

Diese Dokumentation bezieht sich auf den Stand von NHibernate 1.2

1. Vorwort

Dieses Dokument wurde nach bestem Wissen erstellt und erhebt nicht den Anspruch auf 100%tige Korrektheit und Vollständigkeit. Ich bitte vorab jeden Leser nach eigenem Wissen zu prüfen, ob die hier gelieferten Informationen korrekt sind, und mir ggf. Korrekturen und Verbesserungsvorschläge direkt an meine e-Mail Adresse daniel@kreuzhofer.de zu schicken. Ich bin für jeden Kommentar sehr dankbar.

2. Grundsätzliches zu NHibernate Mapping Attributen

NHibernate Mapping Attributes ist ein Addon zu NHibernate (Siehe auch http://www.hibernate.org/hib_docs/nhibernate/1.2/reference/en/html/mapping-attributes.html), welche es einem erspart, die Mappings der NHibernate Klassen auf die Datenbanktabellen und Spalten in einer XML-Datei definieren zu müssen. Stattdessen können die Mappings mit Hilfe spezieller .NET Attribute direkt vor den eigentlichen Klassen und Properties erfolgen. Man hat also auf sehr elegante Weise die Mappings und Klassen/Properties zusammen in einer Definition zusammengefasst. Im Folgenden werden einige Beispiele gegeben, wie diese Mappings zu definieren sind.

An einigen Stellen wird auf die Original NHibernate Dokumentation verwiesen, welche allerdings nur die Erklärungen für die XML-Mappingdeklaration enthält. Es sind allerdings alle Parameter der XML-Definition auch in den NHibernate Mapping Attributes vorhanden, so dass die Dokumentation der Parameter aus der XML-Definition ihre volle Gültigkeit behält und prinzipiell keine eigene Dokumentation nötig ist. Da sich das Mapping mit .NET Attributen aber nicht einfach intuitiv aus der XML-Variante ableiten lässt, habe ich diese Dokumentation verfasst und hoffe damit, den meisten Anwendungsfällen gerecht zu werden. Diese Dokumentation erhebt nicht den Anspruch auf Vollständigkeit und kann die Original NHibernate Dokumentation keinesfalls ersetzen.

2.1. Beispieldateien

Sämtliche Beispiele aus diesem Dokument sind in einer Visual Studio Solution zusammengefasst und können unter der folgenden URL als gezipptes Archiv herunter geladen werden:

<http://www.kreuzhofer.de/upload/NHibernateMappingAttributesTestPackage.zip>

3. NHibernate Klassenattribute

Zum Mapping einer Klasse mit NHibernate muss diese zunächst mit dem Attribut „Class“ versehen werden. Im folgenden Beispiel definieren wir die Klasse „Customer“, die mittels NHibernate auf eine entsprechende Datenbanktabelle gemappt werden soll.

```
[NHibernate.Mapping.Attributes.Class(0)]
public class Customer
{
    // Hier kommen die Property Mappings..
}
```

Das sieht doch schon ziemlich einfach aus. In diesem Fall definieren wir einfach nur die Klasse Customer als NHibernate gemappte Klasse. Wir könnten noch den Parameter „Table“ angeben (Siehe nächstes Beispiel), mit dem wir NHibernate mitteilen wie die Datenbanktabelle heißen soll, die auf diese Klasse gemappt werden soll. Da wir im ersten Beispiel nichts angegeben haben, geht NHibernate davon aus, dass die Datenbanktabelle gleich dem Klassennamen sein soll. Im einfachsten Fall benötigt man nur den Parameter Index, der in diesem Fall „0“ ist. Für alle weiteren Beispiele und Erläuterungen gilt: Der Index-Parameter wird **immer** gesetzt und

definiert, in welcher Reihenfolge die Attribute von NHibernate anzuwenden sind. Dies kommt daher, dass in der sonst üblichen XML-Definition der NHibernate Mappings bestimmte XML-Tags wiederum Child-Tags haben. Bei NHibernate Mapping Attributes soll dieses Parent-Child-Verhältnis mithilfe des Index Parameters abgebildet werden. Es entspricht aber nicht immer 100%tig dem Vorgehen im XML-File, bzw den wirklichen Parent und Child Beziehungen der Tags im XML-File, wie wir später noch sehen werden.

```
[NHibernate.Mapping.Attributes.Class(0, Table = "Kunden", Schema = "tst")]
public class Customer
{
    // Hier kommen die Property Mappings..
}
```

Erklärungen zu den Parametern:

Table: Bezeichnet die Datenbanktabelle, die auf diese Klasse gemappt werden soll

Schema: Das Datenbankschema, in dem die Tabelle angelegt werden soll. Wenn hier im Fall des SQL Servers nichts angegeben wird, wird automatisch das Standardschema (üblicherweise „dbo“) verwendet.

Weitere Parameter vom „Class“-Attribut siehe :

http://www.hibernate.org/hib_docs/nhibernate/1.2/reference/en/html/mapping.html#mapping-declaration-class .

4. Identitätsspalten / Primary Keys

4.1. Bedeutung

Jede Datenbanktabelle hat normalerweise einen Primary Key bzw. eine/mehrere Identitätsspalte(n). Zunächst beschäftigen wir uns mit Identitätsspalten, die mit Autowerten vorbelegt werden. Diese können durch verschiedene Datentypen repräsentiert werden, z.B. „long“ oder „Guid“. Der Primary Key wird in der Regel durch die Datenbank mit einem Wert belegt. Bei Autoincrement Werten bedeutet dies, dass jeder neue Eintrag in der betreffenden Tabelle automatisch die nächste verfügbare Zahl erhält, die in der Regel um 1 größer ist als die bisher höchste Id. Bei Guid-Spalten wird für jeden neuen Eintrag in der Tabelle eine neue Guid erzeugt.

(Details: http://www.hibernate.org/hib_docs/v3/reference/en/html/mapping.html#mapping-declaration-id)

4.2. Mapping Attribute

Im Folgenden Beispiel wird eine Auto-Increment Identitätsspalte vom Typ int, welche auch der Primary Key für die Tabelle sein soll, erzeugt:

```
[NHibernate.Mapping.Attributes.Id(0,
    Column = "ID",
    Name = "ID",
    TypeType = typeof(int),
    UnsavedValue = "0"
)]
[NHibernate.Mapping.Attributes.Generator(1, Class = "native")]
public override int ID
{
    get { return m_ID; }
    set { m_ID = value; }
}

private int m_ID;
```

Das Attribut „Id“ bezeichnet die Identität/Primary Key einer NHibernate Klasse und hat folgende Parameter:

Column: Name der Datenbankspalte

Name: Name des Properties in der NHibernate Klasse. Dieser Parameter ist meist optional. Wenn er fehlt, wird automatisch der Name des dem Attribut folgenden Properties für „Name“ verwendet.

TypeType: Datentyp für die Datenbankspalte (es ist der NHibernate Datentyp gemeint. Siehe auch http://www.hibernate.org/hib_docs/nhibernate/1.2/reference/en/html/mapping.html#mapping-types)

UnsavedValue: Wenn eine neue Instanz dieser NHibernate Klasse erzeugt wird, dann wird das Property mit diesem Wert vorbelegt und somit als „neu und nicht gespeichert“ gekennzeichnet. Wenn dieser Parameter nicht angegeben wird, dann wird von NHibernate automatisch ein Wert vorgegeben, der entsprechend klar kennzeichnet, dass es sich um einen neuen Eintrag handelt.

Zusätzlich muss man nun im Falle eines Auto-Wertes einen sog. „Generator“ angeben. Der wichtigste Parameter ist „Class“, welcher mit „native“ belegt wird, falls es sich um einen Autoincrement-Wert handeln soll. Native ist im Gegensatz zu „identity“, „sequence“ oder „hilo“ die bessere Wahl, da NHibernate in diesem Fall automatisch einen dieser drei Generatortyp verwendet, der in der unterliegenden Datenbank am besten zutrifft.

Im nächsten Beispiel wird eine Identitätsspalte angelegt, die vom Typ Guid ist. Die Guid soll automatisch erzeugt werden.

```
[NHibernate.Mapping.Attributes.Id(0,
    Column = "ID",
    Name = "ID",
    TypeType = typeof(Guid),
    UnsavedValue = "(00000000-0000-0000-0000-000000000000)"
)]
[NHibernate.Mapping.Attributes.Generator(1,
    Class = "guid.comb"
)]
public override System.Guid ID
{
    get { return m_ID; }
    set { m_ID = value; }
}

private Guid m_ID;
```

Im Gegensatz zum vorherigen Beispiel ist hier der Datentyp eine Guid. Damit diese Guid automatisch erzeugt wird, benötigt man zusätzlich das Generator-Attribut mit dem Parameter „Class“, welcher auf „guid.comb“ gesetzt werden sollte. Es gibt auch den Generator „guid“ welcher allerdings wesentlich langsamer arbeitet als „guid.comb“ (Siehe: <http://agile.codebetter.com/blogs/jeffrey.palermo/archive/2006/08/19/148448.aspx>). „guid.comb“ steht allerdings nur bei Microsoft SQL Server ab Version 2000 zur Verfügung.

5. „Einfache“ Felder / Properties

Als nächstes wenden wir uns den „einfachen“ Feldern einer Tabelle zu, die einfache Datentypen darstellen und keine Primary oder Foreign Key Beziehungen repräsentieren.

Generell wird das Mapping eines einfachen Felds mit dem Attribut „Property“ definiert. Dazu zunächst ein Beispiel:

```
[NHibernate.Mapping.Attributes.Property(0,
    Column = "Telefone",
    Name = "Telefone",
    TypeType = typeof(string),
    Length = 50,
    NotNull = false
)]
public virtual string Telefone
{
    get { return m_Telefone; }
    set { m_Telefone = value; }
}

private string m_Telefone;
```

Die in diesem Beispiel verwendeten Parameter sind sicherlich die am häufigsten benötigten und haben folgende Bedeutung:

Column: Die Datenbankspalte für dieses Feld

Name: Der Name des Properties in dieser Klasse, auf das dieses Feld gemappt werden soll.

TypeType: Der NHibernate Typ für dieses Feld (es handelt sich zwar meistens um .NET Typen, aber es gibt auch spezielle NHibernate Typen, die hier angegeben werden können)

Length: Im Fall eines Strings ist dieser Parameter wichtig, um die maximale Länge dieses Felds in

der Datenbank angeben zu können. In diesem Beispiel würde eine Datenbankspalte vom Typ „varchar“ mit der maximalen Länge von 50 Zeichen erzeugt werden.

NotNull: Kann false oder true sein und gibt an, ob dieses Feld in der Datenbank als NotNull deklariert werden soll.

Weitere mögliche Parameter für das „Property“ Attribut siehe:

http://www.hibernate.org/hib_docs/nhibernate/1.2/reference/en/html/mapping.html#mapping-declaration-property

5.1. Weitere Beispiele für einfache Felder

Wir wollen nun in einer Tabelle ein Feld definieren, das eine e-Mail Adresse enthält und zusätzlich vorgeben, dass jede e-Mail Adresse in dieser Tabelle nur genau einmal vorkommen darf.

```
[NHibernate.Mapping.Attributes.Property(0,
    Name = "EMail",
    TypeType = typeof(string),
    Length = 50,
    NotNull = false
)]
[NHibernate.Mapping.Attributes.Column(1,
    Name = "EMail",
    Unique = true,
    UniqueKey = "IX_Contact_Email"
)]
public virtual string EMail
{
    get { return m_Email; }
    set { m_Email = value; }
}

private string m_Email;
```

Für diese Konstellation benötigt man zusätzlich das Attribut „Column“ über welches man detailliertere Informationen zur Datenbankspalte vorgeben kann. Man beachte, dass nun der Parameter „Name“ im Column-Attribut verwendet wird und nicht im „Property“ Attribut. Damit nun jede eMail Adresse nur einmal existiert müssen wir dies NHibernate mit dem Parameter „Unique“ mitteilen. Zusätzlich kann man optional „UniqueKey“ definieren, um dem automatisch in der Datenbank erstellten Unique Key einen aussagekräftigeren Namen zu geben. Wenn man diesen Parameter nicht angibt, erzeugt NHibernate selbst einen sehr kryptischen Schlüsselnamen. In unserem Beispiel haben wir dem UniqueKey die Bezeichnung „IX_Contact_Email“ gegeben, da es sich um das Feld „Email“ in der Tabelle „Contact“ handelt und „IX“, weil es sich um einen Unique-Key handelt (Übernommen aus der üblichen SQL Server Nomenklatur).

Jetzt geben wir noch zwei weitere Beispiele, eines für ein „DateTime“ Feld, ein „double“ Feld und eines für ein „byte[]“ Feld, das in der Datenbank im Fall des SQL Servers mit dem Typ „Image“ repräsentiert wird. Die Beispiele bringen außer den Typen nichts neues und sind daher selbsterklärend.

```

private DateTime m_IssueDate;
private double m_Costs;
private byte[] m_Document;

[NHibernate.Mapping.Attributes.Property(0,
    Name = "IssueDate",
    TypeType = typeof(DateTime),
    Column = "IssueDate",
    NotNull = true
)]
public DateTime IssueDate
{
    get { return m_IssueDate; }
    set { m_IssueDate = value; }
}

[NHibernate.Mapping.Attributes.Property(0,
    Name = "Costs",
    TypeType = typeof(double),
    Column = "Costs",
    NotNull = true
)]
public double Costs
{
    get { return m_Costs; }
    set { m_Costs = value; }
}

[NHibernate.Mapping.Attributes.Property(0,
    Name = "Document",
    TypeType = typeof(byte[]),
    Column = "Document",
    NotNull = true
)]
public byte[] Document
{
    get { return m_Document; }
    set { m_Document = value; }
}

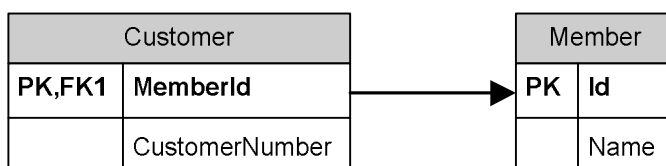
```

6. 1 zu 1 (One To One) Beziehungen

6.1. Bedeutung

Eine 1 zu 1 Beziehung kennzeichnet, dass zwei Tabellen so miteinander verknüpft sind, dass der Primary Key der ersten Tabelle zugleich der Primary Key und Foreign Key der zweiten Tabelle ist. Am einfachsten lässt sich das an einem Beispiel erklären. Wir haben 2 Tabellen: Member und Customer. Member sind in diesem Fall „Mitglieder“ eines Systems. Ein Member hat eine Id und einen Namen. Ein Customer „Ist“ ein Member dieses Systems mit der zusätzlichen Eigenschaft „CustomerNumber“. Diese Art von Beziehung wird im folgenden Diagramm dargestellt.

6.2. Beispieldiagramm



Customer steht zu Member in einer 1 zu 1 Beziehung. In diesem Fall ist das dadurch

gekennzeichnet, dass Customer als Primary Key einen Foreign Key MemberId hat, der auf den Primary Key Id in der Member Tabelle verweist. Das heißt, ein Datensatz in der Tabelle Customer liest sich wie „Ist Member“ und nicht „Hat einen Member“. Einträge in der Customer Tabelle sind also nur möglich, wenn die MemberId auf eine in der Member-Tabelle bereits existierenden Eintrag verweist. Außerdem kann es immer nur einen einzigen Customer-Eintrag, zu einem Member-Eintrag geben und nicht mehrere. Wenn man in so einem Fall versuchen würde einen Eintrag in der Customer-Tabelle mit derselben MemberId zu machen, würde es eine Constraint-Exception geben, da die 1 zu 1 Beziehung dadurch verletzt wäre.

6.3. Mapping Attribute

Um das obige Beispiel mit Nhibernate Mapping Attributes zu mappen, wurden 2 Klassen erstellt, deren Mappings ich anschließend erklären werde. Zunächst einmal die Definition der Klasse Member:

```
[NHibernate.Mapping.Attributes.Class(0)]
public class Member
{
    private Guid m_ID;
    private String m_Name;
    private Customer m_Customer;

    [NHibernate.Mapping.Attributes.Id(0,
        Column = "ID",
        Name = "ID",
        TypeType = typeof(Guid),
        UnsavedValue = "(00000000-0000-0000-0000-000000000000)"
    )]
    public virtual Guid ID
    {
        get { return m_ID; }
        set { m_ID = value; }
    }

    [NHibernate.Mapping.Attributes.Property(0,
        Column = "Name",
        Name = "Name",
        TypeType = typeof(string),
        Length = 50,
        NotNull = false
    )]
    public virtual string Name
    {
        get { return m_Name; }
        set { m_Name = value; }
    }

    [NHibernate.Mapping.Attributes.OneToOne(0,
        ClassType = typeof(Customer),
        Name = "Customer",
        Lazy = NHibernate.Mapping.Attributes.Laziness.Proxy,
        Cascade = NHibernate.Mapping.Attributes.CascadeStyle.All
    )]
    public virtual Customer Customer
    {
        get { return m_Customer; }
        set { m_Customer = value; }
    }
}
```

Man sieht schon – eigentlich ist diese Klasse zu den vorangegangenen Beispielen keine große Veränderung – bis auf das Mapping von Customer, auf das ich nun eingehen werde.

Um die 1 zu 1 Beziehung von Member zu Customer zu definieren legen wir ein Property vom Typ Customer (Customer-Klasse, siehe nächster Code-Abschnitt) an.

Wir verwenden das Attribut „OneToOne“, um eine 1 zu 1 Beziehung zu kennzeichnen. Der Parameter „ClassType“ bezeichnet die Klasse, die die Andere Seite der Beziehung definiert, in diesem Fall also „Customer“. Zusätzliche Parameter sind:

Lazy: Definiert die Lazy-Loading Strategie, die auf dieses Property angewendet werden soll. Proxy bedeutet, dass der Customer zu einem Member genau dann geladen wird, wenn das erste Mal auf das Property lesend zugegriffen wird.

Cascade: Definiert, wie mit abhängigen Objekten im Falle von Save, Update und Delete vorgegangen wird. In diesem Fall möchten wir, dass immer vollständig kaskadiert gespeichert, geupdated und gelöscht wird. Wenn man also ein neues Member Objekt anlegt und dem Property ein neues Customer Objekt zuweist, dann wird dieses automatisch mit abgespeichert, wenn man das Member-Objekt speichert. Genauso verhält es sich beim updaten und löschen. Da ein Customer ohne einen Member nicht existieren kann, wird ein eventuell vorhandener Customer automatisch gelöscht, wenn der zugehörige Member gelöscht wird.

Weitere Erläuterungen siehe:

http://www.hibernate.org/hib_docs/nhibernate/1.2/reference/en/html/mapping.html#mapping-declaration-onetoone

Als nächstes sehen wir uns die Klasse Customer an. Dort sieht das Mapping des Foreign Keys zu Member schon um einiges komplexer aus.

```

[NHibernate.Mapping.Attributes.Class(0)]
public class Customer
{
    private Member m_Member;
    private String m_CustomerNumber;

    [NHibernate.Mapping.Attributes.Id(0,
        Column = "MemberID",
        TypeType = typeof(Guid),
        UnsavedValue = "(00000000-0000-0000-0000-000000000000)")]
    [NHibernate.Mapping.Attributes.Generator(1,
        Class = "foreign")]
    [NHibernate.Mapping.Attributes.Param(2,
        Content = "ID",
        Name = "property")]
    [NHibernate.Mapping.Attributes.OneToOne(3,
        ClassType = typeof(Member),
        Constrained = true,
        OuterJoin =
        NHibernate.Mapping.Attributes.OuterJoinStrategy.Auto,
        ForeignKey = "FK_Customer_Member")]
    public virtual Member ID
    {
        get { return m_Member; }
        set { m_Member = value; }
    }

    [NHibernate.Mapping.Attributes.Property(0,
        Column = "CustomerNumber",
        Name = "CustomerNumber",
        TypeType = typeof(string),
        Length = 50,
        NotNull = false
    )]
    public virtual string CustomerNumber
    {
        get { return m_CustomerNumber; }
        set { m_CustomerNumber = value; }
    }
}

```

Fangen wir von oben an. Zuerst wird der Primary Key definiert. Da der Primary Key in der Customer Tabelle ein Foreign Key ist und im Endeffekt die Id von Member darstellt, benennen wir die Datenbankspalte mit „MemberId“. Damit ist auch klar, woher diese Id stammt. Der TypeType muss identisch mit der Id von Member sein, also in diesem Fall Guid.

Das nächste erklärungsbedürftige Attribut ist „Generator“. Da es sich bei diesem Primary Key nicht um einen Autowert handelt, sondern der Wert aus einer Fremdtabelle bezogen wird, muss hier für den Parameter „Class“ der Wert „foreign“ angegeben werden.

Das folgende Attribut „Param“ bezieht sich insofern noch auf das vorherige „Generator“ Attribut, als das es weitere Optionen für das „Generator“ Attribut definiert: Mit dem Parameter „Content“ wird festgelegt, aus welchem Property im Objekt „Member“ der foreign Key bezogen wird. Mit dem Parameter „Name“ = „property“ wird festgelegt, dass sich der foreign Key Wert auf ein Property bezieht, also auf das Property „ID“ aus dem Member Objekt.

Als letztes Attribut benötigen wir das OneToOne Attribut, wie auch schon in der Member Klasse. Wir legen den „Classtype“ auf „Member“ fest, um dem Mappinggenerator mitzuteilen, dass die zugehörige Klasse auf der anderen Seite der OneToOne Beziehung die Klasse Member ist. „Constrained“ = „true“ ist sehr wichtig und darf nicht vergessen werden. Dieser Parameter

veranlasst eigentlich erst das Anlegen einer Beziehung in der SQL Datenbank zwischen den beiden Tabellen. Somit wird einerseits gewährleistet, dass zu einem Member immer nur ein Customer angelegt werden kann und dass auch der Member nicht gelöscht werden kann, wenn ein Customer dazu existiert.

„OuterJoin“ legt die Art der Join-Strategie bei den Select Statements von Nhibernate fest. Im Normalfall kann hier „`NHibernate.Mapping.Attributes.OuterJoinStrategy.Auto`“ verwendet werden.

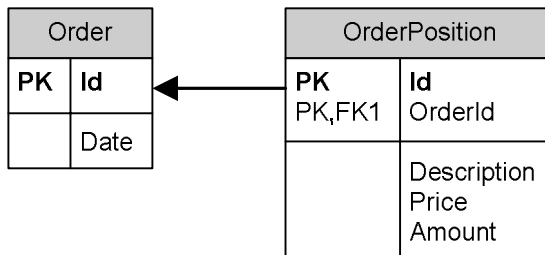
Der Parameter „ForeignKey“ legt wie im vorangegangenen Beispiel einen aussagekräftigen Namen für den Foreign Key fest, der ansonsten von Nhibernate automatisch vergeben wird.

7. One To Many Beziehungen

7.1. Bedeutung

Eine One To Many Beziehung kennzeichnet eine Objektbeziehung zwischen einer Tabelle, zu der ein oder mehrere abhängige Einträge in einer zweiten Tabelle existieren. In unserem folgenden Beispiel zeigen wir das Anhand einer Bestellung (Order) und den dazugehörigen Bestellpositionen (OrderPosition). Eine Bestellung kann eine bis n Bestellpositionen haben.

7.2. Beispieldiagramm



Die One To Many Beziehung zwischen Order und Orderposition wird durch das Vorhandensein eines eigenen Primary Keys in der OrderPosition-Tabelle und eines zusätzlichen Foreign Keys in Form der OrderId-Spalte gekennzeichnet. Die Foreign Key Beziehung verweist auf die Id-Spalte in der Order-Tabelle.

7.3. Mapping Attribute

Wie wird nun diese Beziehung mittels NHibernate Mapping Attributes definiert? Zunächst einmal schauen wir uns das Mapping für die Order-Tabelle an:

```

[NHibernate.Mapping.Attributes.Class(0, Table = "Orders")]
public class Order
{
    private Guid m_ID;
    private IList<OrderPosition> m_Positions = new List<OrderPosition>();

    [NHibernate.Mapping.Attributes.Id(0,
        Column = "ID",
        Name = "ID",
        TypeType = typeof(Guid),
        UnsavedValue = "(00000000-0000-0000-0000-000000000000)"
    )]
    [NHibernate.Mapping.Attributes.Generator(1, Class = "guid.comb")]
    public virtual Guid ID
    {
        get { return m_ID; }
        set { m_ID = value; }
    }

    [NHibernate.Mapping.Attributes.Bag(0,
        Cascade = NHibernate.Mapping.Attributes.CascadeStyle.All,
        Inverse = true
    )]
    [NHibernate.Mapping.Attributes.Key(1,
        Column = "OrderID"
    )]
    [NHibernate.Mapping.Attributes.OneToMany(2,
        ClassType = typeof(OrderPosition)
    )]
    public virtual IList<OrderPosition> Positions
    {
        get { return m_Positions; }
        set { m_Positions = value; }
    }

    // ... weitere Felder ...
}

```

Die Definition der Order Klasse erschließt sich nach den vorangegangenen Beispielen bis auf das Property Positions von alleine.

Positions und die zugehörige Membervariable sind vom Typ `IList<OrderPosition>`. Das bedeutet, das Nhibernate an dieser Stelle die Liste von Positionen, die zu dieser Bestellung gehören, finden wird.

Das erste Attribut „Bag“ definiert, dass es sich hierbei um eine Collection handelt. Man definiert den `CascadeStyle` und, dass es sich um ein inverses Mapping handelt. Das bedeutet, dass wir die Beziehung von `OrderPosition` aus zu `Order` definieren. Man beachte: Name haben wir weggelassen, da Name auch hier, wie bei den meisten Attributen ein optionaler Parameter ist. Wir hätten auch `Name="Positions"` schreiben können, aber dies erkennt Nhibernate auch automatisch, so dass dies nicht nötig ist.

Als nächstes benennen wir die Spalte in der Tabelle `OrderPosition`, die den Foreign Key auf die `Order` Tabelle hält. Das ist in diesem Fall die Spalte „OrderID“.

Zuletzt sagen wir Nhibernate mit dem „OneToMany“-Attribut, dass wir auf der „One“ Seite des Mappings stehen und um welchen „ClassType“ es sich bei der gegenüberliegenden Klasse handelt, die die andere Seite des Mappings definiert. In unserem Fall ist das die Klasse `OrderPosition`, die im nächsten Codeblock abgebildet ist.

```

[NHibernate.Mapping.Attributes.Class(0)]
public class OrderPosition
{
    private System.Guid m_ID;
    private Order m_Order;

    [NHibernate.Mapping.Attributes.Id(0,
        Column = "ID",
        Name = "ID",
        TypeType = typeof(System.Guid),
        UnsavedValue = "(00000000-0000-0000-0000-000000000000)"
    )]
    [NHibernate.Mapping.Attributes.Generator(1, Class = "guid.comb")]
    public virtual System.Guid ID
    {
        get { return m_ID; }
        set { m_ID = value; }
    }

    [NHibernate.Mapping.Attributes.ManyToOne(0,
        ClassType = typeof(Order),
        Column = "OrderID",
        ForeignKey = "FK_Order_OrderPosition",
        Name = "Order",
        NotNull = true
    )]
    public virtual Order Order
    {
        get { return m_Order; }
        set { m_Order = value; }
    }

    // ...weitere Felder ...
}

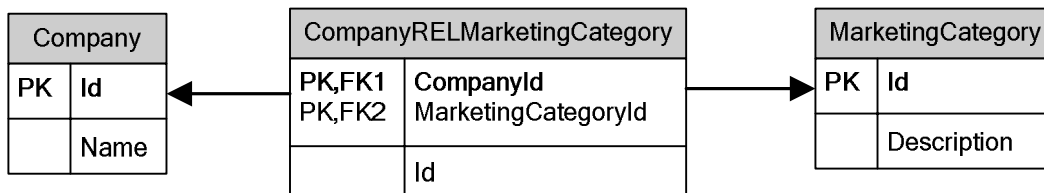
```

8. Many To Many Beziehungen

8.1. Bedeutung

In Many To Many Beziehungen verweist mindestens ein Element in einer Tabelle auf mehrere Elemente in einer zweiten Tabelle. Am einfachsten lässt sich auch das an einem konkreten Beispiel erklären: Eine Firma soll mittels so genannter Marketingkategorien klassifiziert werden. Jeder Kunde kann beliebig vielen Marketingkategorien zugeordnet werden. Dies wird durch eine Many To Many Beziehung abgebildet.

8.2. Beispieldiagramm



In dieser Grafik wird die Many To Many Beziehung zwischen Ccompany und MarketingCategory definiert. Nhibernate soll uns nun das Mapping über die Relationstabelle

CompanyRELMarketingCategory abnehmen.

8.3. Mapping Attribute

Wir möchten selbst nur die Objekte Company und MarketingCategory anlegen. Die Zwischentabelle soll von Nhibernate automatisch erzeugt werden. Dafür definieren wir die Klasse Company wie folgt:

```
[NHibernate.Mapping.Attributes.Class(0)]
public class Company
{
    private Guid m_ID;
    private String m_Name;
    private IList<MarketingCategory> m_MarketingCategories = new
List<MarketingCategory>();

    [NHibernate.Mapping.Attributes.Id(0,
        Column = "ID",
        Name = "ID",
        TypeType = typeof(Guid),
        UnsavedValue = "(00000000-0000-0000-0000-000000000000)"
    )]
    [NHibernate.Mapping.Attributes.Generator(1, Class = "guid.comb")]
    public virtual Guid ID
    {
        get { return m_ID; }
        set { m_ID = value; }
    }

    [NHibernate.Mapping.Attributes.Property(0,
        Column = "Name",
        Name = "Name",
        TypeType = typeof(string),
        Length = 50,
        NotNull = false
    )]
    public virtual string Name
    {
        get { return m_Name; }
        set { m_Name = value; }
    }

    [NHibernate.Mapping.Attributes.Bag(0,
        Cascade = NHibernate.Mapping.Attributes.CascadeStyle.All,
        Lazy = true,
        Table = "CompanyRELMarketingCategories")]
    [NHibernate.Mapping.Attributes.Key(1,
        Column = "CompanyID"
    )]
    [NHibernate.Mapping.Attributes.ManyToMany(2,
        ClassType = typeof(MarketingCategory),
        Column = "MarketingCategoryID",
        ForeignKey = "FK_CompanyRELMarketingCategory_Company"
    )]
    public virtual IList<MarketingCategory> MarketingCategories
    {
        get { return m_MarketingCategories; }
        set { m_MarketingCategories = value; }
    }
}
```

Auch hier gehe ich nur auf die Besonderheit des One To One Mappings ein. Zunächst definieren wir das Bag-Attribut. Wichtig sind hier die Parameter:

Lazy: „true“, damit die abhängigen Objekte per Lazy Loading nachgeladen werden.

Table: Gibt der automatisch erzeugten Relationstabelle einen aussagekräftigen Namen.

Das Key-Attribut gibt den Namen der Spalte in der Relationstabelle an, der den Primary Key dieser Klasse repräsentiert. In diesem Fall wollen wir diese Spalte „CompanyID“ benennen.

Zuletzt geben wir das Attribut ManyToMany an, dass als ClassType den Typen der Klasse benötigt, die auf der anderen Seite der Beziehung steht. In diesem Fall ist das die Klasse MarketingCategory, die anschließend noch als Codeblock gezeigt wird.

Der Parameter Column bezeichnet die Spalte in der Relationstabelle, die den Primary Key dieser gegenüberliegenden Klasse enthält. Der ForeignKey-Parameter gibt dem dazugehörigen Foreign Key, den NHibernate in der Relationstabelle erzeugen soll, einen aussagekräftigen Namen.

Der nächste Codeblock, zeigt das Mapping für die Tabelle MarketingCategory. Dieses Mapping erfolgt genau „spiegelverkehrt“ zu dem in der Klasse Company und ist daher selbsterklärend. Es ist nur der wichtige Code-Teil des ManyToMany-Mappings abgebildet. Den kompletten Code finden Sie in der Datei MarketingCategory.cs.

```
private IList<Company> m_Companies = new List<Company>();

[NHibernate.Mapping.Attributes.Bag(0,
    Cascade = NHibernate.Mapping.Attributes.CascadeStyle.All,
    Lazy = true,
    Table = "CompanyRELMarketingCategories")]
[NHibernate.Mapping.Attributes.Key(1,
    Column = "MarketingCategoryID"
)]
[NHibernate.Mapping.Attributes.ManyToMany(2,
    ClassType = typeof(Company),
    Column = "CompanyID",
    ForeignKey = "FK_CompanyRELMarketingCategory_MarketingCategory"
)]
public virtual IList<Company> Companies
{
    get { return m_Companies; }
    set { m_Companies = value; }
}
```