden, falls der Worker mal unterbrochen wird. Andernfalls muss man sich selbst um die Vermeidung von Doppelausführungen kümmern.

ProcessSingleOrm

```
protected abstract void ProcessSingleOrm(OrmBABase ormBABase);
```

Hier erfolgt die eigentliche Arbeit. Der Worker kann mit dem Datensatz letztlich machen, was er will, und auch andere Datensätze verändern.

Falls `GetQueryable` die `UnitOfWork` des Work-Items verwendet hat, ist kein explizites Speichern erforderlich. Dies erfolgt nach jedem Aufruf der Methode automatisch.

11.4.3. Optionale Implementierungen

Die Standardimplementierung dieser Methoden leer. Sie können bei Bedarf überschrieben werden.

BeforeProcessing

```
protected virtual void BeforeProcessing(bool resume) { }
```

Diese Methode kann überschrieben werden, um Arbeiten vor der eigentlichen Hauptaufgabe zu erledigen. Diese Methode wird exakt einmal bei ersten Start des Workers mit `false` aufgerufen. Falls der Worker, warum auch immer, neu gestartet wird, erfolgt der Aufruf mit `true`.

Typischerweise wird man hier bei `!resume MaxProgress` füllen wollen, sofern die nicht schon bei der Erzeugung des Work-Items (ohne Datenbankabfrage) gefüllt werden konnte.

AfterProcessing

```
protected virtual void AfterProcessing(Exception exception) { }
```

Diese Methode wird nach der eigentlichen Arbeit exakt einmal aufgerufen. Das gilt sowohl für den Fall, dass die Arbeit erfolgreich beendet wurde, als auch für den Fall, dass es zu einem Fehler (unbehandelte Ausnahme) kam. Den Unterschied erkennt man am `State` und ggf. am Inhalt von `ExeptionDuringRun`.

Hier sollten keine wesentlichen Aufgaben mehr erledigt werden, da nur ein geringes Zeitkontingent von einer Minute zur Verfügung steht.

BeforeSuspend

```
protected virtual void BeforeSuspend() { }
```

Wenn die Arbeit unterbrochen wird, z.B. weil eine Zeitüberschreitung erfolgt ist, dann wird die Methode `BeforeSuspend` aufgerufen.

Auch hier sollten keine wesentlichen Aufgaben erledigt werden, da nur ein geringes Zeitkontingent von einer Minute zur Verfügung steht.

OnUserCancelled

```
protected virtual void OnUserCancelled() { }
```

Wenn ein Work-Item abgebrochen wird, wird der aktuelle Datensatz noch fertig bearbeitet und dann

diese Methode aufgerufen. Der Aufruf erfolgt auch dann, wenn der Abbruch vor dem Start erfolgt und `BeforeProcessing` noch nie aufgerufen wurde. Den Unterschied erkennt man am `State`, der in letzterem Fall `Removed` ist, sonst ist er `Cancelled`.

Auch hier sollten keine wesentlichen Aufgaben erledigt werden, da nur ein geringes Zeitkontingent von einer Minute zur Verfügung steht. Falls größere Aufräumarbeiten erforderlich sind, sollten diese in einem eigenen Aufräum-Worker als Nachfolger gestartet werden.

11.4.4. Aktionen auf alle selektieren Datensätze

Um eine Ribbon bar Aktion auf alle selektierten Datensätze mit Hilfe eines Hintegrundprozesses zu implementieren, kann man auf die Implementierung eines eigenen Igniters verzichten. Die Aktion implementiert stattdessen das Interface `IIgnitableAction` und der Worker `IIgnitableWorkItem`

```
public class ClientActionContactsChangeAssignedTo : ClientActionGridMassOperat
ionBase, IIgnitableAction
```

In dem Fall kann der Igniter `OperationFromActionOverSelectedRecordsIgniter` verwendet werden.

```
MassOperationIgniter = typeof(OperationFromActionOverSelectedRecordsIgniter).A
ssemblyQualifiedName;
```

Interface Implementierung

Es müssen eine Reihe von Eigenschaften implementiert werden.

```
string MessageOnSuccess { get; set; }
string MessageOnError { get; set; }
string MessageOnStart { get; set; }
string ProgressBarProcessName { get; set; }
string ProgressBarProcessDescription { get; set; }
string Caption { get; }
```

Eine Methode, die den Typ des entsprechenden Workers liefert.

```
public Type GetWorkItemType()
{
    return typeof(MyWorkerTask);
}
```

In der Methode `Ignite` kann man beispielsweise dem instanziierten Worker weitere Parameter mitgeben.

```
public void Ignite(IIgnitableWorkItem task, Guid tempKey, Guid? taskExecutionI
d, Dictionary<string, object> parameters, JsonFormResult result)
{
    MyWorkerTask myTask = (MyWorkerTask)task;
    myTask.MyParameter = myParameter;
```

```
}
```

11.5. Anwendungsprotokolle

Typischerweise werden Anwendungsprotokolle von den Hintergrundprozessen geschrieben, prinzipiell können sie aber an beliebige Stellen genutzt werden. Die Anwendungsprotokolle haben eine gemeinsame Basis-Datentabelle (Basis.Anwendungsprotokoll / `OrmLogBase`). Daher kann man eigene Datentabellen davon ableiten und bestehende erweitern. Eine konkrete Datentabelle Anwendungsprotokoll / `OrmLogDefault` ist ebenfalls vorhanden. Diese kann genutzt werden, wenn keine speziellen Eigenschaften benötigt werden.

Anwendungsprotokolle sollten nicht mit einer `UnitOfWork` erstellt werden, stattdessen sollte es eine eigene `Session` haben, so dass fehlgeschlagene Transaktionen das Protokollieren nicht verhindert.

Erstellen / Speichern

Ein Protokoll wird identisch wie die anderen Datensätze erstellt. Der Inhalt wird automatisch gespeichert, wenn Einträge hinzugefügt werden. Direkt nach dem Erstellen könnte man es speichern, um sicherzustellen, dass es vorhanden ist. Ansonsten sollte man den Hauptdatensatz des Protokolls nicht mehr speichern. Das Hinzufügen von Einträgen ist thread-safe und ohne LockingException. Daher keine Aufrufe von `Save()` während der Protokollierung!

Worker

Innerhalb eines Workers wird das Protokoll vom System erstellt und verwaltet, man sollte lediglich den Prozess auf den Auswahlwert `EnumLogProcesses` seines Prozesse setzen.

```
LoggingProcess =  EnumLogProcessesExtension.MyLog;
```

Wird eine eigene Log-Datentabelle gewünscht. Setzt man dazu im Konstruktor seines Workers die Guid der Log-Datentabelle, bzw. überlädt auch `InitLogger()`.

```
public MyWorker() : base()
{
    ...
   LoggingMode = WorkerLoggingMode.Always;
   LoggingProcess =  EnumLogProcessesExtension.MyLog;
   LoggerGuid = EnumDataSource.MyLogGuid;
    ...
}
```

```
protected override void InitLogger()
{
    base.InitLogger();
    // set properties
    Logger.Title = "My Title";
```

```
    SetEnumValue(nameof(OrmLogBase.Process), LoggingProcess);
}
```

Protokollsprache

In den Anwendungseinstellungen wird die “Inhaltssprache für Protokollierung” festgelegt. Daher muss man darauf achten nicht die üblichen Übersetzungsmethoden `String.Translate()` und `Api.Text` zu verwenden.

Die eigentlichen Api Methoden übersetzen selbst. Zusätzlich ist auf `OrmLogBase` eine entsprechende `Translate()` Methode vorhanden.

Protokolleinträge

Als Standard Api stehen folgende Methoden zur Verfügung.

```
public void AddInfo(string text, params object[] para)
public void AddWarning(string text, params object[] para)
public void AddError(string text, params object[] para)
public void AddFatal(string text, params object[] para)
public void AddDebug(string text, params object[] para)
```

Mehrzeilige Einträge dürfen nicht mit mehrfachen Aufruf der `Add` Methoden erzeugt werden, da die Auswertung dadurch erheblich erschwert wird. Stattdessen wird es mit `\n` Mehrzeilig gestaltet.

Beispiel:

```
Logger.AddInfo("Text mit {0} {1}", 2, "Platzhalter");
Logger.AddInfo("[INSERT TRANSLATION GUID]", 2, "Platzhalter");
Logger.AddInfo("Zeile 1\nZeile 2\Zeile 3");
```

LogExcpetion

Die Ausnahme `LogExcpetion` kann genutzt werden, um Informationen für Protokolleinträge zu werfen.

```
throw new LogException(EnumLogLevel.Warning, "[INSERT TRANSLATION GUID]");
```

Die entsprechend abgefangen werden muss.

```
catch (LogException logException)
{
    logger.AddEntry(logException);
```

```
}
```

Einrückungen im Protokoll

Einrückungen im Protokoll dienen für die Strukturierung und damit zur besseren Lesbarkeit. Beim Aufruf von `CreateSubEvents()` erhält man als Ergebnis einen Helper, der den Kontext bis zu seinem Dispose-Aufruf aufrecht erhält. Typischer Anwendungsfall:

```
Logger.AddInfo("Starting work XXX");
using (Logger.CreateSubEvents())
    foreach (wasauchimmer)
    {
        Logger.AddInfo("Jetzt bei YYY");
    }
```

Solche Kontexte können geschachtelt werden.

Es ist allerdings nicht zulässig, die Einrückung zu erhöhen, ohne vorher ein Event erzeugt zu haben. Alle neuen Einträge innerhalb des Sub-Entries-Kontextes werden zum letzten Event davor zugeordnet.
Im Besonderen sollte das nicht verwendet werden, um ein zusammengehöriges, mehrzeiliges Event zu ersetzen.

Falls die Protokollierung in dasselbe Log über verschiedene Log-Instanzen erfolgt, hat jede ihren eigenen Einrückungskontext. Dieser wird nicht gespeichert. Immer wenn ein Log neu geöffnet wird, um Einträge hinzuzufügen, beginnt man wieder im Top-Level.

11.6. Übung 7

Konfiguration

Erstellen Sie eine Maske und eine Ansicht für die Anwendungsprotokolle (siehe technisches Handbuch).

Erstellen Sie (falls noch nicht vorhanden) eigene Anwendungsaktionen und fügen Sie alle notwendigen Aktionen ein. Sowie eine Aktion zur Anzeige der Ansicht für die Anwendungsprotokolle.

Hintergrundprozess

Erstellen Sie einen eigenen Hintergrundprozess im Ordner “Worker” für die Verarbeitung. Erstellen Sie im ersten Schritt einen eigenen Prozess, ohne die vereinfachten Möglichkeiten für die Massenverarbeitung.

Übertragen Sie die Funktionalität aus dem Igniter in den Worker. Dabei übergeben Sie dem Prozess die `TaskExecutionId` von `OperationOverSelectedRecordsIgniterBase` und laden Sie damit die selektierten Datensätze im Worker.

Stellen Sie sicher, das kein Datensatz doppelt verarbeitet wird, wenn beispielsweise der Server abstürzt.

Ausnahmebehandlung

Provozieren Sie eine Ausnahme beim dritten Datensatz beim ersten Start und kontrollieren sie das NLog und die Anwendungsprotokolle.

Warum wurde ein Anwendungsprotokoll geschrieben? Und wie kann man es verhindern?

Wurden alle Datensätze korrekt verarbeitet?

Massenverarbeitung

Duplizieren Sie Ihre Aktion, dabei beachten Sie das sie der neuen Aktion eine eigene `Id` geben und die Namen verändern. Stellen Sie die neue Aktion auf auf die Massenverarbeitung um.

Verzögern Sie den Start und erlauben Sie das der Anwender den Prozess abbrechen kann.

Anwendungsprotokoll

Erweitern Sie den zweiten Worker um ein Anwendungsprotokoll.

Ausnahmebehandlung

Provozieren Sie eine Ausnahme in `GetQueryable()` beim ersten Start.

Provozieren Sie eine Ausnahme in `ProcessSingleOrm()` beim zweiten Start und dem dritten

Datensatz.

Was steht im Anwendungsprotokoll? Wurden alle Datensätze verarbeitet?

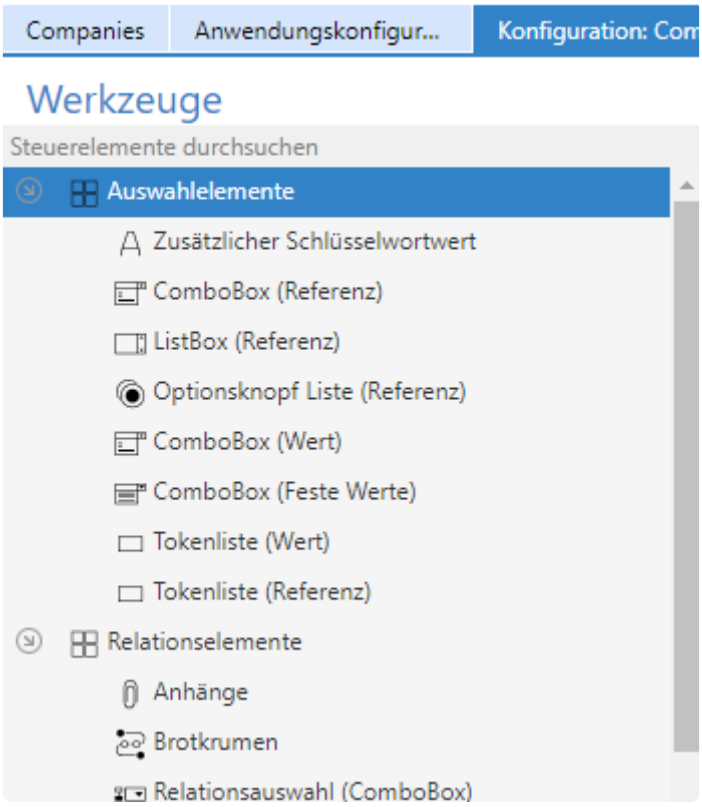
Warum gibt es einen Unterschied zum vorherigen Hintergrundprozess und was müsste man tun, um Fehlerhafte Datensätze nochmal zu verarbeiten?

Lösung

12. Masken

12.1. Maskensteuerelemente

Als Steuerelement werden generell die Komponenten der Business App Anwendungskonfiguration bezeichnet, welche im Designer aus der linken Spalte in die Mitte gezogen und konfiguriert werden können. Maskensteuerelemente sind die Elemente, die beim Erstellen einer Maske zur Auswahl stehen, zum Beispiel eine Text- oder eine Combobox.
Maskensteuerelemente dienen im Allgemeinen dazu, Daten eines Datensatzes anzuzeigen und/oder zu ändern. In einigen Fällen zeigen diese Steuerelemente vordefinierte oder berechnete Werte und Grafiken an.



Ein Maskensteuerelement besteht üblicherweise aus mindestens zwei C# Klassen.

In dem Beispiel, soll ein neues Maskensteuerelement erstellt werden, welches den Wert aus zwei Eingabefeldern mit einem Pipe-Symbol getrennt in ein einzelnes Textfeld der Datentabelle schreibt und natürlich auch von dort wieder zurückholt, aufteilt und anzeigt.

12.1.1. Klasse des Steuerelements

Eine Klasse dient der visuellen Definition des Steuerelements im Designer. Diese Klasse enthält Informationen darüber, welches Symbol vor welchem textuellen Namen des Elements in der linken Spalte angezeigt wird. Des Weiteren wird hier festgelegt, in welcher Gruppe von Elementen – beispielsweise „Auswahlelemente“ oder „Relationselemente“ – das Steuerelement angezeigt werden soll und welche möglichen Einstellungen es in der rechten Spalte des Designers aufweist. Zu guter Letzt gibt es in der C# Klasse des Steuerelements noch die Möglichkeit bestimmte Methoden zu überschreiben, welche die interne Behandlung des Elements in Verbindung mit den generierten Datenmodellen der Maske ändern. Die Klasse des Steuerelements muss mindestens von der Basisklasse `BA.Core.Configuration.ControlBase` abgeleitet sein, oft macht es aber Sinn sie von der Basisklasse `BA.Core.Configuration.Form.Controls.DataControlBase` abzuleiten, da diese Basisklasse bereits eine standardisierte Möglichkeit zur Auswahl einer mit dem Steuerelement verbundenen Spalte aus der der Maske zugrundeliegenden Datentabelle bereitstellt.

Zusätzlich muss das Steuerlement die Schnittstelle `IDevExFormControl` implementieren, damit es von DevExpress in der Form dargestellt werden kann.

In manchen Fällen kann es Sinn machen ein bestehendes Steuerelement als Basis zu verwenden, beispielsweise, wenn man die normale Texteingabe nur leicht abwandeln möchte. In diesem Fall leitet man sein Steuerelement von dem entsprechenden Texteingabesteuerelement ab.

Maskensteuerelemente verwenden ein bestimmtes C# Attribut, welches sie als Steuerelement für Masken kennzeichnet.

```
[Toolbox(EnumConfigurationType.FormConfigurationGuid)]
```

Drag & Drop Regel

Bei jedem Steuerelement, welches im Designer zur Konfiguration genutzt werden soll, muss man darauf achten, dass die Drag & Drop Regeln korrekt gesetzt sind. Näheres dazu findet man in dem speziellen Kapitel.

In der Regel werden für Aktionen auf Basis von `DataControlBase` keine weiteren Regeln benötigt.

Wichtige Klassen zur Definition von Regeln

- `FormConfiguration` Hauptknoten
- `LayoutPanelControl` Layout. Basisknoten für alle Elemente
- `GroupControl` Gruppe für Steuerlemente
- `TabContainerControl` Tab-Container (kann nur Tabs beinhalten)
- `TabControl` Oberster Knoten aller Elemente auf einem Tab
- `PartialRefreshGroupControl` Aktualisierungsgruppe

Beispiel

Die Repräsentation des Maskensteuerelements im Beispiel ist relativ einfach. Neben der Attribut Angabe, was diese Klasse eigentlich zu sagen hat, gibt es nur noch den Konstruktor, über den bestimmte Eigenschaften des Elements gesetzt werden.
Bitte wählen Sie für ein eigenes Steuerelement im Rahmen eines Projektes, einen entsprechend passenden Namespace und achten Sie darauf, dass jedes Steuerelement eine eindeutige ID besitzen muss.

```
namespace BA.Core.Configuration.Form.Controls
{
   [Serializable]
   [Toolbox(EnumConfigurationType.AttributeValues.FormConfiguration)]
   public class ValueAndAliasControl : DataControlBase, IDevExFormControl
   {
       [BARequired]
       [PropertiesGroup("Spezielle Einstellungen")]
       [HelpText("Geben Sie das Trennzeichen zwischen Wert und Alias an.")]
       [StringLength(1)]
       [DisplayName("Trennzeichen")]
       public String Delimiter { get; set; } = "|";

       public ValueAndAliasControl()
       {
           ToolboxName = "[INSERT TRANSLATION GUID]";
           ControlInitName = "ValueAndAlias";
           ToolboxGroupName = "[INSERT TRANSLATION GUID]";
           Icon = Constants.Icons.Designer.Form.TextEdit;
           Id = new Guid("[INSERT UNIQUE GUID]");
           Caption = "[INSERT TRANSLATION GUID]";
           HasErrors = false;
           DesignerHintText = "[INSERT TRANSLATION GUID]";
       }

       protected override Type[] GetTypesOfOrmFieldName(ConfigurationBase conf
iguration)
       {
           return new[] { typeof(OrmTextField) }; ;
       }
   }
}
```

Erläuterung

Die Klasse kann eigene C#-Properties beinhalten, die in der rechten Spalte des Designers als bearbeitbare Eigenschaft angezeigt werden. Die zu diesem Property angewendeten Attribute steuern

bestimmte Eigenschaften des Properties. In diesem Fall beispielsweise muss ein Wert angegeben werden (`[BARequired]`) und der eingegebene Wert darf nur maximal ein Zeichen lang sein (`[StringLength(1)]`). Die Eigenschaft erscheint im Designer in einer eigenen Gruppe namens „Spezielle Einstellungen“, es gibt eine Beschriftung „Trennzeichen“ und eine Feldhilfe (`HelpText`). Diese letzten drei Attribute sind übersetzbar, indem man hier Guids von übersetzten Werten angibt.

Die Klasse des Steuerelements muss als serialisier bar gekennzeichnet sein. Das Toolbox-Attribut bestimmt, dass diese Klasse als Steuerelement für Masken gedacht ist. Die Klasse erbt von `DataControlBase`, das bedeutet, sie beinhaltet automatisch ein Eigenschaftsfeld, in dem Benutzer eine Spalte der zugrundeliegenden Datentabelle auswählen können.

Im Konstruktor dieser kleinen Klasse werden verschiedene Eigenschaften gesetzt.

- `ToolboxName` Name des Steuerelements, wie es im Designer in der linken Spalte angezeigt wird. Dieser Wert ist ein übersetzter Wert, daher wird hier nur die GUID der Übersetzung eingetragen.
- `ControlInitName` Interner Name des Controls, welcher nach Bedarf automatisch mit einer generierten Nummer ergänzt wird.
- `ToolboxGroupName` Name der Gruppe, in welcher das Steuerelement in der linken Spalte des Designers angezeigt wird (übersetzter Wert).
- `Icon` Symbol, welches neben dem `ToolboxName` angezeigt wird, dient der einfacheren Identifizierung des Steuerelements
- `Id` Eindeutige ID des Steuerelements (GUID), muss in jedem Fall neu generiert werden (!)
- `Caption` Die Standardbeschriftung des Steuerelements, Vorbelegung des entsprechenden Feldes bei den Eigenschaften eines in die Maske gezogenen Elements (rechte Spalte, übersetzter Wert).
- `HasErrors` Definiert, ob dieses Steuerelement zunächst im Fehlerzustand sein sollte (optional)
- `DesignerHintText` Beschreibung des Steuerelements, welche im Browser Tooltip erscheint, wenn man mit dem Mauszeiger einige Zeit über dem Element in der linken Spalte des Designers stehen bleibt (übersetzter Wert).

In der überladenen Methode `GetTypesOfOrmFieldName` definiert man die Datentypen der Felder aus der Datentabelle, die zur Auswahl stehen. Mit `OrmTextField` definiert man die Textfelder. Im Packet `BA.Core.Configuration.OrmEntityBase.EntityFields` findet man die möglichen Datentypen.

12.1.2. Klasse des Renderers

„Renderer“ sind C#-Klassen, die sich um die visuelle Repräsentation eines Steuerelements bei der Verwendung in einer Maske kümmern. Sie definieren, ob der Benutzer beispielsweise ein Texteingabefeld oder ein Listenfeld sieht und auch, wie die vom Browser gesendeten Daten aufbereitet werden müssen, damit sie in das der Maske zugehörige Datenmodell gespeichert werden können.

Ein Maskensteuerelement kann bis zu zwei Renderer-Klassen besitzen, die je nach Situation ausgewählt werden. Es besteht die Möglichkeit, eine Renderer-Klasse für die Anzeige des Steuerelementes in der Maske im Browser einzurichten und eine zweite, die verwendet wird, wenn die Maske, die das Steuerelement beinhaltet, ausgedruckt werden soll.

Renderer-Klassen müssen von ihrer jeweiligen Basisklasse ableiten. Für Anzeige-Renderer ist das die Klasse `BA.Core.Configuration.Form.Renderers.ControlDefaultRendererBase` und für Druck-Renderer `BA.Core.Configuration.Form.Renderers.ControlPrintRendererBase`.

Die Zuordnung eines Renderers zu einem bestimmten Steuerelement erfolgt über ein entsprechendes Attribut, welches der Renderer-Klasse zugeordnet wird. In diesem Attribut wird ebenfalls der Verwendungszweck des Renderers angegeben. Der Verwendungszweck ist optional, als Standardwert wird hier „Default“ verwendet, was der Verwendung zur Anzeige in einem Webbrowser entspricht. Beispiel des Druck-Renderers des Dateianhangelements:

```
[FormControlRenderer(typeof(AttachmentControl), EnumFormControlRendererUsage.A
ttributeValues.Print)]
```

Sie können durch die Angabe dieses Attributs in einem Projekt-Modul auch existierende Renderer übersteuern, indem Sie einen existierenden Renderer einfach neu definieren. Die wäre anwendbar, wenn Sie beispielsweise wollen, dass Textfelder immer rote Schrift haben. Sie würden dann einen neuen Renderer für die Klasse `TextEditControl` definieren.

Renderer beinhalten drei Methoden `Edit()`, `Read()` und `Bind()`, von denen `Edit()` und `Read()` zwingend implementiert werden müssen.

Die Edit-Methode sorgt für die korrekte Ansteuerung und Verwendung der visuellen Komponenten (DevExpress) im Bearbeitenmodus und die Read-Methode im Lesemodus. Von den Möglichkeiten unterscheiden sich diese beiden Methoden nicht. Der Entwickler ist dafür Verantwortlich in dem jeweiligen Modus, die entsprechenden Elemente für die UI zu generieren.

Die Bind-Methode sorgt dafür, dass die vom Browser gelieferten Daten korrekt in das Datenmodell der Maske geschrieben werden. Die Standardbehandlung von `Bind()` überträgt den Wert eines angezeigten Steuerelements in ein Feld, falls das angezeigte Feld im Browser den gleichen Namen wie das in C# implementierte Steuerelement hat. Dies funktioniert immer dann, wenn Werte aus dem Feld im Webbrowser exakt übernommen werden können.

Beispiel

Die `Render()` Methode ist das Komplizierteste hieran, daher kümmern wir uns um diese ganz zum Schluss. Werfen wir zunächst einen Blick auf das Gerüst der Klasse, verwenden sie bitte auch hier einen Ihrem Projekt entsprechenden Namespace.

```
 namespace BA.Core.Configuration.Form.Renderers
{
    [FormControlRenderer(typeof(ValueAndAliasControl))]
    public class ValueAndAliasRenderer : ControlDefaultRendererBase
    {
(… Klasseninhalt …)
    }
}
```

Die Klasse selbst erbt vom `ControlDefaultRendererBase` und ist über das Attribut `FormControlRenderer` als Renderer für Steuerelemente des Typs `ValueAndAliasControl` gekennzeichnet.
Der „Klasseninhalt“ besteht folglich aus den Methoden `Edit()`, `Read()` und `Bind()`.

Bind

Schauen wir zunächst auf die `Bind()` Methode:

Die `Bind()` Methode ist dafür zuständig, dass der Wert, der vom `propertyDescriptor` beschrieben wird, in Verbindung mit dem `bindValuePrefix` aus dem `bindValueProvider` geholt und im gewünschten Property des `bindObject` abgelegt wird. Falls eventuell benötigt, wird hier zusätzlich noch eine Liste mit allen auf der Maske verwendeten Steuerelementen `formControls` geliefert.

Die Funktion der `Bind()` Methode ist relativ einfach: Das Steuerelement zeigt in der Benutzeroberfläche zwei Felder, von denen eines `<Feldname>_Value` heißt und ein anderes `<Feldname>_Alias`. Diese beiden Werte müssen aus den vom Browser kommenden Daten ausgelesen, zusammengerechnet und in das `bindObject` geschrieben werden, vorzugsweise unter Verwendung des übergebenen `propertyDescriptors`. Beim Zusammenrechnen wird das Konfigurierte Trennzeichen des Controls verwendet. Das für die Ermittlung des Trennzeichens benötigte Steuerelement wird anhand des aktuellen Feldnamens aus der Liste der `formControls` herausgesucht.

Diese Funktion wird aufgerufen, wenn bei einem Speichervorgang (o. ä.) Daten eingehen, die zu unserem neuen Steuerelement gehören.

```
public override bool Bind(object bindObject, System.Web.Mvc.IValueProvider bin
dValueProvider, string bindValuePrefix, PropertyDescriptor propertyDescripto
r, List<ControlBase> formControls)
{
    // Berechnung der UI-Feldnamen
    String modelFieldName = propertyDescriptor.Name;
```

```
    String bindValueFieldName_Value = bindValuePrefix + modelFieldName + "_Val
ue";
    String bindValueFieldName_Alias = bindValuePrefix + modelFieldName + "_Ali
as";

    // Auslesen der Inhalte aus den Daten vom Client
    String value = bindValueProvider.GetValue(bindValueFieldName_Value)?.Attem
ptedValue ?? "";
    String alias = bindValueProvider.GetValue(bindValueFieldName_Alias)?.Attem
ptedValue ?? "";

    // Ermittlung des dazugehörigen Steuerelements
    ValueAndAliasControl valueAndAliasControl = (ValueAndAliasControl) formCon
trols.First (ff => (ff is ValueAndAliasControl) && ((ValueAndAliasControl)f
f).OrmFieldName == modelFieldName);

    // Setzen des Wertes in den Datensatz
    propertyDescriptor.SetValue(bindObject, value + valueAndAliasControl.Delim
iter + alias);

    return true;
}
```

Edit

Die Feldnamen mit den Suffixen „_Value“ und „_Alias“ werden von der `Edit()` Methode festgelegt. Die Methode verwendet eine private Hilfsmethode (`CreateTextBoxItem`), um das eigentliche DevExpress-Texteingabesteuerelement zu erstellen und zu parametrisieren. Auf diese Methode gehen wir noch im Anschluss ein. Die hier dargestellte oberste Ebene der Methode erstellt zwei Dev Express-Felder, deren Feldnamen jeweils dem Feldnamen des Datenmodells mit unterschiedlichen Suffixen entsprechen. Ein eventuell im Datenmodell verfügbarer Wert wird ausgewertet und der CreateControl-Methode als Vorbelegung übergeben. Nähere Beschreibungen bieten auch die Kommentarzeilen im Beispielquellcode.

Aufgrund der schieren Menge der eventuell zum Rendern notwendigen Daten, hat die `Edit()` Methode ein eigenes Parameterdatenmodell, welches die verschiedensten Informationen beinhaltet.

```
public override void Edit(FormControlRendererParameterModel parameters)
 {
   FormControlRendererUiParameterModel parametersUi = (FormControlRendererUiPa
rameterModel)parameters;
   ValueAndAliasControl valueAndAliasControl = (ValueAndAliasControl)parameter
sUi.Control;
   FormRenderingContextUi renderingContext = (FormRenderingContextUi)parameter
sUi.RenderingContext;
```

```
   // Feldnamen definieren (Datenmodell)
   String modelFieldName = valueAndAliasControl.OrmFieldName;

   // Feldnamen definieren für erstes Feld (Browser UI)
   String uiFieldName = modelFieldName + "_Value";

   // zu verwendendes Trennzeichen
   char delimiter = valueAndAliasControl.Delimiter.ToCharArray()[0];

   // Inhalt des gespeicherten Feldes holen und für das erste Feld bearbeiten
   String fieldContent = (String)parametersUi.BindData.GetPropertyValue(modelF
ieldName);
   fieldContent = fieldContent?.Split(delimiter)?[0] ?? "";

   // Erstes Feld erzeugen und als Objekt hinzufügen
   MVCxFormLayoutItem mvcxItem = CreateTextBoxItem(valueAndAliasControl, uiFie
ldName, renderingContext.FormModel, "Wert", fieldContent);
   parametersUi.MvcxControls.Add(mvcxItem);

   // Inhalt des gespeicherten Feldes holen und für das zweite Feld bearbeiten
   fieldContent = (String)parametersUi.BindData.GetPropertyValue(modelFieldNam
e);
   String[] splt = fieldContent?.Split(delimiter) ?? (new String[] { "", ""
});
   if (splt.Length < 2)
       splt = fieldContent?.Split(delimiter) ?? (new String[] { "", "" });
   fieldContent = fieldContent?.Split(delimiter)?[1] ?? "";

   // Feldnamen definieren für zweites Feld (Browser UI)
   uiFieldName = modelFieldName + "_Alias";

   // Zweites Feld erzeugen und als Objekt hinzufügen
   mvcxItem = CreateTextBoxItem(valueAndAliasControl, uiFieldName, renderingCo
ntext.FormMoel, "Alias", fieldContent);
   parametersUi.MvcxControls.Add(mvcxItem);
}
```

CreateTextBoxItem

Die zuvor angesprochene Hilfsmethode `CreateTextBoxItem()` erstellt unter Zuhilfenahme der übergebenen Werte ein DevExpress-Steuerelement zur Texteingabe. Auch hier wurden erklärende Kommentare in den Quellcode eingefügt, um nähere Erläuterungen zu bestimmten Code-Teilen zu bieten.

```
private MVCxFormLayoutItem CreateTextBoxItem(ValueAndAliasControl valueAndAlia
sControl, String fieldName, DevExFormModel formModel, String label, String pre
```

```
setValue)
{
   MVCxFormLayoutItem mvcxItem = new MVCxFormLayoutItem();

   // Der Typ des neuen Controls => es ist eine "TextBox".
   mvcxItem.NestedExtensionType = FormLayoutNestedExtensionItemType.TextBox;

   // setzen verschiedener Einstellungen auf das MVCxItem, wie zB der Feldnam
e, der im
   // Webbrowser verwendet werden soll, die Beschriftung vor dem Feld, der Hil
fetext, etc
   mvcxItem.Name = fieldName;
   mvcxItem.FieldName = fieldName;
   mvcxItem.HelpText = RenderingUtils.GetHelpText(Api.Text.Format(valueAndAlia
sControl.HelpText));
   mvcxItem.ClientVisible = valueAndAliasControl.Visible;
   mvcxItem.VerticalAlign = FormLayoutVerticalAlign.Top;
   mvcxItem.Caption = label;
   mvcxItem.CaptionSettings.Location = Api.Enum.EnumValueConverter.GetLabelPos
ition(valueAndAliasControl.LabelPosition);
   mvcxItem.CaptionSettings.VerticalAlign = FormLayoutVerticalAlign.Top;

   // Weitere bestimmte Einstellungen sind im sog. "Settings"-Unterobjekt verf
ügbar. Das
   // betrifft zB die Editierbarkeit des Feldes, bestimmte Javascript-Events i
m Browser,
   // die ausgeführt werden sollen, Abmessungen des Feldes, usw.
   TextBoxSettings settings = (TextBoxSettings)mvcxItem.NestedExtensionSetting
s;

   // Horizontale Ausrichtung des Controls
   settings.ControlStyle.HorizontalAlign = HorizontalAlign.Left;

   // ist das Feld eventuell für Bearbeitung gesperrt?
   Dictionary<AttributeEnum, Boolean> attributes = RenderingUtils.GetFormAttri
butesForControl(formModel.AdditionalAttributes, valueAndAliasControl);
   if (attributes.ContainsKey(AttributeEnum.ReadOnly) || valueAndAliasContro
l.ReadOnly)
       settings.ClientEnabled = false;

   // Breite des Feldes
   settings.Width = new Unit(valueAndAliasControl.Width, Api.Enum.EnumValueCon
verter.GetValueWidthType(valueAndAliasControl.WidthType));

   // ist es ein Pflichtfeld?
   if (valueAndAliasControl.Required)
       settings.Properties.ValidationSettings.RequiredField.IsRequired = true;
```

```
   // Feld kann als fehlerhaft markiert werden, falls die Validierung fehlschl
ägt
   if (settings.Properties is EditProperties)
   {
       EditProperties props = (EditProperties)settings.Properties;
       ApplicationSubConfigurationForm appFormSettings = Configurations.Applic
ationConfiguration.GetSubConfiguration<ApplicationSubConfigurationForm>();
       props.ValidationSettings.Display = Api.Enum.EnumValueConverter.GetError
FrameDisplayMode(appFormSettings.ErrorFrameDisplayMode);
       props.ValidationSettings.ErrorTextPosition = Api.Enum.EnumValueConverte
r.GetErrorTextPosition(appFormSettings.ErrorTextPosition);
       props.ValidationSettings.ErrorDisplayMode = Api.Enum.EnumValueConverte
r.GetErrorDisplayMode(appFormSettings.ErrorDisplayMode);
   }

   // Wenn das Feld aktiv wird, müssen eventuell die Menüs des HTML-Controls a
usgeblendet
   // werden (das geht technisch leider nicht, wenn man das HTML-Control verlä
sst, daher
   // müssen sich alle anderen Controls darum kümmern)
   settings.Properties.ClientSideEvents.GotFocus = "BA.Ui.RibbonUtils.HideHtml
EditorTabs";

   // Der "placeholder", also der Text, der in einem leeren Feld angezeigt wer
den soll
   settings.Properties.NullText = Api.Text.Format(valueAndAliasControl.NullTex
t);

   // vorbelegen des übergebenen im Datenmodell schon verfügbaren Wertes
   if (!String.IsNullOrWhiteSpace(presetValue))
       settings.PreRender = (sender, arg) =>
       {
           MVCxTextBox txt = sender as MVCxTextBox;
           txt.Value = presetValue;
           txt.DataBind();
       };

   // spezieller Fix für FireFox, damit Text in Feldern selektiert und kopier
t werden
   // können, die keine Eingabe erlauben
   RenderingUtils.FixItemForFirefox(mvcxItem, settings, valueAndAliasControl.C
ssClass);

   return mvcxItem;
}
```

Read

In der Read-Methode hat man prinzipiell die identischen Möglichkeiten wie in der Edit-Methode. Andiser Stelle sollte man nur Eelemente generieren, die das Bearbeiten nicht ermöhlichen. In diesem Falls wird ein “Label” verwendet.

```
public override void ReadMode(FormControlRendererParameterModel parameters)
{
    FormControlRendererUiParameterModel parametersUi = (FormControlRendererUiP
arameterModel)parameters;
    FormRenderingContextUi renderingContext = (FormRenderingContextUi)paramete
rsUi.RenderingContext;
    String namePrefix = RenderingUtils.GetUIReadyContextPrefix(renderingContex
t);
    ValueAndAliasControl valueAndAliasControl = (ValueAndAliasControl)paramete
rsUi.Control;

    // Feldnamen setzen
    String modelFieldName = valueAndAliasControl.OrmFieldName;

    // Alternativ kann ein MVCxFormLayoutItem auf diese Weise erstellt werden.
    parametersUi.MvcxControls.Add(item =>
    {
        // Typ des Items definieren und Settings casten
        item.NestedExtensionType = FormLayoutNestedExtensionItemType.Label;
        LabelSettings labelSettings = (LabelSettings)item.NestedExtensionSetti
ngs;

        // Wert aus dem Datensatz auslesen und in als Inhalt festlegen
        char delimiter = valueAndAliasControl.Delimiter.ToCharArray()[0];
        String fieldContent = (String)parametersUi.BindData.GetPropertyValue(m
odelFieldName);
        fieldContent = fieldContent?.Split(delimiter)?[0] ?? "";
        labelSettings.Text = fieldContent;

        // Weieter Einstellungen festlegen
        labelSettings.Name = namePrefix + valueAndAliasControl.ControlInternal
Name + parametersUi.MvcxControls.Count;
        labelSettings.Width = new Unit(valueAndAliasControl.Width, Api.Enum.En
umValueConverter.GetValueWidthType(valueAndAliasControl.WidthType));
        labelSettings.ControlStyle.CssClass = valueAndAliasControl.CssClass;
        item.HorizontalAlign = FormLayoutHorizontalAlign.Left;
        item.Caption = valueAndAliasControl.Caption?.Translate() ?? String.Emp
ty;
        item.CaptionSettings.Location = Api.Enum.EnumValueConverter.GetLabelPo
sition(valueAndAliasControl.LabelPosition);
        RenderingUtils.SetHelpText(valueAndAliasControl, item);
```

```
        item.Visible = valueAndAliasControl.Visible;

        // Label setzen, falls konfiguriert
        if (String.IsNullOrWhiteSpace(item.Caption))
            item.ShowCaption = DevExpress.Utils.DefaultBoolean.False;
    });
}
```

12.1.3. Auswahlliste anpassen

In vielen Fällen ist es lediglich notwendig, die Auswahlliste anzupassen. Für diesen Fall ist es lediglich notwendig ein neues Steuerelement zu implementieren. Ein Renderer ist dabei nicht notwendig.

Es gibt eine Reihe verschiedener Steuerelemente, die zur Auswahl von Auswahllistenwerte dienen und entsprechend erweitert werden können.

- Combobox `EnumComboboxControl`
- Tokenbox `EnumTokenboxControl`
- Radiobuttons `EnumRadioButtonListControl`
- Checkboxen `EnumCheckboxListControl`
- Listbox `EnumListboxControl`
- Phasen `EnumPhasesControl`

Um die Auswahlliste programmatisch zu definieren, wird zuerst das entsprechende Steuerelement abgeleitet und ein eigenes Steuerelement geschaffen.

```
[Serializable]
[Toolbox(EnumConfigurationType.FormConfigurationGuid)]
public class MyCombobox : EnumComboboxControl
{
    public MyCombobox()
    {
        ToolboxName = "[INSERT TRANSLATION GUID]";
        ControlInitName = "MyCombobox";
        ToolboxGroupName = "[INSERT TRANSLATION GUID]";
        Icon = "[INSERT ICON NAME]";
        Id = new Guid("[INSERT UNIQUE GUID]");
        Caption = "[INSERT TRANSLATION GUID]";
        HasErrors = false;
        DesignerHintText = "[INSERT TRANSLATION GUID]";
    }
}
```

Damit würde das neue Steuerelement sich exakt genauso verhalten wie das Original. Um die Auswahlliste zu beeinflussen, muss das Property `DataProviderProperties` gesetzt werden. In diesem Beispiel wird die Auswahlliste fest mit einer eignen Auswahlliste verbunden, in dem der Datenprovider für Auswahllistenwerte fest auf die eigene gesetzt wird.

```
DataProviderProperties = new CDPEnumValuesProperties
{
    MasterGuid = EnumMyEnum.MasterGuid,
};
```

Die Standard Datenprovider für Auswahllistenwerte `CDPEnumValues` kann mit Hilfe seiner Eigenschaften `CDPEnumValuesProperties` modifiziert werden. Dazu stehen folgende Eigenschaften zur Verfügung.

- `DisplayIcon` Soll das Icon angezeigt werden.
- `ActiveState` Berücksichtigung das Aktivstatus
- `IncludeNonReadable` Anzeige auch der Werte, die der Benutzer normalerweise nicht lesen kann.
- `OnlyTheseValues` Liste von Werten, die angezeigt werden sollen.

Ist dies nicht ausreichend muss der Datenprovider `CDPEnumValues` und die Eigenschaftenklasse `CDPEnumValuesProperties` entsprechend erweitert werden. In diesem Beispiel wird die Auswahlliste fest belegt und die `OwnProperty` der eigenen Auswahlliste, muss einen bestimmten Wert haben.

```
public class CDPMyEnumValuesProperties : CDPEnumValuesProperties
{
    public string OwnPropertyEqual { get; set; }
    public CDPMyEnumValuesProperties() : base()
    {
        MasterGuid = EnumMyEnum.MasterGuid;
        DataProviderTypeFullName = typeof(CDPMyEnumValues).FullName;
    }
}
```

```
public class CDPMyEnumValues : CDPEnumValues
{
    public override IEnumerable<ValueEnum> AddWhere(IEnumerable<ValueEnum> lis
t)
    {
        IEnumerable<EnumMyEnum> returnList = base.AddWhere(list).Cast<EnumMyEn
um>();
        if (Properties is CDPMyEnumValuesProperties prop)
            returnList = returnList.Where(ff => ff.OwnProperty == prop.OwnProp
ertyEqual);
        return returnList;
    }
}
```

Bei der Zuweisung muss nun der eigene Datenprovider angegeben werden.

```
DataProviderProperties = new CDPMyEnumValuesProperties
{
    OwnPropertyEqual = "Wert",
};
```

12.2. Maskensteuerung

Mit Hilfe der Maskensteuerung ist es beispielsweise möglich abhängig von Feldwerten Elemente in der Maske zu aktivieren bzw. zu deaktivieren oder ganz auszublenden. Dazu werden sogenannte Form-Events implementiert.
Hierbei wird die Klasse `BAFormEventsBase` erweitert und das Attribut `BAFormEventsImplementation` gesetzt. Das Attribut erhält eine eindeutige Guid und einen Namen.

```
[BAFormEventsImplementation("[INSERT UNIQUE GUID]", "[INSTERT TRANSLATION GUID]")]
public class MyFormEvents : BAFormEventsBase
{ ... }
```

Folgende Events existieren

- `BeforeOpening` Wird vor allen Berechnungen oder Laden der Daten ausgeführt.
- `OnOpening` Wird vor dem Öffnen ausgeführt.
- `OnRefresh` Wird beim Aktualisieren ausgeführt. Wird in der Basisklasse bei `OnOpening` ausgeführt.
- `OnSaved` Wird beim Speichern ausgeführt. Wird in der Basisklasse bei `OnOpening` ausgeführt.

Maskensteuerelemente beeinflussen

Mit den Hilfsmethoden `AddAttributeToControl(s)` kann man Attribute bestimmten Steuerelementen hinzufügen. Man kann das Steuerelement selbst ermitteln oder über

- den Feldnamen,
- den Typen des Steuerelementes,
- den Relationstypen / Relationskategorie

bestimmen.

Beispiel: Ausblenden des Steuerelementes für den Feldnamen `LastName` eines Kontaktes.

```
public override void OnOpening(FormEventsParameters parameters)
{
    base.OnOpening(parameters);
    AddAttributeToControl(parameters, nameof(OrmContact.LastName), AttributeEnum.Skip);
}
```

Möglichkeiten von `AttributeEnum`

- `ReadOnly` Lesemodus
- `Skip` Ausblenden (Feld wird nicht gerendert)
- `Visible` Ausblenden (Feld wird unsichtbar gerendert)

- `ReadOnlyIfNotEmpty` Lesemodus, wenn das Feld nicht leer ist.
- `SkipIfNotEmpty` Ausblenden, wenn das Feld nicht leer ist (Feld wird nicht gerendert)
- `SkipRequiredValidation` Überspringt die Pflichtfeldvalidierung.

Ribbon bar-Steuerelemente beeinflussen

Auch die Aktionen in der Ribbon bar können beeinflusst werden. Um die richtigen Aktionen zu erhalten existiert die Hilfsmethode `GetActionsOfType`, um Aktionen eines bestimmten Typen zu ermitteln.

```
protected static IEnumerable<T> GetActionsOfType<T>(FormEventsParameters param
eters) where T : ActionBase
```

Mit der Klasse `RibbonActionsModificationModel` kann man nun die `DynamicClientVisibilit
y` beeinflussen.

```
public override void OnOpening(FormEventsParameters parameters)
{
    base.OnOpening(parameters);
    IEnumerable<ClientActionCreatePDFAndEMail> actions = GetActionsOfType<Clie
ntActionCreatePDFAndEMail>(parameters);
    foreach (ClientActionCreatePDFAndEMail action in actions)
    {
        RibbonActionsModificationModel modifier = new RibbonActionsModificatio
nModel();
        modifier.DynamicClientVisibility.Add(EnumActionVisibility.IfValueIsTru
e);
        modifier.AdditionalClientData.Add("TestValueIsTrue", ((OrmMyDataTabl
e)parameters.Orm).BooleanField);
        parameters.FormModel.RibbonModifications.Add(action.Id, modifier);
    }
}
```

12.3. Aktualisierung von Maskenbereichen

Es ist möglich, Teile der Bedienoberfläche während des Ablaufs von Hintergrundprozessen aktualisieren zu lassen. Ein gutes Beispiel, um dies zu beobachten ist die Serienkorrespondenz.
Beim Start der Serienkorrespondenz werden verschiedene Felder in den Lesemodus versetzt und bestimmte Teile der Maske im Verlauf der Verarbeitung regelmäßig aktualisiert, beispielsweise die Ansicht der Statusdatensätze oder beim Serienbrief die erstellte Briefdatei.

Um diese Funktionalität nutzen zu können, sind einige Voraussetzungen zu erfüllen:

- Die Maske muss aktualisierbare Elemente beinhalten. Diese umfassen Detailansichten, Dateianhänge und Aktualisierungsgruppen.
- Die Bereiche, die aktualisiert werden sollen, müssen in einem Form-Event-Handler „angemeldet“ werden.
- Die im Hintergrundprozess verwendete Session muss so eingerichtet werden, dass beim Speichern von Objekten auch die Objekte im Orm-Cache aktualisiert werden. Dies kann zu Performance-Einbußen führen und sollte nur aktiviert werden, wenn es auch wirklich notwendig ist.

Sind diese Voraussetzungen erfüllt, ist der Hintergrundprozess in der Lage, die Oberfläche zu aktualisieren.

Anmelden der aktualisierbaren Bereiche

Die Anmeldung wird üblicherweise in der `OnOpening` Methode der Maskensteuerung vorgenommen, die natürlich dann in der entsprechenden Maske konfiguriert sein muss.
Das ganze passiert über ein sog. `RefreshGroupModel`:

```
RefreshGroupModel list = new RefreshGroupModel(parameters.FormModel);
```

Die Identifikation eines aktualisierbaren Bereichs erfolgt für die unterschiedlichen Bereichstypen auf unterschiedlichem Wege. Jeder dieser Wege beinhaltet aber die Angabe eines Gruppennamen, über welchen die entsprechenden Bereiche angesprochen werden können. Pro Gruppennamen können auch mehrere Bereiche angemeldet werden, so dass alle diese Bereiche beim entsprechenden Zugriff auf diesen Gruppennamen aktualisiert werden können.

Der einfachste Fall betrifft die Dateianhänge („attachment“ ist in diesem Fall der besagte Gruppenname):

```
list.AddAttachmentAreas(null, "attachment");
```

Der nächste Fall betrifft Aktualisierungsgruppen. Aktualisierungsgruppen haben keine festgelegte ID, so dass diese nur anhand ihres Inhalts bestimmt werden können. Hierfür muss also zur Identifizierung einer Aktualisierungsgruppe, die aktualisiert werden können soll, ein Feld angegeben werden, welches sich innerhalb besagter Gruppe befindet:

```
list.Add("Subject", null, false, "subject");
```

Es ist des Weiteren auch möglich, diese Aktualisierungsgruppen über in Feldern verwendete Relationen zu identifizieren, falls Maskensteuerelemente nicht auf ein konkretes Orm-Feld sondern eben mit einer Relation arbeiten:

```
list.Add(EnumRelationType.AuthorGuid, EnumAuthorRelationSubTypes.DefaultGuid,
false, "properties");
```

Hier wird also der Bereich aktualisiert, in welchem sich das Feld zur Auswahl weiterer Bearbeiter befindet. Auf diese Art und Weise werden auch die Detailansichten innerhalb der aktuellen Maske zur Aktualisierung identifiziert:

```
list.Add(EnumRelationType.AuthorGuid, EnumAuthorRelationSubTypes.DefaultGuid,
false, "detail"));
```

In diesem Beispiel werden alle Detailansichten einer Maske in die Gruppe “detail” aufgenommen.Hier sieht man auch deutlich die Anwendung des Gruppennamen. Am Ende werden hier unter „detail“ ALLE Detailansichten auf der Maske aktualisiert werden können.

```
IEnumerable<SingleDetailGridViewControl> detailViews = parameters.FormConfigur
ation.FlattenControls().OfType<SingleDetailGridViewControl>();
foreach (SingleDetailGridViewControl detailView in detailViews)
    list.Add(detailView.GetRelationDefinition().RelationTypeGuid.ToString(), n
ull, false, "detail");
```

Sind alle Bereiche zur Aktualisierung identifiziert, müssen diese nur noch dem `FormModel` mitgeteilt werden, und die Anmeldung der Aktualisierungsbereiche ist somit abgeschlossen:

```
parameters.FormModel.RefreshGroups = list;
```

Verwendung in einem Hintergrundprozess

Die UI hält den Datensatz in einem Cache vor. Der Datensatz wird nur beim Umschalten in den Bearbeitenmodus neu geladen. Dieser cache wird normalerweise beim Speichern des Datensatzes im Backend nicht aktualisiert, daher kommen Änderungen von einem Hintergrundprozess normalerweise erst beim Benutzer an, wenn dieser den entsprechenden Datensatz neu lädt.

Es ist möglich, den Speichervorgang so zu erweitern, dass auch diese Cacheeinträge beim Speichern aktualisiert werden. Dies sollte nur aktiviert werden, wenn es für die UI-Aktualisierung notwendig ist.

Die Aktivierung sollte in der Session erfolgen, mit der die entsprechenden datensätze geladen wurden.

```
_UoW.UpdateCachedObjects(true);
```

Die eigentliche Aktualisierung erfolgt dann über die Client-Communication-API:

```
Api.ClientCommunication.SendMessage(new ClientCommunicationUpdateControls
{
    Oid = [OID vom Zieldatensatz],
    Keys = new List<string>() { "detail", "subject", "attachment"}
});
```

Hierbei wird die Oid des Datensatzes, dessen Maske aktualisiert werden soll, und eine Liste mit den Gruppenbezeichnern, der zu aktualisierenden Bereiche angegeben. Es ist nicht möglich mehrere Datensätze zu aktualisieren.

12.4. Datensätze initialisieren

Es ist möglich Datensätze zu initialisieren und diese anstatt zu speichern in einer Maske auf einem Tab oder in einem Dialog zu öffnen. Beispielsweise könnte man vor dem Erstellen des Datensatzes einen Dialog voranstellen und auf Basis der Eingaben des Anwenders einen Datensatz initialisieren. Anschließend wird dieser neue und ungespeicherte Datensatz in einer Maske geöffnet.

In BA werden alle neuen Datensätze sofort gespeichert und befinden sich damit in der Datenbank. Dafür erhalten diese Datensätze einen temporären Schlüssel, der sie von den realen Datensätze unterscheidet. Dabei wird nur der Datensatz selbst gespeichert. Beispielsweise werden Relationen noch nicht angelegt. Die Masken selbst arbeiten auf einem Cache, in dem der Datensatz mit allen Informationen, also auch mit den ungespeicherten Relationen, abgelegt ist.

```
// Session mit temporären Schlüssel anlegen
Guid tempKey = Guid.NewGuid();
UnitOfWork uow = Api.ORM.GetNewUnitOfWork(tempKey);
// Neuen Datensatz erstellen
OrmEngine myData = Api.ORM.GetNewORM<OrmEngine>(uow);
myData.AddSource(parentGuid, EnumRelationType.Parent, null);
myData.Name = "Name";
// Datensatz 'speichern'.
myData.Save();
uow.CommitChanges();
// Datensatz in den Cache legen
Guid cacheGuid = myData.SaveToCache();
```

Im folgenden Kapitel wird beschrieben, wie ein Datensatz in einer Maske geöffnet werden kann.

12.5. Datensatz öffnen

Möchte man den Datensatz in einer Maske öffnen, so ist dies auf verschiedenen Arten möglich. Man benötigt dazu die Guid der Maske und die Oid das Datensatzes. Mit einem temporären Schlüssel und einer Cache ID ist es möglich einen ungespeicherten Datensatz zu Öffnen.

Auf einen Tab

Direkt

```
BA.Ui.TabController.TabTools.OpenTab("/" + formGuid + "/index/" + recordOid);
```

Einen ungespeicherten Datensatz

```
BA.Ui.TabController.TabTools.OpenTab("/" + formGuid + "/index/" + recordOid +
"?TemporaryKey=" + temporaryKey + "&cacheId=" + recordCacheId);
```

Über einen Controller

Zuerst wird ein neuer Tab in TypeScript geöffnet, der auf den eigenen Controller verweist.

```
BA.Ui.TabController.TabTools.OpenTab("MyHelper/CreateRecord");
```

Am Ende des Controllers wird ein `Redirect` auf die Form durchgeführt

```
return RedirectToAction("Index", formGuid, new { id = recordOid });
```

Einen ungespeicherten Datensatz

```
return RedirectToAction("Index", formGuid, new { id = recordOid, TemporaryKey
= temporaryKey, cacheId = recordCacheId });
```

In einem Dialog

```
var formModel: Models.FormDialogModel = new Models.FormDialogModel();
formModel.RecordId = recordOid;
formModel.Form = formGuid;
formModel.Title dialogTitle;
BA.Ui.Dialog.OrmDialogManager.OpenDialog(formModel);
```

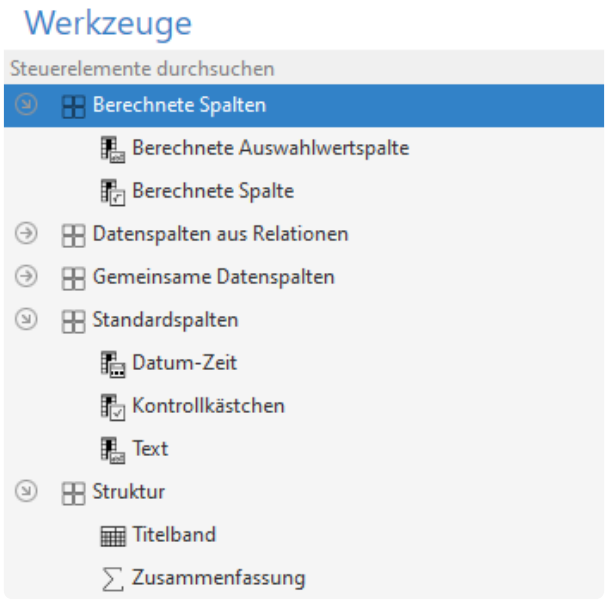
Einen ungespeicherten Datensatz

```
var formModel: Models.FormDialogModel = new Models.FormDialogModel();
formModel.RecordId = recordOid;
formModel.Form = formGuid;
formModel.TemporaryKey = temporaryKey;
formModel.RecordCacheId = recordCacheId;
formModel.Title dialogTitle;
BA.Ui.Dialog.OrmDialogManager.OpenDialog(formModel);
```

13. Ansichten

13.1. Spaltensteuerelemente

Als Steuerelement werden generell die Komponenten der Business App Anwendungskonfiguration bezeichnet, welche im Designer aus der linken Spalte in die Mitte gezogen und konfiguriert werden können. Spaltensteuerelemente sind die Elemente, die beim Erstellen einer Ansicht zur Auswahl stehen.



Ein Spaltensteuerelement besteht üblicherweise aus mindestens zwei C# Klassen.

In dem Beispiel, soll ein neues Spaltensteuerelement erstellt werden, welches den Wert aus einer Spalte anzeigt. Falls dieser Wert `null` ist wird stattdessen ein konfigurierter Wert angezeigt. Zusätzlich wird noch ein Property angeboten, mit dessen Hilfe man `EncodeHTML` beeinflussen kann.

13.1.1. Klasse des Steuerelements

Wie in den Kapiteln über Ribbon bar Aktionen und Maskensteuerelmente ebenfalls beschrieben, wird zunächst das Steuerelement für die Toolbox im Designer benötigt. Mit dem Attribut `Toolbox` wird es dem Ansichtenbereich hinzugefügt.

Um das korrekte Drag & Drop Verhalten zu ermöglichen, wird das Interface `IGridDataColumnControl` implementiert. Damit gilt die neue Spalte als eine normale Datenspalte. Als Basisklasse muss mindestens `GridColumnBase` genutzt werden. Möchte man eine Datenspalte von der aktuellen Datentabelle anzeigen kann man von `GridDataColumnBase` ableiten und hat damit eine Feldauswahl (inkl. Teil-Datentabellen-Felder). Folgende virtuelle Methoden sind dabei zu beachten:

- `GetTypesOfOrmFieldName` Beeinflusst die Auswahl der Felder. Dafür werden die unterstützen Typen von Datentabellenfeldern zurückgegeben. Beispielsweise `OrmTextField` für Textfelder. Die Basis Klasse aller Felder ist `OrmFieldBase`.
- `GetDataType` Ermittelt in der Basisimplementierung den Datentypen der Spalte auf Basis des gewählten Feldes. Beispielweise für `OrmTextField` `string`. Möchte man davon abweichen, ist die Methode zu überladen.

Die Werte von `EnumColumnType` , um den `ColumnType` zu setzen, entsprechen den Werten von `MVCx GridViewColumnType` von Dev Express.

In diesem Beispiel werden zwei Eigenschaften implementiert, welche genutzt werden, damit der Konfigurator das Verhalten dieser neuen Spalte beeinflussen kann.

```
[Serializable]
[Toolbox(EnumConfigurationType.GridConfigurationGuid)]
public class MyTextColumn : GridDataColumnBase, IGridDataColumnControl
{
    public MyTextColumn()
    {
        ColumnType = Api.Enum.EnumValueConverter.GetEnumColumnTypeValue(EnumDa
taTypes.String);
        ToolboxName = "My Text Column";
        Caption = "My Text Column";
        ControlInitName = "MyTextColumn";
        ToolboxGroupName = Constants.Designer.ToolboxGroups.Grid.Basic;
        Icon = Constants.Icons.Designer.Grid.TextBox;
        Id = [INSERT UNIQUE GUID]".ToGuid();
    }

    [DisplayName("Default value")]
    [HelpText("If value is empty this value is used")]
    [PropertiesGroup("My properties group", 10), PropertiesForm(10)]
    public string DefaultValue { get; set; }
```

```
    [DisplayName("Encode HTML")]
    [HelpText("Encode the HTML content")]
    [PropertiesGroup("My properties group", 10), PropertiesForm(20)]
    public bool EncodeHTML { get; set; } = true;
}
```

13.1.2. Klasse des Renderers

Wie für die Maskensteuerelemente werden auch für die Ansichten Spalten und Renderer benötigt. Mit dem Attribut `GridColumnControlRenderer` wird der Renderer dem Steuerelement zugeordnet. Der Renderer muss das Interface `IGridColumnControlRenderer` implementieren.

```
[GridColumnControlRenderer(typeof(MyTextColumn))]
public class MyTextColumnDefaultRenderer : IGridColumnControlRenderer
{ ... }
```

Datenbankabfrage

Der wesentliche Aspekt bei der Performance von Ansichten ist die Laufzeit der Datenbankabfragen. Es ist sehr wichtig darauf zu achten, dass die Ansicht mit **einer** Abfrage geladen werden kann. Beispielsweise kann es passieren, das pro angezeigter Zeile weitere Abfragen gestartet werden, dies ist unbedingt zu vermeiden. Auch auf möglicherweise fehlende Indices ist zu achten.

Für die Abfrage wird die Formelsprache eingesetzt, die den Criteria Operators von Dev Express entspricht. In `CriteriaOperatorBuilder` befinden sich eine Reihe von Hilfsmethoden zum Erstellen solcher Formeln.

Um in Formeln Joins zu ermöglichen, kann man sogenannte Free Joins verwenden. Dev Express optimiert die Abfragen, das heißt identische Free Joins in verschiedenen Spalten münden in einen SQL Join.

In `GridRendererHelper` befinden sich Hilfsmethoden, die zum Erstellen der `GridViewQueryColumnProperties` genutzt werden können. Zu beachten ist, dass sowohl in `GetQueryProperties` als auch später `GetRenderColumns` eine Liste zurück geliefert wird. Damit ist es möglich, dass pro Steuerelement mehrere Spalten generiert werden können. Dies können auch unsichtbare Spalten sein, mit deren Hilfe Funktionalitäten gebaut werden können.

In diesem Beispiel werden die Bestandteile von `GridViewQueryColumnProperties` mit Hilfe von Methoden aus `GridRendererHelper` gefüllt.

- `Name` Der eindeutige Spaltenname. Auch bei mehrfacher Verwendung des Steuerelements, muss der Name eindeutig sein.
- `DataType` Der C# Datentyp der Spalte. Dieser sollte in der Regel dem Typen entsprechen, der im Steuerelement festgelegt wurde. Durch die Hilfsmethode wird dies sichergestellt.
- `Criteria` Die Formel zum Berechnen der Spalte

```
public IEnumerable<GridViewQueryColumnProperties> GetQueryProperties(GridViewC
onfiguration configuration, GridColumnBase column)
{
    MyTextColumn myTextColumn = (MyTextColumn)column;
```

```
    string columnName = GridRendererHelper.GetDataColumnBaseGridColumnName(con
figuration, myTextColumn);
    Type dataType = GridRendererHelper.GetDataColumnBaseDataType(configuratio
n, myTextColumn);
    CriteriaOperator criteria = GridRendererHelper.GetDataColumnBaseCriteria(c
onfiguration, myTextColumn);
    return new[] { new GridViewQueryColumnProperties { Name = columnName, Data
Type = dataType, Criteria = criteria } };
}
```

Um die konfigurierbare Eigenschaft des Steuerelements in der Formel zu berücksichtigen, muss dies eingefügt werden. Dabei die Hilfsmethode `GetDataColumnBaseCriteria` verwendet, um auf die Formel für den Zugriff auf die Spalte zu bekommen. Diese kann dann weiter verwendet werden.

```
CriteriaOperator criteria;
if (!string.IsNullOrEmpty(myTextColumn.DefaultValue))
{
    CriteriaOperator fieldValue = GridRendererHelper.GetDataColumnBaseCriteri
a(configuration, myTextColumn);
    CriteriaOperator expression = fieldValue == new OperandValue(null);
    CriteriaOperator ifpart = new OperandValue(myTextColumn.DefaultValue);
    CriteriaOperator elsepart = fieldValue;
    criteria = new FunctionOperator(FunctionOperatorType.Iif, expression, ifpa
rt, elsepart);
}
else
    criteria = GridRendererHelper.GetDataColumnBaseCriteria(configuration, myT
extColumn);
```

Rendern

Für das Rendern werden MVCxGridViewColumn von DevExpress benötigt. Mit der Hilfsmethode `CreateDataBaseRenderColumn` wird automatisch eine entsprechende Spalte mit allen Einstellungen generiert und man muss lediglich die gewünschten Änderungen anschließend durchführen. In unserem Beispiel ist dies das Setzen von `EncodeHTML`.

```
public IEnumerable<MVCxGridViewColumn> GetRenderColumns(GridViewConfiguration
configuration, GridColumnBase column, GridViewRendererParameter parameter)
{
    MyTextColumn myTextColumn = (MyTextColumn)column;
    MVCxGridViewColumn gridColumn = GridRendererHelper.CreateDataBaseRenderCol
umn(configuration, myTextColumn, parameter);
    gridColumn.PropertiesEdit.EncodeHtml = myTextColumn.EncodeHTML;
    return new[] { gridColumn };
}
```

Hat man in `GetQueryPropertiesq` einen anderen Spaltennamen definiert, muss dieser hier ebenfalls neu auf `Name` und `FieldName` gesetzt werden. Bei Name wird der Präfix “Col” benötigt.

```
gridColumn.Name = "Col" + columnName;
gridColumn.FieldName = columnName;
```

Rendern und Abfragen unterdrücken

Die Abfrage der Spalten und das Render der Spalten kann man mit Hilfe von `CanColumnBeRendered` unterdrücken.

```
public bool CanColumnBeRendered(GridViewConfiguration configuration, GridColum
nBase column)
{
    return true;
}
```

13.1.3. Mehrfachgruppierung

Eine Mehrfachgruppierung zeigt einen Datensatz mehrfach an. Beispielsweise abhängig davon wie viele Auswahlwerte gewählt worden sind.

Das Steuerelement wird wie vorher implementiert, nur für das Drag & Drop wird anstatt `IGridDataColumnControl` das Interface `IGridGroupingColumnControl` implementiert.

Der Renderer implementiert `IGridColumnMultiGroupControlRenderer` welches das Interface `IGridColumnControlRenderer` erweitert. Im Folgenden werden nur die Unterschiede zu den vorherigen Renderern beschrieben.

Datenbankabfrage

In der Formel für die Abfrage in `GetQueryProperties` muss diese Gruppierung generiert werden. In dem Beispiel werden Firmen nach den “Weiteren Bearbeiter” gruppiert.

```
OperandProperty entityTilteCriteria = CriteriaOperatorBuilder.GetCommonField(EnumCommonFields.EntityTitle.InternalName);
CriteriaOperator criteria = CriteriaOperatorBuilder.GetSourcesJoin(false, EnumRelationType.Author, EnumAuthorRelationSubTypes.DefaultGuid, aggregatedExpression: entityTilteCriteria);
```

Zusätzlicher Filter

In der Regel möchte man, dass Datensätze, die keine Werte haben auch nicht in der Ansicht auftauchen, um leere Kategorien zu vermeiden. Dazu kann man in `GetFilterExpression` eine Filter-Formel definieren, die den anderen Filtern hinzugefügt wird. Filter-Formeln müssen einen booleschen Wert zurückliefern.

```
public virtual CriteriaOperator GetFilterExpression(GridColumnBase column)
{
    return CriteriaOperatorBuilder.GetSourcesJoin(false, EnumRelationType.Author, EnumAuthorRelationSubTypes.DefaultGuid, onlyRelations: true, aggregator: Aggregate.Exists);
}
```

Eindeutige Zeilen-ID

Ansichten benötigen eine eindeutige Zeilen-ID. Falls dies in einer mehrfachgruppierten Ansicht nicht korrekt ist verhält sich die Ansicht nicht korrekt. Beispielsweise tauchen leere Zeilen auf. In flachen Ansichten wird automatisch dafür die `Oid` des Datensatzes verwendet. In unserem Beispiel nehmen wir die `Oid` des Relationsdatensatzes.

```
public virtual CriteriaOperator GetKeyExpression(GridColumnBase column)
{
    OperandProperty relationOid = CriteriaOperatorBuilder.GetField(nameof(OrmR
elation.Oid));
    return CriteriaOperatorBuilder.GetSourcesJoin(false, EnumRelationType.Auth
or, EnumAuthorRelationSubTypes.DefaultGuid, onlyRelations: true, aggregatedExp
ression: relationOid);
}
```

13.2. Eigene Datenprovider

Ansichten basieren auf einer Konfiguration vom Typ `GridViewConfiguration`. Konfigurierte und dynamisch generierte Ansichten erzeugen den Datenprovider auf Basis der gesetzten Datentabellen und Spalten automatisch.

In freien Dialogen ist es unter Umständen sinnvoll eine temporäre `GridViewConfiguration` zu erstellen. In diesem Fall muss man einen eigenen Datenprovider implementieren, beispielsweise , wenn man eigene Daten anzeigen möchte oder wenn die Filter zur Laufzeit modifiziert werden sollen.

Dazu setzt man auf `GridViewConfiguration` die Eigenschaft `DataProviderProperties`. Dieses Objekt muss vom Typ `GridDataProviderPropertiesBase` sein. Im einfachsten Fall setzt man dort `DataProviderTypeFullName` auf den vollständigen Klassennamen des eigenen Dataproviders. Dieser muss das Interface `IGridDataProvider` implementieren. Die Properties-Klasse steht im Datenprovider zur Verfügung. Damit hat man Zugriff auf die `GridViewConfiguration`. Wenn man eine eigene Properties-Klasse implementiert, kann man damit das Verhalten des eigenen Datenproviders abhängig implementieren.

Beispiel eigene Properties Klasse

```
public class GDPMyOwnProperties : GridDataProviderPropertiesBase
{
    public bool WithReadPermissions { get; set; }
    public string ConfiguredValueFilter { get; set; }
    public string ConfiguredValueIf { get; set; }

    public GDPMyOwnProperties() : base()
    {
        DataProviderTypeFullName = typeof(GDPMyOwn).FullName;
    }
}
```

Beispiel eigenes Row-Model

```
public class MyOwnRowModel : GridDataProviderRowBase
{
    public string Name { get; set; }
    public string ConfiguredValue { get; set; }
    public string CriteriaValue { get; set; }
}
```

Beispiel eigener Datenprovider

In diesem Beispiel findet man

- Abhängig von den Properties wird ein Query mit oder ohne Leserechte erstellt.
- Abhängig von den Properties werden verschiedene Filter gesetzt.
  - Einer mit Hilfe einer Formel
  - und einer auf ein konfiguriertes Feld.
- Das Row-Model wird erstellt.
  - Zuweisung von programmierten Felder (`Oid`)
  - Zuweisung von konfigurierten Felder (`ConfiguredValue`)
  - Zuweisung mit Hilfe einer Formel

```
public class GDPMyOwn : IGridDataProvider
{
    public GridDataProviderPropertiesBase Properties { get; set; }

    public IQueryable<GridDataProviderRowBase> GetData(GridDataProviderParamet
er parameter = null)
    {
        if (Properties is GDPMyOwnProperties myProperties)
        {
            Session session = Api.ORM.GetNewSession();
            IQueryable<OrmUserProfile> query;
            if (myProperties.WithReadPermissions)
                query = Api.ORM.GetQueryWithReadPermissions<OrmUserProfile>(se
ssion);
            else
                query = Api.ORM.GetQuery<OrmUserProfile>(session);

            if (string.IsNullOrEmpty(myProperties.ConfiguredValueFilter))
                query = query.Where(ff => (string)ff.GetMemberValue("Configure
dValue") == myProperties.ConfiguredValueFilter);
            else
            {
                CriteriaOperator criteriaWhere = new OperandProperty("Configur
edValue") != new OperandValue(null);
                query = query.Where(ff => ff.Fit(criteriaWhere));
            }

            CriteriaOperator expression = new OperandProperty("ConfiguredValu
e") == new OperandValue(myProperties.ConfiguredValueIf);
            CriteriaOperator ifpart = new OperandValue("Replaced value");
            CriteriaOperator elsepart = new OperandProperty("ConfiguredValu
e");
            CriteriaOperator criteria = new FunctionOperator(FunctionOperatorT
```

```
ype.Iif, expression, ifpart, elsepart);

            IQueryable<MyOwnRowModel> selectQuery = query.Select(ff => new MyO
wnRowModel
            {
                Oid = ff.Oid,
                RowId = ff.Oid,
                Name = ff.EntityTitle,
                ConfiguredValue = (string)ff.GetMemberValue("ConfiguredValu
e"),
                CriteriaValue = (string)ff.Evaluate(criteria),
            });
            return selectQuery;
        }

        throw new Exception("Wrong properties");
    }
}
```

Verwendung

Um diesen Datenprovider zu nutzen, muss eine dynamische Ansichtenkonfiguration angelegt werden. In dieser Konfiguration gibt man dann seinen eigenen Datenprovider an. Die Spalten der Gridkonfiguration, müssen zu dem Rowmodel passen.

```
GridViewConfiguration engineGrid = new GridViewConfiguration()
{
    Id = "436656b6-703f-49bf-8a62-1de8d9a6f805".ToGuid(),
    ConfigurationName = "MyGridConfiguration",
    // Spaltennamen sollen dem Modelnamen entsprechen. Kein automatisches uniq
ue machen der Spaltennamen
    UseSourceColumns = true,
    DynamicallyGenerated = true,
    ShowSearchPanel = false,
    ShowFilterRowForCoummns = false,
    // Setzen des eigenen Datenproviders
    DataProviderProperties = new GDPMyProviderProperties()
};
```

14. Workflow-Aktionen

Mit Hilfe von eigenen Aktionen kann der Workflow erweitert werden.

14.1. Steuerelement

Es ist eine Klasse zu erstellen, die von `WorkflowActionBaseControl` erbt (siehe Basis-Infrastruktur von `WorkflowActionBaseControl`).

```
[Serializable]
[Toolbox(EnumConfigurationType.WorkflowConfigurationGuid)]
public class SetErwinAction : WorkflowActionBaseControl
{
    public SetErwinAction()
    {
        Id = new Guid("F6044D2C-BDE0-4BCE-BB64-9BA34CB81756");
        ToolboxName = "Setze Erwin";
        ControlInitName = "WorkflowActionSetErwin";
        ToolboxGroupName = Constants.Designer.ToolboxGroups.Workflow.Actions;
        Icon = Constants.Icons.Designer.Workflow.ChangeFieldAction;
    }
}
```

In der Klasse sind geeignete Eigenschaften für die Konfiguration der Aktion anzulegen. Beispiel:

```
[ComboboxControl]
[CDPOrmFieldsProperties(TypesToShow = new[] { typeof(OrmTextField) })]
[CDPDataSourceModifier]
[BARequired]
[DisplayName("Feldname")]
public String OrmFieldName { get; set; }
```

Für die Funktionalität für die Ausführung der Aktion ist die Methode `ExecuteAction` zu implementieren. Beispiel:

```
protected override void ExecuteAction(OrmBABase orm, LoggerToOrm logger)
{
    orm.Name = "Erwin";
}
```

Die Implementierung sollte normalerweise keine Änderungen an der Steuerelementinstanz selbst vornehmen. Ausnahmen wären Caches, die thread-safe sind und nicht persistieren.
Bitte die Regeln zum Umgang mit Workflow-Aktionen beachten.

Optional kann die Logger-Klasse, die an `ExecuteAction` übergeben wird, dazu genutzt werden, um zusätzliche Informationen in das allgemeine Workflow-Protokoll zu schreiben. Beispiel:

```
logger.AddEntry(LoggerToOrm.Translate("59155713-BADC-4F13-A886-ACF551141018",
```

```
GetControlName(LoggerToOrm.LanguageCode)));
```

Standardmäßig wird bereits protokolliert, wenn eine Workflow-Aktion ausgeführt wird, deaktiviert ist oder es bei der Ausführung zu einer Exception kam.

Die Methode `GetWFControlName` muss implementiert werden. Darin muss das Control einen menschenlesbaren Text liefern, der im Designer im mittleren Bereich für das Control angezeigt werden soll und auch im Workflow-Protokoll verwendet wird.

```
protected override string GetWFControlName(string languageCode)
```

Optional kann die Methode `IsValid` überschrieben werden, um eine individuelle Validierung zu erreichen.

```
public override List<ValidationModel> IsValid(ConfigurationBase configuratio
n, EnumDesignerValidationStateValue validationState);
{   var result = base.IsValid(configuration, validationState);
    if (blöderFehler…)
        result.Add(new ValidationModel(…));
    return result;
}
```

14.2. Basis-Infrastruktur

Jede Workflow-Aktion kann von sich aus bereits folgende Features:

- Ein Property `ControlName` erlaubt dem Benutzer eine Kurzbezeichnung für das Steuerelement im Designer anzugeben. Diese wird, falls vorhanden, vor dem Text, den `GetWFControlName` liefert, bevorzugt.
- Ein Property `Active` ermöglicht dem Anwender, jede Aktion zu deaktivieren. Ein deaktiviertes Steuerelement protokolliert diesen Zustand im Workflow-Protokoll, aber die `ExecuteAction` Methode wird nicht aufgerufen. Standardwert ist aktiv.
- Im Property `Description` kann der Anwender einen beliebigen Freitext als Kommentar hinterlegen, der ausschließlich im Designer angezeigt wird.
- Die Ausführung der Workflowaktion wird immer protokolliert, auch dann, wenn sie deaktiviert ist. Ebenso werden Exceptions bei `ExecuteAction` immer gefangen und protokolliert. Diese halten den weiteren Workflow nicht auf.
- Über das Property `Configuration` kann lesend auf die aktuelle Workflow-Konfiguration zugegriffen werden. Das ist im Besonderen interessant, um die Datentabelle zu ermitteln, zu der der Workflow gehört (`Configuration.OrmType`).

14.3. Umgang mit Workflow-Aktionen

Alle Workflow-Aktionen zu einer Datentabelle laufen in einer `UnitOfWork`, die erst am Ende, wenn alle Aktivitäten ausgeführt wurden, comitted wird. Das bedeutet im Besonderen, dass die Workflow-Aktion niemals selbst `CommitWork` aufrufen darf.

Workflow-Aktionen sind kurzläufige `WorkItems`. Das bedeutet, dass keinerlei länger dauernde Aktivitäten in der `ExecuteAction` Methode durchgeführt werden dürfen. Im Besonderen ist es verboten, eigene schreibende Transaktionen zu starten, andere als den übergebenen Datensatz zu verändern oder über Schnittstellen (Web-Services) auf fremde Systeme zu warten.
Bereits nach 60s werden die Aktionen vom Work-Manager hart beendet (kill), und die Zeit gilt für alle Aktionen in einem konfigurierten Workflow zusammen. Wenn längere Aktionen benötigt werden, sollte die Aktivität der Workflow-Aktion darin bestehen, einen neuen Hintergrundprozess zu starten.

Workflow-Aktionen können niemals rekursiv neue Workflows auslösen. Während der Ausführung des Workflows sind alle Workflow-Trigger deaktiviert. Dies gilt aber nicht für von Workflows gestartete Hintergrundprozesse, diese können weitere Workflows auslösen. Wenn das nicht gewünscht ist, sollte im Hintergrundprozess ein Workflow-Kontext erzwungen werden:

```
var workflowPrincipal = UserHelper.CurrentUser.Clone();
workflowPrincipal.AddIdentity(ExecutionEngine.WorkflowIdentity);
using (var ctx = new UserHelper.LocalUserContext(workflowPrincipal))
    // …
```

Workflow-Aktionen werden immer in der Konfigurationsreihenfolge von oben nach unten ausgeführt. Änderungen, die eine Workflow-Aktion am aktuellen Datensatz vornimmt, werden von nachfolgenden Workflow-Items gesehen. Das gilt für Bedingungen und Aktionen gleichermaßen.

Validierungsfehler an dem evtl. geänderten Datensatz treten immer erst am Ende beim `CommitWork` auf. Das bedeutet, wenn mehrere Feldwerte im Zusammenhang durch mehrere, verschiedene Workflow-Aktionen gesetzt werden, genügt es, wenn das Objekt am Ende nach allen Aktionen wieder einen validen Zustand hat.
Umgekehrt kann aber auch eine einzige Workflow-Aktion das Objekt in einen invaliden Zustand versetzen, der das Speichern als Ganzes verhindert. Dadurch werden dann alle Workflow-Änderungen an diesem Objekt faktisch rückgängig gemacht, also nicht gespeichert. Ein solcher Fehler wird immer protokolliert.

Workflow-Aktionen sollten soweit irgend möglich im Rahmen der `UnitOfWork` für den aktuellen Datensatz ablaufen. Dadurch ist sichergestellt, dass die Aktionen nicht nur zur Hälfte ausgeführt werden. Bei Nichtbeachtung dieser Regel ist es möglich, dass beispielsweise eine E-Mail mit Platzhaltern zu einem Objektzustand versendet werden, der niemals gespeichert wurde, weil anschließend die Validierung fehlgeschlagen ist.
Zur Implementierung des 2 Phase Commit kann z.B. das XPO-Event `orm.Session.AfterCommitTransaction` verwendet werden.

15. Platzhalter

Platzhalter werden typischer Weise in den Vorlagen für E-Mails und Briefe verwendet. Die Technologie der Platzhalter kann auch in eigenen Funktionalitäten genutzt werden.

15.1. Berechnete Eigenschaften

In den Platzhaltern wird häufig mehr Funktionalität zur Berechnung benötigt. Diese können mit Hilfe von eigenen berechneten Eigenschaften umgesetzt werden. Die Berechneten Eigenschaften stehen Grundsätzlich auf allen Datensätzen zur Verfügung und müssen daher so implementiert werden, dass sie mit beliebigen Datensätzen umgehen können.

Dazu wird eine statische Klasse definiert. Diese erhält das Attribut `CalculatedProperties`.

```
[CalculatedProperties]
public static class MyCalculatedProperties
{ ... }
```

Die Methoden werden als C# Erweiterungsmethoden auf `OrmBABase` implementiert und erhalten ein `IReadOnlyDictionary<string, string>` mit den Argumenten. Bei der Benamung der Methode, muss vermieden werden mit anderen Namen zu kollidieren. Daher müssen sie ein Projekt spezifisches Präfix erhalten.

```
[DisplayName("Meine Methode")]
public static string Bat_MyMethod(this OrmBABase Orm, IReadOnlyDictionary<string, string> args)
{
    return "Place holder value";
}
```

15.2. Eigene Datenquelle

Datenquellen in Platzhaltern dienen als Ausgangspunkt für die Eigenschaften, Relationen und die berechneten Eigenschaften. Damit wird beispielweise vermieden, das man in jedem Platzhalter neu über die Relationen gehen muss.

Um eine eigene Datenquelle zu definieren muss das Interface `IPlaceholderDataSource` implementiert werden. Mit der Methode `GetDataSourceName` wird der Name festgelegt, mit dem die Datenquelle angebunden wird. Mit `GetDataSource` wird die Datenquelle dem Platzhaltersystem zur Verfügung gestellt.

```
public class MailMergeDataSourceNextAddress : IPlaceholderDataSource
{
    public string GetDataSourceName()
    {
        return "myDataSource";
    }

    public object GetDataSource(OrmBABase sourceOrm, OrmBABase parentOrm)
    {
        return Api.ORM.GetOrm<MyDataTable>("[MY DATA GUID]".ToGuid(), sourceOr
m.Session);
    }
}
```

16. Migration

Migrationen erlauben die Übernahme von Datenbanken und Konfigurationen aus älteren Programmversionen.

16.1. Allgemeines

Arten von Migrationen

Datenbankmigrationen

Datenbankmigrationen werden ausgeführt, wenn die Anwendung nach einem Update zum ersten Mal startet. Dies erfolgt in 4 Schritten:

1. **Vor dem Start**
   Datenbankänderungen die den Start des ORM-Mappers verhindern.
2. **Nach dem Schema-Update**
   Datenbankänderungen, die bereits die aktualisierten Datenbanktabellen benötigen.
3. **Konfigurationsmigration**
   Aktualisierung der Konfiguration in der Datenbank = Variante von “Konfigurationsmigrationen”
4. **Nach dem Start**
   Anpassungen, die eine initialisierte Anwendung voraussetzen.

Konfigurationsmigrationen

Konfigurationsmigrationen können sowohl beim Anwendungsstart (Schritt 3) als auch beim Import (älterer) Konfigurations-ZIPs ausgeführt werden.
Für Konfigurationsmigrationen gibt es eine spezielle Basisklasse: `ConfigurationMigrationBase`.
Wird eine Konfigurationsmigration anders implementiert, funktioniert sie im Allgemeinen beim Import älterer Konfigurations-ZIPs nicht.

Migrationsnummern

Eine Migration wird über eine Nummer Identifiziert. Diese besteht aus 4 Komponenten:

1. Hauptversionsnummer des Moduls (=Assembly), zum Zeitpunkt der Erstellung, maximal zweistellig, entspricht der ersten Stelle der AssemblyFileVersion.
2. Unterversionsnummer des Moduls (=Assembly), zum Zeitpunkt der Erstellung, maximal zweistellig, entspricht der zweiten Stelle der AssemblyFileVersion.
3. Laufende Nummer des Migrationsskripts innerhalb der eben genannten Version/Unterversion, maximal zweistellig, beginnend bei 1.
4. Versionsnummer des Migrationsskriptes, optional, maximal zweistellig.

Die Migrationsnummer muss für jede Migrationsklasse definiert werden.

Beispiel:

```
[MigrationScript("0.9", 2)]
```

Das bedeutet, das zweite Migrationsskript innerhalb der Modulversion „0.9“.

Darstellung von Migrationsnummern

Modul- und Migrationsversionen werden intern numerisch als Ganzzahl dargestellt. Die Ganzzahl verwendet immer zwei Stellen pro Teil der vollen Modulversion.

Beispiel:
Volle Modulversion lautet „1.2.17.0“, damit wäre die numerische Darstellung „01021700“, die führende Null entfällt natürlich beim Ablegen als Ganzzahl. Die einzelnen Versionsteile dürfen maximal zweistellig sein.

Ausführungsreihenfolge

Alle Migrationen, werden je Migrationsschritt modulweise durchgeführt, das heißt für BACRM beispielsweise:

1. Vor dem Start, BA Core
2. Vor dem Start, BA Core Erweiterungsmodule (Designer)
3. Vor dem Start, BA Module
4. Vor dem Start, BACRM Module
5. Vor dem Start, BA.CRM
6. Vor dem Start, Projekt Erweiterungen
7. Nach dem Schema-Update, BA Core
8. Nach dem Schema-Update, BA Core Erweiterungsmodule (Designer)
9. …

Innerhalb jedes einzelnen Schrittes erfolgt die Ausführung in der Reihenfolge der Migrationsnummer. Ausnahme: Migrationsskriptversionen.

Migrationsskriptversionen

Wird eine Migration nachträglich verändert, so kann man ihre Migrationsskript-Versionsnummer um eins erhöhen. Beispiel:

```
[MigrationScript("0.9", 2, 1)]
```

Das gibt die Version 1 des Skripts und führt dazu, dass dieses Skript erneut ausgeführt wird, auch dann, wenn es in der älteren Version schon gelaufen ist oder längst Nachfolgeskripte gelaufen sind.

Es obliegt dem Migrationsskript, das dieses Feature verwendet, selbst, dabei nicht auf Fehler zu laufen. Das Skript kann über das Kontext-Objekt prüfen, ob und in welcher Version es vorher schon einmal gelaufen ist.

Kompatibilitätsgrenzen

Über Assembly-Attribute kann festgelegt werden, welche alten Datenbestände für eine Migration noch unterstützt werden. Beispiel:

```
[assembly: MigrationCompatibilitySource("0.7")]
```

Dieses Attribut erzwingt, dass die Daten für eine Migration selbiger (oder Konfiguration-ZIPs) mindestens der genannten Versionsnummer des eigenen Moduls entstammen müssen. Ältere Datenbestände müssen zunächst auf eine Zwischenversion der Anwendung migriert werden, die den Datenbestand noch unterstützt.

Die Überprüfung erfolgt modulweise. Nur wenn alle Module ihre jeweilige Vorgängerversion unterstützten, ist ein Anwendungsstart möglich.

Wenn die Kompatibilitätsversionsnummer erhöht wird, können alle Migrationen bis einschließlich zu dieser Versionsnummer aus dem Code entfernt werden.

Sonderfall Service-Releases

Bei Service-Releases kann es vorkommen, dass eine vermeintlich kleinere Versionsnummer nach einer größeren erscheint. Beispiel, in Erscheinungsreihenfolge:

- V0.8
- V0.8.1
- V0.9
- V0.8.2
- V0.9.1

Dies führt dazu, dass eine Migration von V0.8.2 auf 0.9 nicht allgemein unterstützt wird. Die Migrations-Engine nutzt dazu das Build-Datum der Module. Dieses darf sich bei einer Migration nicht reduzieren.

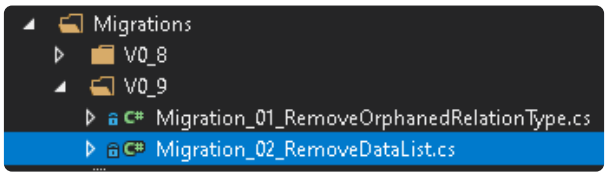
Sollte V0.8.2 unmittelbar nach V0.9 erschienen und eine Kompatibilität dennoch gewährleistet sein, muss das mit einem weiteren Attribut explizit erlaubt werden. Und zwar muss die später erschienene Version V0.8.2 ein Upgrade auf V0.9 erlauben. Es können auch mehrere Versionen erlaubt werden.

```
[assembly: MigrationCompatibilityTarget("0.9")]
```

Dieses Attribut ist beim Erscheinen eines neuen Service-Release im Allgemeinen zu entfernen oder zumindest neu zu bewerten.

16.2. Eigene Migrationen erstellen

Die neue Klasse zur Ausführung der Migration wird im Unterordner *Migrations\[Assembly-Version]* erstellt:



Die Assembly-Version entnimmt man der Datei AssemblyInfo.cs im selben Projekt:

```
[assembly: AssemblyFileVersion("0.9.0.0")]
```

Es sind nur die ersten beiden Zahlen relevant. Die Version bei der Migration darf nicht geändert werden, wenn das Assembly auf eine neue Version gehoben wird. Es ist immer die Versionsnummer zum Zeitpunkt der Erstellung.

Die Klasse sollte sich die nächste freie Migrationsnummer im selben Unterordner nehmen. Die Zählung beginnt mit jeder Versionsnummer von vorne.
Die neue Klasse jetzt mit einem Migration-Script-Attribut versehen:

```
[MigrationScript("0.9", 2)]
```

Darin muss der Versionsstring der Assembly-Version von eben entsprechen und die folgende Nummer ist die laufende Nummer der Migration.

Normale Migration (nur Datenbank)

Datenbank-Migrationen sind all jene Migrationen, bei denen der Inhalt von Konfigurationsdaten unverändert bleibt, also nichts, was im Konfiguration-ZIP stehen könnte:

- keine Änderungen an OrmConfigration
- keine Änderungen an Enums
- keine Änderungen an Übersetzungen
- keine Änderungen an Rollen

Normale Migrationsklassen erben von der Klasse `MigrationScriptBase`.

```
[MigrationScript("0.9", 2)]
```

```
public class Migration_02_RemoveDataList : MigrationScriptBase
{ … }
```

Eine Datenbank-Migration kann einen oder mehrere Migrationsschritte implementieren.

Es wird eine neue Instanz für jeden Migrationsschritt erstellt, auch, wenn eine Klasse mehrere Migrationstypen implementieren sollte. Es ist nicht möglich, sich innerhalb der Klasse Daten zwischen den Migrationsschritten zu merken! Dazwischen kann ein Anwendungsneustart liegen.

Manuelle Datenbankmigration (BeforeStart)

Dieser Bereich sollte explizit nur noch dazu verwendet werden, manuell notwendige Anpassungen an der Datenbankstruktur vorzunehmen. Das sollten ausschließlich Änderungen sein, die einen Start von DevExpress XPO verhindern.

Dafür sind ausschließlich direkte Datenbankbefehle zulässig. Über die Klasse `SQLHelper` kann und sollte das einigermaßen datenbankunabhängig geschehen. Der `SQLHelper` in der Basisklasse implementiert bereits die gängigen Hilfsmethoden, die hier zum Einsatz kommen dürften.

Es obliegt der jeweiligen Migrationsklasse, mit allen unterstützten Datenbanken klar zu kommen.

```
public override bool ExecuteTheScriptBeforeStart(MigrationExecutor executionCo
ntext)
{
    SQLHelper sql = SQLHelper.GetSQLHelper(executionContext.UnitOfWork);
    …
}
```

Die Methode gibt `true` oder `false` zurück, je nachdem ob tatsächlich Änderungen vorgenommen wurden.

Der `executionContext` enthält unter anderem eine `UnitOfWork`, die für die Änderungen verwendet werden kann. Es handelt sich um eine `ExplicitUnitOfWork`, die bis auf Datenbankebene durchgreift. Das System führt automatisch ein Commit durch.

Nach Schema-Update (AfterSchemaUpdate)

Änderungen, die den Start von XPO nicht verhindern, wohl aber den von BA gehören in diese Kategorie. Das sind allen voran Änderungen an Auswahlliste oder Übersetzungen.

Dieser Migrationstyp ist der erste, in dem es erlaubt ist, tatsächliche Daten (Verkehrsdaten sowie Daten für den Betrieb wie Auswahllisten, etc.), die in der Datenbank liegen, per SQL (oder Helper) zu migrieren.

In diesem Bereich kann bereits eingeschränkt mit den Mitteln von XPO gearbeitet werden. Es muss allerdings klar sein, dass die Konfigurationen noch nicht geladen sind. Das bedeutet, dass nicht auf konfigurierbare Entitäten zugegriffen werden sollte.

```
public override bool ExecuteTheScriptAfterSchemaUpdate(MigrationExecutor execu
```

```
tionContext)
{   …}
```

Die Methode gibt true oder false zurück, je nachdem ob tatsächlich Änderungen vorgenommen wurden.

Der `executionContext` enthält unter anderem eine `UnitOfWork`, die für die Änderungen verwendet werden kann. Es handelt sich um eine `ExplicitUnitOfWork`, die bis auf Datenbankebene durchgreift. Das System führt automatisch ein Commit durch.

Nach dem Start (AfterStart)

Der letzte Migrationsschritt dient der Datenmigration per XPO. Hier sind alle die Änderungen vorzunehmen, die den Start der Anwendung nicht verhindern. Dazu zählt auch das Entfernen nicht mehr benötigter Datenbankspalten.

```
public override bool ExecuteTheScriptAfterStart(MigrationExecutor executionCon
text)
{   … }
```

Die Methode gibt `true` oder `false` zurück, je nachdem ob tatsächlich Änderungen vorgenommen wurden.

Der `executionContext` enthält unter anderem eine `UnitOfWork`, die für die Änderungen verwendet werden kann. Es handelt sich nicht um eine `ExplicitUnitOfWork`, sondern nur um eine normale `UnitOfWork`. SQL-Befehle würde an dieser vorbei arbeiten und sollten hier nicht mehr verwendet werden. Ausnahme: Das Entfernen nicht mehr benötigter Datenbankspalten oder Tabellen.

Konfigurations-Migration

Migrationen, die Objekte verändern, die auch Teil des Konfigurations-ZIPs sind, sollten die Basisklasse `ConfigurationMigrationBase` nutzen. Darin gibt es eine API, die das verändern von Datenbank-Konfigurationen und Konfigurationen in einem gerade importierten ZIP gleichermaßen erlaubt.

```
public class Migration_01_RemoveOrphanedRelationType : ConfigurationMigrationB
ase
{
    public Migration_01_RemoveOrphanedRelationType()   : base(EnumConfiguration
Type.RelationConfiguration)   { }
    protected override void MigrateConfiguration(EnumConfigurationType type, G
uid id, ref string xml)
    { … }
}
```

Es muss dabei im Konstruktor für die Basisklasse angegeben werden, welche Typen von Konfigurationen man migrieren möchte. Nur diese werden an die `MigrateConfiguration` Methode

durchgereicht.

Die Methode `MigrateConfiguration` wird für jede einzelne, potentiell zu migrierende Konfiguration einzeln aufgerufen. Dabei wird Typ und ID der Konfiguration zur einfacheren Verarbeitung gleich mitgegeben.

Die Konfiguration wird dabei als XML-String übergeben, der beliebig verändert werden darf. Die Bearbeitung kann mit beliebigen Werkzeugen erfolgen, z.B. LINQ to XML oder DOM.
Wenn der String auf null gesetzt wird, wird die betreffende Konfiguration ersatzlos gelöscht.

Falls für die Migration einer Konfiguration der Inhalt einer anderen Konfiguration erforderlich ist, kann die Methode `GetOtherConfiguration` der Basisklasse genutzt werden, um diese zu lesen.

Gemischte Migrationen

Migrationen können sowohl Konfigurations- als auch normale Migrationsanteile enthalten. Das ist sinnvoll, wenn die Konfigurationsänderungen auch Änderungen an den Daten erfordern. In diesem Fall ist wie bei einer Konfigurationsmigration zu verfahren und zusätzlich die entsprechenden Methoden für normale Migrationen zu überschreiben.

Manuelle Konfigurationsmigrationen

Einige Migrationen können nicht über die vereinfachte API für Konfigurationsmigrationen durchgeführt werden. Dazu zählen:

- Migration von Auswahllisten
- Migration von Rollen
- Migration von Übersetzungen
- Anlegen neuer Konfigurationen durch die Migration

In diesen Fällen ist die Migration zweimal zu implementieren (und zu testen!), einmal für die Datenbank durch überschreiben der Methode `ExecuteTheScriptForConfiguration` und einmal für das ZIP durch überschreiben der Methode `ExecuteConfigurationImportMigration`.

Migration von Migrationen

Beim Einfügen neuer Migrationen kann es erforderlich sein, den Code älterer Migrationen anzupassen. Im Allgemeinen müssen dabei aber nur Migrationen angepasst werden, die nach der neuen Migration laufen. Das sind Migrationen die spätere Migrationsschritte enthalten als die der neuen Migration.

Beispielsweise kann eine neue `BeforeStart` Migration die Anpassung existierender `AfterSchemaUpdate` Migrationen erfordern, da per XPO immer nur in einem Schlag auf das aktuellste Datenbank-Schema migriert werden kann.

16.3. Anhang

Darstellung in der Datenbank

OrmMigrationModulesVersion

AssemblyGuid	OrmSchemaVersion	BuildDate	WhiteList
1F854566-B8DD-4A6A-A305-24EAAC3A8EC2	90000	2020-05-15 11:38:10.000	NULL
39B7505C-A067-4D8C-9BAC-4978E1A84AD8	90000	2020-05-14 13:49:32.000	NULL
C37DFAF1-7DCB-42FB-BEE8-575011972FAC	90000	2020-05-14 13:49:35.000	NULL

Hier ist vermerkt, mit welchem Versionsstand die Datenbank zum letzten Mal erfolgreich hochgefahren ist. Die Aktualisierung erfolgt nur, wenn alle Migrationen erfolgreich waren.

Die Schema-Version ist wie unter 1.2.1 beschrieben kodiert. 90000 entspricht also V0.9.0.0 in `AssemblyFileVersion`.

OrmMigrationLog

Hier wird gespeichert, welche Migrationen in welchen Versionen schon gelaufen sind und was das Ergebnis war.

ExecutionTime	AssemblyGuid	OrmSchemaOld	OrmSchemaNew	ScriptNumber	BeforeStart	Exec
2020-05-13 13:19:49.357	1F854566-B8DD-4A6A-A305-24EAAC3A8EC2	70000	90000	90200	9	1
2020-05-13 13:19:49.343	1F854566-B8DD-4A6A-A305-24EAAC3A8EC2	70000	90000	90100	9	0
2020-05-13 13:19:38.597	1F854566-B8DD-4A6A-A305-24EAAC3A8EC2	70000	90000	90200	3	1
2020-05-13 13:19:38.517	1F854566-B8DD-4A6A-A305-24EAAC3A8EC2	70000	90000	90100	3	0

OrmSchemaOld, OrmSchemaNew

Das sind die Assembly-Versionen (laut `AssemblyFileVersion`) die vor der Migration zuletzt in der Datenbank lief und die während der Migration verwendet wurde. Die Kodierung ist wie unter Allgemeines beschrieben. 90000 entspricht V0.9.0.0. Ein Wert 1 bedeutet durchgeführt mit < V0.9, wo dieses Feature noch nicht unterstützt wurde.

Die Haupt- und Unterversion von `OrmSchemaNew` muss nicht notwendigerweise der des Migrationsskriptes entsprechen. Das Skript kann auch erst mit einer neueren Version ausgeführt werden, wenn eine Version übersprungen wurde. Genau das wird hier dokumentiert.

BeforeStart

- 1 = vor dem Start
- 2 = nach dem Schema-Update
- 3 = Konfigurationsmigration
- 9 (früher 0) = nach dem Start

ExecutionState

- 0 = erfolgreich ausgeführt
- 1 = ausgeführt, aber ohne Änderung
- 2 = übersprungen wegen Neuaufbau der Datenbank
- 10 = wird gerade ausgeführt
- 20 = Mit Fehler abgebrochen

Mit einer Datenbank, in der sich noch ein Eintrag mit einem Status 10 oder höher befindet, kann nicht gestartet werden.

17. Formelsprache

Basis der Formeln in BA sind die Criteria Operator, welche eine Technologie von Dev Express ist. Als Beispiele sind da zu nennen die Datensatzfilter und die Berechnungsformeln bei der Konfiguration der Kalender.

Es gibt sowohl eine Klassenstruktur als auch eine Syntax für eine textuelle Darstellung.

Syntax

Operatoren

17.1. Ausführen von Formeln

Beim Ausführen von Formeln muss beachtet werden, in welchem Kontext diese ausgeführt werden. Beispielsweise können Free Joins (`JoinOperand`) nur im Rahmen von Datenbankabfragen genutzt werden. Im Gegensatz dazu können Zugriffe auf Eigenschaften, wie die Relationsdefinitionen, nur in .Net ausgeführt werden.

> Datenbankabfragen sollten aus Performance Gründen vollständig an den Datenbankserver übermittelt werden können. Dies ist nur mit den Eigenschaften, die Spalten der Tabelle sind oder mit Persistent Aliase möglich. Berechnungen und einfache String oder Datums Operationen in den Abfragen sind ebenfalls in eine Datenbankabfrage umsetzbar.

Es stehen zwei Methoden zur Verfügung, um die Formeln auf die Datensätze anzuwenden. Die erste führt eine Formel aus und liefert das Ergebnis zurück.

```
public object Evaluate(CriteriaOperator expression);
public object Evaluate(string expression);
```

Beispiel:

```
string text = (string)orm.Evaluate("Iif(StartsWith([TextField], 'Inhalt'), [Te
xtField], 'Kein Inhalt da')");
```

Die zweite Methode überprüft eine Bedingung und liefert `true` oder `false` zurück.

```
public bool Fit(string condition);
public bool Fit(CriteriaOperator condition);
```

Beispiel:

```
if(orm.Fit(CriteriaOperatorBuilder.ApplyActiveFilter(null, EnumEntryActiveStat
e.Active)) { ... }
```

Beide Methoden können auch in LINQ Abfragen genutzt werden und müssen dann den entsprechenden Kriterien für Datenbankabfragen entsprechen. Die Übergabe der Criteria als String ist aber nicht möglich! Da sollte vorher ein `CriteriaOperator.Parse(...)` genutzt werden.

> Die Criteria Operators von Dev Express haben schon vor LINQ existiert. Intern transferiert Dev Express alle LINQ Queries in Criteria und anschließend in eine Datenbankabfrage.

17.2. Eigenes Formelfeld

Formeln werden in der Konfiguration als Strings erfasst. Als Eingabeelemente ist es sinnvoll ein Memofeld zu nutzen. Für Bedingungen kann der Bedingungseditor genutzt werden.

Möchte man den Criteria String in Criteria Objekte wandeln, tut man dies mit

```
CriteriaOperator criteria = CriteriaOperator.parse("Iif(StartsWith([TextFiel
d], 'Inhalt'), [TextField], 'Kein Inhalt da')");
```

Für die Syntaxüberprüfung stehen in der Klasse `CriteriaOperatorHelper` Methoden zur Verfügung.

Ausführung in .Net

```
public static Func<ST, SR> CompileExpression<ST, SR>(Type ormType, CriteriaOpe
rator criteria, ICriteriaToExpressionConverter converter = null)
```

Methode zur Kompilierung gegen eine Datentabelle mit einem gewünschten Rückgabetyp

- `ST` ORM Typ
- `SR` Rückgabetyp
- `ormType` Datentabelle, auf deren Basis kompiliert wird.
- `criteria`
- `converter` Optionaler Converter
- Rückgabe: Eine Exception oder eine ausführbare .Net Funktion.

> Die erfolgreich kompilierte Methode kann am effektivsten zur Ausführung der Formel genutzt werden.

Beispiel inkl. Kompilierung und Ausführung

```
try
{
    CriteriaOperator criteria = CriteriaOperator.Parse("Iif(StartsWith([TextFi
eld], 'Inhalt'), [TextField], 'Kein Inhalt da')");
    Func<OrmBase, object> compiled = CriteriaOperatorHelper.CompileExpressio
n<OrmBase, object>(typeof(OrmMyDataTable), criteria);
    string text = compiled(myData)?.ToString();
}
catch (Exception ex) when (!ex.IsThreadAbort())
{
    // Fehler Handling
```

```
}
```

Datenbankformeln

```
public static CriteriaOperator ValidateDBCriteria(Type ormType, Type targetTyp
e, string criteriaString)
```

Validiert den Criteria String und gibt Criteria Objekte zurück.

- `ormType` Die Datentabelle, auf der die Formel ausgeführt werden soll.
- `targetType` Erwarteter Typ der Rückgabe
- `criteriaString` Die Formel
- Rückgabe: Eine Exception oder die Criteria Objekte

Beispiel

```
try
{
    CriteriaOperato criteria = CriteriaOperatorHelper.ValidateDBCriteria(typeo
f(OrmMyDataTable), typeof(bool), "StartsWith([TextField], 'Inhalt')");
    if(orm.Fit(criteria) { ... }
}
catch (Exception ex) when (!ex.IsThreadAbort())
{
     // Fehler Handling
}
```

17.3. CriteriaOperatorBuilder

```
BA.Core.CriteriaOperators.CriteriaOperatorBuilder
```

In der Klasse befindet sich eine Sammlung verschiedener Methoden zum Erstellen komplexer Criteria Operator auf Objekt Basis. Beispielsweise für den Zugriff auf Auswahllisten, Gemeinsame Spalten, Relationen, etc.
Wird eine entsprechende String Repräsentation benötigt kann diese mit Hilfe von `ToString()` auf den Objekten abgerufen werden.

Feldwert

Um auf einen Feldwert zuzugreifen `[FeldName]`

```
public static OperandProperty GetField(string fieldName)
```

Zeitzonen

Die Zeiten werden in BA-Datensätzen als UTC Wert abgespeichert. Sollen diese Werte in Formeln genutzt werden, müssen diese in die aktuelle Zeitzone umgewandelt werden.

```
public static CriteriaOperator GetDateWithTimeZone(CriteriaOperator operand)
```

Auswahllistenwert einer Einzelauswahl

Um den Auswahllistenwert einer Einzelauswahl zu erhalten, gibt es verschiedene Methoden

```
public static CriteriaOperator GetEnumSingleDisplayable(string fieldName, stri
ng language = null)
```

Übersetzung des Wertes

- `fieldName` Der Auswahllistenfeldname im aktuellen Datensatz
- `language` Optional: Sprache der Übersetzung

```
public static CriteriaOperator GetEnumSingleColor(string fieldName)
```

Farbwert des Wertes

- `fieldName` Der Auswahllistenfeldname im aktuellen Datensatz

```
public static CriteriaOperator GetEnumSingleImage(string fieldName)
```

Bild des Wertes

- `fieldName` Der Auswahllistenfeldname im aktuellen Datensatz

```
public static CriteriaOperator GetEnumSingle(string fieldName, CriteriaOperato
r selector, MasterEnum masterEnum = null)
```

Zugriff auf beliebige Felder des Wertes

- `fieldName` Der Auswahllistenfeldname im aktuellen Datensatz
- `selector` Eine Formel, die auf den Auswahllistenwert ausgeführt wird.
- `masterEnum` Optional: Wird benötigt, wenn in der Formel auf Felder zugegriffen wird, die speziell für eine Auswahlliste ist.

Auswahllistenwerte einer Mehrfachauswahl

Der Umgang mit mehrfachen Werten in Formelsprache ist nicht trival und man sollte sich auf die Verwendung der BA eigenen Funktion `BAContainsEnumValue` beschränken.

Gemeinsame Datenfelder

Der Zugriff auf gemeinsame Datenfelder erfolgt über

```
public static OperandProperty GetCommonField(string fieldName)
```

Wobei der Feldname der Eigenschaft `InternalName` aus der Auswahlliste der gemeinsamen Datenfelder entspricht. Beispiel:

```
OperandProperty entityTitle = CriteriaOperatorBuilder.GetCommonField(EnumCommo
nFields.EntityTitle.InternalName);
```

Auswahllistenwerte in gemeinsamen Datenfeldern

Auch in diesem Fall stehen die drei Methoden für den Zugriff auf die Übersetzung, die Farbe und das Bild zur Verfügung

- `GetCommonEnumDisplayable`
- `GetCommonEnumColor`
- `GetCommonEnumImage`

Free Joins für Datenbankabfragen

Die Free Joins in Datenbankabfragen sind sehr komplex, daher stehen in dieser Library eine Reihe Methoden zur Verfügung, um dies zu vereinfachen. Benötigt werden diese, wenn man zum Beispiel eine Spalte für Ansichten implementieren möchte.

Auswahllistenwerte einer Mehrfachauswahl

In Formeln für Datenbankabfragen ist es eventuell trotzdem notwendig, die Joins auf alle Werte zu nutzen. Dies ist beispielsweise der Fall, wenn man eine eigene Spalte für die Mehrfachgruppierung in Ansichten implementieren möchte, Dazu stehen hier eine Reihe von Methoden zur Verfügung.

- `GetEnumMultiDisplayable`
- `GetEnumMultiColor`
- `GetEnumMultiImage`

Die Basis Zugriffsmethode ist `GetEnumMulti`, sie liefert den entsprechenden Free Join als `JoinOperand` zurück.

```
public static CriteriaOperator GetEnumMulti(string fieldName, Guid masterGuid, CriteriaOperator selector, bool onlyRelations = false)
```

- `fieldName` Der Auswahllistenfeldname im aktuellen Datensatz
- `masterGuid` Guid der Auswahlliste
- `selector` Eine Formel, die auf die Auswahllistenwerte ausgeführt wird.
- `onlyRelations` Optional: True, falls man die Relationsdatensätze haben möchte. In dem Fall wird der `selector` auf diese Sätze ausgeführt.

Teil-Datensätze

Für den Zugriff auf Teil-Datensätze steht die zentrale Methode `GetSubRecords` zur Verfügung.

```
public static JoinOperand GetSubRecords(OrmSubRecordField subRecordField, int position = -1, Guid? uniqueKey = null, Aggregate aggregator = Aggregate.Single, CriteriaOperator aggregatedExpression = null)
```

- `subRecordField` Das Steuerelement für Teil-Datensätze aus der Datentabellenkonfiguration
- `position` Optional: `SortOrder` Index des gewünschten Satzes
- `uniqueKey` Optional und alternativ zu `position`: Eindeutiger Schlüssel des Teil-Datensatzes. Dies kann nur genutzt werden, wenn die Verwendung des Teil-Datensatzes in der Datentabelle dies unterstützt.
- `aggregator` Optional: `Aggregate` Typ. Beispielsweise `Aggregate.Single` oder `Aggregate.Exist`
- `aggregatedExpression` Optional: Formel die auf die Sätze der Aggregation ausgeführt wird. Default ist `[This]`, was den Datensatz selbst zurückliefert.

Beispiel:

```
OrmSubRecordField subRecordField = (OrmSubRecordField)Api.Config.OrmEntity(Enu
mDataSourceExtension.MyDataTable).GetField(nameof(OrmMyDataTable.SubDatas));
OperandProperty fieldProperty = new OperandProperty(nameof(OrmSubDataTable.Tex
tField));
JoinOperand subRecordSortOrder0 = CriteriaOperatorBuilder.GetSubRecords(subRec
ordField, position: 0, aggregatedExpression: fieldProperty);
```

Relationen

Um die komplexen Relation-Joins zur vereinfachen, verwendet man die folgenden beiden Methoden. Eine für den Zugriff von Ziel zur Quelle und eine für den umgekehrten Fall.

```
public static JoinOperand GetSourcesJoin(bool onlyPrimary, EnumRelationType re
lationType, string relationCategory, GuidSet sourceTypes = null, bool onlyRela
tions = false, Aggregate aggregator = Aggregate.Single, CriteriaOperator aggre
gatedExpression = null)
```

Um auf die Quellen einer Relation zuzugreifen, steht diese Methode zur Verfügung.

- `onlyPrimary` Flag ob nur der primären Relation gefolgt werden soll. Dies muss auf `true` gesetzt sein, wenn man nur einen Datensatz erwartet.
- `relationType` Der Relationstyp
- `relationCategory` Die Relationskategorie
- `sourceTypes` Optional: Ergebnis wird auf bestimmte Typen eingeschränkt. Ist es nur ein Typ, dann kann `aggregatedExpression` auch konkrete Felder dieses Typs beinhalten.
- `onlyRelations` Optional: Bei True geht der Join nur bis zur `OrmRelation` und nicht zu den Datensätzen selbst. Dann wird `aggregatedExpression` auch auf die Relationsdatensätze ausgeführt.
- `aggregator` Optional: `Aggregate` Typ. Beispielsweise `Aggregate.Single` oder `Aggregate.Exi st`
- `aggregatedExpression` Optional: Formel die auf die Sätze der Aggregation ausgeführt wird. Default ist `[This]`, was den Datensatz selbst zurückliefert.

```
public static JoinOperand GetTargetsJoin(EnumRelationType relationType, strin
g relationCategory, GuidSet targetTypes = null, bool onlyRelations = false, Ag
gregate aggregator = Aggregate.Single, CriteriaOperator aggregatedExpression
= null)
```

Um auf die Ziele einer Relation zuzugreifen, steht diese Methode zur Verfügung.

- `relationType` Der Relationstyp
- `relationCategory` Die Relationskategorie
- `targetTypes` Optional: Ergebnis wird auf bestimmte Typen eingeschränkt. Ist es nur ein Typ, dann

kann `aggregatedExpression` auch konkrete Felder dieses Typs beinhalten.
- `onlyRelations` Optional: Bei True geht der Join nur bis zur `OrmRelation` und nicht zu den Datensätzen selbst. Dann wird `aggregatedExpression` auch auf die Relationssätze ausgeführt.
- `aggregator` Optional: `Aggregate` Typ. Beispielsweise `Aggregate.Single` oder `Aggregate.Exist`
- `aggregatedExpression` Optional: Formel die auf die Sätze der Aggregation ausgeführt wird. Default ist `[This]`, was den Datensatz selbst zurückliefert.

Hierarchie-Relationen

Falls eine Relation Hierarchisch ist, können diese Hilfsmethoden verwendet werden, um beispielsweise den obersten und den nähsten Datensatz zu erhalten.

```
public static JoinOperand GetHierarchyPrimarySourceJoin(EnumRelationType relat
ionType, string relationCategory, EnumPrimaryHierarchyRelationSourceSelector h
ierarchyRelationSourceSelector, int levelCount = 0, GuidSet sourceTypes = nul
l, bool onlyRelations = false, Aggregate aggregator = Aggregate.Single, Criter
iaOperator aggregatedExpression = null)
```

Die Methode wird verwendet, um garantiert nur einen Datensatz zu erhalten, indem dem primären Weg gefolgt wird.

- `relationType` Der Relationstyp
- `relationCategory` Die Relationskategorie
- `hierarchyRelationSourceSelector` Hierüber wird definiert, was in der Hierarchie ermittelt werden soll.
- `levelCount` Optional: Wird nur benötigt, falls der Wert von `hierarchyRelationSourceSelector`, den Level benötigt.
- `sourceTypes` Optional: Ergebnis wird auf bestimmte Typen eindeschränkt. Ist es nur ein Typ, dann kann `aggregatedExpression` auch konkrete Felder dieses Typs beinhalten.
- `onlyRelations` Optional: Bei True geht der Join nur bis zur `OrmHierarchyRelationBase` basierten Tabelle und nicht zu den Datensätzen selbst. Dann wird `aggregatedExpression` auch auf die Relationssätze ausgeführt.
- `aggregator` Optional: `Aggregate` Typ. Beispielsweise `Aggregate.Single` oder `Aggregate.Exist`
- `aggregatedExpression` Optional: Formel die auf die Sätze der Aggregation ausgeführt wird. Default ist `[This]`, was den Datensatz selbst zurückliefert.

```
public static JoinOperand GetHierarchySourcesJoin(bool onlyPrimary, EnumRelati
onType relationType, string relationCategory, EnumHierarchyRelationSourceSelec
tor hierarchyRelationSourceSelector, int levelCount = 0, GuidSet sourceTypes
= null, bool onlyRelations = false, Aggregate aggregator = Aggregate.Single, C
riteriaOperator aggregatedExpression = null)
```

Die Methode wird verwendet, um in der Hierarchie bestimmte Quellen zu finden.

- `onlyPrimary` Flag ob nur der primären Relation gefolgt werden soll. Dies muss auf `true` gesetzt sein, wenn man nur einen Datensatz erwartet.
- `relationType` Der Relationstyp
- `relationCategory` Die Relationskategorie
- `hierarchyRelationSourceSelector` Hierüber wird definiert, was in der Hierarchie ermittelt werden soll.
- `levelCount` Optional: Wird nur benötigt, falls der Wert von `hierarchyRelationSourceSelector`, den Level benötigt.
- `sourceTypes` Optional: Ergebnis wird auf bestimmte Typen eingeschränkt. Ist es nur ein Typ, dann kann `aggregatedExpression` auch konkrete Felder dieses Typs beinhalten.
- `onlyRelations` Optional: Bei True geht der Join nur bis zur `OrmHierarchyRelationBase` basierten Tabelle und nicht zu den Datensätzen selbst. Dann wird `aggregatedExpression` auch auf die Relationssätze ausgeführt.
- `aggregator` Optional: `Aggregate` Typ. Beispielsweise `Aggregate.Single` oder `Aggregate.Exist`
- `aggregatedExpression` Optional: Formel, die auf die Sätze der Aggregation ausgeführt wird. Default ist `[This]`, was den Datensatz selbst zurückliefert.

Zugriff auf das Benutzerprofil

Diese Methode erzeugt einen Join der auf das Benutzerprofil des aktuellen Benutzers verweist.

```
public static JoinOperand GetCurrentUserProfile(CriteriaOperator selector = null)
```

- `selector` Eine Formel, die auf dem Benutzerprofil ausgeführt wird.

Filter

Für Abfragen benötigt man Filter, um die korrekten Datensätze zu erhalten.

Temporary Key Filter

Der wichtigste Filter ist der Temporary Key Filter, um temporäre Datensätze auszuschließen.

```
public static CriteriaOperator ApplyTemporaryKeyFilter(CriteriaOperator criteria = null, Guid? temporaryKey = null)
```

Aktiv Filter

Um beispielsweise nur aktive Datensätze zu erhalten, muss dieser Filter gesetzt werden.

```
public static CriteriaOperator ApplyActiveFilter(CriteriaOperator criteria, EnumEntryActiveState activeState)
```

Datensatzfilter

Abfragen sollten möglichste immer auf die gewünschten Typen eingeschränkt werden, falls globale Abfragen erstellt. Dazu dient diese Methode

```
public static CriteriaOperator ApplyObjectTypeFilter(CriteriaOperator criteri
a, Guid dataSourceGuid, CriteriaOperator selector = null)
```

Leserechte

Um die Leserechte zu berücksichtigen, muss dieser Filter gesetzt werden.

```
public static CriteriaOperator ApplyReadPermissions(CriteriaOperator criteri
a, Type queryOrmType, GuidSet dataSources = null, bool doNotAddOrmTypeCheck =
false)
```

17.4. Eigene Funktionen

Dev Express bietet selbst Erweiterungen an: Custom Functions Dev Express

Diese Möglichkeiten sind für den Einsatz in Business App nicht ausreichend. Da hier die Anforderung besteht komplexe oder variable Criteria Operators für die Konfiguration zu vereinfachen. Dafür wird eine Criteria Operators Funktion implementiert, die als Ergebnis selbst wieder einen Criteria Operator zurückliefert.

> Falls man unabhängig von den hier beschriebenen Möglichkeiten eigene Custom-Functions auf Basis der Criteria Operators Möglichkeiten implementieren möchte, müssen diese beim Start des Systems registriert werden. Dazu wird ein static Konstruktor einer beliebigen Klasse definiert. In dem Konstruktor wird die Registrierung vorgenommen, und die Klasse wird mit dem Attribut `StaticInit` versehen.

Implementierung eigener Funktionen

Ziel unserer Erweiterung von Custom Criteria Operators Funktionen ist es Teile eines Criteria Operator Ausdrucks zu ersetzen. Entsprechende Erweiterungen im Core befinden sich im Packet `BA.Core.CriteriaOperators.Functions`.

Um eine Erweiterung zu implementieren, muss das Interface `ICriteriaOperatorBAFunction` implementiert werden. Dieses erweitert das Interface `ICustomFunctionOperator` von Dev Express. Wird das BA eigene Interface, implementiert werden diese automatisiert beim Starten der Anwendung registriert.

Die Interface Methode `GetCriteriaDB` muss implementiert werden und ersetzt im Criteria Operator Ausdruck die eigene Funktion durch den Rückgabewert. Die Operanden sollten auf ihre Gültigkeit validiert und bei Fehlern Exceptions geworfen werden. Damit wird sichergestellt, dass beim Parsen entsprechende Fehler an den Konfigurator weitergegeben werden.

Zusätzlich müssen folgende Interface-Teile implementiert werden.

- `ICustomFunctionOperator.Name` definiert den Namen innerhalb eines Criteria Operators. Hierbei sollte mit entsprechenden Präfixen gearbeitet werden, um Kollisionen zu vermeiden.
- `ICustomFunctionOperator.ResultType` Definiert den Rückgabewert.
- `ICustomFunctionOperator.Evaluate` Wird benötigt, wenn in .Net die `Evaluate` Methode ausgeführt wird.
- `ICustomFunctionOperator.Convert` Wird benötigt, wenn von .Net `Expression` unterstützt werden soll.

Beispiel:

In diesem Beispiel wird ein neuer Operator “BATMyText” mit einem String Parameter definiert. Dieser

Operator gibt den Parameter String unverändert zurück.

```
Iif([BooleanField], [TextField], BATMyText('Mein Text'))
```

```
public class MyTextOperator : ICriteriaOperatorBAFunction
{
    public const string Name = "BATMyText";

    string ICustomFunctionOperator.Name => Name;

    public Type ResultType(params Type[] operands) => typeof(string);

    public virtual Expression Convert(ICriteriaToExpressionConverter converte
r, params Expression[] operands)
    {
        if (operands.Length == 1 && operands[0].Type == typeof(string))
            return Expression.Default(typeof(string));

        throw new ArgumentException("Die Funktion BATMyText benötigt einen Par
ameter");
    }

    public virtual object Evaluate(params object[] operands)
    {
        if (operands.Length == 1 && operands[0] is string operand1)
            return operand1;

        throw new ArgumentException("Die Funktion BATMyText benötigt einen Par
ameter");
    }

    public virtual CriteriaOperator GetCriteriaDB(FunctionOperator function)
    {
        if (function.Operands.Count == 2 && function.Operands[1] is OperandVal
ue value && value.Value is string operand1)
            return new OperandValue(operand1);

        throw new ArgumentException("Die Funktion BATMyText benötigt einen Par
ameter");
    }
}
```

Funktionen in der Programmierung

Um Funktionen von Dev Express per Programmierung zu definieren, muss ein `FunctionOperator` erstellt werden.

```
FunctionOperator dayOfYear = new FunctionOperator(FunctionOperatorType.GetDayO
fYear, new OperandProperty("DateField"));
```

In diesem Fall wird der Tag des Jahres eines Datumwertes ermittelt. In dem Beispiel ist zu beachten, das in BA die Datumswerte immer in UTC abgespeichert werden. Anstatt wie im Beispiel den Datumswert korrekt zu erhalten, muss dieser in die aktuelle Zeitzone umgewandelt werden. Dies dient hier als Beispiel wie Eigene Funktionen bei der programmatischen Definition von Formeln genutzt werden.

```
FunctionOperator dateValue = new FunctionOperator(ToCurrentTimeZoneOperator.Na
me, new OperandProperty("DateField"));
FunctionOperator dayOfYear = new FunctionOperator(FunctionOperatorType.GetDayO
fYear, dateValue);
```

18. Generierte Masken

Das Business App System stellt die Möglichkeit zur Verfügung aus einer Klasse automatisiert eine Maske zu erzeugen, welche verwendet wird, um die Eigenschaften dieser Klassen vom Anwender bearbeiten zu lassen. Diese Technologie wird z. Zt. bei den Einstellungen, bei den Auswahlwerten und bei der Implementierung eigener Steuerelemente verwendet.

Die Eigenschaften dieser Klassen werden automatisiert in entsprechende Steuerelemente zur Eingabe umgewandelt. Falls dies nicht gewünscht ist muss an der Eigenschaft das Attribut

```
[Browsable(false)]
```

angegeben werden.

18.1. Allgemeines

Labels und Hilfetexte

Mit dem Attribut

```
[DisplayName("Betreff")]
```

Sollte das Label angegeben werden. Die Angabe einer Guid einer Übersetzung ist ebenfalls möglich.

```
[DisplayName("[INSERT TRANSLATION GUID]")]
```

> Es gibt dieses Attribut zweimal. Es muss der Namesspace „System.ComponentModel“ verwendet werden.

Mit dem Attribut

```
[HelpText("[INSERT TRANSLATION GUID]")]
```

kann der Hilfetext definiert werden.

Validatoren

Validatoren werden über ein Attribut definiert. Dabei wird das gleiche Konzept wie bei den Validatoren für Datentabellen genutzt. Ob der Validator genutzt werden kann, ist lediglich davon abhängig, wie die eigentliche Validierung implementiert ist. Manche Validatoren benötigen für ihre Funktionsweise ein ORM Objekt.

Die wichtigsten Validatoren sind:

- `[BARequired]` Pflichtfeld
- `[BAConditionalRequired(propertyName, expressionType, propertyValue)]` Bedingte Pflichtfeldvalidierung
- `[BAConditionalRequired(method)]` Validierung durch eine angegebene Methode mit der Signatur
  `public ValidationResult MyValidation(object value, string propertyName)`

Sichtbarkeitssteuerung

Mit dem Attribut `[ConditionalShow(...)]` kann die Darstellung flexibel gestaltet werden, entweder über eine Expression oder durch eine Methode. Die Methode muss folgende Signatur haben.

```
public bool IsMyVisible()
```

Position und Gruppe

Für die Darstellung der Felder im Designer können Gruppen gebildet werden. Alle Eigenschaften mit der identischen Übersetzung (Gleiche Guid) werden zusammengefasst. Mit der Nummer kann man die Gruppen sortieren. Alle Eigenschaften, die dieses Attribut nicht besitzen werden in die Gruppe “Allgemeine Einstellungen” (`BA62093B-048C-4842-B9E1-9400ECE745B1`) einsortiert.

```
[PropertiesGroup("[INSERT TRANSLATION GUID]", 2)]
```

Für die Sortierung innerhalb einer Gruppe kann dieses Attribut genutzt werden.

```
[PropertiesForm(10)]
```

Sinnvoll ist es bei den Sortierungsnummer Lücken zu lassen, dass Erweiterungen des Steuerelementes diese nutzen können.

18.2. Eingabeelemente

Abhängig vom Datentyp der Eigenschaft werden die Eingabeelemente erstellt. Dies kann im Allgemeinen durch weitere Attribute modifiziert werden.

Zahlen

Für alle Zahlen wird das `SpinEditControl` generiert. Dabei werden als Grenzen die Grenzen der Datentypen gesetzt. Mit dem Attribut `[SpinEditSettings]`, kann man eigene Grenzen definieren. Mit Hilfe eine Modifizierungsmethode, kann man die Eigenschaften des Spinelementes flexible modifizieren. Die Methode sollte folgende Signatur haben

```
public void ModifyMySpinEdit(ConfigurationBase configuration, string propertyN
ame, SpinEditSettingsAttribute settings)
```

Boolean

Für die `Boolean` Eigenschaften wird eine Checkbox `CheckEditControl` generiert.

DateTime

Für die `DateTime` Eigenschaften wird ein `DateEditControl` generiert. Falls das Attribute `[TimeEdi
t]` gesetzt ist wird stattdessen ein `TimeEditControl` erstellt.

String

Für `String` Eigenschaften gibt es eine Vielzahl verschiedener Möglichkeiten per Attribut das Verhalten zu beeinflussen. Gibt man kein entsprechendes Attribut an, wird ein `TextEditControl` erstellt.

Mehrzeilig

Um ein mehrzeiliges Eingabefeld zu erstellen, muss das Attribut `[MemoControl]` gesetzt werden. Es ist möglich die Höhe und die Position des Labels zu definieren.

Übersetzungen

Für die Auswahl oder Neuerstellung von Übersetzungen wird das `[Translate("Description")]` Attribut gesetzt. Als Ergebnis steht die Guid der Übersetzung als String in der Eigenschaft. Dem Attribut wird ein Wert übergeben, der als Description in die `OrmTranslation` Tabelle übernommen wird. Dadurch wäre es möglich die entsprechenden Übersetzungen zu identifizieren.

HTML

Mit `[HTMLControl]` wird ein HTML Eingabeelement erstellt. Die vollständige Funktionalität dieses Elementes wird nicht an allen Stellen garantiert. Insbesondere das Speichern von Inline Images bedarf weitere Mechanismem.

Farbauswahl

Mit `[ColorPicker]` wird eine Farbauswahl erstellt.

Passwort

Mit dem Attribute `[DataType(DataType.Password)]` wird die Eingabe für ein Passwort erstellt. Dabei wird aber der aktuelle Wert aus der Eigenschaft nicht übernommen. Wenn der Inhalt der Maske übertragen wird, ohne dass der Anwender das Passwortfeld belegt hat, wird der Wert der Eigenschaft gelöscht.

Bedingungseditor

Das Attribut `[FilterEditorPropertyControl]` ermöglicht eine vereinfachte Eingabe von Bedingungen in Formeln.

18.3. Auswahllisten für String-Eigenschaften

Möchte man für eine Eigenschaft von Datentyp `String` eine Auswahl anbieten, geschieht dies über die Angabe eines Steuerelementes und eines Datenproviders. Es ist sowohl möglich ein eigenes Steuerelement als auch einen eigenen Datenprovider zu implementieren oder man verwendet vorhandene Elemente.

Steuerelement

Zurzeit existieren zwei Steuerelemente:

```
[ComboboxControl]
[TokenboxControl]
```

Die Combobox wird für die Einzelauswahl und die Tokenbox für die Auswahl mehrerer Werte verwendet. Möchte man ein eigenes Steuerelement implementieren, muss man eine Klasse erstellen die `SelectionControlAttributeBase` erweitert und die Methode `CreateControls` implementiert.

Datenprovider

Es gibt eine Vielzahl verschiedener Datenprovider für Steuerelemente. Zu erkennen sind diese an dem Präfix „CDP“ und befinden sich unterhalb des Namespaces `BA.Core.DataProviders`.

Jeder Datenprovider besteht aus zwei Klassen:

- dem Datenprovider selbst und
- dessen Eigenschaften mit deren Hilfe das Verhalten angepasst werden kann.

Um einen Datenprovider zu verwenden, wird die Klasse mit den Eigenschaften als Attribut angegeben. Beispielsweise:

```
[CDPEnumValuesProperties(masterGuid: EnumWeekDays.Guid)]
```

Jeder Datenprovider hat andere Eigenschaften, mit denen er beeinflusst werden kann. Schauen sie sich die jeweiligen Möglichkeiten genau an.

In manchen Situationen sind die Eigenschaften variabel und können daher nicht am Attribut selbst angegeben werden. Dafür existieren Klassen mit deren Hilfe die Eigenschaften modifiziert werden können. Beispielsweise:

```
[CDPDataSourceModifier(UseConfiguration = true)]
```

Mit dessen Hilfe wird die Eigenschaft `DataSource` oder `DataSources` gesetzt werden. Diese beinhaltet üblicherweise die `Guid` oder das `GuidSet` der Datentabellen. In dem Beispiel wird die Datentabelle aus der Konfiguration übernommen. Weitere Möglichkeiten entnehmen sie der

Klassendokumentation.

Eine einfache Möglichkeiten Modifizierungen vorzunehmen ist das Attribut

```
[CDPModifier(ModifierMethod = nameof(ModifyDataProviders))]
```

Dort gibt man eine Methode aus der aktuellen Klasse an und modifiziert dort die Eigenschaften des Datenproviders

```
public void ModifyDataProviders(ConfigurationBase configuration, string proper
tyName, ControlDataProviderPropertiesBase properties)
{
    if (properties is CDPOrmFieldsProperties prop && propertyName == nameof(Or
mFieldName))
    {  …  }
}
```

Beispiele

Einzelauswahl einer Vorlage beliebigen Typs:

```
[ComboboxControl]
[CDPOrmDataProperties(dataSources: new string[] { EnumDataSource.TemplateBaseG
uid })]
public string DefaultTemplate { get; set; }
```

Mehrfachauswahl von öffentlichen Ordnern. Die Ordner müssen die Datentabellen beinhalten können, welche in `OrmDataSources` angegeben sind.

```
[TokenboxControl]
[CDPRecordCollectionsProperties(onlyPublic: true)]
[CDPDataSourceModifier(ControlDataSourceProperty = nameof(OrmDataSources))]
public string RecordCollection { get; set; }
```

Einzelauswahl einer Rolle, ohne Jeder und System User.

```
[ComboboxControl]
[CDPRolesProviderProperties(filterEveryOne: true, filterSystemUser: true)]
public string EditorRole
```

Maskenaktualisierung

Mit dem Attribut `[AutoSubmit]` wird bei Auswahl eines neuen Wertes, die Maske automatisch aktualisiert, beispielsweise um weitere Eigenschaften ein- oder auszublenden.

18.3.1. Datenprovider Modifiers

Datenprovider benötigen in vielen Fällen Parameter, diese werden im Attribut hinterlegt. In eigen Fälle stehen diese Parameter zum Zeitpunkt der Programmierung nicht zur Verfügung, da sie selbst konfiguriert werden. Beispielsweise benötigt die Feldauswahl die ID der Datentabellenkonfiguration. In Masken und Ansichten liegt die zugehörige Datentabelle als Konfigurationwert vor und kann daher am Datenprovider nicht gesetzt werden.

Um Dynamisch die Datenprovider mit Parameter zu Versorgen stehen sogenannte Modifiers zur Verfügung. Diese sind im Packet `BA.Core.CustomAttributes` abgelegt und haben den Präfix `CDP`. Der wichtigstes Modifier ist dabei der `CDPDataSourceModifier`, dieser ermittelt dynamisch die konfigurierte(n) Datentabelle(n) und versucht ebenso dynamisch diese dem Datenprovider zuzuweisen. Daher kann dieser Modifier ist vielen Kombinationen Konfiguration / Datenprovider genutzt werden.

```
[CDPOrmFieldsProperties(TypesToShow = new[] { typeof(OrmTextField) })]
[CDPDataSourceModifier]
[ComboboxControl]
public String OrmFieldName { get; set; }
```

Generischer Modifier

Der Modifier `CDPModifier` erlaubt es eine Methode im Steuerelement zu implementieren, welche dann den Datenprovider zur Laufzeit anpassen kann.

```
[CDPOrmFieldsProperties(TypesToShow = new[] { typeof(OrmTextField) })]
[CDPModifier(ModifierMethod = nameof(ModifyDataProviderProperties))]
[ComboboxControl]
public String OrmFieldName { get; set; }
```

Die Methode muss eine entsprechende Signatur haben. Keinen Rückgabewert und drei Parameter `ConfigurationBase configuration`, `string propertyName` und `ControlDataProviderPropertiesBase properties`.

```
public virtual void ModifyDataProviderProperties(ConfigurationBase configurati
on, string propertyName, ControlDataProviderPropertiesBase properties)
{
    if (properties is CDPOrmFieldsProperties prop)
    {
        if (configuration != null && configuration != null && propertyName ==
nameof(OrmFieldName))
        {
            GridViewConfiguration gridConfig = (GridViewConfiguration)configur
ation;
            prop.DataSources = GuidSet.Parse(gridConfig.GetDataSource());
```

```
        }
    }
}
```

Eigenen Modifier implementieren

Um einen eigenen Modifier zu implementieren wird eine Klasse erstellt, die die Basisklasse `CDPModifierAttributeBase` erweitert. Dazu muss eine Funktion implementiert werden in der man Zugriff auf die Konfiguration, auf das Steuerelement, auf den Namen der Eigenschaft an der der Datenprovider sitzt und auf die Eigenschaftenklasse des datenprovider selbst, um diesen zu modifizieren.

```
public void ModifyCDPProperties(ConfigurationBase configuration, object control, string propertyName, ControlDataProviderPropertiesBase properties)
{
   ...
}
```

18.3.2. Datenprovider implementieren

Falls kein Datenprovider vorhanden ist, der die Auswahl bietet, kann man einen eigenen Datenprovider implementieren. Dafür sind zwei Klassen notwendig, zum einen der Datenprovider selbst und zum anderen seine Eigenschaften.

Eigenschaften des Datenproviders

Die Eigenschaften

- Die Klasse muss die Basis Klasse `ControlDataProviderPropertiesBase` erweitern.
- Die Eigenschaften sollten prinzipiell simple Datentypen sein. Komplexe Datentypen sollten dringend vermieden werden.
  Beispielsweise sollte die Guid einer Konfiguration, anstatt die Konfiguration selbst, definiert werden.
- Sinnvoll ist es mit `Guid`, `GuidSet`, etc. zu arbeiten, auch wenn diese nicht in den Attributen angeben werden können. Die Umwandlung, sollte dann direkt im Konstruktor erfolgen.
- **WICHTIG:** Der vollständige Klassenname des eigentlichen Datenproviders muss in die Eigenschaft DataProviderTypeFullName gesetzt werden.

Beispiel

```
public class CDPEnumValuesProperties : ControlDataProviderPropertiesBase
{
    public Guid MasterGuid { get; set; }
    public bool DisplayIcon { get; set; } = true;
    public EnumEntryActiveState ActiveState { get; set; }
    public GuidSet WantedValues { get; set; }

    public CDPEnumValuesProperties(string masterGuid = null, bool displayIcon
= true, string activeState = null, string[] wantedValues = null) : base()
    {
        DataProviderTypeFullName = typeof(CDPEnumValues).FullName;

        if (!string.IsNullOrEmpty(masterGuid))
            MasterGuid = masterGuid.ToGuid();

        DisplayIcon = displayIcon;

        if (activeState != null || activeState.IsGuid())
            ActiveState = Api.Enum.GetEnumValue<EnumEntryActiveState>(activeSt
ate.ToGuid());

        WantedValues = GuidSet.Parse(wantedValues);
    }
}
```

Der Datenprovider

- Er muss das Interface `IControlDataProvider` implementieren.
- In `Properties` stehen die Eigenschaften zur Verfügung.
- Die Methode `GetRows()` liefert das Ergebnis zurück.
  - Es ist sinnvoll Datenprovider so zu implementieren, dass diese selbst erweitert werden können. In diesem Beispiel wurde eine virtuelle Methode `AddWhere()` implementiert. Diese kann überschrieben werden, und dadurch können weitere Filter implementiert werden.
  - Die Parameter enthalten den Filter, der in der UI vom Anwender eingegeben wurde. Dieser Filter muss auf die Ergebnisliste angewendet werden. In diesem Beispiel in der Methode `AddFilter()`.
- Die Methode `GetByKey()` liefert genau ein Ergebnis oder null zurück. Diese sollte nicht unbedingt `GetRows()` aufrufen, auch wenn dies ohne Probleme möglich ist. Falls es einen effektiveren Weg gibt, sollte dieser gewählt werden.

```
public class CDPEnumValues : IControlDataProvider
{
    public ControlDataProviderPropertiesBase Properties { get; set; }

    public virtual IQueryable<ControlDataRow> GetRows(ControlDataProviderParam
eter parameter = null)
    {
        if (Properties != null && Properties is CDPEnumValuesProperties prop)
        {
            IEnumerable<ValueEnum> enumValues = AddWhere(Api.Enum.GetEnumValue
s(prop.MasterGuid));

            IEnumerable<ControlDataRow> values = enumValues.Select(ff => new C
ontrolDataRow
            {
                Key = ff.GuidString,
                Title = ff.Translation_Guid.ToString().Translate(),
                DataItem = ff,
            });

            if (parameter != null && !string.IsNullOrWhiteSpace(parameter.Filt
er))
                values = AddFilter(values, parameter.Filter);

            return values.AsQueryable();
        }

        if (Properties == null)
            throw new ArgumentNullException("Data provider: The properties ar
e NULL");

        throw new ArgumentException(String.Format("Data provider: The properti
```

```
es are {0} and not {1}", Properties.GetType().FullName, typeof(CDPEnumValuesPr
operties).FullName));
    }

    public virtual ControlDataRow GetByKey(string key)
    {
        if (key != null && key.IsGuid())
        {
            ValueEnum value = Api.Enum.GetEnumValue(key.ToGuid());
            if (value != null)
                return new ControlDataRow
                {
                    Key = value.GuidString,
                    Title = value.Translation_Guid.ToString().Translate(),
                    DataItem = value,
                };
        }

        return null;
    }

    public virtual IEnumerable<ValueEnum> AddWhere(IEnumerable<ValueEnum> lis
t)
    {
        if (Properties is CDPEnumValuesProperties prop)
        {
            return list.Where(ff => prop.WantedValues != null && prop.WantedVa
lues.Contains(ff.ValueGuid) && (prop.ActiveState == null || prop.ActiveState
== EnumEntryActiveState.All || ff.IsActive == (prop.ActiveState == EnumEntryAc
tiveState.Active)));
        }

        return list;
    }

    public virtual IEnumerable<ControlDataRow> AddFilter(IEnumerable<ControlDa
taRow> list, string filter)
    {
        string conditionFilterPart = filter.Replace("\"", "").ToLower();
        return list.Where(ff => ff.Title != null && ff.Title.ToLower().Contain
s(conditionFilterPart));
    }
}
```

18.4. Eigenschaften modifizieren

Wenn Sie vorhandene Steuerelemente erweitern, können Sie Eigenschaften des Basissteuerelements modifizieren. Beispielsweise könnten sie eine Eigenschaft ausblenden, wenn diese bei der Erweiterung festgesetzt werden soll.

Dafür müssen Sie den Attributen ein `AttributeEntry` hinzufügen

```
Attributes.Add(new AttributeEntry() { PropertyName = nameof(BasisProperty), At
tribute = AttributeEnum.Skip });
```

Folgende Möglichkeiten existieren

- `ReadOnly` Lesemodus
- `Skip` Eigenschaft wird nicht dargestellt
- `Autosubmit` Automatische Aktualisierung
- `DisplayName` Label ändern
- `HelpText` Hilfetext ändern

Für `DisplayName` und `HelpText`, muss ein Wert gesetzt werden. Dazu wird `AttributeEntrySingl eValue` verwendet.

```
Attributes.Add(new AttributeEntrySingleValue() { PropertyName = = nameof(Basis
Property), Attribute = AttributeEnum.DisplayName, Value = "[INSERT TRANSLATIO
N GUID]" });
```

19. Sonstiges

19.1. Dependency Injection

Mit Hilfe von Dependency Injection können in Modulen Funktionalitäten aus BA oder anderen Modulen modifiziert bzw. angepasst werden.

In dem Root Verzeichnis der Vorlage befindet sich eine Datei “DIInit.cs”. Dort können verschiedene Funktionalitäten (Beispielsweise: Registrieren von Datentabellen) definiert werden.

```
[assembly: UseForDI(AssemblyType.Project)]
namespace BA.Training
{
    public class DIInit : NinjectModule
    {
        public override void Load()
        { ... }
    }
}
```

19.2. Assembly Typen

Für die Strukturierung der Assemblies in BA gibt es den sogenannten `AssemblyType`. Dieser legt fest in welcher Reihenfolge die Assemblies geladen werden und welche Implementierungen überladen können.

Innerhalb seines VS Projektes muss der Typ an zwei Stellen definiert sein. Zum einen in der AssemblyInfo.cs

```
[assembly: AssemblyFramework(BA.Core.AssemblyType.Project)]
```

Und als zweites in der Dependency Injection Klasse

```
[assembly: UseForDI(AssemblyType.Project)]
```

In einem BA Modul wird eine Übersetzung definiert, die in einem Projekt verändert werden soll. Dazu muss in der Translation.csv des Projektes lediglich, die Übersetzung mit der identischen Guid nochmals hinterlegt werden. Nun kann man dort neue Texte in den jeweiligen Sprachen hinterlegen. Um zu entscheiden, welche Übersetzung welche ändert, wird der `AssemblyType` herangezogen. In der folgenden Liste werden die Typen und deren Zweck beschrieben.

- `Core` Die Basisanwendung. (Nur BA Intern)
- `CoreExtension` Interne Core Erweiterungen (Nur BA Intern)
- `Module` Core Module wie BA.Contact
- `ModuleExtension` Erweiterungen der Core Module, wie BA.CRM.Contact
- `Product` Diese Ebene bundelt die einzelnen Module zu einem Produkt und sollte in der Regel keine eigenen Programmierungen enthalten. Hier ist es sinnvoll beispielsweise die Konfiguration abzulegen. Auch alle Voraussetzungen zu schaffen, dass die Konfiguration vollständig arbeitet, beispielsweise die konfigurierten Rollen anzulegen.
- `ProductExtension` An dieser Stelle könnte man Erweiterungen des Produktes implementieren.
- `Project` Ein Projekt implementiert an dieser Stelle die notwendigen Erweiterungen. Man entscheidet an dieser Stelle auf welchen Teilen das Projekt basiert. Beispielsweise kann man auf das Produkt BA.CRM inkl. der Konfiguration und Rollen basieren oder man basiert auf einzelne Module, wobei deren Abhängigkeiten zu beachten sind. Falls man beispielsweise nicht auf dem Produkt BA.CRM basiert, kann man entscheiden ob Module wie der Designer, Businessmail oder Report eingebunden werden.
  Die VS Projektvorlage ist auf diese Ebene eingestellt.
- `ProjectExtension` Diese Ebene kann genutzt werden, um Erweiterungen auf einem bestehenden Projekt zu implementieren.

Abhängigkeiten der Assemblies dürfen nur auf gleicher Ebene oder auf mit niedrigeren vorhanden sein. Beispielsweise ist BA.CRM abhängig von BA.CRMContact, dieses ist Abhängig von BA.Contact und dieses von BA.Core.

19.3. Konfigurationen ausliefern

Basieren Programmierungen auf Konfigurationen oder möchte man eine Konfiguration ausliefern, kann diese in das Modul eingebunden werden. Dabei wird sich in jeder Installation gemerkt, welche Konfiguration schon importiert wurde (Siehe SQL Tabelle `OrmImportedConfiguration`). Dadurch wird jede Konfiguration nur einmalig eingespielt. Als Identifier wird der Dateiname der Konfiguration verwendet.

> Daher ist es sinnvoll die Dateinamen zu nummerieren

Dafür legen Sie in dem Ordner “Content” einen Unterordner “Configuration” an und legen dort die ZIP Datei der exportierten Konfiguration ab. Stellen Sie den Buildvorgang der ZIP Datei auf “Eingebettete Ressource”.

> Sinnvoll ist eine eigene Installation, die ausschließlich zur Konfiguration verwendet wird. Die Konfiguration dieser Installation wird exportiert, in das eigene Modul eingebunden und dann in einer Testumgebungen überprüft. Dies sollte man tun, um zu vermeiden, dass Testkonfigurationen in den eigenen Standard gelangen.

19.4. Dynamische Konfigurationen

Konfigurationen werden in der Regel im Benutzerinterface erstellt und bei Bedarf exportiert und in das Projekt eingebunden. Als weitere Möglichkeit können Konfigurationen programmatisch generiert werden. Diese werden zur Laufzeit erstellt und nur in den internen Konfigurationscache abgelegt. Damit sind sie verwendbar können aber im Benutzerinterface nicht geändert werden. Der Vorteil dabei ist, das diese Konfigurationen unter der Kontrolle des Entwicklers sind und beliebig programmatisch geändert werden können.

Beispielsweise kann man eine Ansicht für einen Dialog definieren. Das Beispiel kann in beliebigen Stellen im Code ausgeführt werden. Es muss nur sichergestellt werden, das es vor dem nutzen selbiger ausgeführt wird.

```
Guid GridId = "609C5026-F459-40B6-9878-EA6534DF66BA".ToGuid();
if (!Api.Config.ContainsKey(GridId))
{
    // Ansichtenkonfiguration erstellen
    GridViewConfiguration engineGrid = new GridViewConfiguration()
    {
        Id = GridId,
        ConfigurationName = "MyGridInDialog",
        DynamicallyGenerated = true,
        ShowSearchPanel = false,
        ShowFilterRowForCollumns = false,
    };
    // Initialisieren
    engineGrid.Initialize();
    // Spalten hinzufügen (Hier eine Textspalte)
    engineGrid.FloatingColumns.Columns.Add(new TextBoxColumn()
    {
        OrmFieldName = nameof(OrmMyDataTable.EntityTitle),
        Caption = "Name"
    });
    // In den Konfigurationschache legen
    Api.Config.RefreshSingleConfigurationCache(engineGrid);
}
```

Häufig ist es sinnvoll die Konfigurationen zentral zu erstellen, damit sie beim Start der Maschine erstellt werden. Beispielweise dann, wenn diese Konfigurationen in anderen Konfigurationen genutzt werden sollen. Dazu wird eine Klasse erstellt, die das Interface `ICreateDynamicConfiguration` implementiert

```
namespace BA.Training.Configuration
{
    /// <summary>
```

```
    /// Klasse zum erstellen von dynamischen Konfigurationen
    /// </summary>
    public class CreateDynamicConfigurationsForTraining : ICreateDynamicConfig
uration
    {
        public void CreateConfigurations()
        {
            ...
        }
    }
}
```

19.5. Icons

Icons werden in BA über einen Namen referenziert. Als Basis dient die Icons I-Collection von IconExperience. Um ein Icon zu finden kann man die Suchseite aufrufen. Deren Namen kann man dann beispielsweise in Aktionen verwenden.

Die Icons liegen im SVG Format vor und werden von BA automatisch in PNGs der gewünschten Größe und Farbe konvertiert und zur Verfügung gestellt.

Möchte man bei der Implementierung von Elementen selbst ein Icon verwenden, so kann man über eine Hilfsmethode die URL abrufen und einbinden.

```
string iconUrl = IconHelper.GetIconUrl("information", RequestIconSizeEnum.Size
16x16);
```

Möchte man eigene Icons einbinden legt man eine “icons.zip” Datei im “Content” Ordner an. Diese muss als Eingebette Resource eingebunden werden. in dieser Datei legt man seine SVG Icons ab. Da die Referenzierung über den Namen erfolgt, sollte man ein entsprechendes Präfix verwenden.

19.6. Hintergrund beim Anmelden

Neben der Möglichkeit das Hintergrundbild der Anmeldeseite in einer installierten Umgebung auszutauschen, kann man dies auch in dem Projekt tun. Legen Sie in dem “Content” Ordner eine Datei namens “loginscreen.jpg” ab und stellen Sie den Buildvorgang auf “Eingebettete Ressource”.

19.7. Event beim Anwendungsstart

Beim Anwendungsstart ist es möglich eigene Funktionalitäten auszuführen. Typischerweise überprüft man an dieser Stelle, ob notwendige Rollen vorhanden sind und legt diese an, falls nicht.

Dazu implementiert man eine Klasse und erweitert `InitializationEventBase`. In der Methode `CreateConfigurationOrms()` kann man prüfen, ob notwendige Datensätze vorhanden sind, und diese wenn notwendig erstellen. Dies wird von BA.CRM genutzt, um Rollen anzulegen, die in der Konfiguration verwendet werden. Für diese Funktionalität gibt es eine entsprechende Hilfsmethode `EnsureRole(...)`.

```
public sealed class CreateRoles : InitializationEventBase
{
    public const string MyRoleGuid = "[INSERT THE ROLE GUID]";

    public override void CreateConfigurationOrms()
    {
        using (UnitOfWork uow = Api.ORM.GetNewUnitOfWork())
        {
            EnsureRole(uow, MyRoleGuid, "Meine Rolle", "Beschreibung meiner Ro
lle");
            uow.CommitChanges();
        }
    }
}
```

19.8. Audit Events

Werden Funktionalitäten benötigt, die beim Login Prozess des Anwenders aufgerufen werden, muss man die Klasse `AuditEventBase` erweitern und das Attribut `[AuditEvent]` setzen.

Folgende Exents stehe zur Verfügung

- `OnBeforeLogin`
- `OnLogin`
- `OnLoginSuccessful`
- `OnLoginFailed`

```
[AuditEvent]
public class MyLoginEvent : AuditEventBase
{
    public override void OnLoginSuccessful(CrmUser crmUser)
    {
        throw new Exception("This is a demo. Please implement here something u
seful or remove this class.");
    }
}
```

19.9. Anwendungs- und Benutzereinstellungen

Zugriff

In `Api.Config` befinden sich einige Methoden, um auf Konfigurationen zuzugreifen, auch auf die Anwendungseinstellungen `Api.Config.ApplicationConfiguration()` oder die des aktuellen Anwenders `Api.Config.CurrentUserConfiguration()`. Die allgemeinen Einstellungen befinden sich in der Eigenschaft `Basic`.

Beispiel: Zugriff auf das Vorgabeland in den Anwendungseinstellungen und in den Benutzereinstellungen.

```
EnumCountries country = Api.Config.ApplicationConfiguration().Basic.DefaultCou
ntry;
EnumCountries countryUser = Api.Config.CurrentUserConfiguration().Basic.Defaul
tCountry;
```

Weitere Einstellungen befinden sich in Unter-Konfigurationen. Diese können mit `GetSubConfiguratio n` ausgelesen werden.

```
ApplicationSubConfigurationCorrespondence correspondenceConfiguration = Api.Co
nfig.ApplicationConfiguration().GetSubConfiguration<ApplicationSubConfiguratio
nCorrespondence>();
```

Einstellungen erweitern

Es ist möglich die Einstellungen für die Anwendung und für die Benützer, um eigene Eigenschaften zu erweitern. Dabei hat man viele Freiheiten, seine Einstellungen zu positionieren.

Zuerst muss man eine neue Klasse auf Basis von `ControlBase` erstellen und das Attribut `[Applicati onConfigurationSubConfiguration]` setzen. Wie immer sollte man `[Serializable]` bei Steuerelementen nicht vergessen.

```
[Serializable]
[ApplicationConfigurationSubConfiguration]
public class ApplicationSubConfigurationCorrespondence : ControlBase { ... }
```

Aus den eigenen Eigenschaften werden die Maskenelemente automatisch generiert.

Zusätzlich wird ein Attribut benötigt, welches festlegt ob die Eigenschaft in den Anwendungs- oder / und Benutzereinstellungen definiert werden kann.

```
[ConfigurationType(ConfigurationTypeAttribute.ConfigurationType.Both)]
```

Mit Hilfe der Attribute `System.ComponentModel.DisplayName` und `Tab`, kann man steuern ob die eigenen Einstellungen

- Im allgemeinen Bereich
- In einer eigenen Gruppe
- Auf einem bestimmten Tab
- Auf einem eignen Tab
- Oder sogar in einer eigenen Tab Gruppe dargestellt werden.

Beispiel eigener Tab

```
[Serializable]
[Tab("[INSERT UNIQUE TAB ID]", "Meine Einstellungen")]
[ApplicationConfigurationSubConfiguration]
public class ApplicationSubConfigurationMySettings : ControlBase
{
    public ApplicationSubConfigurationMySettings() { }

    [ConfigurationType(ConfigurationTypeAttribute.ConfigurationType.User)]
    [System.ComponentModel.DisplayName("Ein Text")]
    public String MyValue { get; set; }
}
```

19.10. Eigene Konfigurationen

Konfigurationen sind Einstellungen, die in der Regel im Designer erstellt und bearbeitet werden. Gespeichert werden sie in der Tabelle `OrmConfiguration` in der Spalte `Configuration` als serialisiertes XML. Durch diese Serialisierung können beliebig Eigenschaften hinzugefügt und entfernt werden, ohne dass eine Migration notwendig ist. In manchen Fällen ist trotzdem eine Migration sinnvoll, dafür bietet der Mechanismus extra Möglichkeiten.

Es können eigene Konfigurationen mit eigenen Steuerelementen in der Toolbox erstellt werden. Dazu wird im ersten Schritt ein eigener Eintrag in der Auswahlliste `EnumConfigurationType` benötigt. Dort hat man als zusätzliche Parameter, die Möglichkeit einen Controller für die Bearbeitung zu definieren. In der Regel wird der Designer dafür verwendet und man kann dort `null` angeben. Auch für die Anzeige der Konfiguration kann ein eigener Controller angegeben werden. Beispielsweise wird der entsprechende Controller für Ansichten (“Grid”) oder für Masken (“Form”) angegeben. Eine solche Möglichkeit spielt aber für Projekte eine untergeordnete Möglichkeit.

Anschließend wird eine Klasse implementiert, die die Basiseinstellungen beinhaltet.

```
[Serializable]
[ConfigurationIdentifier(EnumConfigurationTypeExtension.MyConfigurationGuid, 1
000)]
public class MyConfiguration : TreelistDesignerConfigurationBase { ... }
```

Die Basisklasse `TreelistDesignerConfigurationBase` wird mindestens benötigt. Es können durch eine andere Basisklasse oder durch die Implementierung von Schnittstellen weitere Verhalten bestimmt werden.

- `TreelistDBConfigurationBase` Wird diese Klasse als Basisklasse verwendet, erhält man automatisch eine Auswahl einer Datentabelle und durch das implementieren der Methoden `GetListOfFieldsUsed` und `GetListOfRelationDefinitionsUsed` kann dafür gesorgt werden, das Felder und Relationsdefinitionen der Datentabelle nur einmal gewählt werden können. Dies ist Beispielsweise in den Masken der Fall, in denen man jedes Feld nur einmal in der Maske konfigurieren kann.
- `IDataConfiguration` Alternativ zu der Basisklasse `TreelistDBConfigurationBase` kann dieses Interface implementiert werden. In dem Fall muss man die Auswahl der Datentabelle selbst implementieren.
- `IDesignerConfiguration` Dieses Interface wird implementiert, wenn der Designer und damit eigene Steuerelemente genutzt werden.
- `IControlHolder` Sollte genutzt werden, wenn die eigene Konfiguration weitere Steuerelemente beinhaltet. Daher sollte dieses Interface auch implementiert werden, wenn `IDesignerConfiguration` implementiert wird.

Eigene Eigenschaften

Die eigenen Eigenschaften werden automatisiert als Eingabeelemente dargestellt. Dies ist im Abschnitt Generierte Masken beschrieben.

Anzeige auf der Initialisierungsmaske

Mit diesem Attribut wird gesteuert welche Eigenschaften auf der Initialisierungsmaske angezeigt wird.

```
[PropertyEditable(EditPropertyOn.InitForm)]
```

Spezieller Validator

- `[BARequiredConfigurationProperties(requiredOn)]` Bedingte Pflichtfeldvalidierung abhängig davon ob die Eigenschaft gerade auf der Initialisierungsmaske dargestellt wird.

Eigene Steuerelemente

Eine Konfiguration kann im Designer aus Steuerelementen zusammengesetzt werden, wie dies beispielsweise in Masken und Ansichten geschieht. Das Prinzip ist identisch. Um ein eigenes Steuerelement zu implementieren wird dem `Toolbox` Attribut die Guid der eigenen Konfiguration übergeben.

```
[Toolbox(EnumConfigurationTypeExtension.MyConfigurationGuid)]
```

19.11. E-Mails

E-Mail Adressen in Datentabellen einbinden

E-Mail-Adressen die als Adresse eines Datensatzes dienen (Beispiel: E-Mail-Adresse eines Kontaktes), werden als Teil-Datentabelle vom Typ `OrmSubOwnedEmailAddress` angelegt.

`OrmSubOwnedEmailAddress` hat die folgende Felder:

- `EmailAddressType` Dies ist ein Auswahllistenfeld vom Typ `EmailAddressType` und kann als eindeutiger Schlüssel verwendet werden.
- `EmailAddress` Textfeld mit der E-Mail-Adresse.

E-Mail-Adressen als Felder von zu versendenen Datentabellen (Beispiel: Empfänger der E-Mail), werden als Teil-Datentabelle vom Typ `OrmSubLinkedEmailAddress` angelegt.

`OrmSubLinkedEmailAddress` hat die folgende Felder:

- `EmailAddressOwnerId` Guid des Datensatzes, zu der die E-Mail-Adresse zugeordnet ist (generell, ParentGuid von `OrmSubOwnedEmailAddress`). Dieses Feld wird benötigt, um im Nachhinein noch den Empfänger (beispielsweise einen Kontakt) zu identifizieren.
- `EmailAddress` Text mit der E-Mail-Adresse.

> Die Klasse `SubEmailAddressesHelper` beinhaltet verschiedene Hilfsmethoden für Schreiben und Lesen dieser Teildaten.

Versenden von Datensätzen

Das Versenden von E-Mails ist in BA auf Basis beliebiger Datentabellen möglich. Dies geht konfigurativ über die Ribbon Bar Aktionen und auch programmatisch. Dazu gibt man die entsprechenden Felder des Datensatzes an, den man versenden möchte. In der Klasse `MailSend` gibt es eine statische Methode `SendOrm`

```
public static void SendOrm(OrmBABase ormSendable, string fieldNameFrom, string fieldNameTo, string fieldNameReplyTo, string fieldNameCC, string fieldNameBCC, string fieldNameSubject, string fieldNameBody, string fieldNamePriority)
```

Parameter

- `ormSendable` Der zu versendende Datensatz
- `fieldNameFrom` Der Feldname einer Teil-Datentabelle von Typ `OrmSubLinkedEmailAddress`. Es darf nur ein Teil-Datensatz vorhanden sein, der als Absender der E-Mail verwendet wird.
- `fieldNameTo` Der Feldname einer Teil-Datentabelle von Typ `OrmSubLinkedEmailAddress`. Die

Empfänger der E-Mail
- `fieldNameReplyTo` Der Feldname einer Teil-Datentabelle von Typ `OrmSubLinkedEmailAddress`. Die “Antworten-An”-Adressen der E-Mail.
- `fieldNameCC` Der Feldname einer Teil-Datentabelle von Typ `OrmSubLinkedEmailAddress`. Die Kopie-Empfänger der E-Mail
- `fieldNameBCC` Der Feldname einer Teil-Datentabelle von Typ `OrmSubLinkedEmailAddress`. Die Blind-Kopie-Empfänger der E-Mail
- `fieldNameSubject` Textfeld mit dem Betreff der E-Mail
- `fieldNameBody` Textfeld mit dem Inhalt der E-Mail. Dies kann sowohl reiner Text als auch HTML sein. Der HTML-Inhalt der E-Mail erhält die BA-Schriftart und -größe als Vorgabe.
- `fieldNamePriority` Ein Auswahllistenfeld vom Typ `EnumPriority`

Es werden alle Dateianhänge des Datensatzes mit versendet.

Pre- und Postsend Events

In dem System kann sowohl ein Event vor als auch nach dem Versenden ausgeführt werden. Dazu exitistieren zwei Interfaces `IPreSend` und `IPostSend`, die man per Dependency Injection definieren kann.

Da es jeweils nur einen Handler geben kann, muss man dabei darauf achten welche Basisklasse man nutzt, ansonsten werden Funktionalitäten abgeschaltet. Mindestens müssen Sie auf den Klassen `PreSendBase` und `PostSendBase` basieren.

Ist das Modul BA.Activity eingebunden, sollten die Klassen `PreSendHandler` und `PostSendHandler` aus dem Modul verwendet werden, da sonst das `ActivityDate` nicht korrekt gesetzt wird.

Ist das Modul BA.CRM.Activity eingebunden, müssen es die Klassen `CRMPreSendHandler` und `CRMPostSendHandler` sein, weil sonst die Relation zum `ActivityOwner` nicht korrekt gesetzt.

Versenden von E-Mails ohne Verwendung von Datensätzen

Das Versenden von Datensätzen erfolgt asynchron über einen eigenen Hintergrundprozess. Dieser sorgt dafür, dass bei dem Auftreten von Fehlern mehrfach versucht wird, die Mail zu versenden. Möchte man eine E-Mail versenden ohne einen Datensatz zu erzeugen, kann man dies direkt auf dem `ISMTPHelper`. In diesem Fall muss das gesamte Handling der Fehler selbst vorgenommen werden.

```
public virtual BASMTPMail CreateMail(String from, String to, String subject, String body, string cc = null, string bcc = null, string replyTo = null, EnumPriority priority = null, bool checkLimits = true)
```

Parameter

- `from` E-Mail Adresse des Absenders

- `to` Komma oder Semikolon separierte Liste mit E-Mail Adressen der Empfänger
- `subject` Der Betreff der E-Mail
- `body` Unformatierter Text oder HTML. Bilder müssen im HTML eingebettet sein. Der HTML Inhalt der E-Mail erhält die BA-Schriftart und -größe als Vorgabe.
- `cc` Komma oder Semikolon separierte Liste mit E-Mail Adressen der Kopie-Empfänger
- `bcc` Komma oder Semikolon separierte Liste mit E-Mail Adressen der Bliend-Kopie-Empfänger
- `replyTo` Komma oder Semikolon separierte Liste mit E-Mail Adressen der Antworten-An-Adressen
- `priority` Auswahllistenwert von@EnumPriority@
- `checkLimits` Sollen dieGrößen- und die Empfängerlimitationen aus den Anwendungseinstellungen geprüft werden.

```
SMTPHelperBase smtpHelper = (SMTPHelperBase)Activator.CreateInstance(DIKernelP
rovider.Kernel.Get(typeof(ISMTPHelper)).GetType());
smtpHelper.CreateMail("user@company.org", "test@test.de", "Testmail", "Mailinh
alt");
```

19.12. Ordner

Ordner in BA sind ein Datensatz der Datentabelle Folder und können damit wie alle Datentabellen entsprechend erweitert werden. Datensätze in Ordnern werden über eine interne Relation abgebildet.

Über eine eigene API `Api.RecordCollection` kann man Datensätze in Ordner schieben oder daraus entfernen und Ordner erstellen, abfragen oder löschen.

Erstellen

Es kann ein neuer Ordner pro Aufruf erstellt werden

```
public OrmRecordCollection CreateRecordCollection(string name, Guid sourceGrid
ConfigurationGuid, EnumRecordCollectionType type = null, Guid? privateOnFirstU
seRecordCollectionGuid = null)
```

Hinzufügen und Entfernen

Über die verschiedenen Methoden `AddRecordsToRecordCollection(...)` können Datensätze in einen Ordner eingefügt werden, und mit `RemoveRecordsFromRecordCollection(...)` können sie wieder entfernt werden. Dies kann sofort ausgeführt werden oder über einen Hintergrundprozess. Dies wird über `asTask` definiert.

Um Datensätze in Ordner zu schieben oder zu entfernen, wird nur deren Guid benötigt. Daher sollte man das Laden von Datensätzen möglichst vermeiden.

Auslesen

Mit den beiden Methoden `GetContentsOfRecordCollection(...)` kann man den Inhalt eines Ordners auslesen. Falls die Leserechte berücksichtigt werden sollen, muss dies angegeben werden.

Ordnerlisten

Mit den Methoden die mit `GetRecordCollection(s)By*` beginnen, können die Datensätze der Ordner geladen werden.

Mit `GetPublicRecordCollections(...)` bekommt man die öffentlichen und mit `GetVisibleRecordCollections*` die sichtbaren Ordner zurückgeliefert.

Rechte

Mit den Methoden `MayUser*` könnten die Rechte eines Benutzers auf Ordner überprüft werden.

Leeren

Mit `ClearRecordCollection(...)` leert man einen Ordner

Umbenennen

Mit `RenameRecordCollection(...)` benennt man einen Ordner um.

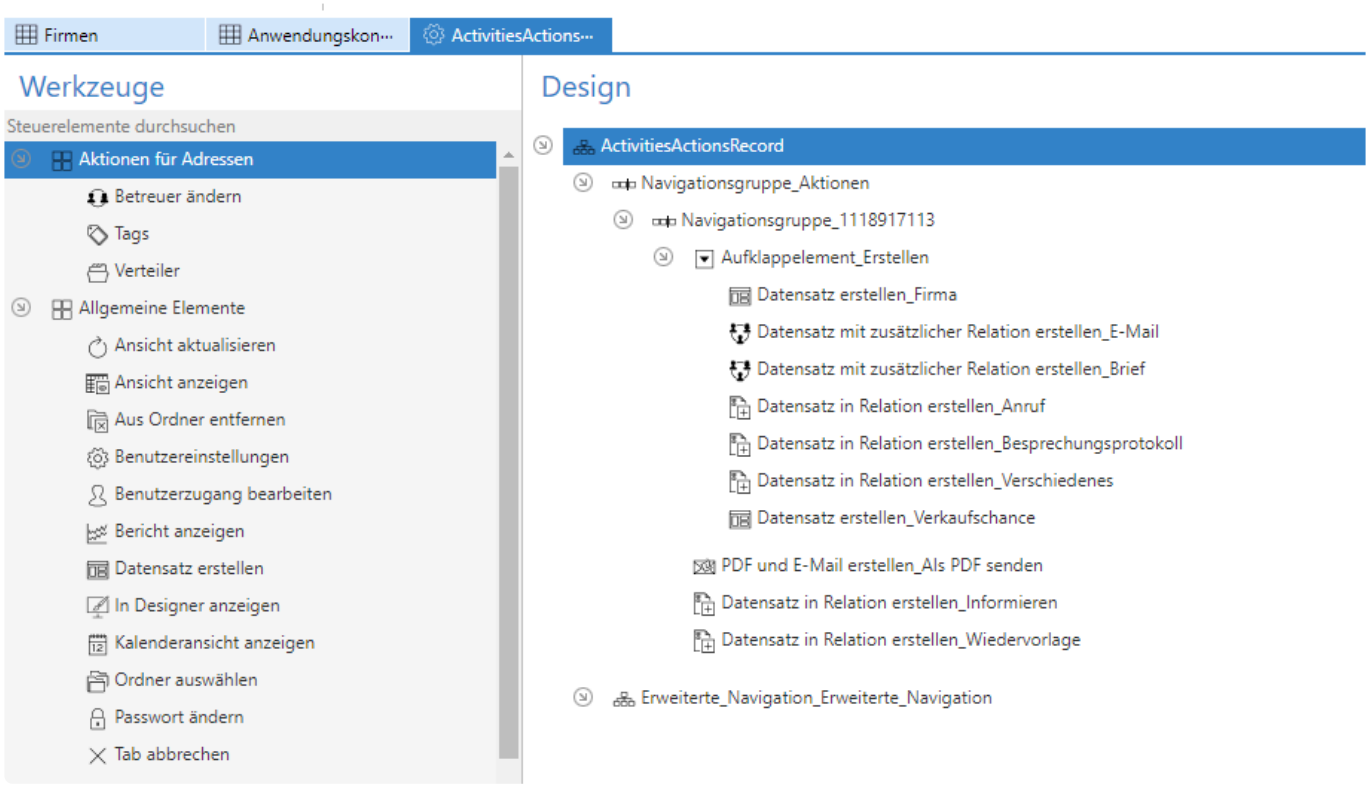
Löschen

Mit `DeleteRecordCollection(...)` löscht man einen Ordner

19.13. Designer Drag & Drop Regeln

Drag & Drop Regeln definieren, welche Steuerelemente sich im Designer wohin ziehen lassen.

Am Bespiel der Ribbon bar Navigation sieht man in der Toolbox Steuerelemente, die sich in den Desgin-Bereich ziehen lassen. Bei allen Navigationen (außer Anwendungsaktionen) ist z.B. erforderlich, dass sich alle allgemeinen Controls innerhalb einer doppelt geschachtelten Navigationsgruppe befinden.



Welche Regeln gelten für die Navigation?

1. Der Hauptknoten kann Navigationsgruppen und erweiterte Navigationsgruppen enthalten
2. Navigationsgruppen dürfen so auch wieder Navigationsgruppen enthalten
3. Innerhalb einer Navigationsgruppe dürfen alle möglichen Arten von Controls sein, aber nur wenn diese mindestens auf der 4. Ebene liegen
   => Zählweise ist hier:
   a. Ebene 1: Hauptebene
   b. Ebene 2: Navigationsgruppe
   c. Ebene 3: Navigationsgruppe
   d. Ebene 4: … alle möglichen anderen Controls
4. Eines dieser Controls aus Ebene 4. kann das Aufklappelement sein
   a. Dieses Element darf wieder alle möglichen anderen Controls haben, aber nicht wieder ein Aufklappelement
5. Außerdem gibt es noch die Erweiterte Navigation, diese darf nichts enthalten

Definition

Üblicherweise werden Regeln in einer eigenen Datei über Assembly Attribute definiert. In BA stehen die Regeln für die Navigation in der “NavigationDndRules”–Datei.

Assembly Attribute werden niemals in eine Klasse oder in einen Namespace geschrieben. Der Vorteil gegenüber normalen Attributen ist, dass sie zur Assembly (.dll) gehören und nicht zu einem bestimmten Datentyp. Daher können Projekte zu BA-Typen Regeln anlegen oder bestehende überschreiben.

Das hierfür verwendete Attribut ist `DnDRuleAttribute`, welches folgende Parameter hat:

```
public DnDRuleAttribute(Type targetType, Type sourceType, string configuration
Type, string configurationSubType = null, DndBehavior dndBehavior = DndBehavio
r.Allow, int minLevel = 0, int maxLevel = 0)
```

- `TargetType` definiert den Datentyp des Controls, auf das ein Control (Target) gezogen (gedraggt) wird. Das gedraggte Control wird dann Kind des Targets.
- `SourceType` definiert das gedraggte Control
- `ConfigurationType` definiert, für welchen Konfigurationstypen diese Regel erstellt wird. Die dort zu hinterlegenden Typen sind in `EnumConfigurationType` zu finden.
- `ConfigurationSubType` ist für Konfigurationen notwendig, bei denen unterschiedliche Regelwerke gebraucht werden. Aktuell ist das nur bei der `NavigationConfiguration` für die `ApplicationActions` benötigt werden.
- `DndBehavior` definiert, ob es sich um eine Erlauben- oder Verbieten-Regel handelt. Grundsätzlich ist alles erst einmal verboten, solange es keine Erlauben-Regel (`DnDBehavior.Allow`) gibt. Man kann jedoch z.B. auf eine Basisklasse oder ein Interface eine Erlauben-Regel anlegen und dann einen konkreten Typen wieder verbieten. Für “Verbieten” gibt es hier zwei Optionen:
  - `DenyCascade` lässt auch unter dem verbotenen Element keine weiteren Children zu,
  - `DenySource` verbietet nur genau das zu draggende Element. Falls dieses in der Zielstruktur aber schon vorhanden ist, kann es trotzdem noch die dazu vorgesehenen Children haben. Benötigt wurde `DenySource` bei den Datensatz-Validatoren, die zu festen, nicht veränderbaren Spalten hinterlegt sind.
- `MinLevel` und `MaxLevel` erlauben den `SourceType` nur auf bestimmten Ebenen (Zählweise siehe vorheriger Abschnitt)

Generell gilt:

Sowohl `SourceType` als auch `TargetType` können entweder direkte Datentypen des Controls, einer ihrer Basisdatentypen oder auch ein Interface sein. Die Kombination `SourceType` und `TargetType` ist pro `configurationType` / `configurationSubType` Kombination eindeutig. Sprich, wenn es mehrere Regeln dazu gibt, wird die zuletzt gelesene andere überschreiben. Es ist garantiert, dass Projekte gegen BA gewinnen.

Außerdem gewinnen Deny-Regeln gegen Allow-Regeln.

Für `MinLevel` und `MaxLevel` gilt eine weitere Besonderheit: Auch wenn die Definition an jeder Regel

möglich ist, kann es pro Steuerelement (in `SourceType`) immer nur einen Min- und einen Maxwert je `ConfigurationType` / `ConfigurationSubType` geben! `MinLevel` / `MaxLevel` berechnen sich nach folgender Regel:

- Es werden alle in Frage kommenden Regeln betrachtet.
- Wie beschrieben gewinnen bei gleichem Source/Target die Projekt-Regeln.
- Aus dem sich ergebenden Regelset werden nur die Source-Typen beachtet, Target ist hier nicht relevant.
- Sollte es zu einem Source-Typ mehrere Regeln geben, gewinnt die Letzte, sofern sie für min oder max etwas anderes als 0 definiert hat. Definiert sie -1 sagt sie damit, dass es egal ist, aber es ist definiert [wird also von vorhergehenden nicht überschrieben]. Min und Max können aus unterschiedlichen Regeln stammen.
- Daraus resultiert dann wieder ein neues Regelset mit nur noch einem Eintrag aus Min und Max pro `SourceType`.

Ein Steuerelement ist immer in einer Klasse, in beliebig vielen Basisklassen, sowie beliebig vielen Interfaces. Für jedes Steuerelement wird zunächst geschaut, ob es in dem soeben berechneten Regelset einen passenden Eintrag zu genau der Klasse gibt. Ist dies nicht so, wird in den Basisklassen in aufsteigender Reihenfolge und danach in den Interfaces gesucht. Hierbei wird der erste Minwert ungleich 0, sowie der erste Maxwert ungleich 0 verwendet.

Die eigentliche Definition sieht dann z.B. so aus, für Navigation

1. Hauptknoten mit Target null
   `[assembly: DnDRule(null, typeof(NavigationConfiguration), EnumConfigurationType.NavigationConfigurationGuid)]`
2. Darunter bis auf maximal Ebene 3 die Navigationsgruppe und einmal die Erweiterte Navigationsgruppe. Letztere erbt übrigens die Ebene, da `ExtendedNavigationGroupControl` von `NavigationGroupControl` erbt (ist hier aber nicht relevant)
   `[assembly: DnDRule(typeof(NavigationConfiguration), typeof(NavigationGroupControl), EnumConfigurationType.NavigationConfigurationGuid, minLevel: -1, maxLevel: 3)]`
   `[assembly: DnDRule(typeof(NavigationConfiguration), typeof(ExtendedNavigationGroupControl), EnumConfigurationType.NavigationConfigurationGuid)]`
3. `NavigationGroupControl` kann auch wieder ein `NavigationGroupControl` beinhalten (Ebenenbeschränkung von 2) bleibt beibehalten
   `[assembly: DnDRule(typeof(NavigationGroupControl), typeof(NavigationGroupControl), EnumConfigurationType.NavigationConfigurationGuid)]`
4. Diversen Controls über ihre Basisklassen auf Ebene erlauben, Kinder von `NavigationGroupControl` zu sein
   `[assembly: DnDRule(typeof(NavigationGroupControl), typeof(ServerActionBase), EnumConfigurationType.NavigationConfigurationGuid, minLevel: 4)]`
   `[assembly: DnDRule(typeof(NavigationGroupControl), typeof(ClientActionBase), EnumConfigurationType.NavigationConfigurationGuid, minLevel: 4)]`
   `[assembly: DnDRule(typeof(NavigationGroupControl), typeof(ToggleButtonBase), EnumConfigurationType.NavigationConfigurationGuid, minLevel: 4)]`

5. Verbieten, dass die `ExtendedNavigation` ein Kind von `Navigation` wird. Das wäre ansonsten erlaubt, da `ExtendedNavigation` von `Navigation` erbt und diese das dürfte
   ```
   [assembly: DnDRule(typeof(NavigationGroupControl), typeof(ExtendedNavigation
   GroupControl), EnumConfigurationType.NavigationConfigurationGuid, null, DndB
   ehavior.DenyCascade)]
   ```
6. Aufklappelement darf Kind von der Navigationsgruppe sein ab Ebene 4
   ```
   [assembly: DnDRule(typeof(NavigationGroupControl), typeof(DropDownAction), E
   numConfigurationType.NavigationConfigurationGuid, minLevel: 4)]
   ```
7. Alles, was wir eben erlaubt haben, muss nun für die `ExtendedConf` wieder verboten werden aufgrund der Vererbung
   ```
   [assembly: DnDRule(typeof(ExtendedNavigationGroupControl), typeof(Navigation
   GroupControl), EnumConfigurationType.NavigationConfigurationGuid, null, DndB
   ehavior.DenyCascade)]
   [assembly: DnDRule(typeof(ExtendedNavigationGroupControl), typeof(ServerActi
   onBase), EnumConfigurationType.NavigationConfigurationGuid, null, DndBehavio
   r.DenyCascade)]
   [assembly: DnDRule(typeof(ExtendedNavigationGroupControl), typeof(ClientActi
   onBase), EnumConfigurationType.NavigationConfigurationGuid, null, DndBehavio
   r.DenyCascade)]
   [assembly: DnDRule(typeof(ExtendedNavigationGroupControl), typeof(ToggleButt
   onBase), EnumConfigurationType.NavigationConfigurationGuid, null, DndBehavio
   r.DenyCascade)]
   [assembly: DnDRule(typeof(ExtendedNavigationGroupControl), typeof(DropDownAc
   tion), EnumConfigurationType.NavigationConfigurationGuid, null, DndBehavio
   r.DenyCascade)]
   ```
8. Das Aufklappelement darf die gleichen Elemente beinhalten, wie in 4) genannt
   ```
   [assembly: DnDRule(typeof(DropDownAction), typeof(ServerActionBase), EnumCon
   figurationType.NavigationConfigurationGuid)]
   [assembly: DnDRule(typeof(DropDownAction), typeof(ClientActionBase), EnumCon
   figurationType.NavigationConfigurationGuid)]
   [assembly: DnDRule(typeof(DropDownAction), typeof(ToggleButtonBase), EnumCon
   figurationType.NavigationConfigurationGuid)]
   ```

Bestehende modifizieren

Definiert man in einem Projekt eine Regel, die dasselbe Source und Target hat, wie die Regeln aus BA, kann man diese verändern, z.B aus einer Allow Rule eine Deny Rule machen oder die Ebenen begrenzen.

Es könnte aber auch vorkommen, dass ein Projekt die Regeln umfangreicher ändern will, ohne diese überschreiben zu müssen, also ggf. Regeln komplett entfernen.
Möchte ein Projekt so etwas tun, muss es das Interface `IDragAndDropRulesModifier` in einer beliebigen Klasse implementieren. Dort muss die Methode `ModifyDnDRules` implementiert werden.

```
void ModifyDnDRules(Guid configurationType, Guid? configurationSubType, Dictio
```

```
nary<DndRuleKey, DndRuleOrderedAttribute> dragAndDropRules);
```

Hier kann man das übergebene Dictionary frei anpassen. Dies sollte nur in Ausnahmefälle getan werden. In der Regel müssen die Attribute verwendet werden.

Das übergebene Dictionary enthält alle Regeln, die ohne dieses Interface aktiv würden, also inklusive derer, die im Projekt dazu gekommen sind und ohne die, die überschrieben wurden.

Fehlersuche

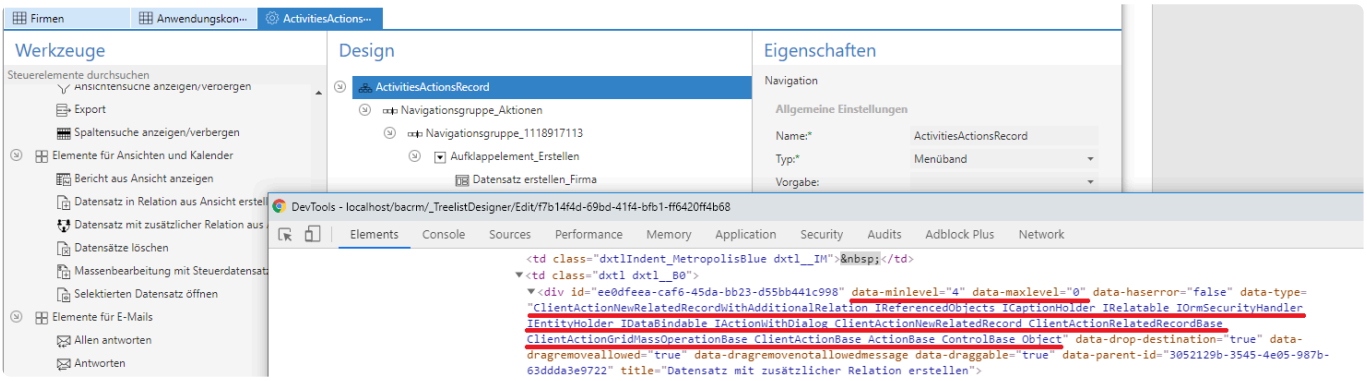
http://…/bacrm/bamaintain/CreateRules

Über den Aufruf kann man sich ausgeben lassen, mit welchen Regeln das System arbeitet. Diese Regeln wurden dazu bereits in JSON Format übersetzt, damit sie der Client verstehen kann. Es kann z.B. gut sein, dass man einfach eine Regel definiert hat, die nicht erreichbar ist, weil Zwischenknoten fehlen.

Es empfiehlt sich, den Code über ein Notpad++ Plugin oder eine Webseite, wie https://jsonformatter.curiousconcept.com/ hübsch formatieren zu lassen.

Untersuchen per Browser auf dem jeweiligen Control

Hier kann man erkennen, welche Klasse, Basisklassen und Intefaces ein Control benutzt, sowie welche Werte als erlaubte Verschachtelungstiefen errechnet wurden.



Ein Control ist nicht immer das Control, von dem man annimmt, dass es das ist

Dieses Beispiel kommt aus der `FormConfiguration`.

Angenommen es gibt folgende Regeln:

```
[assembly: DnDRule(typeof(GroupControl), typeof(TabContainerControl), EnumConf
igurationType.FormConfigurationGuid)]
[assembly: DnDRule(typeof(GroupControl), typeof(IDefaultControl), EnumConfigur
ationType.FormConfigurationGuid)]
```

Und erreicht werden soll die Verschachtelung Gruppe > Gruppe > Tab-Container

Dann ist Gruppe > Gruppe bereits möglich, da `GroupControl` auch `IDefaultControl` implementiert. Gruppe > Tab-Container ist ebenfalls möglich, da dies in der ersten Rule definiert wurde.
Gruppe > Gruppe > Tab-Container ist aber nicht möglich, weil nach den angegebenen Regeln die zweite Gruppe nicht als „Gruppe“ bekannt ist, sondern nur als `IDefaultControl`. Und da kein zulässiges Mapping von `IDefaultControl` nach `TabContainerControl` existiert, ist die innere Verschachtelung nicht zulässig.

Möchte man dies doch erreichen, benötigt man zusätzlich die Rule

```
[assembly: DnDRule(typeof(GroupControl), typeof(GroupControl), EnumConfigurati
onType.FormConfigurationGuid)]
```

Damit kann die zweite Gruppe auch eine Gruppe sein und nicht nur ein `IDefaultControl` und damit dann auch `TabContainer` beinhalten.

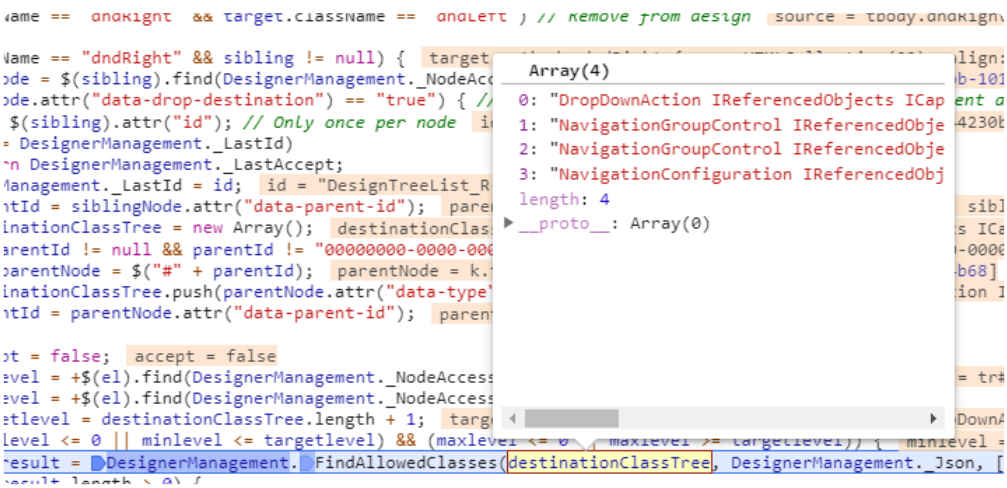
Grundsätzlich werden Rekursionen automatisch erkannt, und es wäre auch eine Verschachtelung von 100 Gruppen im Designer möglich … wie auch immer das dann in der UI (Form) aussieht.

Client debuggen

Hat alles nichts geholfen, kann man versuchen, im Browser beim Drag and Drop zu debuggen. Dazu im F12 den Aufruf der Methode `FindAllowedClasses` mit einem Breakpoint versehen.

Dann F12-Debugger wieder wegdrücken und mit dem Drag beginnen, bis man direkt über oder unter dem gewünschten Zielknoten ist. **Maus nicht los lassen**, während dem Dragvorgang erneut F12 drücken und die Maus dann auf den Zielknoten schieben.

Wenn bei Destination-Class-Tree an Position 0 das Control steht, dass man als Target hat, hat man die richtige Position erwischt und kann durchsteppen. Ansonsten wieder 2x F12 drücken und erneut versuchen.



19.14. Dubelettensuche

Die Dublettenprüfung soll verhindern, dass Benutzer gleiche bzw bereits existierende Datensätze innerhalb einer Business App Instanz mehrfach anlegen. Die Prüfung erfolgt wahlweise beim Speichern eines Datensatzes oder direkt bei der Eingabe von Daten über den Vergleich generierter Phonetikzahlenfolgen. Diese Zahlenfolgen werden generell bei jedem Speichern eines Datensatzes nach Bedarf aktualisiert bzw. initial beim Start der Anwendung erzeugt. Die eigentliche Prüfung erfolgt über eine „beginnt mit“ Suche auf diese phonetischen Zahlenfolgen.
Die Dublettenprüfung wird über sog. Adapterklassen gesteuert, die in Projekten erstellt oder auch überschrieben werden können. Die Funktionalität selbst ist stark UI-abhängig und als Hilfe für den Endanwender gedacht; sie soll nicht programmatisch aufgerufen werden.
Bei BA.CRM werden zwei vorgefertigte Adapter mitgeliefert (Firmen & Kontakte), welche im Folgenden als Beispiele herangezogen werden.

Dublettenprüfadapter

Zur Steuerung der Dublettenprüfung werden wie schon erwähnt „Adapter“ verwendet, die programmatisch erstellt werden müssen. Diese zu erstellenden Klassen müssen einen generischen Typen besitzen, der quasi die Basisklasse des zu überprüfenden Datentyps darstellt.

Im Falle des ausgelieferten Firmenadapters ist die Deklaration wie folgt:

```
public class CompanyAdapter<T> : DuplicateSearchAdapterBase<T> where T : OrmCR
MCompany
```

Das bedeutet: Streng typisiert verwendet dieser Adapter die Datentabelle `OrmCRMCompany` und funktioniert damit auch mit allen davon abgeleiteten Klassen (z. B. die interne -Custom-Klasse oder vom Konfigurator der Anwendung erstellte Ableitungen).

Die Adapterklasse muss außerdem mit Attributen dekoriert werden, welche der Prüfungsfunktionalität mitteilen, welche Datentabelle hier geprüft wird (`[DuplicateSearchMainDataSource]`), welche Datentabelle dazu gehören (`[DuplicateSearchSubDataSource]`), welche Felder geprüft werden sollen (`[DuplicateSearchFieldDefinition]` & `[DuplicateSearchSubRecordFieldDefiniti on]`) und ob und wenn welche bestehenden Relationen bei Änderungen geprüft werden sollen (`[Dupli cateSearchRelationDefinition]`).

Beschreibung der Attribute

DuplicateSearchMainDataSource

Dieses Attribut definiert, welche Datentabelle primär auf Dubletten geprüft wird. Es darf pro Projekt immer nur einen Adapter mit der Angabe einer bestimmten Datentabelle existieren. Die Angabe von zwei oder mehr Adaptern mit der gleichen Datentabelle führt zu undefiniertem Verhalten. Ebenso darf dieses Attribut pro Adapter nur einmalig verwendet werden.

Parameter:

- `dataSourceGuid` Guid der Datentabelle

DuplicateSearchSubDataSource

Über dieses Attribut werden weitere Datentabellen deklariert, welche mit der Datentabelle in irgendeiner Verbindung stehen und die in die Dublettenprüfung einbezogen werden sollen. Das Attribut kann an einem Adapter mehrfach angegeben werden.

Parameter:

- `dataSourceGuid` Guid der Datentabelle

DuplicateSearchFieldDefinition

Mit dem Attribut `DuplicateSearchFieldDefiniton` wird festgelegt, dass ein Feld dublettenrelevant ist und sich somit Änderungen auf die Anzahl der möglichen Dubletten auswirken.

Parameter:

- `dataSourceGuid` Guid der Datentabelle
- `fieldName` Name des Felds in der DB
- `isMandatoryForDuplicateSearch` Pflichtfeld, ohne dessen Inhalt eine Dublettensuche garnicht erst gestartet wird
- `indexPriority` Indexpriorität, beeinflusst die Reihenfolge der Spalten in einem Index in der DB. Kleinere Zahlen sind wichtiger, sprich Felder mit hoher selektivität oder solche, die am ehesten ausgefüllt werden, sollten kleinere Zahlen haben. Felder die IsMandatory gesetzt haben, sind im DB-Index aber immer ganz vorne und werden erst an zweiter Stelle nach Priorität sortiert
- `skipPhoneticTranslation` Das Feld wird trotz Dublettenrelevanz nicht phonetisch übersetzt. Sinnvoll bei Zahlenfeldern wie PLZ, da Zahlen in der Kölner Phonetic ignoriert werden
- `mandatoryGroupId` Damit die Suche nach Dubletten überhaupt gestartet wird, muss pro Gruppe, die hier vergeben wird, mindestens ein Feld ausgefüllt sein.
  Das Setzen von mandatoryGroupId setzt automatisch isMandatoryForDuplicateSearch auf true.

DuplicateSearchSubRecordFieldDefinition

Dieses Attribut wird speziell dafür verwendet, um Felder zu erfassen, die sich auf Teil-Datensätzen einer Datentabelle befinden. Es erbt von `DuplicateSearchFieldDefinition` und implementiert alle dessen Parameter, fügt aber noch weitere zur Identifikation eines bestimmten Teildatensatzes hinzu.

Parameter:

- `subtableFieldName` Name des Teil-Datensatzfeldes der Datentabelle (z. B. „Addresses“ auf Firmen und Kontakten in BA.CRM)
- `sortOrder` Der Teil-Datensatz mit dem angegebenen Index soll bei der Dublettenprüfung einbezogen werden. Die Angabe von `uniqueKey` überschreibt dies.
- `uniqueKey` Teil-Datensätze können auch über ihren eindeutigen Schlüssel identifiziert werden (z. B. der Anschriftentyp bei Anschriften in BA.CRM, um eine bestimmte Anschrift zu adressieren).

DuplicateSearchRelationDefinition

Hier wird definiert, welche Relation zwischen MainDataSource und SubDataSource verfolgt werden soll, im Beispiel-Szenario 1 betrifft das die primäre Anschrift. Es ist zu beachten, dass diese Relationen nur geprüft werden können, wenn sie in einem Dialog über dem Hauptdatensatz (MainDataSource) geöffnet werden, und nicht etwa im eigenen Tab. Das Attribut kann mehrfach vorkommen.

Parameter:

- `targetDataSourceGuid` Datentyp-Guid des Relationsziels
- `sourceDataSourceGuid` Datentyp-Guid der Relationsquelle
- `relationTypeGuid` Art der zu prüfenden Relation

Beschreibung der überschreibbaren Methoden

GetDuplicateSearchBaseQuery

Die Funktion `GetDuplicateSearchBaseQuery` enthält die Logik zur Ermittlung der zu selektierenden Daten.

Wird diese Methode nicht explizit überschrieben, so wird hier ein `IQueryable<T>` aus allen lesbaren Elementen des Typs `T` zurückgeliefert (der Typ `<T>` ist der generische Typ des Adapters).

Der ausgelieferte Kontaktadapter überschreibt diese Methode beispielsweise, weil Dubletten nicht über alle Kontakte sondern nur über alle Kontakte der gleichen Firma geprüft werden sollen:

```
public override IQueryable<T> GetDuplicateSearchBaseQuery(T mainRecord, Sessio
n session)
{
    if (mainRecord is OrmContact contact && contact.GetFirstFoundParentOrm<Orm
CRMCompany>() is OrmCRMCompany company)
        return Api.ORM.GetQueryWithReadPermissions<T>(session).Where(cont => c
ompany.RelatedContacts.Any(ff => ff == cont));

    return Api.ORM.GetQueryWithReadPermissions<T>(session);
}
```

GetCurrentRecordTitle

Mit `GetCurrentRecordTitle` wird implementiert, wie der Titel eines Datensatzes ermittelt wird. Dabei geht es in den jeweiligen Adaptern um die Live-Daten, die zu den gespeicherten Daten abweichen.

In der Basisimplementierung liefert diese Methode den `EntityTitle` des gerade editierten Hauptdatensatzes (z. B. Firma) zurück. Sollte dieser nicht ermittelbar sein, ist der Rückgabewert null.

Hier als Beispiel die Implementierung dieser Methode im ausgelieferten Firmenadapter:

```
public override string GetCurrentRecordTitle(T mainRecord, Guid recordId, Dupl
icateChangeInfo[] changeInfos)
{
    if (changeInfos != null && changeInfos.FirstOrDefault(ff => ff.ElementNam
e == "Name")?.Value is string name)
        return name;
eturn base.GetCurrentRecordTitle(mainRecord, recordId, changeInfos);
}
```

Das Ziel ist es hier, aus den Daten der Live-Prüfung die Änderungen am Feld „Name“ zu ermitteln. Sollte es keine Änderung an diesem Feld geben, wird die Basisimplementierung verwendet.

Verwendete Phonetik

Die Art und Weise, wie die für die Dublettenprüfung relevanten Daten in Phonetiken umgesetzt wird, ist in Projekten über Dependency-Injection anpassbar. Im Normalzustand wird die sog. „Kölner Phonetik“ verwendet.

Anmeldung der Phonetik:

```
Bind<IPhonetics>().To<DefaultPhonetics>();
Implementierung der Phonetik:
public class DefaultPhonetics : IPhonetics
{
   public string GetPhonetics(string input) => ColognePhonetics.GetPhonetics(i
nput);
}
```

Notwendige Anpassungen bei der Erstellung eigener Prüfungen

Es muss lediglich ein neuer Adapter implementiert werden. Beim nächsten Anwendungsstart werden automatisch die benötigten Spalten angelegt und die Berechnung der Phonetiken durchgeführt.

Wenn Dublettenprüfungen auf Datentabellen erstellt werden, die bereits in einer anderen Form auf Dubletten geprüft wurden (z.B. erst Anschrift in Kombination mit Kontakt, jetzt noch Anschrift in Kombination mit Firma), und sich dabei die dublettenrelevanten Felder unterscheiden, kann es hilfreich sein, alle zugehörigen Indizes auf die Phonetik-Spalten dieser Tabellen auf der Datenbank zu löschen. Diese werden nach einem optimierten Verfahren erneut angelegt.

Wenn ein Adapter wieder gelöscht wird, sollten die zugehörigen Phonetik-Spalten in den Tabellen manuell ebenfalls gelöscht werden.

Beispiel: Dublettenprüfung für Firmen

Grundsätzlich soll gelten:
Bei Firmen liegt möglicherweise eine Dublette vor,

- wenn sich der Firmenname stark ähnelt und
- wenn Straße und Postleitzahl der primären Anschrift ebenfalls phonetisch ähnlich sind.

Szenario 1 (alte Situation in BA.CRM):
Anschriften von Firmen liegen in getrennten Datensätzen, die über Relationen mit der Firma verknüpft sind. Es existiert eine Relation, um eine zur Firma gehörende Anschrift als Primäranschrift zu klassifizieren.

Die Attribute hierfür müssen wie folgt angegeben werden:

```
    [DuplicateSearchMainDataSource(EnumDataSourceExtension.CompanyGuid)]
    [DuplicateSearchSubDataSource(EnumDataSource.AddressGuid)]
    [DuplicateSearchFieldDefinition(EnumDataSourceExtension.CompanyGuid, "Nam
e", true)]
    [DuplicateSearchFieldDefinition(EnumDataSource.AddressGuid, "Address")]
    [DuplicateSearchFieldDefinition(EnumDataSource.AddressGuid, "PostalCode",
skipPhoneticTranslation: true)]
    [DuplicateSearchRelationDefinition(EnumDataSourceExtension.CompanyGuid, En
umDataSource.AddressGuid, EnumRelationType.PrimaryAddressGuid)]
```

Erläuterungen:

- `MainDataSource` Dieser Adapter prüft Datensätze des Typs „Firma“
- `SubDataSource` Es gibt zu prüfende Unterdatensätze des Type „Address“
- `FieldDefinition` Das Feld „Name“ der Firma muss geprüft werden, die ganze Prüfung braucht aber überhaupt nicht erst zu beginnen, so lange dieses Feld keinen Wert beinhaltet.
- `FieldDefinition` Des Weiteren gehört das Feld „Address“ auf Unterdatensätzen des Typs „Address“ zur Prüfung, genauso wie das Feld „PostalCode“. Für letzteres sollen keine Phonetiken angelegt werden, da Zahlen nicht abgebildet werden können.
- `RelationDefinition` Das Betrifft nur Anschriften, die mit der Firma über die Relation „PrimaryAddress“ verknüpft sind.

Szenario 2 (aktuelle Situation in BA.CRM):
Anschriften von Firmen liegen in Teil-Datensätzen vor, welche logisch Teil des Hauptdatensatzes sind. Die Primäranschrift wird durch Auswahl eines entsprechenden Wertes im Teil-Datensatz definiert.

Die Attribute hierfür müssen wie folgt angegeben werden:
bc. [DuplicateSearchMainDataSource(EnumDataSourceExtension.CompanyGuid)]
[DuplicateSearchFieldDefinition(EnumDataSourceExtension.CompanyGuid, “Name”, true)]
[DuplicateSearchSubRecordFieldDefinition(EnumDataSourceExtension.CompanyGuid, “Addresses”, “Address”, uniqueKey: EnumAddressTypes.MainAddressGuid)]
[DuplicateSearchSubRecordFieldDefinition(EnumDataSourceExtension.CompanyGuid, “Addresses”,

“PostalCode”, uniqueKey: EnumAddressTypes.MainAddressGuid, skipPhoneticTranslation: true)]

Erläuterungen:

- `MainDataSource` Dieser Adapter prüft Datensätze des Typs „Firma“
- `FieldDefinition` Das Feld „Name“ der Firma muss geprüft werden, die ganze Prüfung braucht aber überhaupt nicht erst zu beginnen, so lange dieses Feld keinen Wert beinhaltet.
- `SubRecordFieldDefinition` Des Weiteren gehören die Felder „Address“ und „PostalCode“ in Teildatensätzen des Properties „Addresses“ zur Prüfung. Der eindeutige Schlüssel des zu prüfenden Teildatensatzes muss „MainAddress“ lauten. Für das Feld „PostalCode“ sollen keine Phonetiken angelegt werden, da Zahlen nicht abgebildet werden können.

19.15. CSV-Konverter

Business App bietet neben dem bereits etablierten Datenimport über Importkonfigurationen die Möglichkeit, Daten aus CSV-Listen zu importieren, die nicht direkt mit dem Datenmodell von Business App übereinstimmen.

Für diese Funktionalität, den „CSV-Import“, wird durch einen Anwender eine einzelne CSV-Datei bereitgestellt, welche Informationen unterschiedlicher Teile des Datenmodells beinhalten darf.

Da aber die Importfunktion nicht neu erfunden und die bereits existierenden, umfangreichen Möglichkeiten des Datenimports wiederverwendet werden sollen, benötigen wir eine Schnittstelle zwischen der Importfunktion für diese einzelne Liste und dem bereits integrierten Datenimport.

Ein gutes Beispiel für eine entsprechende Anwendung ist eine Datei mit einer Liste von Kontakten, deren Anschriften und E-Mail-Adressen (dieses Beispiel wird in diesem Dokument immer wieder herangezogen, da diese Funktionalität mit ausgeliefert wird).

In einer minimalisierten Version einer Kontaktliste beinhaltet diese beispielsweise folgende Daten:

Vorname	Nachname	Straße	PLZ	Ort	E-Mail
Lars	Weis	Brückenmühle 93	36100	Petersberg	info@gedys-intraware.de

Der Standarddatenimport kann mit dieser Datei nichts anfangen, er ist zwar sehr mächtig, es müssen aber bestimmte Vorgaben erfüllt sein: So zum Beispiel muss es für den Standarddatenimport immer eine Spalte „MigrationID“ geben, und Teildaten wie Anschriften oder E-Mail-Adressen müssen in gesonderte Dateien ausgelagert werden.

CSV-Konverter dienen dazu, diese oben beschriebene „einfache“ Eingabedatei so in neue CSV-Dateien umzuwandeln, dass der Standarddatenimport diese verstehen und importieren kann.

Der Job des CSV-Konverters in diesem Fall wäre also, aus der einen Datei oben drei Dateien zu machen und diese in einem vorgegebenen Verzeichnis abzulegen:

001 OrmContact.csv

MigrationID	FirstName	LastName
CONT001	Lars	Weis

001 OrmContact.Addresses.csv

Parent	Address	PostalCode	City	AddressType
CONT001	Brückenmühle 93	36100	Petersberg	Main Address

001 OrmContact.EmailAddresses.csv

Parent	EmailAddress	EmailAddressType
CONT001	info@gedys-intraware.de	Primary

CSV-Konverter sind zweckgebunden und können jeweils nur Dateien für einen bestimmten Zieltyp erstellen (im obigen Beispiel wäre der Zieltyp der über die Dependency Injection festgelegte Typ für Kontakte, der Einfachheit halber verwenden wir hier `OrmContact`). Im Umkehrschluss heißt das, dass es in der aktuellen Ausbaustufe des Features nicht möglich ist, einen Konverter zu entwickeln, der eine Liste mit unterschiedlichen Datensatztypen bearbeiten kann (zum Beispiel Firmen und Kontakte in einer CSV-Datei).

Erweiterung per Attribut

CSV-Konverter können in Projekten per Attribut erweitert oder gar ausgetauscht werden. Wenn der mitgelieferte Konverter für Kontakte nicht den Projektanforderungen entspricht, so kann er durch einen eigenen CSV-Konverter ersetzt werden.

Die Deklaration hierzu lautet:

```
[ImportFileConverter(ConstantsContact.ImportFileConverter.Contact, "Meine Kont
akte")]
public class ProjectContactCSVConverter : ImportFileConverterBase { … }
```

Das Attribut `ImportFileConverter` nimmt zwei Parameter an, der erste ist die ID des Konverters, der zweite ein optional übersetzbarer Anzeigewert für die Konverterauswahl des CSV-Import Navigationssteuerelements und zur Anzeige und Auswahl im Dialog der Aktion.

Die gezeigte Konstante sollte verwendet werden, wenn der Konverter ausgetauscht werden soll, ansonsten ist die ID des Konverters aber ein frei wählbarer String. Es wird allerdings empfohlen hier entweder einen Guid-String oder einen String mit Namespaces zu verwenden um unbeabsichtigte Kollisionen zu vermeiden.

Die Klasse muss nicht zwingend von der Basisklasse `ImportFileConverterBase` ableiten, muss aber das Interface `IImportFileConverter` implementieren, so dass eine Erweiterung der Basisklasse einfacher ist.

Interface und Basisklasse

Dreh- und Angelpunkt der CSV-Konverter ist neben dem Attribut zur Deklaration eines solchen die Implementierung der Schnittstelle `IImportFileConverter`.

Folgende Eigenschaften müssen implementiert werden:

- `Id` Dient dem Abruf der über das Attribut angegebenen Konverter-ID.

- `DisplayName` Dient dem Abruf des über das Attribut angegebenen Anzeigenamen.

Folgende Methoden müssen implementiert werden:

```
void Convert(ImportFileConverterParameterModel parameters)
```

Hauptmethode, welche für die eigentliche Umwandlung zuständig ist.

```
string GetAdditionalInformationDialogId()
```

Liefert optional eine Dialog-ID zurück, welche in der UI nach der Anzeige des Spaltenzuordnungsdialogs aufgerufen und das Ergebnis an die Convert-Methode gegeben wird.

```
Dictionary<string, string> GetFieldMappings(FileInfo importFile)
```

Liefert optional eine vorgefertigte automatische Zuordnung der CSV-Spalten der gegebenen Datei zu Zielspalten/-eigenschaften auf dem Zieltyp des Konverters.

```
Type GetTargetType()
```

Liefert den Datensatztypen, der durch diesen Konverter behandelt wird (z.B. OrmContact). Dies **muss** ein Orm-Typ sein.

```
Type GetOptionalParentType()
```

Liefert optional den Datensatztypen, zu dem der Zieltyp beim Import zugeordnet werden kann (z.B. OrmCompany)

```
string[] GetFieldMappingDialogDescriptions()
```

Liefert optional eine Reihe von Erklärungen zur Funktion des CSV-Konverters (übersetzbar), die ganz oben im Spaltenzuordnungsdialog angezeigt werden.

Erwähnenswert ist hierbei auch das Übergabe-Model der Convert-Methode `ImportFileConverterParameterModel`, welches folgende Eigenschaften beinhaltet:

- `ImportTag` Ein Kennzeichen, welches den zu erstellenden Datensätzen in irgendeiner Art und Weise zugeordnet werden soll (z.B. `OrmContact` -> `AddressTags`).
- `ImportDirectory` Name des Hauptverzeichnisses des Importvorgangs. Dieser Name kann mit Hilfe der Methode `ImportJobConfiguration.GetJobDirectory()` in einen absoluten Pfad umgewandelt werden. Die zu importierende Datei liegt hierbei immer im Unterverzeichnis „ListImportData“ (`BA.Import.Constants.ListImportSubDirectory`).
- `FieldMappingJson` Spaltenzuordnungen, die vom Benutzer in der UI zugewiesen wurden.
- `AdditionalConverterParametersJson` Serialisierte Ergebnisdaten, welche von dem optionalen

„Additional-Information-Dialog“ zurück geliefert wurden.

Die Basisklasse `ImportFileConverterBase` ist abstrakt und implementiert zunächst alle Methoden und Eigenschaften des Interfaces, die Methoden `Convert`, `GetFieldMappings` und `GetTargetType` sind abstrakt und müssen in der eigentlichen Konverterklasse implementiert werden. Alle anderen Methoden sind `virtual` und damit überschreibbar, alle diese Methoden liefern `null` zurück bis auf `GetFieldMappingDialogDescriptions`, welches eine allgemeingültige Standarderklärung zurückgibt. Die Eigenschaften `Id` und `DisplayName` sind fest implementiert und greifen über den `TypeCache` auf das `ImportFileConverter` Attribut der Klasse zu, um die entsprechenden Daten auszulesen.

Ablauf eines Imports mit Konvertierung

Werfen wir zunächst einen Blick auf den Ablauf einer solchen Konvertierung im Kontext mit dem Importvorgang und der zugehörigen UI:

1. Der Hauptdialog, in welchem sich der Dateiupload für die CSV-Datei, die Auswahl des zu verwendenden Konverters und die Angabe eines Import-Tags befinden, wurde aufgerufen.
   Bei der Erstellung dieses Dialogs werden von allen verfügbaren Konvertern die optionalen Dialog-IDs für zusätzlich benötigte Parameter und Informationen abgefragt (`GetAdditionalInformationDialogId`) und der client-seitigen JavaScript-Verarbeitung zur Verfügung gestellt. Ein Click auf „OK“ in diesem Dialog führt zunächst dazu, dass der Dialog für die Spaltenzuordnungen aufgerufen wird.
2. Bei der Erstellung des Zuordnungsdialogs wird der gewählte CSV-Konverter über eine Hilfsfunktion abgerufen:
   `IImportFileConverter converter = ImportFileConverterTools.GetConverter(selectedConverterId);`
3. Von dem Konverter werden als erstes die vorgeschlagenen Spaltenzuordnungen (`GetFieldMappings`) und anschließend der Zieltyp (`GetTargetType`) abgerufen.
   Der Zieltyp MUSS hierbei ein Orm-Typ sein, da es sonst zu Fehlern in der weiteren Verarbeitung kommen wird.
4. Beim Aufbau des Dialogs werden neben den Mapping-Informationen die Beschreibungen (`GetFieldMappingDialogDescriptions`) und der optionale Elterntyp (`GetOptionalParentType`) herangezogen und in der UI verarbeitet.
5. Anschließend wird dieser Dialog angezeigt und harrt der Bedienung durch einen Benutzer. Die vom Konverter gelieferten Field-Mappings sind (sofern fehlerfrei möglich) in diesem Dialog voreingestellt.
6. Wird dieser Dialog mit „Ok“ bestätigt, wird anschließend geprüft, ob es für den gewählten Konverter einen Konverterdialog gibt. Ist dem so, wird dieser Dialog jetzt aufgerufen.
   Der Konverterdialog ist frei in all seinen Aktivitäten, sein Ergebnis wird beim Schließen des Dialogs in Json serialisiert und für die weitere Verarbeitung vorbereitet. Das Ergebnis des Dialogs muss dabei die `ButtonId` „okButton“ aufweisen, ist dies nicht der Fall, wird der ganze Vorgang abgebrochen.
7. Ist alles soweit in Ordnung, wird die Konvertierung angestoßen. Das Parameter-Model wird erzeugt und die methode „Convert“ des Konverters aufgerufen. Fehler innerhalb der Convert-Methode sollten mit Exceptions signalisiert werden. Die Verarbeitung geht davon aus, dass die Convert-Methode das mitgegebene Importverzeichnis final vorbereitet, d.h. dass die vom Standarddatenimport zu importierenden Dateien dort in einem Unterverzeichnis „Import“ liegen und nicht mehr (wie vermutlich in einem Zwischenschritt) in dem Verzeichnis, in welchem die CSV-Datei abgelegt ist („ListImportData“).

8. Nach der Konvertierung wird eine temporäre `ImportJobConfiguration` angelegt, welche dann zeitnah vom Datenimport ausgeführt wird. „Temporär“ hat hierbei zur Folge, dass
   - die Konfiguration nicht in der Liste der Importkonfigurationen angezeigt wird,
   - der Import unter den Rechten des aktuellen Benutzers stattfindet,
   - der Importvorgang in der Fortschrittsanzeige dargestellt wird und
   - die Importkonfiguration und alle im Importverzeichnis gespeicherten Daten nach Beendigung des Vorgangs wieder gelöscht werden.

Implementierung eines eigenen CSV-Konverters

Ich erläutere die Implementierung eines eigenen CSV-Konverters als Beispiel anhand des bereits mitgelieferten Konverters für Kontaktlisten. Ein paar Sachen werde ich auslassen (beispielsweise die Erstellung der Vorgabespaltenzuweisung, die doch einigermaßen kompliziert und umfangreich ist).

Beginnen wir mit dem Klassenrumpf:

```
[ImportFileConverter(ConstantsContact.ImportFileConverter.Contact, "1f62f2e2-c
1ba-43df-a91a-0d2ca7a5d19a")]
public class ContactImportFileConverter : ImportFileConverterBase { … }
```

So weit, so gut. Die Klasse heißt `ContactImportFileConverter`, sie befindet sich im Namespace (s.o.), ist über das Attribut als CSV-Konverter deklariert und erbt von der Basisklasse. Die Parameter des Attributs sind eine von uns festgelegte Guid als ID (erster Parameter) und eine Guid, die in den Text „Kontakte“ übersetzt wird (zweiter Parameter).

Schauen wir uns nun die kleinen Methoden an, die entweder virtuelle Basismethoden überschreiben oder abstrakte implementieren:

```
public override string[] GetFieldMappingDialogDescriptions()
{
    return new string[] { "fd9d5259-5744-479d-83c7-8e7f26597b4f", "06fa14a7-72
97-4e5c-9b52-25642a2b4f95", "ed5bcfee-74f0-4746-a021-7cf1fb7e484b" };
}
```

`GetFieldMappingDialogDescriptions` liefert einfach nur ein Array von Texten zurück, welche im Spaltenzuordnungsdialog ganz oben angezeigt werden sollen. Diese Texte können wie hier Guids von übersetzbaren Strings sein, oder feste Textbausteine.

```
private OrmDataSourceCacheModel TargetType = null;
public override Type GetTargetType()
{
    if (TargetType == null)
        TargetType = Api.ORM.GetOrmTypeCacheValue(EnumDataSourceExtension.Cont
actGuid);
    return TargetType;
```

```
}
```

Wir erklären, dass der Typ der Datensätze, die dieser Konverter erstellen wird, `OrmContact` ist. In unserem Fall wird der Typ noch im Konverter zwischengespeichert, weil in der Convert-Methode in einer Schleife darauf zugegriffen wird.

```
public override Type GetOptionalParentType()
{
    return Api.ORM.GetOrmTypeCacheValue(EnumDataSourceExtension.CompanyGuid);
}
```

Es wird deklariert, dass die von diesem Konverter erstellten Datensätze möglicherweise (!) durch eine Standardelternrelation mit Datensätzen des Typs `OrmCompany` verknüpft sein könnten.
Diese Verknüpfung muss in der Spaltenzuordnung zugewiesen sein, wenn nicht, findet sie auch nicht statt, d.h. vom Framework her ist eine Elternverknüpfung immer optional. Sollte das bei den von Ihnen erstellten Datensätzen nicht der Fall sein, so muss die Convert-Methode sicherstellen, dass es eine Elternverknüpfung gibt (und ggf. abbrechen, wenn dem nicht so ist).

```
public override Dictionary<string, string> GetFieldMappings(FileInfo importFil
e)
{
    Dictionary<string, string> result = new Dictionary<string, string>();
    …
    return result;
}
```

Wir werden hier nicht auf die tatsächliche Ermittlung der Feldzuordnungen eingehen, weil die in diesem Konverter doch sehr speziell und kompliziert ist. Grundsätzlich verläuft sie aber so, dass die übergebene Datei, also die Datei, die importiert werden soll, geöffnet und die Titelzeile ausgelesen wird. Für jeden Titel wird dann über ein internes Mapping geprüft, zu welcher Eigenschaft des Zieltyps dieser Titel passen könnte. Behalten Sie bitte auch im Hinterkopf, dass diese Spaltenzuordnung, die hier getroffen wird, nur der Vorschlag für den Benutzer ist, und diese Werte keinen anderen Zweck haben, als diese in der UI des Dialogs vorzubelegen.

Als Inhalte des zurückgegebenen Dictionarys werden erwartet:

- Als Schlüssel: der Titel aus der CSV-Datei
- Als Wert: die Zuordnung in ein Feld des Zieltyps

Das heißt also, in einem einfachen Fall wäre das zum Beispiel
„Vorname“ -> „FirstName“ oder „Familienname“ -> „LastName“

In einem etwas komplizierteren Fall muss das Ziel in einen Teildatensatz gemappt werden, so in etwa „Straße“ -> „Addresses.Address“, „Ort“ -> „Addresses.City“ oder „E-Mail“ -> „EmailAddresses.EmailAddress“.

Die korrekte Spaltenzuordnung für das Beispiel aus der Einführung sähe also so aus:

- „Vorname“ -> „FirstName“
- „Nachname“ -> „LastName“
- „Straße“ -> „Addresses.Address“
- „PLZ“ -> „Addresses.PostalCode“
- „Ort“ -> „Addresses.City“
- „E-Mail“ -> „EmailAddresses.EmailAddress“

> Die Spaltentitel „MigrationID“ und „Oid“ sollten übersprungen werden; sie stehen auch im Spaltenzuordnungsdialog nicht zur Verfügung. Der Versuch über diesen Mechanismus eine Aktualisierung von Daten durchzuführen birgt gewisse Risiken und wird von uns nicht empfohlen oder unterstützt.

Und dann wären wir auch schon bei der Convert-Methode. Die Convert-Methode des Kontaktkonverters besteht aus drei logischen Teilen, in die ich die Methode zur Erklärung aufteilen werde: das wären die Vorbereitung, die Verarbeitung und die Nachbereitung.

Die Vorbereitung

```
public override void Convert(ImportFileConverterParameterModel parameters)
{
    Dictionary<string, string> mappings = Api.JsonHelper.Deserialize<Dictionar
y<string, string>>(parameters.FieldMappingJson);
    string importPath = Path.Combine(ImportJobConfiguration.GetJobDirectory(pa
rameters.ImportDirectory), BA.Import.Constants.ListImportSubDirectory);
    DirectoryInfo di = new DirectoryInfo(importPath);
    FileInfo[] files = di.GetFiles();
    if (files == null || !files.Any())
        return;
    XPQuery<OrmCompany> parents = Api.ORM.GetDefaultSession().Query<OrmCompan
y>();
    ILookup<string, string> parentMapping = null;
    FileInfo file = files.First();
    …
```

Was passiert:

- Die vom Benutzer gewählten Feldzuordnungen werden aus den Parametern in ein `Dictionary<string, string>` deserialisiert. Dieses Dictionary hat das gleiche Format wie schon bei `GetFieldMappings()` erklärt.
- Der Pfad zu dem Verzeichnis mit der Import-Datei wird hergestellt. Der Parameter „ImportDirectory“ ist nur eine Guid, welche das Verzeichnis im Kontext des Importmechanismus beschreibt. Der zugehörige absolute Pfad auf der Festplatte des Servers muss daraus erst berechnet werden. Zusätzlich muss hier auf ein definiertes Unterverzeichnis zugegriffen werden.

- Die Inhalte des Verzeichnisses werden ausgelesen. Ist es leer oder nicht existent, wird die Verarbeitung beendet.
- Es wird eine Datenbank-Query auf die möglichen Elterndatensätze erstellt. Diese wird aber erst später ausgeführt, wenn es auch wirklich eine Elternverknüpfung gibt.
- Die Importdatei wird abgerufen.

Die Verarbeitung

```
using (ImportFileConverterCsvHelper csv = new ImportFileConverterCsvHelper(Get
TargetType(), mappings, file, 0))
{
    while (csv.Read())
    {
```

Zur Verarbeitung wird die Klasse `ImportFileConverterCsvHelper` herangezogen, welche die Verteilung der Daten auf mehrere Dateien stark vereinfacht. Die Klasse ist Disposable und wird daher in einem `using` benutzt. Als Parameter bekommt sie den Zieltypen, die Spaltenzuordnungen, die Eingabedatei selbst und ein numerisches Sortierkennzeichen.

Aus dem Zieltyp, den Spaltenzuordnungen und dem Sortierkennzeichen werden die Dateinamen der Zieldateien ermittelt und diese angelegt (in unserem Beispiel wären das also nun „00 OrmContact.csv“, „00 OrmContact.Addresses.csv“ und „00 OrmContact.EmailAddresses.csv“).

Als nächstes geht es mit einer while-Schleife so lange über die CSV-Eingabedatei, bis keine Daten mehr verfügbar sind.

```
foreach (string title in csv.ImportCsvTitles)
{
    if (title.Equals("oid", StringComparison.OrdinalIgnoreCase) || title.Equal
s("migrationid", StringComparison.OrdinalIgnoreCase))
        continue;
    string val = csv.GetSourceField(title);
    if (mappings.TryGetValue(title, out string csvTitle) && csvTitle.StartsWit
h($"{Import.Constants.ParentLinkIdentifier}."))
    {
        if (parentMapping == null)
        {
            string parentProperty = csvTitle.Substring(csvTitle.IndexOf('.')
+ 1);
            parentMapping = parents.ToLookup(ff => (string)ff.GetMemberValue(p
arentProperty), ff => ff.Oid.ToString());
        }
        if (parentMapping.Contains(val))
            csv.SetTargetField("$REL_Parent", parentMapping[val].FirstOrDefaul
t() ?? "");
        else
```

```
            csv.SetTargetField("$REL_Parent", "");
    }
    else
        csv.SetTargetField(title, val);
}
```

Pro gelesenem Datensatz werden nun die Inhalte verarbeitet. Hierfür wird in einer Schleife über jeden Titel der Eingabedatei gegangen und die beinhalteten Daten gelesen. Dann wird überprüft, ob die Spaltenzuordnung für diesen Titel eine Elternverknüpfung vorsieht. Ist dem so, wird nun die vorbereitete Datenbank-Query auf die Firmen ausgeführt und ein sog. Lookup auf den für die Elternverknüpfung konfigurierten Feldnamen erstellt (Schlüssel ist also beispielsweise der Firmenname, und der Wert des Lookups dann die Oid, die zu diesem Firmennamen gehört; ein Lookup deshalb, weil zumindest theoretisch ein Firmenname mehrfach in der Datenbank liegen könnte, in dem Fall wäre nicht definiert, zu welcher dieser Firmen der Kontakt zugeordnet würde). Mit diesen Informationen wird in der Zieldatei die Spalte „$REL_Parent“ befüllt, über welche dann beim eigentlichen Import die Elternverknüpfung generiert wird. Ist der aktuelle Titel nicht als Elternverknüpfung vorgesehen, werden die Daten einfach per Helper gesetzt, welcher dann auch das Mapping auf den Zielspaltennamen übernimmt.

```
 ImportFileConverterCsvRecord record = csv.GetCsvRecordBySubTable("EmailAddres
ses");
if (record != null && record.CurrentRecordIsDirty && record.Record is IDiction
ary<string, object> dyn1 && (!dyn1.ContainsKey("EmailAddressType") ||  strin
g.IsNullOrWhiteSpace(dyn1["EmailAddressType"] as string)))
    record.SetField("EmailAddressType", "Primary");
record = csv.GetCsvRecordBySubTable("Addresses");
if (record != null && record.CurrentRecordIsDirty && record.Record is IDiction
ary<string, object> dyn2 && (!dyn2.ContainsKey("AddressType") || string.IsNull
OrWhiteSpace(dyn2["AddressType"] as string)))
    record.SetField("AddressType", "Main address");
if (!string.IsNullOrWhiteSpace(parameters.ImportTag))
{
    record = csv.GetMainCsvRecord();
    if (record != null && record.CurrentRecordIsDirty && record.Record is IDic
tionary<string, object> dyn3)
    {
        string addressTags = (dyn3.ContainsKey("AddressTags") ? dyn3["AddressT
ags"] as string : null) ?? "";
        if (!string.IsNullOrWhiteSpace(addressTags))
            addressTags += ",";
        addressTags += parameters.ImportTag;
        record.SetField("AddressTags", addressTags);
    }
}
```

Sind alle Titel der Eingabedatei verarbeitet, wird nun geprüft, ob ein Datensatz für eine E-Mail-Adresse angelegt wurde. Ist das der Fall, wird diesem Datensatz zusätzlich noch der Adresstyp „Primary“

vergeben, sollte der Adresstyp nicht bereits durch den Import gesetzt worden sein. Das gleiche wird entsprechend für den Anschriftendatensatz gemacht und anschließend noch das gewünschte ImportTag an das Feld „AddressTags“ angehängt. Das alles wird nur dann gemacht, wenn der jeweilige Datensatz auch bis dahin schon gespeichert werden soll (dirty).

```
csv.NextRecord();
}
csv.SaveAll();
file.Delete();
}
```

Mit `csv.NextRecord()` werden die Daten intern abgelegt und für das finale Speichern zwischengespeichert und die Strukturen für den aktuellen Datensatz geleert. Der „Cursor“ rückt sozusagen in die nächste Zeile.

Sind alle Daten durch die while-Schleife verarbeitet, werden die csv-Dateien auf die Festplatte geschrieben (`csv.SaveAll()`) und die Eingabedatei gelöscht.

Die Nachbereitung

```
files = di.GetFiles();
if (files == null || !files.Any())
    return;
string importDir = Path.Combine(di.Parent.FullName, "Import");
di = Directory.CreateDirectory(importDir);
foreach (FileInfo fi in files)
    fi.MoveTo(Path.Combine(di.FullName, fi.Name));
}
```

Wie schon an anderer Stelle erwähnt, wird vom Konverter erwartet, dass er ein valides Importverzeichnis zurücklässt. Das heißt, alle Dateien, die nun noch im „ListImportData“ Verzeichnis liegen, müssen eigentlich Dateien für den Standarddatenimport sein. Diese werden nun noch nach nebenan ins Verzeichnis „Import“ verschoben.

Damit ist die Konvertierung beendet.

19.16. Logging (NLog)

Neben den Anwendungsprotokollen ist es sinnvoll auch Logeinträge zu erstellen, um bei einer Installation besser Fehler zu identifizieren. Dazu hat BA NLog in der Version 4 integriert.

19.17. Asynchrone Prozesse

Asynchrone Verarbeitungen sollten in der Regel durch Hintergrundprozesse durchgeführt werden. In manchen Situationen ist dies nicht praktikabel, dann kann ein Task gestartet werden. Beispielsweise werden ORM Events synchron ausgeführt. Dies bedeutet, dass die dortige Implementierung beispielsweise das Speichern eines Datensatzes verlangsamt. Es muss klar sein, das Berechnungen in einem Task vom Speichern eines Datensatzes und damit von seiner `UnitOfWork` abgekoppelt sind. Daher muss man genau überlegen ob dies wirklich sinnvoll ist.

Für die Implementierungen von Tasks die unter Benutzerrechten laufen stehen in der `Api.User` Methoden zur Verfügung.

```
Api.User.RunWithContext(() => { ... });
```

ORM Events werden auch ausgeführt, wenn der Datensatz im Backend verarbeitet wird. Führt man nun Funktionalitäten asynchron aus, und es werden viele Datensätze verarbeitet, wird der Server mit einer Vielzahl von asynchronen Tasks belastet. Daher gibt es eine weitere Möglichkeit die Abspaltung eines Tasks davon abhängig zu gestalten, ob der Vorgang im Vordergrund oder im Hintergrund ausgeführt wird. Bei Ausführen im Hintergrund wird der Code nun doch synchron ausgeführt und entlastet den Server bei Massenverarbeitungen, wie den Import.

```
Api.User.RunWithContextIfForeground(() => { ... });
```

20. Lösungen

20.1. Übung 1

Übersetzungen

```
"3d31e87a-1b67-4472-b53a-575bfe2e184b";"Druckmaschine";"Printing machin
e";"";"";"";"False";"0"
"5b9e094a-745d-47ea-93c7-a1fd5022374f";"Verpackungsmaschine";"Packing machin
e";"";"";"";"False";"0"
"64563a48-4bdf-4d6a-a952-7be9c90fef68";"Mechnische Presse";"Mechanical pres
s";"";"";"";"False";"0"
```

Auswahlliste

```
namespace BA.Training.Enums
{
    [EnumDefinition("Engine Types", false, false, true, true, true, true)]
    public class EnumEngineTypes : ValueEnum<EnumEngineTypes>
    {
        public const string Guid = "CC49C2BB-584D-47BF-98CA-7153AB7B7A92";

        public const string PrintingMachineGuid = "AFABDD8E-F233-4A2E-A72F-F65
D09193B80";
        public const string PackingMachineGuid = "7D924E88-393D-4CA3-BAAE-3C33
F991A2B5";
        public const string MechanicalPressGuid = "A93A551D-54AC-47D4-A374-D98
794081F16";

        public static readonly EnumEngineTypes PrintingMachine = new EnumEngin
eTypes(PrintingMachineGuid, 0, "3D31E87A-1B67-4472-B53A-575BFE2E184B", "Druckm
aschine für Zeitungen");
        public static readonly EnumEngineTypes PackingMachine = new EnumEngine
Types(PackingMachineGuid, 1, "5B9E094A-745D-47EA-93C7-A1FD5022374F", "Automati
sche Verpackung");
        public static readonly EnumEngineTypes MechanicalPress = new EnumEngin
eTypes(MechanicalPressGuid, 2, "64563A48-4BDF-4D6A-A952-7BE9C90FEF68", "Mechan
ische Müllpresse");

        [Browsable(false)]
        public string Description { get; set; }

        public EnumEngineTypes(String valueGuid, int sortOrder, String transla
tionGuid, String description) : base(valueGuid, sortOrder, translationGuid)
        {
            Description = description;
        }
```

```
        public EnumEngineTypes() { }
    }
}
```

20.2. Übung 2

Download als Visual Studio Template

20.3. Übung 3

Erweiterung der Übersetzungen

```
"1582c2b1-cef2-4d56-bb3d-1bdcf8aa9de6";"Responsible";"Responsible";"";"";"";"F
alse";"0"
"507E00CC-9520-4095-8528-17FF9DBE22CB";"Engine Part";"Engine Part";"";"";"";"F
alse";"0"
```

Erweiterung EnumRelationType

```
namespace BA.Training.Enums.Extensions
{
    [EnumExtension(typeof(EnumRelationType))]
    public static class EnumRelationTypeExtension
    {
        public const string ResponsibleGuid = "495F5CC6-8C75-4B55-A600-531BC49
FE416";
        public static readonly EnumRelationType Responsible = new EnumRelation
Type(ResponsibleGuid, 1000, "1582C2B1-CEF2-4D56-BB3D-1BDCF8AA9DE6", nameof(Res
ponsible));
    }
}
```

Erweiterung EnumParentRelationSubtypes

```
namespace BA.Training.Enums.Extensions
{
    [EnumExtension(typeof(EnumParentRelationSubtypes))]
    public class EnumParentRelationSubtypesExtension
    {
        public const string EnginePartGuid = "7FAFF96A-2A30-49E7-8BC7-75BB7073
1858";
        public static readonly EnumParentRelationSubtypes EnginePart = new Enu
mParentRelationSubtypes(EnginePartGuid, 1000, "507E00CC-9520-4095-8528-17FF9DB
E22CB", nameof(EnginePart));
    }
}
```

Erstellung des Relationtyps

```
namespace BA.Training
```

```
{
    public class RelationCreation : IRelationCreation
    {
        public void CreateRelationConfigurations(KeyedDictionary<Guid, Relatio
nTypeConfiguration> relationConfigurations)
        {
            relationConfigurations.Add(ResponsibleRelation());
        }

        private RelationTypeConfiguration ResponsibleRelation()
        {
            RelationTypeConfiguration relationConfiguration = new RelationType
Configuration()
            {
                Id = EnumRelationTypeExtension.Responsible,
                Type = EnumRelationCardinality.SomethingToAnythingGuid,
            };

            relationConfiguration.Initialize();

            relationConfiguration.SourceTypes.Add(EnumDataSource.UserProfileGu
id);
            relationConfiguration.ConfigurationName = "Responsible";
            relationConfiguration.Description = "Responsible relation";
            relationConfiguration.ShowInDesigner = true;

            return relationConfiguration;
        }
    }
}
```

Erweiterung OrmEngine

```
[Newtonsoft.Json.JsonIgnore]
[DontShowFieldInDesigner]
[RelationDefinitionTarget("67D7204E-EE21-4926-B2F7-A9E82A5B8552", EnumRelation
TypeExtension.ResponsibleGuid, MinCount = 0, MaxCount = 1)]
public OrmUserProfile RelatedResponsible
{
    get
    {
        return GetSourcePrimary<OrmUserProfile>(EnumRelationTypeExtension.Resp
onsible);
    }
}
```

```
[Newtonsoft.Json.JsonIgnore]
[DontShowFieldInDesigner]
[RelationDefinitionTarget("00016884-8F07-40A0-BB20-A15C623C197C", EnumRelation
Type.ParentGuid, EnumParentRelationSubtypesExtension.EnginePartGuid, MinCount
= 0, MaxCount = 0)]
public IQueryable<OrmEngine> RelatedMainEngines
{
    get
    {
        return GetSources<OrmEngine>(EnumRelationType.Parent, EnumParentRelati
onSubtypesExtension.EnginePartGuid);
    }
}

[Newtonsoft.Json.JsonIgnore]
[DontShowFieldInDesigner]
[RelationDefinitionSource("00016884-8F07-40A0-BB20-A15C623C197C")]
public IQueryable<OrmEngine> RelatedEngineParts
{
    get
    {
        return GetTargets<OrmEngine>(EnumRelationType.Parent, EnumParentRelati
onSubtypesExtension.EnginePartGuid);
    }
}
```

Inject der Relationsdefinition

```
[assembly: RelationDefinitionSourceInject(typeof(OrmUserProfile), "RelatedResp
onsibleOfEngines", "67D7204E-EE21-4926-B2F7-A9E82A5B8552")]
```

20.4. Übung 4

TypeScript Datei

```
module BA.Training {
    "use strict";
    export class Actions {
        public static ClientActionAddService(event: any, customData: CustomDat
a) {
            let igniterParams: CustomData = {};
            igniterParams['RemarkMessage'] = customData.RemarkMessage;
            customData["MassOperationIgniterParameters"] = JSON.stringify(igni
terParams);
            BA.Ui.Actions.GridActions.StartMassOperation(event, customData);
        }
    }

    window.setTimeout(
        function () {
            BA.Ui.Actions.ActionHandler.RegisterAction("BA.Training.ClientActi
onAddService", Actions.ClientActionAddService);
        },
        10
    );
}
```

Bundle Integration

```
namespace BA.Training.App_Start
{
    public class BundleConfig : IBundleConfig
    {
        public void RegisterBundles(BundleCollection bundles)
        {
            string assemblyName = GetType().Assembly.GetName().Name;

            // Script Bundle ermitteln und falls es noch nicht existiert erste
llen
            Bundle bundle = bundles.FirstOrDefault(ff => ff.Path == "~/bundle
s/scripts");
            if (bundle == null)
            {
                bundle = new ScriptBundle("~/bundles/scripts");
                bundles.Add(bundle);
```

```
            }

            // Eigene JavaScript Dateien einbinden (Auf Groß- und Kleinschreib
ung achten)
            bundle.Include(
                $"~/scripts/BA.Training/Actions.js?{ assemblyName }"
            );
        }
    }
}
```

Igniter

```
namespace BA.Training.Igniters
{
    public class AddServiceIgniter : OperationOverSelectedRecordsIgniterBase
    {
        public AddServiceIgniter(IgnitionModel ignitionModel) : base(ignitionM
odel)
        {
            AllRecordsIfNothingIsSelectedIndicator = false;
            CreateTemporaryRecordsIndicator = false;
            UniqueRecordListIndicator = true;
        }

        public override ActionResult Ignite()
        {
            if (SomethingSelected && IgnitionModel.Parameters.TryGetValue("Rem
arkMessage", out object remarkObj) && remarkObj is string remark && !string.Is
NullOrWhiteSpace(remark))
            {
                Session session = Api.ORM.GetNewSession();
                IEnumerable<Guid> oids = Records.Select(ff => ff.Oid);
                IQueryable<OrmEngine> engines = Api.ORM.GetQuery<OrmEngine>(se
ssion).Where(ff => oids.Contains(ff.Oid));
                DateTime date = DateTime.Now;
                foreach (OrmEngine engine in engines)
                {
                    OrmSubEngineServices service = engine.Services.AddNewObjec
t();
                    service.SortOrder = engine.Services.Count() - 1;
                    service.ServiceDate = date;
                    service.Remark = remark;
                    engine.Save();
                }
```

```
                return new JsonResult() { Data = new JsonFormResult() { Refres
hGrid = true } };
            }

            return null;
        }
    }
}
```

Ribbon bar Aktion

```
namespace BA.Training.Configuration.Navigation
{
    [Serializable]
    [Toolbox(EnumConfigurationType.NavigationConfigurationGuid, true)]
    [ControlFilter("NavigationConfigurationType", ExpressionType.Equal, EnumNa
vigationConfigurationType.RibbonNavigationGuid, EnumControlFilterApplyState.If
Positive)]
    public class ClientActionAddService : ClientActionGridMassOperationBase
    {
        [DisplayName("41ABA083-E37A-4709-98D4-1D685496459C")]
        public string RemarkMessage { get; set; }

        public ClientActionAddService() : base()
        {
            ToolboxName = "DEBED3B0-36A7-4948-B647-116D76A64504";
            Caption = "DEBED3B0-36A7-4948-B647-116D76A64504";
            ControlInitName = "TrainingActionAddService";
            ToolboxGroupName = "C007681C-8644-4BB0-A4A0-4A643265EABD";
            Id = "97C171CA-F872-4245-8AC5-43BF169155E7".ToGuid();
            Icon = "wrench";
            IconName = Icon;

            VisibilityForParentTypes.Add(EnumActionVisibleForParentType.Grid);

            MassOperationIgniter = typeof(AddServiceIgniter).AssemblyQualified
Name;

            DynamicClientVisibility.Add(EnumActionVisibility.SomethingSelecte
d);
            SomethingMustBeSelected = true;

            AdditionalClientData.AddOrUpdate("ActionMethodId", "BA.Training.Cl
ientActionAddService");
```

```
        }

        public override void AdditionalRibbonButtonAssignment(DevExpress.Web.R
ibbonButtonItem ribbonItem, EnumActionVisibleForParentType parentType, DevExUI
ModelBase uiModel = null)
        {
            base.AdditionalRibbonButtonAssignment(ribbonItem, parentType, uiMo
del);
            AdditionalClientData.AddOrUpdate("RemarkMessage", RemarkMessage);
        }
    }
}
```

20.5. Übung 5

TypeScript Aktion

```
public static ClientActionAddService(event: any, customData: CustomData) {
    let paramter: CustomData = {};
    paramter.RemarkMessage = customData.RemarkMessage;
    paramter.ServiceDate = customData.ServiceDate;
    BA.Ui.Dialog.DialogManager.OpenDialog("BA.Training.AddServiceDialog", cust
omData, customData,
        function (result: BA.Ui.Dialog.DialogResult, customData: CustomData) {
            if (result.ButtonId == 'okButton') {
                let igniterParams: CustomData = {};
                igniterParams.RemarkMessage = result.Data.Remark;
                igniterParams.ServiceDate = result.Data.Date;
                customData["MassOperationIgniterParameters"] = JSON.stringif
y(igniterParams);
                BA.Ui.Actions.GridActions.StartMassOperation(event, customDat
a);
            }
        }
    );
}
```

Dialog Model

```
public class AddServiceDialogModel
{
    [BARequired]
    public string Remark { get; set; }

    [BARequired]
    public DateTime Date { get; set; }
}
```

Dialog

```
[DialogImplementation("BA.Training.AddServiceDialog")]
public class AddServiceDialog : DialogImplementationBase
{
    public override void CreateDialogContent(DevExFormModel formModel, HttpReq
uestBase request, ModelStateDictionary modelState, Dictionary<String, Object>
parameter, object bindObject = null)
```

```
    {
        AddServiceDialogModel dataModel;
        if (bindObject == null)
            dataModel = new AddServiceDialogModel { Date = DateTime.Now, Remar
k = parameter.Get("RemarkMessage")?.ToString() };
        else
            dataModel = (AddServiceDialogModel)bindObject;

        formModel.Title = "95437073-8D05-4650-8F75-4B53AEB4A456".Translate();
        formModel.FormGuid = "A758783B-D133-4359-8A63-B91574C15220".ToGuid();
        formModel.DataSource = dataModel;
        formModel.Width = 600;

        LayoutPanelControl layout = new LayoutPanelControl()
        {
            Id = "DEB77CD6-1823-445F-A70D-A0E6001E4351".ToGuid(),
            ColumnCount = 2,
            WrapContentAtWidth = 400,
            //StretchLastItem = true,
        };
        layout.AddBreakpoint(new BreakpointRule() { Name = "S", ColumnCount =
1, MaxWidth = 500 });
        Controls.Add(layout);

        DateEditControl dateTime = new DateEditControl()
        {
            Id = "70C9B90B-832C-460A-9A9F-E83463C0CC1F".ToGuid(),
            Caption = "B2A43538-EF7C-416F-89AB-F501D09AE6E3",
            ShowTimeSection = false,
            ColSpan = 1,
            OrmFieldName = nameof(AddServiceDialogModel.Date)
        };
        layout.Controls.Add(dateTime);

        TextEditControl textControl = new TextEditControl()
        {
            Id = "6DF2D790-8710-4A69-9530-49DAA3F31B36".ToGuid(),
            Caption = "5D49A7FE-8165-4FDC-953F-A194BCFFCE26",
            ColSpan = 1,
            OrmFieldName = nameof(AddServiceDialogModel.Remark)
        };
        layout.Controls.Add(textControl);

        formModel.AddButton("BA.Training.InputButton", "5d49a7fe-8165-4fdc-953
f-a194bcffce26".Translate(), false, "BA.Training.Actions.InputButton");
        formModel.AddButton(DialogButtonIds.OkButton, "OK", true);
        formModel.AddButton(DialogButtonIds.CancelButton, "Cancel", false, "B
```

```
A.Ui.Dialog.DialogManager.DialogDefaultCancel");

        MVCxFormLayoutItemCollection formControls = DevExFormLayoutTranslato
r.TranslateControls(Controls, formModel.DataSource, formModel, null, requestCo
ntext: request);

        DevExFormPartModel partModel = new DevExFormPartModel();
        partModel.Controls = formControls;
        partModel.FormName = "dialogPart";
        formModel.LayoutName = EnumFormLayout.SingleColumn.LayoutName;
        formModel.LayoutPanels.Add(partModel);

        request.ConfigurationGuid(formModel.FormGuid);
    }

    public override DialogResultModel HandleAction(HttpRequestBase request, Mo
delStateDictionary modelState, Dictionary<String, Object> parameter, String bu
ttonId, object bindObject, string propertyPrefix = "")
    {
        DialogResultModel result = base.HandleAction(request, modelState, para
meter, buttonId, bindObject, propertyPrefix);
        if(buttonId == DialogButtonIds.OkButton && modelState.IsValid)
        {
            AddServiceDialogModel dataModel = (AddServiceDialogModel)bindObjec
t;
            result.Data = dataModel;
        }

        return result;
    }
}
```

TypeScript Button

```
public static InputButton(button: BA.Ui.Dialog.BADialogButton, evt: ASPxClient
ButtonClickEventArgs) {
    var texts = [];
    var titleGuid = "41aba083-e37a-4709-98d4-1d685496459c";
    var messageGuid = "5d49a7fe-8165-4fdc-953f-a194bcffce26";
    texts.push(titleGuid);
    texts.push(messageGuid);
    BA.Ui.Translations.TranslationTools.GetTranslations(texts, function (resul
ts) {
        BA.Ui.MessageBox.ShowPrompt(results[titleGuid], results[messageGuid],
200, function (result) {
```

```
            let textBoxName: string = BA.Ui.Dialog.DialogManager.GetDialogCont
rolName(button, "Remark");
            if (window[textBoxName]) {
                let box: BAClientTextBox = <BAClientTextBox>window[textBoxNam
e];
                box.SetValue(result);
            }
        });
    }, false);
}
```

Angepasster Igniter

```
public override ActionResult Ignite()
{
    if (SomethingSelected)
    {
        if (IgnitionModel.Parameters.TryGetValue("RemarkMessage", out object r
emarkObj) && remarkObj is string remark && !string.IsNullOrWhiteSpace(remark))
        {
            Session session = Api.ORM.GetNewSession();
            IEnumerable<Guid> oids = Records.Select(ff => ff.Oid);
            IQueryable<OrmEngine> engines = Api.ORM.GetQuery<OrmEngine>(sessio
n).Where(ff => oids.Contains(ff.Oid));
            DateTime date;
            if (IgnitionModel.Parameters.TryGetValue("ServiceDate", out objec
t dateObj) && dateObj is DateTime)
                date = (DateTime)dateObj;
            else
                date = DateTime.Now;
            int counter = 0;
            foreach (OrmEngine engine in engines)
            {
                OrmSubEngineServices service = engine.Services.AddNewObject();
                service.SortOrder = engine.Services.Count() - 1;
                service.ServiceDate = date;
                service.Remark = remark;
                engine.Save();
                counter++;
            }

            Api.ClientCommunication.CreateSuccess(Api.User.CurrentUserGuid(),
"dd1ceadc-6808-4583-b6b7-dda73188b5a8".Translate(counter));
            return new JsonResult() { Data = new JsonFormResult() { RefreshGri
d = true } };
```

```
        }
        else
            Api.ClientCommunication.CreateError(Api.User.CurrentUserGuid(), "d
49ef787-3ad1-4f66-bb12-b2660fd71738".Translate());
    }

    return null;
}
```

20.6. Übung 6

Translation

```
"bd0d11a9-9e71-4980-9265-c4a037432d48";"Dialogberechtigte";"Dialog authorize
d";"";"";"";"False";"0"
```

Aktion

Interface `IOrmSecurityHandler` hinzufügen

```
public bool CanHandleOrm(object DataObject, OrmBABase orm)
{
    bool canHandle;
    if (orm != null)
        canHandle = orm.IsAllowed(EnumTableOperations.Edit);
    else
    {
        OrmEntityConfiguration entityConfig = Api.Config.OrmEntity(EnumDataSou
rceExtension.Engine.ValueGuid);
        canHandle = entityConfig.IsAllowed(EnumTableOperations.Edit) != EnumTa
bleOperations.Denied;
    }
    AdditionalClientData.AddOrUpdate("UserHasRole", canHandle);
    return canHandle;
}
```

Wird die Aktion in einer Maske ausgeführt, erhält man den Datensatz `orm` und kann dort die Prüfung konkret vornehmen. Wird die Aktion in einer Ansicht aufgerufen, kann nur geprüft werden, ob der Anwender prinzipiell Rechte haben könnte.

Dynamische Sichtbarkeit

```
DynamicClientVisibility.Add(EnumActionVisibility.IfUserHasRole);
```

Eigenschaft

```
[TokenboxControl]
[CDPRolesProviderProperties(dataSources: new[] { EnumDataSource.RoleGuid })]
[DisplayName("BD0D11A9-9E71-4980-9265-C4A037432D48")]
public RoleSet DialogRoles { get; set; }
```

AdditionalRibbonButtonAssignment

```
AdditionalClientData.AddOrUpdate("DialogAuthorized", Api.User.CurrentUserIsInR
ole(DialogRoles, false));
```

Igniter

```
public override ActionResult Ignite()
{
    if (SomethingSelected)
    {
        if (IgnitionModel.Parameters.TryGetValue("RemarkMessage", out object r
emarkObj) && remarkObj is string remark && !string.IsNullOrWhiteSpace(remark))
        {
            DateTime date;
            if (IgnitionModel.Parameters.TryGetValue("ServiceDate", out objec
t dateObj) && dateObj is DateTime)
                date = (DateTime)dateObj;
            else
                date = DateTime.Now;

            int successful = 0;
            int missingRights = 0;
            Session session = Api.ORM.GetNewSession();
            IEnumerable<Guid> oids = Records.Select(ff => ff.Oid);
            IQueryable<OrmEngine> engines = Api.ORM.GetQuery<OrmEngine>(sessio
n).Where(ff => oids.Contains(ff.Oid));
            foreach (OrmEngine engine in engines)
            {
                if (engine.IsAllowed(EnumTableOperations.Edit))
                {
                    OrmSubEngineServices service = engine.Services.AddNewObjec
t();
                    service.SortOrder = engine.Services.Count() - 1;
                    service.ServiceDate = date;
                    service.Remark = remark;
                    engine.Save();
                    successful++;
                }
                else
                    missingRights++;
            }

            if (missingRights == 0)
                Api.ClientCommunication.CreateSuccess(Api.User.CurrentUserGui
```

```
d(), "dd1ceadc-6808-4583-b6b7-dda73188b5a8".Translate(successful));
            else
                Api.ClientCommunication.CreateSuccess(Api.User.CurrentUserGui
d(), "e3f652e7-8019-4444-9e6c-f3c30528e862".Translate(successful, missingRight
s));
            return new JsonResult() { Data = new JsonFormResult() { RefreshGri
d = true } };
        }
        else
            Api.ClientCommunication.CreateError(Api.User.CurrentUserGuid(), "d
49ef787-3ad1-4f66-bb12-b2660fd71738".Translate());
    }

    return null;
}
```

Mit der Überprüfung der Bearbeiterrechte werden automatisch die Leserechte mitgeprüft, so dass bei der Abfrage auf die Prüfung der Leserechte verzichtet werden kann.

Type Script

```
public static ClientActionAddService(event: any, customData: CustomData) {
    if (customData.DialogAuthorized) {
        let paramter: CustomData = {};
        paramter.RemarkMessage = customData.RemarkMessage;
        paramter.ServiceDate = customData.ServiceDate;
        BA.Ui.Dialog.DialogManager.OpenDialog("BA.Training.AddServiceDialog",
customData, customData,
            function (result: BA.Ui.Dialog.DialogResult, customData: CustomDat
a) {
                if (result.ButtonId == 'okButton') {
                    Actions.StartAddServiceMassOperation(event, customData, re
sult.Data.Remark, result.Data.Date);
                }
            }
        );
    } else if (!customData.RemarkMessage || customData.RemarkMessage == "")
        BA.Ui.Translations.TranslationTools.GetTranslation("d49ef787-3ad1-4f6
6-bb12-b2660fd71738", function (result) {
            BA.Ui.Toast.Warning(result);
        });
    else
        Actions.StartAddServiceMassOperation(event, customData, customData.Rem
```

```
arkMessage, customData.ServiceDate);
}

public static StartAddServiceMassOperation(event: any, customData: CustomDat
a, remark: any, date: any) {
    let igniterParams: CustomData = {};
    igniterParams.RemarkMessage = remark;
    igniterParams.ServiceDate = date;
    customData["MassOperationIgniterParameters"] = JSON.stringify(igniterParam
s);
    BA.Ui.Actions.GridActions.StartMassOperation(event, customData);
}
```

20.7. Übung 7

Übersetzungen

```
"ba98d63f-8b15-4b68-852f-102cbeb70d6e";"Service hinzufügen (Massenverarbeitun
g)";"Add service (Mass worker)";"";"";"";"False";"0"
"281e4152-6d03-40b4-9971-10728e673a00";"Services hinzufügen";"Add service
s";"";"";"";"False";"0"
"61bca157-96a6-436b-8719-0521b98462b9";"Starte Prozess";"Start proces
s";"";"";"";"False";"0"
"227b79dd-31f2-4f6d-a255-94eb8aca0e89";"Neustart Prozess";"Restart proces
s";"";"";"";"False";"0"
"03a9ae68-226f-4f19-91cd-47fa4370230b";"{0} zu verarbeiten";"{0} to proces
s";"";"";"";"False";"0"
"e3d32867-83eb-4514-abec-c7e596ae192b";"{0} von {1} verarbeitet";"{0} of {1} p
rocessed";"";"";"";"False";"0"
"1cba5efa-7842-4764-abb8-05e2fab5bde3";"Maschine {0} wird verarbeitet.";"Engin
e {0} processing";"";"";"";"False";"0"
"0dd299df-c558-44fd-86ec-b3b2acd47530";"Keine Bearbeitungrechte für {0}";"No e
dit rights for {0}";"";"";"";"False";"0"
"cf6889ab-e880-42c9-8755-bfb15f9f01af";"Temporärer Datensatz wurde nicht gefun
den: {0} / {1}";"Temporary record not found: {0} / {1}";"";"";"";"False";"0"
"160c2db4-e013-4569-8ab1-a37bdf776707";"Meine Ausnahme";"My exceptio
n";"";"";"";"False";"0"
```

Hintergrundprozess

Worker

```
public class AddServiceWorker : WorkItemBase
{
    public Guid TaskExecutionId { get; set; }
    public string Remark { get; set; }
    public DateTime Date { get; set; }
    public int MissingRights { get; set; }

    public AddServiceWorker(): base()
    {
        ExpectedRunTime = EnumExpectedRunTime.Long;
    }

    protected override void Run()
    {
        IQueryable<OrmTempSelectedRecords> query = Api.ORM.GetQuery<OrmTempSel
```

```
ectedRecords>(UnitOfWork);
        query = query.Where(ff => ff.TaskExecutionId == TaskExecutionId);

        if (MaxProgress == 0)
            MaxProgress = query.Count();

        IQueryable<Guid> oids = query.Select(ff => ff.SelectedRecordOid);
        IQueryable<OrmEngine> engines = Api.ORM.GetQuery<OrmEngine>(UnitOfWor
k).Where(ff => oids.Contains(ff.Oid));
        foreach (OrmEngine engine in engines)
        {
            if (engine.IsAllowed(EnumTableOperations.Edit))
            {
                OrmSubEngineServices service = engine.Services.AddNewObject();
                service.SortOrder = engine.Services.Count() - 1;
                service.ServiceDate = Date;
                service.Remark = Remark;
                engine.Save();
            }
            else
                MissingRights++;

            query.Where(ff => ff.SelectedRecordOid == engine.Oid).FirstOrDefau
lt()?.Delete();

            if (CurrentProgress == 2 && NumberOfStarts == 1)
                throw new Exception(Logger.Translate("160C2DB4-E013-4569-8AB
1-A37BDF776707"));

            CurrentProgress++;
            Save();
        }
    }

    protected override void WorkItemFinished()
    {
        if (State == EnumWorkItemState.Finished)
            if (MissingRights == 0)
                Api.ClientCommunication.CreateSuccess(Api.User.CurrentUserGui
d(), "dd1ceadc-6808-4583-b6b7-dda73188b5a8".Translate(CurrentProgress));
            else
                Api.ClientCommunication.CreateSuccess(Api.User.CurrentUserGui
d(), "e3f652e7-8019-4444-9e6c-f3c30528e862".Translate(CurrentProgress - Missin
gRights, MissingRights));

        base.WorkItemFinished();
    }
```

```
}
```

Igniter-Änderung

Die temporären Datensätze müssen angelegt werden.

```
CreateTemporaryRecordsIndicator = true;
```

Anstatt der Verarbeitung, wird nun der Hintergrundprozess erstellt.

```
public override ActionResult Ignite()
{
    if (SomethingSelected)
    {
        if (IgnitionModel.Parameters.TryGetValue("RemarkMessage", out object r
emarkObj) && remarkObj is string remark && !string.IsNullOrWhiteSpace(remark))
        {
            DateTime date;
            if (IgnitionModel.Parameters.TryGetValue("ServiceDate", out objec
t dateObj) && dateObj is DateTime)
                date = (DateTime)dateObj;
            else
                date = DateTime.Now;

            Api.Worker.CreateOrUpdate(new AddServiceWorker
            {
                TaskExecutionId = TaskExecutionId,
                Remark = remark,
                Date = date
            });
        }
        else
            Api.ClientCommunication.CreateError(Api.User.CurrentUserGuid(), "d
49ef787-3ad1-4f66-bb12-b2660fd71738".Translate());
    }

    return null;
}
```

Antworten

Warum wurde ein Anwendungsprotokoll geschrieben? Und wie kann man es verhindern?
Der Vorgabewert von `LoggingMode` ist `WorkerLoggingMode.Auto`, damit ist dem System gestattet im Zweifelsfall ein Protokoll zu erstellen.
`LoggingMode = WorkerLoggingMode.Never` würde dies verhindern.

Wurden alle Datensätze korrekt verarbeitet?
Ja. Da der Arbeitsvorrat (die temporäre Tabelle) immer aktuell gehalten wurde. Beim Neustart wurde damit genau an dem Datensatz fortgesetzt bei dem der Fehler auftrat.

Massenverarbeitung

Aktion

```
[Serializable]
[Toolbox(EnumConfigurationType.NavigationConfigurationGuid, true)]
[ControlFilter("NavigationConfigurationType", ExpressionType.Equal, EnumNaviga
tionConfigurationType.RibbonNavigationGuid, EnumControlFilterApplyState.IfPosi
tive)]
public class ClientActionAddServiceMassWorker : ClientActionGridMassOperationB
ase, IOrmSecurityHandler, IIgnitableAction
{
    [DisplayName("41ABA083-E37A-4709-98D4-1D685496459C")]
    public string RemarkMessage { get; set; }

    [TokenboxControl]
    [CDPRolesProviderProperties(dataSources: new[] { EnumDataSource.RoleGuid
})]
    [DisplayName("BD0D11A9-9E71-4980-9265-C4A037432D48")]
    public RoleSet DialogRoles { get; set; }

    [Browsable(false)]
    public string MessageOnSuccess { get; set; }

    [Browsable(false)]
    public string MessageOnError { get; set; }

    [Browsable(false)]
    public string MessageOnStart { get; set; }

    [Browsable(false)]
    public string ProgressBarProcessName { get; set; }

    [Browsable(false)]
    public string ProgressBarProcessDescription { get; set; }

    public ClientActionAddServiceMassWorker() : base()
    {
        ToolboxName = "BA98D63F-8B15-4B68-852F-102CBEB70D6E";
        Caption = "BA98D63F-8B15-4B68-852F-102CBEB70D6E";
        ControlInitName = "TrainingActionAddServiceMassWorker";
        ToolboxGroupName = "C007681C-8644-4BB0-A4A0-4A643265EABD";
```

```
        Id = "D6F15F40-5A3F-4D02-AF5B-491C8705B7BC".ToGuid();
        Icon = "wrench";
        IconName = Icon;

        VisibilityForParentTypes.Add(EnumActionVisibleForParentType.Grid);

        MassOperationIgniter = typeof(OperationFromActionOverSelectedRecordsIg
niter).AssemblyQualifiedName;

        DynamicClientVisibility.Add(EnumActionVisibility.IfUserHasRole);
        DynamicClientVisibility.Add(EnumActionVisibility.SomethingSelected);
        SomethingMustBeSelected = true;

        AdditionalClientData.AddOrUpdate("ActionMethodId", "BA.Training.Client
ActionAddService");
    }

    public override void AdditionalRibbonButtonAssignment(DevExpress.Web.Ribbo
nButtonItem ribbonItem, EnumActionVisibleForParentType parentType, DevExUIMode
lBase uiModel = null)
    {
        base.AdditionalRibbonButtonAssignment(ribbonItem, parentType, uiMode
l);
        AdditionalClientData.AddOrUpdate("RemarkMessage", RemarkMessage);

        AdditionalClientData.AddOrUpdate("DialogAuthorized", Api.User.CurrentU
serIsInRole(DialogRoles, false));
    }

    public bool CanHandleOrm(object DataObject, OrmBABase orm)
    {
        bool canHandle;
        if (orm != null)
            canHandle = orm.IsAllowed(EnumTableOperations.Edit);
        else
        {
            OrmEntityConfiguration entityConfig = Api.Config.OrmEntity(EnumDat
aSourceExtension.Engine.ValueGuid);
            canHandle = entityConfig.IsAllowed(EnumTableOperations.Edit) != En
umTableOperations.Denied;
        }

        AdditionalClientData.AddOrUpdate("UserHasRole", canHandle);
        return canHandle;
    }

    public Type GetWorkItemType()
```

```
    {
        return typeof(AddServiceMassWorker);
    }

    public void Ignite(IIgnitableWorkItem task, Guid tempKey, Guid? taskExecut
ionId, Dictionary<string, object> parameters, JsonFormResult result)
    {
        AddServiceMassWorker worker = (AddServiceMassWorker)task;
        if (parameters.TryGetValue("RemarkMessage", out object remarkObj) && r
emarkObj is string remark && !string.IsNullOrWhiteSpace(remark))
        {
            DateTime date;
            if (parameters.TryGetValue("ServiceDate", out object dateObj) && d
ateObj is DateTime)
                date = (DateTime)dateObj;
            else
                date = DateTime.Now;
            worker.Remark = remark;
            worker.Date = date;
        }
    }
}
```

Erweiterung von EnumLogProcess

```
namespace BA.Training.Enums.Extensions
{
    [EnumExtension(typeof(EnumLogProcesses))]
    public static class EnumLogProcessesExtension
    {
        public const string AddServiceGuid = "FBF67FBA-9E64-4254-B168-A0C8DF80
6C63";
        public static readonly EnumLogProcesses AddService = new EnumLogProces
ses(AddServiceGuid, 1000, "281E4152-6D03-40B4-9971-10728E673A00");
    }
}
```

Worker

```
public class AddServiceMassWorker : MassWorkItem, IIgnitableWorkItem
{
    public Guid TemporaryGuid { get; set; }
    public Guid TaskExecutionId { get; set; }
    public string MessageOnSuccess { get; set; }
    public string MessageOnError { get; set; }
```

```
    public string ProgressBarProcessName { get; set; }
    public string ProgressBarProcessDescription { get; set; }
    public IIgnitableAction ClientAction { get; set; }

    public string Remark { get; set; }
    public DateTime Date { get; set; }
    public int MissingRights { get; set; }

    public AddServiceMassWorker() : base()
    {
        ScheduledStartTime = DateTime.UtcNow.AddMinutes(1);
        IsCancellable = true;
        LoggingProcess = EnumLogProcessesExtension.AddService;
        Caption = "281e4152-6d03-40b4-9971-10728e673a00";
        Title = "281e4152-6d03-40b4-9971-10728e673a00";
    }

    protected override IQueryable<OrmBABase> GetQueryable()
    {
        if (NumberOfStarts == 1)
            Logger.AddInfo("61BCA157-96A6-436B-8719-0521B98462B9");
        else
            Logger.AddInfo("227B79DD-31F2-4F6D-A255-94EB8ACA0E89");

        IQueryable<OrmTempSelectedRecords> query = Api.ORM.GetQuery<OrmTempSel
ectedRecords>(UnitOfWork);
        query = query.Where(ff => ff.TaskExecutionId == TaskExecutionId);

        if (MaxProgress == 0)
        {
            MaxProgress = query.Count();
            Logger.AddInfo("03A9AE68-226F-4F19-91CD-47FA4370230B", MaxProgres
s);
            throw new Exception(Logger.Translate("160C2DB4-E013-4569-8AB1-A37B
DF776707"));
        }
        else
            Logger.AddInfo("E3D32867-83EB-4514-ABEC-C7E596AE192B", CurrentProg
ress, MaxProgress);

        IQueryable<Guid> oids = query.Select(ff => ff.SelectedRecordOid);
        return Api.ORM.GetQuery<OrmEngine>(UnitOfWork).Where(ff => oids.Contai
ns(ff.Oid));
    }

    protected override void ProcessSingleOrm(OrmBABase ormBABase)
    {
```

```
        OrmEngine engine = (OrmEngine)ormBABase;
        Logger.AddInfo("1CBA5EFA-7842-4764-ABB8-05E2FAB5BDE3", engine.Name);

        if (CurrentProgress == 2)
            throw new Exception(Logger.Translate("160C2DB4-E013-4569-8AB1-A37B
DF776707"));

        if (engine.IsAllowed(EnumTableOperations.Edit))
        {
            OrmSubEngineServices service = engine.Services.AddNewObject();
            service.SortOrder = engine.Services.Count() - 1;
            service.ServiceDate = Date;
            service.Remark = Remark;
            engine.Save();
        }
        else
        {
            Logger.AddWarning("0DD299DF-C558-44FD-86EC-B3B2ACD47530", engine.N
ame);
            MissingRights++;
        }

        IQueryable<OrmTempSelectedRecords> query = Api.ORM.GetQuery<OrmTempSel
ectedRecords>(UnitOfWork);
        query = query.Where(ff => ff.TaskExecutionId == TaskExecutionId && f
f.SelectedRecordOid == engine.Oid);
        OrmTempSelectedRecords tempRecord = query.FirstOrDefault();
        if (tempRecord != null)
            tempRecord.Delete();
        else
            Logger.AddWarning("CF6889AB-E880-42C9-8755-BFB15F9F01AF", TaskExec
utionId, engine.Oid);
        CurrentProgress++;
    }

    protected override void WorkItemFinished()
    {
        if (State == EnumWorkItemState.Finished)
            if (MissingRights == 0)
            {
                Api.ClientCommunication.CreateSuccess(Api.User.CurrentUserGui
d(), "dd1ceadc-6808-4583-b6b7-dda73188b5a8".Translate(CurrentProgress));
                Logger.AddInfo("dd1ceadc-6808-4583-b6b7-dda73188b5a8", Current
Progress);
            }
            else
            {
```

```
                Api.ClientCommunication.CreateError(Api.User.CurrentUserGui
d(), "e3f652e7-8019-4444-9e6c-f3c30528e862".Translate(CurrentProgress - Missin
gRights, MissingRights));
                Logger.AddInfo("e3f652e7-8019-4444-9e6c-f3c30528e862", Current
Progress - MissingRights, MissingRights);
            }

        base.WorkItemFinished();
    }
}
```

Antworten

Was steht im Anwendungsprotokoll?

Zeitstempel	Schweregrad	Ereignistext
10.02.2021 10:40:26	Information	Start process
10.02.2021 10:40:26	Information	3 to process
10.02.2021 10:40:26	Fehler	Error in background process 'AddServiceMassWorker'. Try to restart. My exception
10.02.2021 10:41:26	Information	Restart process
10.02.2021 10:41:26	Information	0 of 3 processed
10.02.2021 10:41:26	Information	Engine Hydraulic Pumpe processing
10.02.2021 10:41:26	Information	Engine Müllpresse processing
10.02.2021 10:41:26	Information	Engine Papierpresse processing
10.02.2021 10:41:26	Fehler	Error when editing the record 'Papierpresse'. My exception
10.02.2021 10:41:26	Information	2 engines changed.

Anzahl: 10

Wurden alle Datensätze verarbeitet?
Nein, der Datensatz mit der Exception wurde übersprungen.

Warum gibt es einen Unterschied zum vorherigen Hintergrundprozess, und was müsste man tun, um Fehlerhafte Datensätze nochmal zu vearbeiten?
Das `MassWorkItem` überpringt Datensätze bei denen Ausnahmen auftreten.
Man muss in `WorkItemFinished` den Worker in dem Fall mit `AddSuccessor` nochmal starten. Dies macht aber nur Sinn, wenn der Fehler nicht dauerhaft ist.

20.8. Anwendung nach Übung 7

Anwendung als Vorlage nach Übung 7

Zusätzlich zu den Übungen sind folgende Programmierungen enthalten

1. Die Ribben bar Aktion `ClientActionCreateDefaultEngineRecord`, die einen Maschienendatensatz erstellt und das Namensfeld vorbelegt.
   Entweder wird der konfigurierte Wert genommen oder von einem selektierten Datensatz.
2. Im Dialog wird eine Ansicht mit dynamischer Ansichtenkonfiguration und einem dazugehörigen eigenen Datenprovider eingebunden.
3. Ein Maskensteuerelement `ValueAndAliasControl` mit entsprechenden Renderer
   Dieses implementiert zwei Eingabefelder deren Werte mit einem Delimter in ein Datenfeld geschrieben werden. Im Lesemodus wird nur der Wert angezeigt.
4. Die Workflow-Aktion `SetTextFieldWFAction` zum Setzen eines Textfeldes.
5. Eine weitere eigene Formelfunktion `ResponsibleOperator`, für den Zugriff auf den Betreuer in Relation.

Download