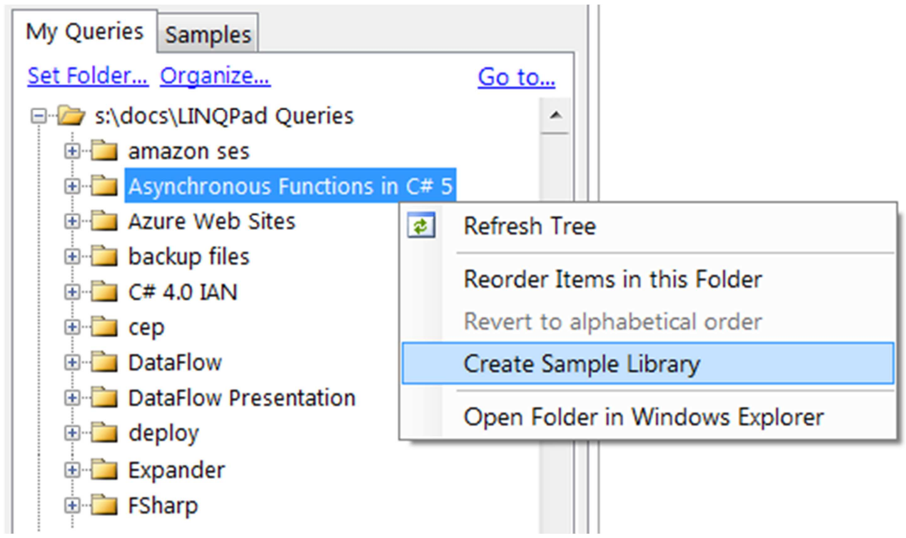
LINQPAD SAMPLES INTEGRATION GUIDE

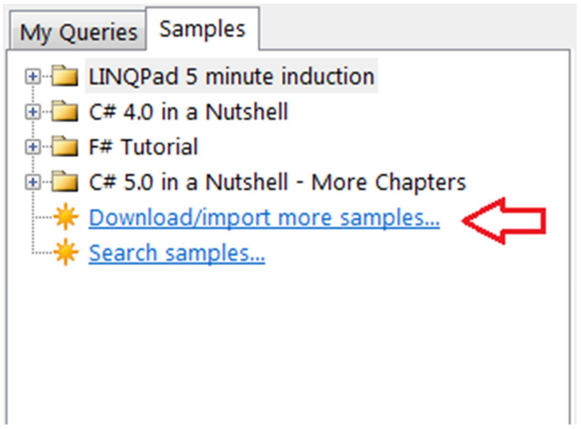
Joseph Albahari
Version 1.1 Last updated 2013-9-11

QUICK START

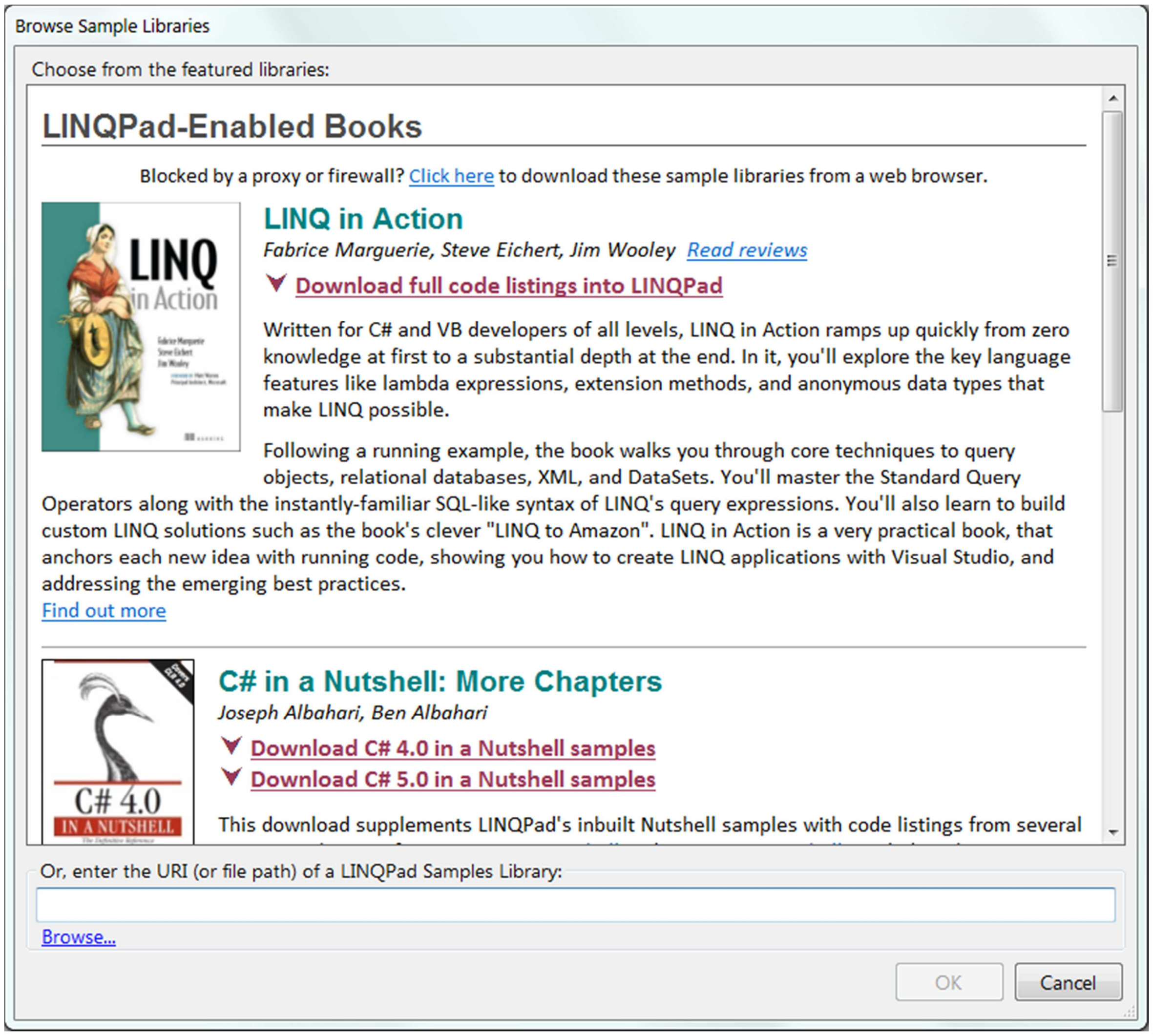
Creating a LINQPad sample library is simply a matter of adding .LINQ files to a .zip file (along with any dependencies). You can do this from within LINQPad by right-clicking a folder in **My Queries** and choosing the option **Create Sample Library**:



The zip file that this creates can then be imported into LINQPad's samples tree by clicking **Download/import more samples**.


My Queries
Samples
LINQPad 5 minute induction
C# 4.0 in a Nutshell
F# Tutorial
C# 5.0 in a Nutshell - More Chapters
Download/import more samples...
Search samples...

Here's the window that appears:



LINQPad-Enabled Books

Blocked by a proxy or firewall? Click here to download these sample libraries from a web browser.

LINQ in Action

Fabrice Marguerie, Steve Eichert, Jim Wooley Read reviews

Download full code listings into LINQPad

Written for C# and VB developers of all levels, LINQ in Action ramps up quickly from zero knowledge at first to a substantial depth at the end. In it, you'll explore the key language features like lambda expressions, extension methods, and anonymous data types that make LINQ possible.

Following a running example, the book walks you through core techniques to query objects, relational databases, XML, and DataSets. You'll master the Standard Query Operators along with the instantly-familiar SQL-like syntax of LINQ's query expressions. You'll also learn to build custom LINQ solutions such as the book's clever "LINQ to Amazon". LINQ in Action is a very practical book, that anchors each new idea with running code, showing you how to create LINQ applications with Visual Studio, and addressing the emerging best practices.

Find out more

C# in a Nutshell: More Chapters

Joseph Albahari, Ben Albahari

Download C# 4.0 in a Nutshell samples

Download C# 5.0 in a Nutshell samples

This download supplements LINQPad's inbuilt Nutshell samples with code listings from several

Or, enter the URI (or file path) of a LINQPad Samples Library:
Browse...
OK
Cancel

If your library is popular, you can ask for it to be featured in LINQPad's online samples gallery along with the sample libraries for books such as *LINQ in Action* and *C# in a Nutshell*. Otherwise, the end user can import the library by typing its URI into the textbox or browsing to the .zip file.

WHY CREATE A LINQPAD SAMPLES LIBRARY?

A LINQPad samples library provides a great way for users to interactively work through a series of examples, without the complexity and mess of Visual Studio solutions. Your samples are neatly organized into a tree, and users can move from one sample to the next with a single click. Users also get to enjoy LINQPad's other benefits:

- Rich output formatting with Dump()—including expansion of nested collections and object graphs
- Support for C#, VB, F#, OData, LINQ-to-SQL and Entity Framework
- Automatic translation of (IQueryable-based) query expressions into fluent syntax, and C#/VB into IL

And LINQPad is not just for LINQ queries: it will execute any C#, VB or F# expression, statement or program—with or without a data context. Think of it as general-purpose code scratchpad. To illustrate, here's a query from the **C# 5.0 in a Nutshell** library that demonstrates the use of task combinators and asynchronous functions:

```
async void Main()
{
        Task<int> winningTask = await Task.WhenAny (Delay1(), Delay2(), Delay3());
        Console.WriteLine ("Done");
        Console.WriteLine (winningTask.Result);   // 1
}
```

```
        async Task<int> Delay1() { await Task.Delay (1000); return 1; }
        async Task<int> Delay2() { await Task.Delay (2000); return 2; }
        async Task<int> Delay3() { await Task.Delay (3000); return 3; }
```

Samples can reference custom DLLs and even NuGet packages that download automatically when the query is first opened.

If your library is featured in the LINQPad Samples Gallery, you retain copyright to all your samples.

WILL LINQPAD CONTINUE TO BE FREE?

LINQPad is supported by a robust "freemium" model: the standard version is (and will continue to be) free while users pay a small fee for autocompletion.

FRAMEWORK 4.X OR 3.5?

There are two versions of LINQPad:

- **LINQPad 4.x** targets .NET Framework 4.0/4.5
- **LINQPad 2.x** targets .NET Framework 3.5.

LINQPad 4.x works with the compilers for C# 4, C# 5, VB 10, VB 11 and F# 3.

Users can also download a library from the local file system or URI.

FINE-TUNING

CREATING THE ZIP FILE

Use subdirectories to reflect your samples hierarchy. Any level of nesting is OK. For instance:

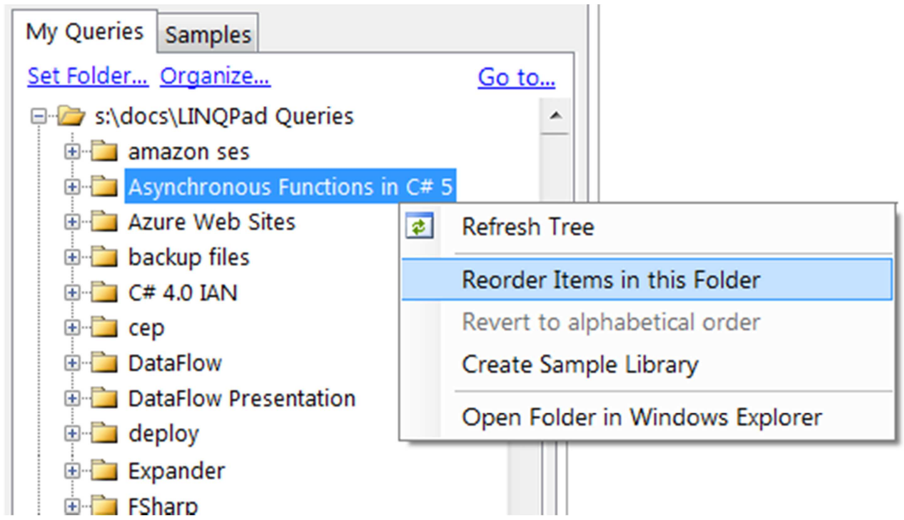
- Introduction
  - Read me first.linq
- Chapter 1
  - Introducing LINQ Queries
    - My First LINQ Query.linq
    - My Second LINQ Query.linq
    - ...
  - What Your Mom Never Told You About Entity Framework
    - ...
- Chapter 2
  - ...

The name of your zip file becomes the name of your samples library.

You can include .linq files that consist purely of comments—this is useful for copyright and introductory messages. If you rely on sample databases, it's a good idea to include a .LINQ file with the query language set to SQL that creates the schema and data. This comes in handy with users for whom the "attach file" option fails (see "Sample Databases and Custom Assemblies").

ORDERING

You'll usually want to have your samples appear in a logical order. To do this, right-click the folder in **My Queries** and choose **Reorder Items in this Folder**:



This brings up a reordering dialog for your queries and subfolders. After rearranging and clicking OK, LINQPad creates a file called *FileOrder.txt.*

Another way to order your queries is to prefix their names with a number, optionally in square brackets:

```
1.1 My First Query          // Number visible
[1.1] My First Query        // Number invisible
```

If you use square brackets, LINQPad strips away the prefix completely when the library is imported. Multi-part numbers (x.y.z) are recognized and you don't need to pad numbers with leading zeros.

OPTIONAL: THE HEADER.XML FILE

You can optionally create a file called **header.xml** at the top level in the folder hierarchy to specify your library name:

```
<SampleLibrary>
    <Name>The Ultimate Guide to LINQ</Name>
</SampleLibrary>
```

Without this file, the library will take on the name of the .zip file when imported.

WHERE DO SAMPLE LIBRARIES END UP?

When you import a sample library, LINQPad copies your ZIP file to the following folder:

`%APPDATA%\LINQPad\Samples\`*`Your Library Name`*`\`

LINQPad extracts the .LINQ files from your sample library directly into memory on demand (it does not extract them to the hard drive). However, any files with an extension other than .LINQ *are* extracted, so you can refer to them in your queries as necessary.

You can get the physical location of a file in your samples library using LINQPad's **Util.GetSampleFilePath** method:

```
string xmlPath = Util.GetSampleFilePath ("Ultimate Guide to LINQ", "SomeData.xml");
```

NUGET REFERENCES

LINQPad sample libraries work great with NuGet references. If a query references a NuGet package, LINQPad will ask the user if they want to download the appropriate package(s) and everything will resolve automatically. **This works even if the user has the free**

edition of LINQPad. **(**What's missing from the free edition is the ability to browse the online gallery of NuGet packages, not the ability to download packages required by an existing query.)

CUSTOM ASSEMBLY REFERENCES

You can include custom assemblies in your .zip file, and reference them in your queries. When LINQPad saves a query, it encodes references with both absolute and relative path information, so it still works if moved to another computer.

LINQPad tokenizes system folders, so it's also safe to reference assemblies from the %APPDATA%\ folder.

WORKING WITH DATABASES

If you include .MDF or other database files in your sample library, it's safest to set up your connection so that it points directly to the databases's final destination:

`%APPDATA%\LINQPad\Samples\`*`Your Library Name`*`\`

For queries that use a connection, LINQPad writes the connection details to the .LINQ file. Hence, the connection appears automatically in the Schema Explorer tree when the user clicks on the query.

A limitation of SQL Server is that you can't include a connection that works for everyone: the connection properties vary depending on whether the user is running SQL Express, LocalDB, or a full edition of SQL Server. This means the user may need to edit the details of the connection that appears to make it work for their system, by right-clicking the connection and choosing **Properties**.

> *When editing the connection, the user can check '**Remember this connection**' before clicking **OK**. This makes the edit apply to all other queries in your sample library that use that connection.*

MAINTAINING A SINGLE IDENTITY PER CONNECTION

You'll want queries that refer to the same connection have the same *connection ID*. This is so that if a user edits a connection as described above (for example, to change **.\SQLEXPRESS** to **(local)** or to uncheck **User Instance**), the change applies to other queries after checking **Remember this connection** and clicking **OK**.

You can view the connection ID for a query by examining its .linq file:

```
<Query Kind="Expression">
  <Connection>
    <ID>03f8fbdc-557b-45af-9da2-50b6c711c554</ID>
    <Server>.\SQLEXPRESS</Server>
    ...
  </Connection>
</Query>

from c in Customers
where c.FirstName.StartsWith ("A")
select c
```

The most common cause of multiple identities is creating queries on different machines. To fix, simply copy and paste the connection ID from one .LINQ file to another.

FORMAT OF A .LINQ FILE

You can edit .linq files directly. A .linq file is just an XML header followed by the query:

```
<Query Kind="Expression">
  <Namespace>System.Net.NetworkInformation</Namespace>
</Query>

from site in new[]
{
	"www.albahari.com",
	"www.linqpad.net",
	"www.oreilly.com",
	"www.takeonit.com"
}
.AsParallel()
let p = new Ping().Send (site)
select new
{
	site,
	Result = p.Status,
	Time = p.RoundtripTime
}
```