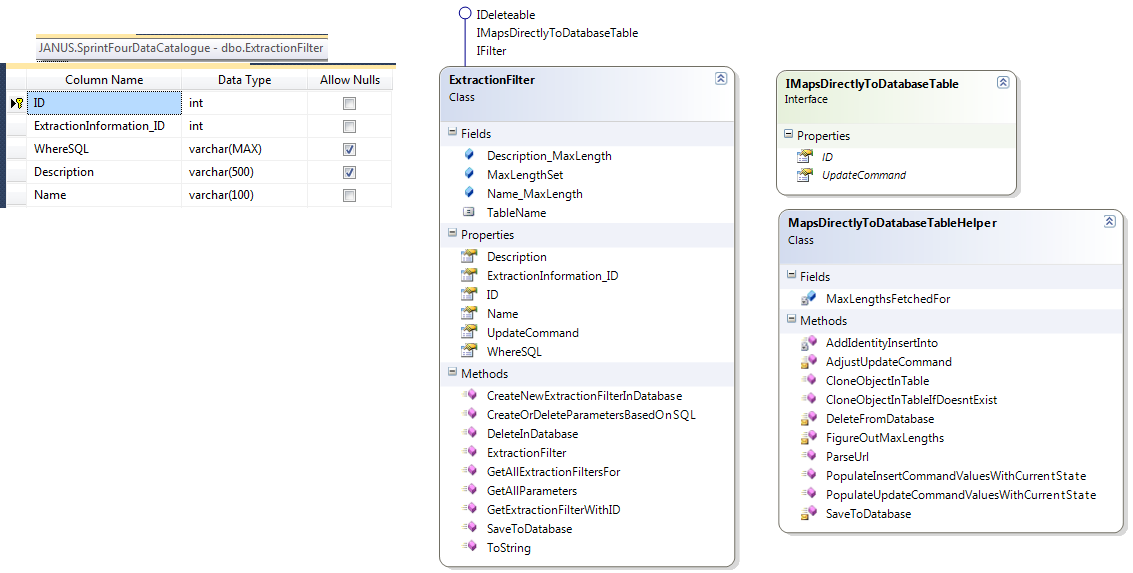
IMapsDirectlyToDatabaseTable

# Purpose

This design pattern allows us to use rapidly add new Columns to Microsoft SQL Database tables and update our classes to support these new columns with minimal effort. It is also designed to reduce the amount of explicit SQL required to create/save/update entities that exist in both Memory (as C# classes) and in the Database (as a form of serialization).

# Diagram

The diagram below shows both the interface (IMapsDirectlyToDatabaseTable) and the Helper class. It also shows an example of an object which implements the pattern (ExtractionFilter).



# Rules

Any class that implements IMapsDirectlyToDatabaseTable:

1. Must have the same Class name as a database table (this allows c# Reflection to be used to determine where to run queries)
2. Must have a single Property for each Column in the database table (so that they can be populated via Reflection). The Property must have the same name as the column in the database.
3. May have a single static field of type int for each variable length Column in the database (e.g. varchar) with the name of the Column followed by “\_MaxLength”. If this is included then you can use the Helper method FigureOutMaxLengths to populate these static fields.
4. Must have an ID property which matches an identity (1,1) column in the database table
5. Should have a private constructor which initializes the object when passed an appropriate SqlDataReader (row)
6. Should have a public static method CreateNewXInDatabase which inserts a new record into the database and returns the ID of the new record (only!)
7. Should have a public static method GetXWithID which:
   1. Performs a Select \* e.g. 
   2. Uses a SqlCommandBuilder to create an UPDATE command based on this SELECT \* query.
   3. Calls the private constructor passing the data row returned from the SELECT
8. The UPDATE command from the SqlCommandBuilder should be used to fulfil the Interface property ‘UpdateCommand’. Make sure to call AdjustUpdateCommand to clean out extra rubbish that is put in by SqlCommandBuilder

# Advantages:

## Add new columns

Assuming all these rules are met then you can write your code and the only changes needed to add a new property to your object are:

1. Add column to database table
2. Add property to object
3. Add setter for the property in the private constructor
4. Optionally add a static \_MaxLength field for the property (e.g. varchar)

That is all.

## Save/Clone

You can save the state of your object into the database at any time by calling SaveToDatabase on MapsDirectlyToDatabaseTableHelper

You can clone objects between databases using CloneObjectInTable on MapsDirectlyToDatabaseTableHelper note this is an **exact** clone including ID which means the record cannot exist yet in the target database.