



Business App Leitfaden für Anpassungen

7.0 — Letzte Änderung: 11 March 2024

GEDYS IntraWare GmbH

Inhaltsverzeichnis

1. Einleitung zur Version 7.0	8
1.1. Konfiguration vs. Programmierung	9
1.2. Voraussetzungen	10
1.3. Business App Module	11
1.4. Änderungen zur vorherigen Version	17
1.4.1. Änderungen	18
1.4.1.1. Attachment Handling und API	28
1.4.1.2. Migration von Renderern	32
1.4.1.3. Migration von Dialogen	36
1.4.1.4. Migration von Print-Renderern	37
1.4.2. Neuheiten	39
2. Eigenes Projekt	41
2.1. Von Vorlage erstellen	42
2.2. Veröffentlichen	44
2.3. Tipps zu Nuget Updates/Installation	47
3. Mehrsprachigkeit	48
3.1. Übersetzungen definieren	49
3.2. Backend-API	50
3.3. Frontend-API	51
4. Auswahllisten	52
4.1. Eigene erstellen	54
4.2. Bestehende erweitern	57
4.3. Referenz vs. Werte	58
4.4. API	59
4.5. Übung 1	61
5. Datentabellen	62
5.1. Definition	63
5.2. Feldtypen definieren	65
5.2.1. Text	66
5.2.2. HTML	67
5.2.3. Boolisches	68
5.2.4. Datum-Zeit	69
5.2.5. Zeitraum	71
5.2.6. Zahlen	72
5.2.7. Auswahlwerte	73
5.2.8. Dateianhänge	75
5.2.9. Berechnete Spalten	76
5.2.10. Standardtitel und -hilfetexte definieren	78
5.3. Teil-Datentabelle	79
5.3.1. Definieren	80
5.3.2. In einer Datentabelle definieren	81

5.4. Gemeinsame Felder	83
5.5. Basis-Datentabellen definieren	84
5.6. Validatoren	85
5.6.1. Standard Validatoren	86
5.6.2. Eigene Validatoren	88
5.7. Automatische Nummernvergabe	91
5.8. Datensätze API (ORMs)	92
5.8.1. Erstellen und Bearbeiten	93
5.8.2. Laden, Abfragen	96
5.8.3. Löschen	98
5.8.4. Dateianhänge	99
5.8.5. Klonen	101
5.8.6. .Net Typ der Datensätze	102
5.8.7. URL Links erzeugen	103
5.9. Events für Datensätze und Teil-Datensätze	104
5.9.1. Life Cycle	105
5.9.2. Validierung	106
5.9.3. Teil-Datensätze	107
5.9.4. Feldänderungen	108
5.9.5. Import Life Cycle	109
5.9.6. Deaktivierung von Events	111
5.10. Übung 2	112
6. Relationen	115
6.1. Relationstypen	116
6.2. Relationsdefinitionen	119
6.3. Hinzufügen, Setzen und Entfernen	123
6.4. Abfragen	125
6.5. Hierarchieabfragen	126
6.6. Übung 3	128
7. Type Script	129
7.1. Eigene Bibliothek	130
7.2. Funktionen für Aktionen	132
7.3. UI Events	133
7.4. BA-Bibliotheken	138
8. Ribbon bar Aktionen	139
8.1. Client Aktionen	140
8.2. Verarbeitung von selektierten Datensätzen	145
8.3. Übung 4	148
9. Dialoge	149
9.1. Toaster	150
9.2. Messageboxen	153
9.3. Umgang mit Dialogen	155
9.4. Standard Dialoge	157

9.5. Auswahl von Datensätzen	161
9.6. Datensätze im Dialog	164
9.7. Eigene Dialoge implementieren	167
9.8. Maskensteuerelemente für Dialoge	176
9.9. Übung 5	187
10. Berechtigungen	188
10.1. Rollen und Benutzer	189
10.2. Berechtigungen für Datensätze	190
10.3. Vergabe von Berechtigungen	191
10.4. Berechtigungsprüfungen	192
10.5. Eigene Berechtigungsprüfungen	194
10.6. Performance-Aspekte	200
10.7. Übung 6	201
11. Hintergrundprozesse	203
11.1. Allgemeines	204
11.1.1. Work-Items	206
11.1.2. Queues und Priorität	211
11.2. Die Work-Manager API	212
11.3. Implementierung eigener Work-Items	216
11.3.1. Allgemeine, öffentliche Eigenschaften	217
11.3.2. Eigenschaften für Fortschrittsanzeige	221
11.3.3. Geschützte Eigenschaften	222
11.3.4. Geschützte Methoden	223
11.3.5. Überschreibbare Methoden	225
11.3.6. Eigene, persistierte Eigenschaften	229
11.3.7. Wiederkehrende Work-Items	230
11.3.8. Nachfolge-Worker starten	231
11.3.9. XPO Exceptions selbst fangen	232
11.4. Massenverarbeitung von Datensätzen	233
11.4.1. Ablauf	234
11.4.2. Abstrakte Methoden	235
11.4.3. Optionale Implementierungen	236
11.4.4. Aktionen auf alle selektieren Datensätze	238
11.5. Anwendungsprotokolle	241
11.6. Übung 7	244
12. Masken	246
12.1. Maskensteuerelemente	247
12.1.1. Klasse des Steuerelements	248
12.1.2. Klasse des Renderers	251
12.1.3. Vorgehen	259
12.1.3.1. Auswahlliste anpassen	260
12.1.3.2. Name	262
12.1.3.3. Sichtbarkeit	263
12.1.3.4. Label	264

12.1.3.5. Hilfetext	265
12.1.3.6. Adaptive und Responsive Layout	266
12.1.3.7. Datensatz als geändert anzeigen	267
12.1.3.8. Umgang mit Feldwerten	268
12.1.3.9. Automatisches setzen vom Fokus	269
12.2. Maskensteuerung	271
12.3. Aktualisierung von Maskenbereichen	273
12.4. Datensätze initialisieren	276
12.5. Datensatz öffnen	277
13. Seiten und Widgets	279
13.1. Klassen	280
13.2. Interfaces	282
13.3. Controllern	283
13.4. Parameter	284
13.5. Typescript	285
13.6. UI Events	288
13.7. Eigene Widgets	290
13.7.1. Einfaches Widget	291
13.7.2. Parameter konsumieren	297
13.7.3. Einen Parameter bereitstellen	301
13.7.4. Mehrere Parameter bereitstellen	306
13.7.5. Einstellung durch den Anwender	313
13.7.6. Widget Vorschau implementieren	316
13.7.7. Listen-Widget	320
14. Ansichten	336
14.1. Spaltensteuerelemente	337
14.1.1. Klasse des Steuerelements	338
14.1.2. Klasse des Renderers	340
14.2. Formeln basierende Spalten	343
14.3. Mehrfachgruppierung	345
14.4. Eigene Datenprovider	347
15. Workflow-Aktionen	350
15.1. Steuerelement	351
15.2. Basis-Infrastruktur	353
15.3. Umgang mit Workflow-Aktionen	354
16. Platzhalter	355
16.1. Berechnete Eigenschaften	356
16.2. Eigene Datenquelle	357
17. Migration	358
17.1. Allgemeines	359
17.2. Eigene Migrationen erstellen	362
17.3. Migration von gespeicherten Ansichten (ab 7.0.2)	366
17.4. Anhang	369

18. Formelsprache	371
18.1. Ausführen in .Net	373
18.2. In LINQ-Queries	375
18.3. Formelfeld in Steuerelementen	377
18.4. Methoden zum Bauen einer Formel	378
18.5. Eigene Formelfunktionen	384
18.6. Aufruf einer Formel-Funktion	386
18.7. Verkettung von Formeln	387
18.8. Nicht-Deterministische Funktionen	388
19. Generierte Masken	389
19.1. Allgemeines	390
19.2. Eingabeelemente	392
19.3. Auswahllisten für String-Eigenschaften	394
19.3.1. Datenprovider Modifiers	396
19.3.2. Datenprovider implementieren	398
19.4. Eigenschaften modifizieren	401
20. Sonstiges	402
20.1. Grundlagen	403
20.1.1. Assembly Typen	404
20.1.2. Dependency Injection	405
20.1.3. Eigener Controller	406
20.1.4. Dynamische Konfigurationen	407
20.1.5. Event beim Anwendungsstart	409
20.1.6. Zurücksetzen von Transaktionen	410
20.1.7. Audit Events	411
20.1.8. Umgang mit HTML-Strings	412
20.1.9. Asynchrone Prozesse	414
20.1.10. Translatable Exception	415
20.1.11. Designer Drag & Drop Regeln	417
20.1.12. Das ControlFilter Attribut	423
20.2. UI	424
20.2.1. Eigene Themes ausliefern	425
20.2.2. Icons	426
20.2.3. Hintergrund beim Anmelden	428
20.2.4. Eigene Styles einbinden	429
20.3. Funktionalitäten	430
20.3.1. E-Mails	431
20.3.2. Auswahl von E-Mail-Adressen anpassen	434
20.3.3. Ordner	435
20.3.4. Dubelettensuche	437
20.3.5. CSV-Konverter	443
20.3.6. Dateiparser für die Suche	453
20.4. Konfiguration und Auslieferung	455
20.4.1. Anwendungs- und Benutzereinstellungen	456

20.4.2. Eigene Konfigurationen	458
20.4.3. Konfigurationen ausliefern	460
20.4.4. Lizenzdialog erweitern	461
20.4.5. Logging (NLog)	462
20.4.6. Rollen des ersten Benutzers	463
21. Lösungen	464
21.1. Übung 1	465
21.2. Übung 2	467
21.3. Übung 3	468
21.4. Übung 4	471
21.5. Übung 5	474
21.6. Übung 6	478
21.7. Übung 7	482
21.8. Anwendung nach Übung 7	492

1. Einleitung zur Version 7.0

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 6.0](#)

1.1. Konfiguration vs. Programmierung

Die Anlage von [Übersetzungen](#) und [Auswahllisten](#), sowie die Definitionen von [Datentabellen](#) und [Relationen](#) sind sowohl konfiguratив als auch programmatisch möglich. Programmatisch ist es auch möglich mit den konfiguratив 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 Funktionalitäten 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 (Grundkenntnisse)

Technisch / Lizenzen

- MS Visual Studio (C# .Net Entwicklungsumgebung) (Empfohlen 2017 oder 2019)
- .Net 4.8 (Developer Pack)
- TypeScript 4.3
- DevExpress ASP.MVC und XPO (Produkt ASP.Net) 22.2.6
- MS SQL Server (Empfohlen 2019) alternativ PostgreSQL ab 12
- 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 Kontakttable

- 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. Dazu stehen zwei Vorgehen zur Verfügung. Zum Einen ein AddIn und zum Anderen eine Schnittstelle, das BusinessMail for Outlook angebunden werden kann.

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 überarbeitet. 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.Project

Beinhaltet

- Vollständige Funktionalität zum Anlegen und Verwalten von Projekten

Voraussetzung

- BA.Core
- BA.Contact
- BA.Activity
- BA.FollowUp
- BA.Appointment

- BA.CRM.Contact
- BA.CRM.Opportunity

BA.CRM.GoMobilePro

Beinhaltet

- Schnittstelle für die Anbindung von goMobilePro

Voraussetzung

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

BA.CRM

Beinhaltet

- Die Konfiguration der BA.CRM Anwendung

Voraussetzung

- BA.Core
- BA.Contact
- BA.Activity
- BA.Designer
- BA.Report
- BA.FollowUp
- BA.BusinessMail
- BA.Correspondence
- BA.CRM.Contact
- BA.CRM.Activity
- BA.CRM.Correspondence
- BA.CRM.Opportunity
- BA.CRM.FollowUp
- BA.CRM.Project
- BA.CRM.GoMobilePro

1.4. Änderungen zur vorherigen Version

[Vorherige Version 6.0](#)

[Hier](#) sind Änderungen beschrieben, die möglicherweise Anpassungen an bestehenden Implementierungen notwendig machen.

[Hier](#) sind Neuheiten bzw. Erweiterungen beschrieben, um einen Überblick über neue programmatische Möglichkeiten zu geben

[Hier](#) sind Hinweise die im Rahmen von Nuget Updates zu berücksichtigen sind.

1.4.1. Änderungen

IGridDataProvider erweitert [Compilerfehler]

IGridDataProvider hat eine neue Methode `GetQuery` erhalten. Diese liefert die Abfrage inklusive aller Filter aber ohne das `Select` der Daten für die Spalten. In der Regel also Abfragen auf konkrete Datentabellen. Beispiel `IQueryable<OrmCRMCompany>` oder `IQueryable<OrmBABase>`. Bei Ansichten die keine Datensätze anzeigen die entsprechenden anderen Datentypen.

Attachment Handling

Der Umgang mit Dateianhänge wurde angepasst. Genauer im Unterkapitel [Attachment Handling](#)

Überarbeitung des FormHandlings

Aus technischen Gründen, musste das FormHandling komplett überarbeitet werden. Daher müssen eigene [Renderers](#) und eigene [Dialoge](#) angepasst werden.

Folgende Nugets können deinstalliert werden

Von Business App werden folgende Nugets nicht mehr benötigt und können, wenn im Projekt keine Abhängigkeit vorhanden ist, deinstalliert werden.

- Elasticsearch.Net
- NEST

ConfigurationBase.DynamicallyGenerated jetzt automatisch

Das Property wird jetzt von BA.Core automatisch ermittelt und muss und kann nicht mehr zugewiesen werden.

=> In dynamischen Konfigurationen die Zeile

```
configuration.DynamicallyGenerated = true;
```

entfernen. Der Compiler weist im Zweifel auf den Fehler hin.

ConfigurationBase-Events jetzt alle protected

Folgende Events waren früher public:

```
protected virtual void BeforeSave() { }  
protected virtual void AfterCacheRefresh() { }  
protected virtual void AfterSave() { }
```

Falls die Methoden in projekteigenen Konfigurationen überschrieben wurden, müssen sie auf `protected` geändert werden. Der Compiler weist im Zweifel auf den Fehler hin.

IDataConfiguration geändert in IEntityConfigurationSpecifier

Das Interface `IDataConfiguration` wurde ersetzt durch `IEntityConfigurationSpecifier`.

Compilerfehler beim Zugriff auf `config.GetDataSource()`

- Ersetzen durch `config.OrmDataSource`

Falls man es öfter hat, führ eine globale Ersetzung von `.GetDataSource()` durch `.OrmDataSource` zum Ziel.

Eigene Konfigurations-Klassen, die das Interface implementiert haben

- Interface durch das neue ersetzen.
- Die Methode `GetDataSource()` umbenennen in `GetOrmEntityConfigurationGuid()`.
- Die Methoden `GetListOfFieldsUsed()` und `GetListOfRelationDefinitionsUsed()` ersatzlos entfernen.

Die Eigenschaft `ControlBase.HasErrors` existiert nicht mehr.

Alle Zuweisungen des Feldes sind ersatzlos zu entfernen.

Ebenfalls entfernt wurden `HasDesignErrors` und `DesignErrorTooltipMessage`.

Änderungen an `CDPOrmFieldsProperties`

Bei der Auswahl von Datenfeldern wird typischerweise `CDPOrmFieldsProperties` eingesetzt. Nun gibt es jetzt Datenspalten die nicht beschreibbar sind und Datenspalten die in Masken im Normalfall nicht beschrieben werden sollen. Technisch aber beschreibbar sind. In diesem Zusammenhang wurde der Datenprovider überarbeitet.

Read-Only-Spalten [immer prüfen]

Die Auswahl von Datenspalten muss jetzt sauber differenzieren, ob auch nur lesbare Spalten gewünscht sind oder nicht. Dies geschieht über den Parameter `includeReadOnly`:

- `ReadOnly` (default) – Es werden alle Spalten zur Auswahl angeboten, berechnete und normal

bearbeitbare.

Diese Auswahl sollte verwendet werden, wenn die gewählte Spalte gelesen werden soll. Dann funktionieren auch automatisch die neuen, berechneten Spalten.

- `None` – Es werden nur Spalten angeboten, die auch in Masken pflegbar sind.

Diese Auswahl ist empfohlen, wenn eine Spalte zum Schreiben/Befüllen gewählt werden soll.

- `NoUIWrite` – Es werden auch Spalten angeboten, die zwar technisch schreibbar sind, aber nicht für die Änderung durch Endbenutzer vorgesehen sind, weil sie z.B. automatisch zugewiesen werden wie `AttachmentState`. Diese Auswahl wird z.B. für die Workflow-Aktion `ChangeFieldAction` verwendet.

Datentypeinschränkung [nur bei Compilerfehler bearbeiten]

Die Optionen `TypesToShow` und `TypesNotToShow` haben jetzt den Datentyp `ICollection<EnumDataTypes>`. Dem sollte i.a. ein Array von `EnumDataTypes[]` zugewiesen werden.

Die Auswahl ist jetzt unabhängig von den verwendeten Controls in der Entity-Konfiguration und analog zu `CDPCommonFieldsProperties.TypesToShow`.

Die Werte sind im allgemeinen wie folgt zu ersetzen. (Einige der Ersetzungen sind bereits Arrays, andere brauchen noch ein `new[]`.)

- `OrmTextField -> EnumDataTypes.String`
- `OrmTextFieldBase -> EnumDataTypes.TextTypes`
- `OrmNumericFieldBase<> -> EnumDataTypes.NumberTypes`
- `OrmTextFieldBase, @OrmNumericFieldBase<> -> EnumDataTypes.TextNumberTypes@`
- `OrmDecimalField -> EnumDataTypes.Decimal`
- `OrmDateTimeField -> EnumDataTypes.DateTime`
- `OrmHTMLField -> EnumDataTypes.HTML`
- `OrmAttachmentsField -> EnumDataTypes.Attachments`
- `OrmEnumFieldBase -> EnumDataTypes.EnumTypes`
- `OrmEnumField -> ReferenceEnumTypes`
- `OrmSubRecordField -> EnumDataTypes.SubRecords`

Wenn `TypesToShow` direkt in einem Attribut angegeben wird, müssen statt dessen die `EnumDataSource.XXXGuid` Werte verwendet werden:

```
[CDPOrmFieldsProperties(typesToShow: new[] { EnumDataTypes.SubRecordsGuid })]
```

Eigene Steuerelemente für Masken

Wie im vorherigen Abschnitt erläutert wurde der Datenprovider `CDPOrmFieldsProperties` überarbeitet. Typischerweise erweitern Steuerlemente für Masken `DataControlBase`, welche eine virtuelle Methode `GetTypesOfOrmFieldName` zur Beeinflussung der Feldauswahl zur Verfügung stellt. Deren Signatur hat sich nach dem identischen Konzept wie im vorherigen Abschnitt erläutert umgestellt.

```
protected override ICollection<EnumDataTypes> GetTypesOfOrmFieldName()
```

Konfigurations-Datenprovider mit DataSources Filter [Compiler-Fehler oder Exception]

Wenn eigene Controls Eigenschaften haben, die fremde Konfigurationen (z.B. Masken) auswählen und einen DataSource-Filter nutzen, dann ist folgendes zu beachten:

- Die Enum-Werte lauten jetzt
MatchAnyTypeOrAscendant und MatchAnyTypeExact -> MatchSubSet
MatchAllTypesOrAscendants -> MatchSuperSet
- Zudem ist die Auswahl einer Match-Option jetzt obligatorisch, sobald ein DataSource-Filter verwendet wird. Andernfalls kommt eine ArgumentException zu Laufzeit. Das ist aber selten, weil die alte StandardOption MatchAnyTypeExact nicht korrekt mit Basisdatentypen umgehen konnte und deshalb kaum verwendbar war.
- Wenn nur Konfigurationen mit exakt übereinstimmenden Datentypen ausgewählt werden sollen, steht zusätzlich die Option MatchExact zur Verfügung.

Projektcodemigration: AttributesCache R.I.P [nur bei Compilerfehler]

Die Klasse AttributesCache existiert nicht mehr. Statt dessen wird eine .NET Collection verwendet. Alle Methoden der Klasse stehen jetzt als kompatible Extensions zur Verfügung.

- Alle Vorkommen von AttributesCache durch IReadOnlyCollection ersetzen.
- Falls das Field AttributesCache.Attributes verwendet wurde, “.Attributes” entfernen. Die .NET Collection ist bereits die Liste der Attribute.

Änderungen in den Nugets, um eigene MVC-Views zu implementieren

Möchte man in einem Projekt mit MVC-Views (also .cshtml) arbeiten, braucht man RazorGenerator.MsBuild als Nuget.

Ansonsten bekommt man diese Fehlermeldung

```
"die Ansicht ... oder die entsprechende Gestaltungsvorlage wurde nicht gefunden, oder keines der Ansichtsmodule unterstützt die durchsuchten Speicherorte. die folgenden Speicherorte wurden durchsucht: ..."
```

Kann über den Nuget Package Manager deinstalliert bzw installiert werden.

OrmBase.ModelState existiert nicht mehr [bei Compilerfehler]

ModelState ist eine Eigenschaft einer Maske nicht eines Datensatzes. Eventuelle Validierungsfehler

(z.B. in `OnBeforeValidation`) müssen in `OrmBase.ValidationResults` geschrieben werden. Von dort werden sie automatisch in den `ModelState` der Maske übertragen.

Die Syntax ist:

```
orm.ValidationResults.Add(new BAValidationResult(new ValidationContext(...),  
...));
```

Klasse `BAValidationBase` existiert nicht mehr [bei Compilerfehler]

Klassen, die davon geerbt haben, sollten jetzt direkt von `ValidationAttribute` erben und dessen Methode `IsValid(object value, ValidationContext validationContext)` überschreiben. Diese liefert statt einem `boolean` ein `ValidationResult`.

Für die Implementierung wird entweder die Hilfsmethode `ValidationAttributeHelper.CreateValidationResult` empfohlen, die den passenden Ergebnistyp aus der Boolean-Bedingung erzeugt, oder `new BAValidationResult(validationContext, ...)` im Fehlerfall.

`HtmlToPlainText`, `PlainTextToHtml`, `CleanHtml`, `HtmlEquals` umgezogen [optional]

Die Funktionen sind von `Api.Text` nach `API.Html` umgezogen. Die alten Funktionen sind obsolet, funktionieren aber noch.

ORM-Event `OnRelationsCreated`

Dieses Event wurde aufgrund eines Fehlers vor Release 7.0 auch bei Aktualisierungsgruppen in Masken aufgerufen. Das ist jetzt nicht mehr der Fall.

Falls sich Implementierungen auf dieses Verhalten verlassen haben, um z.B. auf aktuelle Benutzereingaben in der Maske zu reagieren, müssen sie angepasst werden und zusätzlich `OnUIRefreshing` implementieren. Dieses wird bei Aktualisierungsgruppen aufgerufen.

`LoadFromCache: createlfNotExists` entfernt [bei Compilerfehler]

Falls der optionale Parameter `createlfNotExists` von `ormabase.LoadFromCache` in Projektcode verwendet wurde, muss dieser beim Aufruf entfernt werden.

`web.config`

Die Nugets “`BundleTransformer.Core`” und “`BundleTransformer.NUGlify`”, deren Installation automatisch

folgende Einträge in der web.config-Datenbank vornimmt:

```
<sectionGroup name="bundleTransformer">
    <section name="core" type="BundleTransformer.Core.Configuration.CoreSettings, BundleTransformer.Core" />
    <section name="uglify" type="BundleTransformer.NUglify.Configuration.NUglifySettings, BundleTransformer.NUglify" />
</sectionGroup>
```

Die Konfiguration dieser Pakete erfolgt ebenfalls in der web.config Datei, direkt im Bereich <configuration>. Sollte durch die Installation der Nuget-Pakete eine Standardkonfiguration hinzugefügt worden sein, ist diese zu löschen.

Folgende Konfiguration muss in der web.config-Datei hinzugefügt werden, damit die Optimierung reibungsfrei durchgeführt werden kann:

```
<bundleTransformer xmlns="http://tempuri.org/BundleTransformer.Configuration.xsd">
    <core>
        <css defaultMinifier="NUglifyCssMinifier">
            <translators>
                <add name="NullTranslator"
                    type="BundleTransformer.Core.Translators.NullTranslator, BundleTransformer.Core"
                    enabled="false"/>
            </translators>
            <postProcessors>
                <add name="UrlRewritingCssPostProcessor"
                    type="BundleTransformer.Core.PostProcessors.UrlRewritingCssPostProcessor, BundleTransformer.Core"
                    useInDebugMode="false"/>
            </postProcessors>
            <minifiers>
                <add name="NullMinifier"
                    type="BundleTransformer.Core.Minifiers.NullMinifier, BundleTransformer.Core"/>
                <add name="NUglifyCssMinifier"
                    type="BundleTransformer.NUglify.Minifiers.NUglifyCssMinifier, BundleTransformer.NUglify"/>
            </minifiers>
            <fileExtensions>
                <add fileExtension=".css"
                    assetTypeCode="Css"/>
                <add fileExtension=".css?BA"
                    assetTypeCode="Css"/>
                <add fileExtension=".css?BA.Appointment"
```

```

        assetTypeCode="Css"/>
    <add fileExtension=".css?BA.Dashboard"
        assetTypeCode="Css"/>
    <add fileExtension=".css?BA.FollowUp"
        assetTypeCode="Css"/>
    <add fileExtension=".css?BA.Report"
        assetTypeCode="Css"/>
    <add fileExtension=".css?BA.Activity"
        assetTypeCode="Css"/>
    <add fileExtension=".css?BA.CRM.Contact"
        assetTypeCode="Css"/>
    <add fileExtension=".css?BA.CRM.Activity"
        assetTypeCode="Css"/>
    <add fileExtension=".css?BA.CRM.Opportunity"
        assetTypeCode="Css"/>
    <add fileExtension=".css?BA.CRM.FollowUp"
        assetTypeCode="Css"/>
    <add fileExtension=".css?BA.CRM.Project"
        assetTypeCode="Css"/>
    <add fileExtension=".css?BA.CRM.Appointment"
        assetTypeCode="Css"/>
    <add fileExtension=".css?BA.CRM.Marketing"
        assetTypeCode="Css"/>
    </fileExtensions>
</css>
<js defaultMinifier="NUglifyJsMinifier">
    <translators>
        <add name="NullTranslator"
            type="BundleTransformer.Core.Translator
s.NullTranslator, BundleTransformer.Core"
            enabled="false"/>
    </translators>
    <minifiers>
        <add name="NullMinifier"
            type="BundleTransformer.Core.Minifiers.NullMinifier, BundleTransformer.Core"/>
        <add name="NUglifyJsMinifier"
            type="BundleTransformer.NUglify.Minifier
s.NUglifyJsMinifier, BundleTransformer.NUglify"/>
    </minifiers>
    <fileExtensions>
        <add fileExtension=".js"
            assetTypeCode="JavaScript"/>
        <add fileExtension=".js?BA"
            assetTypeCode="JavaScript"/>
        <add fileExtension=".js?BA.Event"
            assetTypeCode="JavaScript"/>
    </fileExtensions>
</js>

```



```
<add fileExtension=".js?BA.Appointment"
      assetTypeCode="JavaScript"/>
<add fileExtension=".js?BA.Dashboard"
      assetTypeCode="JavaScript"/>
<add fileExtension=".js?BA.FollowUp"
      assetTypeCode="JavaScript"/>
<add fileExtension=".js?BA.Report"
      assetTypeCode="JavaScript"/>
<add fileExtension=".js?BA.Contact"
      assetTypeCode="JavaScript"/>
<add fileExtension=".js?BA.Activity"
      assetTypeCode="JavaScript"/>
<add fileExtension=".js?BA.Correspondence"
      assetTypeCode="JavaScript"/>
<add fileExtension=".js?BA.BusinessMail"
      assetTypeCode="JavaScript"/>
<add fileExtension=".js?BA.CRM.Contact"
      assetTypeCode="JavaScript"/>
<add fileExtension=".js?BA.CRM.Activity"
      assetTypeCode="JavaScript"/>
<add fileExtension=".js?BA.CRM.Correspondence"
      assetTypeCode="JavaScript"/>
<add fileExtension=".js?BA.CRM.Opportunity"
      assetTypeCode="JavaScript"/>
<add fileExtension=".js?BA.CRM.FollowUp"
      assetTypeCode="JavaScript"/>
<add fileExtension=".js?BA.CRM.Project"
      assetTypeCode="JavaScript"/>
<add fileExtension=".js?BA.CRM.Appointment"
      assetTypeCode="JavaScript"/>
<add fileExtension=".js?BA.CRM.Evalanche"
      assetTypeCode="JavaScript"/>
<add fileExtension=".js?BA.CRM.Marketing"
      assetTypeCode="JavaScript"/>
    </fileExtensions>
  </js>
</core>
<nuglify>
  <css allowEmbeddedAspNetBlocks="false"
        blocksStartOnSameLine="NewLine"
        ignoreAllErrors="false"
        ignoreErrorList=""
        indentSize="4"
        indentType="Space"
        lineBreakThreshold="2147482647"
        outputMode="SingleLine"
        preprocessorDefineList=""
```

```
        termSemicolons="false"
        colorNames="Hex"
        commentMode="Important"
        decodeEscapes="true"
        fixIE8Fonts="true"
        minifyExpressions="true"
        removeEmptyBlocks="true"
        severity="0"/>
<js allowEmbeddedAspNetBlocks="false"
    blocksStartOnSameLine="NewLine"
    ignoreAllErrors="false"
    ignoreErrorList=""
    indentSize="4"
    indentType="Space"
    lineBreakThreshold="2147482647"
    outputMode="SingleLine"
    preprocessorDefineList=""
    termSemicolons="false"
    alwaysEscapeNonAscii="false"
    amdSupport="false"
    collapseToLiteral="true"
    constStatementsMozilla="false"
    debugLookupList="Debug,$Debug,WAssert,Msn.Debug,Web.Debug"
    errorIfNotInlineSafe="false"
    evalLiteralExpressions="true"
    evalTreatment="MakeImmediateSafe"
    ignoreConditionalCompilation="false"
    ignorePreprocessorDefines="false"
    inlineSafeStrings="true"
    knownGlobalNamesList=""
    localRenaming="CrunchAll"
    macSafariQuirks="true"
    manualRenamesProperties="true"
    noAutoRenameList="$super"
    preserveFunctionNames="false"
    preserveImportantComments="true"
    quoteObjectLiteralProperties="false"
    removeFunctionExpressionNames="true"
    removeUnneededCode="true"
    renamePairs=""
    reorderScopeDeclarations="false"
    scriptVersion="EcmaScript6"
    strictMode="false"
    stripDebugStatements="true"
    severity="0"/>
</nuglify>
</bundleTransformer>
```

Fügt ein Service-Projekt eigene JavaScript- und/oder CSS-Bundles hinzu, so müssen an dieser Stelle die “fileExtensions” entsprechend hinzugefügt werden (`.css?<assembly-name>` oder `.js?<assembly-name>`).

1.4.1.1. Attachment Handling und API

- Inline-Attachments gehören jetzt immer zu genau einem HTML-Feld in der Datentabelle. Dessen Feldname steht jetzt analog zu normalen Attachments in der Spalte `OrmAttachment.Category`.
- Die `ContentID` eines Inline-Attachments ist kein starker Schlüssel. Diese kann aus Fremdsystemen kommen und durch Copy&Paste in verschiedenen Datensätzen und/oder verschiedenen HTML-Feldern gleich sein. Zwar sollten gemäß RFC2392 Content-IDs weltweit eindeutig sein und daher immer auf denselben Content verweisen, das gilt aber nicht für die Leserechte des Datensatzes in BA, wenn z.B. eine Termineinladung in Kalender mehrerer Benutzer repliziert wird.
- Für den Zugriff auf Inline-Attachments muss jetzt immer `OrmAttachments.Oid` verwendet werden. Diese steht jetzt auch im HTML in der Datenbank. Syntax: `img src="cid:AttachmentOID"`
- Die Eigenschaft `OrmAttachment.IsInline` existiert nicht mehr. Da jedes Attachment notwendigerweise zu genau einem Feld = einer Kategorie gehört und diese auch für den Zugriff erforderlich ist, ist die Eigenschaft `IsInline` immer inhärent eindeutig. Immer wenn die Kategorie ein HTML-Feld ist, ist es notwendigerweise ein Inline-Attachment und vice versa.
- Die Zuweisung von HTML mit Inline-Images zu anderen Datensätzen und/oder Datenspalten ist jetzt direkt möglich. Die darin referenzierten Inline-Attachments werden beim Speichern des Zieldatensatzes automatisch mit übernommen und die AttachmentOIDs umgeschlüsselt.
- Die Attachment-Funktionen auf Datensatzebene interagieren nicht mehr mit dem ORM-Cache. Es obliegt dem Aufrufer ggf. `LoadFromCache` / `SaveToCache` aufzurufen.

Constants.Attachments... [bei Compilerfehler]

Die Konstanten existieren nicht mehr, da es sich jetzt um Feldnamen von Datensätzen handelt.

- `DefaultAttachmentsCategory => nameof(OrmBABase.DefaultAttachments)`
- `InlineImagesCategory`
Hier muss jetzt immer der Name des HTML-Feldes angegeben werden, zu dem das Inline-Image gehört.

GetAttachmentById, GetAttachments, AddAttachment, RemoveAttachment: kein ORM Cache

Die Methoden auf Datensatzebene haben früher standardmäßig die Daten des aktuellen Datensatzes mit den Werten aus dem ORM-Cache überschrieben und die Änderungen danach in selbigen zurückgeschrieben, falls `RecordCacheId` gefüllt war und nicht explizit `reloadOrmFromCache: false` übergeben wurde. Das ist jetzt nicht mehr der Fall und der Parameter `reloadOrmFromCache` existiert nicht mehr.

Wenn Code darauf angewiesen ist, dass eventuelle, ungespeicherte Maskendaten im ORM Cache gelesen oder geschrieben werden sollen, muss das explizit erfolgen (wie bei allen anderen Datensatz-API-Funktionen auch).

Die datensatzunabhängigen Funktionen in `Api.Attachments` können wie bisher den ORM-Cache benutzen.

GetAttachments Übergabeparameter [bei Compilerfehler]

Beispiel:

```
bc... // alt
ormBABase.GetAttachments(new string[] { Name }, AttachmentInlineType.Both)

// neu
ormBABase.GetAttachments(Name)
```

AttachmentInlineType inline

Eine Filterung nach `IsInline` ist mangels dieser Eigenschaft nicht mehr möglich und nicht mehr notwendig. Der entsprechende Übergabeparameter existiert nicht mehr und muss ggf. entfernt werden.

categories -> category

Die Methoden zum Abrufen von Attachments `OrmBABase.GetAttachments` und `Api.Attachment.GetAttachments` akzeptieren jetzt nur noch eine Kategorie pro Aufruf, da alles andere logisch nicht sinnvoll ist. Die Aufrufe sind entsprechend umzustellen.

Falls doch mehrere Kategorien auf einmal abgerufen werden sollen (aber nicht alle), sollte direkt eine Abfrage auf `OrmAttachment` verwendet werden.

GetAttachmentByContentID [bei Compilerfehler]

Die Methode existiert nicht mehr, da das Ergebnis nicht eindeutig ist, wenn dasselbe Inline-Attachment beispielsweise für mehrere Terminteilnehmer repliziert ist. Es sollte statt dessen `GetAttachmentById` verwendet werden. Die dafür erforderliche OID steht bei HTML-Spalten auch in der Datenbank.

In `Api.Html...` gibt es Hilfsmethoden, um an die Attachment-OIDs zu kommen bzw. sie umzuschlüsseln.

Api.Attachments.GetAttachmentById [bei Compilerfehler]

Die Methode hat zusätzliche optionale Parameter zum filtern. Falls eine explizite Session übergeben wurde sollte das jetzt per benanntem Parameter geschehen: `GetAttachmentById(..., session: mySession);`

Api.Attachments.CreateAttachment [bei Compilerfehler]

Die Parameter Kategorie (= Feldname) und Session sind jetzt Pflicht.

category: Falls die Standard-Kategorie gemeint ist, muss `nameof(OrmBABase.DefaultAttachmen`

ts) übergeben werden.

Bei Inline-Images muss der Feldname, zu dem das Attachment gehören soll, übergeben werden. Andernfalls funktioniert das Inline-Attachment nicht.

`session`: Es muss immer die Session des Objektes angegeben werden, zu dem das Attachment hinzugefügt werden soll. Alles andere gab schon immer eine Exception.

=> Alternativ kann und sollte wo möglich die neue, leistungsfähigere Methode `orm.GetOrAddAttachment` verwendet werden, die das Erstellen und Hinzufügen eines Attachments bündelt und auch einige Sonderfälle wie z.B. doppelte ContentIDs oder Attachments von Konfigurationen behandelt.

Api.Attachments.AddAttachmentToOrm [bei Compilerfehler]

Der optionale Parameter `Session` existiert nicht mehr. Es wird nun automatisch die Session des übergebenen `OrmAttachment`-Objekts verwendet. Jede andere hat schon immer eine Exception gegeben.

ControlHelperController.AttachmentDownloadByContentId

Eine Funktion zum Abrufen eines Attachments anhand seiner Content-ID ist nicht mehr möglich, da diese nicht eindeutig ist und dadurch unter anderem falsche Leserechte greifen könnten.

Statt dessen sollte `DownloadAttachment` verwendet werden, was als Parameter die OID des `OrmAttachment` benötigt. Diese steht bei HTML-Spalten jetzt auch als "`cid:OID`"-URL in der Datenbank.

`Api.Html.ConvertInlineAttachmentsToDownloadUrl` kann die URLs in Inline-Images automatisch anpassen.

GetTotalFileSizeOfAttachmentControl [bei Compilerfehler]

Die Methode existiert nicht mehr, weil die Performance schlecht war. Es sollte `Api.Attachments.GetReadableSize(size)` verwendet werden. Die Größe bekommt man (wie bisher) über `Api.Attachments.GetTotalFileSize`

GetFileSizeInBytes [bei Compilerfehler]

Die Methode heißt jetzt `Api.Attachments.GetTotalFileSize`.

Export von HTML mit Inline-Images

Falls Projekt-Code HTML mit Inline-Images exportiert bzw. an Fremdsysteme weiter reicht, sollten die darin enthaltenen URLs der Inline-Images von OID auf Content-ID umgeschrieben werden. Das geht mit

der Funktion `Api.Html.ConvertHtmlInlineToContentId`. Diese liefert auch gleich die Liste der benötigten Attachments.

1.4.1.2. Migration von Renderern

1. Wahl des passenden Basisrenderers,

Je nach Funktionsumfang des zu migrierenden Renderers muss die Basisklasse auf einer der folgenden abgeändert werden

- `ControlDefaultRendererBase`: Renderer OHNE Unterteilung in Read- und EditMode-Methoden und OHNE Bind-Unterstützung
- `ControlDefaultBindableRendererBase`: Renderer OHNE Unterteilung in Read- und EditMode aber MIT Bind-Unterstützung
- `ControlDefaultReadEditRendererBase`: Renderer MIT Unterteilung in Read- und EditMode-Methoden und OHNE Bind-Unterstützung
- `ControlDefaultReadEditBindableRendererBase`: Renderer MIT Unterteilung in Read- und EditMode-Methoden und MIT Bind-Unterstützung

Renderer, die migriert werden müssen, erben allerdings schon oftmals von einem anderen Core- oder CRM-Renderer, so dass sich diese Frage nicht stellt und die Migration entsprechend der übergeordneten Basisklassen erfolgen muss.

2. Renderer ohne Unterteilung in Read- und Edit-Methoden

Häufig gibt es in den alten Implementierungen Stellen, an denen die ReadMode-Methode einfach per => auf die EditMode-Methode verweist (oder umgekehrt). Gerade für dieses frühere Vorgehen ist die Nutzung dieser Basisklassen sinnvoll.

Die implementierende Methode sollte in `Render()` umbenannt und eine möglicherweise verweisende Methode gelöscht werden.

Dieser Schritt kann auch nötig sein, wenn der abgeleitete Core-/CRM-Renderer nun eine entsprechende Basisklasse verwendet.

3. Entfernen der Übergabeparameter von Read-/EditMode und Bind

Renderer sind nun stateful und die Übergabeparameter an `ReadMode()`, `EditMode()` und `Bind()` entfallen.

Das Äquivalent von "parametersUi" ist das Instanz-Property "RenderParameters".

4. Entfernen der Variableninitialisierung

Viele Renderer-Methoden haben direkt am Anfang folgende Initialisierung:


```
FormControlRendererUiParameterModel parametersUi = (FormControlRendererUiParameterModel)parameters;  
FormRenderingContextUi renderingContext = (FormRenderingContextUi)parametersUi.RenderingContext;  
string namePrefix = RenderingUtils.GetUIReadyContextPrefix(renderingContext);
```

Eigentlich können diese Zeilen einfach weg genommen und die Zugriffe entsprechend auf die `RenderProperties` umgebogen werden.

Um Kompatibilität zum Rest der Implementierung zu wahren, kann der Block allerdings auch mit leichten Änderungen stehen bleiben:

```
FormControlRendererParameterModel parametersUi = RenderParameters;  
FormRenderingContext renderingContext = parametersUi.RenderingContext;  
string namePrefix = parametersUi.NamePrefix;
```

Konkrete Änderungen sind:

- `parametersUi` -> `RenderParameters`
- `renderingContext` -> `RenderParameters.RenderingContext`
- `namePrefix` -> `RenderParameters.NamePrefix`

5. Optional: Abruf der Property-Inhalte in den Render-Methoden nach Konvention

Der Zugriff auf das Property, muss in jedem Fall über die Deskriptoren erfolgen. Ein direktes Auslesen beispielsweise über `GetPropertyValue()` ist untersagt.

Für den Feldnamen des Controls auf dem Bind-Objekt wird garantiert, dass beim Abrufen von Daten zunächst `GetValueFromModel()` und anschließend `ConvertFromModelPropertyType()` aufgerufen wird.



Renderer müssen sich an diese Konvention halten!

Sollten weitere Werte aus dem Bind-Objekt benötigt werden, so werden diese ebenfalls immer unter Verwendung von `PrepareBindProperty()` und `GetValueFromModel()` abgerufen; der Aufruf auf `ConvertFromModelPropertyType()` ist in diesem Fall optional und muss von der Methode unterstützt werden.

Hieraus folgt:

Direkte Zugriffe auf `RenderParameters.BindData` sind immer kritisch zu betrachten. Hier ist man nur richtig, wenn tatsächlich der top-level-Datensatz verwendet werden soll. Das gültige Bind-Objekt für das aktuell angesteuerte Property ist `BindObject`! Was beispielsweise auch ein Teil-Datensatz sein kann.



Weiterhin ist die Nutzung von `GetProperty` und `SetProperty` auf egal welchem Bind-Objekt ab sofort untersagt. Es MUSS zwingend mit den Deskriptoren gearbeitet werden.

Beispiel:

Der Feldname des Controls ist `"Addresses[1].AddressType"`, `BindData` ist `OrmCRMCompany`. Durch den Aufruf `PrepareBindProperty("Addresses[1].AddressType")` passiert folgendes:

- `RenderParameters.BindData` bleibt wie es war eine Instanz von `OrmCRMCompany`
- `BindObject` wird auf den angesteuerten Teildatensatz verschoben
- `PreparedProperty` beinhaltet nun den `PropertyDescriptor` des Feldes `"AddressType"` für den Typen des Teildatensatzes

6. Pflege des ViewState

Der `ViewState` ist neu und enthält wichtige Informationen für die Steuerelemente und insbesondere für das Binding. Folgendes muss gepflegt werden.

- Ist ein Client-Control aus irgendeinem Grund für den Benutzer nicht editierbar, so muss `ViewState.ShouldBind = false` gesetzt werden.
- `ViewState.ViewStateData` muss, wenn möglich, unter dem Schlüssel `BindKeys.ModelProperty` den serverseitig verwendeten Feldnamen (Property-Namen auf dem `BindObject`) enthalten. Ist das Control ein `IPropertyControl` mit der Methode `GetFieldName()`, so wird dies vom Framework automatisch gemacht.
- `ViewState.ViewStateData` **solte** unter dem Schlüssel `BindKeys.ClientProperty` den clientseitig verwendeten Feldnamen (inkl. `RenderParameter.NamePrefix`, also z.B. `"DLG1.SearchNames"`) enthalten.
- Das zugehörige Control kann während des Bind-Vorgangs nicht verlässlich ermittelt werden; daher sollten Werte, die im Binding benötigt werden, ebenfalls unter eigenen Schlüsseln in `ViewStateData` abgelegt und während des Bindens wieder ausgelesen werden.



Werte, die in `ViewStateData` untergebracht werden sollen, **MÜSSEN** serialisierbar sein!

Erweitert der eigene Renderer nicht die Basisklassen, sondern einen Renderer vorhandener Steuerelementen, wird die Pflege schon durchgeführt, falls man gesichert die `base.Render()`, `base.ReadMode()` oder `base.EditMode()` aufruft.

7. Optional: Eigenes Binden von Werten nach Konvention

Für den Feldnamen des Controls auf dem Bind-Objekt wird garantiert, dass beim Binden von Daten zunächst `GetValueFromRequest()`, dann `ConvertToModelPropertyType()` und schließlich `Write`

`eValueToModel()` aufgerufen wird.



Renderer mit eigenem Binding müssen sich an diese Konvention halten!

Sollten weitere Werte in das Bind-Objekt geschrieben werden, so werden diese ebenfalls immer unter Verwendung von `GetValueFromRequest()`, `PrepareBindProperty()` und `WriteValueToModel()` verarbeitet; der Aufruf auf `ConvertToModelPropertyType()` ist in diesem Fall optional und muss von der Methode unterstützt werden.

Ansonsten gilt alles weitere aus Punkt 5.



Die Verwendung von `GetValueFromRequest()` wird stark empfohlen, da diese Methode Request-Validierungen vornimmt, um die Sicherheit vor feindlichen Inhalten zu gewährleisten.

8. Optional: Übertrag und ggf Anpassung von `ICustomBindingControl.AssignValue` nach `ConvertFromModelPropertyType`

Die Methode `ICustomBindingControl.AssignValue` diente früher dazu, Werte auf eine angepasste Weise aus dem Orm in das FormModel zu übertragen. Diese Methode wird innerhalb der Renderer durch `ConvertFromModelPropertyType()` abgelöst. Damit also beispielsweise der Renderer generisch mit einer `List<Guid>` arbeiten kann, muss `ConvertFromModelPropertyType()` dafür sorgen, dass der Wert, der hineingegeben wird, auch als `List<Guid>` zurückgegeben wird, egal, ob das nun eine einzelne `Guid`, ein `String` mit oder ohne Kommas oder ein `GuidSet` war.

9. Übertrag und ggf Anpassung von `ICustomBindingControl.Bind` nach `ConvertToModelPropertyType`

Wie Punkt 7 nur in Gegenrichtung.

Die Implementierung der Bind-Methode in den Basisrenderern ist für den Normalgebrauch vollkommen ausreichend. Es ist üblich, für einen eigenen Renderer die Convert-Methoden zu überschreiben, um Beispielsweise einen `string` in eine `Guid` oder eine `Guid` in einen `ValueEnum` umzuwandeln und damit weiterzuarbeiten. Weniger üblich – wenn auch dennoch valide – ist es, die `WriteValueToModel`-Methode zu überschreiben, wenn ein Wert beispielsweise gar nicht über ein einfaches `SetValue` auf dem `PropertyDescriptor` schreibbar ist, sondern vorher noch verarbeitet oder in mehrere Felder geschrieben werden muss. Am seltensten muss die Bind-Methode selbst überschrieben werden, und das auch nur, wenn zum Beispiel mehrere Werte zu einem Steuerelement gebunden werden müssen.

1.4.1.3. Migration von Dialogen

Dialoge arbeiten nun in der Implementierung nur noch mit Controls, nicht mehr mit `MVCxLayoutItems`. Diese werden vom Framework aus der Control-Liste erzeugt.

Das bedeutet, dass die Erstellung einer `MVCxFormLayoutItemCollection` per `DevExpressFormLayoutTranslator.Translate` in jedem Fall entfällt.

Daraus resultierend entfällt ebenfalls die Erstellung und Zuweisung eines `DevExpressFormPartModels` (`formModel.LayoutPanels.Add`) und das Setzen eines Layouts (`formModel.LayoutName`).

Ebenfalls überflüssig ist der Aufruf auf `request.ConfigurationId(...)`.

Zuweisungen von `formModel.FormGuid` und `formModel.FormInDialog` müssen (sollten) entfernt werden. Dieser Wert wird später überschrieben und die Zuweisung ist überflüssig.

Die Methode `HandleAction` bleibt unverändert.

Zwei weitere Methoden können nun überschrieben werden, um in abgeleiteten Dialogen Änderungen an den erstellten Controls und an den erstellten `MVCxFormLayoutItems` zu machen:

```
void AfterContentCreated(DevExpressFormModel formModel, HttpRequestBase request, ModelStateDictionary modelState, Dictionary<string, object> parameter, object bindObject)
```

Abfolge ist:

1. `CreateDialogContent`
2. Übernahme von `formModel.DataSource` als `bindObject`
3. `AfterContentCreated`

Das bedeutet, dass man in `AfterContentCreated` Änderungen an `this.Controls` durchführen kann, bevor die Controls in MVCx-Items umgewandelt werden.

Theoretisch erfüllt ein Überschreiben von `CreateDialogContent` mit Aufruf von `base.CreateDialogContent` den gleichen Zweck, man spart sich bei der Verwendung von `AfterContentCreated` allerdings, dass man an den base-Aufruf denken muss, d.h. `AfterContentCreated` ist die offizielle Schnittstelle, um Änderungen an den vom Dialog erstellten Controls zu machen.

```
void AfterControlsRendered(DevExpressFormModel formModel, HttpRequestBase request)
```

Hier können Änderungen an den umgewandelten MVCx-Items gemacht werden.

Diese finden sich unter `formModel.LayoutPanels` (Liste von `DevExpressFormPartModel`) und sind nach ihrer Hierarchie angeordnet (d.h. wenn man nicht weiß, nach was man wo sucht, muss man wahrscheinlich zunächst `FlattenControls` bemühen müssen).

1.4.1.4. Migration von Print-Renderern

Print-Renderer wurden als Renderer-Typ entfernt. Das Rendern für den Druck-Modus übernimmt nun ebenfalls der Standard-Renderer für das jeweilige Steuerelement.

Standard-Renderer implementieren je nach Basisklasse die Standard `Print()` Methode als Umleitung auf die `Render()` oder `ReadMode()` Methode. Diese kann von eigenen Renderern überschrieben werden.

Die anzuwendenden Regeln bei der Migration der Renderer entsprechen denen im Bereich des Lesemodus.

Beispiel

6.0: `HTMLControlPrintRenderer.ReadMode()`

```
public override void ReadMode(FormControlRendererParameterModel parameters)
{
    FormControlRendererPrintParameterModel parametersPrint = (FormControlR
endererPrintParameterModel)parameters;
    ViewableControlBase control = (ViewableControlBase)parameters.Control;
    FormRenderingContextPrint renderingContext = (FormRenderingContextPrin
t)parametersPrint.RenderingContext;

    String text = parametersPrint.BindData as String;

    if (!String.IsNullOrEmpty(text))
    {
        List<StringPairModel> replaceList = ConfigurationHelper.CidSto
rageToEditor(text);

        foreach (StringPairModel stringPair in replaceList)
            text = text.Replace(stringPair.String1, stringPair.Str
ing2);
    }

    StaticLabelControl htmlStaticPrint = new StaticLabelControl
    {
        Id = Guid.NewGuid(),
        Caption = control.Caption,
        EncodeHtml = false,
        LabelPosition = control.LabelPosition,
        LabelText = text?.ToString(),
        Visible = control.Visible
    };

    if (HttpContext.Current.Request.Browser.Type.Contains("Firefox"))
```

```

    {
        htmlStaticPrint.Width = 600;
        htmlStaticPrint.WidthType = EnumWidthType.Pixel;
    }

    parametersPrint.PrintControls.Add(htmlStaticPrint);
}

```

Wurde überführt in den `HTMLControlDefaultRenderer` in 7.0:

```

public override void Print()
{
    HTMLControl control = (HTMLControl)RenderParameters.Control;

    object val = GetValueFromModel();
    string text = (string)ConvertFromModelPropertyType(val) ?? "";

    Guid oid = Guid.Empty;
    string appendUrl = null;
    if (RenderParameters.BindData is OrmBase ormBind)
        appendUrl = $"parent={ormBind.Oid}";
    else if (RenderParameters.BindData is ApplicationConfiguration appConf
ig)
        appendUrl = $"parent={appConfig.Id}";

    if (!RenderParameters.RenderingContext.FormModel.RecordCacheId.IsNullo
rDefault())
        appendUrl += $"&recordCacheId={RenderParameters.RenderingConte
xt.FormModel.RecordCacheId}";

    Api.Html.ConvertInlineAttachmentsToDownloadUrl(text, appendUrl);

    RenderAsStaticLabel(text, (staticLabelControl) => staticLabelControl.E
ncodeHtml = false);
}

```

1.4.2. Neuheiten

Berechnete Eigenschaften in Platzhalter und HTML

Wenn man eine [Berechnete Eigenschaft](#) implementiert, kann man nun generiertes HTML auch automatisch einbetten lassen, ohne das der Parameter `type="html"` angegeben werden muss.

Berechnete Spalte in Datentabellen

In den Datentabellen kann man nun [berechnete Spalten](#) definieren.

Standardtitel und -hilfetexte für Datenfelder

es ist jetzt möglich Standardtitel und -hilfetexte für [Datenspalten](#) und [Quellrelationsdefinitionen](#) zu definieren.

Hilfe bei der Einführung

Falls bei der Migration der Anwendung programmierte Felder Standardtitel und -hilfetexte erhalten sollen, stellt sich die Frage wie diese in der Konfiguration etabliert werden. Also, wie stellt man Masken und Ansichten um?

Da die programmierten Standardtexte über die Konfiguration geändert werden können, sticht bei bestehenden Datenbanken immer die Konfiguration. Nur bei der ersten Migration auf das neue Release werden die programmierten Werte einmalig übernommen.

Daraus ergibt sich die Abhängigkeit, dass für die Analyse der optimalen Texte schon das neue Release erforderlich ist, bevor man die Werte programmieren kann. Dann kann man folgende Controller-Action nutzen:

```
.../bamaintain/AutoFillDefaultEntityTexts?inheritance=ApplyToAll
```

Diese erzeugt eine Liste alle Kandidaten für Standardtexte. Die Programmierung der Attribute muss manuell erfolgen. Man kann das Ergebnis von *AutoFillDefaultEntityTexts* nur zum kopieren der richtigen GUIDs nutzen.

Danach muss entweder die Migration der Datenbank wiederholt werden, damit die programmierten Werte (einmalig) angewendet werden, oder über die Controller-Action

```
.../bamaintain/AutoFillDefaultEntityTexts?assignFromCode=True&commit=True
```

können alle programmierten Werte auch nachträglich übernommen werden. Falls programmierte Werte nachträglich nochmal geändert und in die Konfiguration übernommen werden sollen:

```
.../bamaintain/AutoFillDefaultEntityTexts?assignFromCode=True&forceOverwrite=True&commit=True
```

Jetzt müssen noch die Masken und Ansichten auf die Nutzung der Standardtexte umgestellt werden. Dazu ist die Controller-Action

```
.../bamaintain/AutoSetUseDefaultEntityTexts?commit=True
```

aufzurufen. Diese wiederholt das Verhalten der Standardmigration und setzt die Häkchen für die Nutzung der Standardtexte wo immer dieselbe Übersetzung verwendet wird. Bei der letzten Controller-Aktion kann es zu Warnungen kommen, falls zwar die Übersetzung gleich ist, aber nicht deren GUID. Falls dort auch immer die Standardübersetzung verwendet werden soll, muss das Häkchen in der jeweiligen konfiguration manuell gesetzt werden.

BAValidationResult

Wenn eigene Validierungsfehler erzeugt werden, sollte zukünftig die Klasse `BAValidationResult` anstelle von `ValidationResult` verwendet werden. Diese bietet zwei zusätzliche Features:

1. Es kann zusätzlich zum technischen Namen ein Anzeigenamen für das zu validierende Feld übergeben werden.
2. Der Validierungsfehler wird nachgelagert übersetzt, was die Ausgabe in verschiedenen Sprachen ermöglicht, um z.B. in ein Anwendungsprotokoll in einer vom aktuellen Benutzer abweichenden Sprache schreiben zu können.

In eigenen Validatoren sollte anstelle von `new ValidationResult` entweder der Konstruktor `BAValidationResult(ValidationContext validationContext, string errorMessage, params object[] args)` oder alternativ die Hilfsmethode `ValidationAttributeHelper.CreateValidationResult` verwendet werden.

Falls an der jeweiligen Stelle kein `ValidationContext` zur Verfügung steht, muss der Konstruktor `BAValidationResult(string memberName, string displayName, string errorMessage, params object[] args)` verwendet und die Ermittlung des (unübersetzten) Anzeigenamens aus den `DisplayNameAttribute` selbst implementiert werden.

Es sollten keine übersetzten Texte an den Konstruktor übergeben werden.

Keinesfalls sollte die Fehlermeldung den Feldnamen als Präfix in der Art "[Fehlerhaftes Feld]: " enthalten. Das wird bei den Ausgabekanälen, wo das nicht aus dem Kontext hervor geht (z.B. Logfile), automatisch hinzugefügt.

Die alte Klasse `ValidationResult` funktioniert weiterhin, kann aber zu unschönen Fehlermeldungen führen.

Api.Html

Es gibt einen neuen API-Block `Api.Html`. Darin sind Funktionen rund um die Behandlung und Konvertierung von HTML-Datenfeldern. [Siehe auch](#)

2. Eigenes Projekt

2.1. Von Vorlage erstellen

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

[Download Vorlage Version 7.0](#)

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.

Neues Projekt konfigurieren

BA.Template

Projektname

Ort

Projektmappe

Name der Projektmappe ⓘ

☐ Platzieren Sie die Projektmappe und das Projekt im selben 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.

In den Eigenschaften des Projektes wird der Assemblyname und der Standardnamespace festgelegt. Es muss darauf geachtet werden, das diese beiden identisch sind.

Anwendung

Build

Web

Momentaufnahmebugger

Web packen/veröffentlichen

SQL packen/veröffentlichen

Buildereignisse

Ressourcen

Einstellungen

Verweispfade

Konfiguration: Nicht zutreffend Plattform: Nicht zutreffend

Assemblyname: BA.Training Standardnamespace: BA.Training

Zielframework: .NET Framework 4.8 Ausgabebetyp: Klassenbibliothek

☒ Bindungsumleitungen automatisch generieren

Startobjekt: (Nicht festgelegt)

Assemblyinformationen...

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/msbuild/2003">
  <PropertyGroup>
    <WebPublishMethod>MSDeploy</WebPublishMethod>
    <ADUsesOwinOrOpenIdConnect>False</ADUsesOwinOrOpenIdConnect>
    <LastUsedBuildConfiguration>Debug</LastUsedBuildConfiguration>
    <LastUsedPlatform>Any CPU</LastUsedPlatform>
    <SiteUrlToLaunchAfterPublish>http://server.company.de/ba_training</SiteUrlToLaunchAfterPublish>
    <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_training;Integrated Security=True" Name="data source=SQLServer;integrated security=SSPI;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\*.*

```

```
<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>
```

2.3. Tipps zu Nuget Updates/Installation

DevExpress Version

Die DevExpress Version im eigenen Projekt, muss zu der Version in den Nugets passen. Die jeweils aktuelle Version finden Sie [hier](#).

Änderungen berücksichtigen

Notwendige Änderungen am Sourcecode finden sie [hier](#).

Neue Möglichkeiten finden sie [hier](#).

NinjectWebCommon.cs entfernen

Sollte nach dem Update oder der Installation von Nugets die Datei NinjectWebCommon.cs im Projekt sein (typischerweise in "App_Start"), muss diese entfernt werden.

“RouteDebugger:Enabled” auf “false” setzen

Nach der Installation von neuen BA Nugets (Bekannt bei BA.CRM) wird in die web.config der Eintrag “RouteDebugger:Enabled” eingetragen. Falls dieser auf “true” steht, muss er auf “false” gestellt werden.

Folgender Fehler wird damit verhindert: “Die angegebene Route ist bereits in der Routenauflistung enthalten. Die Auflistung darf keine doppelten Routen enthalten.”

Assembly-Versionskonflikte

Alle Compiler-Warnungen bzgl. Assembly-Versionskonflikten sollten gelöst werden. Hier ist die [Anleitung](#) von MS:

Das wird realistisch betrachtet zukünftig bei allen größeren BA Nuget Upgrades vonnöten sein.

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.TranslationsTools` 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 mehrere 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 Auswahlliste in 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 Auswahlliste 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 visibleInManagment: 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 erzeugte 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(FirstValueGuid, 0, "[INSERT TRANSLATION 1 GUID]");
    public static readonly EnumMyEnum SecondValue = new EnumMyEnum(SecondValueGuid, 1, "[INSERT TRANSLATION 2 GUID]");

    public EnumMyEnum(String valueGuid, int sortOrder, String translationGuid) : 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(FirstValueGuid, 0, "[INSERT TRANSLATION 1 GUID]", "my value 1", "telephone2");
    public static readonly EnumMyEnum SecondValue = new EnumMyEnum(SecondValueGuid, 1, "[INSERT TRANSLATION 2 GUID]", "my value 2", "information");
}
```

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

public EnumMyEnum(String valueGuid, int sortOrder, String translationGuid, String ownProperty, String image = null) : base(valueGuid, sortOrder, translationGuid, 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(Data  
aSource1Guid, 200, "[INSERT TRANSLATION 1 GUID]");  
    public static readonly EnumDataSource DataSource2 = new EnumDataSource(Data  
aSource2Guid, 200, "[INSERT TRANSLATION 2 GUID]");  
    public static readonly EnumDataSource DataSource3 = new EnumDataSource(Data  
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 Referenz 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 Auswahllisten ist.

```
Api.Enum
```

Werte laden

Man kann einzelne Werte einer Auswahlliste laden.

```
EnumMyEnum firstValue = Api.Enum.GetEnumValue<EnumMyEnum>(EnumMyEnum.FirstValue.ValueGuid);  
ValueEnum secondValueBase = Api.Enum.GetEnumValue(EnumMyEnum.SecondValue.ValueGuid);  
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.MasterGuid);
```

Auswahlliste laden

Man kann einen einzelnen oder alle Auswahllisten laden.

```
MasterEnum master = Api.Enum.GetMasterEnum(EnumMyEnum.MasterGuid);  
ICollection<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.GetEnumValuesByTranslation(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.GetOrCreateEnumValueByTranslation(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 verschiedene 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

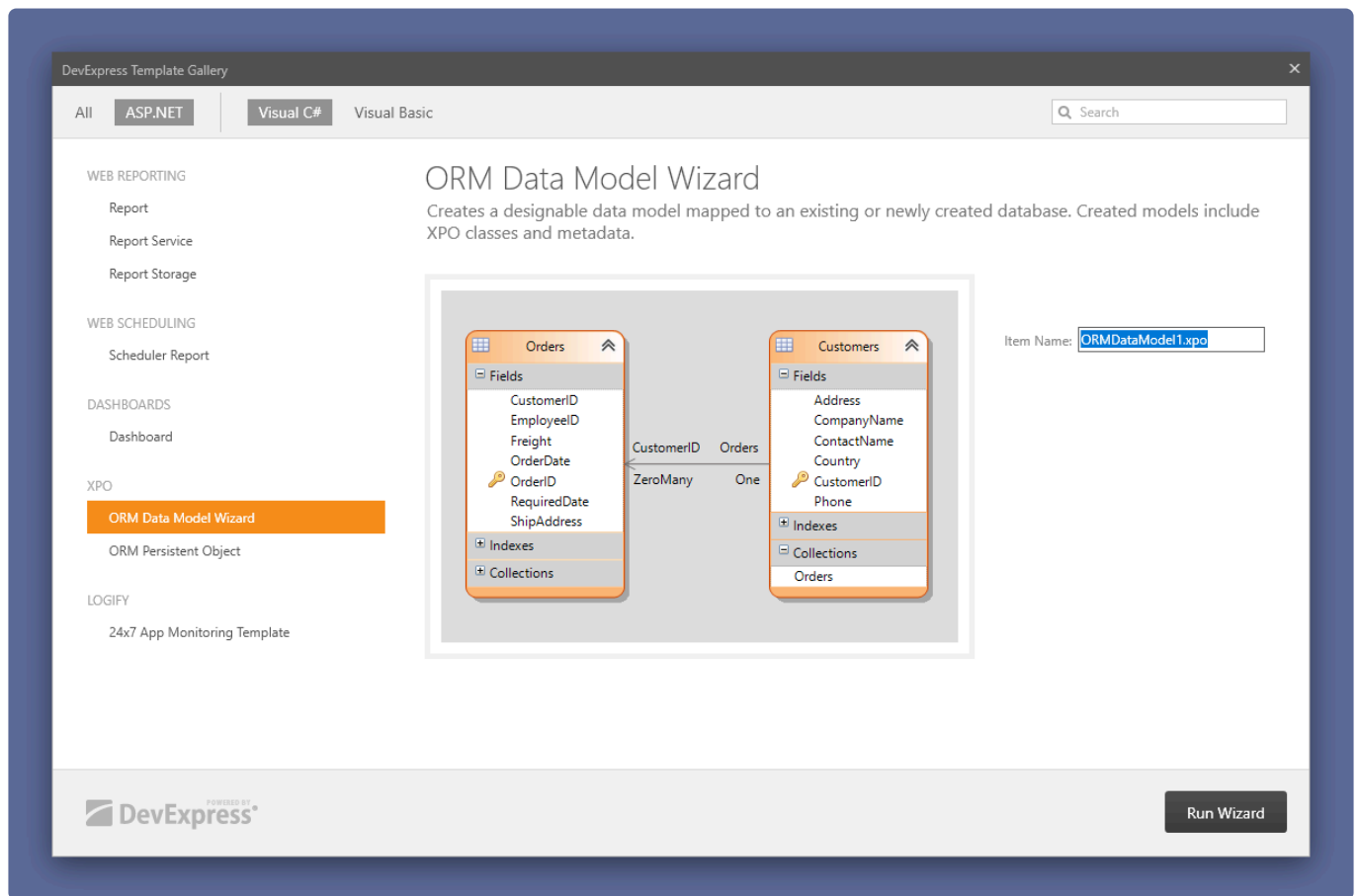
- Es gibt eine zugehörige Konfiguration.
- Validierung
- Created / CreatedBy
- Modified / ModifiedBy
- Änderungsverfolgung
- API zum Umgehen mit Auswahllisten
- Unterstützung von Events
- API für Relationen
- API für Berechtigungsprüfungen

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.

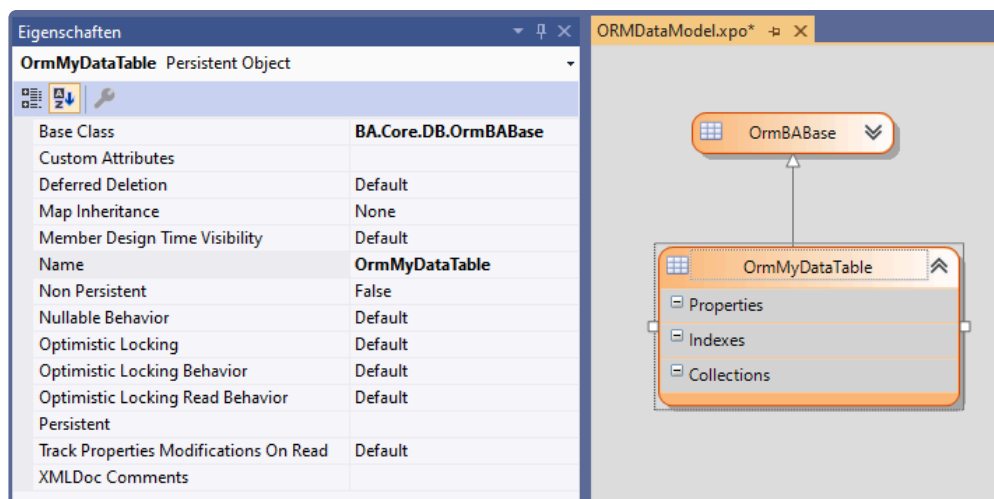
5.1. Definition

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(MyD  
ataTableGuid, 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 `false` kann die Datentabelle aus der Auswahl im Designer ausgeschlossen werden und mit `Description` kann man der Datentabelle eine Beschreibung geben.

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

Als Letztes wird die neue Datentabelle per [Dependency Injection](#) bekannt gemacht.

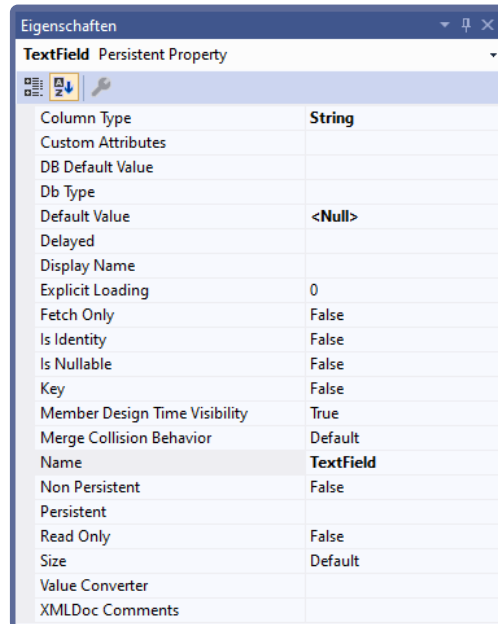
```
Bind<IOrmMyDataTable>().To<OrmMyDataTable>();
```

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 programmierte Feldnamen vermeidet Fehler.



Die Eigenschaft “Display Name” sollte nicht verwendet werden. [Hier](#) finden Sie das in BA sinnvolle Verfahren.



Beim nachträglichen Hinzufügen neuer Wertespalten ohne Nullwert ist zu beachten, dass die neue Spalte in der Datenbank immer erst mal den Wert NULL hat. Dies kann bei Zahlenspalten und Booleschen Werten zu unerwartetem Verhalten führen, weil Datenbankbedingungen wie “Spaltenname = 0” (z.B. in Ansichten) für alte Datensätze zunächst nicht anschlagen, obwohl der Wert in Masken und Reports augenscheinlich als 0 ausgewiesen wird. Es wird empfohlen in solchen Fällen ein Datenbankmigrationsklasse [Link auf Doku hinzufügen] zu schreiben, die die neue Spalte mit einem Initialwert füllt.

5.2.1. Text

Eigenschaften

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

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

Siehe auch [Standardtitel und -hilfetext](#)

5.2.2. HTML

Eigenschaften

- Column Type: `String`
- Name: [Beliebiger Feldname]
- Size: `Unlimited`
- Custom Attributes: `BA.Core.CustomAttributes.HTMLField`
- Is Nullable: `True`



HTML Felder sind in Teil-Datentabellen nicht möglich.

Siehe auch [Standardtitel und -hilfetext](#)

5.2.3. Boolisches

Eigenschaften

- Column Type: `Boolean`
- Name: [Beliebiger Feldname]
- Is Nullable: `False`

Siehe auch [Standardtitel und -hilfetext](#)

5.2.4. Datum-Zeit

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.

Siehe auch [Standardtitel und -hilfetext](#)

Is Nullable

Konfigurativ sind grundsätzlich alle Datumsspalten "Nullable" `DateTime?`. Programmatisch kann man sich an dieser Stelle entscheiden, ob die Spalte eventuell doch nicht "Nullable" `DateTime` sein sollte. Dies kann Vorteile haben, wenn man programmatisch mit diesen Spalten umgehen möchte. Abfragen wie `datetime.HasValue` und den Zugriff per `datetime.Value` sind dann überflüssig.

Aber nur wenn das Datenmodell definitiv immer einen Wert benötigt, sollte die "Nullable" Eigenschaft deaktiviert werden. Wir empfehlen eine sinnvolle Vorbelegung und eine Validierung, damit keine unerwünschten Nebeneffekte auftreten. Beispielsweise das Speichern des Default-Wertes von `DateTime`. Dieser könnte in bestimmten Situationen in den Steuerelementen zur Eingabe erscheinen.

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.

5.2.5. Zeitraum

Wenn man Zeitraumbfelder programmieren möchte, ist wie folgt vorzugehen:

```
[OrmDateRangeField(nameof(Start), nameof(End))]  
[MemberDesignTimeVisibility(false), HideInFilterEditor]  
public Range<DateTime> Time  
{  
    get => new Range<DateTime>(Start, End);  
    set => (Start, End) = value;  
}
```

oder, falls NULL-Werte zulässig sind:

```
[OrmDateRangeField(nameof(Start), nameof(End))]  
[MemberDesignTimeVisibility(false), HideInFilterEditor]  
public Range<DateTime>? Time  
{  
    get => Start.HasValue && End.HasValue ? new Range<DateTime>(Start, End) :  
default;  
    set => (Start, End) = value.HasValue ? (value.Value.From, value.Value.To)  
: default((DateTime?, DateTime?));  
}
```

Das neue Property kann nicht im XPO-Designer angelegt werden. Es muss in der Klasse ausprogrammiert werden. Die beiden Unterfelder Start und End müssen wie bisher ganz normal existieren und vom Datentyp DateTime oder DateTime? sein. Eine nachträgliche Gruppierung von zwei existierende Datumsspalten zu einem Zeitraum erfordert keine Migration. Eine Datumsspalte kann nur einem Zeitraum hinzugefügt werden.

5.2.6. Zahlen

Eigenschaften

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

Siehe auch [Standardtitel und -hilfetext](#)

Es wird empfohlen Zahlen als Nullable zu definieren. Tut man das nicht, kann es in Masken zum unerwarteten Verhalten kommen, dass automatisch der Wert "0" gespeichert wird, auch wenn der Benutzer gar keine Eingabe gemacht hat.

Gültige Zahlentypen sind:

- „Ganze Zahl“ (`int = Int32`)
- „Ganze 64-Bit-Zahl“ (`long = Int64`)
- „Gleitkommazahl“ (`double`)
- „Dezimale Gleitkommazahl“ (`decimal`)

Is Nullable

Konfigurativ sind grundsätzlich alle Zahlenspalten "Nullable" `int?`. Programmatisch kann man sich an dieser Stelle entscheiden, ob die Spalte eventuell doch nicht "Nullable" `int` sein sollte. Dies vereinfacht das programmatische Rechnen mit diesen Spalten. Abfragen wie `number.HasValue` und den Zugriff per `number.Value` sind dann überflüssig.

Aber nur wenn das Datenmodell definitiv immer einen Wert benötigt, sollte die "Nullable" Eigenschaft deaktiviert werden. Wir empfehlen eine sinnvolle Vorbelegung und eine Validierung, damit keine unerwünschten Nebeneffekte auftreten. Beispielsweise das Ändern des Wertes auf "0", wenn das Datenfeld vom Anwender geleert wird.

5.2.7. Auswahlwerte

Es gibt zwei Arten von [Auswahllisten](#). In der Ersten wird lediglich eine Referenz auf den eigentlichen Wert gespeichert und in der Zweiten werden die Werte als einen Komma separierten String abgespeichert. Primär wird die Referenz Variante [empfohlen](#).

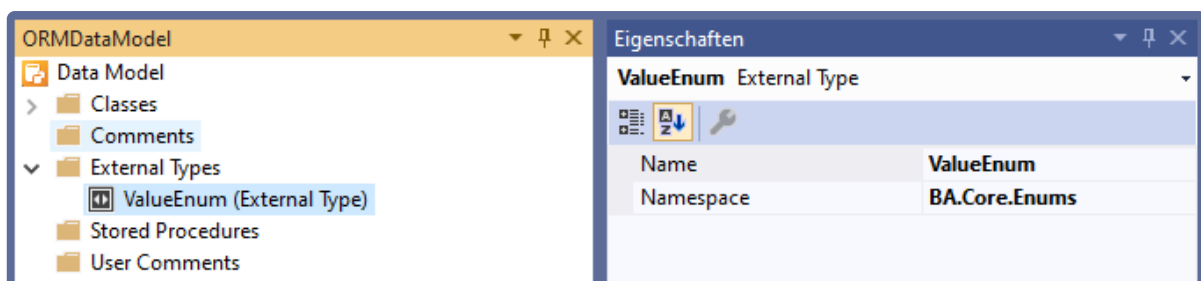
Referenz

Auf Basis des Referenzauswahllistenwert können sowohl Einzelauswahlwert als auch Mehrfachauswahlwert implementiert werden. Dabei unterscheiden sich diese Beiden lediglich in der Deklaration der "Custom Attributes". Als Typ wird entweder `TypeEnumField.Multiple` (Wie im unteren Beispiel) oder `TypeEnumField.Single`

Eigenschaften

- Column Type: [Der Enumtyp. Beispiel: `BA.MyProject.EnumMyEnum`]
- Custom Attributes: `BA.Core.CustomAttributes.EnumFieldAttribute(BA.Training.Enums.EnumMyEnum.Guid, BA.Core.CustomAttributes.TypeOfEnumField.Multiple)`
- 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.

Siehe auch [Standardtitel und -hilfetext](#)

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.EnumFieldAttribute(BA.Training.Enums.EnumMyEnum.Guid, BA.Core.CustomAttributes.TypeOfEnumField.Simple)`
- **Name:** [Beliebiger Feldname]
- **Size:** Möglichst konkret (im Zweifelsfall oder Multi-Value unlimited)

Siehe auch [Standardtitel und -hilfetext](#)

5.2.8. Dateianhänge

Die programmatische Erstellung eines eigenen Feldes erfolgt Analog zu anderen Feldern, indem auf der Orm-Klasse eine Eigenschaft implementiert wird. Dabei sollte der Name des Feldes immer eindeutig sein. Die Groß- und Kleinschreibung spielt dabei keine Rolle und ist dabei identisch mit der Kategorie des Anhangs.

```
public IReadOnlyList<OrmAttachment> MyAttachments
{
    get
    {
        return AttachmentsInternal.Where(ff => nameof(MyAttachments).Equals(f
f.Category, StringComparison.OrdinalIgnoreCase)).ToArray();
    }
}
```

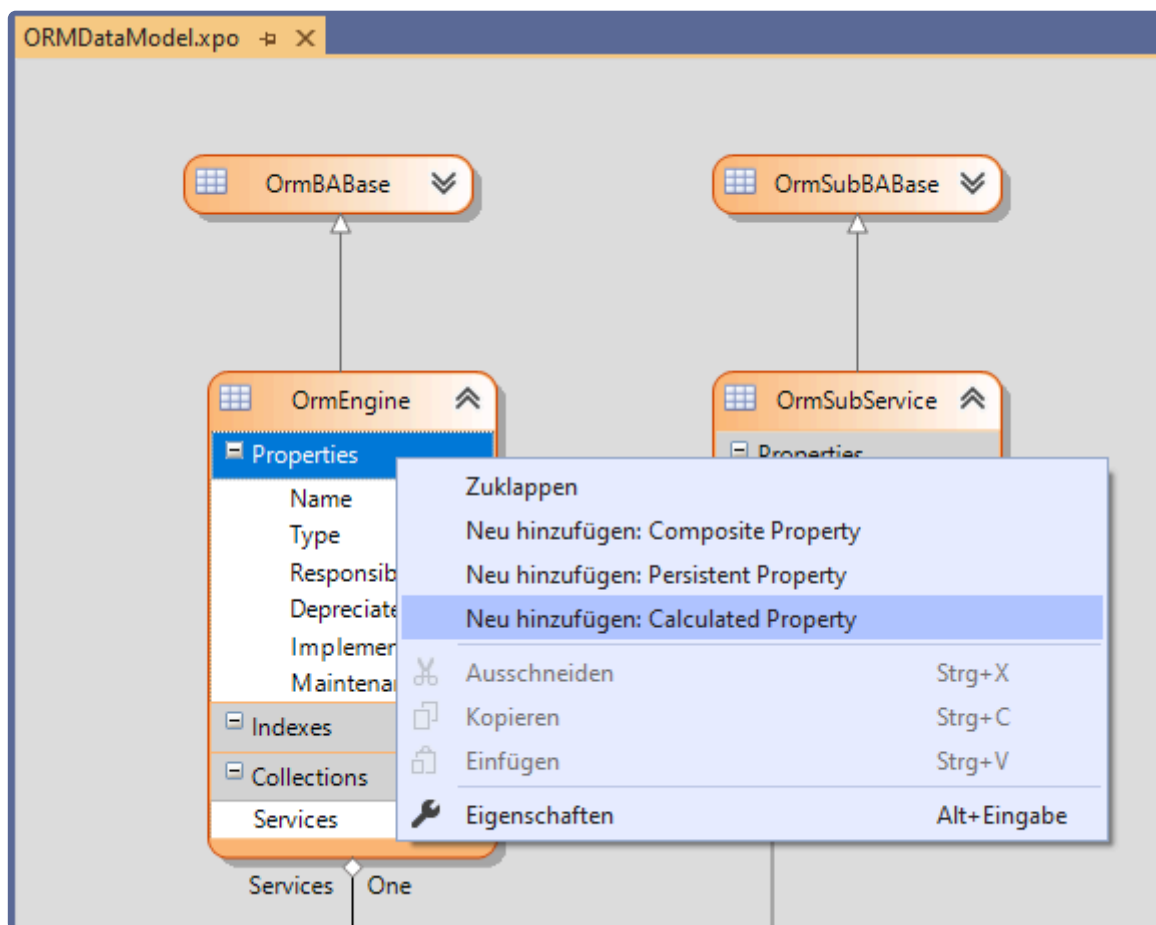
Diese Feld erlaubt den Zugriff auf die aktuelle Liste der Anhänge dieses Feldes. Wie programmatisch mit den Anhängen umgegangen wird finden Sie [hier](#).

5.2.9. Berechnete Spalten

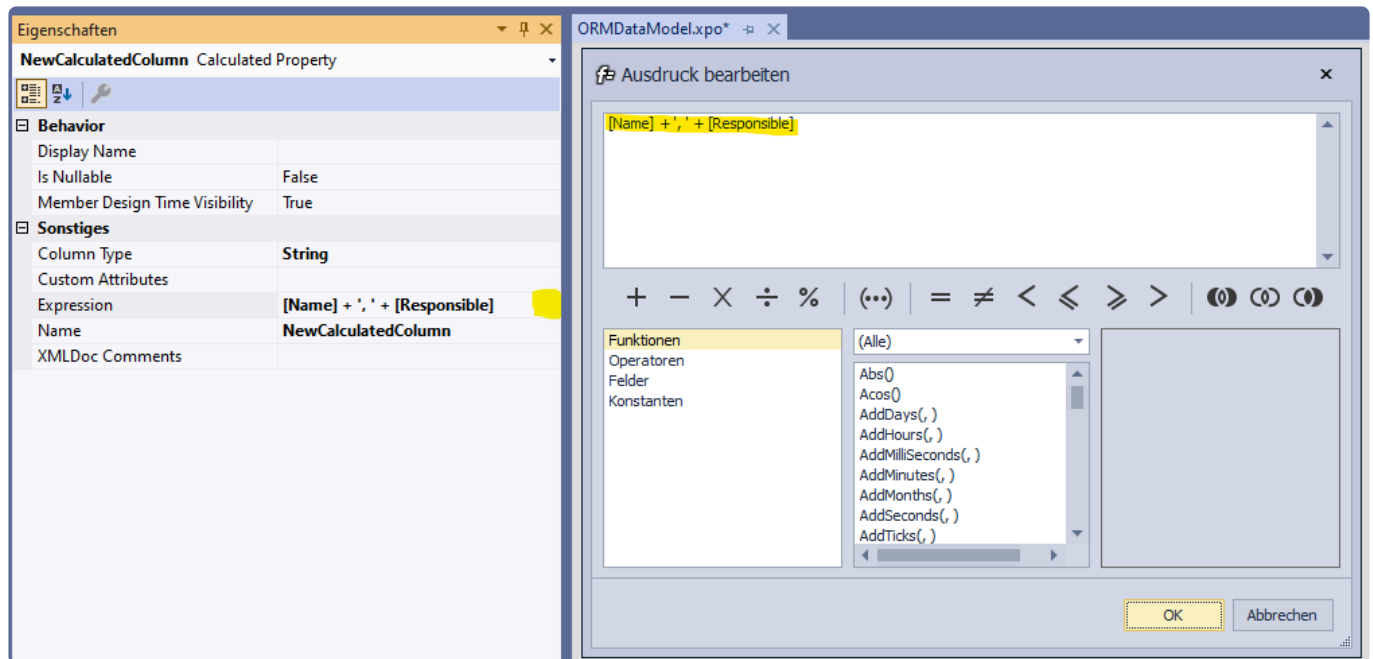
Neben normalen, persistierten Datenbankfeldern können Datentabellen auch berechnete Felder enthalten. Deren Werte werden zur Laufzeit immer neu berechnet, sind also immer aktuell. Das bedeutet aber auch dass komplexe Berechnungen mit Zugriffen auf fremde Datensätze die Performance bei der Nutzung dieser Felder erheblich beeinträchtigen können.

Die Berechnungsformel wird als Formel angegeben. Es können alle BA-Funktionen einschließlich eventueller Erweiterungen in Projekten verwendet werden, sofern dort nicht explizit Einschränkungen angegeben sind.

Mit einem rechten Mausklick kann ein "Calculated Property" erstellt werden.



Dann Datentyp und gewünschte Criteria-Formel eingeben:



Es kann nur Text, HTML Text, Boolean, Datum-Zeit und Zahlen als Datentyp verwendet werden.

5.2.10. Standardtitel und -hilfetexte definieren

Um die Standardtitel und -hilfetexte zu definieren, muss in den "Custom Attributes" weitere Attribute hinzugefügt werden

Standardtitel

- Custom Attributes: `System.ComponentModel.DisplayName("Übersetzungs-GUID")`

Standardhilfetext

- Custom Attributes: `BA.Core.CustomAttributes.HelpText("Übersetzungs-GUID")`

5.3. Teil-Datentabelle

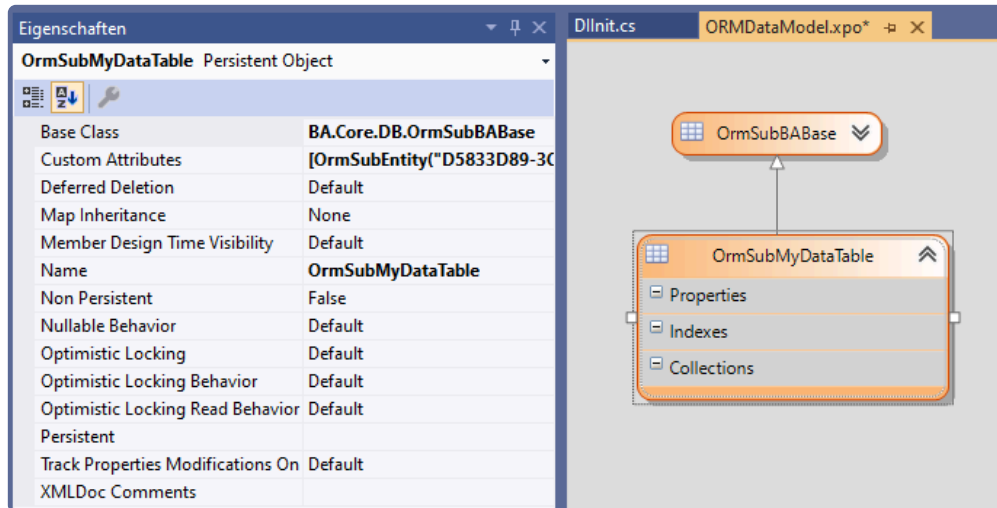
Es ist möglich neue Teil-Datentabellen zu definieren. Das Erweitern vorhandener Teil-Datentabellen ist zur Zeit nur konfiguratativ möglich. Die programmatische Definition ist dann sinnvoll, wenn Funktionalitäten mit konkreten Teil-Datentabellen und Feldern umgehen können müssen. Eine programmatische Anlage stellt dies sicher und kann vom Konfigurator auch nicht mehr verändert werden. Aber auch mit Feldern die konfiguratativ 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. Diese Klasse muss als `abstract` definiert werden, da sichergestellt werden muss, dass von diesem Objekt keine Instanzen erstellt werden können.



Bis auf wenige Ausnahmen können dieselben [Feldtypen](#) wie bei den Haupt-Datentabellen verwendet werden.

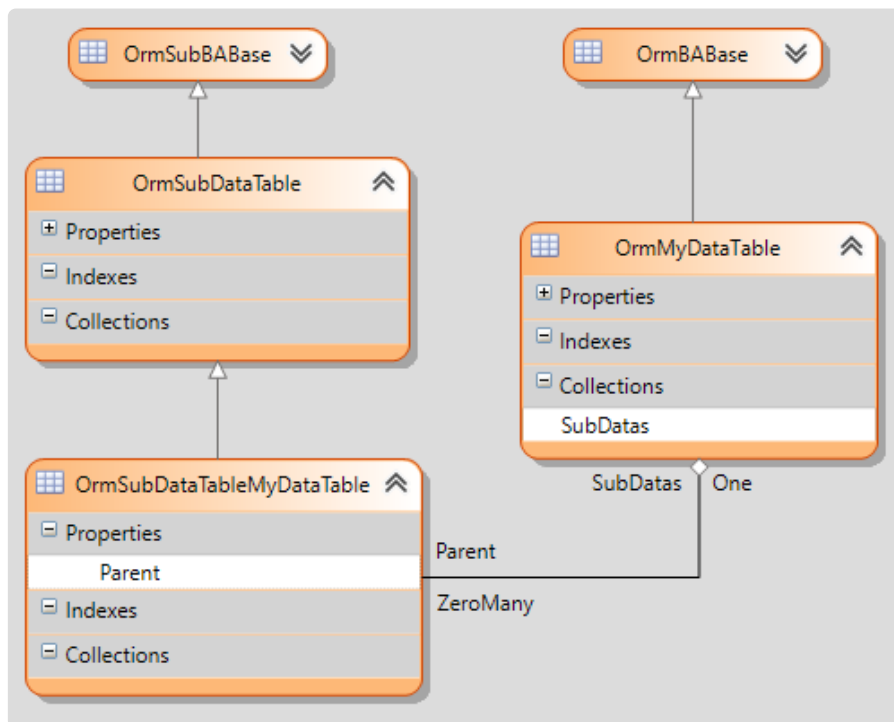
Um die neue Teil-Datentabelle dem System bekannt zu machen muss auf dieser ein Attribut gesetzt werden.

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

Damit steht die Teil-Datentabelle zur Verfügung und kann konfiguratativ 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
- Delayed: "Enabled: True; Update Modified Only = True"

The screenshot shows the 'Edit Delayed Property' dialog box. It has the following fields and values:

- Enabled:** True
- Update Modified Only:** True
- Field Name:** (empty text box)
- Group Name:** (empty text box)

At the bottom, there are 'Ok' and 'Cancel' buttons.

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`

Optional kann man nun in den Eigenschaften der automatisch erzeugten Collection in der Haupttabelle einen oder mehrere Validator setzen. Mit dem Validator für Teildatensätze `SubRecordFieldAttribute` kann man die minimale und maximale Anzahl festlegen, sowie einen Eindeutigenschlüssel definieren.

- Custom Attributes: `BA.Core.CustomAttributes.SubRecordField(0, 0, "ExampleFieldName")`



Als eindeutigen Schlüssel für Teildatensätze können ausschließlich Einzelwertauswahlfelder verwendet werden.

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 EnumCommonFields(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. 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](#). Die Klasse muss aber `abstract` definiert sein, damit nicht zufällig Datensätze davon angelegt werden können

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

Damit ist schon eine Basis-Tabelle definiert.



Existierende Basis-Datentabellen können programmatisch nicht erweitert werden.

5.6. 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.

5.6.1. 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(regularExpression, System.Text.RegularExpressions.RegexOptions)
```

IBAN Validator

Attribute

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

BIC Validator

Attribute

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

5.6.2. 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, ValidationContext validationContext)
    {
        if (value == null)
            return ValidationResult.Success;

        ErrorMessage = "";
        if (value is string valueString && !valueString.StartsWith(StartsWith))
        {
            return ValidationAttributeHelper.CreateValidationResult("[INSERT TRANSLATION GUID]", ErrorMessage, validationContext, false);
        }

        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, EnumConfigurationType.OrmSubEntityConfigurationGuid, ShowInToolbox = true)]
public class MyValidator : FieldValidatorBase
{
}
```



```
// Kann in der Konfiguration gesetzt werden
[DisplayName("Beginnt mit")]
[HelpText("Erklärung von 'Beginnt mit'")]
[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 configuration = null, EnumDesignerValidationState validationState = null)
{
    List<ValidationModel> result = base.IsValid(configuration, validationState);

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

    return result;
}
}
```

Attribut mit Steuerelement verknüpfen

Mit `GetValidationAttribute()` verknüpft man das Steuerelement mit dem Attribut. Möchte man das Attribut schon in der programmierten Datentabelle 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 MyValidator()  
    {  
        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), EnumConfiguratio  
nType.OrmEntityConfigurationGuid)]
```

5.7. Automatische Nummernvergabe

Um Nummerkreise zu verwenden, muss man einen Nummernkreis definieren, in dem man einen Auswahllistenwert vom Typ `EnumNumberRange` [anlegt](#).

Per Attribut

Dieser Nummerkreis kann dann bei der [Felddefinition](#) mit dem Attribut `[BA.Core.CustomAttributes.NumberRangeInitializer(EnumExecuteAt.OnXXX, EnumNumberRange.MyValueGuid)]` verwendet werden.

Der erste Parameter definiert den Zeitpunkt der Nummernvergabe:

- `OnCreated`: Die Nummer wird sofort beim erstellen eines neuen Datensatzes vergeben. Sie ist damit in einer Maske sofort sichtbar.
Falls der Datensatz nicht gespeichert wird, ist die Nummer aber verbraucht.
- `OnSaving`: Die Nummer wird erst bei (ersten) Speichern des Datensatzes vergeben, und zwar nur dann, wenn bis dahin keine Nummer auf einem anderen Weg eingetragen wurde, das Feld also noch leer ist. Eine Maske mit einem neuen Datensatz zeigt ein leeres Feld.

Der zweite Parameter wählt den gewünschten Nummernkreis.

Über weitere, optionale Parameter kann die Art der Nummernvergabe beeinflusst werden. Siehe implizite Dokumentation der Parameter in VisualStudio.

Per expliziter Programmierung

Soll die Nummernvergabe komplett selbst programmiert werden, z.B. in einem [ORM-Event](#), so kann statt dessen an der gewünschten Stelle die Funktion `Api.ORM.GetNextNumber(...)` verwendet werden. Eine damit abgerufene Nummer gilt sofort als verbraucht, auch wenn noch nichts gespeichert wurde.

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.DefaultSession();  
session = Api.ORM.NewSession();
```

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.NewUnitOfWork();  
session = uow;  
uow.CommitChanges();
```

5.8.1. Erstellen und Bearbeiten

ORM Datensatz erstellen

Um einen neuen Datensatz zu 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 Datentabelle.

```
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.FirstValue))
    // Löscht alle Werte
    newMyData.ClearEnumValues(nameof(OrmMyDataTable.EnumField));

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

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

// Holt alle Werte (Änderungen an der Liste haben keine Auswirkungen)
IEnumerable<EnumMyEnum> enumValues = newMyData.GetEnumValues<EnumMyEnum>(nameof(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.SortOrder == 0).FirstOrDefault();
```

Mit einem bestimmten unique Key

```
OrmSubDataTableMyDataTable keySubData = newMyData.SubDatas.Where(ff => ff.EnumContains(nameof(OrmSubDataTable.EnumField), EnumMyEnum.FirstValue)).FirstOrDefault();
```

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.ValueGuid, "[INSERT ORM GUID"].ToGuid(), session);  
OrmMyDataTable myData = Api.ORM.GetOrm<OrmMyDataTable>("[INSERT ORM GUID"].ToGuid(), 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<OrmMyDataTable>(session);
```

Mit [LINQ to XPO](#) kann man nun diese Abfragen gestalten.

```
queryWith = queryWith.Where(ff => ff.TextField != null && ff.TextField.Contains("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>(session);  
IQueryable<OrmActivityBase> activitiesWith = Api.ORM.GetQueryWithReadPermissions<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(OrmContact)) || 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<OrmBABase>(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.TextField, 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, dass 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. Dateianhänge

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("Dateiname.txt",  
data, nameof(MyDataTable.MyAttachments), session: orm.Session);  
orm.AddAttachment(attachment, false);  
orm.Save();
```

Abfragen

Einzelne Datei

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

Alle Dateien eines Feldes

```
IReadOnlyList<OrmAttachment> attachments = orm.DefaultAttachments;
```

Alle Dateien

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

Eingebettete 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);
```

5.8.5. Klonen

Es gibt eine generische Methode zum Klonen von Datensätzen: `OrmBABase.Clone()`.

Diese Methode erstellt logisch gesehen **eine Kopie** der ORMs. Die Kopie enthält standardmäßig

- alle **persistenten Eigenschaften**,
- alle **Teildatensätze**, ebenfalls als Kopie, da diese fest zu einem ORM gehören,
- alle **Attachments** und
- alle **Source-Relationen**. Die dabei referenzierten Objekte werden natürlich nicht kopiert.

Nicht geklont werden:

- Die Record History,
- MigrationID und
- andere mit ``[PropertyNotClonable]`` gekennzeichnete Ausnahmen.

Wenn man diese Funktionalität nutzen möchte, muss man darauf achten, dass für den gewünschten ORM-Typ gegebenenfalls Ausnahmen mit ``[PropertyNotClonable]`` definiert werden müssen. Das ist vor allem bei ID-Properties aus externen Systemen (Beispiel MigrationID) oder bei programmierten XPO-Relation zu anderen, klonbaren Typen zu empfehlen.

Auch Teildatensätze selbst können geklont werden (`OrmSubBABase.Clone`). Dabei wird ein neuer Teildatensatz erstellt, der zunächst **nicht** zu einem Eltern-Datensatz gehört. Das muss danach über das Hinzufügen zu einer Teildatensatz-Collection erledigt werden.

5.8.6. .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(EnumDataSourceExtension.MyDataTable.ValueGuid);  
IQueryable<OrmBABase> ormsWithGuid = Api.ORM.GetQueryWithReadPermissions(cachedType.Guid, session);  
  
IReadOnlyList<OrmDataSourceCacheModel> cachedTypes = Api.ORM.GetOrmTypeCacheValuesByBase(BA.Activity.Enums.Extensions.EnumDataSourceExtension.ActivityBase.ValueGuid);  
Type[] types = cachedTypes.Select(ff => ff.Type).ToArray();  
IQueryable<OrmBABase> ormsWithType = Api.ORM.GetQueryWithReadPermissions<OrmBABase>(session, types);
```

5.8.7. URL Links erzeugen

Es gibt eine API-Methode, die die URL zum Öffnen des angegebenen Datensatzes erzeugt:

BA.API.Api ORM.GetRecordUrl()

Die Methode hat einen zwingenden Parameter, die OID des Datensatzes: **recordOid**.

Die folgenden Parametern sind optional:

ormType – Id des Orm-Typs.

form – Masken-ID, in der der Datensatz geöffnet werden soll.

openInDialog – Indikator, dass der Datensatz im Dialog geöffnet werden soll.

forceWeb – Erzwingt das Öffnen der Webanwendung anstelle der mobilen Anwendung. Override, wenn die Url von einer mobilen Plattform aus geöffnet wird.

absolutePath – Indikator, ob der generierte Link relativ oder absolut (einschließlich des Basis-URL-Teils) sein wird.

Die Struktur des Links sieht wie folgt aus:

```
OpenRecord/recordOid[?optionalParameters]
```

Bei der Erstellung der URL werden die Rechte des Benutzers nicht berücksichtigt. Es wird nicht geprüft, ob der aktuelle Benutzer das Recht hat, die angegebene Maske oder die Standardmaske zu verwenden, und auch nicht, ob der Benutzer den angegebenen Datensatz sehen darf. Die Prüfung der Rechte erfolgt dann zur Laufzeit, wenn über die URL der Datensatz geöffnet wird.

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, dass diese Funktionalitäten performant implementiert sind.

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

Ausnahmen

Erwartete Ausnahmen, sollten im Rahmen von Orm Events als [TranslatableException](#) geworfen werden.

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)
- OnCloned(OrmBABase clone, OrmBABase original)
- OnSaving(OrmBABase orm)
- OnSaved(OrmBABase orm)
- OnRelationsCreated(OrmBABase orm)
- OnLoaded(OrmBABase orm)
- OnMovingToRecycleBin(OrmBABase orm)
- OnMovedToRecycleBin(OrmBABase orm)
- OnRestoring(OrmBABase orm)
- OnRestored(OrmBABase orm)
- OnDeleting(OrmBABase orm)
- OnDeleted(OrmBABase orm)
- OnCloned(OrmBABase orm)



In ORM-Events vor dem Speichern (`OnSaving`, `OnDeleting`) dürfen jetzt auch Exceptions geworfen werden, die dann dem Benutzer angezeigt werden und das Speichern verhindern. Um übersetzbar zu sein, sollte man die Klasse `TranslatableException` nutzen.

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 ausgeführt. Weitere Validierungsfehler fügt über `ValidationResults` auf dem Datensatz hinzu.

```
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.ValidationResults.Add(new ValidationContext(myDataTable)
{ MemberName = nameof(OrmMyDataTable.TextField) }, "[INSERT TRANSLATION GUI
D]");
        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, OrmSubRecordBase subRecord)
{
    OrmMyDataTable myDataTable = (OrmMyDataTable)orm;
    if (propertyName == nameof(OrmMyDataTable.SubDatas))
    {
        OrmSubDataTable subData = (OrmSubDataTable)subRecord;
    }
}

public override void OnSubRecordRemoved(OrmBABase orm, string propertyName, OrmSubRecordBase 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, object 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 subTable, string propertyName, string subTablePropertyName, int entryIndex, object 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);
}
```

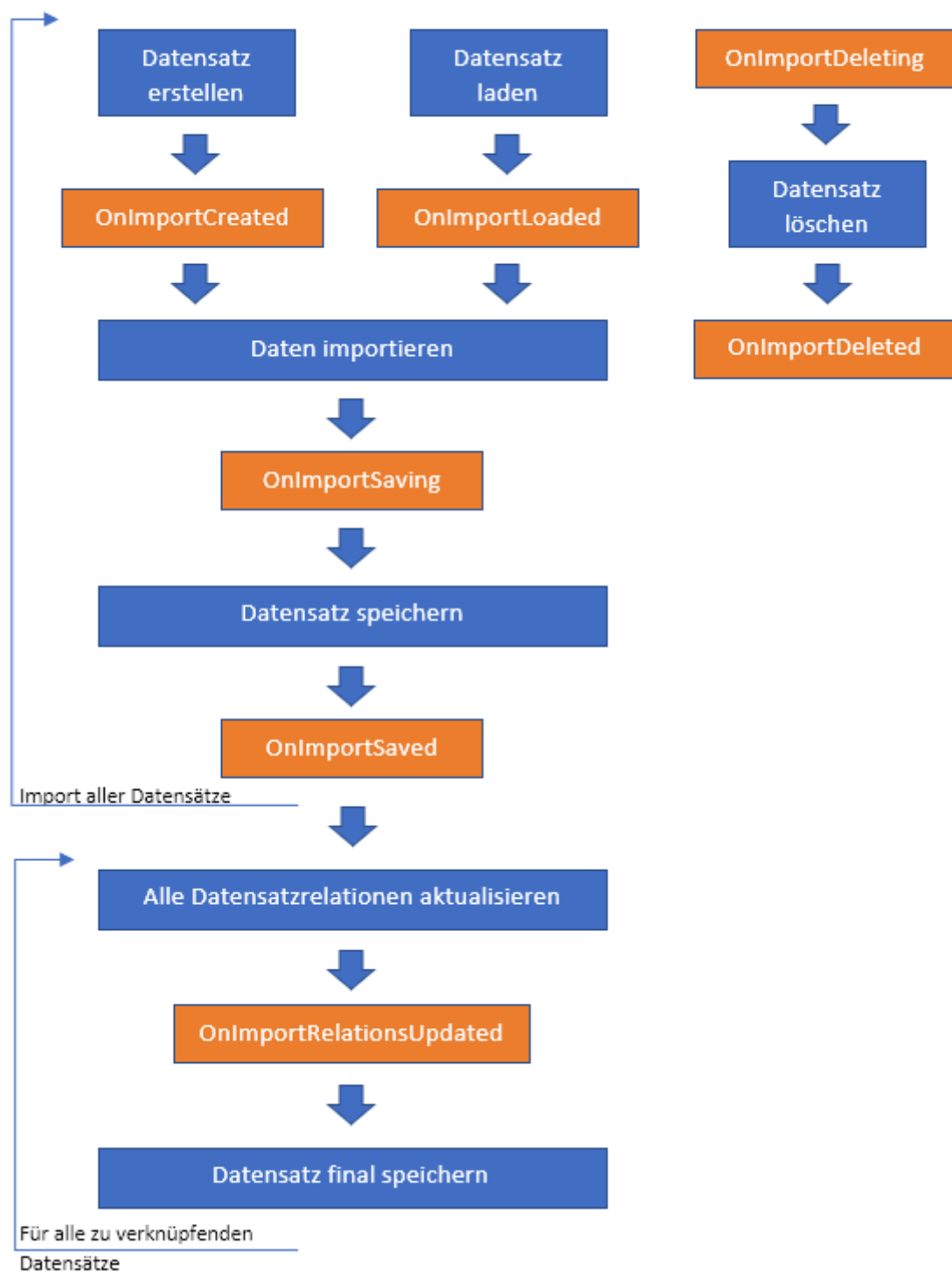
Folgende Events stehen zur Verfügung

- **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 Attribut 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(CreateAuthorRelationOfOrmBABase), typeof(CreateAuthorRelationOfOrmBABaseOnImport) })]  
public class OrmEventModifyMyDataTable : BAOrmEventsBase {}
```

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

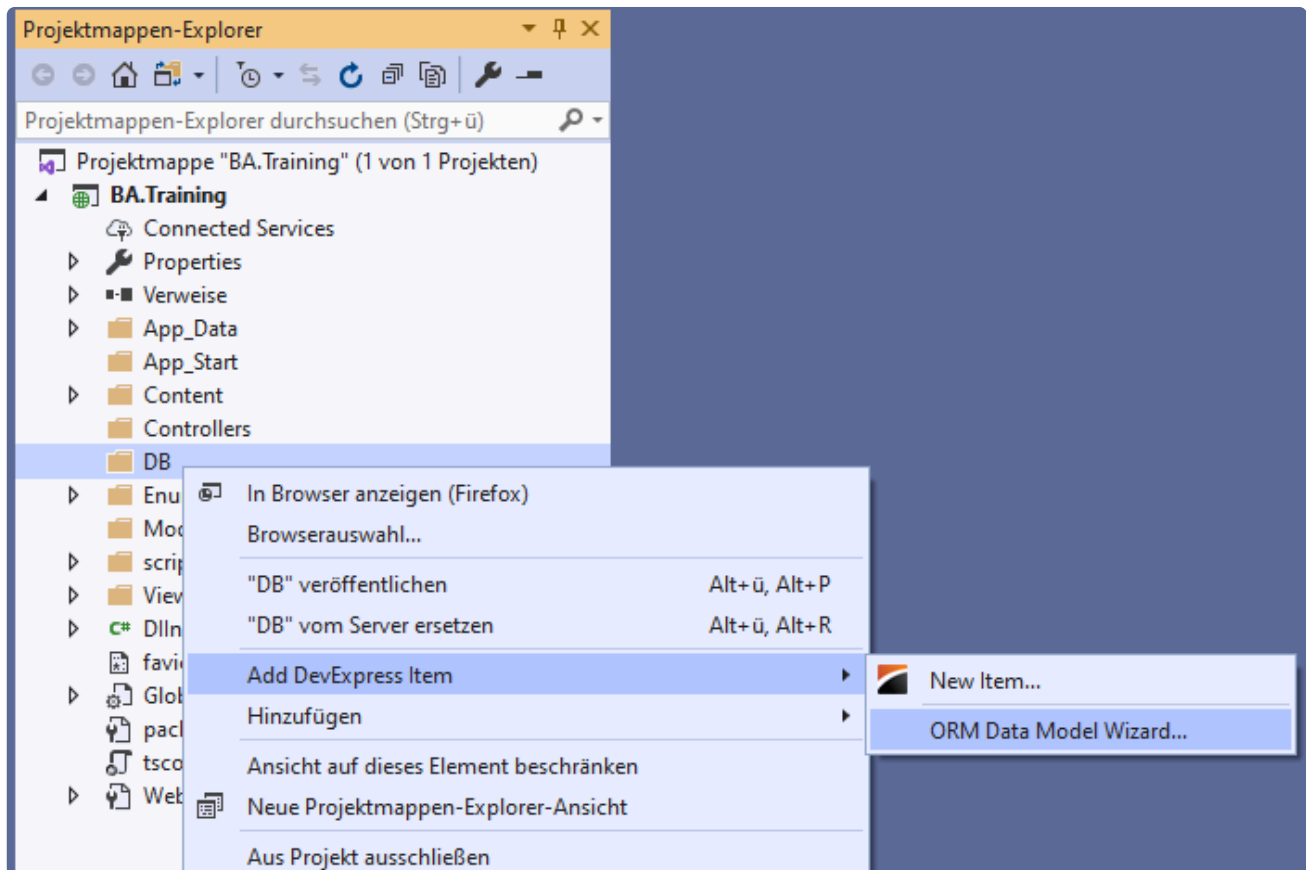
- BA.Core.OrmEvents
- BA.Activity.OrmEvents
- BA.Contact.OrmEvents
- etc.

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
- Legen Sie eine Seite an.
 - Fügen Sie das Widget "Navigationsmenü" hinzu und tragen Ihre Applikationsnavigation ein.
 - Tragen Sie die Seite in den Anwendungseinstellungen unter Startseite 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

Programmatrischer 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 gesetzte 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 EnumRelationType  
(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, RelationType  
Configuration> 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 Kontaktnamen.

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. Beispielsweise in #OrmMyDataTable#
Maximale Anzahl == 1, Keine Relationskategorie

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

Beispiel 2

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

```
[Newtonsoft.Json.JsonIgnore]
[DontShowFieldInDesigner]
[RelationDefinitionTarget("[INSERT RELATION DEFINITION GUID EXAMPLE 2]", EnumRelationTypeExtension.MyRelationGuid, EnumMyRelationCategory.FirstValueGuid, MinCount = 0, MaxCount = 0)]
public IQueryable<OrmContact> RelatedContacts
{
    get {
        return GetSources<OrmContact>(EnumRelationTypeExtension.MyRelation, EnumMyRelationCategory.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. Beispielsweise 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, EnumMyRelationCategory.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), "RelatedConformation", "[INSERT RELATION DEFINITION GUID]", EnumRelationTypeExtension.MyRelation, 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), "RelatedConfirmedContacts", "[INSERT RELATION DEFINITION GUID]")]
[assembly: RelationDefinitionSourceInject(typeof(OrmEmail), "RelatedConfirmedContacts", "[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), "RelatedConformation", "[INSERT RELATION DEFINITION GUID]", EnumRelationType.ConformationGuid, MinCount = 0, MaxCount = 1, Validate = false)]
[assembly: RelationDefinitionSourceInject(typeof(OrmActivityBase), "RelatedConfirmedContacts", "[INSERT RELATION DEFINITION GUID]")]
```

Standardtitel und -hilfetext

Auch bei den Relationsdefinitionen können Standardtitel und -hilfetexte definiert werden. Dazu gibt es entsprechende eigenschaften an den Attributen `RelationDefinitionTarget` und `RelationDefinitionTargetInject`.

6.3. Hinzufügen, Setzen 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, otherOrmGuid);
// Falls man das Orm zur Verfügung hat. Ansonsten ist dies langsamer
OrmMyDataTable otherOrm = Api.ORM.GetOrm<OrmMyDataTable>(otherOrmGuid, session);
RelationData relationDataOrm = Api.ORM.GetRelationDataModel(otherOrm);
```

Es gibt Methoden für Abfragen und Listen von Guids.

```
IQueryable<OrmMyDataTable> queryWith = Api.ORM.GetQueryWithReadPermissions<OrmMyDataTable>(session);
queryWith = queryWith.Where(ff => ff.TextField != null && ff.TextField.Contains("Inhalt"));
IQueryable<RelationData> relationDatasQuery = Api.ORM.GetRelationDataModels(queryWith);
IQueryable<RelationData> relationDataGuids = Api.ORM.GetRelationDataModels<OrmActivityBase>(session, new[] { guid1, guid2 });
```

Hinzufügen

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

```
newMyData.AddSource(relationDataOrm, EnumRelationTypeExtension.MyRelation, null);
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, relationDataNewMyData, EnumRelationTypeExtension.MyRelation, null);
relation.Save();
```

Falls möglich sollte diese Methode in der Massenverarbeitung eingesetzt werden.

Setzen

Mit `SetSource` können alle Relationen einer Relationstyp / -kategorie-Kombination auf einmal zugewiesen werden. Das bedeutet, dass für diese Kombination der Datensatz nur noch exakt diese Relationen hat.

```
newMyData.SetSource(relationDataOrm, EnumRelationTypeExtension.MyRelation, null);  
newMyData.Save();
```



Eine Methode, um eine Liste von Quellen zu setzen, ist ebenfalls vorhanden

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, null);  
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, newMyData.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>(EnumRelationTypeExtension.MyRelation);
Guid? companyOid = newMyData.GetSourcePrimaryOid(EnumRelationTypeExtension.MyRelation);
IQueryable<OrmBABase> sources = newMyData.GetSources<OrmBABase>(EnumRelationTypeExtension.MyRelation);
List<Guid> sourcesOids = newMyData.GetSourcesOids(EnumRelationTypeExtension.MyRelation);
```

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

```
IQueryable<OrmMyDataTable> targets = company.GetTargets<OrmMyDataTable>(EnumRelationTypeExtension.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.Oid, session, EnumRelationTypeExtension.MyRelation);
IQueryable<OrmMyDataTable> targets = Api.Relation.GetTargets<OrmMyDataTable>(company.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 angegeben werden.

6.5. Hierarchieabfragen

Hierarchische Relationen erlauben performante Abfragen über die Hierarchie. In der `Api.Relation` gibt es Methoden um Quelldatensätze über die gesamte Hierarchie zu finden. Die Angabe von Typen ist dabei optional.

Der nächste Quelldatensatz

Der nächste Datensatz bestimmter Typen

```
OrmBABase firstCompany = Api.Relation.GetHierarchyPrimarySourceNearest(EnumRelationType.Parent, BA.Contact.Enums.Extensions.EnumDataSourceExtension.Company, null, session, newMyData.Oid);
OrmBABase firstCompanyOrContact = Api.Relation.GetHierarchyPrimarySourceNearest(EnumRelationType.Parent, new Guid[] { BA.Contact.Enums.Extensions.EnumDataSourceExtension.Company, BA.Contact.Enums.Extensions.EnumDataSourceExtension.Contact }, null, session, newMyData.Oid);
```

Die Oid des nächsten Datensatz bestimmter Typen

```
Guid firstCompanyOid = Api.Relation.GetHierarchyPrimarySourceNearestOid(EnumRelationType.Parent, BA.Contact.Enums.Extensions.EnumDataSourceExtension.Company, null, session, newMyData.Oid);
Guid firstCompanyOrContactOid = Api.Relation.GetHierarchyPrimarySourceNearestOid(EnumRelationType.Parent, new Guid[] { BA.Contact.Enums.Extensions.EnumDataSourceExtension.Company, BA.Contact.Enums.Extensions.EnumDataSourceExtension.Contact }, null, session, newMyData.Oid);
```

Der oberste Quelldatensatz

Der oberste Datensatz bestimmter Typen

```
OrmBABase topCompany = Api.Relation.GetHierarchyPrimarySourceTop(EnumRelationType.Parent, BA.Contact.Enums.Extensions.EnumDataSourceExtension.Company, null, session, newMyData.Oid);
OrmBABase topCompanyOrContact = Api.Relation.GetHierarchyPrimarySourceTop(EnumRelationType.Parent, new Guid[] { BA.Contact.Enums.Extensions.EnumDataSourceExtension.Company, BA.Contact.Enums.Extensions.EnumDataSourceExtension.Contact }, null, session, newMyData.Oid);
```

Die Oid des obersten Datensatz bestimmter Typen

```
Guid topCompanyOid = Api.Relation.GetHierarchyPrimarySourceTopOid(EnumRelation
```

```
Type.Parent, BA.Contact.Enums.Extensions.EnumDataSourceExtension.Company, null, session, newMyData.Oid);
Guid topCompanyOrContactOid = Api.Relation.GetHierarchyPrimarySourceTopOid(EnumRelationType.Parent, new Guid[] { BA.Contact.Enums.Extensions.EnumDataSourceExtension.Company, BA.Contact.Enums.Extensions.EnumDataSourceExtension.Contact }, null, session, newMyData.Oid);
```

Komplexe Abfragen

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

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

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

```
IQueryable<Guid> phonecallsOids = hierarchyRelationQuery.Where(ff => ff.Source == newMyData.Oid && ff.TargetType == EnumDataSource.PhoneCall).Select(ff => ff.Target);
IQueryable<OrmPhoneCall> phonecallsWithRead = Api.ORM.GetQueryWithReadPermissions<OrmPhoneCall>(session);
IQueryable<OrmPhoneCall> phonecalls = phonecallsWithRead.Where(ff => phonecallsOids.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.

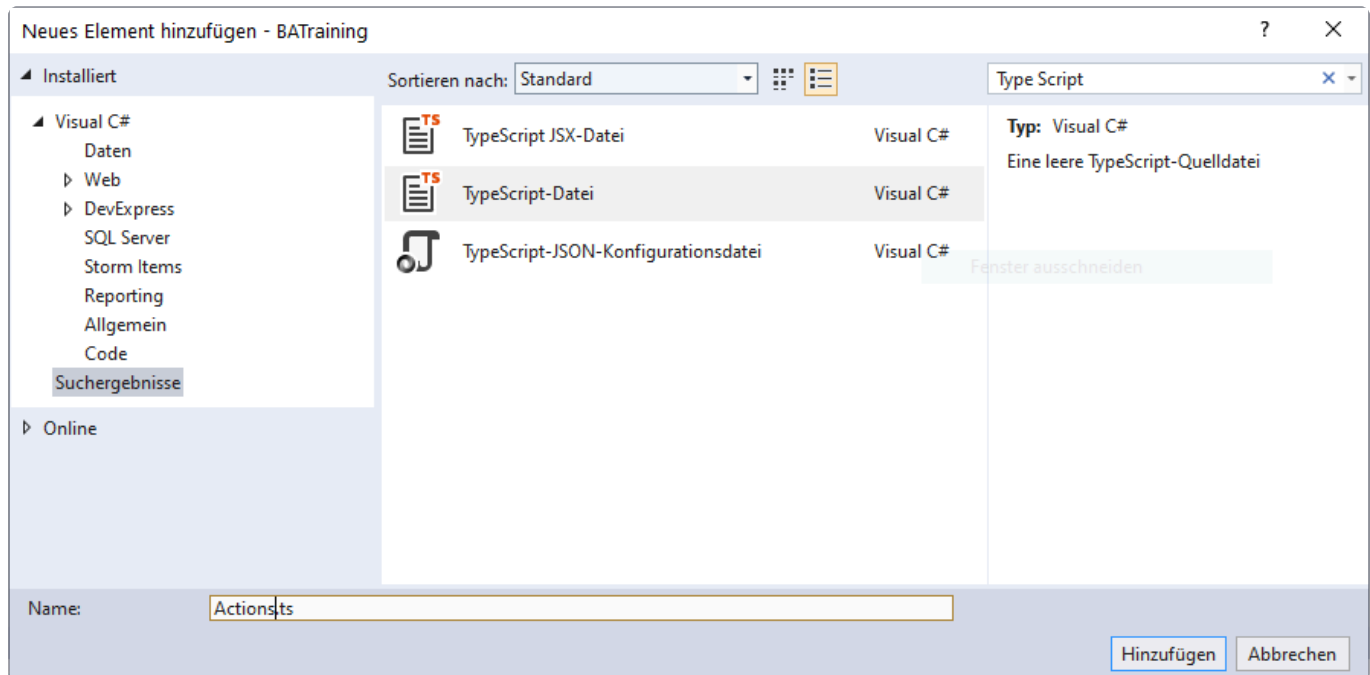
[Lösung](#)

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

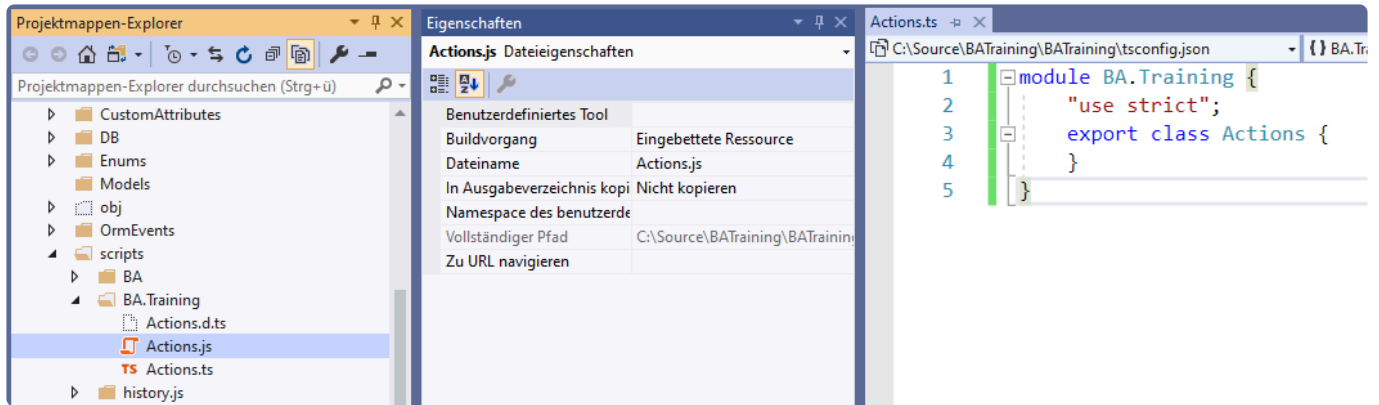
Zur Anlage und Integration der ersten Bibliothek sollte ein eigener Ordner unterhalb des “Scripts” Ordner angelegt werden. In diesem Ordner wird die neue Type Script Bibliothek erstellt.



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 die Klasse `BABundleConfigBase` erweitern und im Standard-Namespace des Projektes liegen. Das wesentliche Bundle in BA ist `"~/bundles/scripts"`, dort wird die Java Script Datei hinzugefügt.

```
internal sealed class BundleConfig : BABundleConfigBase
{
    public override void RegisterBundles()
    {
        ScriptBundle("~/bundles/scripts").Include("~/scripts/BA.Training/Actions.js");
    }
}
```

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



Achten sie bei `Bundle.Include()` 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. UI Events

BA bietet mit dem `EventManager` eine einfache Möglichkeit für Projekte im Browser Ereignisse aller Art auszulösen und darauf zu reagieren. Von dem Basissystem werden eine Reihe von Standardevents automatisiert ausgelöst, die entsprechend verwendet werden können.

Standard-Events

Für die Standard-Event-Typen stehen clientseitig Konstanten in der Klasse `BA.Ui.Event.StandardEventTypes` zur Verfügung.

Die zur Verfügung stehenden Events sind:

- `FieldUpdate` reserviert, bitte über die im nachfolgenden beschriebenen API "Feldänderung eines Datensatzes" nutzen
- `TabOpening` Wird gefeuert, wenn ein Tab geöffnet werden soll. Der Rückgabewert der Handler entscheidet, ob der Tab dann tatsächlich geöffnet wird.
- `TabOpened` Wird gefeuert, nachdem ein Tab geöffnet und dessen Inhalt vollständig geladen wurde.
- `TabChanging` Wird gefeuert, wenn der aktuelle Tab in der UI gewechselt wird.
- `TabActivated` Wird gefeuert, wenn der aktuelle Tab in der UI gewechselt wurde.
- `TabClosing` Wird gefeuert, wenn ein Tab geschlossen werden soll. Der Rückgabewert der Handler entscheidet, ob der Tab dann tatsächlich geschlossen wird.
- `TabClosed` Wird gefeuert, wenn ein Tag final geschlossen wurde.
- `RecordSaving` Wird gefeuert, wenn ein Datensatz gespeichert werden soll. Der Rückgabewert der Handler entscheidet, ob der Datensatz dann tatsächlich gespeichert wird.
- `RecordSaved` Wird gefeuert, wenn ein Datensatz erfolgreich gespeichert wurde.
- `RefreshGrids` Wird gefeuert, wenn die Menübandaktion "Ansichten aktualisieren" betätigt wird.

Alle Tab-Events bekommen in den Daten das betreffende Tab-Objekt unter dem Schlüssel `Tab` geliefert. Ausnahme hiervon ist `TabOpening`, weil zum Aufrufzeitpunkt noch kein Tab-Objekt existiert; dieser Event bekommt alle Kontextinformationen geliefert, die in der Funktion `OpenTab` zur Verfügung stehen:

```
let eventData: CustomData = {
    url: baseUrl + tmpUrl,
    willActivate: tmpUrlLwr.indexOf("andredirect") < 0,
    overrideTabReference: overrideTabReference,
    asLastVisibleTab: asLastVisibleTab,
    isUniqueTab: isUniqueTab,
    predecessor: predecessor,
};
```

Alle Save-Events bekommen die `Oid` des betreffenden Datensatzes unter dem Schlüssel `Oid` und den `Orm`-Typen unter `OrmType` geliefert.

```
let eventData: CustomData = {  
    oid: oid,  
    ormType: ormType,  
};
```

Einfaches Beispiel (in der Browser-Konsole nachstellbar):

```
BA.Ui.Event.EventManager.AddEventHandler(BA.Ui.Event.StandardEventTypes.TabOpening, "myHandlerUniqueId", (data) => { console.log(data); return true; }, null);
```

"myHandlerUniqueId" entspricht einer eindeutigen Id des eigenen Event-Handlers. Kann beispielsweise bei der Deregistrierung verwendet werden. Genauere API Beschreibung folgt.

Öffentliche API / Eigene Events

Auf der TypeScript-Klasse `BA.Ui.Event.EventManager` stehen folgende Funktionen zur öffentlichen Verwendung zur Verfügung:

```
public static AddEventHandler(type: string, id: string, handler: Function(data: CustomData), scope: string = null): void
```

Die Funktion prüft zunächst, ob bereits ein Handler mit derselben Id registriert ist und entfernt diesen, falls dies der Fall ist.

Das heißt im Umkehrschluss, dass immer nur ein Handler mit derselben Id registriert sein kann. Dies gilt für die gesamte Applikation, die in einem Browser-Fenster geöffnet ist.



Es können keine Events zwischen unterschiedlichen Browser-Fenstern angemeldet werden.

Die Handler-Funktion ist eine Funktion mit einem Parameter des Typs `CustomData`, welcher ereignisspezifische Daten beinhaltet, welche beim Triggern des entsprechenden Events übergeben werden können. Die Handler-Funktion wird automatisch in dem Fensterkontext (iFrame) ausgeführt, aus welchem heraus sie registriert wird.

Der Parameter `scope` dient zur Beschränkung der Gültigkeit eines Events beispielsweise auf den aktuellen Tab.

Hierfür kann die Gültigkeit eines Events z.B. auf die Id des aktuellen Tabs eingeschränkt werden. Diese Einschränkung sollte dann bei `TriggerEvent` ebenfalls mit berücksichtigt werden.

```
public static RemoveEventHandler(id: string): boolean
```

Diese Funktion deregistriert den konkreten Event-Handler mit der gegebenen Id und gibt `true` zurück, sollte dies gelungen sein.

Wenn unter der gegebenen Id keine Handler-Funktion registriert ist, gibt die Funktion entsprechend `false` zurück.

Es ist nicht zwingend notwendig, alle Handler innerhalb eines Tabs zu deregistrieren, wenn der Tab geschlossen wird; dies erfolgt automatisch.

Sollten Ereignisse im Kontext von Dialogen angemeldet worden sein, so müssen diese beim Schließen des Dialogs vom Entwickler deregistriert werden.

```
public static TriggerEvent(type: string, scope: string = null, data: CustomData = null): boolean
```

Diese Funktion führt zum Aufruf aller Event-Handler, welche für den gegebenen Typen und den gegebenen Gültigkeitsbereich `scope` registriert sind.

Die Reihenfolge, in der die Handler-Funktionen ausgeführt werden, ist undefiniert.

Der Rückgabewert dieser Funktion ist `true` solange alle Handler-Funktionen `true` zurückgeben. Gibt nur eine der Handler-Funktionen `false` zurück, ist der Rückgabewert hier auch `false`.

Alle Handler-Funktionen bekommen dasselbe Datenobjekt `data` übergeben.

Beispiel

Einfaches Beispiel welches in der Browser-Konsole nachstellbar ist.

Schritt 1: Registrieren zweier Event-Funktionen für einen eigenen Event-Typen mit unterschiedlichen Geltungsbereichen

```
BA.Ui.Event.EventManager.AddEventHandler("MyEventType", "myUniqueId-1", (data) => { console.log(data); return true; }, top.tabController.ActiveTab.Guid);
BA.Ui.Event.EventManager.AddEventHandler("MyEventType", "myUniqueId-2", (data) => { console.log(data); return true; }, "ownScope");
```

Schritt 2.1: Triggern eines konkreten Events

```
BA.Ui.Event.EventManager.TriggerEvent("MyEventType", "ownScope", {mydata: "Event was triggered!"});
```

Es wird nur der Event-Handler ausgeführt, der unter dem angegebenen Geltungsbereich registriert wurde.

Schritt 2.2: Triggern aller Events

```
BA.Ui.Event.EventManager.TriggerEvent("MyEventType", null, {mydata: "Event was triggered!"});
```

Weil der angegebene Geltungsbereich null ist, werden BEIDE Event-Handler ausgeführt.

Schritt 3: Deregistrieren der Event-Handlers

```
BA.Ui.Event.EventManager.RemoveEventHandler("myUniqueId-1")
BA.Ui.Event.EventManager.RemoveEventHandler("myUniqueId-2")
```

Eigene Events auswählbar in der Konfiguration

Die Applikation beinhaltet eine Auswahlliste, in welcher alle Event-Typen, die unterstützt werden, hinterlegt sind. Diese Liste heißt `BA.Core.Enums.EnumUiEventType`.

Die hier beinhalteten Werte dienen der Auswahl von Events in entsprechenden Steuerelementen und generell nur zur serverseitigen Verwendung.

Die Enum-Werte selbst besitzen eine Eigenschaft, die besagt, ob sie in der UI auswählbar sein sollen, und den Typ-String, welcher in der UI als Event-Typ verwendet wird.

Dieser darf nicht doppelt vorkommen. Eine Überprüfung findet aber nicht statt. Bei Erweiterung der Auswahlliste sollten daher passende Prä- oder Postfixes verwendet werden und sollten nur aus Buchstaben bestehen.

Feldänderung eines Datensatzes

Die `EventManager` Klasse bietet eine spezielle API für die Änderung der Felder. Soll auf Feldänderungen reagiert werden oder wird ein eigenes [Maskensteuerlement](#) implementiert, muss diese API verwendet werden.

```
public static AddFieldUpdateHandler(fieldName: string, handler: Function(data: UpdateFieldEventData), scope: string): string
```

Diese Funktion registriert einen Event-Handler, der bei einer Wertänderung des gegebenen Feldes in der UI automatisch getriggert wird. Sie gibt eine generierte Id zurück, welche zusammen mit `RemoveFieldUpdateHandler` verwendet werden kann, um den Handler zu deregistrieren.

Die übergebenen Daten der Klasse `UpdateFieldEventData` beinhalten folgende Werte:

- `Type: string` Typ des gefeuerten Events (üblicherweise in diesem Fall `"FieldUpdate"`)
- `FieldName: string` Name des geänderten Feldes (abstrahierter Feldname, ggf mit Teildatensatzbezeichner `"Addresses.PostalCode"` aber ohne Dialog-Prefix und ohne Teildatensatzindex)
- `Context: Window` Kontext der Handler-Funktion
- `Scope: string` Gültigkeitsbereich des Event-Handlers
- `FullFieldName: string` Voller Name des geänderten Feldes (mit Index bei Änderung eines Feldes auf einem Teildatensatz `"Addresses[0].PostalCode"` oder mit Dialog-Prefix in Dialogen `"DLG1.Addresses[0].PostalCode"`)
- `AdditionalData: CustomData` zusätzliche Informationen (im Standard leer)


```
public static RemoveFieldUpdateHandler(fieldName: string, scope: string, id: string = null): boolean
```

Deregistriert einen konkreten Handler unter Ferwendung des zugehörigen Feldnamens und der von AddFieldUpdateHandler zurückgelieferten Id.

```
public static TriggerFieldUpdated(fieldName: string, scope: string, fullFieldName: string = null, additionalData: CustomData = null): boolean
```

Triggert alle Feldaktualisierungs-Handler für ein bestimmtes Feld. Diese Funktion wird im Standard automatisch bei allen Änderungen von editierbaren Feldern aufgerufen.



Wird bei einem Relationsauswahlsteuerelement eine Änderung vorgenommen, so wird im Feldnamen die Guid der geänderten Relationsdefinition (string, upper-case) übergeben.

7.4. 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 beispielsweise

- `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 beispielsweise

- `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 beispielsweise

- `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 serialisierbar 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(nameof(NavigationConfiguration.NavigationConfigurationType), ExpressionType.Equal, EnumNavigationConfigurationType.RibbonNavigationGuid, EnumControlFilterApplyState.OptIn)]
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."; // Translation Guid verwendbar
    VisibilityForParentTypes.Add(EnumActionVisibleForParentType.Form);
}
```

Mit dem Objekt `VisibilityForParentTypes` kann man steuern für welche Objekte (Masken / Ansichten / Detailansichten) 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.

```
ActionMethodId = "BA.Training.ClientActionMyAction";
```

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.")] // Übersetzungs GUID verwendbar
public string Message { get; set; }
```

Übertragung von Werten an die Type Script Funktion

In der Methode `AdditionalRibbonButtonAssignment` können Daten in das `AdditionalClientData` 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(RibbonButtonItem ribbonItem, Dictionary<string, object> additionalClientData, EnumActionVisibleForParentType parentType, DevExUIModelBase uiModel, OrmBABase orm)
{
    base.AdditionalRibbonButtonAssignment(ribbonItem, additionalClientData, parentType, uiModel, orm);
    additionalClientData["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.

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.

Rechte bei einer Maskenaktion

Aktionen sind häufig nur dann möglich, wenn der Benutzer bestimmte Rechte besitzt. Beispielsweise Bearbeitungsrechte auf einem Datensatz. Falls dies bei der eigenen Aktion der Fall ist, muss im Konstruktor folgende Zeile hinzugefügt werden:

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

Benötigte Client Daten

```
additionalClientData["UserHasRole"] = [true/false];
```

Beispiel für die Bearbeitungsrechte

```
additionalClientData["UserHasRole"] = orm?.UserCanEdit() ?? true;
```

Rechte in einer Ansichtenaktion

Im Gegensatz zu einer Aktion in einer Maske, kann in einer Ansicht nicht die Rechte auf einen konkreten Datensatz geprüft werden. In diesem Fall werden die Rechte des Anwenders auf der Datenbanktabelle geprüft und abhängig von den selektierten Datensätzen und deren Typ, die Aktion aktiv/inaktiv geschaltet. Werden alle Datensätze der Ansicht selektiert, wird automatisch angenommen, das auch alle Typen die in der Ansicht dargestellt werden können, markiert sind. In diesem Fall wird keine Prüfung aller Datensätze vorgenommen.

Falls dies bei der eigenen Aktion der Fall ist, muss im Konstruktor folgende Zeile hinzugefügt werden:

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

Benötigte Client Daten

```
additionalClientData["TypesUserCanHandle"] = [Komma separierte Liste der Typ-Guid mit Großbuchstaben];
```

Beispiel für Bearbeitungsrechte

```
// Die Typen aller in der Ansicht konfigurierten Datentabellen ermitteln
// Basistypen werden dabei aufgelöst
IEnumerable<OrmDataSourceCacheModel> ormDataSources = Api.ORM.GetOrmTypeCacheValuesByBase(GuidSet.ParseGUIDs(gridConfiguration.OrmDataSource));

// Keine Basistypen (Von denen können keine Datensätze existieren) und alle Datentabelle auf die der Anwender Bearbeitungsrechte prinzipiell besitzen könnte
ormDataSources = ormDataSources.Where(ff => !ff.IsBaseType && Api.User.CurrentUserIsAllowed(ff.Guid, EnumTableOperations.Delete));

// Umwandeln in eine Komma separierte Guid-Liste mit Großbuchstaben
additionalClientData["TypesUserCanHandle"] = GuidSet.JoinGUIDs(ormDataSources.Select(ff => ff.Guid)).ToUpper();
```

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["TestValueContains"] = "Hallo";
additionalClientData["TestFormContains"] = "Subject";
```

Maskenwert beinhaltet einen definierten Wert nicht

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

```
additionalClientData["TestValueNotContains"] = "Hallo";  
additionalClientData["TestFormNotContains"] = "Subject";
```

Clientside Funktion liefert True zurück

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

```
additionalClientData["IfMethodReturnsTrueMethod"] = "BA.Customer.Project.Ui.Utils.CheckVisibility";
```

Type Script:

```
public static CheckVisibility(menuItem: BAClientRibbonItem | BAClientDetailGridMenuItem, customData: CustomData) { ... }
```

Sichtbar in Masken und/oder Ansichten

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

Selektierte Datensätze sind von bestimmten Datetabellen

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

```
additionalClientData["TypesUserCanHandle"] = allowedTypes.ToString().ToUpper();
```


8.2. Verarbeitung von selektierten Datensätzen

Damit eine Aktion alle selektierten 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<OrmTempSelectedRecords>(session);
        query = query.Where(ff => ff.TaskExecutionId == TaskExecutionId);
        // Laden der Guids
        IQueryable<Guid> guids = query.Select(ff => ff.SelectedRecordOid);
        // Laden der Relationsdaten
    }
```

```

        IQueryable<RelationData> relationData = query.Select(ff => new RelationData() { Oid = ff.SelectedRecordOid, OrmType = ff.SelectedRecordType });
        // Laden von beliebigen Datensätze
        IQueryable<OrmBABase> orms = Api.ORM.GetQuery<OrmBABase>(session).Where(ff => query.Select(q => q.SelectedRecordOid).Contains(ff.Oid));
        // Laden von konkreten Datensätze
        IQueryable<OrmMyDataTable> myDataTable = Api.ORM.GetQuery<OrmMyDataTable>(session).Where(ff => query.Select(q => q.SelectedRecordOid).Contains(ff.Oid));

        // 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, dass 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(igniterParams);
    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, BA.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.Client Communication` beeinflusst werden.

- `userId` 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
- `unlimitedTimeToDisplay` 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 userId, String message, String title = "", String url = "", String linkText = "", Boolean unlimitedTimeToDisplay = false, String evalString = null, uint displayTimeout = ErrorTime, 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 userId, String message, String 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 message, String title = "", String url = "", String linkText = "", Guid? oid = null)
```

Beispiele

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

Falls man anstatt der `Oid` den `OrmUser` angibt.

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

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.CurrentUserGuid(), message1);
Api.ClientCommunication.CreateSuccess(Api.User.CurrentUserGuid(), message2, oid: toastGuid);
Api.ClientCommunication.HideToast(toastGuid);
```

Clientseitig

Zur Anzeige von Toasts wird das Framework [Toastr](#) verwendet. Die TypeScript 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?: ToastrOptions, oid?: BA.Classes.Guid): BA.Classes.Guid
public static Warning(message: string, title?: string, overrides?: ToastrOptions, oid?: BA.Classes.Guid): BA.Classes.Guid
public static Info(message: string, title?: string, overrides?: ToastrOptions, oid?: BA.Classes.Guid): BA.Classes.Guid
public static Success(message: string, title?: string, overrides?: ToastrOptions, 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  
    function 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: CustomData, 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", paramter, 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: ASPxClientButtonEventArgs)
```

```
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: ASPxClientButtonEventArgs)
```

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.FormHiddenDataModel
```

Serverseitig

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

```
public void OpenDialog(String dialogIdentifier, object dialogParameter, object 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 Auswahllisten 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.DialogDefaultCancel"}, {Title: "My other Button", Method: "BA.Ui.Dialog.DialogTools.DefaultComboBoxControlCallback"}]
```

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:"EnumEmailAddressType", Preselected: ["12FE59CA-196D-42D6-A9B1-3D31DD16F5E7"], AddButtons: [{Title: "MyButton1", Method: "BA.Ui.Dialog.DialogManager.DialogDefaultCancel"}, {Title: "My other Button", Method: "BA.Ui.Dialog.DialogTools.DefaultComboBoxControlCallback"}] },
    {custom: „data“,
    function (result: BA.Ui.Dialog.DialogResult) { console.log(result); console.log(result.Data); }
    );
```

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:" 6B1CFDAF-C222-45FE-932F-CEE5B7E89FA4"},
    {custom: „data“,
    function (result: BA.Ui.Dialog.DialogResult) { console.log(result); console.log(result.Data); }
    );
```

Checkbox-Liste

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

Listbox

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

Combobox

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

Combobox (Mit eigenem Daten Provider)

```
BA.Ui.Dialog.DialogManager.OpenDialog(
    "BA.Core.Dialogs.Combobox",
    { DialogTitle: "Combobox", Caption: "Bitte wählen", ModelType: "DPStandardUserProfileList", IsEnumDialog: false, "DataProvider": "76739A3A-67C5-44AD-A3CF-B1F573E8F3D1"},
    {custom: "Data"},
    function (result: BA.Ui.Dialog.DialogResult) { console.log(result); console.log(result.Data); }
);
```

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: string) { }  
}
```

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(customData); }
);
```

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.FormDialogModel();
            formModel.RecordId = selected[0] as string;
            formModel.RefreshGridOnSuccess = "GridView";

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

```

        // 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 uiMode
l, OrmBABase orm)
{
    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, or
m);
}

```

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 Dialoges
        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 Parameter 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, parameter, buttonId,
                                                    bindObject, propertyPrefix);
        return result;
    }
}
```

Die Klasse muss zwei Voraussetzungen erfüllen

1. Sie muss die Klasse `DialogImplementationBase` erweitern.

2. Über das Attribut `DialogImplementation` wird der Dialogidentifizier 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.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");
}
```

Mit Hilfe eines Datenmodells 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";

    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");
}

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.
- **IsEnabledOptional**: Soll der Button aktiv sein (Vorgabe: true)
- **HintText Optional**: Hinweistext / Tooltip wenn die Maus über dem Button ist.

HandleAction

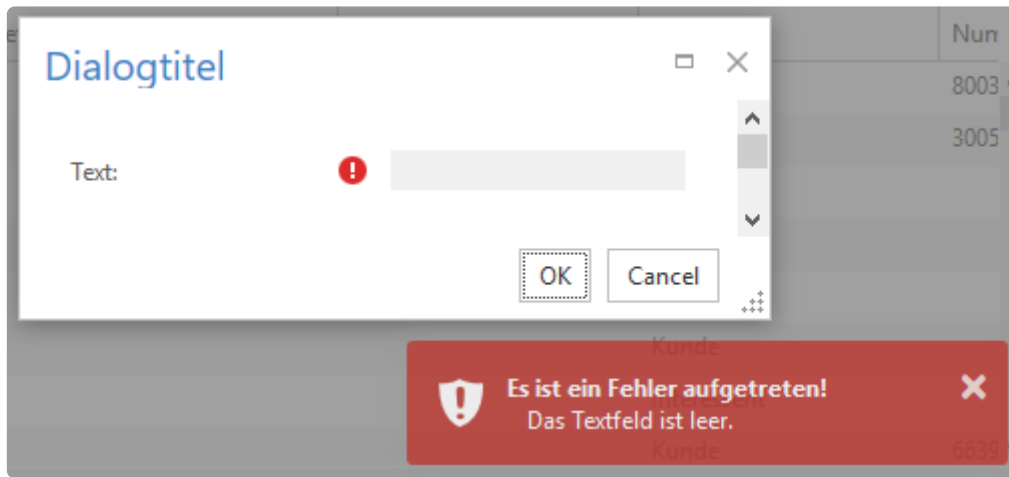
In **HandleAction** werden die Buttons verarbeitet, die auf **serverCallback=true** stehen. Per Default wird der Dialog geschlossen. Ein anderes Verhalten wird an dieser Stelle implementiert. In dem Beispiel wird das Feld **TextEdit** validiert und eine entsprechende Fehlermeldung ausgegeben.

```

public override DialogResultModel HandleAction(HttpRequestBase request, ModelStateDictionary modelState, Dictionary<String, Object> parameter, String buttonId, object bindObject, string propertyPrefix = "")
{
    DialogResultModel result = base.HandleAction(request, modelState, parameter, buttonId, bindObject, propertyPrefix);
    if (buttonId == DialogButtonIds.OkButton)
    {
        MyDialogModel dlgData = (MyDialogModel)bindObject;
        if (String.IsNullOrEmpty(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: ASPxClientButtonClickEventArgs) {
    let textBoxName: string = BA.Ui.Dialog.DialogManager.GetDialogControlName(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)

The screenshot shows a wide dialog box titled "Auswahlwert bearbeiten". It features a two-column layout where labels are positioned to the left of the input fields. The fields include a dropdown for "Auswahlwert" (set to "Algerien"), a spinner for "Sortierreihenfolge*" (set to "2"), a checkbox for "Aktiv status*" (checked), and several text boxes for "Bild:", "Wähle Berechtigung...", "ISO Code:" (set to "DZ"), "Anrede:", "Adresse:" (set to "{Address}\n{Addition...}"), "Briefanrede (männlich):", "Briefanrede (weiblich):", "Briefanrede (neutral):", and "Briefgruß:". At the bottom right are "Aktualisieren" and "Abbrechen" buttons.

Mittlerer Dialog (Zweispaltig / Labels über dem Feld)

The screenshot shows a medium-width dialog box titled "Auswahlwert bearbeiten". The layout is similar to the wide version but with labels positioned above the input fields. The "Adresse:" field contains the text "{Address}\n{AdditionalAddress1}\n". The "Aktiv status*" checkbox is checked. The "Aktualisieren" and "Abbrechen" buttons are at the bottom right.

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, MaxWidth = 700 });
layout.AddBreakpoint(new BreakpointRule() { Name = "M", ColumnCount = 2, MaxWidth = 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, MaxWidth = 300 });
group.AddBreakpoint(new BreakpointRule() { Name = "M", ColumnCount = 2, MaxWidth = 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 Standarddialogs 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.

Zwei Methoden können überschrieben werden, um in abgeleiteten Dialogen Änderungen an den erstellten Controls und an den erstellten MVCxFormLayoutItems zu machen:

```
void AfterContentCreated(DevExpressFormModel formModel, HttpRequestBase request, ModelStateDictionary modelState, Dictionary<string, object> parameter, object bindObject)
```

Abfolge ist:

1. CreateDialogContent
2. Übernahme von `formModel.DataSource` als `bindObject`
3. AfterContentCreated

Das bedeutet, dass man in `AfterContentCreated` Änderungen an `this.Controls` durchführen kann, bevor die Controls in MVCx-Items umgewandelt werden.

Theoretisch erfüllt ein Überschreiben von `CreateDialogContent` mit Aufruf von `base.CreateDialogContent` den gleichen Zweck, man spart sich bei der Verwendung von `AfterContentCreated` allerdings, dass man an den base-Aufruf denken muss, d.h. `AfterContentCreated` ist die offizielle Schnittstelle, um Änderungen an den vom Dialog erstellten Controls zu machen.

```
void AfterControlsRendered(DevExpressFormModel formModel, HttpRequestBase request)
```

Hier können Änderungen an den umgewandelten MVCx-Items gemacht werden.

Diese finden sich unter `formModel.LayoutPanels` (Liste von `DevExpressFormPartModel`) und sind nach ihrer Hierarchie angeordnet (d.h. wenn man nicht weiß, nach was man wo sucht, muss man wahrscheinlich zunächst `FlattenControls` bemühen müssen).

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 Checkboxes

```
// Control
EnumCheckboxListControl cblast = 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 rblast = 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 TypeScript 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))  
{ }
```

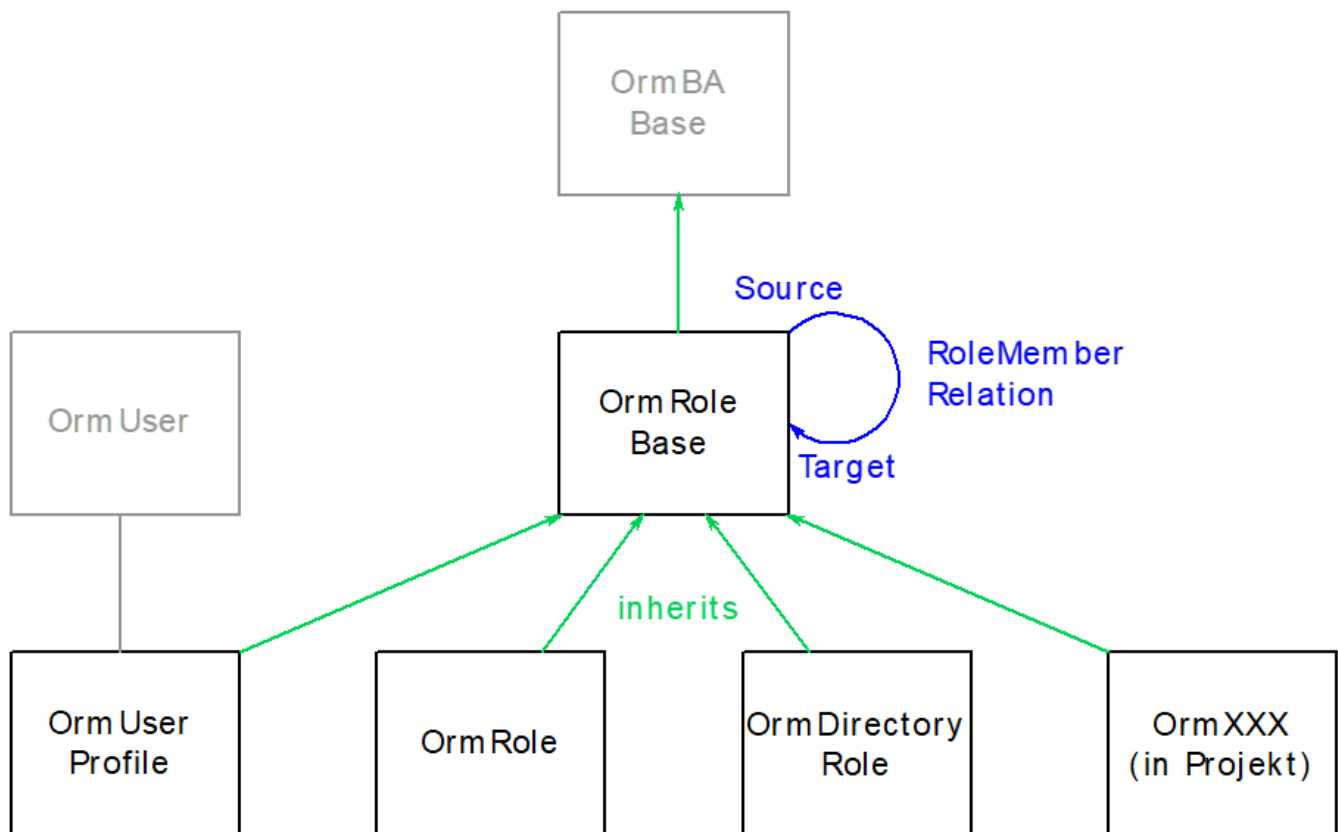
Wenn man einen Thread startet, kennt dieser den aktuellen Benutzer nicht. Einen Thread mit den Berechtigungen erhält man mit

```
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 Pflégbarkeit 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, dass 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 action) .
```

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, EnumTableOperations.Create);
```

Es können auch mehrere Berechtigungen auf einmal geprüft werden:

```
bool userCanCreateAndEdit = Api.User.CurrentUserIsAllowed(EnumDataSource.Company, 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, IEnumerable<Type> dataSources = null) where T : OrmBABase
IQueryable<OrmBABase> Api.ORM.GetQueryWithReadPermissions(Guid type, Session session, IEnumerable<Type> dataSources = null)
IQueryable<OrmBABase> Api.ORM.GetQueryWithReadPermissions(Type ormType, Session 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.GetQueryWithReadPermissions<OrmActivityBase>(session);
```

Wenn der gewünschte Datentyp dynamisch aus der Konfiguration kommt:

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

Wenn mehrere Datentypen dynamisch aus der Konfiguration kommen:

```
IEnumerable<Guid> configuredTypeGuids = ...;
IEnumerable<Type> types = configuredTypeGuids.Select(ff => Api.ORM.GetOrmTypeCacheValue(ff).Type);
IQueryable<OrmBABase> records = Api.ORM.GetQueryWithReadPermissions<OrmBABase>(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 session, 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 des aktuellen Benutzer prüfen

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

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

Erstere prüft, ob der 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(EditorRolesString), true);
```

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

Effektive Berechtigungen eines beliebigen Benutzers

Um alle effektiven Berechtigungen eines beliebigen Benutzers (oder auch einer beliebigen Rolle) *schnell* abzufragen gibt es eine API-Funktion. Diese liefert ein Set aller Rollen-Guids, die ein Benutzer effektiv,

also auch inklusive geerbter Rollen hat. Das Ergebnis enthält immer auch den Benutzer selbst sowie die Everyone-Rolle.

```
if (!Api.User.GetEffectivePermissions(userProfileGuidToCheck).Overlaps(controls.AllowedRoles))  
    // Permission denied
```



Die Funktion sollte nicht dazu verwendet werden, um eine Berechtigung des aktuell angemeldeten Users zu prüfen! Dafür sollte `Api.User.CurrentUserIsInRole` verwendet werden, das die Rollen zum Zeitpunkt des Logins prüft und nicht potentielle Änderungen seit dem.

Der Aufruf von `GetEffectivePermissions(Guid)` kann manchmal einen Datenbankzugriff auslösen. Die Methode sollte daher nicht in einer Schleife aufgerufen werden. In diesem Fall ist die Überladung `GetEffectivePermissions(IReadOnlyCollection<Guid>)` zu bevorzugen, die dasselbe mit maximal einem DB-Zugriff erledigt.

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.CurrentUserGuid()))
        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 auf den Startzeitpunkt des nächsten, noch nicht zur Ausführung bereiten Items gesetzt, falls eines existiert.

Abarbeitung

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

- sich ein Item beendet. Dann prüft er, ob ein anderes den freiwerdenden Prozess nutzen kann oder ein davon abhängiges Item nun gestartet werden darf.
- ein Item neu eingeplant oder aktualisiert wird. Dann prüft er, ob es sofort bereit zu Ausführung ist und ob dafür ein Prozess frei ist.
- ein Item abgebrochen werden soll. Dann signalisiert er dem Item, den Wunsch abubrechen, und setzt seinen Status auf `CancellingByUser`.
- 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.

Ende

Wenn die Anwendung beendet oder neu gestartet wird, versucht der Workmanager alle noch laufenden Prozesse zu beenden. Das geschieht in mehreren Schritten:

1. Zuerst wird das Starten neuer Prozesse unterbunden. (Es können aber noch neue Prozesse geplant werden.)
2. Dann bekommen alle noch laufenden Prozesse ein asynchrones Signal zum Abbrechen (`WorkItemSh`

ouldFinish) und den Status ShutdownRequest.

3. Wenn alle Prozesse darauf reagiert haben, wird dem IIS signalisiert, dass die Anwendung jetzt beendet werden kann.
4. Wenn sich ein Prozess *nicht binnen 1 Minute* nach dem Signal aus Punkt 2 beendet, wird die Anwendung trotzdem beendet. Der Prozess wird dabei hart abgebrochen.

Hinweis: Bei einem Anwendungsneustart kann die neue Anwendungsinstanz erst starten, wenn die vorherige beendet wurde. Aufgrund des o.g. Szenarios kann dieser Vorgang um bis zu 1 Minute verzögert werden. Die neue Anwendung bleibt dann im Status "Waiting for old application instance to terminate." hängen.

11.1.1. 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 Anwendungsfall.

- 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 richtet sich nach `ExpectedRunTime` sowie der geplanten Startzeit.

Zustände von Work-Items

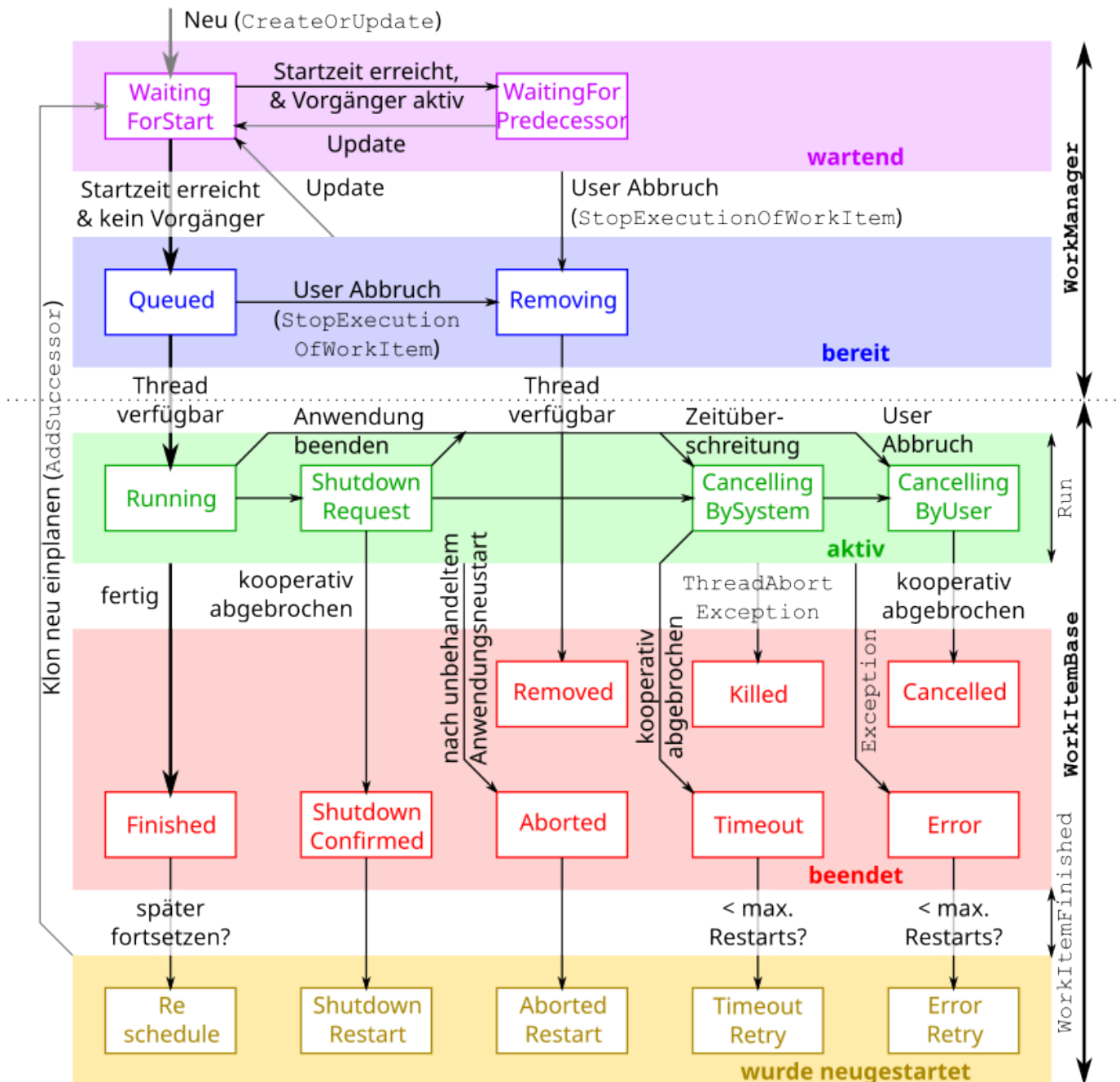
- Bei allen Zuständen, bei denen das `WorkItem` noch nicht (=nie) ausgeführt wurde, gehört die Instanz dem Work-Manager. Nur er darf Veränderungen vornehmen.
- Bei allen Zuständen ab dem Start der Ausführung gehört die Instanz dem `WorkItem` selbst. Nur dieses darf noch Veränderungen vornehmen.
- Der Zustand von Instanzen von Work-Items ist immer ein gerichteter Graph. Wenn an einem noch nicht aktiven Work-Item eine Veränderung vorgenommen wird, wird eine neue Instanz erstellt und die alte gelöscht.
- Die Zustände gehören zu Kategorien, die sich bezüglich vieler Eigenschaften gleich verhalten.

Kategorie	Zustand	Beschreibung
Warten auf Startbedingung	WaitingForStart	Das Work-Item liegt in der Datenbank und wartet bis der Zeitpunkt für seine Ausführung gekommen ist.
Warten auf Startbedingung	WaitingForPredecessor	Das Work-Item ist bereit zur Ausführung, muss aber noch auf die Beendigung eines Vorgängers warten. Dies passiert z.B. wenn ein Prozess einen Nachfolger schon eingeplant hat, selbst aber noch in <code>WorkItemFinished</code> hängt.
Warten auf Ausführung	Queued	Das Work-Item ist bereit zur Ausführung, steht aber noch in der Warteschlange für eine freie Prozess-Ressource.
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	ShutdownRequest	Die Anwendung wird gerade heruntergefahren und das Work-Item sollte sich in Kürze beenden und einen Nachfolger einplanen.
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.
Laufend	CancellingByUser	Das Work-Item hat den Befehl zum Abbrechen bekommen und sollte sich in Kürze beenden.
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	ShutdownConfirmed	Das Work-Item hat sich aufgrund eines Anwendungsneustarts vorzeitig beendet. Dieser Status ist normalerweise transient und wird sofort zu <code>ShutdownRestart</code> weiter geschaltet.
Beendet	Aborted	Das Work-Item wurde aufgrund eines Anwendungsneustarts hart abgebrochen und wird neu eingeplant. Dieser Fall tritt nur ein, wenn ein Work-Item nicht binnen angemessener Zeit auf <code>ShutdownRequest</code> reagiert hat oder der Anwendungspool abgestürzt ist. Dieser Status ist transient und wird sofort zu <code>AbortedRestart</code> weiter geschaltet.
Beendet	Timeout	Das Work-Item hat auf den <code>CancellingBySystem</code> Status reagiert und sich selbst beendet. Standardmäßig wird das Work-Item (wenige Male) mit leichtem Zeitversatz neu eingeplant. In diesem Fall ist der Status transient und wird sofort in <code>TimeoutRetry</code> weiter geschaltet. Wenn die Anzahl der automatischen Restarts überschritten ist, wird dieser Status persistent.
Beendet	Killed	Das Work-Item hat seine Arbeit nicht binnen der gesetzten Zeit beendet und wurde vom Work-Manager gekillt – gefährlich!
Beendet	Canceled	Das Work-Item wurde vom Benutzer (bzw. über die API) abgebrochen und hat sich selbst beendet.
Beendet	Error	Eine unbehandelte Ausnahme ist aufgetreten. Bei einer unbehandelten Ausnahme wird das Work-Item (wenige Male) mit leichtem Zeitversatz neu eingeplant. In diesem Fall ist der Status transient und wird sofort in <code>ErrorRetry</code> weiter geschaltet. Wenn die Anzahl der automatischen Restarts überschritten ist, wird dieser Status persistent.
Neustart	Reschedule	Das Work-Item hat sich erfolgreich beendet und einen Nachfolger eingeplant. Das kann sowohl aufgrund eines Wiederkehrenden Work-Items der Fall sein

		als auch wegen eines Timeouts oder sonstiger Limitierungen.
Neustart	ShutdownRestart	Das Work-Item hat sich aufgrund eines Anwendungsneustarts vorzeitig beendet und erneut eingeplant.
Neustart	AbortedRestart	Das Work-Item wurde aufgrund eines Anwendungsneustarts hart abgebrochen und sich (nach dem Neustart) neu eingeplant.
Neustart	TimeoutRetry	Das Work-Item wurde nach einem Timeout neu eingeplant.
Neustart	ErrorRetry	Das Work-Item wurde nach einer Exception neu eingeplant und wird seine Arbeit bald in einer neuen Instanz fortsetzen.

Statusübergänge



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 ersetzt werden.
- Für jede einmal erfolgreich eingeplante Instanz (gleiche `InstanceGuid`), die nicht durch eine erneute Einplanung aktualisiert wurde, 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 Einplanung 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.

Laufzeitüberwachung

Methode Run

Sobald ein laufendes Work-Items die maximale Laufzeit überschreiten hat oder aber die Anwendung gerade neu gestartet werden soll, bekommt es asynchron das Signal `WorkItemShouldFinish` und der Status wechselt auf `CancellingBySystem` (Zeitüberschreitung bzw. `ShutdownRequest` (Neustart)). Auf dieses Signal kann mit einer Frist von standardmäßig 1 Minute reagiert werden, indem es sich selbst beendet (Methode `Run` kehrt zurück). Der Status wechselt dann auf `TimeOut`. Standardmäßig wird die Implementierung von `WorkItemFinished` sie daraufhin `MaxNumberOfRestarts` mal neu einplanen. In letzterem Fall wechselt der Status auf `TimeoutRetry`.

Reagieren ein Work-Item nicht in der gesetzten Frist, wird der Worker Thread hart getötet, 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.

Methode WorkItemFinished

Die Laufzeit dieser Methode ist auf 1 Minute begrenzt. Wird diese Zeit überschritten, wird der Prozess hart getötet, indem eine `ThreadAbortException` injiziert wird. Der Vorgang wird im Server-Log protokolliert und, falls aktiviert, auch im Anwendungs-Log des Prozesses. Es erfolgt kein zweiter Versuch, und das Work-Item verbleibt in dem finalen Status ohne Nachfolger..

11.1.2. Queues und Priorität

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 programmierter 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 ist dafür 1 Worker reserviert.
2. **Langläufer**
Die Queue für Langläufer führt alle Arten von Work-Items (außer **Urgent**) aus. Standardmäßig stehen dafür 2 Worker 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.

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.

Nach einer erfolgreichen Erstellung oder Aktualisierung hat der Work-Manager im übergebenen Work-Item das Feld `Id` ausgefüllt. Der Wert ändert sich auch bei jeder Aktualisierung. Ob genau diese Instanz jemals ausgeführt wird, ist aber nicht gesichert, denn es könnten ja noch weitere Updates folgen.

Aktualisierungen

Wenn für ein bereits geplantes Work-Item `CreateOrUpdate` aufgerufen wird – maßgeblich ist die gleiche `InstanceGuid` – dann gewinnt immer nur der letzte Aufruf. Maßgeblich ist der Zeitpunkt des Datenbank-Commits.



Einmal laufende Work-Items können niemals umgeplant werden. Wenn für ein laufendes Work-Item `CreateOrUpdate` aufgerufen wird, dann wird dieser Aufruf ignoriert. Es kann auch kein Nachfolger “auf Vorrat” eingeplant werden, solange der Prozess noch läuft. Das Beenden eines Work-Items ist auch eine Aktualisierung, die die eben erfolgte Einplanung überschreibt.

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 `InstanceGuid`.

Nachfolger

Falls es sich um ein wiederkehrendes Work-Item handelt, muss in `WorkItemFinished` der Nachfolger

eingepplant werden, ggf. mit aktualisierten Parametern. Üblicherweise geschieht das über die Methode `workItemBase.AddSuccessor`, `CreateOrUpdate` würde aber auch funktionieren.

Es können auch beliebige andere Work-Items (Folgeprozesse) mit anderen Instance-Guids in `workItemFinished` als Nachfolger eingepplant werden, auch mehrere.

Laufendes oder geplantes Work-Item abbrechen

```
EnumWorkItemState StopExecutionOfWorkItem(Guid instanceGuid, bool cancelRunning = true)
```

Mit dieser Methode kann ein bereits eingepplantes 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 noch nicht gestartet wurde, wird die Methode `workItemFinished` mit Status `Removed` aufgerufen.
Das schließt auch den Fall ein, dass das Work-Item zwar schon bereit zur Ausführung war, aber noch keinen freien Prozess bekommen hat.
- Wenn das Work-Item schon läuft, wird der Status asynchron auf `CancellingByUser` gesetzt und es bekommt ein Signal (`workItemShouldFinish`). Darauf sollte es reagieren und sich binnen einer Minute beenden (Run-Methode kehrt zurück). Danach wechselt der Status auf `Cancelled`.

Als Rückgabewert erhält man den *vorherigen* Status des Work-Items oder `null`, wenn kein geplantes oder laufendes Work-Item mit dieser `instanceGuid` gefunden wurde.

Aus dem vorherigen Status geht auch hervor, was tatsächlich passiert ist.

Kategorie	Status	neuer Status	Aktion
Waiting	alle	Removing	Die Einplanung des Work-Items wurde widerrufen.
Ready	Queued	“	“
“	Removing	unverändert	keine (redundanter Aufruf von <code>StopExecutionOfWorkItem</code>)
Active	Running	CancellingByUser	Abbruch-Signal gesendet.
“	ShutdownRequest CancellingBySystem	“	Abbruch-Signal aktualisiert. Der Abbruch kam während eines Anwendungsneustarts oder eines Timeouts. Ersteres hat aber Priorität. Deshalb wird der Status auf <code>CancellingByUser</code> aktualisiert, was im Besonderen einen Neustart des Work-Items verhindert.
“	CancellingByUser	unverändert	keine (redundanter Aufruf von <code>StopExecutionOfWorkItem</code>)
Final Restart	alle	unverändert	keine (Beendete Prozesse können nicht mehr abgebrochen werden.)

ed			
----	--	--	--

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 `WorkItem Finished` neu einplanen.

Informationen über Work-Items

Der Status eines Work-Items wird durch die Klasse `WorkItemProgress` repräsentiert. Diese Informationen entstammen vollständig den nicht serialisierten Spalten aus `OrmWorkItem` und sind von außen nur lesbar.

Der Status eines Work-Items ist immer transient. Bei allen nicht-abgeschlossenen Work-Items können sich die Werte jederzeit ändern. Mit technischen Exceptions (Data Race) muss man beim Lesen der Eigenschaften aber nicht rechnen.

Liste von Work-Items

```
List<WorkItemProgress> GetWorkItems(IEnumerable<EnumWorkItemStateCategory> categories, Guid user = default, ICollection<Guid> instanceGuids = null)
```

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.

Das Ergebnis der Funktion ist eine Liste der passenden Work-Items nach aktuellem Stand. Aufgrund der transienten Natur von Work-Items kann sich der Status aber jederzeit ändern. Deshalb muss man auch bei einer Abfrage auf nur laufende Work-Items in Einzelfällen mit einem (gerade erreichten) finalen Status rechnen.

Parameter categories

Die folgende Liste gibt an, welche Statuskategorien abgefragt werden sollen. Zur Auswahl stehen:

- `EnumWorkItemStateCategory.Waiting` Alle Work-Items, die noch nicht bereit zur Ausführung sind.
- `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.
- `EnumWorkItemStateCategory.Active` Alle laufenden Work-Items einschließlich derer, die ein

Abbruch-Signal bekommen, sich daraufhin aber noch nicht beendet haben.

- `EnumWorkItemStateCategory.Final` Alle Work-Items, die abgeschlossen sind, einschließlich abgebrochener, fehlerhafter etc.
- `EnumWorkItemStateCategory.Restared` Alle Work-Items, die abgeschlossen sind und danach neu eingeplant wurden, egal warum.

Wenn `null` übergeben wird, werden alle Kategorien abgefragt.

Bei Abfrage der finalen Katergorien `Final` und/oder `Restarted` können mehrerer, bereits ausgeführte Worker-Instanzen mit derselben `InstanceGuid` zurückgeliefert werden. In allen anderen Fällen ist die `InstanceGuid` eindeutig.

Parameter user

Wenn dieser Parameter nicht initial ist, werden nur Work-Items, die für den angegebenen User (typischerweise der aktuelle User) sichtbar sind zugückgeliefert. Das sind neben seinen eigenen auch alle mit `IsVisibleForAllUsers`.

Mit den Standardwert `default` werden immer alle Work-Items unabhängig von ihrem User zurückgeliefert.

Parameter instanceGuids

Hiermit können mehrere `InstanceGuids` auf einmal abgefragt werden. Die anderen Filter-Parameter greifen aber zusätzlich.

Der Standardwert `null` bewirkt keine Einschränkung.

Beispiel:

```
Api.Worker.GetWorkItems(new[] { EnumWorkItemStateCategory.Waiting, EnumWorkItemStateCategory.Ready, EnumWorkItemStateCategory.Active }, UserHelper.CurrentUserGuid())
```

Information über ein einzelnes Work-Item

```
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
    }
}
```

Um eine *Massenverarbeitung von Datensätzen* zu implementieren, sollte besser die Klasse [MassWorkItem](#) verwendet werden.

Am Wahrscheinlichsten muss man neben `Run` noch die Methode `WorkItemFinished` überschreiben, siehe [Dokumentation](#).



Eine länger dauernde Implementierung von `Run` ($\geq 30s$) sollte immer abbrechbar sein. Andernfalls kommt es zu undefiniertem Verhalten beim Anwendungsneustart. Das erreicht man, indem man entweder regelmäßig auf `!IsRunning` prüft oder auf `workItemShouldFinish()` reagiert.

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 constructor</summary>
    public MassOperationRelationCleanupTask(Guid holderRecordOid, Guid? ...
```


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, MaxNumberOfRestarts

Wie oft wurde diese Instanz des Work-Items bereits gestartet und wie oft darf eine Neustart maximal erfolgen.

Diese Parameter werden verwendet, um zu entscheiden, ob ein automatischer Neustart nach einem Fehler oder einem Timeout noch angemessen ist. Standardmäßig werden maximal 3 Neustarts 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 / Zustände](#).

Für einige, häufig benötigte Zustände gibt es eigene Properties:

bool IsRunning

Der Prozess ist gerade aktiv und hat kein Signal zu Abbrechen bekommen. (`EnumWorkItemState.Running`)

bool IsError

Der Prozess ist mit einer Exception abgebrochen:

- `EnumWorkItemState.Error` Es gab eine unbehandelte Ausnahme in der Methode `Run`.
- `EnumWorkItemState.Killed` Der Prozess hat nicht auf die Anforderung zum Beenden nach **Zeitüberschreitung** (`EnumWorkItemState.CancellingBySystem`) reagiert und wurde mit einer `ThreadAbortException` abgebrochen.
- `EnumWorkItemState.Aborted` Der Prozess hat nicht auf die Anforderung zum Beenden wegen **Anwendungsneustart** (`EnumWorkItemState.ShutdownRequest`) reagiert und wurde mit einer `ThreadAbortException` abgebrochen.

bool IsDone

Der Prozess hat seine aktive Arbeit beendet und ist in seinem finalen Status. Das ist jeder Status ab der

Kategorie “final”, und es bedeutet nicht, dass die Arbeit auch erfolgreich (fehlerfrei) war.

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.

bool IsIndeterminate

Der aktuelle Fortschritt ist nicht definiert. `CurrentProgress / MaxProgress` ist keine nicht-negative Zahl.

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.

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 Run()

Hier muss die eigentliche Arbeit implementiert werden.



Die Implementierung von Workern sollte immer abbrechbar sein auch wenn `IsCancelable = false` ist, andernfalls kann es zu Problemen beim Anwendungsneustart kommen.

Man kann entweder auf Statusänderungen reagieren (`!IsRunning`) oder sich von `WorkItemShouldFinish` aktiv unterbrechen lassen.

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. Alternativ kann aber auch innerhalb der Implementierung von `Run` regelmäßig die Bedingung `!IsRunning` geprüft und ggf. die Bearbeitung abgebrochen werden.

Der Aufruf erfolgt mit einem der Status `ShutdownRequest`, `CancelledBySystem` oder `CancelledByUser`. Falls mehrere Abbruchgründe eintreten, kann die Methode sogar mehrfach aufgerufen werden. Allerdings kann ein Grund nur durch einen wichtigeren ersetzt werden. Der wichtigste ist `CancelledByUser`.



Die Implementierung dieser Methode muss thread-safe und sogar reentrant 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:

Status	Beschreibung	Standardimplementierung
Finished	Die Methode Run ist ordnungsgemäß zurückgekehrt oder das Work-Item wurde zwar abgebrochen, aber es war sowieso gerade fertig (<code>CurrentProgress == MaxProgress</code>)	keine Aktion
Removed	Das Work-Item war schon einmal eingeplant und wurde abgebrochen, bevor es starten konnte.	keine Aktion
ShutdownConfirmed	Die Anwendung wird gerade heruntergefahren und das Work-Item hat sich auf Aufforderung beendet.	Das Work-Item wird neu eingeplant (für nach dem nächsten Neustart).
Timeout	Das Work-Item hat die maximale Verarbeitungsdauer überschritten und sich daraufhin beendet.	Solange <code>MaxNumberOfRestarts</code> noch nicht erreicht ist, wird das WorkItem neu eingeplant. Die Aktion wird mit LogCriticalError protokolliert.
Killed	Das Work-Item hat die maximale Verarbeitungsdauer überschritten und sich nicht binnen der gesetzten Nachfrist beendet.	Der Vorfall wird mit LogCriticalError protokolliert.
Canceled	Das Work-Item hat auf eine Abbruchanforderung durch den Benutzer reagiert und sich selbst beendet.	keine Aktion
Aborted	Die Ausführung wurde durch einen Anwendungsneustart unerwartet unterbrochen. Dieser Aufruf erfolgt natürlich erst nach dem Neustart der Anwendung.	Das Work-Item wird neu eingeplant.
Error	Die Run-Methode ist mit einer unbehandelten Ausnahmen abgebrochen. Diese steht in <code>ExceptionDuringRun</code> .	Solange <code>MaxNumberOfRestarts</code> noch nicht erreicht ist wird das WorkItem neu eingeplant. Der Fehler wird mit LogCriticalError protokolliert.

Wenn sich das Work-Item nach der Ausführung der Methode neu eingeplant hat, wechselt der Status automatisch von `Finished` auf `Reschedule`, von `ShutdownConfirmed` auf `ShutdownRestart`, von `Aborted` auf `AbortedRestart`, von `Timeout` auf `TimeoutRetry` bzw. 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 darf die Methode nicht selbst `Save` aufrufen. Das würde die transaktionale Integrität brechen. Das Work-Item wird danach immer automatisch gespeichert.



Es dürfen keinesfalls 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. Falls längere Folgetätigkeiten bei der Beendigung eines Work-Items erforderlich sind, kann ein anderer Worker mit diesen an dieser Stelle eingeplant werden.

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();
}
```

void 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.

void LogCriticalError(Exception ex, string message = null)

Damit wird eine kritische, also nicht sinnvoll behandelbare Ausnahme protokolliert. Die Protokollierung erfolgt standardmäßig 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. Bei System-Work-Items erfolgt die Signalisierung an alle User. 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 XmlDocument.

WorkItemBase CreateSuccessor()

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 `WaitingForStart` (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` überschrieben 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 selbst fangen

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. Und zuguterletzt müssen neue, ungespeicherte Objekte verworfen werden. Das alles erledigt `UnitOfWork.UndoChanges()`. 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())
{
    // UOW aufräumen!
    UnitOfWork.UndoChanges();
    // Fehler protokollieren oder was auch immer
}
```

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 eigenen 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 `ExceptionDuringRun`.

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 aufrufen. 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.

AfterProcessingOrm

Diese Methode wird nach jedem Aufruf von `ProcessSingleOrm` aufgerufen, und zwar unabhängig davon, ob eine Exception aufgetreten ist. Über den Rückgabewert von `AfterProcessingOrm` kann man steuern, ob die Schleife über die einzelnen Orms ganz normal fortgesetzt werden soll (`Continue`), oder ob sie beendet werden soll. Beenden erfolgt über `Finish` oder `Retry`, die sich darin unterscheiden, dass im Falle einer Wiedereinplanung, um die sich die überladene Klasse selbst kümmern muss, das aktuelle ORM erneut aufruft, oder beim nächsten fortsetzt.

11.4.4. Aktionen auf alle selektieren Datensätze

Für die Abarbeitung der Datensätze aus `OrmTempSelectedRecords` und damit der Selektierten steht `TemporaryRecordMassWorkItem<T>` zur Verfügung. Diese Basisklasse bietet folgendes Verhalten:

- Gesicherte Abarbeitung der temporären Einträge aus dem [Igniter](#) ().
- Transaktion für die Abarbeitung jedes Datensatzes. Bei Ausnahmen wird der Datensatz übersprungen und `OnError` aufgerufen.
- Transparente Paketierung der Abarbeitung. Dadurch steigt die Serverlast nicht, wenn eine größere Menge an Einträgen selektiert sind. Die Standardgröße ist 200 und kann im Konstruktor `ChunkSize` angepasst werden.
- Zähler für aufgetretene Ausnahmen
- Events
 - `BeforeProcessing(bool resume)` Optional
 - `ProcessSingleOrm(T orm)`
 - `AfterProcessing(Exception exception)` Optional
 - `OnUserCancelled()` Optional
 - `BeforeSuspend()` Optional
 - `OnError(T orm, Exception exception)` Optional.

Beispiel

Einfachste Implementierung für das Setzen eines Datenfeldes.

```
public class MyMassWorker : TemporaryRecordMassWorkItem<MyDataTable>
{
    public string MyProperty { get; set; }

    protected override void ProcessSingleOrm(MyDataTable myData)
    {
        myData.TextField = MyProperty;
    }
}
```

Optionale Übergabe der Anzahl der Datensätze

Ist die Anzahl der Datensätze bekannt, können diese dem `TemporaryRecordMassWorkItem` übergeben werden. Beispielsweise kann die Anzahl aus `OperationOverSelectedRecordsIgniterBase.RecordCount` nach dem Erstellen der temporären Datensätze ausgelesen werden. In diesem Fall, muss man einen entsprechenden Konstruktor anlegen.

```
public MyMassWorker (int count) : base()
```

```
{  
    MaxProgress = count;  
}
```

Dadurch muss der Worker keine `Count()` Abfrage durchführen.

BeforeProcessing

An dieser Stelle kann man typischerweise erste Informationen in das Anwendungsprotokoll schreiben.

```
protected override void BeforeProcessing(bool resume)  
{  
    if (NumberOfStarts == 1)  
        Logger.AddInfo("Prozess Start");  
    else  
        Logger.AddInfo("Neustart des Prozesses");  
}
```

ProcessSingleOrm

Die Verarbeitung der einzelnen Datensätze findet in dieser Methode statt. Daher muss diese implementiert werden.

AfterProcessing

Wird am Ende der Verarbeitung aufgerufen.

```
protected override void AfterProcessing(Exception exception)  
{  
    Logger.AddInfo("Datensätze geändert: " + CurrentProgress);  
}
```

OnUserCancelled

Wenn der Benutzer in der UI den Prozess abbrechen können soll, muss im Konstruktor `IsCancellable = true`; gesetzt werden. Wenn dann der Benutzer dies tut, wird diese Methode aufgerufen,

```
protected override void OnUserCancelled()  
{  
    Logger.AddInfo("Abbruch durch Benutzer.", CurrentProgress);  
}
```

BeforeSuspend

Wenn der Prozess seine maximale Laufzeit überschritten hat wird er von System beendet und später neu gestartet, falls die Anzahl der maximalen Neustarts nicht überschritten ist.

```
protected override void BeforeSuspend()
{
    Logger.AddInfo("Prozesslaufzeit überschritten", CurrentProgress);
}
```

OnError

Dieses Event wird aufgerufen, wenn ein Laufzeitfehler bei der Verarbeitung auftritt. Die Standardimplementierung schreibt Protokolleinträge in das Anwendungsprotokoll, falls es vorhanden ist, und in das Server Log.

Bei einem Fehler bei der Verarbeitung, beispielsweise fehlende Bearbeitungsrechte oder der Datensatz ist nicht valide, werden die Protokolleinträge geschrieben und der Datensatz übersprungen. Möchte man stattdessen den Prozess abbrechen und später neu versuchen, muss an dieser Stelle eine Ausnahme geworfen werden.

```
protected override void OnError(MyDataTable orm, Exception exception)
{
    throw new Exception("Bei der Verarbeitung ist ein Fehler aufgetreten: " +
        exception.Message);
}
```


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

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;
```

Anschließend kann man Einträge hinzufügen. Das Protokoll wird spätestens beim Hinzufügen des ersten Eintrags erstellt. Genauerer kann man ggf. über `LoggingMode` steuern.

Soll ein bestimmtes, existierendes Anwendungsprotokoll (weiter) geschrieben werden, Setzt man dazu im Konstruktor seines Workers die OID des Logs.

```
public MyWorker() : base()
{
    // ...
    LoggingProcess = EnumLogProcessesExtension.MyLog;
    LoggerGuid = myLogGuid;
    LoggingMode = WorkerLoggingMode.Always;
    // ...
}
```

Wenn man spezielle Anforderungen bzgl. der Erstellung des Anwendungsprotokolls wie z.B. eine eigene Log-Datentabelle hat, kann man `InitLogger()` überschreiben.

```
protected override void InitLogger()
{
    // Special logger type
    OrmMySpecialLog logger = LoggerGuid == Guid.Empty
        ? new OrmMySpecialLog(Api.ORM.GetNewSession(), LoggingProcess, Name)
        : Api.ORM.GetOrm<OrmMySpecialLog>(LoggerGuid, Api.ORM.GetNewSession());
}
```

```
Logger = logger;  
// set properties  
logger.Title = "My Title";  
logger.MyProeprty = 42;  
}
```



Protokolle müssen immer eine eigene XPO-Session nutzen und keine `UnitOfWork`, damit die Protokollierung unabhängig von Fehlern im Work-Item ist.

Protokolleinträge

Das Hinzufügen von Einträgen ist thread-safe, asynchron und ohne `LockingExceptions`. Die Einträge bleiben auch dann erhalten, wenn die aktuelle `UnitOfWork` wegen eines Fehlers zurückgerollt wird. Keinsesfalls darf `Logger.Save()` während der Protokollierung verwendet werden!

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\nZeile 3");
```

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.

TranslatableException

Die Ausnahme `TranslatableException` kann genutzt werden, um Informationen für Protokolleinträge zu werfen. [Siehe](#)

Die entsprechend abgefangen werden sollte.

```
catch (TranslatableException tex)
{
    logger.AddEvent(tex);
}
```

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, dass 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

Erstellen Sie einen zweiten Worker auf Basis von `TemporaryRecordMassWorkItem`. Implementieren Sie dort die notwendige Funktionalität, verzögern Sie den Start und erlauben Sie das der Anwender den Prozess abbrechen kann.

Duplizieren Sie Ihre Aktion, dabei beachten Sie dass Sie der neuen Aktion eine eigene `Id` geben und die Namen abändern. Stellen Sie die neue Aktion auf den neuen Worker um.

Anwendungsprotokoll

Erweitern Sie den zweiten Worker um ein Anwendungsprotokoll.

Ausnahmebehandlung

Provozieren Sie eine Ausnahme in `BeforeProcessing()` 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?

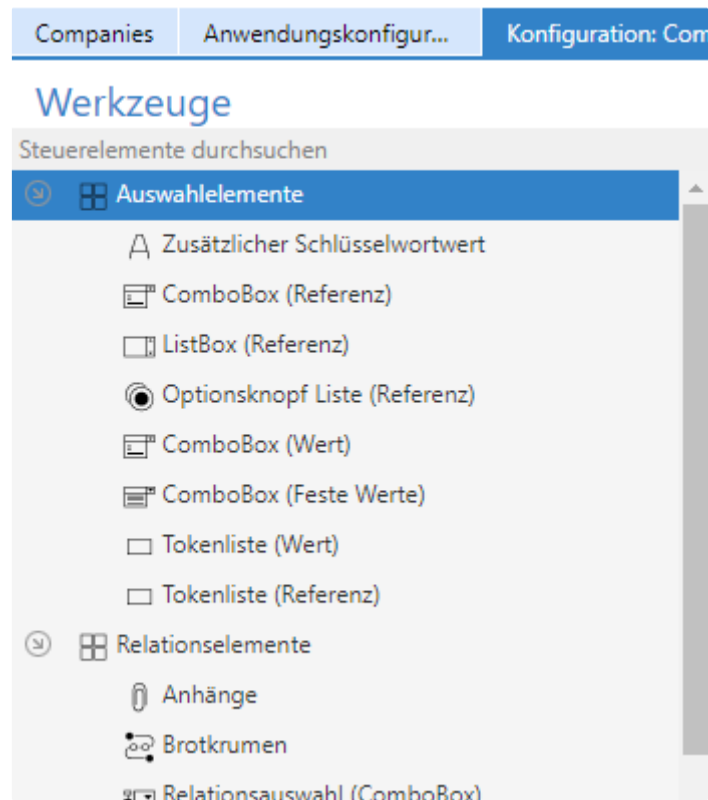
[Lösung](#)

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.

Sinnvolle Basisklassen

- `BA.Core.Configuration.FormBase.Controls.DataControlBase`: Diese Basisklasse stellt bereits eine standardisierte Möglichkeit zur Auswahl einer mit dem Steuerelement verbundenen Spalte aus der der Maske zugrundeliegenden Datentabelle bereit.
- `BA.Core.Configuration.FormBase.Controls.DataSelection`: Diese Basisklasse stellt bereits eine standardisierte Möglichkeit zur Auswahl einer mit dem Steuerelement verbundenen Relationsdefinition aus der der Maske zugrundeliegenden Datentabelle bereit.
- `BA.Core.Configuration.FormBase.Controls.PropertyControlBase`: Diese Basisklasse vereinfacht im Renderer das Binding, die beiden oberen Basisklassen erweitern diese.
- `BA.Core.Configuration.ControlBase`: Diese Basisklasse muss mindestens genutzt werden und stellt keine weiteren Funktionalitäten für Maskensteuerelemente bereit.

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
- `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.FormConfigurationGuid)]
    public class ValueAndAliasControl : DataControlBase
    {
        [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]";
            DesignerHintText = "[INSERT TRANSLATION GUID]";
        }

        protected override IReadOnlyCollection<EnumDataTypes> GetTypesOfOrmFieldName()
        {
            {
                return new[] { EnumDataTypes.String };
            }
        }
    }
}
```

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 (`[BRequired]`) 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 serialisierbar 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).
- `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 `EnumDataTypes.String` definiert man die Textfelder. In `EnumDataTypes` findet man die möglichen Datentypen.



Um ein konsistentes Verhalten aller Eingabesteuerelemente zu erhalten, sollte sichergestellt werden, dass das eigene Steuerelemente auch ein UI-Event für Feldänderungen auslöst.

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 Listefeld sieht und auch, wie die vom Browser gesendeten Daten aufbereitet werden müssen, damit sie in das der Maske zugehörige Datenmodell gespeichert werden können.

Je nach Funktionsumfang muss die Basisklasse einer der folgenden sein

- `ControlDefaultRendererBase`: Renderer OHNE Unterteilung in Read- und EditMode-Methoden und OHNE Bind-Unterstützung
- `ControlDefaultBindableRendererBase`: Renderer OHNE Unterteilung in Read- und EditMode aber MIT Bind-Unterstützung
- `ControlDefaultReadEditRendererBase`: Renderer MIT Unterteilung in Read- und EditMode-Methoden und OHNE Bind-Unterstützung
- `ControlDefaultReadEditBindableRendererBase`: Renderer MIT Unterteilung in Read- und EditMode-Methoden und MIT Bind-Unterstützung

Die Zuordnung eines Renderers zu einem bestimmten Steuerelement erfolgt über ein entsprechendes Attribut:

```
[FormControlRenderer(typeof(AttachmentControl))]
```

Sie können durch die Angabe dieses Attributs in einem Projekt-Modul auch existierende Renderer übersteuern, indem Sie einen existierenden Renderer einfach neu definieren. Eine komplette neu Implementierung beispielsweise des `TextEditControl` ist zwar möglich aber nicht zu empfehlen.

Der Renderer beinhaltet im einfachsten falls nur die Methode `Render()` alternativ die beiden Methoden `EditMode()` und `ReadMode()`. Optional ist die Implementierung von `Bind()`.

Die Render-Methoden sorgen für die korrekte Ansteuerung und Verwendung der visuellen Komponenten (bspw. DevExpress) für den entsprechenden Modus. Von den Möglichkeiten unterscheiden sich diese 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. Dies kann unter drei Bedingungen automatisch durchgeführt werden.

1. `ViewState.ViewStateData` muss unter dem Schlüssel `BindKeys.ModelProperty` den serverseitig verwendeten Feldnamen (Property-Namen auf dem `BindObject`) enthalten. Ist das Steuerlement ein `DataControlBase`, so wird dies vom Framework automatisch gemacht.
2. `ViewState.ViewStateData` muss unter dem Schlüssel `BindKeys.ClientProperty` den clientseitig verwendeten Feldnamen (inkl. `RenderParameter.NamePrefix`) enthalten.
3. Die ausgelesenen Daten, können ohne Typkonvertierung direkt in das Model geschrieben werden.

In der Bind-Methode hat man keinen Zugriff auf das Steuerelement selbst, daher müssen alle notwendigen Informationen für das Binden in `ViewState.ViewStateData` geschrieben werden. Dies können nur serialisierbare Daten sein.

Beispiel

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 : ControlDefaultReadEditBindableRender
erBase
    {
        (... Klasseninhalt ...)
    }
}
```

Die Klasse selbst erbt vom `ControlDefaultReadEditBindableRendererBase` und ist über das Attribut `FormControlRenderer` als Renderer für Steuerelemente des Typs `ValueAndAliasControl` gekennzeichnet.

Der „Klasseninhalt“ besteht folglich aus den Methoden `EditMode()`, `ReadMode()` und in unserem Beispiel auch `Bind()`.

EditMode

Die Feldnamen mit den Suffixen „_Value“ und „_Alias“ werden von der `EditMode()` Methode festgelegt und in `ViewStateData` für die `Bind()` Methode hinterlegt. 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 ausgelesen und der `CreateControl`-Methode als Vorbelegung übergeben. Nähere Beschreibungen bieten auch die Kommentarzeilen im Beispielquellcode.

```
public override void EditMode()
{
    ValueAndAliasControl valueAndAliasControl = (ValueAndAliasControl)RenderPa
rameters.Control;

    // Feldnamen definieren (Datenmodell)
    string modelFieldName = valueAndAliasControl.OrmFieldName;
    ViewState.ViewStateData.AddOrUpdate(BindKeys.ModelProperty, modelFieldNam
e);
}
```

```

// Feldnamen definieren für die Felder im Browser definieren
string uiValueFieldName = modelFieldName + "_Value";
ViewState.ViewStateData.Add("ValueField", uiValueFieldName);
string uiAliasFieldName = modelFieldName + "_Alias";
ViewState.ViewStateData.Add("AliasField", uiAliasFieldName);

// zu verwendendes Trennzeichen
char delimiter = valueAndAliasControl.Delimiter.ToCharArray()[0];
ViewState.ViewStateData.Add("Delimiter", delimiter);

// Inhalt des gespeicherten Feldes holen und für die Felder vorbereiten
string fieldContent = (string)ConvertFromModelPropertyType(GetValueFromModel());
string[] splt = fieldContent?.Split(delimiter) ?? (new string[] { "", "" });
string fieldContentPart1 = splt?[0] ?? "";
string fieldContentPart2 = splt?.Length > 1 ? splt[1] : "";

// Erstes Feld erzeugen und als Objekt hinzufügen
MVCxFormLayoutItem mvcxItem = CreateTextBoxItem(valueAndAliasControl, uiValueFieldName, RenderParameters.RenderingContext.FormModel, "Wert", fieldContentPart1);
RenderParameters.MvcxControls.Add(mvcxItem);

// Zweites Feld erzeugen und als Objekt hinzufügen
mvcxItem = CreateTextBoxItem(valueAndAliasControl, uiAliasFieldName, RenderParameters.RenderingContext.FormModel, "Alias", fieldContentPart2);
RenderParameters.MvcxControls.Add(mvcxItem);
}

```

Konvention: Abruf der Werte in den Render-Methoden

Der Zugriff auf die Werte, muss in jedem Fall über die Deskriptoren erfolgen.



Ein direktes Auslesen beispielsweise über `GetPropertyValue()` oder ähnliches ist untersagt.

Für den Feldnamen des Steuerelementes auf dem Bind-Objekt für die Basisklasse `PropertyControlBase` und damit auch für `DataSelection` und `DataControlBase` wird garantiert, dass beim Abrufen von Daten zunächst `GetValueFromModel()` und anschließend `ConvertFromModelPropertyType()` aufgerufen wird.



Renderer müssen sich an diese Konvention halten!

Sollten weitere Werte aus dem Bind-Objekt benötigt werden, so werden diese ebenfalls immer unter Verwendung von `PrepareBindProperty()` und `GetValueFromModel()` abgerufen; der Aufruf auf `ConvertFromModelPropertyType()` ist in diesem Fall optional und muss von der Methode unterstützt werden.

Hieraus folgt:

Direkte Zugriffe auf `RenderParameters.BindData` sind immer kritisch zu betrachten. Hier ist man nur richtig, wenn tatsächlich der top-level-Datensatz verwendet werden soll. Das gültige Bind-Objekt für das aktuell angesteuerte Property ist `BindObject`! Was beispielsweise auch ein Teil-Datensatz sein kann.



Weiterhin ist die Nutzung von `GetPropertyValue` und `SetPropertyValue` auf egal welchem Bind-Objekt ab sofort untersagt. Es MUSS zwingend mit den Deskriptoren gearbeitet werden.

Beispiel:

Der Feldname des Controls ist `"Addresses[1].AddressType"`, `BindData` ist `OrmCRMCompany`. Durch den Aufruf `PrepareBindProperty("Addresses[1].AddressType")` passiert folgendes:

- `RenderParameters.BindData` bleibt wie es war eine Instanz von `OrmCRMCompany`
- `BindObject` wird auf den angesteuerten Teildatensatz verschoben
- `PreparedProperty` beinhaltet nun den `PropertyDescriptor` des Feldes `"AddressType"` für den Typen des Teildatensatzes

Bind()

Die `Bind()` Methode ist dafür zuständig, dass der Wert, der vom Request des Browsers, in das Model übertragen wird.

Die Funktion der `Bind()` Methode ist relativ einfach: Das Steuerelement zeigt in der Benutzeroberfläche zwei Felder, deren Namen aus `ViewState.ViewStateData` ausgelesen werden. Diese beiden Felder müssen aus den vom Browser kommenden Daten ausgelesen, zusammengerechnet und in das Model geschrieben werden. Beim Zusammenrechnen wird das konfigurierte Trennzeichen des Steuerelementes verwendet, welches ebenfalls aus `ViewState.ViewStateData` ausgelesen wird.

```
public override void Bind()
{
    // Auslesen der Daten von dem Value
    string bindValueFieldName_Value = (string)ViewState.ViewStateData["ValueField"];
    string value = (string)ConvertToModelPropertyType(GetValueFromRequest(bindValueFieldName_Value) ?? "");
```

```

// Auslesen der Daten von dem Alias
string bindValueFieldName_Alias = (string)ViewState.ViewStateData["AliasField"];
string alias = (string)ConvertToModelPropertyType(GetValueFromRequest(bindValueFieldName_Alias) ?? "");

// Berechnung des Wertes für das Model
string delimiter = (string)ViewState.ViewStateData["Delimiter"];
string modelValue = value + delimiter + alias;

// Schreiben des Wertes in das Model
WriteValueToModel(modelValue);
}

```

Konvention: Binden von Werten

Um eigene Werte zu binden muss zunächst `GetValueFromRequest()`, dann `ConvertToModelPropertyType()` und schließlich `WriteValueToModel()` aufgerufen werden.



Es muss `GetValueFromRequest()` verwendet werden, da diese Methode Request-Validierungen vornimmt, um die Sicherheit vor feindlichen Inhalten zu gewährleisten.

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 valueAndAliasControl,
    string fieldName, DevExpressFormModel formModel, string label, string presentValue)
{
    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 Feldname, der im
    // Webbrowser verwendet werden soll, die Beschriftung vor dem Feld, der Hilfetext, etc
}

```

```

    mvcxItem.Name = fieldName;
    mvcxItem.FieldName = fieldName;
    //mvcxItem.HelpText = RenderingUtils.GetHelpText(Api.Text.Format(valueAndAliasControl.HelpText));
    mvcxItem.ClientVisible = valueAndAliasControl.Visible;
    mvcxItem.VerticalAlign = FormLayoutVerticalAlign.Top;
    mvcxItem.Caption = label;
    mvcxItem.CaptionSettings.Location = Api.Enum.EnumValueConverter.GetLabelPosition(valueAndAliasControl.LabelPosition);
    mvcxItem.CaptionSettings.VerticalAlign = FormLayoutVerticalAlign.Top;

    // Weitere bestimmte Einstellungen sind im sog. "Settings"-Unterobjekt verfügbar.
    // Das betrifft zB die Editierbarkeit des Feldes, bestimmte Javascript-Events im Browser,
    // die ausgeführt werden sollen, Abmessungen des Feldes, usw.
    TextBoxSettings settings = (TextBoxSettings)mvcxItem.NestedExtensionSettings;

    // Horizontale Ausrichtung des Controls
    settings.ControlStyle.HorizontalAlign = HorizontalAlign.Left;

    // Ist das Feld eventuell für Bearbeitung gesperrt?
    if (!IsControlEditable())
    {
        settings.ClientEnabled = false;
        ViewState.ShouldBind = false;
    }

    // Breite des Feldes
    settings.Width = new Unit(valueAndAliasControl.Width, Api.Enum.EnumValueConverter.GetValueWidthType(valueAndAliasControl.WidthType));

    // Feld kann als fehlerhaft markiert werden, falls die Validierung fehlschlägt
    if (settings.Properties is EditProperties)
    {
        EditProperties props = (EditProperties)settings.Properties;
        ApplicationSubConfigurationForm appFormSettings = Api.Config.ApplicationConfiguration().GetSubConfiguration<ApplicationSubConfigurationForm>();

        props.ValidationSettings.Display = Api.Enum.EnumValueConverter.GetErrorFrameDisplayMode(appFormSettings.ErrorFrameDisplayMode);
        props.ValidationSettings.ErrorTextPosition = Api.Enum.EnumValueConverter.GetErrorTextPosition(appFormSettings.ErrorTextPosition);
        props.ValidationSettings.ErrorDisplayMode = Api.Enum.EnumValueConverter.GetErrorDisplayMode(appFormSettings.ErrorDisplayMode);
    }

```



```

        settings.ShowModelErrors = true;
    }

    // Der "placeholder", also der Text, der in einem leeren Feld angezeigt werden soll
    settings.Properties.NullText = Api.Text.Format(valueAndAliasControl.NullText);

    // vorbelegen des übergebenen im Datenmodell schon verfügbaren Wertes
    if (!string.IsNullOrEmpty(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 kopiert werden
    // können, die keine Eingabe erlauben
    RenderingUtils.FixItemForFirefox(mvcxItem, settings, valueAndAliasControl.CssClass);
    return mvcxItem;
}

```

Read

In der Read-Methode hat man prinzipiell die identischen Möglichkeiten wie in der Edit-Methode. An dieser Stelle sollte man nur Elemente verwenden, die das Ändern der Daten nicht ermöglichen. In diesem Fall wird ein "Label" zur Anzeige verwendet.

```

public override void ReadMode()
{
    string namePrefix = RenderingUtils.GetUIReadyContextPrefix(RenderParameters.RenderingContext);
    ValueAndAliasControl valueAndAliasControl = (ValueAndAliasControl)RenderParameters.Control;

    // Feldnamen setzen
    string modelFieldName = valueAndAliasControl.OrmFieldName;

    // Alternativ kann ein MVCxFormLayoutItem auf diese Weise erstellt werden.
    RenderParameters.MvcxControls.Add(item =>
    {
        // Typ des Items definieren und Settings casten
        item.NestedExtensionType = FormLayoutNestedExtensionItemType.Label;
    });
}

```

```
LabelSettings labelSettings = (LabelSettings)item.NestedExtensionSettings;

// Wert aus dem Datensatz auslesen und in als Inhalt festlegen
char delimiter = valueAndAliasControl.Delimiter.ToCharArray()[0];
string fieldContent = (string)ConvertFromModelPropertyType(GetValueFromModel());
fieldContent = fieldContent?.Split(delimiter)?[0] ?? "";
labelSettings.Text = fieldContent;

// Weitere Einstellungen festlegen
labelSettings.Name = namePrefix + valueAndAliasControl.ControlInternalName + RenderParameters.MvcxControls.Count;
labelSettings.Width = new Unit(valueAndAliasControl.Width, Api.Enum.EnumValueConverter.GetValueWidthType(valueAndAliasControl.WidthType));
labelSettings.ControlStyle.CssClass = valueAndAliasControl.CssClass;
item.HorizontalAlign = FormLayoutHorizontalAlign.Left;
item.Caption = valueAndAliasControl.Caption?.Translate() ?? string.Empty;

item.CaptionSettings.Location = Api.Enum.EnumValueConverter.GetLabelPosition(valueAndAliasControl.LabelPosition);
RenderingUtils.SetHelpText(valueAndAliasControl, item);
item.Visible = valueAndAliasControl.Visible;

// Label setzen, falls konfiguriert
if (string.IsNullOrEmpty(item.Caption))
    item.ShowCaption = DevExpress.Utils.DefaultBoolean.False;
});
}
```

12.1.3. Vorgehen

In diesem Abschnitt werden verschiedene Vorgehen beschrieben, um Standardanforderungen an Maskenelementen abzubilden. Diese Vorgehen dienen als Beispiele. Eine Voraussetzung für die Implementierung sind sie in der Regel nicht. In diesen Beispielen werden DevExpress MVC Steuerelemente gerendert. Prinzipiell können in den Rendern beliebige und auch ganz eigene Elemente erzeugt werden.

12.1.3.1. 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]";
        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 des 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> list)
    {
        IEnumerable<EnumMYEnum> returnList = base.AddWhere(list).Cast<EnumMYEnum>();
        if (Properties is CDPMYEnumValuesProperties prop)
            returnList = returnList.Where(ff => ff.OwnProperty == prop.OwnPropertyEqual);
        return returnList;
    }
}
```

Bei der Zuweisung muss nun der eigene Datenprovider angegeben werden.

```
DataProviderProperties = new CDPMYEnumValuesProperties
{
    OwnPropertyEqual = "Wert",
};
```

12.1.3.2. Name

Als Name bzw. HTML Id wird wenn möglich der Feldname aus der Datentabelle verwendet. Dies ist nicht in allen Fällen möglich, beispielsweise bei berechneten Feldern oder bei Relationsauswahlen.

DevExpress erstellt für die einzelnen Maskenelemente JavaScript Objekte, die unter `window` zur Verfügung stehen. Beispiel:

```
window.Name.GetValue()
```

Daher muss darauf geachtet werden, dass nicht die identischen Namen zweimal innerhalb eines Tabs verwendet werden. Dazu zählen auch alle Dialoge, die innerhalb eines Tabs geöffnet werden. Daher erhalten die Elementnamen neben dem Feldnamen, noch einen weiteren Präfix.

```
TextEditControl control = (TextEditControl)parametersUi.Control;
string namePrefix = RenderingUtils.GetUIReadyContextPrefix(renderingContext);
MVCxFormLayoutItem mvcxItem = new MVCxFormLayoutItem
{
    NestedExtensionType = FormLayoutNestedExtensionItemType.TextBox,
    Name = namePrefix + control.OrmFieldName,
    FieldName = control.OrmFieldName,
};
```

Werden nach dem hier dargestellten Schema verfahren werden die Daten auch automatisch gebunden.

Werden Elemente generiert, die nicht konkreten Feldern aus der Datentabelle zuzuordnen sind, werden die IDs der Steuerelemente aus der Maskenkonfiguration genutzt, um eindeutige Namen zu generieren. Vereinfacht kann man einfach `ControlInternalName` dafür verwenden.

```
string namePrefix = RenderingUtils.GetUIReadyContextPrefix(renderingContext);
MVCxFormLayoutItem mvcxItem = new MVCxFormLayoutItem
{
    Name = namePrefix + parametersUi.Control.ControlInternalName,
    FieldName = parametersUi.Control.ControlInternalName,
};
```

12.1.3.3. Sichtbarkeit

Werden Steuerelemente aufgrund von Sichtbarkeitsregeln versteckt, werden für diese die Renderer erst gar nicht aufgerufen. Ausnahme davon ist das Flag `Visible` von `ViewableControlBase`. In diesem Fall soll das Steuerelement zwar gerendert werden aber in der UI nicht sichtbar sein. Dies ist dann wichtig, wenn man in der UI ein Steuerlement benötigt, um von diesem über JavaScript Werte auszulesen oder zu setzen.

Dieses Flag kann direkt auf dem DevExpress Item gesetzt werden.

```
mvcxItem.ClientVisible = control.Visible;
```

12.1.3.4. Label

Abhängig von den Eigenschaften `Caption` und `LabelPosition` wird das Label in dem Element dargestellt. Wurde kein Label konfiguriert wird keines dargestellt und auch das Sternchen zur Darstellung des Pflichtfeldes wird ausgeblendet. Wird ein Label dargestellt, sollte auch die Positionierung korrekt gesetzt werden.

```
if (string.IsNullOrEmpty(control.Caption))
{
    mvcxItem.ShowCaption = DevExpress.Utils.DefaultBoolean.False;
    mvcxItem.RequiredMarkDisplayMode = FieldRequiredMarkMode.Hidden;
}
else
{
    mvcxItem.Caption = control.Caption?.Translate();
    mvcxItem.CaptionSettings.Location = Api.Enum.EnumValueConverter.GetLabelPosition(control.LabelPosition);
}
```


12.1.3.5. Hilfetext

In `ViewableControlBase` ist die Eigenschaft `HelpText` für den Hilfetext definiert. Für die Anzeige des Hilfetextes gibt es eine globale und benutzerabhängige Konfiguration. Damit dies vollständig berücksichtigt wird, gibt es eine entsprechende Hilfsmethode in `RenderingUtils`.

```
RenderingUtils.SetHelpText(control, mvcxItem);
```

12.1.3.6. Adaptive und Responsive Layout

Werden DevExpress MVC Controls für die Steuerelemente verwendet, müssen die entsprechenden Eigenschaften `RowSpan`, `ColumnSpan` und `SpanRules` des [Adaptiven Layouts](#) entsprechend gesetzt werden. Insbesondere bei dem Einsatz des Steuerlementes in Dialogen ist das korrekte Setzen wichtig.

Dabei werden die Eigenschaften aus `ControlBase` entsprechend an das MVC Element übertragen.

```
mvcxItem.ColumnSpan = control.ColSpan;
mvcxItem.RowSpan = control.RowSpan;
foreach (Models.SpanRule spanrule in control.SpanRules) {
    mvcxItem.SpanRules.Add(
        new DevExpress.Web.SpanRule() {
            BreakpointName = spanrule.BreakpointName,
            ColumnSpan = spanrule.ColumnSpan,
            RowSpan = spanrule.RowSpan
        });
}
```

12.1.3.7. Datensatz als geändert anzeigen

Der Tab, in dem die Maske dargestellt wird, erhält ein * falls der Datensatz sich geändert hat. Damit dies auch bei eigenen Elementen funktioniert, muss nach der Änderung eines Feldwertes der Tab auf "Dirty" gesetzt werden.

```
TextBoxSettings settings = (TextBoxSettings)mvcxItem.NestedExtensionSettings;  
settings.Properties.ClientSideEvents.TextChanged = "BA.Ui.Controls.ControlEvents.SetTabAsDirty";
```

12.1.3.8. Umgang mit Feldwerten

Ist es notwendig auf die Felder des Datensatzes zuzugreifen, so befindet sich dieser in `parameters.BindData`. Für generische Steuerelemente, die sowohl in freien Dialogen als auch in Datensatz-Masken eingesetzt werden können, sollte man den Zugriff möglichst generisch gestalten.

Auslesen eines Feldwertes

```
TypeCacheModel recordType = Api.Types.GetByType(parameters.BindData.GetType());  
PropertyCacheModel propertyType = recordType.Properties[control.OrmFieldName];  
string stringValue = propertyType.Property.GetValue(parameters.BindData)?.ToString();
```

Setzen eines Feldwertes

```
TypeCacheModel recordType = Api.Types.GetByType(parameters.BindData.GetType());  
PropertyCacheModel propertyType = recordType.Properties[control.OrmFieldName];  
propertyType.Property.SetValue(parameters.BindData, stringValue);
```



Für den Umgang mit den Werten in der `Bind` Methode siehe die Beschreibung im Kapitel [Renderer](#)

12.1.3.9. Auomatisches setzen vom Fokus

Wenn in ein Steuerelement implementiert wird, welches den Default-Fokus erhalten soll, muss das Interface `IDefaultFocus` implementiert werden. In diesem Fall muss im Renderer sichergestellt werden, dass das Flag `DefaultFocus` ausgewertet wird und ggf. das Steuerelement die entsprechende CSS-Klasse erhält, um korrekt fokussiert zu werden.

Steuerelement

`IDefaultFocus` definiert, dass dieses Steuerelement ein Property namens `DefaultFocus` haben muss:

```
[DisplayName("D2507FDE-58B5-4FA5-80D4-CF8CB23968F3")]
[PropertiesGroup("CC746B4E-7655-43C4-B57A-37C77FBAD021", 200)]
[HelpText("56C897A2-53B7-4EE5-8F57-6C32F12ACF71")]
public bool DefaultFocus { get; set; }
```

Renderer

Im Renderer ist folgender Code erforderlich (hier am Beispiel einer Combobox):

```
if (!readOnly && control.DefaultFocus && control.Visible)
{
    comboModel.Settings.ControlStyle.CssClass += RenderingUtils.GetDefaultFocusStyles(renderingContext);
    SubRecordsControl subRecordsControl = RenderingUtils.FindSubRecordsControl(renderingContext);
    if (subRecordsControl != null)
        comboModel.Settings.ControlStyle.CssClass += " " + Constants.DefaultFocus.SubFocus;
    if (subRecordsControl == null || (subRecordsControl != null && subRecordsControl.DefaultFocus))
        comboModel.Settings.ControlStyle.CssClass += " " + Constants.DefaultFocus.Focus;
}
```

Erklärung: Wenn das Flag `DefaultFocus` gesetzt ist erweitern wir die Styles dieses Controls um eine oder zwei CSS-Klasse ergänzt – je nachdem ob dieses Element in Teildaten verwendet wird oder direkt in einer Maske steht. Falls es in Teildaten verwendet wird, und das Teildatensatzelement selbst auch das `DefaultFocus` Flag gesetzt hat bekommt dieses Element beide CSS-Klassen.

Zur Vereinfachung gibt es eine Helper-Methode

```
if (!readOnly && control.DefaultFocus && control.Visible)
    comboModel.Settings.ControlStyle.CssClass += RenderingUtils.GetDefaultFocusStyles(renderingContext);
```

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 parameters) where T : ActionBase
```

Mit der Klasse `RibbonActionsModificationModel` kann man nun die `DynamicClientVisibility` beeinflussen.

```
public override void OnOpening(FormEventsParameters parameters)
{
    base.OnOpening(parameters);
    IEnumerable<ClientActionCreatePDFAndEmail> actions = GetActionsOfType<ClientActionCreatePDFAndEmail>(parameters);
    foreach (ClientActionCreatePDFAndEmail action in actions)
    {
        RibbonActionsModificationModel modifier = new RibbonActionsModificationModel();
        modifier.DynamicClientVisibility.Add(EnumActionVisibility.IfValueIsTrue);
        modifier.AdditionalClientData.Add("TestValueIsTrue", ((OrmMyDataTable)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 Gruppename):

```
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.FormConfiguration.FlattenControls().OfType<SingleDetailGridViewControl>();
foreach (SingleDetailGridViewControl detailView in detailViews)
    list.Add(detailView.GetRelationDefinition().RelationTypeGuid.ToString(), null, 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 des 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. Seiten und Widgets

Eine Seite stellt den Inhalt eines Tabs dar und besteht aus Widgets. Ein Beispiel wäre die Startseite, wo man unterschiedliche Widgets mit generellen Informationen anzeigen kann; oder eine Bereichsseite mit Informationen bspw. zu Adressen mit einem "Zuletzt verwendet" Widget, welches nur Firmen und Kontakte anzeigt.

Mit Widgets wird ermöglicht beliebige Inhalte in Seiten anzuzeigen. Jedes Widget braucht einen Controller und eine View. Zusätzlich wird ein View-Model benutzt, um beliebige Daten dem View mitzugeben. Um ein Widget in einer Seite konfigurieren zu können, braucht man ein Steuerelement für das Widget, wo man verschiedene Einstellungen konfigurieren kann.

13.1. Klassen

WidgetBase

Interne Basis von das Widget System. Stellt basis, überschreibbare Funktionalität für die Entwicklung von Widgets. Überschreibbare Methoden für Widget Entwicklern:

- `IsValid` für die Validierung des Widgets in der Konfiguration,
- `SkipRender` für den Fall, dass unter Umstände das Widget nicht gerendert sein soll,
- `GetStyleString` für die Inline Style eines Widgets,
- `GetCssClass` um eine Css Klasse auf höchstes Niveau in der Widgets HTML zu setzen,
- `GetTSClassName` um eine angepasste .ts Klasse für das Widget zu setzen,
 - `GetControlName` um den im Designer angezeigten Namen des Widgets zu setzen,
 - `GetWidgetTitle` um den in der Seite angezeigten Titel des Widgets zu setzen,
 - `GetWidgetTitleOrFallback` ruft `GetWidgetTitle` aber liefert ein Fallback Wert, wenn `GetWidgetTitle` leer oder null zurück liefert,
- `GetUISetting` um Benutzerspezifische UI Widget Einstellungen zu lesen,
- `SetUISetting` um Benutzerspezifische UI Widget Einstellungen zu speichern,
- `GetUISettingValueForUserOrConfig` um Benutzerspezifische UI Widget Einstellungen zu lesen oder aus der Konfiguration, wenn es keine Benutzerspezifisches Wert gibt,
- `GetControllerParameters` um spezifischen Parameters für den Controller zu setzen,

Widget Ausrichtungen

`WidgetBase` beinhaltet auch die Definitionen zur vertikalen und horizontalen Ausrichtung. Werden die beiden Feldern konfiguriert, erhält das Widget entsprechende CSS-Klassen. In der Regel sind diese aber nicht ausreichend. Die Ausrichtung des eigenen Inhaltes muss man entsprechend den Einstellungen mithilfe eigener CSS-Klassen selbst sicherstellen.

LayoutWidget

`LayoutWidget` erbt von `WidgetBase` und wird für Widgets, die als Container anderen Widgets gedacht sind und eine Struktur in der Seite definieren. Zum Beispiel die Zeile und Spalten Widgets.

`LayoutWidget` bereitet die Funktionalität für das `SkipRender` von Widgets, indem alle die unterliegenden Widgets überprüft werden.

Widget

`Widget` erbt von `WidgetBase` und wird die Basis für alle die Widgets, die für die Anzeige gedacht sind. Zum Beispiel ein `NavigationMenuWidget` oder ein `GoogleMapsWidget`.

DataWidget

Widgets können Daten von der Seite oder anderen Widgets konsumieren. Dafür soll das Widget von der Klasse `DataWidget` erben und mittels `WidgetParameterDefinition` definieren, welche Parameter Typen das Widget lesen kann. Mehr dazu im Kapitel [Parameter](#) und im Kapitel [Parameter konsumieren](#).

OrmDataWidget

Erbt von `DataWidget`. Widgets, die `OrmWidgetParameter` konsumieren, erben von dieser Klasse.

13.2. Interfaces

IProvideWidgetParameter

Widgets können Daten für andere Widgets bereitstellen. Dafür muss das Widget das Interface `IProvideWidgetParameter` implementieren und mittels `WidgetParameterDefinition` definieren, welche Parameter das Widget bereitstellen kann. Mehr dazu im Kapitel [Parameter](#) und im Kapitel [Parameter konsumieren](#).

IPreviewableWidget

Widgets können eine alternative, kleinere Darstellung haben, die als Übersicht benutzt wird. Dafür muss das Widget das Interface `IPreviewableWidget` implementieren und mittels der entsprechenden Felder und Methoden definieren, wie diese Darstellung gerendert wird.

Widgets, die diese Interface implementieren, werden als Übersicht gerendert, wenn die in einem `PreviewContainerWidget` konfiguriert sind.

13.3. Controllers

WidgetControllerBase

Basis Implementierung eines Widget-Controllers. Dieser hat die `RequestedPage`, `RequestedWidget` und `TabGuid`.

DataWidgetControllerBase

Erbt von `WidgetControllerBase` und hat zusätzlich `ParameterJsonString` (Parameter für das Widget im JSON Format) und den `ParameterType`, dass das Widget konsumiert.

13.4. Parameter

Parameter ermöglichen die Interaktion zwischen Seite und Widget sowie Widget und Widget. Ein Parameter braucht eine .cs Klasse und eine entsprechende .ts Klasse.

Widgets definieren durch `WidgetParameterDefinition` welche Parameter sie bereitstellen bzw. konsumieren können.

WidgetParameterDefinition

Definiert `DisplayName`, `ParameterName` und `ParameterType` eines Widget-Parameters von `IProvideWidgetParameter` Widgets bereitgestellt.

OrmsWidgetParameterDefinition

Erbt von `WidgetParameterDefinition` und definiert zusätzlich `OrmDataSource`.

WidgetParameterBase

Basis für alle die Parameter Typen.

OrmsWidgetParameter

Beinhaltet eine Liste von `OrmData` Objekten. Ein `OrmData` Objekt beinhaltet die benötigte Info, um einen Datensatz aus der DB oder Cache zu lesen.

GridSelectionWidgetParameter

Erbt von `OrmsWidgetParameter` und wird für die Selektion in einer Ansicht genutzt.

GridViewConfigWidgetParameter

Beinhaltet die benötigte Info, um eine `GridViewConfiguration` zu laden.

13.5. Typescript

Widget.ts

Ist die Basis für die Widget Funktionalitäten im Client und bietet die Möglichkeit an, manche davon anzupassen.

SetWidgetState

Widgets haben einen Status, der „NotLoaded“, „UnLoaded“, „Loading“, „Loaded“, „Blocked“, „Error“, „ContentBlocked“ und „ContentError“ sein kann. Was für einen Status ein Widget hat, hängt von dem Laden des Widgets ab und wird automatisch gesetzt. Die Funktion `SetWidgetState` kann aufgerufen werden, um unter Umständen den Status vom Widget zu ändern. Zum Beispiel bei Fehlern.

TriggerParameterChangedEvent

Wenn das Widget einen bestimmten Parameter bereitstellt, triggert diese Funktion das UI Event `WidgetParameterUpdated`.

OnParameterExternallyChanged

Wenn ein Widget den konsumierten Parameter ändert, muss diese Methode aus dem bereitstellenden Widget aufgerufen werden. Beispiel: Maske ändert den Datensatz und ruft diese Funktion auf. Um einem Listen Widget zu informieren. Das Listen Widget reagiert darauf und triggert das UI Event, damit andere konsumierende Widgets auch auf die Änderung reagieren können.

ReadSourceParameterJsonString

Wenn das Widget einen Parameter braucht, liefert diese Funktion den Parameter in einer JSON formatierte String aus der Quelle zurück.

GetProvidedParameterJsonString

Wenn das Widget einen Parameter mit Namen bereitstellt, wird diese Funktion den Parameter in einer JSON formatierte String aus der Quelle zurückliefern.

GetProvidedParameter

Wenn das Widget einen Parameter von Typ `T` bereitstellt, wird diese Funktion das entsprechende Objekt zurückliefern.

SetProvidedParameter

Wenn das Widget einen Parameter von Typ `T` bereitstellt, wird diese Funktion das als Funktionsparameter gegebene Objekt als Parameter vom Widget setzen und ein entsprechende UI Event triggern, damit

andere Widgets sich aktualisieren können, um die neuen Daten zu nutzen.

OnParameterChanged

Funktion die aufgerufen wird, wenn das Parameter, das das Widget nutzt, aktualisiert worden ist.

AddWidgetAction

Fügt eine Aktion in der Widget UI hinzu. Die Aktionen werden oben rechts angezeigt, wenn ein Benutzer den Maus über das Widget zieht. Basis Aktionen und überschreibbare Funktionalität sind:

Refresh

Läd das Widget neu.

Collapse

Verstecken oder anzeigen des Widgets. Bei einem Widget mit verstecktem Inhalt, wird nur der Titel und die Aktionen angezeigt. Für diese Aktion sind folgende Funktionen überschreibbar: `BeforeCollapsing` / `BeforeExpanding`, um Code direkt vor der Collapse/Expand Aktion auszuführen, beziehungsweise `false` zurück zu liefern, um die Aktion abubrechen; und `OnCollapsed` / `OnExpanded`, um Code direkt nach der Collapse/Expand Aktion auszuführen.

Maximize

Zeigt das Widget vergrößert (volle Seite) an, oder in der originalen Größe. Für diese Aktion sind folgende Funktionen überschreibbar: `BeforeMaximizing` / `BeforeSizeRestoring`, um Code direkt vor der Maximize/Restore Aktion auszuführen, beziehungsweise `false` zurückliefern, um die Aktion abubrechen; und `OnMaximized` / `OnSizeRestored`, um Code direkt nach der Maximize/Restore Aktion auszuführen.

Remove

Schliesst das Widget in der Seite und verschiebt es in den Katalog. Diese Aktion wird nur angezeigt, wenn das Widget entfernt werden kann. Für diese Aktion sind folgende Funktionen überschreibbar: `BeforeRemoving`, um Code direkt vor dem Entfernen auszuführen, beziehungsweise `false` zurückliefern, um die Aktion abubrechen; und `OnRemoved`, um Code direkt nach der Entfernen Aktion auszuführen.

WidgetSettingsDialog

Öffnet einen Dialog in welchem der Benutzer für sich individuelle Einstellungen zur Anzeige des Widgets festlegen kann.

- `OnWidgetSettingsChanged`, um Code nach dem Verändern von `WidgetSettings` auszuführen
- `OnWidgetSettingsDialogClosed`, um Code nach dem Schließen des Dialogs auszuführen

AttachDragAndDropEvents

Werden Widgets per Drag and Drop bewegt, wird diese Event ausgeführt.

Unload

Wenn ein Widget entfernt wird, wird es auch entladen. Diese Methode erlaubt Widgets zu definieren, was beim Entladen passiert. Hier gehören zum Beispiel Status Variablen zurückzusetzen oder sich von UI Events zu lösen.

DefaultFocus

Handelt das Setzen des Focus im Widget, wenn das Widget den Vorgabe-Focus in der Seite erhält.

GetUISetting

Liest Benutzerspezifische UI Widget Einstellungen.

SetUISetting

Speichert Benutzerspezifische UI Widget Einstellungen.

Init

Bietet die Möglichkeit an, Code auszuführen als das Widget im Browser initialisiert wird. In diesem Zeitpunkt könnte der Inhalt des Widgets noch nicht gerendert sein.

OnContentLoaded

Bietet die Möglichkeit an, Code auszuführen als der Inhalt des Widgets im Browser geladen ist.

Parameters

Die Parameter Klassen in Typescript entsprechen die [.cs Klassen](#).

13.6. UI Events

WidgetDragStart

Wird gefeuert, wenn das Draggen in einer Seite startet.

- Scope: ID der Seite.
- Data: { draggedWidget, sourceWidget }

WidgetDragged

Wird mit WidgetDragStart gefeuert, aber für ein spezifisches Widget.

- Scope: ID der Seite + „.“ + ID des Widgets.
- Data: { sourceWidget }

WidgetHoverIn

Wird gefeuert, wenn während des Draggens das Widget über einen Container bewegt wird.

- Scope: ID der Seite + „.“ + ID des ContainerWidgets.
- Data: { draggedWidget, sourceWidget }

WidgetHoverOut

Wird gefeuert, wenn während des Draggens das Widget nicht mehr über einen Container ist.

- Scope: ID der Seite + „.“ + ID des ContainerWidgets.
- Data: { draggedWidget, sourceWidget }

WidgetDroppedOut

Wird gefeuert, wenn ein Widget aus einem Container gedroppt wird.

- Scope: ID der Seite + „.“ + ID des ContainerWidgets.
- Data: { droppedWidget, targetWidget, sibling }

WidgetDropped

Wird gefeuert, wenn ein Widget gedroppt wird.

- Scope: ID der Seite + „.“ + ID des Widgets.
- Data: { targetWidget, sourceWidget, sibling }

WidgetDroppedIn

Wird gefeuert, wenn ein Widget in einen Container gedroppt wird.

- Scope: ID der Seite + „.“ + ID des ContainerWidgets.
- Data: { `droppedWidget`, `sourceWidget`, `sibling` }

WidgetDragEnd

Wird gefeuert, wenn das Draggen in einer Seite endet.

- Scope: ID der Seite.
- Data: { `draggedWidget` }

WidgetDragCancelled

Wird gefeuert, wenn das Draggen in einer Seite abgebrochen wird.

- Scope: ID der Seite.
- Data: { `draggedWidget`, `sourceWidget` }

WidgetRemoved

Wird gefeuert, wenn ein Widget aus einer Seite entfernt wird.

- Scope: ID der Seite + „.“ + ID des Widgets.
- Data: { }

WidgetActionClicked

Wird gefeuert, wenn eine Aktion von einem Widget geklickt wird.

- Scope: ID der Seite + „.“ + ID des Widgets.
- Data: { `actionName` }

13.7. Eigene Widgets

13.7.1. Einfaches Widget

Ein Widget ist ein Steuerelement, das von `Widget` erbt. Ein einfaches Widget ist hier ein Widget, dass selbständig funktioniert und keine Information aus der Seite oder aus einem anderen Widgets benötigt.

Ein Widget benötigt:

- Das Steuerelement.
- Einen Controller.
- Falls der View Daten vom Controller benötigt, wird empfohlen, ein View-Model zu erstellen.
- Ein View. Der Name der View sollte wie die Aktion heißen und in einem Ordner mit gleichem Namen des Controllers liegen.
- Falls die View auch Custom Scripts im Browser brauchen, ein `.ts` Datei. Zum Beispiel `NavigationMenuWidget.ts`.

In diesem Beispiel wird ein einfaches Widget `HelloWorldWidget` erstellt.

Steuerelement

Das Widget muss von `Widget` erben. `Widget` erbt von `WidgetBase` und damit hat das Widget die Basis Felder und Funktionen.

In diesem Beispiel wird eine feste CSS-Klasse im Feld `CssClass` gesetzt und eine zweite CSS Klasse, die abhängig von einer konfigurierbaren Option `Bold` ist, wird durch die Überschreibung der Methode `GetCssClass` gesetzt.

Die Methode `GetControllerParameters` wird benutzt, um den Parameter `textLanguage` an den Controller zu übermitteln. Der Controller könnte aber auch direkt auf die Seite und das Widget zugreifen.

Zudem muss das Widget ein `ControllerName` definieren.

Widgets können in `ActionName` die Aktion im Controller definieren. Wird dies nicht gesetzt, nutzt das Widget die Aktion „Index“.

Man kann auch Standardwerte in anderen Feldern setzen, wie hier die Höhe und Ausrichtungen. Oder wie bei den anderen Steuerelementen, so können auch hier Eigenschaften von Basisklassen [modifiziert](#) werden.

Das `HelloWorldWidget` braucht zusätzliche Scripts in der UI. Daher ist die Methode `GetTSClassName` überschrieben, um die entsprechende `.ts` Klasse für das Widget zu definieren.

```
[Serializable]
public class HelloWorldWidget : Widget
{
```

```

        [DisplayName("Fett")]
        public bool Bold { get; set; }

        [DisplayName("Sprache")]
        public string Language { get; set; }

        public HelloWorldWidget() : base()
        {
            Id = "16B8262B-0461-4871-A07A-66F74EF21559".ToGuid();
            ToolboxName = "HelloWorld";
            Icon = "earth2";
            ControllerName = nameof(HelloWorldWidget);
            PixelHeight = 300;
            HeightType = EnumWidgetHeightType.Pixel;
            HorizontalAlign = EnumHorizontalAlign.Center;
            VerticalAlign = EnumVerticalAlign.Middle;
            CssClass = "hello-world";
        }

        public override string GetTSClassName()
        {
            return "BA.Ui.Widgets.HelloWorldWidget";
        }

        public override string GetCssClass()
        {
            return base.GetCssClass() + " " + (Bold ? "bold-text" : "regular-text");
        }

        public override object GetControllerParameters()
        {
            return new { textLanguage = Language };
        }
    }

```

View Model

Das View Model beinhaltet die Informationen, die die View benötigt.

```

public class HelloWorldWidgetViewModel
{
    public string SalutationText { get; set; }

    public string ToSalutate { get; set; }
}

```

```
        public string ErrorText { get; set; }  
    }
```

Controller

Der Controller hat eine Aktion `Index`, die das View Model für das Widget vorbereitet und dann die entsprechende PartialView zurückliefert. In dem Fall wird kontrolliert, ob eine Sprache konfiguriert wurde. Falls nicht, wird das Widget einen Fehlertext beinhalten. Wenn „DE“ als Sprache konfiguriert ist, wird der Text auf Deutsch vom Widget ausgegeben. Ansonsten auf Englisch.

Die Aktion soll ein Partial View zurückliefern.

```
public class HelloWorldWidgetController : WidgetControllerBase  
{  
    public ActionResult Index(string textLanguage)  
    {  
        HelloWorldWidgetViewModel viewModel = new HelloWorldWidgetView  
Model();  
  
        if (string.IsNullOrEmpty(textLanguage))  
            viewModel.ErrorText = "No language configured";  
        else  
            switch (textLanguage.ToUpper())  
            {  
                case "DE":  
                    viewModel.SalutationText = "Hallo";  
                    viewModel.ToSalutate = "Welt";  
                    break;  
  
                default:  
                    viewModel.SalutationText = "Hello";  
                    viewModel.ToSalutate = "world";  
                    break;  
            }  
  
        return PartialView(viewModel);  
    }  
}
```

View

Die View (`/Views/HelloWorldWidget/Index.cshtml`) für das `HelloWorldWidget` zeigt den Inhalt nur wenn kein Fehler auftritt. Der Fehlertext befindet sich in dem `data-errortext` Attribut.

```
@model BA.Core.Models.HelloWorldWidgetViewModel
<div class="hello-world-content" data-errortext="@Model.ErrorText">
    @if (string.IsNullOrEmpty(Model.ErrorText))
    {
        <span>@Model.SalutationText</span> <span>@Model.ToSalutate</span>
    }
</div>
```

Um die Ausrichtungen optisch ansprechend umzusetzen, sollte man entsprechende eigenen CSS-Styles anlegen. In diesem Beispiel werden die Klassen benutzt, um die gewünschten Ausrichtungen zu erzeugen. Jeder Fall wird aber durch die Anforderungen erheblich unterschiedlich sein. Für dieses Beispiel wird folgendes CSS benutzt:

```
.hello-world .widget-content { position: relative; }
.hello-world .hello-world-content { position: absolute; width: fit-content; }
.hello-world.horizontalAlignRight .hello-world-content { right: 0; }
.hello-world.horizontalAlignCenter .hello-world-content { right: 0; left: 0; margin: auto; }
.hello-world.verticalAlignMiddle .hello-world-content { top: 41%; }
.hello-world.verticalAlignBottom .hello-world-content { bottom: 0 }
.hello-world.bold-text .hello-world-content { font-weight: bolder; }
.hello-world.regular-text .hello-world-content { font-weight: normal; }
```

UI Script

Die TypeScript Klasse wird bei der Erstellung des Widgets im Browser benutzt. Das `HelloWorldWidget` überschreibt die `OnContentLoaded` Funktion, um einen möglichen Fehler zu behandeln.

```
module BA.Ui.Widgets {
    "use strict";
    export class HelloWorldWidget extends Widget {
        public override OnContentLoaded(): void {
            let errorText: string = $(this.GetWidgetJQIdSelector() + " .hello-world-content").data("errortext");

            if (errorText && errorText !== "")
                this.SetWidgetState(WidgetState.ContentError,
                    true, errorText, false);
        }
    }
}
```

Ergebniss

Jetzt kann das Widget in einer Seite konfiguriert werden. Als Beispiel werden zwei konfiguriert: Eins mit fehlender Sprache und eins auf Deutsch

The screenshot shows the BA CRM configuration interface. The top navigation bar includes 'Start', 'Firmen', 'Active Live AG', 'Anwendungskonfigurator', 'Hello World', 'Adressen', and 'Hello World'. The main area is divided into three panels: 'Werkzeuge' (Tools), 'Design', and 'Eigenschaften' (Properties).

Werkzeuge: Contains 'Steuerelemente durchsuchen' and a list of widgets including 'Struktur', 'Drag & Drop-Bereich', 'Spalte', 'Vorschau-Container', 'Widgetset', 'Zeile', 'Ansichtenwidget', 'Dashboard', 'Firmenstruktur', 'Google Maps', 'HelloWorld', 'Kampagnencockpit', 'Kampagneninfo', 'Kampagnenmonitor', 'Logo-Widget', and 'Maskenwidget'.

Design: Shows a tree structure with 'Hello World' as the root, containing 'Inhalt' and 'Widgetkatalog'. Under 'Inhalt', there are two items: 'HelloWorld_Korrekt' (selected) and 'HelloWorld_Fehlerhaftes'.


Eigenschaften: Displays the configuration for the selected 'HelloWorld_Korrekt' widget. The 'ALLGEMEINE EINSTELLUNGEN' section includes fields for 'Name' (Korrekt), 'Titel' (empty), 'Fett' (checked), 'Sprache' (DE), 'Laden verzögern' (unchecked), 'Aktualisieren erlauben' (unchecked), 'Minimieren erlauben' (unchecked), 'Maximieren erlauben' (unchecked), and 'Benutzerspezifische ...' (unchecked). The 'ANZEIGE' section includes 'Höhenangabe*' (Pixel), 'Höhe (Pixel)' (300), 'Horizontale Ausrichtung' (Zentriert), and 'Vertikale Ausrichtung' (Zentriert).


The screenshot shows the BA CRM configuration interface. The top navigation bar includes 'Start', 'Firmen', 'Active Live AG', 'Anwendungskonfigurator', 'Hello World', 'Adressen', and 'Hello World'. The main area is divided into three panels: 'Werkzeuge' (Tools), 'Design', and 'Eigenschaften' (Properties).


Werkzeuge: Contains 'Steuerelemente durchsuchen' and a list of widgets including 'Struktur', 'Drag & Drop-Bereich', 'Spalte', 'Vorschau-Container', 'Widgetset', 'Zeile', 'Ansichtenwidget', 'Dashboard', 'Firmenstruktur', 'Google Maps', 'HelloWorld', 'Kampagnencockpit', 'Kampagneninfo', 'Kampagnenmonitor', 'Logo-Widget', and 'Maskenwidget'.


Design: Shows a tree structure with 'Hello World' as the root, containing 'Inhalt' and 'Widgetkatalog'. Under 'Inhalt', there are two items: 'HelloWorld_Korrekt' and 'HelloWorld_Fehlerhaftes' (selected).


Eigenschaften: Displays the configuration for the selected 'HelloWorld_Fehlerhaftes' widget. The 'ALLGEMEINE EINSTELLUNGEN' section includes fields for 'Name' (Fehlerhaftes), 'Titel' (empty), 'Fett' (unchecked), 'Sprache' (empty), 'Laden verzögern' (unchecked), 'Aktualisieren erlauben' (unchecked), 'Minimieren erlauben' (unchecked), 'Maximieren erlauben' (unchecked), and 'Benutzerspezifische ...' (unchecked). The 'ANZEIGE' section includes 'Höhenangabe*' (Pixel), 'Höhe (Pixel)' (300), 'Horizontale Ausrichtung' (Zentriert), and 'Vertikale Ausrichtung' (Zentriert).


 Start



 * HelloWorldSeite

 HelloWorld





Hallo Welt

No language configured

13.7.2. Parameter konsumieren

In diesem Beispiel wird das `HelloWorldWidget` aus dem vorherigen Kapitel geändert, um ein Widget zu werden welches einen aktuellen Datensatz nutzt, um seinen Inhalt aufzubauen.

Steuerelement

Um Parameter nutzen zu können muss das Widget von `DataWidget` erben. In diesem Fall wird das Widget von `OrmDataWidget` erben, da das Widget Infos bezüglich Datensätze nutzt.

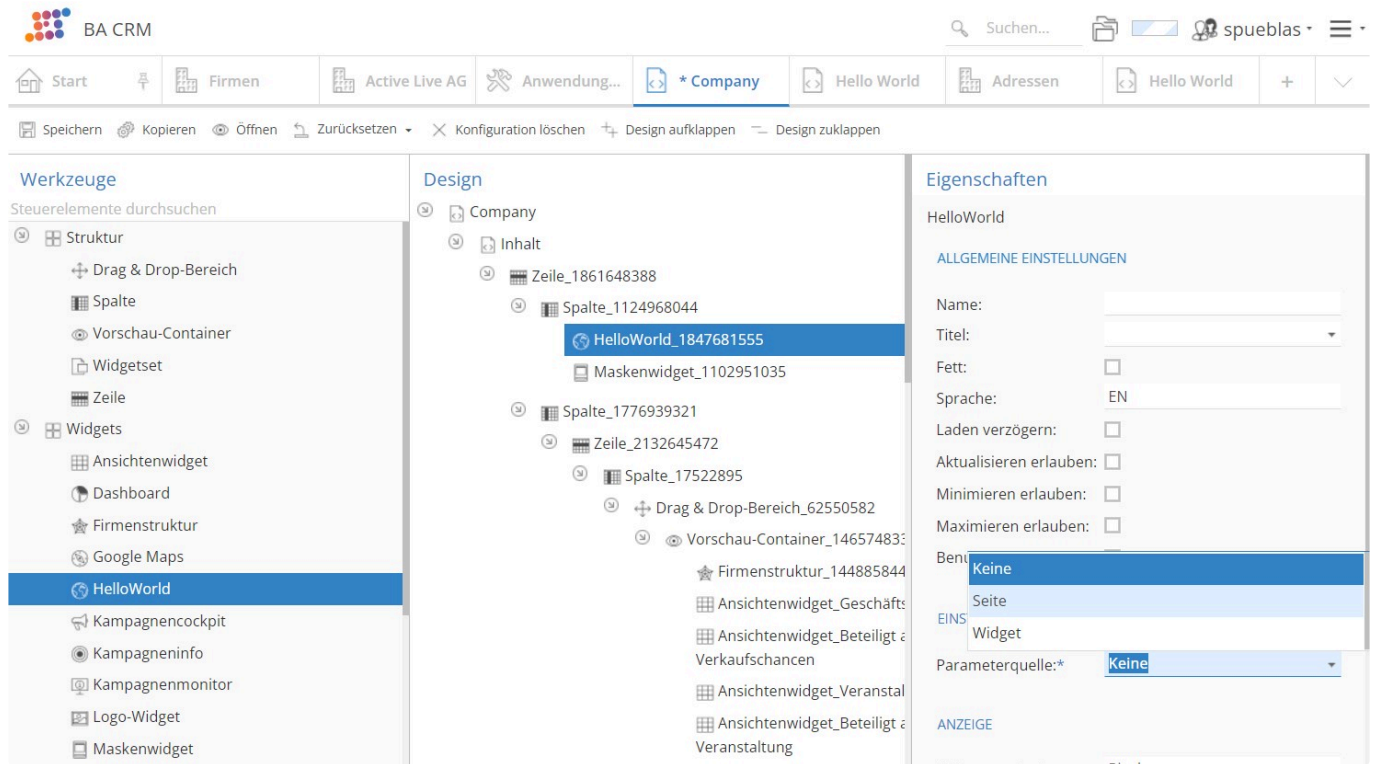
```
public class HelloWorldWidget : OrmDataWidget
```

`DataWidgets` müssen die Funktion `GetConsumedParameterDefinitions` implementieren, welche eine Liste von Kompatibel `WidgetParameterDefinition` zurückliefert.

```
public override IReadOnlyList<WidgetParameterDefinition> GetConsumedParameterDefinitions()
{
    return new List<WidgetParameterDefinition> { { new OrmsWidgetParameterDefinition() } };
}
```

Das `HelloWorldWidget` kann jetzt mit einer Parameterquelle konfiguriert werden:

- Keine: Wenn das Widget keine braucht,
- Seite: Wenn die Seite einen kompatiblen Parametertyp bereitstellt. Seiten können entweder `GridViewConfigWidgetParameter` oder `OrmsWidgetParameter` bereitstellen,
- Widget: Wenn die Quelle ein anderes in der Seite konfiguriertes Widgets sein soll. In dem Fall wird ein zusätzliche Auswahl angezeigt, in der man ein kompatibles Widget selektieren kann.



Controller

Der `HelloWorldWidgetController` **erweitert** nun `DataWidgetControllerBase`. Damit hat der Controller den Parameter im JSON Format in der Eigenschaft `ParameterJsonString` und kann ihn de-serialisieren. Wenn der Controller ein Parameter erhält, wird das `HelloWorldWidget` nun einen anderen Text anzeigen.

```
public class HelloWorldWidgetController : DataWidgetControllerBase
{
    public ActionResult Index(string textLanguage)
    {
        HelloWorldWidgetViewModel viewModel = new HelloWorldWidgetView
Model ();

        if (string.IsNullOrEmpty(textLanguage))
            viewModel.ErrorText = "No language configured";
        else
            switch (textLanguage.ToUpper())
            {
                case "DE":
                    viewModel.SalutationText = "Hallo";
                    viewModel.ToSalutate = "Welt";
                    break;

                default:
                    viewModel.SalutationText = "Hello";
                    viewModel.ToSalutate = "world";
            }
    }
}
```

```

        break;
    }

    string jsonParameter = ParameterJsonString;

    if (!string.IsNullOrEmpty(jsonParameter))
    {
        OrmsWidgetParameter parameter = JsonConvert.DeserializeObject<OrmsWidgetParameter>(jsonParameter);

        if (parameter != null && parameter.Orms != null && parameter.Orms.Any())
        {
            OrmData ormData = parameter.Orms.First();
            OrmBABase orm = API.Api.ORM.GetOrm(ormData.OrmTypeGuid, ormData.OrmGuid);

            viewModel.ToSalutate = orm.EntityTitle;
        }
    }

    return PartialView(viewModel);
}
}

```

BA CRM

Suchen... spueblas

Start Firmen Active Live ... Anwendun... Company * Hello World Adressen Hello World

Hello Active Live AG

ÜBERSICHT

KARTE

Danemark
Hamburg
Berlin
Amsterdam
Niederlande
Belgien
Luxemburg
Deutschland
Tschechien
Prag
Google
Kurzbeispiele
Kartendaten
Nutzungsbedingungen
Fahrzeit: 04:28h, Strecke: 415km

DASHBOARD

MEINE TERMINE

MEINE WIEDERVORLAGEN

Kunde
Active Live AG
[Brandstücken 27, 22549 Hamburg](#)

Name: Active Live AG

Geschäftsbeziehungen
Beteiligt an Verkaufschancen
Veranstaltungen
Beteiligt an Veranstaltung
Kampagnen

Anmerkungen

Wenn das DataWidget den OrmTyp wissen soll

OrmDataWidget können den OrmTyp mittels die Methode GetOrmTypeGuid lesen:

```
Guid? OrmType = GetOrmTypeGuid();
```

Wenn das Widget nicht ohne Parameter gerendert sein kann

Dann muss das Widget die folgende Funktion so überschreiben:

```
public override Guid[] WidgetParameterSourceFilter(string propertyName)
{
    List<Guid> ret = new List<Guid>();

    if (CanUseWidgetParameterFromPage)
        ret.Add(EnumWidgetParameterSource.Page);

    ret.Add(EnumWidgetParameterSource.OtherWidget);

    return ret.ToArray();
}
```

13.7.3. Einen Parameter bereitstellen

Ein Widget kann ein Parameter im Browser bereitstellen. Damit andere Widgets es als Parameterquelle auswählen können, soll es das korrekte Interface implementieren. Das wäre das Interface `IProvideWidgetParameter`.

In diesem Beispiel wird das `TopUserProfilesWidget` erstellt, welches die ersten 3 Benutzer Profile anzeigt und den geklickten Eintrag als Parameter bereitstellt.

TopUserProfilesWidget.cs

`TopUserProfilesWidget` implementiert das Interface `IProvideWidgetParameter`. Das Widget hat deshalb die Methode `GetProvidedParameterDefinitions`. Die Methode wird eine `OrmsWidgetParameterDefinition` für den Parametertyp `OrmsWidgetParameter`. Damit werden alle `DataWidgets` `TopUserProfilesWidgets` als Parameterquelle selektieren können, die diesen Parametertyp benötigen.

Das Objekt `WidgetParameterDefinition` bekommt als Parameter des Constructors einen Anzeigetext des Parameters (kann eine Übersetzungs-Guid sein), einen technischen Namen und den Parameter Typ. Der Technische Name ist das, was `DataWidgets` für die Kommunikation nutzen werden. Das Objekt `OrmsWidgetParameterDefinition` bekommt zusätzlich die entsprechende Guid für die Datentabelle. In diesem Fall für Benutzerprofile.

```
[Serializable]
public class TopUserProfilesWidget : Widget, IProvideWidgetParameter
{
    public TopUserProfilesWidget() : base()
    {
        Id = "669B0B80-6CFC-4388-AA09-FBAC5BC4D213".ToGuid();
        ToolboxName = "TopUserProfiles";
        Icon = "bank_building";
        ControllerName = nameof(TopUserProfilesWidget);
        PixelHeight = 300;
        HeightType = EnumWidgetHeightType.Pixel;
        CssClass = "top-user-profiles";
    }

    public override string GetTSCClassName()
    {
        return "BA.Ui.Widgets.TopUserProfilesWidget";
    }

    public IReadOnlyList<WidgetParameterDefinition> GetProvidedParameterDefinitions()
    {
        return new List<WidgetParameterDefinition>()
```

```

        {
            { new OrmsWidgetParameterDefinition("42F7A053-91FC-44F3-9A6B-EA6E6B3B176D", "clickedUserProfile", EnumDataSource.UserProfileGuid, typeof(OrmsWidgetParameter)) }
        };
    }
}

```

TopUserProfilesWidgetViewModel.cs

Das ViewModel beinhaltet in dem Fall eine Liste von `Tuple<string, OrmData>`, mit den EntityTitles und die OrmData für den Parameter im Browser.

```

public class TopUserProfilesWidgetViewModel
{
    public List<(string EntityTitle, OrmData ormData)> TopUserProfiles;

    public TopUserProfilesWidgetViewModel()
    {
        TopUserProfiles = new List<(string EntityTitle, OrmData ormData)>();
    }
}

```

TopUserProfilesWidgetController.cs

Der Controller wird die erste 3 BenutzerProfile aus der DB lesen und die das ViewModel packen.

```

public class TopUserProfilesWidgetController : WidgetControllerBase
{
    public ActionResult Index()
    {
        TopUserProfilesWidgetViewModel viewModel = new TopUserProfilesWidgetViewModel();
        IQueryable<OrmUserProfile> query = API.Api.ORM.GetQuery<OrmUserProfile>();
        foreach (OrmUserProfile uProfile in query.OrderBy(ff => ff.LastName).Take(3))
        {
            OrmData uProfileData = new OrmData()
            {
                OrmTypeGuid = uProfile.OrmType,
                OrmGuid = uProfile.Oid,
                CacheId = uProfile.RecordCacheId,
                TemporaryKey = uProfile.TemporaryKey
            }
        }
    }
}

```

```

        };

        viewModel.TopUserProfiles.Add((uProfile.EntityTitle, uProfileData));
    }

    return PartialView(viewModel);
}
}

```

View: /Views/TopUserProfilesWidget/Index.cshtml

```

@model BA.Core.Models.TopUserProfilesWidgetViewModel

<div>
    <ul>
        @foreach ((string entityTitle, BA.Core.Configuration.Pages.Parameters.OrmData ormData) in Model.TopUserProfiles)
        {
            string jsonOrmData = Newtonsoft.Json.JsonConvert.SerializeObject(ormData);
            <li data-ormdata="@jsonOrmData">@entityTitle</li>
        }
    </ul>
</div>

```

BA.Ui.Widgets.TopUserProfilesWidget.ts

Die TypeScript Klasse überschreibt die `OnContentLoaded` Funktion, um die Click Events auf die Einträge anzuhängen. Wenn ein Eintrag geklickt wird, wird den Parameter vom `TopUserProfilesWidget` aktualisiert und alle die Widgets, die den lesen, werden darauf reagieren.

```

module BA.Ui.Widgets {
    "use strict";
    export class TopUserProfilesWidget extends Widget {
        public override OnContentLoaded(): void {
            let othis: TopUserProfilesWidget = this;
            $(this.GetWidgetJQIdSelector() + " ul li").click(function () {
                let parameter: BA.Ui.Widgets.Parameters.OrmsWidgetParameter =
                    new BA.Ui.Widgets.Parameters.OrmsWidgetParameter();
                parameter.Orms.push($(this).data("ormdata"));
                othis.SetProvidedParameter<BA.Ui.Widgets.Parameters.OrmsWidgetParameter>("clickedUserProfile", parameter);
            });
        }
    }
}

```

```
}  
}  
}
```

Eine Alternative ist das Objekt `OrmData` nicht als JSON im Browser zu rendern, sondern die einzelne Infos (Oid, OrmTyp, Cacheld und TemporaryKey). In dem Fall kann man damit eine Instanz von `OrmData` in `.ts` erstellen.

Die Funktion `SetProvidedParameter` bekommt den Name des Parameters in der `WidgetParameterDefinition` als String und den neuen Parameter. In diesem Fall „clickedUserProfile“.

Ergebnisse

Nun kann das Widget in einer Seite konfiguriert werden. Als Beispiel wird es mit einem `HelloWorldWidget` konfiguriert, das von `TopUserProfilesWidget` den Parameter liest.

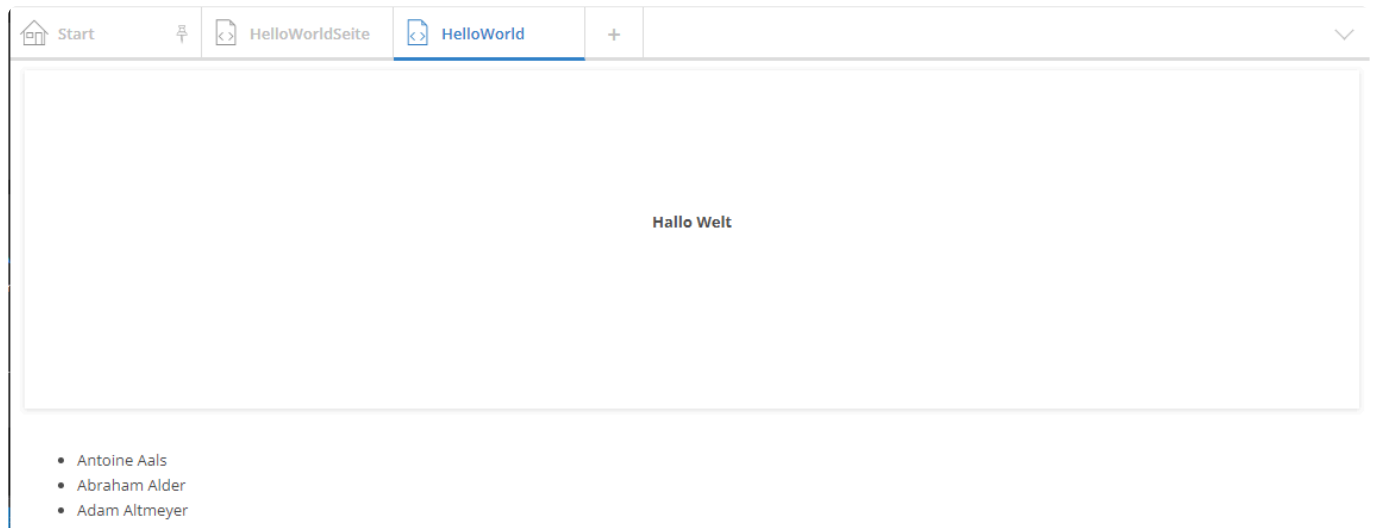
The screenshot shows the BA CRM configuration interface. The top navigation bar includes 'Start', 'Firmen', 'Active Live AG', 'Anwendung...', '* Company', '* Hello World', 'Adressen', and 'Hello World'. Below the navigation bar is a toolbar with icons for 'Speichern', 'Kopieren', 'Öffnen', 'Zurücksetzen', 'Konfiguration löschen', 'Design aufklappen', and 'Design zuklappen'. The main area is divided into three panels: 'Werkzeuge' (Tools), 'Design', and 'Einstellungen' (Settings).

Werkzeuge: A list of widgets to search for, including 'Ansichtenwidget', 'Dashboard', 'Firmenstruktur', 'Google Maps', 'HelloWorld', 'Kampagnencockpit', 'Kampagneninfo', 'Kampagnenmonitor', 'Logo-Widget', 'Maskenwidget', 'Navigationsmenü', 'Suche', 'Tag auf einen Blick', 'TopUserProfiles' (highlighted), 'Website', 'Wiedervorlagen', and 'Zuletzt verwendet'.

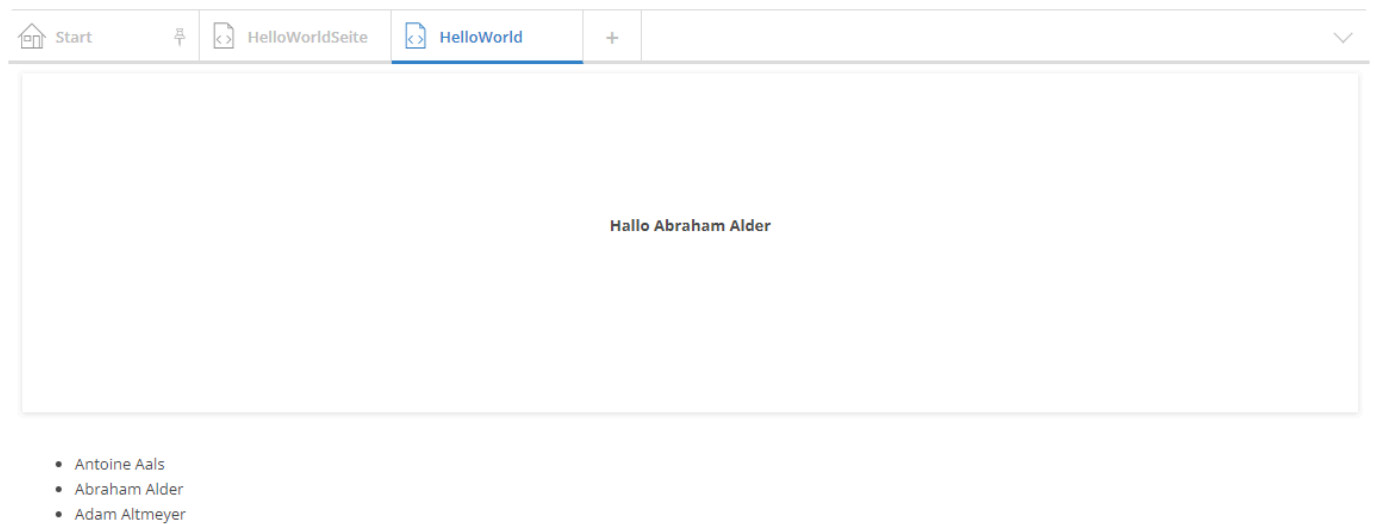
Design: A tree view showing the widget hierarchy. The 'Hello World' widget is selected, and its 'Inhalt' (Content) is expanded, showing 'HelloWorld_581724801' and 'TopUserProfiles_344948467'.

Einstellungen: Configuration options for the selected widget. The 'Parameterquelle' (Parameter source) is set to 'Widget', the 'Widget' is 'TopUserProfiles_344948467', and the 'Parameter' is 'Angeklicktes Benutzerprofil'. Other settings include 'Fett' (Bold), 'Sprache' (Language), 'Laden verzögern' (Load delay), 'Aktualisieren erlauben' (Allow update), 'Minimieren erlauben' (Allow minimize), 'Maximieren erlauben' (Allow maximize), 'Benutzerspezifische ...' (User-specific ...), 'Höhenangabe' (Height specification), 'Höhe (Pixel)' (Height (Pixel)), 'Horizontale Ausrichtung' (Horizontal alignment), and 'Vertikale Ausrichtung' (Vertical alignment).

Beim Laden kann das `HelloWorldWidget` keinen Parameter lesen, wenn nichts angeklickt war.



Auf „Abraham Alder“ Klicken:



13.7.4. Mehrere Parameter bereitstellen

Ein Widget kann auch mehreren Parametern im Browser bereitstellen. Dafür muss das Widget in der Methode `GetProvidedParameterDefinitions` mehrere `WidgetParameterDefinition` Objekte zurück liefern und im Browser mehrmals die Methode `SetProvidedParameter` aufrufen.

So kann ein Widget z.B. mit einem Parameter Benutzerprofile und mit einem weiteren Parameter Ordner bereitstellen. Das Widget könnte die Ordner eines Benutzers anzeigen und wählen lassen, damit ein anderes Widget den Inhalt der Ordner anzeigt. In diesem Beispiel wird das Widget die ersten 3 Ordner auflisten. In dem Fall werden DataWidgets in der Lage sein zu entscheiden, welchen Parameter sie nutzen.

TopUserProfilesWidget.cs

Das Widget liefert in diesem Fall zwei `WidgetParameterDefinition` zurück:

```
[Serializable]
public class TopUserProfilesWidget : Widget, IProvideWidgetParameter
{
    public TopUserProfilesWidget() : base()
    {
        Id = "669B0B80-6CFC-4388-AA09-FBAC5BC4D213".ToGuid();
        ToolboxName = "TopUserProfiles";
        Icon = "bank_building";
        ControllerName = nameof(TopUserProfilesWidget);
        PixelHeight = 300;
        HeightType = EnumWidgetHeightType.Pixel;
        CssClass = "top-user-profiles";
    }

    public override string GetTSCClassName()
    {
        return "BA.Ui.Widgets.TopUserProfilesWidget";
    }

    public IReadOnlyList<WidgetParameterDefinition> GetProvidedParameterDe
finitions()
    {
        return new List<WidgetParameterDefinition>()
        {
            { new OrmsWidgetParameterDefinition("42F7A053-91FC-44F
3-9A6B-EA6E6B3B176D", "clickedUserProfile", EnumDataSource.UserProfileGuid, ty
peof(OrmsWidgetParameter)) },
            { new OrmsWidgetParameterDefinition("26210EC1-320A-42C
A-9CD5-7D1210365735", "clickedFolder", EnumDataSource.RecordCollectionGuid, ty
```

```

peof(OrmsWidgetParameter)) }

        };

    }

}

```

TopUserProfilesWidgetViewModel.cs

Das ViewModel beinhaltet in dem Fall zwei Listen von `Tuple<string, OrmData>`, mit den `EntityTitles` und die `OrmData` für beide Parametern im Browser.

```

public class TopUserProfilesWidgetViewModel
{
    public List<(string EntityTitle, OrmData ormData)> TopUserProfiles;
    public List<(string EntityTitle, OrmData ormData)> TopFolders;

    public TopUserProfilesWidgetViewModel()
    {
        TopUserProfiles = new List<(string EntityTitle, OrmData ormData)>();
        TopFolders = new List<(string EntityTitle, OrmData ormData)>();
    }
}

```

TopUserProfilesWidgetController.cs

Der Controller liest die ersten 3 BenutzerProfile und die ersten 3 Ordner aus der DB und setzt diese in das ViewModel.

```

public class TopUserProfilesWidgetController : WidgetControllerBase
{
    public ActionResult Index()
    {
        TopUserProfilesWidgetViewModel viewModel = new TopUserProfilesWidgetViewModel();
        IQueryable<OrmUserProfile> query = API.Api.ORM.GetQuery<OrmUserProfile>();
        IQueryable<OrmRecordCollection> queryF = API.Api.ORM.GetQuery<OrmRecordCollection>();

        foreach (OrmUserProfile uProfile in query.OrderBy(ff => ff.LastName).Take(3))
        {

```

```

        OrmData uProfileData = new OrmData()
        {
            OrmTypeGuid = uProfile.OrmType,
            OrmGuid = uProfile.Oid,
            CacheId = uProfile.RecordCacheId,
            TemporaryKey = uProfile.TemporaryKey
        };

        viewModel.TopUserProfiles.Add((uProfile.EntityTitle, u
ProfileData));
    }

    foreach (OrmRecordCollection folder in queryF.OrderBy(ff => f
f.Name).Take(3))
    {
        OrmData folderData = new OrmData()
        {
            OrmTypeGuid = folder.OrmType,
            OrmGuid = folder.Oid,
            CacheId = folder.RecordCacheId,
            TemporaryKey = folder.TemporaryKey
        };

        viewModel.TopFolders.Add((folder.Name, folderData));
    }

    return PartialView(viewModel);
}
}

```

View: /Views/TopUserProfilesWidget/Index.cshtml

```

@model BA.Core.Models.TopUserProfilesWidgetViewModel

<div id="userProfiles">
    <ul>
        @foreach ((string entityTitle, BA.Core.Configuration.Pages.Par
ameters.OrmData ormData) in Model.TopUserProfiles)
        {
            string jsonOrmData = Newtonsoft.Json.JsonConvert.Seria
lizeObject(ormData);
            <li data-ormdata="@jsonOrmData">@entityTitle</li>
        }
    </ul>
</div>

```

```

<div id="folders">
    <ul>
        @foreach ((string entityTitle, BA.Core.Configuration.Pages.Parameters.OrmData ormData) in Model.TopFolders)
        {
            string jsonOrmData = Newtonsoft.Json.JsonConvert.SerializeObject(ormData);
            <li data-ormdata="@jsonOrmData">@entityTitle</li>
        }
    </ul>
</div>

```

BA.Ui.Widgets.TopUserProfilesWidget.ts

Die TypeScript Klasse überschreibt die `OnContentLoaded` Funktion, um die Click Events auf die Einträge anzuhängen. Wenn ein Eintrag aus den Benutzerprofilen angeklickt wird, wird der Parameter vom `TopUserProfilesWidget` für die Benutzerprofile aktualisiert und nur die Widgets, die Benutzerprofile als Kontext selektiert haben, werden darauf reagieren. Genauso wird es passieren, wenn ein Eintrag aus Ordnern geklickt wird.

```

module BA.Ui.Widgets {
    "use strict";
    export class TopUserProfilesWidget extends Widget {
        public override OnContentLoaded(): void {
            let othis: TopUserProfilesWidget = this;

            $(this.GetWidgetJQIdSelector() + " #userProfiles ul li").click(function () {
                let parameter: BA.Ui.Widgets.Parameters.OrmsWidgetParameter = new BA.Ui.Widgets.Parameters.OrmsWidgetParameter();
                parameter.Orms.push($(this).data("ormdata"));
                othis.SetProvidedParameter<BA.Ui.Widgets.Parameters.OrmsWidgetParameter>("clickedUserProfile", parameter);
            });

            $(this.GetWidgetJQIdSelector() + " #folders ul li").click(function () {
                let parameter: BA.Ui.Widgets.Parameters.OrmsWidgetParameter = new BA.Ui.Widgets.Parameters.OrmsWidgetParameter();
                parameter.Orms.push($(this).data("ormdata"));
                othis.SetProvidedParameter<BA.Ui.Widgets.Parameters.OrmsWidgetParameter>("clickedFolder", parameter);
            });
        }
    }
}

```

Ergebnisse

DataWidgets können jetzt Auswählen, welchen Parameter sie nutzen. Als Beispiel wird die Seite zwei `HelloWorldWidget` beinhalten, jeweils nutzt ein Parameter vom `TopUserProfilesWidget`. Also, ein `HelloWorldWidget` wird auf Benutzerprofile reagieren und das Andere auf Ordner.

BA CRM

Start

Firmen

Active Live AG

Anwendungsko...

* Hello World

Adressen

Hello World

+

✓

Speichern

Kopieren

Öffnen

Zurücksetzen

Konfiguration löschen

Design aufklappen

Design zuklappen

Werkzeuge

Steuerelemente durchsuchen

Widgets

Ansichtenwidget

Dashboard

Firmenstruktur

Google Maps

HelloWorld

Kampagnencockpit

Kampagneninfo

Kampagnenmonitor

Logo-Widget

Maskenwidget

Navigationsmenü

Suche

Tag auf einen Blick

TopUserProfiles

Website

Wiedervorlagen

Design

Hello World

Inhalt

Zeile_282553324

Spalte_1508282723

HelloWorld_Hallo Benutzerprofil

Spalte_2037663652

HelloWorld_Hallo Ordner

TopUserProfiles_39019584

Widgetkatalog

Eigenschaften

HelloWorld

ALLGEMEINE EINSTELLUNGEN

Name:

Hallo Benutzerprofil

Titel:

Hallo Benutzerprofil

Fett:

☒

Sprache:

DE

Laden verzögern:

☐

Aktualisieren erlauben:

☐

Minimieren erlauben:

☐

Maximieren erlauben:

☐

Benutzerspezifische ...

☐

EINSTELLUNGEN STEUERELEMENT

Parameterquelle:*

Widget

Widget:*

TopUserProfiles_39019584

Parameter:*

Angeklicktes Benutzerprofil

BA CRM

Start

Firmen

Active Live AG

Anwendungsko...

* Hello World

Adressen

Hello World

+

✓

Speichern

Kopieren

Öffnen

Zurücksetzen

Konfiguration löschen

Design aufklappen

Design zuklappen

Werkzeuge

Steuerelemente durchsuchen

Widgets

Ansichtenwidget

Dashboard

Firmenstruktur

Google Maps

HelloWorld

Kampagnencockpit

Kampagneninfo

Kampagnenmonitor

Logo-Widget

Maskenwidget

Navigationsmenü

Suche

Tag auf einen Blick

TopUserProfiles

Website

Wiedervorlagen

Design

Hello World

Inhalt

Zeile_282553324

Spalte_1508282723

HelloWorld_Hallo Benutzerprofil

Spalte_2037663652

HelloWorld_Hallo Ordner

TopUserProfiles_39019584

Widgetkatalog

Eigenschaften

HelloWorld

ALLGEMEINE EINSTELLUNGEN

Name:

Hallo Ordner

Titel:

Hallo Ordner

Fett:

☒

Sprache:

DE

Laden verzögern:

☐

Aktualisieren erlauben:

☐

Minimieren erlauben:

☐

Maximieren erlauben:

☐

Benutzerspezifische ...

☐

EINSTELLUNGEN STEUERELEMENT

Parameterquelle:*

Widget




Widget:*

TopUserProfiles_39019584

Parameter:*

Angeklickter Ordner

Beim Laden können die HelloWorldWidgets keinen Parameter lesen, weil nichts geklickt war:

 HelloWorld	 Anwendungskon...	 HelloW
--	--	--

HALLO BENUTZERPROFIL

Hallo Welt




HALLO ORDNER

Hallo Welt

- Antoine Aals
- Abraham Alder
- Adam Altmeyer

- Ordner 1
- Ordner 2
- Ordner 3

Auf „Abraham Alder“ Klicken. Dann aktualisiert sich nur das linke HelloWorldWidget:

 HelloWorld	 Anwendungskon...	 HelloW
--	--	--

HALLO BENUTZERPROFIL

Hallo Abraham Alder




HALLO ORDNER

Hallo Welt

- Antoine Aals
- Abraham Alder
- Adam Altmeyer

- Ordner 1
- Ordner 2
- Ordner 3

Auf „Ordner 3“ Klicken. Dann aktualisiert sich nur das rechte HelloWorldWidget:

 HelloWorld	 Anwendungskon...	 HelloW
---	---	---

HALLO BENUTZERPROFIL

Hallo Abraham Alder

HALLO ORDNER

Hallo Ordner 3

- Antoine Aals
- Abraham Alder
- Adam Altmeyer

- Ordner 1
- Ordner 2
- Ordner 3



Für dieses Beispiel ist das HelloWorldWidget angepasst um den Name der Ordner anzuzeigen, da OrmRecordCollection kein EntityTitle hat.

13.7.5. Einstellung durch den Anwender

In der Widget Klasse

Widget Settings ermöglichen das ändern von Einstellungen durch den Anwender.

WidgetSettingsAttribute

Beim `WidgetSettingsAttribute` handelt es sich um ein Attribut mit welchem Eigenschaften innerhalb eines Widgets als Widget-Setting gekennzeichnet werden können. Ihr erscheinen innerhalb des Dialogs, kann mithilfe eines Parameters oder einer Funktion gesteuert werden.

```
[WidgetSettings(nameof(Example), WidgetUIScope.WidgetOnPage, true)]  
public virtual bool Example { get; set; }
```

```
[WidgetSettings("Example", WidgetUIScope.WidgetInTab, SkipRenderMethod = nameo  
f(SkipRenderOfWidgetSettings))]  
public virtual bool Example { get; set; }
```

Das `WidgetSettingsAttribute` verfügt über folgende Eigenschaften:

- `UISettingName` Der Name unter dem die Einstellung gespeichert wird
- `Scope` Das `WidgetUIScope` auf das sich die Einstellung bezieht
- `SkipRender` Indikator, ob die Einstellung im Dialog angezeigt werden soll oder nicht (optional)
- `SkipRenderMethod` Methode um die Anzeige im Dialog dynamisch zu steuern

Hilfsmethoden

In der Klasse `WidgetBase` finden sich diverse Methoden zur Interaktion mit den Settings.

```
ShowWidgetSettingsWidgetAction
```

Überschreibbare Methode zur Kontrolle der Anzeige der `WidgetSettingsDialog`-Option für den Anwender

```
GetWidgetSettingOrConfigVal
```

Methode um den für die aktuellen Wert zu erhalten. Diese Methode schaut zuerst, ob ein gültiger Wert des Anwenders vorliegt, falls nicht wird der konfigurierte Wert zurückgegeben. Diese Methode sollte immer dann genutzt werden, wenn der Wert der Eigenschaft verwendet wird.



Der direkte Zugriff auf die Eigenschaft sollte nicht mehr erfolgen

```
GetWidgetSetting
```

Überschreibbare Methode um benutzerspezifische Widgeteinstellungen zu lesen.

bs. RemoveWidgetSetting

Überschreibbare Methode zum Löschen von Settings für den aktuellen Benutzer innerhalb des aktuellen Widgets.

```
RemoveWidgetSettingsForWidgetProperties
```

Überschreibbare Methode zum Entfernen aller Settings des aktuellen Benutzers für alle mit dem `WidgetSettingsAttribut` gekennzeichneten Eigenschaften des aktuellen Widgets.

```
SetWidgetSetting
```

Überschreibbare Methode zum Speichern benutzerspezifischer Widgeteinstellungen.

TypeScript

Auch in Typescript stehen Funktionen zum Umgang mit den Settings zur Verfügung.

Widget

```
GetWidgetSetting
```

Funktion zum Lesen eines benutzerspezifischen Settings

```
SetWidgetSetting
```

Funktion zum Speichern eines benutzerspezifischen Settings

```
RemoveWidgetSetting
```

Funktion zum Entfernen eines benutzerspezifischen Settings

WidgetSettingsUtils

Auch in Typescript steht eine Klasse mit Hilfsfunktionen zur Verfügung. Diese verfügt über folgende Methoden

```
SetWidgetSetting
```

Ermöglicht das Speichern eines Settings.

```
GetWidgetSetting
```

Ermöglicht das Lesen eines Settings.

```
RemoveWidgetSetting
```

Ermöglicht das Entfernen eines Settings.

```
SetDefaultWidgetSettings
```

Speichert alle Default-Settings für ein Widget und einen Benutzer.

```
RemoveAllWidgetSettings
```

Setzt alle Settings für ein Widget und einen Benutzer auf die Defaulteinstellungen zurück.

13.7.6. Widget Vorschau implementieren

Widget

Damit ein Widget in die Übersicht gezogen werden und eine Vorschau anzeigen kann, muss dieses Widget das Interface `IPreviewableWidget` implementieren.

Dieses Interface definiert die beiden Eigenschaften

```
bool ShowsTitleInPreview
string PreviewActionName
```

und die Methode

```
IWidgetPreviewProvider GetWidgetPreviewProvider()
```

welche dann auch im Widget implementiert werden müssen.

bool ShowsTitleInPreview

Diese Eigenschaft bestimmt, ob das Widget in seiner Vorschau einen Titel anzeigen soll oder nicht. Soll es einen Titel anzeigen, so kann diese Eigenschaft von einem Konfigurator oder programmatisch aktiviert werden (abhängig von den Attributen, welche in der Implementierung verwendet werden).

string PreviewActionName

Diese Eigenschaft bestimmt den Namen der Controller-Aktion, welche auf dem zu dem Widget gehörenden Controller aufgerufen wird, um die Vorschau zu rendern. Üblicherweise ist diese Eigenschaft vor Konfiguratoren verborgen und wird programmatisch gesetzt.

IWidgetPreviewProvider GetWidgetPreviewProvider()

Diese Methode ruft den Preview-Provider für das Widget ab, welcher die Daten für die Vorschau liefert. Um einen Preview-Provider anhand seiner Id abzurufen, kann folgende Hilfsmethode verwendet werden:

```
WidgetTools.GetWidgetPreviewProviderById(<id>)
```

Controller

Der Controller muss die im Widget angegebene Aktion implementieren, zum Beispiel

```
public PartialViewResult Preview()
```

Dort wird üblicherweise der Preview-Provider des Widgets abgerufen und auf diesem die Methode `GetPreviewData` mit einem hier ebenfalls erstellten Parameter-Objekt aufgerufen. Welche Inhalte oder von welchem Typ dieses Parameter-Objekt haben muss, hängt vollkommen vom verwendeten Preview-Provider ab.

Der grundsätzliche Ablauf der Preview-Aktion ist also wie folgt:

```
IWidgetPreviewProvider provider = ((IPreviewableWidget) RequestedControl).GetWidgetPreviewProvider();
WidgetPreviewDataModelBase previewModel = provider.GetPreviewData(providerParameters);
return PartialView(provider.PreviewViewPath, previewModel);
```

Die aufgerufene Razor-View kann dann anhand des empfangenen Models eine beliebige UI implementieren.

PreviewProvider:

Eine Klasse wird durch zwei Dinge zu einem Preview-Provider:

- die Klasse implementiert das Interface `IWidgetPreviewProvider` und
- die Klasse hat das Attribut `WidgetPreviewProvider`

IWidgetPreviewProvider-Interface

Das Interface definiert die beiden Eigenschaften

```
string PreviewViewPath { get; }
Type ExpectedParameterType { get; }
```

und die Methode

```
WidgetPreviewDataModelBase GetPreviewData(object parameter)
```

string PreviewViewPath

Gibt den üblicherweise virtuellen Pfad zurück, unter dem die MVC-View (cshtml) liegt, welche die Darstellung der zugehörigen Vorschau beinhaltet, also z.B. `"~/Views/Widgets/MyWidget/Preview.cshtml"`.

Type ExpectedParameterType

Liefert dem aufrufenden System die Information darüber, welcher Typ für den Parameter beim Aufruf von `GetPreviewData` von diesem Preview-Provider erwartet wird.

WidgetPreviewDataModelBase GetPreviewData(object parameter)

Diese Methode macht die ganze Backend-Arbeit und liefert die Daten als Klasse des Typs `WidgetPreviewDataModelBase` zurück, die von der Preview-View verarbeitet werden sollen. Entwickler können selbstverständlich ganz eigene Models implementieren, die von `WidgetPreviewDataModelBase` ableiten, es stehen aber auch bereits einige fertige Models zur Verfügung, welche eine gute Menge an Fällen abdecken dürften. Diese Models verwenden für ihre Werte üblicherweise sog. `WidgetPreviewToken<T>`, welche die notwendigen Daten eines bestimmten Typs aufnehmen können.

Folgende vorgefertigte Models stehen zur Auswahl:

`DefaultGridViewWidgetPreviewModel`, definiert:

```
public List<WidgetPreviewToken<object>> AdditionalInformation
```

`GenericDataWidgetPreviewModel`, definiert:

```
public WidgetPreviewToken<object> Data
```

`MultipleNumbersWidgetPreviewModel`, definiert:

```
public List<WidgetPreviewToken<int>> Numbers // Eine Liste mit Ganzzahl-Preview-Tokens
public WidgetPreviewToken<int> EmptyContent // Ein einzelnes Ganzzahl-Preview-Token, das verwendet werden soll, falls die Liste leer ist
```

`SingleNumberLatestAndNextWidgetPreviewModel`, definiert:

```
public WidgetPreviewToken<DateTime> Latest
public WidgetPreviewToken<DateTime> Next
```

`SingleNumberWidgetPreviewModel`, definiert:

```
public WidgetPreviewToken<int> MainInformation
```

`VariableDataWidgetPreviewDataModel`, definiert:

```
public Dictionary<string, WidgetPreviewToken<int>> DataByKey
```

WidgetPreviewToken

Diese Klasse existiert, da zu einer Information normalerweise mehr gehört als zum Beispiel nur eine Ganzzahl. Hier können neben der Hauptinformation vom Typ noch weitere Informationen für die Preview-View untergebracht werden. Die Verwendung dieser Informationen ist selbstverständlich von

der jeweiligen Implementierung der Vorschau-View abhängig.

`WidgetPreviewToken<T>` enthält folgende Informationen:

- `public string Description`
Die Beschreibung des gelieferten Werts, üblicherweise ein einzelnes Wort oder eine kurze Phrase, welche in der Vorschau angezeigt werden kann.
- `public string Icon`
Ein eventuelles Icon, welches zusätzlich zu anderen Informationen angezeigt werden soll.
- `public T Data`
Die eigentlichen Daten dieses Tokens.
- `public string Tooltip`
Der zu den Daten anzuzeigende Tooltip.
- `public bool IsEmphasis`
Die angezeigten Daten sollen optisch hervorgehoben werden.
- `public bool UseIconAsLabel`
Das Icon soll gar nicht als Bild dargestellt werden sondern der Name des Icons soll als Text ausgegeben werden

WidgetPreviewProvider-Attribut

Die Angabe des Attributs beinhaltet die Angabe einer eindeutigen ID und eines darstellbaren Namens für diesen Preview-Provider und optional n Typen von Orm-Klassen, mit denen dieser PreviewProvider arbeiten kann.

- Die Id wird selbstverständlich verwendet, um den Preview-Provider zu identifizieren und abrufen zu können. Hier hat es sich bewährt, den vollen Typnamen (also inklusive Namespace) der Klasse zu verwenden.
- Der Name wird von dem oben erwähnten Data-Provider verwendet, um den Preview-Provider mit irgendetwas Anderem als seiner Id zur Auswahl anzubieten.
- Die optionalen Typen können von einem ebenso optionalen `CDPModifier` verwendet werden, um die Auswahl der Preview-Provider einzuschränken.

13.7.7. Listen-Widget

Grundlagen

Der Listen-Widget-Framework ist in erster Linie dazu gedacht einen Entwickler in die Lage zu versetzen, einfach und unkompliziert Daten aller Art in einer Liste innerhalb der Anwendung anzeigen lassen zu können.

Er bietet aber ebenso die Möglichkeit komplexer Anpassungen, um beispielsweise – auch grafisch – eigene Filter zu implementieren (wie im Wiedervorlagen-Widget des „BA.FollowUp“-Moduls geschehen) und verschiedenste Aktionen auf die angezeigten Daten auszuführen.

Der Framework besteht aus einem generischen Listen-Widget-Controller mit zugehöriger Listen-Widget-Mvc-View, einer Listen-Widget-TypeScript-Klasse und mehreren auf einander aufbauenden Basisklassen, die ein Entwickler zur Erstellung eines eigenen Listen-Widgets erweitern muss. Die TypeScript-Klasse verwendet ein gesondertes client-seitiges Listensteuerelement, welches selbstständig mit dem Listen-Widget-Controller zusammenarbeitet. Die Kommunikation zwischen Controller und Listensteuerelement erfolgt über HTTP und JSON.

Für die (Meta-)Datenversorgung des Listen-Widgets stehen dem Entwickler zahlreiche Model-Klassen zur Verfügung.

Das Listen-Widget-Steuerelement

Basisklassen und Schnittstellen

Die Basisimplementierung des Listen-Widgets besteht aus vier auf einander aufbauenden Widget-Klassen und einer Schnittstelle `IListWidget`. Die Schnittstelle legt fest, welche Methoden und Eigenschaften ein List-Widget haben muss, allerdings implementieren die Basisklassen viele davon schon selbst, so dass in der Endimplementierung des Entwicklers hauptsächlich die abstrakten Methoden implementiert werden müssen. Alle anderen Methoden können nach Belieben ebenfalls überschrieben werden.

Die Listen-Widget-Basisklassen haben einen – optional zwei – generische Parameter; das und die Vererbungshierarchie sehen wie folgt aus:

Die oberste Vererbungsebene bildet die Klasse

```
public abstract class ListWidget<T, C> : Widget, IListWidget where C : ListControlContextBase, new()
```

Davon erben einmal die Vereinfachung

```
public abstract class ListWidget<T> : ListWidget<T, ListControlContextBase>
```


und die abstrakte Basisklasse

```
public abstract class OrmListWidget<T, C> : ListWidget<T, C> where T : OrmBase
    where C : ListControlContextBase, new()
```

Letztere hat eine weitere Vereinfachung

```
public abstract class OrmListWidget<T> : OrmListWidget<T, ListControlContextBase>
    where T : OrmBase
```

Der generische Parameter `T` bezeichnet immer den Typen der Quelldaten, die dieses Widget anzeigen wird.

In der Grundimplementierung `ListWidget` können diese Daten von einem beliebigen Typen sein, in der konkreteren Implementierung `OrmListWidget` ist dieser Typ auf `OrmBase` (oder konkreter) festgelegt.

Der zweite (optionale) Parameter `C` bezeichnet den Typen für das Laufzeitkontextobjekt, welches in jede in der Schnittstelle definierte Methode hineingegeben und über die Methode `CreateContextq` erstellt wird.

Wird `C` nicht angegeben, so wird standardmäßig `ListControlContextBase` verwendet.

Das implementierte Listen-Widget ist dafür zuständig sowohl die eigentlichen Anzeigedaten als auch die Metadaten für das Listensteuerelement entsprechend der Schnittstelle zu liefern.

Folgende Methoden müssen zwingend vom Entwickler implementiert werden:

```
ListControlColumn[] GetColumns(C context)
```

Liefert eine Reihe von Definitionen für die Spalten, die das Widget in der Liste in der gelieferten Reihenfolge anzeigen soll.

```
IQueryable<T> GetDataQuery(C context)
```

Erzeugt die Basis-Query auf die anzuzeigenden Daten.

```
IOrderedQueryable<T> OrderQuery(IQueryable<T> query, C context)
```

Sortiert die übergebene Query nach Belieben aber auf eine konstante Art und Weise; ist notwendig, um die Paging-Funktionalitäten „Skip“ und „Take“ verwenden zu können.

```
ListControlRowDataBase[] CreateRowData(IQueryable<T> query, C context)
```

Äquivalent zu Datenprovidern wird hier aus der gegebenen Query ein Array von Rückgabedaten erstellt.



Die zur Metadatenbeschaffung gehörenden Standardimplementierungen gehen davon aus, dass ein Laufzeitkontext existiert und dieser in der Eigenschaft `ControllerResult` ein bereits existierendes Objekt vom Typ `ListControlGetMetaData` (oder konkreter) beinhaltet!

Datenbeschaffung / GetData

Die `GetData`-Methode auf dem Widget hat eine Standardimplementierung in `ListWidget<T, C>`, die im Notfall überschrieben werden kann und die üblicherweise von einem Daten-Controller während der Aktion `GetData` aufgerufen wird.

Der Ablauf in dieser Methode ist:

- `GetDataQuery`
- `ApplyFiltersToQuery` (in diesem Zuge dann `ApplyFilter`)
- `CalculateTotalCount` (falls `RequireTotalCount` angefragt wurde)
- `OrderQuery` (mit anschließendem `Skip` und `Take`, das ist nicht anpassbar)
- `CreateRowData`



Die Standardimplementierung geht davon aus, dass der Laufzeitkontext existiert und dieser in der Eigenschaft `ControllerResult` ein bereits existierendes Objekt vom Typ `ListControlGetData` (oder konkreter) beinhaltet!

Der Datenrückgabety `ListControlRowDataBase` der `CreateRowData`-Methode ist leer und abstrakt, d. h. er muss von einem Entwickler, der entsprechend Daten liefern möchte, um die benötigten Eigenschaften erweitert werden. Der Name der Spaltenmetadaten korreliert mit den hier vergebenen Eigenschaftsnamen (siehe Beispiel, Firmenname „Name“).

Die Daten, welche hier zurück geliefert werden, lassen sich grob in zwei Arten aufteilen: Daten, die zur Berechnung und im Hintergrund benötigt werden und Daten, welche tatsächlich angezeigt werden sollen.

Bei ersteren empfiehlt es sich, als Variablentyp im Row-Data-Model einen einfachen und unverfänglichen C#-Typen zu verwenden (string, int, bool, Guid, etc.).

Daten, die angezeigt werden sollen, jedoch unterliegen üblicherweise Formatierungen (Schriftgrößen, ~~arten und -farben~~, Datums und Nummernformatierungen, etc.), daher sollten solche Daten im Row-Data-Model über die Model-Klassen im Unter-Namespace `DataDefinitions` definiert werden.

Als konkrete Beispiele könnten man hier eine Icon-Spalte nennen, die entsprechend so deklariert werden sollte:

```
public ListControlDataImage StatusImage { get; set; }
```

Ein weiteres Beispiel wäre eine Datumsspalte:

```
public ListControlDataDate EndDate { get; set; }
```

Eine solche Deklaration wie die zweite hier sagt also dem Listen-Framework, dass die Daten jeder Zelle in der Spalte `EndDate` eine Standarddatumsformatierung erhalten sollen. Der verwendete Typ im Row-Data-Model definiert also den Zellentypen der Liste.

Das ist wichtig zu verstehen, da jeder dieser Datendefinitionstypen (außer Image) mehrzeilig sein kann und diese Formatierung auf jede Zeile angewendet wird, wenn dies in der zugehörigen Inhaltklasse nicht anders angegeben wird.

Folgende Zellentypen/Datendefinitionstypen stehen zur Verfügung:

- `ListControlDataText`
- `ListControlDataNumber`
- `ListControlDataDate`
- `ListControlDataDateTime`
- `ListControlDataImage`

Image tanzt ein wenig aus der Reihe, hier kann (muss) man bei der Erstellung ausschließlich die Url auf das Bild und einen Tooltip für selbiges angeben:

```
new ListControlDataImage (string url, string title)
```

Mehr kann man hier auch schon nicht machen, daher gelten alle folgenden Informationen für alle anderen Datendefinitionsklassen!

Diese anderen Klassen funktionieren alle mehr oder weniger identisch und haben die gleichen Konstruktoren (daher das Ganze einfach nur am Beispiel `ListControlDataText`):

```
new ListControlDataText(string text)
new ListControlDataText(string text, string cssClass)
new ListControlDataText(params ListControlDataContent[] lines)
```

Der hier gezeigte Parameter `string text` variiert natürlich bei den anderen Inhaltstypen entsprechend.

Die ersten beiden Konstruktoren sind selbsterklärend, es wird in der Zelle ein Text angezeigt und optional mit einer css-Klasse versehen. Beim letzten Konstruktor können beliebig viele Inhaltsobjekte angegeben werden, die mehrere Textzellen darstellen.

Die Klasse `ListControlDataContent` wird intern von den anderen Konstruktoren verwendet, es ist über diesen letzten Konstruktor möglich, Objekte dieses Typs direkt zu übergeben. Instanziiert werden können sie wie folgt:

```
new ListControlDataContent()
new ListControlDataContent(object content)
new ListControlDataContent(object content, string cssClass)
new ListControlDataContent(object content, string cssClass, string convertFunc
tion)
```

Der Inhalt ist hier `object`, da diese Klasse für jeden Inhaltstypen verwendet wird. Es kann eine `css`-Klasse übergeben werden, die auf die entsprechende Zeile angewendet wird, und – und das ist nun besonders interessant – es kann eine JavaScript-Funktion angegeben werden, die client-seitig die String-Formatierung übernimmt. Diese Konvertierungsfunktion wird von manchen Inhaltstypen automatisch gesetzt, wenn sie nicht schon vom Entwickler in einem Content-Objekt selbst angegeben wurde.

Das heißt, der Inhaltstyp `ListControlDataDate` wird die Standardfunktion zur Textkonvertierung für Datumswerte (ohne Zeit) `ListControl.ConvertDateToString` setzen, wenn ich als Entwickler nicht über ein Content-Objekt selbst sage, dass ich eine andere Funktion verwenden will.

Das bringt uns in die Lage, die Inhalte von Zellen auch mischen zu können. Es ist also am Ende möglich, in einer Datumszelle drei Zeilen unterzubringen, bei denen die ersten beiden Zeilen als normales Datum formatiert werden, die dritte aber als Nummer oder normaler Text.

Wie alles andere im Framework lassen sich auch diese Inhaltsklassen sehr leicht um eigene Klassen erweitern. Eine entsprechende Klasse muss den Basistypen `ListControlDataBase` erweitern.

Einfachstes Beispiel eines Listen-Widgets

Das quasi einfachste Listen-Widget, das man implementieren kann, wäre (in diesem Beispiel) eins, das nur eine Liste von Firmennamen anzeigt. Dazu muss es eine Row-Data-Klasse implementieren, die den Firmennamen beinhaltet, es muss die Spalte definieren, die den Firmennamen anzeigen soll, es muss die Query auf die Firmendaten erzeugen, diese sortieren und daraus am Ende Row-Data-Objekte erzeugen.

```
[Serializable]
[Toolbox(EnumConfigurationType.PageConfigurationGuid)]
public class CompanyWidget : OrmListWidget<OrmCompany>
{
    public class CompanyRowBase : ListControlRowDataBase
    {
        public ListControlDataText Name { get; set; }
    }

    public CompanyWidget()
    {
        Id = "[INSERT YOUR WIDGET GUID]".ToGuid();
        ToolboxName = "Company-Widget";
    }
}
```

```

        protected override ListControlColumn[] GetColumns(ListControlContextBase context)
        {
            return new ListControlColumn[]
            {
                new ListControlColumn { DataField = nameof(CompanyRowBase.Name), MinWidth = 200 }
            };
        }

        protected override IQueryable<OrmCompany> GetDataQuery(ListControlContextBase context)
        {
            return Api.ORM.GetQuery<OrmCompany>();
        }

        protected override IOrderedQueryable<OrmCompany> OrderQuery(IQueryable<OrmCompany> query, ListControlContextBase context)
        {
            return query.OrderBy(ff => ff.Name);
        }

        protected override ListControlRowDataBase[] CreateRowData(IQueryable<OrmCompany> query, ListControlContextBase context)
        {
            return query.Select(ff => new CompanyRowBase {
                Name = new ListControlDataText(ff.Name)
            }).ToArray();
        }
    }

```

Das Beispiel um Aktionen erweitern

Im Prinzip gibt es drei Arten von Aktionen, die in solch einem Widget verwendet werden können:

- Aktion bei Einzelklick auf einen Listeneintrag
- Aktion bei Doppelklick auf einen Listeneintrag
- Aktionen, die beim Betreten mit dem Mauszeiger angezeigt und per Einzelklick aktiviert werden können

Um diese Aktionen nutzen zu können, müssen die folgenden Methoden im Widget entsprechend überschrieben werden:

```

protected override ListControlRowAction[] GetRowActions(ListControlContextBase context)

```

```

{
    ListControlRowAction[] actions = new ListControlRowAction[]
    {
        new ListControlRowAction
        {
            ActionMethodId = "My.Edit.ActionId",
            Title = "Edit",
            IconUrl = IconHelper.GetIconUrl("pencil", RequestIconSizeEnum.Size32x32, ColorTranslator.FromHtml("#4D4D4D"))
        },
        new ListControlRowAction
        {
            ActionMethodId = "My.Finish.ActionId",
            Title = "Finish",
            IconUrl = IconHelper.GetIconUrl("ok", RequestIconSizeEnum.Size32x32, ColorTranslator.FromHtml("#4D4D4D"))
        }
    };

    return actions;
}

protected override string GetRowClickActionMethodId(FollowUpContext context)
{
    return "My.SingleClick.ActionId";
}

protected override string GetRowDoubleClickActionMethodId(FollowUpContext context)
{
    return "My.DoubleClick.ActionId";
}

```



Die Aktionen für Einzelklick und Doppelklick sollten nicht gemeinsam verwendet werden, weil technisch bedingt ein Doppelklick immer auch mindestens einen Einzelklick auslöst. Wenn unbedingt notwendig, kann dieses Problem durch kluge Programmierung der beiden Aktionen umgangen werden.



Wenn hier `ActionId` steht, wird hier die Id einer Action erwartet, wie sie bei den Ribben bar Aktionen definiert wurden.



Die Implementierung der Aktion bekommt als Parameter den Click-Event (1) und die

Daten der Zeile (2) übergeben. Sollten Berechnungen und/oder Bedingungen behandelt werden müssen, so müssen entsprechende Daten in den Zeilendaten (RowData) vorgesehen werden (wenn es kein Column-Objekt dafür gibt, werden sie nicht angezeigt, aber vorhanden sind sie!).

Das Beispiel um Filter erweitern

Häufig will man gar nicht alle Daten anzeigen sondern den Benutzern die Möglichkeit geben, diese Daten auf bestimmte Fälle einzuschränken. Hierfür benötigt man Filter, und um diese definieren und verwenden zu können, muss im Widget Folgendes implementiert werden:

```
protected override ListControlFilterBase[] GetFilters(ListControlContextBase context)
{
    ListControlFilterBase[] result = new ListControlFilterBase[]
    {
        new ListControlFilterSelect
        {
            Id = "[INSERT FILTER GUID]".ToGuid(),
            Width = 170,
            Items = new ListControlFilterItem[]
            {
                new ListControlFilterCriteriaItem
                {
                    Caption = "All companies",
                    Key = "all",
                    Criteria = new OperandValue(true)
                },
                new ListControlFilterCriteriaItem
                {
                    Caption = "Name starts with 'A'",
                    Key = "a",
                    Criteria = new FunctionOperator(FunctionOperatorType.StartsWith, new OperandProperty(nameof(OrmCompany.Name)), new OperandValue("A"))
                },
            },
            DefaultSelectedKey = "all",
        }
    }
}
```

Dieses Beispiel sorgt dafür, dass in der Toolbar der Liste eine Combobox mit den Einträgen „All companies“ und „Name starts with ‚A‘“ erscheint. Die Verwendung von `ListControlCriteriaFilterItems` ist hier möglich, da es sich bei dem Widget um die Ableitung eines `OrmWidgets` handelt,

Formel-Filterelemente können im Standard automatisch ausgeführt bzw. auf die Query angewendet werden.

Wäre die Query nicht von einem `OrmBase`-Typen, so könnten nur Filterelemente des Typs `ListControlFilterItem` verwendet werden. In diesem Fall müsste zusätzlich die Methode `ApplyFilter` überschrieben werden, um entsprechende Filterelemente auf die Query anzuwenden.

Wenn ganz eigene Filter und Filterelementtypen zum Einsatz kommen oder Abläufe angepasst werden müssen, kann es auch sein, dass zusätzlich noch die Filterverwaltungsmethode `ApplyFiltersToQuery` überschrieben werden muss, die sich dann um die Anwendung aller aktiven Filter kümmern muss.

Wer interessiert ist und Zugriff auf den Quellcode hat: Das Wiedervorlagen-Widget im Modul „BA.FollowUp“ ist hierfür ein Beispiel.

Tiefergehende Anpassungen

Für komplexe Anpassungen können im Konstruktor der Widget-Klasse auch ein eigener Listen-Controller (Eigenschaft `ControllerName`) und eine eigene TypeScript-Widget-Klasse („`GetTSClassName()`“) angegeben werden, wobei Letzteres allerdings Teil des generellen Widget-Frameworks ist.

Der Listen-Widget-Controller und die Listen-Widget-Mvc-View

Der `ListWidgetController` ist ein Standard-Widget-Controller, der die für die Versorgung eines Listensteuerelements notwendige Schnittstelle `IListControlDataSource` implementiert und daher mit einem Listensteuerelement verwendet werden kann. Als Widget-Controller implementiert er zusätzlich eine `Index`-Aktion, die beim Aufruf des Widgets aufgerufen wird. Die Methode `GetMetaData` ist zwar vollständig implementiert, sie wird aber in der Standardauslieferung nicht verwendet.

Im Widget-Fall werden die Metadaten in der `Index`-Methode von der Listen-Widget-Implementierung gesammelt, über ein View-Model an die `Mvc-View` weitergegeben und dort in `data`-Attributen des Listen-Basis-DIV-Elements (`anchor`) hinterlegt.

Die generelle Abfolge beim Abruf der Metadaten, also auch bei „`GetMetaData`“, ist folgende:

- `CreateContext`
- `GetRowActions`
- `GetColumns`
- `GetFilters`
- `GetRowClickActionMethodId`
- `GetRowDoubleClickActionMethodId`
- `GetKey`
- `GetLocale`

Beim Abruf von Daten, sei es durch Scrollen der Liste, Anwenden eines Filters oder einfach nur durch Drücken des Refresh-Knopfes, wird die Aktion `GetData` aufgerufen. Hier ist der Ablauf wie folgt:

- `CreateContext`
- `GetData`

Die Standardimplementierungen der Daten- und Metadaten-Methoden auf den Listen-Widget-Basisklassen gehen davon aus, dass der übergebene Laufzeitkontext nicht `null` ist und die Eigenschaft `ControllerResult` ein gültiges, vorgefertigtes Rückgabeobjekt beinhaltet.

Dieses Objekt ist für Daten vom Typ `ListControlGetData` und für Metadaten vom Typ `ListControlGetMetaData` (oder jeweils konkreter).

Die ListWidget-TypeScript-Klasse

Die zu jedem Widget gehörende TypeScript-Klasse erbt von der Klasse `Widget` und überschreibt im Falle des Listen-Widgets die Funktion `OnContentLoaded`, um dort das client-seitige Listensteuerelement zu erstellen.

Diese Klasse kann bei Bedarf durch eine eigene Klasse erweitert werden; in einem solchen Fall muss die neue Klasse über die Eigenschaft `GetTSClassName()` der Widget-Implementierung bekannt gegeben werden.

Die TypeScript-Klasse beinhaltet folgende Funktionen, die in abgeleiteten Klassen überschrieben werden können:

```
OnContentLoaded(): void
```

Ermittelt die Erstellparameter für das Listensteuerelement durch einen Aufruf an `GetCreateParameters` und erstellt anschließend das Listensteuerelement durch den Aufruf von `CreateListControl`.

```
GetControlId(): string
```

Berechnet die Id der DevExtreme-Komponente des Listensteuerelements.

```
GetListContainer(): JQuery
```

Ermittelt das JQuery-Element des Listenbasiselements (anchor). Ruft `GetControlId` auf.

```
GetCreateParameters(): BA.Ui.ClientSideControls.List.ListControlCreateParameters
```

Ermittelt die Erstellparameter aus den data-Attributen des Listen-Containers. Ruft `GetListContainer` und `InitCustomFilters` auf.

```
CreateListControl(createParameters: BA.Ui.ClientSideControls.List.ListControlCreateParameters): BA.Ui.ClientSideControls.List.ListControl | JQueryPromise<B
```

```
A.Ui.ClientSideControls.List.ListControl>
```

Erstellt das Listensteuerelement. Der Rückgabewert hängt von der in abgeleiteten Klassen verwendeten Erstellmethodik ab, im Standard ist dies aber immer einfach nur `ListControl`.

```
InitCustomFilters(_ : BA.Ui.ClientSideControls.List.ListFilter[]) : void
```

Erlaubt die grafische Initialisierung eigener Filter (für Beispiel siehe Wiedervorlagen-Widget im Modul „BA.FollowUp“).

Anhang (Klassenbeschreibungen)

Alle C#-Model-Typen befinden sich unterhalb des Namespace `BA.Core.Models.ListControlModels`.

Controller-Models (C#)

ListControlGetData

Das Model ist selbst ein `ActionResult` und wird automatisch nach JSON serialisiert.

- `int? TotalCount`
Die Gesamtzahl an möglichen Datensätzen, die in der Liste angezeigt werden könnten, wenn man ganz nach unten scrollt. Muss nur befüllt werden, wenn dies bei der Anfrage entsprechend angegeben war (s. `RequireTotalCount`).
Hinweis: Wird als `totalCount` an den Client ausgeliefert.
- `ListControlRowDataBase[] Data`
Array von Daten, welche an den Client ausgeliefert werden. Dies sind ALLE Daten, die der Anforderung entsprechen, sie werden aber nicht zwingend alle angezeigt (von den Metadaten abhängig).
- `Dictionary<string, object> CustomProperties`
Zusätzliche Daten, die von Anpassungen verwendet werden können, im Standard ist dieses Dictionary leer.
Hinweis: Die verwendeten Werttypen müssen für eine ordnungsgemäße Funktion serialisierbar sein!

ListContolGetMetaData

Das Model ist selbst ein `ActionResult` und wird automatisch nach JSON serialisiert.

- `public string RowClick { get; set; }`
`ActionMethodId`, welche ausgeführt wird, wenn eine Zeile per Einzelklick angeklickt wird.
Hinweis: Wird als `rowclick` an den Client ausgeliefert.
- `public string RowDoubleClick { get; set; }`
`ActionMethodId`, welche ausgeführt wird, wenn eine Zeile per Doppelklick angeklickt wird.
Hinweis: Wird als `rowdoubleclick` an den Client ausgeliefert.
- `public ListControlColumn[] Columns { get; set; }`

Die Spaltendefinitionen, die das Listensteuerelement anzeigen soll.

Hinweis: Wird als `columns` an den Client ausgeliefert.

- `public ListControlFilterBase[] Filters { get; set; }`

Die Filter, die das Listensteuerelement anzeigen soll.

Hinweis: Wird als `filters` an den Client ausgeliefert.

- `public ListControlRowAction[] RowActions { get; set; }`

Die Zeilenaktionen, die das Listensteuerelement anzeigen soll.

Hinweis: Wird als `rowactions` an den Client ausgeliefert.

- `public Dictionary<string, object> CustomProperties { get; set; } = new Dictionary<string, object>();`

Zusätzliche Daten, die von Anpassungen verwendet werden können, im Standard ist dieses Dictionary leer.

Hinweis: Wird als `customproperties` an den Client ausgeliefert.

- `public bool ShowTotalCount { get; set; }`

Flag, ob unterhalb der Liste eine Zeile mit der Gesamtzahl an Listeneinträgen angezeigt werden soll.

Hinweis: Wird als `showtotalcount` an den Client ausgeliefert.

- `public string Locale { get; set; }`

Das Kürzel für die Spracheinstellungen des Benutzers (z. B. "en").

Hinweis: Wird als `locale` an den Client ausgeliefert.

- `public string Key { get; set; }`

Name der Spalte, deren Inhalt für einen Eintrag als Schlüssel dient, dieser Wert ist im Standard leer.

Hinweis: Wird als `key` an den Client ausgeliefert.

ListControlGetDataOptions

Das Model ist eine Zusammenfassung von Parametern, welche vom Listensteuerelement beim Holen von Daten mitgeschickt werden, um den Abruf der Daten zu konkretisieren.

- `int Skip`

Anzahl an Datensätzen, die vom Beginn der Daten übersprungen werden sollen, bevor das Ergebnis befüllt wird. Dient dem Paging der DevExtreme-Komponente.

- `int Take`

Anzahl an Datensätzen, die ab dem ermittelten Beginn der Daten zurückgeliefert werden sollen. Dient dem Paging der DevExtreme-Komponente.

- `bool RequireTotalCount`

True, wenn die DevExtreme-Komponente zum erfolgreichen Verarbeiten der Anforderung die mögliche Gesamtzahl aller Daten benötigt.

- `string ActiveFilters`

JSON-Serialisiertes Objekt des Typs `Dictionary<Guid, string>`, welches eine Zuordnung zwischen einer Filter-Id und dem zu diesem Filter aktuell aktiven Filtereintrag (Key) enthält.

Metadaten-Models (C#)

ListControlColumn

Spaltendefinitionen `ListControlColumn` legen grundsätzlich fest, welche Daten in welcher Reihenfolge angezeigt werden und wie breit diese Spalten sein sollen. Objekte dieses Typs werden

sofort in ein Format umgewandelt, welches client-seitig direkt von der DevExtreme-Komponente als Spaltendefinition verwendet werden kann.

Der Typ beinhaltet häufig verwendete Eigenschaften, kann aber über „CustomProperties“ beliebige der von der Komponente angebotenen Eigenschaften abbilden:

- `string DataField`
Name der Eigenschaft der Zeilendaten (Ableitung von `ListControlRowDataBase`), welche in dieser Spalte dargestellt werden soll.
Hinweis: Wird als `dataField` an den Client ausgeliefert.
- `int? Width`
Genaue Breite der Spalte in Pixeln (fix), muss nicht zwingend gefüllt sein (dynamisch).
Hinweis: Wird als `width` an den Client ausgeliefert.
- `int MinWidth = 20`
Minimale Breite der Spalte in Pixeln, verwendet 20px, wenn nichts anderes angegeben wurde.
Hinweis: Wird als `minWidth` an den Client ausgeliefert.
- `bool Visible = true`
Flag, ob die Spalte in der UI sichtbar ist oder nicht.
Hinweis: Wird als `visible` an den Client ausgeliefert.
- `string Alignment`
Ausrichtung des Spalteninhalts, akzeptiert „left“, „center“ und „right“.
Hinweis: Wird als `alignment` an den Client ausgeliefert.
- `Dictionary<string, object> CustomProperties`
Inhalte in diesem Dictionary werden auf das Rückgabeobjekt erweitert, das heißt, dass Entwickler hier einfach zusätzliche Eigenschaften hineinlegen können, die dann von der DevExtreme-Komponente direkt ausgewertet werden. [Dokumentation](#)

ListControlFilterBase / ListControlFilterMultiItemBase / ListControlFilterSelect

Filterdefinitionen `ListControlFilterBase` definieren jeweils einen Filter, der eine oder mehrere Auswahlmöglichkeiten (FilterItems) hat. Diese Basisklasse ist abstrakt und muss durch eine konkrete Implementation erweitert werden; im Standard ist dies `ListControlFilterMultiItemBase` (ebenfalls abstrakt und definiert Filter mit mehreren Filtereinträgen) und davon abgeleitet `ListControlFilterSelect`, welches der Typ ist, der im Standard final an dieser Stelle verwendet werden kann. Dieser enthält folgende Eigenschaften:

- `Guid Id`
Identifikations-GUID des gesamten Filters.
Hinweis: Wird als `id` an den Client ausgeliefert.
- `string FilterType`
Typ des Filters. Unterstützt im Standard aktuell „selection“ und „custom“. Wenn „custom“ gesetzt wird, muss außerdem `CustomFilterType` mit einem eigenen frei wählbaren Typnamen belegt werden.
Hinweis: Wird als `type` an den Client ausgeliefert.
- `string CustomFilterType`
Eigener Typname, wenn `FilterType` „custom“ ist.
Hinweis: Wird als `customType` an den Client ausgeliefert.
- `int Width`

Breite des UI-Elements. Wird bei „custom“ nicht zwingend ausgewertet (Entwickler ist dafür zuständig).

Hinweis: Wird als `width` an den Client ausgeliefert.

- `ListControlFilterItem[] Items`

Enthaltene Filterelemente.

Hinweis: Wird als `items` an den Client ausgeliefert.

- `string DefaultSelectedKey`

Schlüssel (Key) eines Filterelements, welches direkt nach dem erstellen der Liste automatisch ausgewählt sein soll.

Hinweis: Wird als `defaultSelected` an den Client ausgeliefert.

Der Typ muss außerdem die Methode

```
ListControlFilterItem GetItem(string key = null)
```

implementieren, um ein konkretes Filterelement aus dem Filter abrufen zu können.

ListControlFilterItem

Der Typ definiert einen einzelnen Filtereintrag, der in der Ui angezeigt werden kann.

- `Caption`

Anzeigetext des Filtereintrags

- `Key`

Hintergrundwert des Filtereintrags, der auch zum Server kommuniziert wird.

Hiervon abgeleitet ist:

ListControlCriteriaFilterItem

Der Typ implementiert zusätzlich zu `ListControlFilterItem`

- `Criteria`

Ein programmierter `CriteriaOperator` (Formel), der verwendet werden kann, um diesen Filter automatisch auf eine Query eines Orm-Typs anzuwenden.

Hinweis: Funktioniert nicht mit nicht-Orm-Queries, dieser Parameter wird nicht serialisiert.

ListControlRowAction

Der Typ definiert eine Aktion, welche auf einer Datenzeile ausgeführt werden kann. Das beinhaltet die auszuführende Aktion selbst sowie verschiedene Anzeigedaten.

- `Guid Id`

Eindeutige ID der Aktion.

- `string Title`

Text des Tooltips, der über dem Icon der Aktion angezeigt wird.

- `string IconUrl`

URL des Icons, welches in der UI als Schaltfläche der Aktion verwendet wird.

- `string ActionMethodId`
ActionMethodId, welche über den Action-Framework aufgerufen wird, wenn die Aktion geklickt wird.
- `string VisibilityFunction`
Optional: Name einer JavaScript-Funktion, welche aufgerufen wird, um zu entscheiden, ob die entsprechende Aktion client-seitig gerendert wird oder nicht.
- `Dictionary<string, object> ActionParameters`
Optional: Key/Value-Paare, welche serverseitig ermittelt an die Aktion übergeben werden sollen. Diese stehen dann als CustomData-Objekt innerhalb der übergebenen Daten unter dem Schlüssel `ActionParameters` zur Verfügung.

Datenbezogene Models (C#)

Siehe `GetData` in den vorherigen Abschnitten

Wichtige client-seitige Typen (TypeScript)

ListControlCreateParameters

Wird zur Erstellung des Listensteuerelements verwendet.

- `anchor: ListContainer`
Ein HTML-DIV-Element, welches als Basiselement für das DevExtreme `dxDataGrid` verwendet wird.
- `locale: string`
Das Sprachkürzel des Benutzers.
- `listDataController: string`
Name des Controllers, der die Daten für die Liste liefert. Das ist wirklich nur der Name (`ListData`, ohne `Controller`), der Controller muss sich in der Applikation befinden.
- `rowActions: ListRowAction[]`
Array mit Definitionen der Aktionen, welche über jeder Zeile eingeblendet werden, wenn der Benutzer mit der Maus darüber fährt.
- `columns: ListColumn[]`
Array mit Definitionen der anzuzeigenden Spalten.
- `toolbarItems: ListToolbarItem[]`
Array mit Definitionen der anzuzeigenden Toolbar-Items.
- `rowClickMethodId: string`
ActionMethodId der Aktion, die beim einfachen Klicken einer Listenzeile ausgeführt werden soll.
- `rowDoubleClickMethodId: string`
ActionMethodId der Aktion, die beim doppelten Klicken einer Listenzeile ausgeführt werden soll.
- `showTotalCount: boolean`
Flag, ob unter der Liste noch eine Zeile mit der Gesamtzahl an Zeilen angezeigt werden soll.
- `additionalServiceParameters: CustomData`
Liste von Key/Value-Paaren, die bei Anfragen an den angegebenen Controller immer genau so mitgeschickt werden sollen.
- `customProperties: CustomData`
Zusätzliche Daten bei Anpassungen

ListContainer

Das Interface erbt von `HTMLDivElement` und beinhaltet nach der Listenerstellung und während des Betriebes folgende Informationen:

- `customProperties: CustomData`
Zusätzliche Daten bei Anpassungen
- `ActiveFilters: CustomData`
Enthält alle derzeit aktiven Filterelemente (`Key`) pro Filter (`Id`).
- `ListInstance: List`
Die erstellte Instanz des `dxDataGrid`

List

Das Interface `List` erbt momentan einfach nur von `DevExpress.ui.dxDataGrid` und hat keine weiteren Inhalte. [Referenz](#)

ListRowAction

Das Interface definiert eine Aktion, welche auf einer Datenzeile ausgeführt werden kann und entspricht in seinen Inhalten dem C#-Typen `ListControlRowAction` (siehe C#-Typ: `ListControlRowAction`).

Die Angabe von `VisibilityFunction` und `ActionParameters` ist optional.

ListColumn

Das Interface `ListColumn` erbt momentan einfach nur von `DevExpress.ui.dxDataGridColumn` und hat keine weiteren Inhalte. Es entspricht den Inhalten von `ListControlColumn` (s. C#-Typ: `ListControlColumn`) und wird direkt vom `dxDataGrid` interpretiert. [Referenz](#)

ListToolbarItem

Das Interface `ListToolbarItem` erbt von `DevExpress.ui.dxToolbarItem` und beinhaltet als zusätzliche Information

```
type: string
```

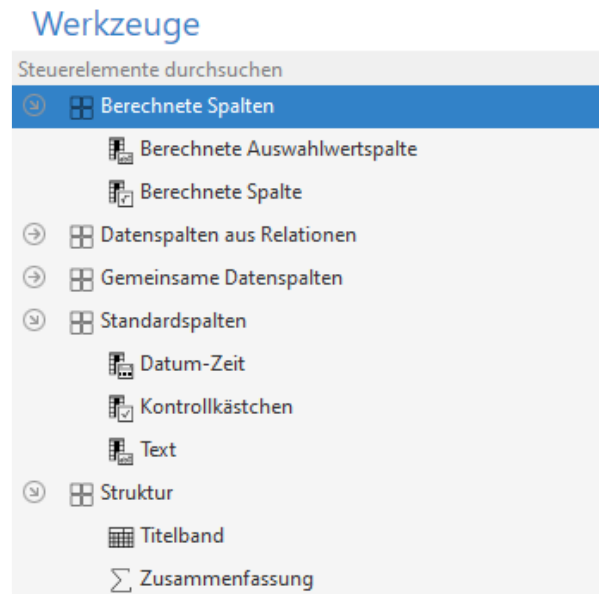
Objekte des Typs `ListControlFilterSelect` (siehe C#-Typ: `ListControlFilterBase / ListControlFilterMultiItemBase / ListControlFilterSelect`) und `ListControlFilterItem` (siehe C#-Typ: `ListControlFilterItem / ListControlFilterCriteriaItem`) werden unter Zuhilfenahme der statischen Funktion `CreateFilter` auf der Klasse `List.ToolbarItems` in Objekte dieses Typs konvertiert.

Es ist möglich (wird aber im Standard nicht getan) der Toolbar außer den Filtern des Listen-Frameworks noch weitere, eigene Elemente hinzuzufügen. [Referenz](#)

14. Ansichten

14.1. 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.

14.1.1. Klasse des Steuerelements

Wie in den Kapiteln über [Ribbon bar Aktionen](#) und [Maskensteuerelemente](#) 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 `GridColumnBase` 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 `EnumDataTypes.String` für Textfelder..
- `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(EnumDataTypes.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;  
}
```

14.1.2. 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, dass 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(configuration, myTextColumn);
        Type dataType = GridRendererHelper.GetDataColumnBaseDataType(configuration, myTextColumn);
        CriteriaOperator criteria = GridRendererHelper.GetDataColumnBaseCriteria(configuration, myTextColumn);
        return new[] { new GridViewQueryColumnProperties { Name = columnName, DataType = 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.GetDataColumnBaseCriteria(configuration, myTextColumn);
    CriteriaOperator expression = fieldValue == new OperandValue(null);
    CriteriaOperator ifpart = new OperandValue(myTextColumn.DefaultValue);
    CriteriaOperator elsepart = fieldValue;
    criteria = new FunctionOperator(FunctionOperatorType.Iif, expression, ifpart, elsepart);
}
else
    criteria = GridRendererHelper.GetDataColumnBaseCriteria(configuration, myTextColumn);

```

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.CreateDataBaseRenderColumn(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(GridConfiguration configuration, GridColumnBase column)  
{  
    return true;  
}
```

14.2. Formeln bassierende Spalten

Spalten die auf Formeln bassieren auf `CalculatedColumnBase`, dieses Steuerelement hat einen Standard-Renderer `CalculatedColumnBaseDefaultRenderer`. Damit können eigene Spalten auf Basis einer Formel implementiert werden.

Neben dem Setzen vom `ColumnType`, muss auch die Methode `GetDataType` überschrieben werden. Diese definiert den Datentyp welcher durch die Formel zurückgegeben wird.

Die Formel selbst wird in `GetCriteria` implementiert. In diesem Beispiel wir die Anzahl der Datensätze in einem Ordner gezählt. Dabei werden die Leserechte auf die Datensätze berücksichtigt.

```
[Serializable]
[Toolbox(EnumConfigurationType.GridConfigurationGuid)]
public class CalculatedRecordCollectionCount : CalculatedColumnBase
{
    public CalculatedRecordCollectionCount()
    {
        ToolboxName = "Record collection count column";
        Caption = "Record collection count column";
        ControlInitName = "RecordCollectionCount";
        ToolboxGroupName = Constants.Designer.ToolboxGroups.Grid.Basic;
        Icon = "sum";
        Id = [INSERT UNIQUE GUID].ToGuid();
        ColumnType = Api.Enum.EnumValueConverter.GetEnumColumnTypeValue(EnumDataTypes.Int32);
    }

    public override CriteriaOperator GetCriteria(bool extendedValidation)
    {
        // Bedingung für den OrmBABase join
        CriteriaOperator recordCondition = new OperandProperty("^." + nameof(OrmRelation.Target)) == new OperandProperty(nameof(OrmBABase.Oid)) & new OperandProperty(nameof(OrmBABase.IsInRecycleBin)) == new OperandValue(false);
        CriteriaOperator readCondition = CriteriaOperatorBuilder.ApplyReadPermissions(null, typeof(OrmBABase), new[] { EnumDataSource.BABase.ValueGuid });
        if (!CriteriaOperatorHelper.IsNull(readCondition))
            recordCondition &= readCondition;

        // OrmBABase join
        JoinOperand recordJoin = new JoinOperand(nameof(OrmBABase), recordCondition, Aggregate.Exists, null);

        // Bedingung für den OrmRelation join
        CriteriaOperator relationCondition = new OperandProperty(nameof(OrmRelation.Source)) == new OperandProperty("^." + nameof(OrmBABase.Oid)) & new Oper
```

```
andProperty(nameof(OrmRelation.RelationType)) == new OperandValue(EnumRelation
Type.RecordCollection.ValueGuid);
    relationCondition &= recordJoin;

    // Rückgabe des OrmRelation joins
    return new JoinOperand(nameof(OrmRelation), relationCondition, Aggregate
te.Count, null);
}

public override Type GetDataType(GridViewConfiguration gridConfiguration)
{
    return typeof(int);
}
}
```


14.3. 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.

Definition der Joins

Für die Definition der Joins muss die Methode `GetJoins` implementiert werden. In dem Beispiel werden zwei Joins definiert. Der Erste joined die Relationstabelle und beschränkt diese auf eine bestimmte Relation. Die Zweite joined den Datensatz der die Quelle der Relation ist.

Wird eine Ansicht mehrfach gruppiert, muss sichergestellt sein, das die Row Id **eindeutig** ist. Wenn dies nicht der Fall ist, ist die Ansicht in einem undefinierten Zustand und es können dann beispielsweise leere Zeilen erscheinen. In einer nicht mehrfachgruppierten Ansicht entspricht die Row Id der Oid des dargestellten Datensatzes. Bei einer mehrfachgruppierten Ansicht ist die Oid nicht mehr eindeutig. Daher muss beim Join angegeben werden ob die Oid des Join Queries als Teil der Row Id verwendet werden soll.

In dem Beispiel wird die Oid der Relation als Row Id genutzt `RowIdPart = true`. Diese ist in diesem Fall dann eindeutig.

```
public IEnumerable<GridViewQueryJoinProperties> GetJoins(GridColumnBase column)
{
    return new[]
    {
        new GridViewQueryJoinProperties
        {
            // Eindeutiger Name des Joins
            Name = "RelationJoin",
            // Der .Net Typ der Orm Klasse
            EntityType = typeof(OrmRelation),
            // Eigenschaft des übergeordneten Queries als Key für diesen Join
            MainKey = nameof(OrmBABase.Oid),
            // Eigenschaft des Join Queries als Key für diesen Join
            Key = nameof(OrmRelation.Target),
            // Bedingung zur Einschränkung der Menge in Formel
            Where = CriteriaOperatorBuilder.GetRelationJoinCondition(true, false, relationType, relationCategory),
        }
    };
}
```

```

        // Flag ob die Oid des Join Queries in die Row Id mit aufgenommen
        werden soll.
        RowIdPart = true,
    },
    new GridViewQueryJoinProperties
    {
        Name = "SourceJoin",
        EntityType = Api.ORM.GetOrmTypeCacheValue(dataSourceGuid),
        MainKey = nameof(OrmRelation.Source),
        Key = nameof(OrmBABase.Oid),
    }
};
}

```



Die definierten Joins werden von dem System in LINQ Joins umgewandelt. DevExpress erlaubt in LINQ Queries eine maximale Tiefe von zwei Joins. Dies muss in jedem Fall an dieser Stelle beachtet werden.

Datenbankabfrage

Soll die Formel für eine Eigenschaft in `GetQueryProperties` auf den Datensatzverweisen, welcher über die Joins hinzugebunden ist, muss in den `GridViewQueryColumnProperties` die Eigenschaft `UseMultiGroupJoins` auf `true` gesetzt werden.

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);
}

```

14.4. 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(GridDataProviderParameter parameter = null)
    {
        if (Properties is GDPMyOwnProperties myProperties)
        {
            Session session = Api.ORM.GetNewSession();
            IQueryable<OrmUserProfile> query;
            if (myProperties.WithReadPermissions)
                query = Api.ORM.GetQueryWithReadPermissions<OrmUserProfile>(session);
            else
                query = Api.ORM.GetQuery<OrmUserProfile>(session);

            if (string.IsNullOrEmpty(myProperties.ConfiguredValueFilter))
                query = query.Where(ff => (string)ff.GetMemberValue("ConfiguredValue") == myProperties.ConfiguredValueFilter);
            else
            {
                CriteriaOperator criteriaWhere = new OperandProperty("ConfiguredValue") != new OperandValue(null);
                query = query.Where(ff => ff.Fit(criteriaWhere));
            }

            CriteriaOperator expression = new OperandProperty("ConfiguredValue") == new OperandValue(myProperties.ConfiguredValueIf);
            CriteriaOperator ifpart = new OperandValue("Replaced value");
            CriteriaOperator elsepart = new OperandProperty("ConfiguredValue");

            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,
    ShowSearchPanel = false,
    ShowFilterRowForColumns = false,
    // Setzen des eigenen Datenproviders
    DataProviderProperties = new GDPMyProviderProperties()
};

```

15. Workflow-Aktionen

Mit Hilfe von eigenen Aktionen kann der Workflow erweitert werden.

15.1. 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]
[BAResquired]
[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 configuration, EnumDesignerValidationStateValue validationState);  
{  
    var result = base.IsValid(configuration, validationState);  
    if (blöderFehler...)  
        result.Add(new ValidationModel(...));  
    return result;  
}
```


15.2. 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`).

15.3. Umgang mit Workflow-Aktionen

Alle Workflow-Aktionen zu einer Datentabelle laufen in einer `UnitOfWork`, die erst am Ende, wenn alle Aktivitäten ausgeführt wurden, `committed` 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.

16. 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.

16.1. 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 `IR`eadOnlyDictionary<string, string> mit den Argumenten. Bei der Benennung 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<stri
ng, string> args)
{
    return "Place holder value";
}
```

HTML einbetten

Um generiertes HTML interpretiert einzubetten. Kann an Jedem Platzhalter der Parameter `type="html"` angegeben werden. Falls die eigene Methode HTML generiert und auch ohne die Angabe des Parameters das HTML interpretiert eingebettet werden soll, muss man das Attribut `[HTMLResult]` an der Methode angeben.

16.2. Eigene Datenquelle

Datenquellen in Platzhaltern dienen als Ausgangspunkt für die Eigenschaften, Relationen und die berechneten Eigenschaften. Damit wird beispielsweise vermieden, dass 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(), sourceOrm != null ? sourceOrm.Session : Api.ORM.GetDefaultSession());
    }
}
```



Achten sie darauf, dass sowohl `sourceOrm` als auch `parentOrm` null sein können.

17. Migration

Migrationen erlauben die Übernahme von Datenbanken und Konfigurationen aus älteren Programmversionen.

17.1. 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.

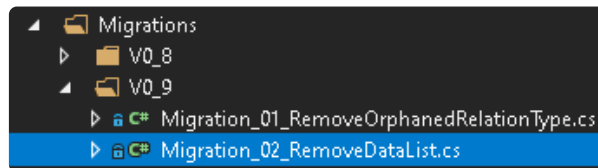
Sollte V0.8.2 unmittelbar nach V0.9 erschienen und eine Kompatibilität dennoch gewährleistet sein, muss das mit einem weiteren Attribut explizit erlaubt werden. Und zwar muss die später erschienene Version V0.8.2 ein Upgrade auf V0.9 erlauben. Es können auch mehrere Versionen erlaubt werden.

```
[assembly: MigrationCompatibilityTarget("0.9")]
```

Dieses Attribut ist beim Erscheinen eines neuen Service-Release im Allgemeinen zu entfernen oder zumindest neu zu bewerten.

17.2. 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 *OrmConfiguration*
- 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 executionContext)
{
    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 executionContext)
{ ... }
```

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 : ConfigurationMigrationBase
{
    public Migration_01_RemoveOrphanedRelationType() : base(EnumConfigurationType.RelationConfiguration) { }
    protected override void MigrateConfiguration(EnumConfigurationType type, Guid 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.

17.3. Migration von gespeicherten Ansichten (ab 7.0.2)

Wenn Projektseitig per Zip-Datei Grids ausgeliefert werden, bei denen Spalten-Steuerelemente durch andere Spalten-Steuerelemente ersetzt werden, gehen in diesem Moment dazu gespeicherte Ansichten "kaputt" in dem Sinne, dass die ursprüngliche Spalte nicht mehr angezeigt wird und die neue Spalte nur in der Spaltenauswahl vorhanden ist, aber nicht standardmäßig sichtbar ist.

Um dieses Problem zu umgehen, kann eine Config-Migration geschrieben werden, die in den gespeicherten Ansichten die alten IDs der Spalten durch die neuen ersetzt. Um die Migration zu vereinfachen, gibt es eine neue Hilfsfunktion:

`ClientLayoutSerializer.UpdateTechnicalNamesOfGridColumnns()`

Beispielhafte Implementierung:

```
/// <summary>
/// Migration to update control internal names within saved layouts
/// </summary>
[MigrationScript("7.0", 1)]
public class Migration_01_UpdateSavedLayouts : ConfigurationMigrationBase
{
    /// <summary>
    /// ctor
    /// </summary>
    public Migration_01_UpdateSavedLayouts() : base(EnumConfigurationType.GridConfiguration) { }

    /// <inheritdoc/>
    public override bool ExecuteTheScriptForConfiguration(MigrationExecutor executionContext)
    {
        // should be only executed when importing zips
        return false;
    }

    /// <inheritdoc/>
    public override bool ExecuteTheScriptAfterStart(MigrationExecutor executionContext)
    {
        if (ConfigurationManager.AppSettings.GetBool("BA:Configuration.DisableImplicitImport", false))
            return false;

        // it is wanted here to call *another* base method
        return base.ExecuteTheScriptForConfiguration(executionContext);
    }
}
```

```
}

/// <inheritdoc>
protected override void MigrateConfiguration(EnumConfigurationType type,
Guid id, ref string xml)
{
    GridViewConfiguration existingGridViewConfig = Api.Config.TryGrid(i
d);

    if (existingGridViewConfig == null || existingGridViewConfig.ProtectA
gainstUpdate)
        return;

    IEnumerable<KeyValuePair<string, string>> mappings = GetMappingsForCo
nfiguration(id);
    if (mappings == null)
        return;

    IReadOnlyList<GridViewLayoutConfiguration> layouts = Api.Config.GetGr
idLayouts(id);
    foreach (GridViewLayoutConfiguration layout in layouts)
    {
        string layoutDataNew = ClientLayoutSerializer.UpdateTechnicalName
sOfGridColumn(layout.LayoutData, mappings);
        layout.LayoutData = layoutDataNew;
        layout.Save();
    }
}

private IEnumerable<KeyValuePair<string, string>> GetMappingsForConfigura
tion(Guid correspondingConfiguration)
{
    List<KeyValuePair<string, string>> mappings = new List<KeyValuePair<s
tring, string>>();

    if (correspondingConfiguration == "12aeb0e6-f070-45ba-bfb2-96778fe812
b5".ToGuid()) // Contacts
    {
        // old id, new id
        mappings.Add(new KeyValuePair<string, string>("CalculatedColumn75
afd3dab52d4405a73760991ccb18d4", "parentIndustry559460599"));
    }
    // else if ... etc

    if (mappings.Any())
        return mappings;
    else
        return null;
}
```

```
    }  
}
```


17.4. 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	Executed
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.

18. 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](#)

Repräsentationsformen

String

Jede Criteria-Expression ist als String darstellbar. Diese Form ist für Benutzereingaben und Serialisierungen das Mittel der Wahl.

CriteriaOperator

Die Basisklasse `CriteriaOperator` ist eine technische Darstellung des Ausführungsbaumes.

Dialekte

Formeln in .NET

Diese Variation findet immer dann Anwendung, wenn ein Ausdruck in .NET ausgewertet und nicht an den XPO-Layer übergeben (IQueryable) wird.

Formeln in Reports fallen ebenfalls in diese Kategorie, weil die Report-Engine komplett im Speicher abläuft. Einzige Ausnahme: Filter auf Datenquellenebene. Diese werden von BA bisher aber noch nicht unterstützt.

.NET Formeln können auf alle Public Properties eines Objektes lesend zugreifen, auch jene, die nur programmiert sind und nicht nach SQL übersetzt werden können. Die Möglichkeit des Upcasting erlaubt auch auf Eigenschaften zuzugreifen, die zu konkreteren Datentabellen gehören (Siehe [DevExpress](#)). Beispiel: `<OrmPhoneCall>Phone`. Das castet erst auf `OrmPhoneCall` und nimmt dann das Property `Phone`.

Formeln in Datenbankabfragen (XPO)

Neben .NET können Formeln auch vom XPO-Layer nach SQL übersetzt werden. Das ist z.B. bei Filter-Expressions für Grids oder Kalender der Fall. Ebenso in berechneten Spalten derselben.

Restriktionen

- Es können nur XPO-kompatible Properties verwendet werden. Das sind neben den im Datenmodell definierten Spalten und Collections nur noch berechnete Properties mit einem PersistentAliasAttribut.
- Auch wenn es manchmal anders scheint, Formeln können niemals Mengen liefern, sondern immer nur einzelne skalare Werte. Jeder Zugriff auf ein XPO-Collection-Property oder einen Free-Join muss daher mit einer Aggregatfunktion abgeschlossen werden.
- Aggregatfunktionen können nicht in Bedingungen geschachtelt werden.

`[<OrmBABase>][[Oid]=[<OrmRelation>][[RelationType]={AF6FD532-FDE0-4732-964B-648B6E8B8415} && [Target]=[^^.Oid] && [IsPrimary]].Single([Source])].Single([EntityTitle])` funktioniert nicht. Statt dessen muss die Subexpression in das Aggregat geschrieben werden:

`[<OrmRelation>][[RelationType]={AF6FD532-FDE0-4732-964B-648B6E8B8415} && [Target]=[^.Oid] && [IsPrimary]].Single([<OrmBABase>][[Oid]=[^.Source]].Single([EntityTitle]))`

Hybrid Formeln .NET/XPO

Bei bestimmten Formeln findet ein Übergang zwischen XPO und .NET statt. Das bedeutet, dass ein Teil des Ausdrucks in .NET verarbeitet wird und ein anderer Teil in XPO.

Beispiele

```
[IsActive && [RelatedParents][IsActive].Exists()]
```

Diese Criteria, in .NET verwendet, führt das erste `IsActive` in .NET aus. Für den Zugriff auf `RelatedParents` wird allerdings eine DB-Zugriff benötigt. Dadurch wird das zweite `IsActive` und auch das `Exists`-Aggregat auf der Datenbank ausgeführt.

```
[<OrmSubAddressCompany>][[Parent] = ^.RelatedOwner.Oid && [SortOrder] = 0].Single([EntityTitle])
```

Auch das wird, in .NET ausgeführt, nahezu komplett auf der Datenbank ausgewertet – wie auch sonst bei einem [Free-Join](#). Einzig der Teilausdruck `^.RelatedOwner.Oid` wird in .NET berechnet, denn durch den Parent-Operator verlässt dieser den Geltungsbereich des Free-Join und muss entsprechend in .NET vorberechnet werden.

18.1. Ausführen in .Net

Für die Ausführung von Formeln in .NET sollte immer die Compilation verwendet werden. Die Verwendung von `Fit` und `Evaluate` ist zwar technisch möglich aber aus Gründen der Performance nicht zu empfehlen.

Kompilierung einer Formel für die Ausführung in .Net:

```
Func<OrmBABase, bool> filter = CriteriaOperatorHelper.CompileExpression<OrmBABase, bool>(dataSourceGuid, criteriaString);
```

Dies interpretiert den Criteria-String als .NET Criteria und compiliert und validiert alle Ausdrücke so, dass sie für Objekte vom Typ `dataSourceGuid` immer funktionieren und den korrekten Rückgabety, hier `bool` liefern.

Das Ergebnis der Compilation sollte immer gecacht werden, da selbige etwa 1000-fach langsamer ist, als die Ausführung.

Beschreibung

```
public static Func<ST, SR> CompileExpression<ST, SR>(Type ormType, CriteriaOperator criteria, Action<LambdaExpression> check = null)
```

Methode zur Kompilierung gegen eine Datentabelle mit einem gewünschten Rückgabety

- `ST` ORM Typ
- `SR` Rückgabety
- `ormType` Datentabelle, auf deren Basis kompiliert wird.
- `criteria`
- `converter` Optionaler Converter
- Rückgabe: Eine Exception oder eine ausführbare .Net Funktion.

Beispiel inkl. Kompilierung und Ausführung

```
try
{
    string criteria = "Iif(StartsWith([TextField], 'Inhalt'), [TextField], 'Kein Inhalt da')";
    Func<OrmBase, object> compiled = CriteriaOperatorHelper.CompileExpression<OrmBase, object>(typeof(OrmMyDataTable), criteria);
    string text = compiled(myData)?.ToString();
}
catch (Exception ex) when (!ex.IsThreadAbort())
{
    // Fehler Handling
}
```

}

18.2. In LINQ-Queries

Es stehen zwei Methoden zur Verfügung, um die Formeln auf die Datensätze anzuwenden. Dazu benötigt man die Objektrepräsentation einer Formel. Hat man einen String mit einer Formel, muss diese zuerst mit `CriteriaOperatorHelper.ParseDBCriteria` in die Objekte umgewandelt werden.

Ausführung einer Formel:

```
public object Evaluate(CriteriaOperator expression);
```

Beispiel:

```
IQueryable<OrmMyDataTable> query = Api ORM.GetQueryWithReadPermissions<OrmMyDa  
taTable>(session);  
string formulaStr = "Iif(StartsWith([TextField], 'Inhalt'), [TextField], 'Kei  
n Inhalt da')";  
CriteriaOperator formula = CriteriaOperatorHelper.ParseDBCriteria(formulaStr);  
query.Select(ff => new  
{  
    Name = ff.Name,  
    TextField = ff.Evaluate(formula),  
});
```

Die zweite Methode überprüft eine Bedingung und liefert `true` oder `false` zurück.

```
public bool Fit(CriteriaOperator condition);
```

Beispiel:

```
IQueryable<OrmMyDataTable> query = Api ORM.GetQueryWithReadPermissions<OrmMyDa  
taTable>(session);  
query = query.Where(ff => ff.Fit(CriteriaOperatorBuilder.ApplyActiveFilter(nul  
l, EnumEntryActiveState.Active)));
```



Beim Ausführen von Formeln muss beachtet werden, in welchem Kontext diese ausgeführt werden. Beispielsweise dürfen Relationsdefinitionen (z.B. `RelatedParents` oder `RelatedChildren`, nur in .Net genutzt 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.



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.

18.3. Formelfeld in Steuerelementen

Formeln werden in der Konfiguration als Strings erfasst. Damit diese erkannt werden können erhalten Sie das Attribut `[CriteriaProperty]`.

Als [Eingabeelemente](#) ist es sinnvoll ein Memofeld `[MemoControl]` zu nutzen. Für Bedingungen kann der Bedingungseditor `[FilterEditorPropertyControl]` genutzt werden. Insbesondere wegen [Free Joins](#) sollte man die Felder mit dem Attribut `[AllowHtml]` kennzeichnen.

Für die Syntaxüberprüfung für .Net Formeln stehen in der Klasse `CriteriaOperatorHelper` die Methoden `CompileExpression` zur Verfügung (Siehe [Kapitel](#)). Beim compilieren wird die Expression vollständig validiert, so dass es zur Laufzeit nahezu keine Fehler mehr geben kann. Bei einem Validierungsfehler kommt eine Exception.

Für die Validierung von Formeln, die in Datenbankabfragen genutzt werden sollen, steht `ValidateDBCriteria` zur Verfügung. Die Methode validiert nach bestem Wissen und Gewissen, indem sie eine minimale SQL-Query auf das Ergebnis absetzt. Die Validierung ist allerdings nicht in allen Fällen vollständig, da DevExpress bestimmte Prüfungen erst zur Laufzeit durchführt.

```
public static CriteriaOperator ValidateDBCriteria(Type ormType, Type targetType, 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(typeof(OrmMyDataTable), typeof(bool), "StartsWith([TextField], 'Inhalt')");
}
catch (Exception ex) when (!ex.IsThreadAbort())
{
    // Fehler Handling
}
```

18.4. Methoden zum Bauen einer Formel

```
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)
```

Zeitzone

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, string 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 GetEnumeratorSingleImage(string fieldName)
```

Bild des Wertes

- `fieldName` Der Auswahllistenfeldname im aktuellen Datensatz

```
public static CriteriaOperator GetEnumeratorSingle(string fieldName, CriteriaOperator 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 trivial und man sollte sich auf die Verwendung der BA eigenen Funktion `BAContainsEnumValue` beschränken.

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.

- `GetEnumeratorMultiDisplayable`
- `GetEnumeratorMultiColor`
- `GetEnumeratorMultiImage`

Die Basis Zugriffsmethode ist `GetEnumeratorMulti`, sie liefert den entsprechenden Free Join als `JoinOperand` zurück.

```
public static CriteriaOperator GetEnumeratorMulti(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
e, 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(EnumDataSourceExtension.MyDataTable).GetField(nameof(OrmMyDataTable.SubDatas));
OperandProperty fieldProperty = new OperandProperty(nameof(OrmSubDataTable.TextField));
JoinOperand subRecordSortOrder0 = CriteriaOperatorBuilder.GetSubRecords(subRecordField, 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 relationType, string relationCategory, GuidSet sourceTypes = null, bool onlyRelations = false, Aggregate aggregator = Aggregate.Single, CriteriaOperator aggregatedExpression = 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.Exist`
- `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, string relationCategory, GuidSet targetTypes = null, bool onlyRelations = false, Aggregate 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 nächsten Datensatz zu erhalten.

```
public static JoinOperand GetHierarchyPrimarySourceJoin(EnumRelationType relationType, string relationCategory, EnumPrimaryHierarchyRelationSourceSelector hierarchyRelationSourceSelector, int levelCount = 0, GuidSet sourceTypes = null, bool onlyRelations = false, Aggregate aggregator = Aggregate.Single, CriteriaOperator 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 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.

```
public static JoinOperand GetHierarchySourcesJoin(bool onlyPrimary, EnumRelationType relationType, string relationCategory, EnumHierarchyRelationSourceSelector hierarchyRelationSourceSelector, int levelCount = 0, GuidSet sourceTypes = null, bool onlyRelations = false, Aggregate aggregator = Aggregate.Single, CriteriaOperator 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 möglichst immer auf die gewünschten Typen eingeschränkt werden, falls globale Abfragen erstellt. Dazu dient diese Methode

```
public static CriteriaOperator ApplyObjectTypeFilter(CriteriaOperator criteria, Guid dataSourceGuid, CriteriaOperator selector = null)
```

Leserechte

Um die Leserechte zu berücksichtigen, muss dieser Filter gesetzt werden.

```
public static CriteriaOperator ApplyReadPermissions(CriteriaOperator criteria, Type queryOrmType, GuidSet dataSources = null, bool doNotAddOrmTypeCheck = false)
```

18.5. Eigene Formelfunktionen

DevExpress bietet [Möglichkeiten](#), um die Formeln (Criteria Operators) zu erweitern. Für verschiedene Möglichkeiten steht jeweils ein [Interface](#) zur Verfügung, wobei auch eine Kombination möglich ist und sinnvoll sein kann.

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.

Zu beachten

- Der Name der Klasse muss mit dem Namen der Implementierten Custom-Funktion übereinstimmen. Der Namespace ist egal.
- Wenn in einem Modul oder Projekt eine gleichnamige Klasse definiert, wird die Klasse des Core bzw. des darunter liegenden Moduls nicht mehr verwendet und nur noch die Überladung benutzt. Es kann dabei optional von der ursprünglichen Implementierung geerbt werden.
- Die Registrierung von eigenen Funktionen erfolgt in BA automatisch.

Makro Funktionen (BA intern)

Ziel unserer Erweiterung von Custom Criteria Operators Funktionen ist es Teile eines Criteria Operator Ausdrucks zu ersetzen. Also sozusagen eine Art Makro. Entsprechende Erweiterungen im Core befinden sich im `Packet BA.Core.CriteriaOperators.Functions`. Um eine Erweiterung zu implementieren, muss das Interface `ICriteriaOperatorBAFunction` implementiert werden.

Dieses Interface kann auch mit den folgenden Interfaces kombiniert werden, wenn nur ein Teil der Funktionalität über Standard Criteria implementiert ist.

Beispiel für `ICriteriaOperatorBAFunction`

Zuerst sollte man die Parameter prüfen, ob diese der erwarteten Anzahl und Typen entsprechen. Wenn nicht sollten Aussagekräftige Ausnahmen in Anwendersprache geworfen werden. Diese werden bei der Verwendung der Funktion in einem Formelfeld dem Konfigurator angezeigt.

Wenn alles passt wird am Ende die neue Formel zurückgegeben.

```
public class BAExample : ICriteriaOperatorBAFunction
{
    public virtual CriteriaOperator GetCriteria(CriteriaOperatorBAFunctionPatcher visitor, FunctionOperator function)
    {
        CriteriaOperatorCollection operands = function.Operands;
        if (operands == null || operands.Count != 1 || !(operands[1] is OperandValue operand1) || !(operand1.Value is string stringValue))
            throw new ArgumentException("Es wird ein String Parameter erwartet");
    }
}
```



```
        return new OperandValue(stringValue);  
    }  
}
```

In .Net Ausführbar (DevExpress)

Mit dem Interface [ICustomFunctionOperatorConvertibleToExpression](#) kann die eigene Criteria-Funktion in .NET ausgeführt werden. Klassen, die nur dieses Interface implementieren, funktionieren in Datenbankabfragen (Ansichten, etc.) nicht. Dazu muss zusätzlich auch `ICustomFunctionOperatorFormattable` implementiert werden.

Technisch gesehen müssen zwei Interfaces programmiert werden

- `ICustomFunctionOperator`
Dies wird für den Criteria-Interpreter von DevExpress verwendet, der Criteria-Ausdrücke zur Laufzeit interpretiert. Das wird z.B. im Kontext von Reports verwendet.
- `ICustomFunctionOperatorConvertibleToExpression`
Das ist für kompilierbare Criteria-Funktionen. Diese werden von BA überall verwendet außer in Ansichten und Reports. Aus den .NET Criteria wird dann zuerst ein LINQ-Expression-Tree erzeugt, welcher dann weiter in .NET IL und Maschinencode übersetzt wird.

Datenbankabfragen erweitern (DevExpress)

Das Interface [ICustomFunctionOperatorFormattable](#) ist ausschließlich für Datenbankabfragen und erlaubt es, individuell angepasstes SQL zu erzeugen. Die Implementierung muss dabei auf eventuelle Besonderheiten unterschiedlicher Datenbanken Rücksicht nehmen.

Klassen, die nur dieses Interface implementieren, funktionieren im .NET-Kontext nicht. Dazu muss zusätzlich auch `ICustomFunctionOperatorConvertibleToExpression` implementiert werden.

18.6. Aufruf einer Formel-Funktion

Um Funktionen per Programmierung zu definieren, muss ein `FunctionOperator` erstellt werden.

```
FunctionOperator dayOfYear = new FunctionOperator(FunctionOperatorType.GetDayOfYear, new OperandProperty("DateField"));
```

In diesem Fall wird der Tag des Jahres eines Datumwertes ermittelt. In dem Beispiel ist zu beachten, dass 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(nameof(BAToCurrentTimeZone), new OperandProperty("DateField"));
FunctionOperator dayOfYear = new FunctionOperator(FunctionOperatorType.GetDayOfYear, dateValue);
```

18.7. Verkettung von Formeln

Der mit Abstand häufigste Anwendungsfall ist die Verkettung zweier Formeln. Dabei wird die zweite Formel logisch auf das Ergebnis der Ersten angewendet. Aufgrund der Struktur der Criteria-Language ist das eine nichttriviale Operation.

Die Methode `CriteriaOperatorHelper.MergeCriteria` implementiert diesen Vorgang, indem sie einen neuen `CriteriaOperator` erstellt, der das Ergebnis der Verkettung darstellt.

Anwendungsfälle

- Wenn ein Control zwei Felder bereit stellt, eines für die Auswahl eines Datensatzes und ein zweites für die Auswahl des Anzeigetextes, dann muss letzterer Ausdruck auf das Ergebnis der ersten angewendet werden.
- Wenn man zum Rendern eines Wertes mehrere verschiedene Darstellungsformen oder Unterfelder des vom Benutzer abgegebenen Objektes benötigt, beispielsweise Text und Icon eines Enums oder Links-Adresse und Entity-Title eines ORMs. Auch hier, muss die vom Konfigurator vorgegebenen Expression nachträglich erweitert werden.

Validierung

Eine Validierung verketteter Operatoren ist im allgemeinen nur im Kontext möglich. Das bedeutet, um die zweite Expression zu validieren, muss man faktisch das Ergebnis der Verkettung validieren. Eine separate Validierung würde nur funktionieren, wenn man den Ergebnistyp des ersten Ausdrucks immer exakt kennt.

18.8. Nicht-Deterministische Funktionen

Wenn das Ergebnis einer eigenen Formel nicht deterministisch ist, also nicht ausschließlich von den Funktionsargumenten abhängt, weil es z.B. den aktuellen Benutzer oder dessen Sprache verwendet, müssen besondere Maßnahmen ergriffen werden.

- Falls nicht ausschließlich `ICustomFunctionOperatorConvertibleToExpression` verwendet wird, muss der Funktionsaufruf vom Typ `FunctionOperatorType.CustomNonDeterministic` sein. Dies kann über die Hilfsmethode `CriteriaOperatorHelper.MakeNonDeterministic` in der Implementierung von `GetCriteria` sichergestellt werden. `ICustomFunctionOperatorConvertibleToExpression` ist dadurch faktisch für solche Funktionen obligatorisch.
- Falls nicht-deterministische Operanden (Parameter) in dem gelieferten Criteria enthalten sind, müssen diese über die Klasse `ComputedOperandValue` gebunden werden. Diese stellt sicher, dass der Wert bei jeder Ausführung neu berechnet wird.

19. 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.

19.1. 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 Namespace „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.

19.2. 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 propertyName, SpinEditSettingsAttribute settings)
```

Boolean

Für die `Boolean` Eigenschaften wird eine `Checkbox CheckEditControl` generiert.

DateTime

Für die `DateTime` Eigenschaften wird ein `DateEditControl` generiert. Falls das Attribute `[TimeEdit 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 Mechanismen.

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.

19.3. 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 `Selectio`
`nControlAttributeBase` 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 propertyName, ControlDataProviderPropertiesBase properties)
{
    if (properties is CDPOrmFieldsProperties prop && propertyName == nameof(OrmFieldName))
    {
        ...
    }
}
```

Beispiele

Einzelauswahl einer Vorlage beliebigen Typs:

```
[ComboboxControl]
[CDPOrmDataProperties(dataSources: new string[] { EnumDataSource.TemplateBaseGuid })]
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.

19.3.1. Datenprovider Modifiers

Datenprovider benötigen in vielen Fällen Parameter, diese werden im Attribut hinterlegt. In eigenen Fällen 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 Paket `BA.Core.CustomAttributes` abgelegt und haben den Präfix `CDP`. Der wichtigste 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 in 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 configuration,
string propertyName, ControlDataProviderPropertiesBase properties)
{
    if (properties is CDPOrmFieldsProperties prop)
    {
        if (configuration != null && configuration != null && propertyName ==
nameof(OrmFieldName))
        {
            GridViewConfiguration gridConfig = (GridViewConfiguration)configuration;
            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)  
{  
    ...  
}
```

19.3.2. 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 angegeben 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(ControlDataProviderParameter parameter = null)
    {
        if (Properties != null && Properties is CDPEnumValuesProperties prop)
        {
            IEnumerable<ValueEnum> enumValues = AddWhere(Api.Enum.GetEnumValues(prop.MasterGuid));

            IEnumerable<ControlDataRow> values = enumValues.Select(CreateRow);

            values = AddFilter(values, parameter?.Filter);

            return values.AsQueryable();
        }

        if (Properties == null)
            throw new ArgumentNullException("Data provider: The properties are NULL");

        throw new ArgumentException(String.Format("Data provider: The properties are {0} and not {1}", Properties.GetType().FullName, typeof(CDPEnumValuesProperties).FullName));
    }

    public virtual ControlDataRow GetByKey(string key)
    {
        if (Guid.TryParse(key, out Guid guid) && Api.Enum.GetEnumValue(guid) is ValueEnum value)
```

```

        return CreateRow(value);
    else
        return null;
    }

    public virtual IEnumerable<ValueEnum> AddWhere(IEnumerable<ValueEnum> list)
    {
        if (Properties is CDPEnumValuesProperties prop)
        {
            return list.Where(ff => (prop.OnlyTheseValues == null || prop.OnlyTheseValues.Contains(ff.ValueGuid))
                && ((prop.WantedValues != null && prop.WantedValues.Contains(ff.ValueGuid))
                    || ((prop.IncludeNonReadable || ff.CurrentUserCanSelect())
                        && (prop.ActiveState == null || prop.ActiveState == EnumEntryActiveState.All || ff.IsActive == (prop.ActiveState == EnumEntryActiveState.Active)))));
        }

        return list;
    }

    protected virtual ControlDataRow CreateRow(ValueEnum value)
    {
        return new ControlDataRow
        {
            Key = value.GuidString,
            Title = value.Translation_Guid.ToString().Translate(),
            DataItem = value,
        };
    }

    public virtual IEnumerable<ControlDataRow> AddFilter(IEnumerable<ControlDataRow> list, string filter)
    {
        if (string.IsNullOrEmpty(filter))
            return list; // no filter

        string conditionFilterPart = filter.Replace("\"", "").ToLower();
        return list.Where(ff => ff.Title != null && ff.Title.ToLower().Contains(conditionFilterPart));
    }
}

```


19.4. 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), Attribute = 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 `AttributeEntrySingleValue` verwendet.

```
Attributes.Add(new AttributeEntrySingleValue() { PropertyName = = nameof(BasisProperty), Attribute = AttributeEnum.DisplayName, Value = "[INSERT TRANSLATION GUID]" });
```

20. Sonstiges

20.1. Grundlagen

20.1.1. 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 bündelt 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`.

20.1.2. 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.

```
namespace BA.Training
{
    public class DIInit : NinjectModule
    {
        public override void Load()
        { ... }
    }
}
```

20.1.3. Eigener Controller

Um einen eigenen Controller zu implementieren ist für BA nichts spezieller zu beachten. Lediglich bei der Absicherung vor unautorisiertem Zugriff sollte das Attribut `[BAAuthorize]` anstatt dem Standard `[Authorize]` verwendet werden.

20.1.4. 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, dass 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, dass 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",
        ShowSearchPanel = false,
        ShowFilterRowForColumns = false,
    };
    // Initialisieren
    engineGrid.Initialize();
    // Spalten hinzufügen (Hier eine Textspalte)
    engineGrid.FloatingColumns.Columns.Add(new TextBoxColumn()
    {
        OrmFieldName = nameof(OrmMyDataTable.EntityTitle),
        Caption = "Name"
    });
    // In den Konfigurationscache legen
    Api.Config.RefreshSingleConfigurationCache(engineGrid);
}
```

Häufig ist es sinnvoll die Konfigurationen zentral zu erstellen, damit sie beim Start der Maschine erstellt werden. Beispielsweise 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 : ICreateDynamicConfiguration
{
    public void CreateConfigurations()
    {
        ...
    }
}
```


20.1.5. 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 Rolle");
            uow.CommitChanges();
        }
    }
}
```

20.1.6. Zurücksetzen von Transaktionen

Wir haben 3 Funktionen zur Auswahl, mit der man UnitOfWork's bereinigen kann. Je nach Anwendungsfall ist die eine oder andere sinnvoller. Neben dem eigentlichen Zurücksetzen der Transaktionen gehen die einzelnen Methoden unterschiedlich mit den geladenen Datensätzen im Cache um.

- `RollbackTransaction()`: Datensätze bleiben erhalten.
- `DropIdentityMap()` Datensätze werden ungültig
Bevorzugt, falls man sicher ist, dass es keine Referenzen mehr zu den Datensätzen existieren.
- `UndoChanges()` (Erweiterungsmethode von GI in `SessionExtensions`) Datensätze werden aus DB neu geladen, bzw. neue Sätze werden verworfen.
Bevorzugt, falls man nicht sicher sein kann, dass es weitere Referenzen zu den Datensätzen existieren.

20.1.7. 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.");
    }
}
```

20.1.8. Umgang mit HTML-Strings

In der `Api.Html` sind hilfreiche Methoden für den Umgang mit HTML-Strings

HtmlToPlainText

Die Methode extrahiert aus einem HTML nur den Text. Das Verhalten ist ähnlich zu Exchange, wenn es eine Mail sendet und den Inhalt auch als Plain-Text erzeugt, falls der Empfänger kein HTML kann oder aktiviert hat.

Es werden unterstützt:

- Absätze jeglicher Art -> Zeilenumbruch
- ` ` -> Zeilenumbruch
- Aufzählungen -> `* ...`
- HTML Entities (`ö` etc.)

Skripte und CSS-Stile werden entfernt.

PlainTextToHtml

Wandelt einen nicht-HTML Text in HTML um. Dabei werden:

- Zeilenumbrüche zu ` ` ,
- aufeinanderfolgende Leerzeichen zu ` `, ` ` und
- Sonderzeichen zu HTML Entitäten. (Kein XSS möglich)

CleanHtml

HTML aufräumen, neu formatieren und nur den Body extrahieren. (für HTML-Spalten)

HtmlEquals

Zwei HTML Inhalte semantisch vergleichen. Die Funktion liefert im Gegensatz zum String-Vergleich auch dann true, wenn der HTML-Inhalt *gleich aussieht*, z.B.: falls sich die HTML Inhalte nur in der Formatierung oder den Header-Informationen unterscheiden. Das funktioniert auch bei augenscheinlich stark unterschiedlichen Fragmenten, aber eine 100% Garantie gibt es natürlich nicht. False negatives sind aber nicht zu erwarten.

Funktionen für Inline-Images

Alle diese Funktionen liefern den übergebenen String unverändert zurück, falls er nicht zu ihrem Verwendungszweck passt. Dadurch können mehrere Konverter nacheinander aufgerufen werden.

Beispiel:

```
html = Api.Html.VisitHtmlInlineTags(html,
    tag => Api.Html.VisitImgSrc(tag,
        src => Api.Html.InlineAttachmentToBase64Image(src)));
```

- `VisitHtmlInlineTags`
Durchsucht HTML Text nach Tags mit möglichem Inline-Content und ruft Visitor-Funktion auf, die diesen auch verändern kann.
- `VisitImgSrc`
Ruft für img-Tags Visitor für dessen src-Attribut auf, der dieses auch verändern kann.
Gedacht für die Benutzung in `VisitHtmlInlineTags`.
- `Base64ImageToInlineAttachment`
Konvertiert Inline-Image-URL mit Base64 Inhalt in `OrmAttachment`, das per Content-ID ("cid:...") über dessen OID referenziert wird.
Gedacht für die Benutzung in `VisitImgSrc`.
- `InlineAttachmentToBase64Image`
Konvertiert Inline-Image-URL mit Content-ID in ein Base64 Inline-Image, dass keine externen Daten benötigt.
Vorsicht, das kann riesige Datenmengen erzeugen!
Gedacht für die Benutzung in `VisitImgSrc`.
- `ConvertInlineAttachmentsToDownloadUrl`
Konvertiert alle Inline-Attachment-URLs zu Download-URLs auf die aktuelle Anwendungsinstanz, die im Browser als img src funktionieren.
Der Benutzer benötigt Leserechte, um die Download-URLs aufzulösen. Daher ist das weder für die langfristige Speicherung noch für die Weitergabe geeignet.
- `ConvertInlineAttachmentsToContentId`
Konvertiert alle Inline-Attachments-URLs zu Content-ID URLs. Diese funktionieren nur innerhalb von MIME-Nachrichten, und auch nur wenn die Dabei verwendeten Attachments auch in der MIME-Message hinzugefügt werden. Die Funktion liefert optional auch die erforderlichen Attachments.
- `ConvertContentIdToInlineAttachment`
Konvertiert alle Content-ID URLs zu internen Inline-Attachment URLs. Dies ist erforderlich, wenn HTML mit Inline Content von externen MIME Quellen importiert wird.

20.1.9. 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, dass 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(() => { ... });
```

20.1.10. Translatable Exception

Um übersetzbare Meldungen für Anwendungsprotokolle und/oder für Benutzer im Backend-Code (z.B. Hintergrundprozesse oder ORM-Events) zu erzeugen gibt es die Klassen `TranslatableException` und `TranslatableMessage`. Beiden gemein ist, dass die Übersetzung nachgelagert erfolgt. Dieselbe Meldung kann dadurch in verschiedenen Sprachen an verschiedene Kanäle ausgegeben werden. Z.B. bei einer Massenoperation in der Benutzersprache an den Anwender, in der Sprache für Anwendungsprotokolle in selbige und ggf. in Englisch in das Server-Log.



Es ist darauf zu achten, dass diesem Klassen niemals bereits übersetzte Texte übergeben werden, auch nicht bei den Parametern eines Meldungstextes. Auch Datumswerte und Zahlen sollten nicht vorher in Strings konvertiert werden, um die sprachspezifischen Formatierungen zu ermöglichen.

TranslatableException

Wird eine `TranslatableException` geworfen, wird diese im allgemeinen dem Benutzer als Fehler angezeigt oder bei Hintergrundprozessen ins Protokoll geschrieben. BA geht davon aus, dass es sich um eine für den Benutzer gedachte Meldung handelt, die nicht weiter angereichert oder verändert werden soll.

Andere Exceptions werden von BA als unerwartete Ausnahmen behandelt, und ggf. noch mit technischen Kontextinformationen, wie z.B. dem ORM-Event, in dem sie aufgetreten sind, angereichert. Typischerweise ergibt sich daraus kein besonders Endkundentauglicher Fehlertext.

Optional kann einer `TranslatableException` auch ein **Schweregrad** mitgegeben werden. Diese beeinflusst das Log-Level bzw. die Farbe des Toasters. Das hat aber keinen Einfluss auf die Ablaufsteuerung; es bleibt wie jede andere Exception ein Abbruch der Transaktion.

TranslatableMessage

Das ist im Prinzip dasselbe, aber eben ohne Exception. Diese Klasse eignet sich vor allem im Kontext von Anwendungsprotokollen in Hintergrundprozessen. Sie kann direkt zu einem Anwendungsprotokoll hinzugefügt werden: `Logger.AddEvent(translatableMessage)`

Die Angabe des Schweregrades ist hier Pflicht. Zum einfacheren Erstellen gibt es statische Fabrikmethoden, z.B.: `TranslatableMessage.Info(...)`

Man kann auch mehrere Meldungen mit **`TranslatableCompoundMessage`** zu einem Meldungstext zusammenfassen. Dadurch ist es möglich, mehrere, Meldungstexte mit individuellen Parametern zusammenzufassen, ohne auf die Nachtägliche Übersetzbarkeit verzichten zu müssen.

Beispiel

```
var msg = new TranslatableCompoundMessage { Delimiter = "\n" };  
foreach ((EnumStatus status, int anzahl, DateTime von DateTime bis) in verarbe
```

```
itetNachStatus)
    if (anzahl != 0)
        // "{0:N0} Datensätze im Status '{ 1 }' im Zeitraum {2:D} bis {3:D} ve
rarbeitet."
        msg.Add(TranslatableMessage.Success("49FB98BF-A18D-4F44-BD7D-BDC14DD6F
62A", anzahl, status, von, bis));
    if (failed != 0)
        // "{0:N0} Datensätze fehlgeschlagen."
        msg.Add(TranslatableMessage.Warning("43B4AB04-3E0E-44EE-AFBB-296A6BF78FA
5", failed));
    Logger.AddEvent(msg);
```

Dabei könnte z.B. ein solcher Text heraus kommen:

```
2.738 Datensätze im Status 'versendet' im Zeitraum Montag, 15. Juni 2009 bis M
ittwoch, 17. Juni 2009 verarbeitet.
24 Datensätze im Status 'in Arbeit' im Zeitraum Mittwoch, 17. Juni 2009 bis Do
nnerstag, 18. Juni 2009 verarbeitet.
2 Datensätze fehlgeschlagen.
```

bzw.

```
Processed 2,738 records in state 'sent' in the range Monday, June 15, 2009 to
Wednesday, June 17, 2009.
Processed 24 records in state 'in progress' in the range Wednesday, June 17, 2
009 to Thursday, June 18, 2009.
2 records failed.
```

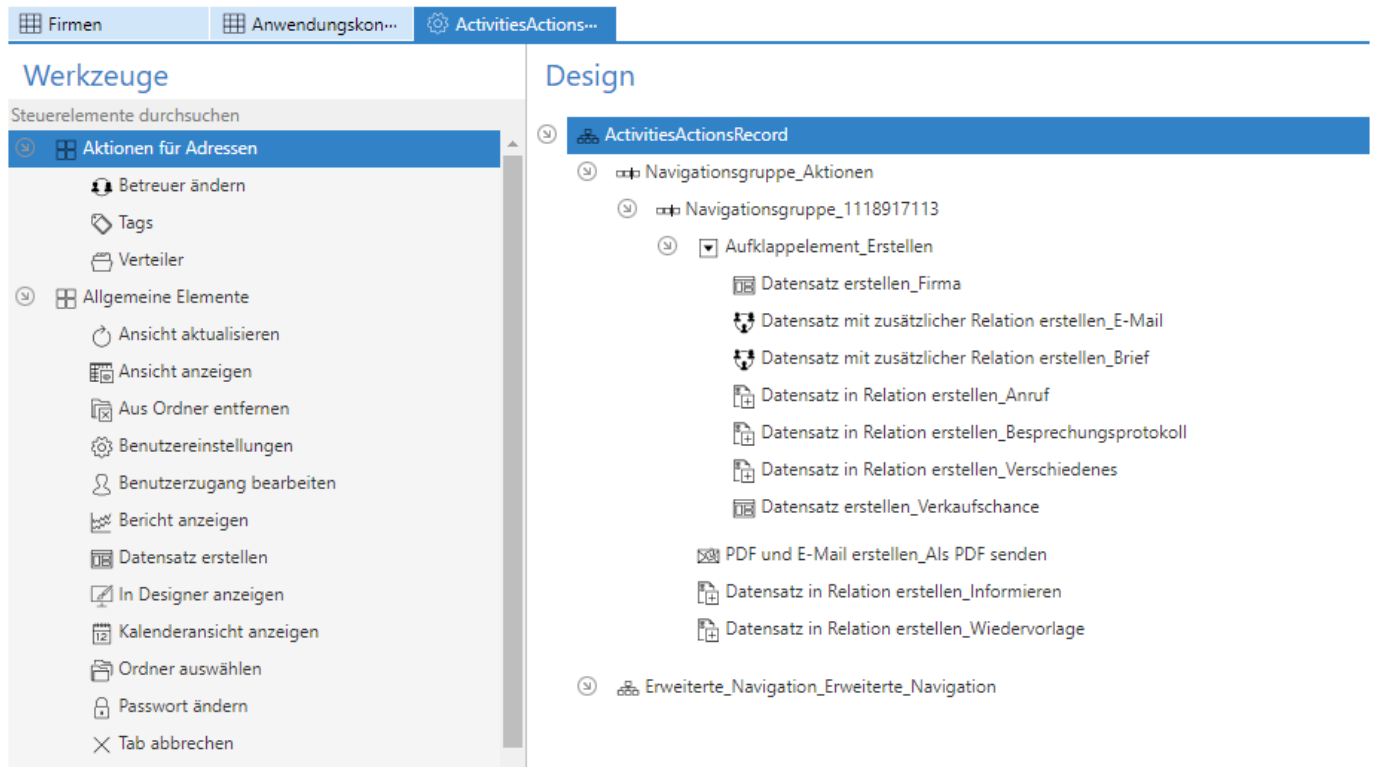
Die Nachgelagerte Übersetzung berücksichtigt jetzt, dass die die Parameter mehrfach im Gesamttext vorkommen und immer an die korrekten Werte gebunden werden. Das wäre anders nicht möglich.

Als Bonus-Feature hat die zusammengesetzte Meldung jetzt den Schweregrad “Warnung” (gelb), weil mindestens eine solche dabei ist. Ohne die letzte Zeile wäre es “Success” (grün).

20.1.11. Designer Drag & Drop Regeln

Drag & Drop Regeln definieren, welche Steuerelemente sich im Designer wohin ziehen lassen.

Am Beispiel der Ribbon bar Navigation sieht man in der Toolbox Steuerelemente, die sich in den Design-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 configurationType, string configurationSubType = null, DndBehavior dndBehavior = DndBehavior.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 den Datentyp des gedraggten Controls.
- `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.
 - `Allow` (default) erstellt eine Erlauben-Regel.
 - `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 bekommen und auch diese können ggf. auch weitere Controls bekommen, sofern es passende Erlauben-Regeln gibt.
 - `DenyCascade` lässt auch unter dem verbotenen Element keine weiteren Children zu, und zwar rekursiv für alle eventuellen weiteren Children.
Das Hinzufügen von Controls ist dann also für den kompletten Teilbaum ab dem Source-Control verboten, unabhängig von der Art der Children.
 - `Default` Diese Regel *entfernt* eine Regel. Das ist nur beim überschreiben von Core-Regeln in Projekten sinnvoll. Das Regelwerk wird dann so behandelt, als ob es diese Regel niemals gegeben hätte. Falls für das betreffende Control noch allgemeinere Regeln existieren, gelten nun diese.
- `MinLevel` und `MaxLevel` erlauben den `SourceType` nur auf bestimmten Ebenen (Zählweise siehe vorheriger Abschnitt)

Generell gilt:

- Grundsätzlich ist alles erst einmal verboten, solange es keine Erlauben-Regel (`DndBehavior.Allow`) gibt.
- Die Kombination `SourceType` und `TargetType` ist pro `configurationType` / `configurationSubType` Kombination eindeutig. Wenn es mehrere Regeln dazu gibt, wird die zuletzt gelesene andere überschreiben. Es ist garantiert, dass Projekte gegen BA gewinnen.
- Sowohl `SourceType` als auch `TargetType` können entweder direkte Datentypen des Controls, einer

ihrer Basisdatentypen oder auch ein Interface sein.

Regeln für konkretere Datentypen gewinnen gegenüber Regeln für Basisklassen, und die gewinnen gegen Regeln für Interfaces.

Man kann also z.B. auf eine Basisklasse oder ein Interface eine Erlauben-Regel anlegen und dann einen konkreteren Typen wieder verbieten und sogar dann für einen exakten Typ wieder erlauben.

- Wenn eine Regel für `SourceType` spezieller und für `TargetType` allgemeiner ist als eine andere Regel, gewinnt erstere.

MinLevel und MaxLevel

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.

Beispiel

Die eigentliche Definition sieht dann z.B. so aus, für Navigation

1. 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)]
```
2. `NavigationGroupControl` kann auch wieder ein `NavigationGroupControl` beinhalten (Ebenenbeschränkung von 2) bleibt beibehalten


```
[assembly: DnDRule(typeof(NavigationGroupControl), typeof(NavigationGroupControl), EnumConfigurationType.NavigationConfigurationGuid, minLevel: -1, maxLevel: 2)]
```

```
trol), EnumConfigurationType.NavigationConfigurationGuid)]
```

3. **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)]
```

4. **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(ExtendedNavigationGroupControl), EnumConfigurationType.NavigationConfigurationGuid, null, DndBehavior.DenyCascade)]
```

5. **Aufklappelement darf Kind von der Navigationsgruppe sein ab Ebene 4**

```
[assembly: DnDRule(typeof(NavigationGroupControl), typeof(DropDownAction), EnumConfigurationType.NavigationConfigurationGuid, minLevel: 4)]
```

6. **Das Aufklappelement darf die gleichen Elemente beinhalten, wie in 3) genannt**

```
[assembly: DnDRule(typeof(DropDownAction), typeof(ServerActionBase), EnumConfigurationType.NavigationConfigurationGuid)]
```

```
[assembly: DnDRule(typeof(DropDownAction), typeof(ClientActionBase), EnumConfigurationType.NavigationConfigurationGuid)]
```

```
[assembly: DnDRule(typeof(DropDownAction), typeof(ToggleButtonBase), EnumConfigurationType.NavigationConfigurationGuid)]
```

Bestehende Regeln 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 oder eine Regel komplett entfernen.

Es könnte aber auch vorkommen, dass ein Projekt die Regeln umfangreicher ändern will, ohne diese überschreiben zu müssen, also ggf. Regeln komplett umbauen oder eine eigene Prioritätsreihenfolge implementieren.

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, List<DnDRuleAttribute> dragAndDropRules);
```

Hier kann man die übergebene Liste frei anpassen. Dies sollte nur in Ausnahmefälle getan werden. In der Regel sollten die Attribute verwendet werden.

Das übergebene Liste enthält alle Regeln, die ohne dieses Interface aktiv würden in der Reihenfolge ihrer Priorisierung, also inklusive derer, die im Projekt dazu gekommen sind und ohne die, die

überschrieben wurden.

Fehlersuche

<http://.../bacrm/bamaintain/CreateRules>

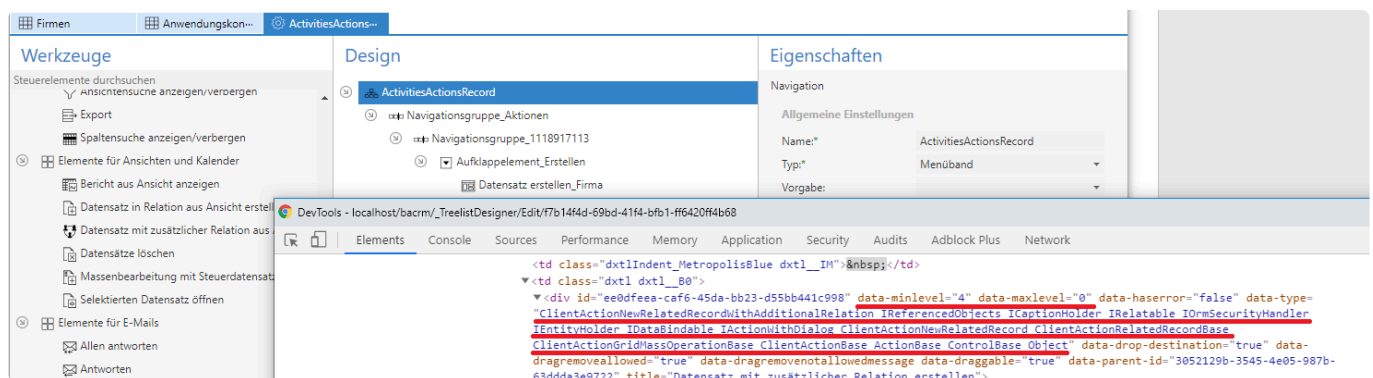
Über den Aufruf kann man sich ausgeben lassen, mit welchen Regeln das System arbeitet. Diese Regeln wurden dazu bereits in JSON Format übersetzt, damit sie der Client verstehen kann. Es kann z.B. gut sein, dass man einfach eine Regel definiert hat, die nicht erreichbar ist, weil Zwischenknoten fehlen.

Es empfiehlt sich, den Code über ein Notepad++ 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 Interfaces ein Control benutzt, sowie welche Werte als erlaubte Verschachtelungstiefen errechnet wurden.



Client debuggen

Hat alles nichts geholfen, kann man versuchen, im Browser beim Drag and Drop zu debuggen. Dazu im F12 in der Function `InitDAD` in die anonyme Function für `accepts` einen Breakpoint setzen.

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.

```
// ##### Drag & Drop #####
DesignerManagement.InitDAD = function () {
    DesignerManagement._Json = JSON.parse($("#DragAndDropRules").val());
    // See also fix in css file: .dndLeft .gu-transit { display: none; }
    DesignerManagement.drake = dragula([], {
        revertOnSpill: false,
        removeOnSpill: false,
        moves: function (el, container, handle) {
            if (DesignerManagement.clickNotAllowed)
                return false;
            var element = $(el).find(DesignerManagement._NodeAccessor);
            if (container.className == "dndRight" && (DesignerManagement._LastClick
                if (handle == null || handle.tagName == null || handle.tagName.toLowerCase() == "input" || handle.tagName.toLowerCase() == "button" || handle.tagName.toLowerCase() == "a")
                    handle = $(el).find(DesignerManagement._NodeAccessorForHandle)
                if (DesignerManagement.clickNotAllowed)
                    return false;
                DesignerManagement._LastClickedNodeId = element.attr("id");
                //DesignerManagement.Processing(true);
                if (handle)
                    handle.click();
                else {
                    //DesignerManagement.Processing(false);
                    return false;
                }
                DesignerManagement._LastFocusedNodeId = window.DesignTreeList.GetFocusedNode();
                return false;
            }
            else if (container.className == "dndLeft")
                handle.click();
            if (element.attr("data-draggable") != "true")
                return false;
            if (container.className == "dndLeft") {
                DesignerManagement.drake.options.copy = true;
                DesignerManagement.drake.options.revertOnSpill = true;
            }
            else if (container.className == "dndRight") {
                DesignerManagement.drake.options.copy = false;
                DesignerManagement.drake.options.revertOnSpill = false;
            }
            return true; // elements are always draggable by default
        }
    },
    accepts: function (el, target, source, sibling) {
        if (!DesignerManagement._DraggedElement) {
            DesignerManagement._DraggedElement = el.id;
            var element = $(el).find(DesignerManagement._NodeAccessor);
            if (source.className == "dndRight") {
                DesignerManagement.HideAllChildNodes(element.attr("id")); // +
                var ce = $(el).find("[class*='dxTreeList_ExpandedButton_MetropolisBlue']");
                ce.removeClass("dxTreeList_ExpandedButton_MetropolisBlue");
                ce.addClass("dxTreeList_collapsedButton_MetropolisBlue");
            }
        }
    }
    );
}
```

20.1.12. Das ControlFilter Attribut

Das `ControlFilter` Attribut dient dazu Steuerelemente in der Toolbox wahlweise ein oder auszublenden. Insbesondere wird dies benötigt, um Steuerelemente in der Navigationskonfiguration nur bei bestimmten Typen der Navigationskonfiguration anzuzeigen.

Dazu hat das Attribut zwei Möglichkeiten zu definieren, ob das Steuerelement in der Toolbox sichtbar sein soll oder nicht. Die Optionen sind `OptOut` und `OptIn`. Wenn das Attribut eine `OptOut`-Option verwendet und die Bedingung erfüllt ist, wird das Steuerelement nicht in der Toolbox angezeigt. Wenn das Attribut die Option `OptIn` verwendet und die Bedingung erfüllt ist, wird das Steuerelement angezeigt. `OptOut` hat Vorrang vor `OptIn`, d. h. wenn das Steuerelement aufgrund von `OptOut` nicht sichtbar ist, wird `OptIn` überhaupt nicht beachtet.

```
[Serializable]
[Toolbox(EnumConfigurationType.NavigationConfigurationGuid)]
[ControlFilter("NavigationConfigurationType", ExpressionType.Equal, EnumNavigationConfigurationType.BusinessMailNavigationGuid, EnumControlFilterApplyState.OptOut)]
[ControlFilter("NavigationConfigurationType", ExpressionType.Equal, EnumNavigationConfigurationType.OrmActionsNavigationGuid, EnumControlFilterApplyState.OptOut)]
public class ClientActionUserSettings : ClientActionBase
```

Dieses Steuerelement ist in der Toolbox verfügbar, wenn der Konfigurationstyp nicht `BusinessMail` und der Typ nicht `OrmActions` ist.

```
[Serializable]
[Toolbox(EnumConfigurationType.NavigationConfigurationGuid)]
[ControlFilter(nameof(NavigationConfiguration.NavigationConfigurationType), ExpressionType.Equal, EnumNavigationConfigurationType.RibbonNavigationGuid, EnumControlFilterApplyState.OptIn)]
[ControlFilter(nameof(NavigationConfiguration.NavigationConfigurationType), ExpressionType.Equal, EnumNavigationConfigurationType.MainNavigationGuid, EnumControlFilterApplyState.OptIn)]
[ControlFilter(nameof(NavigationConfiguration.NavigationConfigurationType), ExpressionType.Equal, EnumNavigationConfigurationType.MenuNavigationGuid, EnumControlFilterApplyState.OptIn)]
public class ServerActionShowDashboard : ServerActionBase
```

Dieses Steuerelement ist in der Toolbox verfügbar, wenn der Konfigurationstyp der Navigation `Ribbon`, `Main` oder `Menu` ist.

20.2. UI

20.2.1. Eigene Themes ausliefern

Wenn man Themes direkt im Projekt mitausliefern will, kann man eine css-Datei im Ordner Content ablegen. Diese Datei muss Embedded Ressource sein und mit "Theme" beginnen. Außerdem darf sie nicht einfach "Theme.css" heißen, sonst kann sie nicht zur Auswahl angezeigt werden. Mögliche Dateinamen wären "ThemeBlau.css", "Theme_blau.css", "Theme-blau.css", etc.

Wenn eine css Datei diese Kriterien erfüllt, wird sie in den Anwendungseinstellungen als mögliches Theme zur Auswahl angezeigt.

20.2.2. 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.

SVG Icons einbinden

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.Size16x16);
```

Icon Farbe bestimmen

Soll ein Icon eine bestimmte Farbe haben muss es eine css Klasse erhalten und dann ein entsprechender Filter auf genau dieses Element gelegt werden.

CSS Beispiel, wenn man die Standard Highlight Farbe setzen möchte

```
.icon-mit-filter-in-highlight-farbe {  
    filter: var(--icon-color-highlight);  
    -webkit-filter: var(--icon-color-highlight);  
    -moz-filter: var(--icon-color-highlight);  
}
```

CSS Beispiel, für den Farbwert #00ff00 (Grün)

```
.icon-mit-filter-in-highlight-farbe {  
    filter: invert(66%) sepia(32%) saturate(7077%) hue-rotate(84deg) brightness(111%) contrast(133%);  
    -webkit-filter: invert(66%) sepia(32%) saturate(7077%) hue-rotate(84deg) brightness(111%) contrast(133%);  
    -moz-filter: invert(66%) sepia(32%) saturate(7077%) hue-rotate(84deg) brightness(111%) contrast(133%);  
}
```

Filter kann man berechnen lassen bei z.B. [Codepen](#) oder <https://angel-rs.github.io/css-color-filter-generator/> (Vorsicht: Sie rechnen einmal und geben dann an, wie genau gerechnet wurde. Wenn die Farbabweichung zu hoch ist, nochmal rechnen lassen, bis "This is a perfect result" ausgegeben wird und/oder die Farbabweichung so gering ist, wie man sie gerne hätte)

Eigene SVG Icons

Möchte man eigene SVG Icons einbinden, legt man eine "icons.zip" Datei im "Content" Ordner an. Diese muss als Eingebettete Resource eingebunden werden. In dieser Datei legt man seine SVG Icons ab. Da die Referenzierung über den Namen erfolgt, sollte man ein entsprechendes Präfix für die eigenen Icons verwenden.

Eigene Icons

Bindet man eigene Icons ein, ohne das BA interne System der SVG Icons und der Farbvergabe zu nutzen, muss verhindert werden, dass die BA Filter auf diese Icons angewendet werden. Der Filter würde sich sonst über ein nicht-schwarzes Icon legen und falsche Farben produzieren. Dazu muss man ein CSS Style setzen.

```
.icon-ohne-filter {  
    filter: unset;  
    -webkit-filter: unset;  
    -moz-filter: unset;  
}
```

20.2.3. 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".

20.2.4. Eigene Styles einbinden

Um eine eigene CSS Datei in die Anwendung, wird diese in den “Content” Ordner abgelegt und in den Eigenschaften wird bei “Build Vorgang” “Embedded Resource” eingestellt. Wie bei den [JavaScript](#) Dateien.

Nun wird eine Klasse `BundleConfig` im Ordner “App_Start” erstellt, falls diese noch nicht vorhanden ist. Die Klasse muss `BABundleConfigBase` erweitern und im Standard-Namespace des Projektes liegen. Das Bundle in BA ist “~/bundles/styles”, dort wird die CSS Datei hinzugefügt.

```
public void RegisterBundles(BundleCollection bundles)
{
    public override void RegisterBundles()
    {
        StyleBundle("~/bundles/styles").Include("~/Content/MyModule.css");
    }
}
```

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



Nutzen Sie entweder einen eignen Ordner oder einen



Achten Sie auf die Groß- und Kleinschreibung der Ordner.

20.3. Funktionalitäten

20.3.1. 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 versendenden 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 fieldNameBC, 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 existieren 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 die Größen- 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");
```

20.3.2. Auswahl von E-Mail-Adressen anpassen

Aus technischen Gründen, kann für die Auswahl von E-Mail-Adressen keine konfigurierte Ansicht verwendet werden. Daher kann man in Modulen die Ansicht entsprechend anpassen. Dafür wird das Interface `IEmailAddressSelectionControlModifier` implementiert und über [Dependency Injection](#) im System bekannt gemacht.

Es kann auch die Basisimplementierung `EmailAddressSelectionControlModifierBase` erweitert und entsprechend angepasst werden. Dies erspart die Implementierung aller Methoden.

Folgendes ist zur Zeit möglich:

- Definition der Breite des Dialoges
- Ein-/Ausblenden der Icon Spalte
- Ein-/Ausblenden der Entity-Title Spalte
- Ein-/Ausblenden der Anzeigenamen Spalte
- Definition der Formel zur Berechnung des Anzeigenamens
- Definition bis zu drei weiteren Textspalten (Spaltentitel und Formel)



Die E-Mail-Typ Spalte ist als erste weitere Textspalte definiert

20.3.3. 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

20.3.4. 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.



Prinzipiell funktioniert die Dublettenprüfung nur in Masken, die in einem Tab dargestellt werden. Innerhalb von Dialogen funktioniert sie nicht

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 : OrmCRMCompany
```

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]` & `[DuplicateSearchSubRecordFieldDefinition]`) und ob und wenn welche bestehenden Relationen bei Änderungen geprüft werden sollen (`[DuplicateSearchRelationDefinition]`).

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 `DuplicateSearchFieldDefinition` 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 gar nicht 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, Session session)
{
    if (mainRecord is OrmContact contact && contact.GetFirstFoundParentOrm<OrmCRMCompany>() is OrmCRMCompany company)
        return Api.ORM.GetQueryWithReadPermissions<T>(session).Where(cont => company.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, DuplicateChangeInfo[] changeInfos)
{
    if (changeInfos != null && changeInfos.FirstOrDefault(ff => ff.ElementName == "Name")?.Value is string name)
        return name;
    return 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(input);
}
```

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, "Name", true)]
[DuplicateSearchFieldDefinition(EnumDataSource.AddressGuid, "Address")]
[DuplicateSearchFieldDefinition(EnumDataSource.AddressGuid, "PostalCode",
skipPhoneticTranslation: true)]
[DuplicateSearchRelationDefinition(EnumDataSourceExtension.CompanyGuid, EnumDataSource.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.

20.3.5. 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 Kontakte")]
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 harret 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-c1ba-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", "06fa14a7-7297-4e5c-9b52-25642a2b4f95", "ed5bcfee-74f0-4746-a021-7cf1fb7e484b" };  
}
```

`GetFieldMappingDialogDescriptions` liefert einfach nur ein Array von Texten zurück, welche im Spaltenzuordnungsdialo 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.ContactGuid);  
    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 importFile)
{
    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 „Familiennamen“ -> „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<Dictionary<string, string>>(parameters.FieldMappingJson);
    string importPath = Path.Combine(ImportJobConfiguration.GetJobDirectory(parameters.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<OrmCompany>();
    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(GetTargetType(), 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.Equals("migrationid", StringComparison.OrdinalIgnoreCase))
        continue;
    string val = csv.GetSourceField(title);
    if (mappings.TryGetValue(title, out string csvTitle) && csvTitle.StartsWith($"{Import.Constants.ParentLinkIdentifier}.")
    {
        if (parentMapping == null)
        {
            string parentProperty = csvTitle.Substring(csvTitle.IndexOf('.') + 1);
            parentMapping = parents.ToLookup(ff => (string)ff.GetMemberValue(parentProperty), ff => ff.Oid.ToString());
        }
        if (parentMapping.Contains(val))
            csv.SetTargetField("$REL_Parent", parentMapping[val].FirstOrDefault() ?? "");
        else
```

```

        csv.SetTargetField("$REL_Parent", "");
    }
    else
        csv.SetTargetField(title, val);
}

```

Pro gelesenen 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("EmailAddresses");
    if (record != null && record.CurrentRecordIsDirty && record.Record is IDictionary<string, object> dyn1 && (!dyn1.ContainsKey("EmailAddressType") || string.IsNullOrEmpty(dyn1["EmailAddressType"] as string)))
        record.SetField("EmailAddressType", "Primary");
    record = csv.GetCsvRecordBySubTable("Addresses");
    if (record != null && record.CurrentRecordIsDirty && record.Record is IDictionary<string, object> dyn2 && (!dyn2.ContainsKey("AddressType") || string.IsNullOrEmpty(dyn2["AddressType"] as string)))
        record.SetField("AddressType", "Main address");
    if (!string.IsNullOrEmpty(parameters.ImportTag))
    {
        record = csv.GetMainCsvRecord();
        if (record != null && record.CurrentRecordIsDirty && record.Record is IDictionary<string, object> dyn3)
        {
            string addressTags = (dyn3.ContainsKey("AddressTags") ? dyn3["AddressTags"] as string : null) ?? "";
            if (!string.IsNullOrEmpty(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.

20.3.6. Dateiparser für die Suche

Damit die Suche Texte in Dateien finden kann, müssen aus diesen der textuelle Teil herausgefiltert werden. Falls notwendige Dateitypen durch den Standard nicht abgedeckt werden, können eigene Parser implementiert werden.

Vorgehensweise

Um einen eigenen Dateiparser zu entwickeln muss eine neue Klasse erstellt werden, die das Interface `IAttachmentParser` implementiert.

Damit ist folgende Methodensignatur vorgegeben `public void Parse(AttachmentInfo attachment);`

Weiterhin ist es erforderlich, dass die Klasse das Attribut `AttachmentParserAttribute` verwendet. Dieses bekommt als Parameter einen eindeutigen Namen, sowie eine Liste mit Dateierweiterungen, für die es anspricht. Diese Liste ist ein String, der die Dateitypen kommasepariert enthält. Als eindeutiger Name wird Namespace + Klassenname des Parsers empfohlen, andere Namen sind zulässig.

Wichtig: Parser dürfen für Attachments aus BusinessApp nur die Property `TextContent` verändern und ggf. `InnerAttachments` befüllen. Die anderen Properties werden zum Erkennen von Änderungen am ORM benutzt.

- `TextContent` sollte die Wörter, die über die Suche einzeln gefunden werden sollen, per Leerzeichen getrennt enthalten. Groß- und Kleinschreibung spielt dagegen keine Rolle.
- `InnerAttachments` wird benötigt, falls ein Parser für z.B. ZIP-Dateien oder andere Archivformate entwickelt wird. Für `InnerAttachments` werden wieder Objekte vom Typ `AttachmentInfo` angelegt. In diesem Fall dürfen auch andere Properties, wie z.B. `FileName` befüllt werden.

Auf Basis der [Assembly Typen](#) werden die Parser vorgezogen, falls für eine Dateierweiterung mehrere Parser existieren. Darüber hinaus kann diese Vorgehensweise durch die `Customer.config` überschrieben werden. (Siehe Doku der `Customer.config`).

Beispiel für die Implementierung

Soll ein eigener Excel-Parser implementiert werden, bietet sich z.B. folgender Code an:

```
[AttachmentParser("BA.ProjectName.AttachmentParser.ExcelParser", ".xlsx,.xls\nm")]
public class ExcelParser : IAttachmentParser
{
    public void Parse(AttachmentInfo attachment)
    {
        using (MemoryStream stream = new MemoryStream(attachment.Data))
        {
```

```
        string parseResult = GetTextFromExcel(stream); // business logic for parsing the file
        attachment.TextContent = parseResult;
    }
}
```

Falls die Datei, die geparsed werden soll, ein Container / Archiv ist, also selbst Dateien enthält, muss für die inneren Dateien jeweils wieder ein Parser aufgerufen werden.

Dies geht mit

```
AttachmentParserLocator.GetParser(string fileExtension).Parse(AttachmentInfo attachment)
```

20.4. Konfiguration und Auslieferung

20.4.1. 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.DefaultCountry;
EnumCountries countryUser = Api.Config.CurrentUserConfiguration().Basic.DefaultCountry;
```

Weitere Einstellungen befinden sich in Unter-Konfigurationen. Diese können mit `GetSubConfiguration` ausgelesen werden.

```
ApplicationSubConfigurationCorrespondence correspondenceConfiguration = Api.Config.ApplicationConfiguration().GetSubConfiguration<ApplicationSubConfigurationCorrespondence>();
```

Einstellungen erweitern

Es ist möglich die Einstellungen für die Anwendung und für die Benutzer, 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 `[ApplicationConfigurationSubConfiguration]` 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; }
}
```

20.4.2. 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, 1000)]
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, dass 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.
- `IEntityConfigurationSpecifier` 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)]
```

20.4.3. 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.

20.4.4. Lizenzdialog erweitern

Zum Hinzufügen von neuen Einträgen für den Lizenzdialog ist folgendes erforderlich:

Anlegen einer Datei mit dem Namen: `Licenses.csv` als eingebettete Ressource unter `/Content/ExternalLicenses`

Diese Datei enthält keine Überschriften, sondern nur Inhalt wie folgt:

```
GUID;Autor;Produktname;LizenzName
```

also z.B:

```
94318be4-5a0e-4458-ac40-30ecacc172fd;Microsoft;System.Security.Principal.Windows;MIT License
```

Die *GUID* ist frei wählbar, muss aber eindeutig sein.

Autor und *Produktname* erscheinen wie angegeben im Gridview des Lizenzdialogs. Der Lizenzname ist der Schlüssel für die Lizenz, welche ebenfalls im selben Ordner als eingebettete Ressource vorliegen muss *.txt* also z.B. *MIT License.txt*

Lizenzen, deren Name bereits im Core / CRM genannt werden, werden automatisch berücksichtigt und sollten nicht erneut hinzugefügt werden, um keine Lizenzdatei aus dem Core zu überschreiben.

20.4.5. 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.

20.4.6. Rollen des ersten Benutzers

Wenn Business App neu installiert wird, wird ein erster Anwender benötigt. Dieser wird über “https://servername.tld/XXX/Account/createdefaultuser” erzeugt. Dabei erhält dieser Anwender die “Administrator” Rolle und bei Installation von BA.CRM bestimmte weitere Rollen.

Sind für diesen ersten Benutzer weitere Rollen wichtig, können diese über das Interface `IDefaultUser` und der Methode `AdditionalRoles()` hinzugefügt werden. Möchte man das die BA.CRM Rollen ebenfalls hinzugefügt werden, muss die eigene Implementierung `BA.CRM.Identity.DefaultUser` erweitern.

Die eigene Klasse muss [gebunden](#) werden.

21. Lösungen

21.1. Übung 1

Übersetzungen

```
"3d31e87a-1b67-4472-b53a-575bfe2e184b";"Druckmaschine";"Printing machine";"";"";"";"False";"0"
"5b9e094a-745d-47ea-93c7-a1fd5022374f";"Verpackungsmaschine";"Packing machine";"";"";"";"False";"0"
"64563a48-4bdf-4d6a-a952-7be9c90fef68";"Mechnische Presse";"Mechanical press";"";"";"";"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-F65D09193B80";
        public const string PackingMachineGuid = "7D924E88-393D-4CA3-BAAE-3C33F991A2B5";
        public const string MechanicalPressGuid = "A93A551D-54AC-47D4-A374-D98794081F16";

        public static readonly EnumEngineTypes PrintingMachine = new EnumEngineTypes(PrintingMachineGuid, 0, "3D31E87A-1B67-4472-B53A-575BFE2E184B", "Druckmaschine für Zeitungen");
        public static readonly EnumEngineTypes PackingMachine = new EnumEngineTypes(PackingMachineGuid, 1, "5B9E094A-745D-47EA-93C7-A1FD5022374F", "Automatische Verpackung");
        public static readonly EnumEngineTypes MechanicalPress = new EnumEngineTypes(MechanicalPressGuid, 2, "64563A48-4BDF-4D6A-A952-7BE9C90FEF68", "Mechanische Müllpresse");

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

        public EnumEngineTypes(String valueGuid, int sortOrder, String translationGuid, String description) : base(valueGuid, sortOrder, translationGuid)
        {
            Description = description;
        }
    }
}
```

```
        public EnumEngineTypes() { }  
    }  
}
```

21.2. Übung 2

[Download](#) als Visual Studio Template

21.3. Übung 3

Erweiterung der Übersetzungen

```
"1582c2b1-cef2-4d56-bb3d-1bdcf8aa9de6";"Responsible";"Responsible";"";"";"";"False";"0"
"507E00CC-9520-4095-8528-17FF9DBE22CB";"Engine Part";"Engine Part";"";"";"";"False";"0"
```

Erweiterung EnumRelationType

```
namespace BA.Training.Enums.Extensions
{
    [EnumExtension(typeof(EnumRelationType))]
    public static class EnumRelationTypeExtension
    {
        public const string ResponsibleGuid = "495F5CC6-8C75-4B55-A600-531BC49FE416";
        public static readonly EnumRelationType Responsible = new EnumRelationType(ResponsibleGuid, 1000, "1582C2B1-CEF2-4D56-BB3D-1BDCF8AA9DE6", nameof(Responsible));
    }
}
```

Erweiterung EnumParentRelationSubtypes

```
namespace BA.Training.Enums.Extensions
{
    [EnumExtension(typeof(EnumParentRelationSubtypes))]
    public class EnumParentRelationSubtypesExtension
    {
        public const string EnginePartGuid = "7FAFF96A-2A30-49E7-8BC7-75BB70731858";
        public static readonly EnumParentRelationSubtypes EnginePart = new EnumParentRelationSubtypes(EnginePartGuid, 1000, "507E00CC-9520-4095-8528-17FF9DBE22CB", nameof(EnginePart));
    }
}
```

Erstellung des Relationtyps

```
namespace BA.Training
```

```

{
    public class RelationCreation : IRelationCreation
    {
        public void CreateRelationConfigurations(KeyedDictionary<Guid, RelationTypeConfiguration> relationConfigurations)
        {
            relationConfigurations.Add(ResponsibleRelation());
        }

        private RelationTypeConfiguration ResponsibleRelation()
        {
            RelationTypeConfiguration relationConfiguration = new RelationTypeConfiguration()
            {
                Id = EnumRelationTypeExtension.Responsible,
                Type = EnumRelationCardinality.SomethingToAnythingGuid,
            };

            relationConfiguration.Initialize();

            relationConfiguration.SourceTypes.Add(EnumDataSource.UserProfileGuid);

            relationConfiguration.ConfigurationName = "Responsible";
            relationConfiguration.Description = "Responsible relation";
            relationConfiguration.ShowInDesigner = true;

            return relationConfiguration;
        }
    }
}

```

Erweiterung OrmEngine

```

[Newtonsoft.Json.JsonIgnore]
[DontShowFieldInDesigner]
[RelationDefinitionTarget("67D7204E-EE21-4926-B2F7-A9E82A5B8552", EnumRelationTypeExtension.ResponsibleGuid, MinCount = 0, MaxCount = 1)]
public OrmUserProfile RelatedResponsible
{
    get
    {
        return GetSourcePrimary<OrmUserProfile>(EnumRelationTypeExtension.Responsible);
    }
}

```

```
[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")]
```

21.4. Übung 4

TypeScript Datei

```
module BA.Training {
    "use strict";
    export class Actions {
        public static ClientActionAddService(event: any, customData: CustomData) {
            let igniterParams: CustomData = {};
            igniterParams['RemarkMessage'] = customData.RemarkMessage;
            customData["MassOperationIgniterParameters"] = JSON.stringify(igniterParams);
            BA.Ui.Actions.GridActions.StartMassOperation(event, customData);
        }
    }

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

Bundle Integration

```
namespace BA.Training.App_Start
{
    public class BundleConfig : BABundleConfigBase
    {
        public override void RegisterBundles()
        {
            ScriptBundle("~/bundles/scripts").Include($"~/scripts/BA.Training/Actions.js");
        }
    }
}
```

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(nameof(NavigationConfiguration.NavigationConfigurationType), ExpressionType.Equal, EnumNavigationConfigurationType.RibbonNavigationGuid, EnumControlFilterApplyState.OptIn)]
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).AssemblyQualifiedName;

        DynamicClientVisibility.Add(EnumActionVisibility.SomethingSelected);

        SomethingMustBeSelected = true;

        ActionMethodId = "BA.Training.ClientActionAddService";
    }

    public override void AdditionalRibbonButtonAssignment(DevExpress.Web.RibbonButtonItem ribbonItem, EnumActionVisibleForParentType parentType, DevExpressUI.ModelBase uiModel = null)
    {
        base.AdditionalRibbonButtonAssignment(ribbonItem, parentType, uiModel);

        AdditionalClientData.AddOrUpdate("RemarkMessage", RemarkMessage);
    }
}

```

21.5. 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", customData, 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.stringify(igniterParams);
                BA.Ui.Actions.GridActions.StartMassOperation(event, customData);
            }
        }
    );
}
```

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, HttpRequestBase request, ModelStateDictionary modelState, Dictionary<String, Object> parameter, object bindObject = null)
```

```
{
    AddServiceDialogModel dataModel;
    if (bindObject == null)
        dataModel = new AddServiceDialogModel { Date = DateTime.Now, Remark = parameter.Get("RemarkMessage").ToString() };
    else
        dataModel = (AddServiceDialogModel)bindObject;

    formModel.Title = "95437073-8D05-4650-8F75-4B53AEB4A456".Translate();
    formModel.DataSource = dataModel;
    formModel.Width = 600;

    Group group = new Group()
    {
        Id = "DEB77CD6-1823-445F-A70D-A0E6001E4351".ToGuid(),
        Caption = "",
        ColumnCount = 2,
        WrapContentAtWidth = 400,
        //StretchLastItem = true,
    };
    group.AddBreakpoint(new BreakpointRule() { Name = "S", ColumnCount = 1, MaxWidth = 500 });
    Controls.Add(group);

    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)
    };
    group.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)
    };
    group.Controls.Add(textControl);

    formModel.AddButton("BA.Training.InputButton", "5d49a7fe-8165-4fdc-953f-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");
    }

    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 remarkObj) && remarkObj is string remark && !string.IsNullOrEmpty(remark))
        {
            Session session = Api ORM.GetNewSession();
            IEnumerable<Guid> oids = Records.Select(ff => ff.Oid);
            IQueryable<OrmEngine> engines = Api ORM.GetQuery<OrmEngine>(session).Where(ff => oids.Contains(ff.Oid));
            DateTime date;
            if (IgnitionModel.Parameters.TryGetValue("ServiceDate", out object 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(), "ddlceadc-6808-4583-b6b7-dda73188b5a8".Translate(counter));
            return new JsonResult() { Data = new JsonResult() { RefreshGrid = true } };
        }
        else
        {
            Api.ClientCommunication.CreateError(Api.User.CurrentUserGuid(), "d49ef787-3ad1-4f66-bb12-b2660fd71738".Translate());
        }

        return null;
    }
}
```

21.6. Übung 6

Translation

```
"bd0d11a9-9e71-4980-9265-c4a037432d48";"Dialogberechtigte";"Dialog authorize
d";"";"";"";"False";"0"
```

Aktion

Methode `AdditionalRibbonButtonAssignment` **erweitern**

```
public override void AdditionalRibbonButtonAssignment(RibbonButtonItem ribbonItem, Dictionary<string, object> additionalClientData, EnumActionVisibleForParentType parentType, DevExUIModelBase uiModel, OrmBABase orm)
{
    // Die Basemethode sollte immer als erste ausgeführt werden
    base.AdditionalRibbonButtonAssignment(ribbonItem, additionalClientData, parentType, uiModel, orm);

    // Vorheriger Inhalt
    .....

    // Ermittlung ob die Aktion ausgeführt werden kann
    bool canHandle;
    if (orm != null)
        // Überprüfung eines konkreten Orms, ob der Benutzer es bearbeiten kann. Für Aktionen in Masken.
        // Hinweis: Oben wurde die Aktion nur für Ansichten definiert. Daher wird dieser Abschnitt
        // nicht durchlaufen und dient nur als Beispiel
        canHandle = orm.IsAllowed(EnumTableOperations.Edit);
    else
    {
        // In einer Ansicht kann nicht ein konkretes Orm angefragt werden, daher wird nur geprüft ob der Benutzer theoretisch
        // Datensätze dieses Typs bearbeiten kann. Dazu wird die Datentabellenkonfiguration geladen.
        OrmEntityConfiguration entityConfig = Api.Config.OrmEntity(EnumDataSourceExtension.Engine.ValueGuid);
        canHandle = entityConfig.IsAllowed(EnumTableOperations.Edit) != EnumTableOperations.Denied;
    }

    // Ergebnis zum Client übertragen. Siehe dazu im Konstruktor DynamicClient
```

```
Visibility.Add(EnumActionVisibility.IfUserHasRole);
    additionalClientData.AddOrUpdate("UserHasRole", 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.CurrentUserIsInRole(DialogRoles, false));
```

Igniter

```
public override ActionResult Ignite()
{
    if (SomethingSelected)
    {
        if (IgnitionModel.Parameters.TryGetValue("RemarkMessage", out object remarkObj) && remarkObj is string remark && !string.IsNullOrEmpty(remark))
        {
            DateTime date;
            if (IgnitionModel.Parameters.TryGetValue("ServiceDate", out object 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>(session).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();
                successful++;
            }
            else
            {
                missingRights++;
            }
        }

        if (missingRights == 0)
        {
            Api.ClientCommunication.CreateSuccess(Api.User.CurrentUserGuid(), "ddlceadc-6808-4583-b6b7-dda73188b5a8".Translate(successful));
        }
        else
        {
            Api.ClientCommunication.CreateSuccess(Api.User.CurrentUserGuid(), "e3f652e7-8019-4444-9e6c-f3c30528e862".Translate(successful, missingRights));
        }

        return new JsonResult() { Data = new JsonFormResult() { RefreshGrid = true } };
    }
    else
    {
        Api.ClientCommunication.CreateError(Api.User.CurrentUserGuid(), "d49ef787-3ad1-4f66-bb12-b2660fd71738".Translate());
    }

    return null;
}

```



Mit der Überprüfung der Bearbeiterrrechte 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: CustomData) {
                if (result.ButtonId == 'okButton') {
                    Actions.StartAddServiceMassOperation(event, customData, result.Data.Remark, result.Data.Date);
                }
            }
        );
    } else if (!customData.RemarkMessage || customData.RemarkMessage == "")
        BA.Ui.Translations.TranslationTools.GetTranslation("d49ef787-3ad1-4f66-bb12-b2660fd71738", function (result) {
            BA.Ui.Toast.Warning(result);
        });
    else
        Actions.StartAddServiceMassOperation(event, customData, customData.RemarkMessage, customData.ServiceDate);
}

public static StartAddServiceMassOperation(event: any, customData: CustomData, remark: any, date: any) {
    let igniterParams: CustomData = {};
    igniterParams.RemarkMessage = remark;
    igniterParams.ServiceDate = date;
    customData["MassOperationIgniterParameters"] = JSON.stringify(igniterParams);
    BA.Ui.Actions.GridActions.StartMassOperation(event, customData);
}
```

21.7. Übung 7

Übersetzungen

```
"ba98d63f-8b15-4b68-852f-102cbef70d6e";"Service hinzufügen (Massenverarbeitung)";"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 process";"";"";"";"False";"0"
"227b79dd-31f2-4f6d-a255-94eb8aca0e89";"Neustart Prozess";"Restart process";"";"";"";"False";"0"
"03a9ae68-226f-4f19-91cd-47fa4370230b";"{0} zu verarbeiten";"{0} to process";"";"";"";"False";"0"
"e3d32867-83eb-4514-abec-c7e596ae192b";"{0} von {1} verarbeitet";"{0} of {1} processed";"";"";"";"False";"0"
"1cba5efa-7842-4764-abb8-05e2fab5bde3";"Maschine {0} wird verarbeitet."; "Engine {0} processing";"";"";"";"False";"0"
"0dd299df-c558-44fd-86ec-b3b2acd47530";"Keine Bearbeitungsrechte für {0}";"No edit rights for {0}";"";"";"";"False";"0"
"cf6889ab-e880-42c9-8755-bfb15f9f01af";"Temporärer Datensatz wurde nicht gefunden: {0} / {1}";"Temporary record not found: {0} / {1}";"";"";"";"False";"0"
"160c2db4-e013-4569-8ab1-a37bdf776707";"Meine Ausnahme";"My exception";"";"";"";"False";"0"
```

Hintergrundprozess

Worker

```
namespace BA.Training.Worker
{
    /// <summary>
    /// Dieser Hintergrundprozess verarbeitet die selektierten Datensätze. Die
    /// se wurden vom Igniter in die temporäre Tabelle geschrieben.
    /// Über die <see cref="TaskExecutionId"/> können die Datensätze ermittelt
    /// werden.
    ///
    /// Wenn eine Verarbeitung von Datensätzen selbst implementiert wird. Muss
    /// beim Konzept beachtet werden, das der Prozess auf Fehler stoßen
    /// und die Verarbeitung beispielsweise durch Herunterfahren der Anwendung
    /// unterbrochen werden kann.
    ///
    /// In diesem Beispiel ist die zu verarbeitende Menge durch die Einträge in
    /// der temporären Tabelle definiert. Wenn ein Datensatz verarbeitet
```

```
/// worden ist, wird der entsprechende Eintrag in der temporären Tabelle g
elöscht.
///
/// Durch das Verwenden der <see cref="UnitOfWork"/> wird sichergestellt,
das das Löschen des Eintrags und die Verarbeitung eine Transaktion ist.
/// </summary>
public class AddServiceWorker : WorkItemBase
{
    // Id der Datensätze in der temporären Tabelle
    public Guid TaskExecutionId { get; set; }

    // Die zu setzende Bemerkung
    public string Remark { get; set; }

    // Das zu setzende Datum
    public DateTime Date { get; set; }

    // Zähler für Datensätze bei denen die Rechte fehlen
    public int MissingRights { get; set; }

    public AddServiceWorker() : base()
    {
        // Die erwartete Laufzeit ist land. Damit die entsprechende Queue
genutzt wird.
        ExpectedRunTime = EnumExpectedRunTime.Long;
    }

    /// <summary>
    /// Implementierung des Prozesses. Wird bei Jedem Start aufgerufen
    /// </summary>
    protected override void Run()
    {
        // Abfrage auf die temporäre Tabelle
        IQueryable<OrmTempSelectedRecords> query = Api.ORM.GetQuery<OrmTem
pSelectedRecords>(UnitOfWork);
        query = query.Where(ff => ff.TaskExecutionId == TaskExecutionId);

        // Maximale Anzahl setzen. Wenn sie größer als 0 ist. Wird der Pro
zess neugestartet und der Count der Abfrage enthält nicht
        // mehr die schon verarbeiteten Datensätze.
        if (MaxProgress == 0)
            MaxProgress = query.Count();

        // Abfrage der zu verarbeitenden Maschinen
        IQueryable<Guid> oids = query.Select(ff => ff.SelectedRecordOid);
        IQueryable<OrmEngine> engines = Api.ORM.GetQuery<OrmEngine>(UnitOf
Work).Where(ff => oids.Contains(ff.Oid));
```

```

        foreach (OrmEngine engine in engines)
        {
            // Abprüfung der Rechte
            if (engine.IsAllowed(EnumTableOperations.Edit))
            {
                // Verarbeitung des aktuellen Datensatzes
                OrmSubEngineServices service = engine.Services.AddNewObject();

                service.SortOrder = engine.Services.Count() - 1;
                service.ServiceDate = Date;
                service.Remark = Remark;
                engine.Save();
            }
            else
            {
                MissingRights++;

                // Löschen des temporären Eintrages
                query.Where(ff => ff.SelectedRecordOid == engine.Oid).FirstOrDefault()?.Delete();

                // Exception, wenn der dritte Datensatz im ersten Durchlauf verarbeitet wird.
                if (CurrentProgress == 2 && NumberOfStarts == 1)
                    throw new Exception(Logger.Translate("160C2DB4-E013-4569-8AB1-A37BDF776707"));

                // Zähler für die verarbeiteten Datensätze
                CurrentProgress++;
                Save();
            }
        }

        /// <summary>
        /// Prozess ist beendet und dem Anwender wird eine Nachricht zugestellt.
        /// </summary>
        protected override void WorkItemFinished()
        {
            if (State == EnumWorkItemState.Finished)
            {
                if (MissingRights == 0)
                {
                    // Der Anwender hatte auf alle Datensätze ausreichend Rechte

                    Api.ClientCommunication.CreateSuccess(Api.User.CurrentUser.Guid(), "dd1ceadc-6808-4583-b6b7-dda73188b5a8".Translate(CurrentProgress));
                }
                else
                {
                    // Auf einige Datensätze hatte der Anwender nicht genügend Rechte

```

```

        Api.ClientCommunication.CreateSuccess(Api.User.CurrentUser
Guid(), "e3f652e7-8019-4444-9e6c-f3c30528e862".Translate(CurrentProgress - Mis
singRights, 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.IsNullOrEmpty(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

```
namespace BA.Training.Configuration.Navigation
{
    /// <summary>
    /// Aktion mit einem Hintergrundprozess. Die Funktionalität wird asynchr
    n ausgeführt und der Benutzer muss NICHT warten.
    /// </summary>
    [Serializable]
    [Toolbox(EnumConfigurationType.NavigationConfigurationGuid, true)]
    [ControlFilter("NavigationConfigurationType", ExpressionType.Equal, EnumNa
    vigationConfigurationType.RibbonNavigationGuid, EnumControlFilterApplyState.If
    Positive)]
    public class ClientActionAddServiceMassWorker : ClientActionGridMassOperat
    ionBase
    {
        [DisplayName("41ABA083-E37A-4709-98D4-1D685496459C")]
        public string RemarkMessage { get; set; }

        [TokenboxControl]
        [CDPRolesProviderProperties(dataSources: new[] { EnumDataSource.RoleGu
    id })]
        [DisplayName("BD0D11A9-9E71-4980-9265-C4A037432D48")]
        public RoleSet DialogRoles { 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;

        // Action ist nur für Ansichten
        VisibilityForParentTypes.Add(EnumActionVisibleForParentType.Grid);

        // Igniter zum Erstellen des Workers setzen
        MassOperationIgniter = typeof(AddServiceWorkerIgniter).AssemblyQualified
        Name;

        // Es muss etwas selektiert sein. Auf false setzen, wenn die Aktion
        auch funktioniert, wenn nichts gesetzt ist.
        // Siehe Eigenschaft AllRecordsIfNothingIsSelectedIndicator im Ign
        iter
        SomethingMustBeSelected = true;

        // Sichtbarkeitssteuerung bei welcher von Selektionen die Aktion s
        ichtbar ist
        // Weitere Möglichkeiten: OneSelected, OneOrNothingSelectedGuid, N
        othingSelected, ManySelected
        DynamicClientVisibility.Add(EnumActionVisibility.SomethingSelecte
        d);

        // Wird in CanHandleOrm mit true oder false belegt
        DynamicClientVisibility.Add(EnumActionVisibility.IfUserHasRole);

        // Clientseitige Aktion, die ausgeführt wird
        ActionMethodId = "BA.Training.ClientActionAddService";
    }

    /// <inheritdoc>
    public override void AdditionalRibbonButtonAssignment(RibbonButtonItem
    m ribbonItem, Dictionary<string, object> additionalClientData, EnumActionVisib
    leForParentType parentType, DevExUIModelBase uiModel, OrmBABase orm)
    {
        // Die Basemethode sollte immer als erste ausgeführt werden
        base.AdditionalRibbonButtonAssignment(ribbonItem, additionalClient
        Data, parentType, uiModel, orm);

        // Übertragung der konfigurierten Werte zum Client
        additionalClientData.AddOrUpdate("RemarkMessage", RemarkMessage);
        additionalClientData.AddOrUpdate("DialogAuthorized", Api.User.Curr
        entUserIsInRole(DialogRoles, false));

        // Ermittlung ob die Aktion ausgeführt werden kann
        bool canHandle;
        if (orm != null)
            // Überprüfung eines konkreten Orms, ob der Benutzer es bearbe

```

```

iten kann. Für Aktionen in Masken.
        // Hinweis: Oben wurde die Aktion nur für Ansichten definiert.
        // Daher wird dieser Abschnitt
        // nicht durchlaufen und dient nur als Beispiel
        canHandle = orm.IsAllowed(EnumTableOperations.Edit);
    else
    {
        // In einer Ansicht kann nicht ein konkretes Orm angefragt werden,
        // daher wird nur geprüft ob der Benutzer theoretisch
        // Datensätze dieses Typs bearbeiten kann. Dazu wird die Datentabellenkonfiguration geladen.
        OrmEntityConfiguration entityConfig = Api.Config.OrmEntity(EnumDataSourceExtension.Engine.ValueGuid);
        canHandle = entityConfig.IsAllowed(EnumTableOperations.Edit)
        != EnumTableOperations.Denied;
    }

    // Ergebnis zum Client übertragen. Siehe dazu im Konstruktor DynamicClientVisibility.Add(EnumActionVisibility.IfUserHasRole);
    additionalClientData.AddOrUpdate("UserHasRole", canHandle);
}
}
}

```

Erweiterung von EnumLogProcess

```

namespace BA.Training.Enums.Extensions
{
    [EnumExtension(typeof(EnumLogProcesses))]
    public static class EnumLogProcessesExtension
    {
        public const string AddServiceGuid = "FBF67FBA-9E64-4254-B168-A0C8DF806C63";
        public static readonly EnumLogProcesses AddService = new EnumLogProcesses(AddServiceGuid, 1000, "281E4152-6D03-40B4-9971-10728E673A00");
    }
}

```

Worker

```

namespace BA.Training.Worker
{
    /// <summary>
    /// Die Verarbeitung der Einträge der temporären Tabelle wird von dieser Basisklasse automatisch sicher gestellt.

```



```

    /// Daher ist die Komplexität viel geringer.
    /// </summary>
    public class AddServiceMassWorker : TemporaryRecordMassWorkItem<OrmEngine>
    {
        public string Remark { get; set; }
        public DateTime Date { get; set; }
        public int MissingRights { get; set; }

        public AddServiceMassWorker() : base()
        {
            // Der Prozess wird um eine Minute verzögert gestartet.
            ScheduledStartTime = DateTime.UtcNow.AddMinutes(1);

            // Der Prozess kann von dem Benutzer in der UI unterbrochen werde
n.

            IsCancellable = true;

            // Beschreibende Informationen
            LoggingProcess = EnumLogProcessesExtension.AddService;
            Caption = "281e4152-6d03-40b4-9971-10728e673a00";
            Title = "281e4152-6d03-40b4-9971-10728e673a00";

            // Die ChunkSize gibt an in welchen Paketen die zu verarbeitenden
Datensätze geladen werden
            // Default ist 200
            ChunkSize = 200;
        }

        protected override void BeforeProcessing(bool resume)
        {
            if (NumberOfStarts == 1)
            {
                // Erster Start des Prozesses
                Logger.AddInfo("61BCA157-96A6-436B-8719-0521B98462B9");
                Logger.AddInfo("03A9AE68-226F-4F19-91CD-47FA4370230B", MaxProg
ress);

                // Exception zum Testen des Verhaltens
                throw new Exception(Logger.Translate("160C2DB4-E013-4569-8AB
1-A37BDF776707"));
            }
            else
            {
                // Neustart des Prozesses
                Logger.AddInfo("227B79DD-31F2-4F6D-A255-94EB8ACA0E89");
                Logger.AddInfo("E3D32867-83EB-4514-ABEC-C7E596AE192B", Current
Progress, MaxProgress);
            }
        }
    }

```

```
    }
}

/// <summary>
/// Verarbeitung eines Datensatzes
/// </summary>
/// <param name="engine"></param>
protected override void ProcessSingleOrm(OrmEngine engine)
{
    Logger.AddInfo("1CBA5EFA-7842-4764-ABB8-05E2FAB5BDE3", engine.Name);

    // Exception zum Testen des Verhaltens
    if (CurrentProgress == 2)
        throw new Exception(Logger.Translate("160C2DB4-E013-4569-8AB1-A37BDF776707"));

    // Abprüfung der Rechte
    if (engine.IsAllowed(EnumTableOperations.Edit))
    {
        // Verarbeitung des aktuellen Datensatzes
        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.Name);

        MissingRights++;
    }
}

protected override void AfterProcessing(Exception exception)
{
    if (MissingRights == 0)
    {
        Api.ClientCommunication.CreateSuccess(Api.User.CurrentUserGuid(), "dd1ceadc-6808-4583-b6b7-dda73188b5a8".Translate(CurrentProgress));
        Logger.AddInfo("dd1ceadc-6808-4583-b6b7-dda73188b5a8", CurrentProgress);
    }
    else
    {
        Api.ClientCommunication.CreateError(Api.User.CurrentUserGui
```

```

d(), "e3f652e7-8019-4444-9e6c-f3c30528e862".Translate(CurrentProgress - MissingRights, MissingRights));
        Logger.AddInfo("e3f652e7-8019-4444-9e6c-f3c30528e862", CurrentProgress - MissingRights, MissingRights);
    }
}
}
}
}

```

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 im zweiten Hintergrundprozess nochmal zu verarbeiten?

Im ersten Worker wurde durch die Ausnahme der gesamte Prozess abgebrochen. Dies geschah bevor der Eintrag aus der temporären Tabelle entfernt wurde. Nach einer Ausnahme wird ein Prozess neu eingeplant und beim Neustart wurde bei dem Datensatz fortgefahren.

Der `TemporaryRecordMassWorkItem` überpringt Datensätze bei denen Ausnahmen auftreten und der Prozess wird nicht abgebrochen. Um dies zu ändern, muss in `OnError` eine Ausnahme geworfen werden.

21.8. 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 Maschinendatensatz 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. Zwei eigene Formelfunktionen `BATResponsible`, für den Zugriff auf den Betreuer in Relation und `BATWriteText`, die auch in .Net ausgeführt werden kann.

[Download](#)