# Visual LINQ Query Builder

## Description

# Visual LINQ Query Builder is a Visual Studio 2008 designer that helps you building Linq to Sql queries visually. It is functionally the same experience you had using the Microsoft Access query builder for example but in the Linq domain. The entire UI of this add-in is using Windows Presentation Foundation. The goal of this tool is to help you getting more familiar with the Linq syntax. It's also a project that could help you to see how to start a Visual Studio 2008 add-in using Windows Presentation Foundation.

# Using the query bag and the query designer

## Using the query bag

The *query bag* is the main interface of the query designer. It enables you to manage a list of queries in your project. From the query bag, you can do the following:

* Create a new query.

Click the **Create new query** button to go to the query designer and create the query.

* Edit a query.

Click the **Edit query** button to go to the query designer and edit the query.

* Delete a query.
* Group queries by type, the two types being VLinq-generated queries and custom-written queries. See "Custom queries" below.
* Select a query.

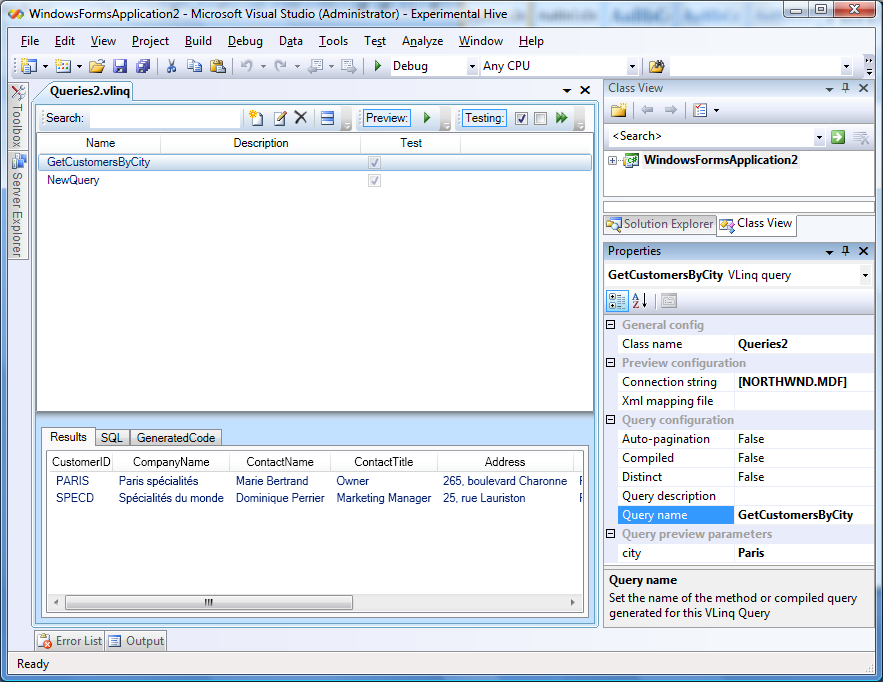
The selected query is used for Edit, Delete, and Preview actions. You can also edit the selected query's properties, including the name and the description, in the Properties window.

* Preview a query.

First you have to expand the Preview toolbar by clicking the Preview button. The preview pane appears at the bottom of the designer. You can then click the **Run** button. The designer runs the query according to the provider configuration. (See "Preview and test requirements" below.) Now the preview pane provides three tabs:

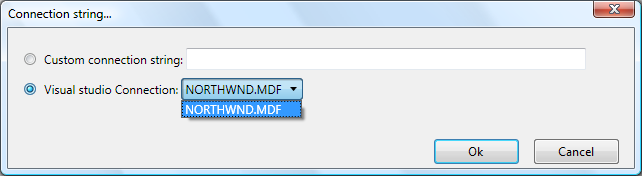
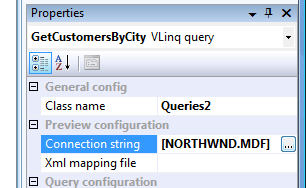
* **Results**. The result of the query displayed in a data grid.
* **SQL**. The generated SQL string.
* **Generated code**. The generated code (C# and Visual Basic).
* Run testing.

First you have to expand the Testing toolbar by clicking the Testing button. A Test column appears in the list of queries. You can use this column to select or clear the selection of each query for testing. The toolbar offers **Select All** and **Unselect All** shortcuts. You can then run the test from the **Testing** toolbar. This will run all the selected queries against the database. The goal is not to preview the results, but to be sure that the queries are running correctly. Each test result (**Ok** or **Failed**) appears in the Test column. "Failed" appears in red and you can click it to see the error details.



## Preview and test requirements

To enable the running of previews or tests, you will have to configure the query bag to do this. Except for running previews and tests, the designer is not related to any specific database. The designer only generates LINQ to SQL code. For previews and tests, you have to configure the query bag’s connection string. You can find the connection string, in either query bag or query designer mode, in the Properties window. A dialog box helps you to either provide a connection string manually, by using a text box, or select an existing Visual Studio connection string like you can do in Server Explorer.



Some queries have parameters. The parameters of the currently selected query dynamically appear in the Properties window. You just have to provide your test values. All of these parameters are saved in the VLinq file, so you can retrieve them at any time when you go back to preview mode. These parameters are also used when you run tests; they are considered as your test values.

## Custom queries

You can manage all VLinq-generated queries in the query bag. Having all your project queries in the same place helps you to organize them. Because you might write some queries manually, after you become more familiar with LINQ or when you write queries that are too complex for VLinq to generate, the query bag enables you to display and test them in the same environment.

VLinq generates a static class that is partial. (See the **Class name** attribute in the Properties window.)

You can extend this partial class manually to add methods that meet the following guidelines:

* The method must be static.
* The method must have a first parameter of type **DataContext**.
* The method must return an **IQueryable<T>**.
* The method must have the **Browsable(true)** attribute.

The following code shows how to entend a partial class:

public static partial class NorthwindQueries

{

[Browsable(true)]

public static IQueryable<Customer> GetAllCustomersByCity(DataContext db, string city)

{

return

from c in db.GetTable<Customer>()

where c.City == city

select c;

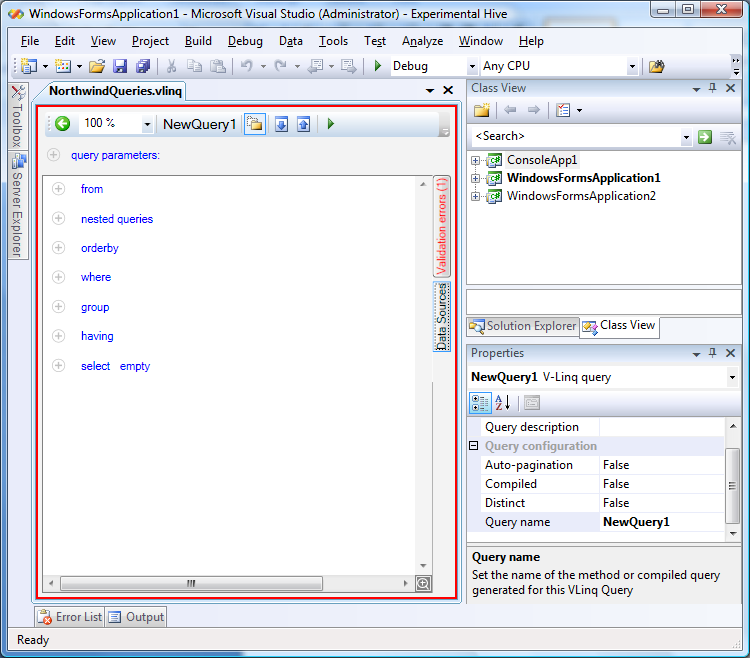
}

…

After the project is compiled, the query designer will automatically display the methods. Of course, you won’t be able to edit or delete the custom queries, but you will have a global view of all your project queries and you will be able to run previews and include them in tests.

## Using the query designer

The query designer enables you to visually build and test a query. The following illustration shows what the query designer looks like after you create a query.



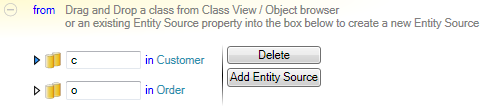
Following are the features of the query designer:

* The Properties window enables you to edit values and options.
* You can name your query to give it a unique name within the set of queries.
* You can add a description to the query.
* From the query list, you can search for text against names and descriptions.
* **Auto-pagination** adds two parameters to the generated method: page number and page size. This property adds functionalities to add a query and skip to the query.
* **Compiled** generates compiled queries instead of methods.
* The toolbar in the query designer allows you to do the following:
* Go back to the query list.
* Adjust the text size in the query list.
* See the name of the query.
* Make all the features visible. By default, only the non-empty features are visible, except for from, where, and select, which are always visible. The Show all button enables or disables this feature. This button is comparable to the Show all files button in Solution Explorer.
* Expand all visible features by using the Expand All button.
* Collapse all visible features by using the Collapse All button.

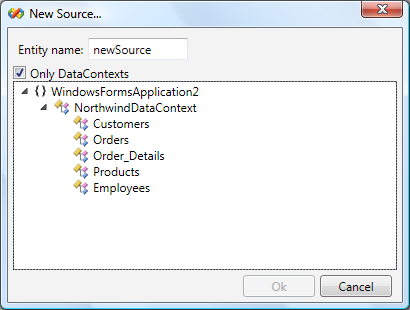
### *From* section feature

The **From** section enables you to do the following:

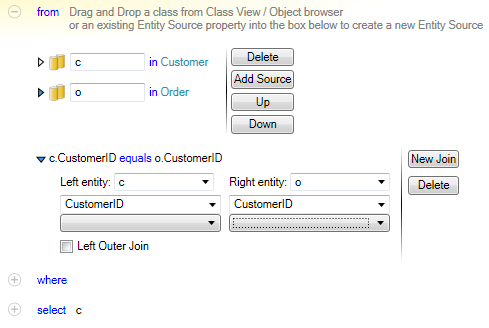
* Manage sources.
* You can drag a class from the class view.



* You can add a source by using the **Add Entity Source** button. A window appears that enables you to edit the name and the type. Select **Only DataContexts** if you want to see only DataContexts rather than all available types.



* You can drag a source from an existing source and child relationship. This enables you to create sources like the following: from o in c.Orders
* After you create a source, you can rename it by using the associated text box.
* The **Delete** button enables you to delete the selected data sources.
* Manage joins.
* You can create joins by using the **New join** button.
* You have to enter two entities—right and left. The UI dynamically adds a column-to-column association that you can edit. After an association is completed, the UI creates a new one. To remove an association, you must clear both the left column and the right column; the association will disappear automatically.

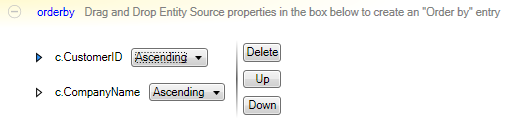


* A check box enables you to define whether you want the join to be inner or outer.

### *OrderBy* section features

The **OrderBy** section enables you to do the following:

* Add columns.
* You can add a column by dragging one from any data source property.
* You can also add a column by using the context menu of any data source property.

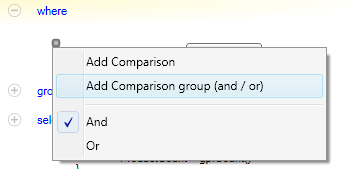


* Use buttons to change the ordering direction (ascending or descending), delete items, and sort items.

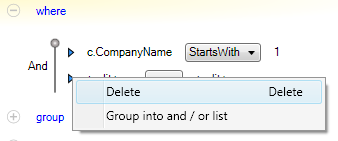
### *Where* and *Having* section features

The **Where** and **Having** sections enable you to do the following:

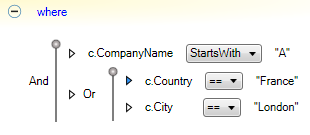
* Add constraints.
* You can add constraints by using the context menu on the list.
* The constraint must define both left and right operands and an operator.
* By default, the constraints are grouped with a logical **And** operator. The context menu on the list enables you to choose between **And** and **Or**.



* Group constraints.
* You can group constraints by using multiple selection (CTRL+click), right-clicking the group, and then clicking **Group into and / or list**.



* + You can also ungroup a constraint to take it back to the parent list.



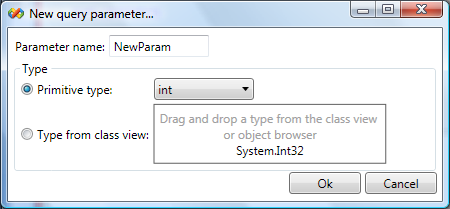
There are a few differences between the **Where** section and the **Having** section:

* **Having** can appear only if a **Group by** is defined.
* **Having** occurs after the **Group by** operation; **Where** occurs before.
* Therefore, in the **Having** section, you can access the result of the **Group by** operation (but not the original entity sources), and in the **Where** section, you can access the original entity sources (but not the result of the **Group by** operation).

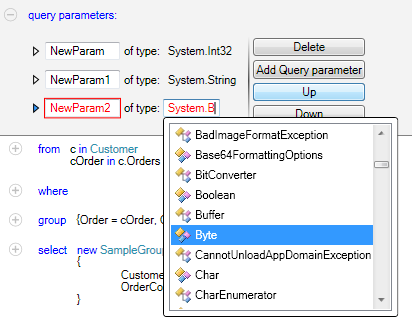
### *Query parameters* section features

A query can be parameterized to enable client code to modify the execution of the request. These parameters can be used in the **Where**, **Having**, and **Select** sections of the query and of its nested queries (see "Nested queries" below) as data sources. They appear in IntelliSense.

To add a parameter, click **Add Query parameter** and then either select a predefined type or drag a type from class view or the Object Browser.



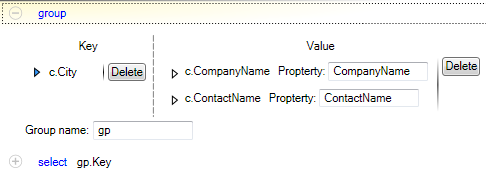
You can then rename or reset the type of a parameter:



### *Group by* section features

The query designer enables you to group lists by defining a key (simple or composite) and the group projection (simple or composite).

* You can create a projection:
* By dragging an item from an entity source or property into the **Key** or **Value** drop zone.
* By clicking **Add to group key** or **Add to group value** on the context menu of an entity source or property.
* You can modify a group key or value by clicking the **Delete** button to the right of an operand in the **Key** or **Value** section.
* You can create a composite key or value by dropping other entity sources or properties.
* You can rename the generated property names of each operand of a composite key or value.



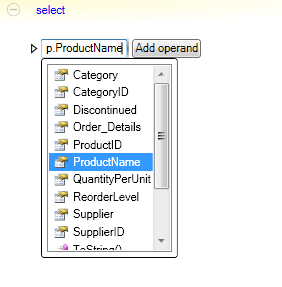
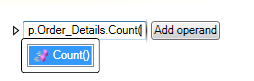
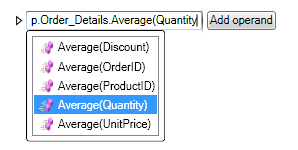
### *Select* section features

The last section of the query is the **Select** section. It defines what will be projected as a result of the query.

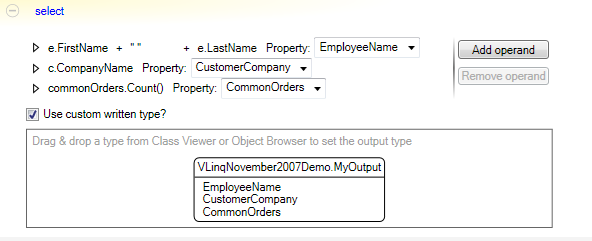
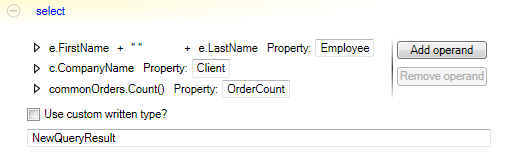
You can specify either one operand or multiple operands.

### What is an operand?

The nature of an operand is the same as that for the **Where** and **Having** sections: it can be an entity source, a parameter, an entity source property, an aggregate method, a concatenation, or a numeric operation. Here are some visual examples:

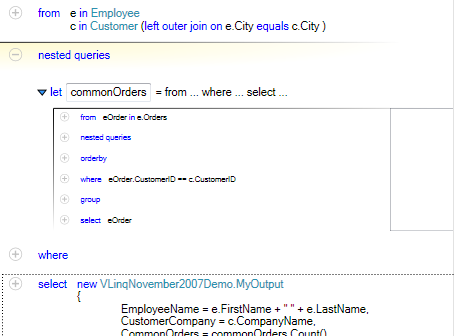


If you have more than one operand in your projection, you have to define an output type because methods cannot return anonymous types. By default, VLinq generates the output type for you, but you can also define your own output type by dragging a type from class view or the Object Browser.



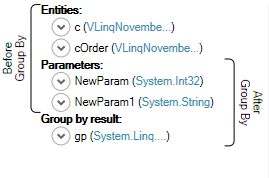
### *Nested queries* section

You can define nested queries to get a subset of data and use the result in the other sections of the query. The interface used to create a nested query is the same as that of the main query editor, but it is smaller. (You can zoom in to have a better view of the nested query.)



### Data Sources summary

On the right side of the interface is a **Data sources** button. You can use it to obtain a view of the available data sources and to perform drag-and-drop operations without having to scroll between the various sections.



### Validation summary

VLinq validates the query and highlights errors at design time. It provides a summary with error messages and a preview of any invalid query elements in a pop-up window:

