ASYNC LINQ

with the Reactive Extensions for .NET (Rx)

Orion Edwards @ CodeCamp 2010

LINQ Recap

C#2 - Unified Interface for Collections

```
IEnumerable<T>
```

C#2 - Lazy Evaluation

```
public IEnumerable<Company> LoadCompanies()
{
    foreach(var c in Database.Table.Row)
        yield return new Company(c);
}
```

C#3 - LINQ Standard Query Operators

```
IEnumerable<Company> companies;

var managers = companies
    .Where(company => company.IsNotBankrupt)
    .SelectMany(company => company.Employees)
    .Where(employee => employee.IsManager)
    .OrderByDescending(employee => employee.Salary);
```

C#3 - Query Syntax

```
IEnumerable<Company> companies;

var managers = from company in companies
    where company.IsNotBankrupt
    from employee in company.Employees
    where employee.IsManager
    orderby employee.Salary descending
    select employee;
```

Asynchronous Programming Recap

Fundamentals

- Something is going to happen, but we can't be sure when
- Can't block and wait because we need to do other things
- Use asynchronous API's which run a callback when the thing happens

Problem: User input

```
while(true)
{
      if(MouseMoved())
      {
            // ...
      }
}
```

Our program will only ever be able to process MouseMove

Solution: Events

```
MouseMove += (object sender, MouseEventArgs e) => {
    // ...
};
```

Callback will run when the event happens

Problem: Expensive Calculations

Parallel task library can speed this up
...but it's still going to take a while

```
if(flag)
{
    digit = CalculateMillionthDigitOfPi();
    DisplayDigit();
}
```

Our program will stall until the calculation is finished

Solution: Run in the background

```
var worker = new BackgroundWorker();
worker.DoWork += (s, e) => digit = CalculateMillionthDigitOfPi();
worker.RunWorkerCompleted += (s, e) => DisplayDigit();
worker.RunWorkerAsync();
```

Callback will run when the calculation finishes

Problem: Blocking I/O

```
var pictures = DataAccess.LoadPictures(person.Id);
foreach(var tmp in pictures)
{
    var picture = tmp;
    viewModel.Pictures.Add(picture);
    var bytes = DataAccess.LoadImage(person.Id, picture.Id);
    picture.Image = bytes;
}
```

Our program will stall while loading

Problem: Blocking I/O

```
var pictures = DataAccess.LoadPictures(person.Id);
foreach(var tmp in pictures)
{
    var picture = tmp;
    viewModel.Pictures.Add(picture);
    var bytes = DataAccess.LoadImage(person.Id, picture.Id);
    picture.Image = bytes;
}
```

Our program will stall while loading

Problem: Blocking I/O

```
var pictures = DataAccess.LoadPictures(person.Id);
foreach(var tmp in pictures)
{
    var picture = tmp;
    viewModel.Pictures.Add(picture);
    var bytes = DataAccess.LoadImage(person.Id, picture.Id);
    picture.Image = bytes;
}
```

Our program will stall while loading

Problem: Blocking I/O

```
var pictures = DataAccess.LoadPictures(person.Id);
foreach(var tmp in pictures)
{
    var picture = tmp;
    viewModel.Pictures.Add(picture);
    var bytes = DataAccess.LoadImage(person.Id, picture.Id);
    picture.Image = bytes;
}
```

Our program will stall while loading

Problem: Blocking I/O

```
var pictures = DataAccess.LoadPictures(person.Id);
foreach(var tmp in pictures)
{
    var picture = tmp;
    viewModel.Pictures.Add(picture);
    var bytes = DataAccess.LoadImage(person.Id, picture.Id);
    picture.Image = bytes;
}
```

Our program will stall while loading

Problem: Blocking I/O

```
var pictures = DataAccess.LoadPictures(person.Id);
foreach(var tmp in pictures)
{
    var picture = tmp;
    viewModel.Pictures.Add(picture);
    var bytes = DataAccess.LoadImage(person.Id, picture.Id);
    picture.Image = bytes;
}
```

Our program will stall while loading

Solution: Nonblocking I/O

```
DataAccess.BeginLoadPictures(person.Id, (asyncResult) => {
    var pictures = DataAccess.EndLoadPictures(asyncResult);
    foreach(var tmp in pictures)
    {
        var picture = tmp;
        viewModel.Pictures.Add(picture);
        DataAccess.BeginLoadImage(person.Id, picture.Id, (ar2) => {
            var bytes = DataAccess.EndLoadImage(ar2);
            picture.Image = bytes;
        }, null);
    }
}, null);
```

Solution: Nonblocking I/O

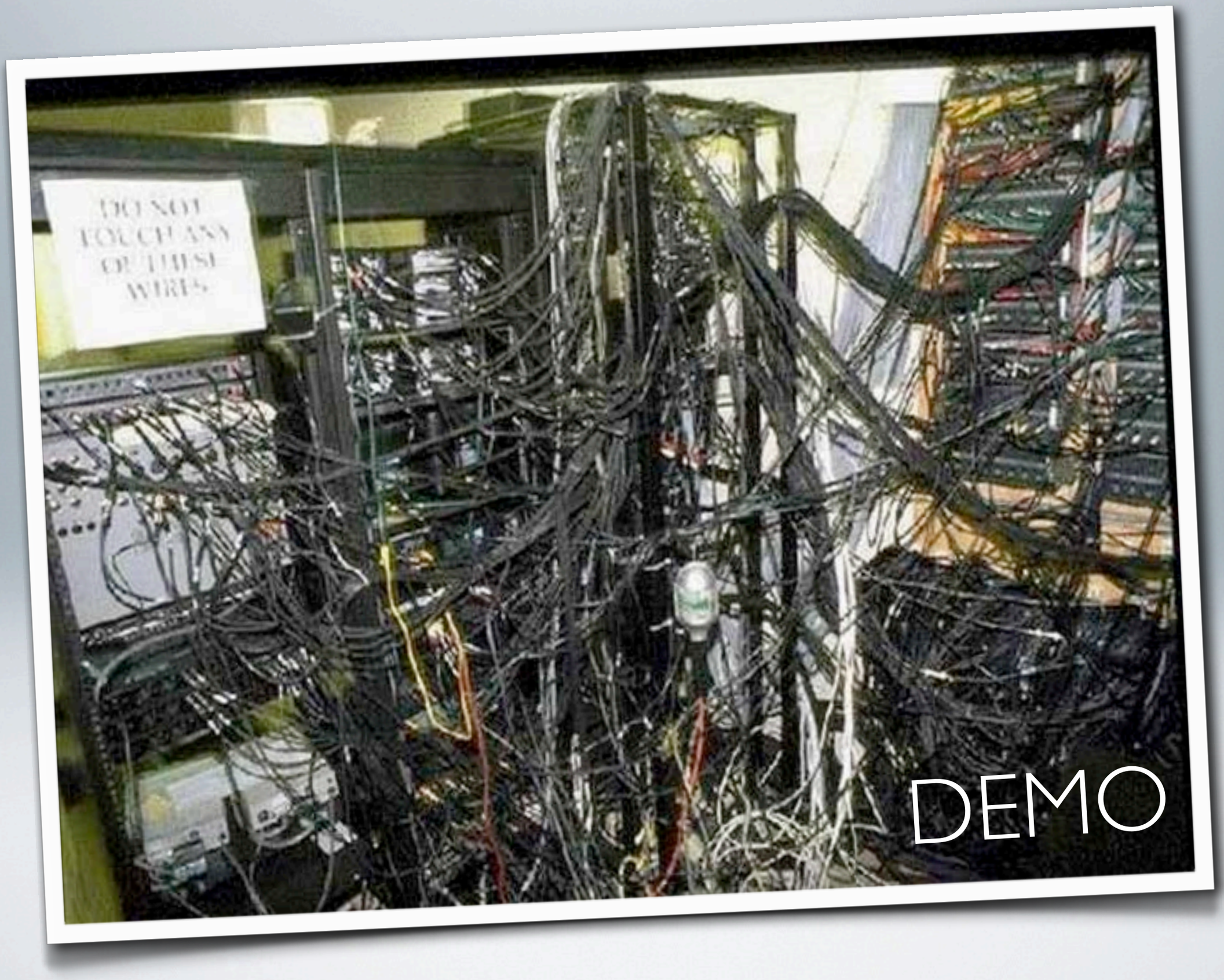
```
DataAccess.BeginLoadPictures(person.Id, (asyncResult) => {
    var pictures = DataAccess.EndLoadPictures(asyncResult);
    foreach(var tmp in pictures)
    {
        var picture = tmp;
        viewModel.Pictures.Add(picture);
        DataAccess.BeginLoadImage(person.Id, picture.Id, (ar2) => {
            var bytes = DataAccess.EndLoadImage(ar2);
            picture.Image = bytes;
        }, null);
    }
}, null);
```

Callback will run when the data is loaded

We can do all this already

We can do all this already

but the API's are hard to use and inflexible


DO NOT
TOUCH ANY
OF THESE
WIRES
DEMO

Rx

Google

reactive extensions

About 753,000 results (0.07 seconds)

Everything
More

The web
Pages from New Zealand

Any time
Past 2 weeks

More search tools

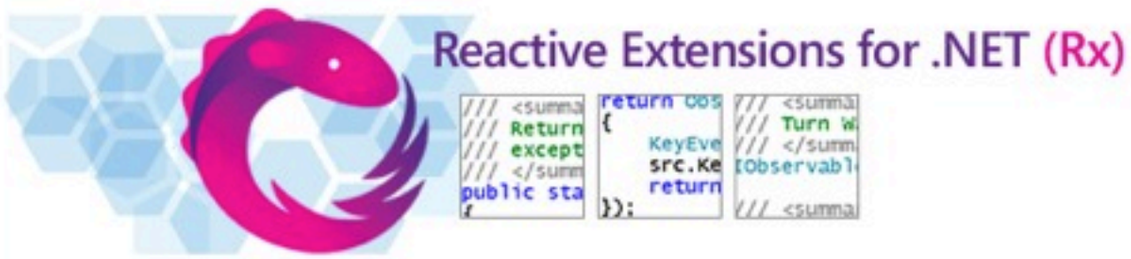
Reactive Extensions for .NET (Rx)
We're pr[illegible]d to announce the availability of **Reactive Extensions** for Javascript. This port brings t[illegible]ower of Reactive programming to JavaScript. ...
msdn.micr[illegible]oft.com/en-us/devlabs/ee794896.aspx - Cached - Similar

Querying the Future With **Reactive Extensions**
26 Mar 2010 ... **Reactive extensions** inverts this model by taking what I would call a "pull" model of events. Using these extensions, you can treat the ...
haacked.com/archive/2010/03/26/enumerating-future.aspx - Cached

A quick look at the **Reactive Extensions** (Rx) for .Net - CodeProject
18 Dec 2009 ... Introduction and some words about what I've been doing lately Hello everyone, first of all, let me tell you that I've been very busy at work ...
www.codeproject.com/.../A-quick-look-at-the-**Reactive-Extensions**-Rx-for-Net.aspx - Cached - Similar

First encounters with **Reactive Extensions** - Jon Skeet: Coding Blog
16 Jan 2010 ... I've been researching **Reactive Extensions** for the last few days, with an eye to writing a short section in chapter 12 of the second edition ...
msmvps.com/.../first-encounters-with-**reactive-extensions**.aspx - Cached - Similar

DevLabs: Reactive Extensions for .NET (Rx)



Rx News

We're proud to announce the availability of Reactive Extensions for Javascript. This port brings the power of Reactive programming to JavaScript. It allows you to use the Rx combinators in JavaScript, and it does this in a download size of less than 7Kb (GZipped). RxJS provides easy-to-use conversions from existing DOM, XmlHttpRequest, and jQuery events to Rx push-collections, allowing users to seamlessly plug Rx into their existing JavaScript-based web sites.

To give RxJs a try, download the installer. The installer comes with documentation and a small animation-based sample. You can provide feedback on RxJS on the regular Rx forum, and of course help the Rx community to convert the 101 Rx samples to JavaScript. For an introduction, watch the Rx for JavaScript video on Channel 9.

About Rx

Rx is a library for composing asynchronous and event-based programs using observable collections.

The "A" in "AJAX" stands for asynchronous, and indeed modern Web-based and Cloud-based applications are fundamentally asynchronous. In fact, Silverlight bans all blocking networking and threading operations. Asynchronous programming is by no means restricted to Web and Cloud scenarios, however. Traditional desktop applications also have to maintain responsiveness in the face of long latency IO operations and other expensive background tasks.

Another common attribute of interactive applications, whether Web/Cloud or client-based, is that they are event-driven. The user interacts with the application via a GUI that receives event streams asynchronously from the mouse, keyboard, and other inputs.

Rx is a superset of the standard LINQ sequence operators that exposes asynchronous and event-based computations as push-based, observable collections via the new .NET 4.0 interfaces IObservable<T> and IObserver<T>. These are the mathematical dual of the familiar IEnumerable<T> and IEnumerator<T> interfaces for pull-based, enumerable collections in the .NET Framework.

The IEnumerable<T> and IEnumerator<T> interfaces allow developers to create reusable abstractions to consume and transform values from a wide range of concrete enumerable collections such as arrays, lists, database tables, and XML documents. Similarly, Rx allows programmers to glue together complex event processing and asynchronous computations

Related

- Rx Team Blog
- B# .NET Blog
- Wes Dyer's Blog
- Jeffrey Van Gogh's Blog
- Danny van Velzen's Blog
- Lambda the Ultimate Blog
- Rx on Channel 9
- Rx EULA

Get Rx

Download Reactive Extensions for .NET (Rx), a library for composing asynchronous and event-based programs using observable collections.

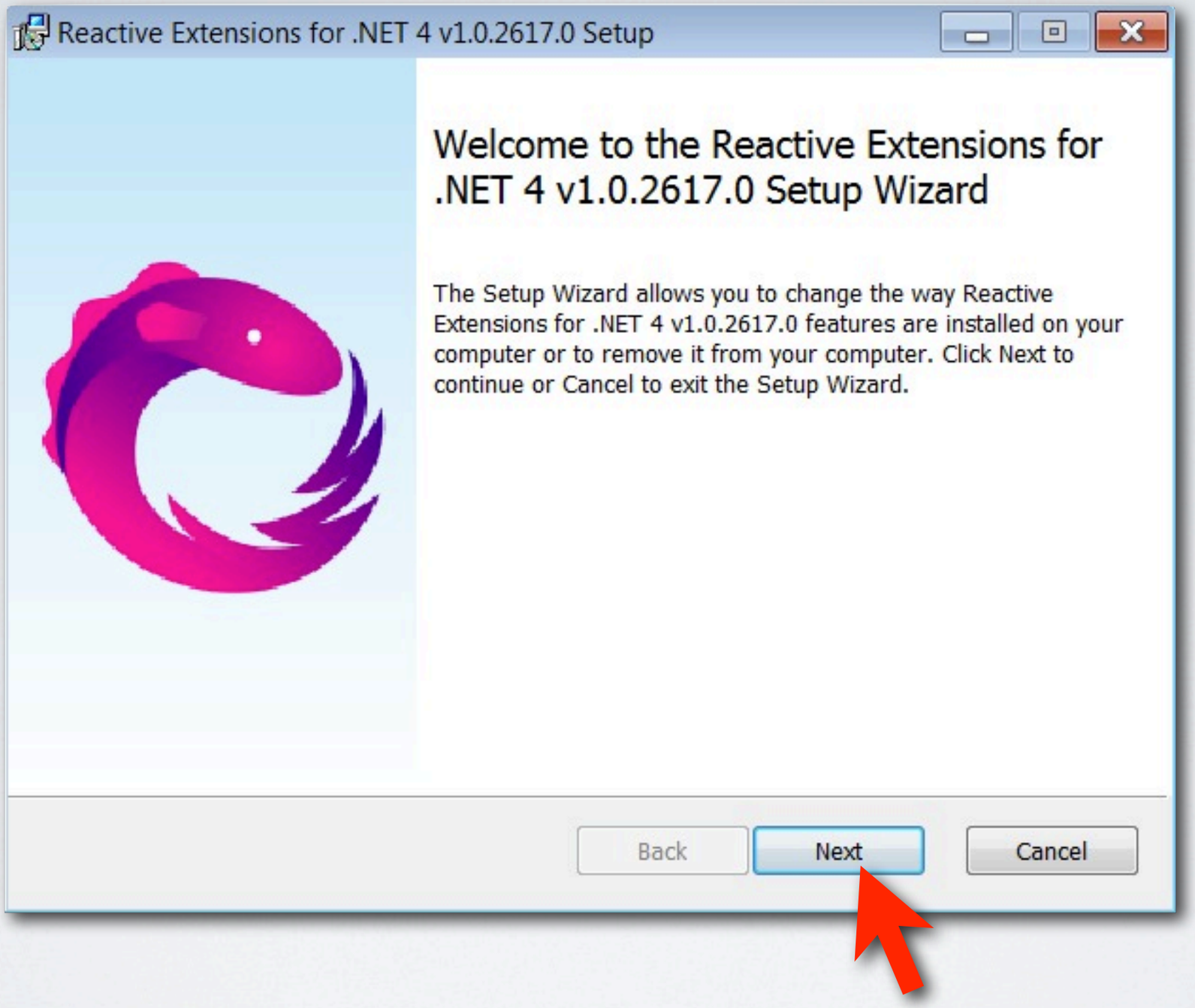
Rx for .NET Framework 3.5 SP1

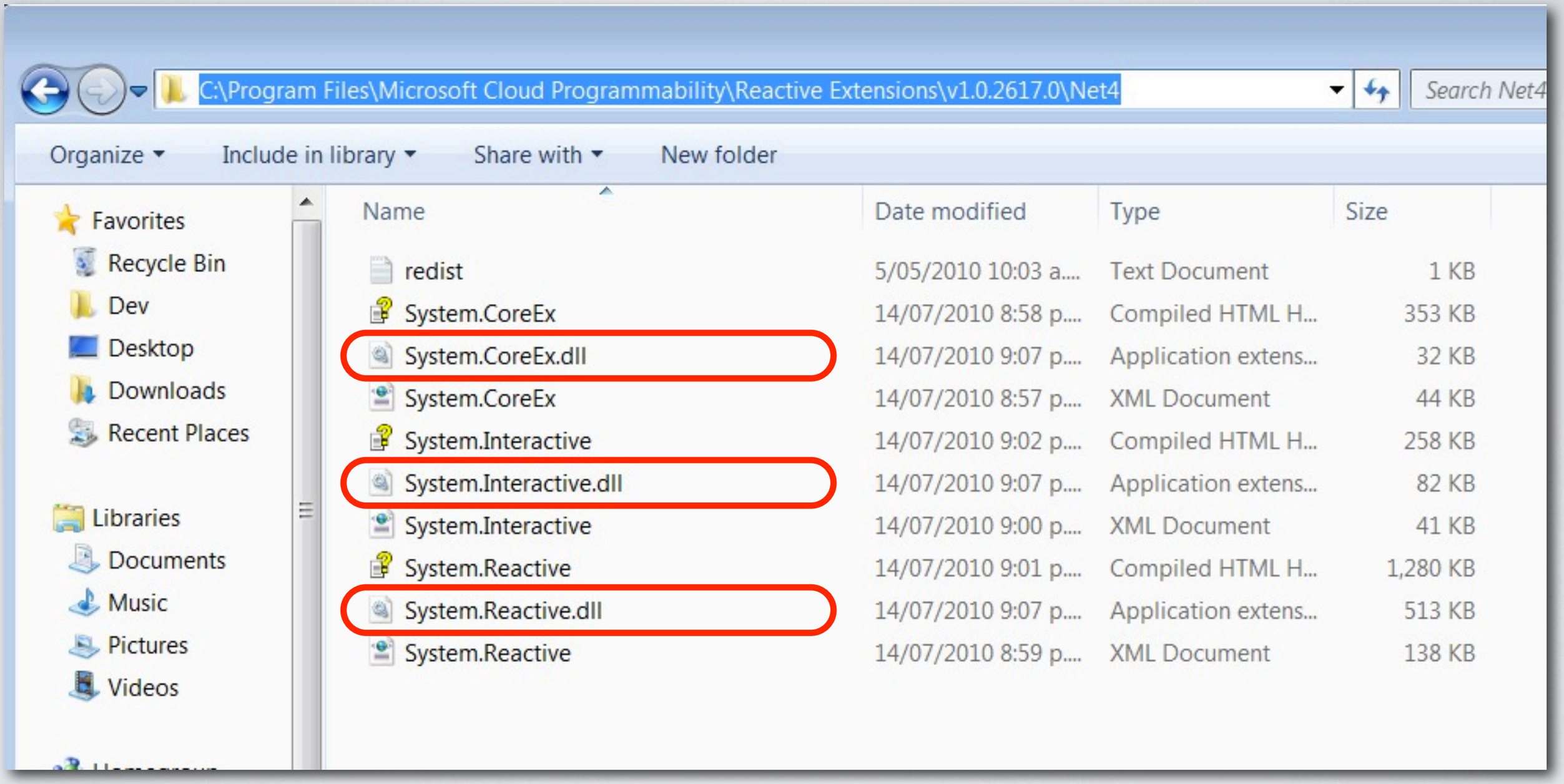
Rx for .NET Framework 4.0

Rx for Silverlight 3

Rx for Silverlight 4

Rx for JavaScript


Reactive Extensions for .NET 4 v1.0.2617.0 Setup
Welcome to the Reactive Extensions for .NET 4 v1.0.2617.0 Setup Wizard
The Setup Wizard allows you to change the way Reactive Extensions for .NET 4 v1.0.2617.0 features are installed on your computer or to remove it from your computer. Click Next to continue or Cancel to exit the Setup Wizard.
Back
Next
Cancel


C:\Program Files\Microsoft Cloud Programmability\Reactive Extensions\v1.0.2617.0\Net4
Search Net4
Organize
Include in library
Share with
New folder
Favorites
Recycle Bin
Dev
Desktop
Downloads
Recent Places
Libraries
Documents
Music
Pictures
Videos
Name
Date modified
Type
Size
redist 5/05/2010 10:03 a.... Text Document 1 KB
System.CoreEx 14/07/2010 8:58 p.... Compiled HTML H... 353 KB
System.CoreEx.dll 14/07/2010 9:07 p.... Application extens... 32 KB
System.CoreEx 14/07/2010 8:57 p.... XML Document 44 KB
System.Interactive 14/07/2010 9:02 p.... Compiled HTML H... 258 KB
System.Interactive.dll 14/07/2010 9:07 p.... Application extens... 82 KB
System.Interactive 14/07/2010 9:00 p.... XML Document 41 KB
System.Reactive 14/07/2010 9:01 p.... Compiled HTML H... 1,280 KB
System.Reactive.dll 14/07/2010 9:07 p.... Application extens... 513 KB
System.Reactive 14/07/2010 8:59 p.... XML Document 138 KB

LINQ Fundamentals

LINQ Fundamentals

- Any collection or stream is just a series of values

LINQ Fundamentals

- Any collection or stream is just a series of values
- Can be lazy loaded - values over time

LINQ Fundamentals

- Any collection or stream is just a series of values
- Can be lazy loaded - values over time
- IEnumerable<T> provides a unified interface to pull values over time from any source

LINQ Fundamentals

- Any collection or stream is just a series of values
- Can be lazy loaded - values over time

- IEnumerable<T> provides a unified interface to pull values over time from any source
- Use this unified interface to write reusable libraries that work with anything (LINQ standard query operators, etc)

Rx Fundamentals

Rx Fundamentals

- Asynchronous operations are *also* a sequence of values over time, but the values are pushed

Rx Fundamentals

- Asynchronous operations are *also* a sequence of values over time, but the values are pushed

- Rx provides IObservable<T>, a unified interface for any source to push values over time

Rx Fundamentals

- Asynchronous operations are *also* a sequence of values over time, but the values are pushed

- Rx provides IObservable<T>, a unified interface for any source to push values over time

- IObservable<T> is the mathematical dual of IEnumerable<T>

NOT (P OR Q) = (NOT P) AND (NOT Q)
NOT (P AND Q) = (NOT P) OR (NOT Q)

Rx Fundamentals

- Asynchronous operations are *also* a sequence of values over time, but the values are pushed

- Rx provides IObservable<T>, a unified interface for any source to push values over time

- IObservable<T> is the mathematical dual of IEnumerable<T>

- Use this unified interface to write LINQ standard query operators (and many others) that work on any push source

IEnumerable for PULL based sequences

```
public interface IEnumerable<T>
{
    IEnumerator<T> GetEnumerator();
}

public interface IEnumerator<T> : IDisposable
{
    T Current { get; }
    bool MoveNext();
    void Dispose();
}
```

IEnumerable for PULL based sequences

```
public interface IEnumerable<T>
{
    IEnumerator<T> GetEnumerator();
}

public interface IEnumerator<T> : IDisposable
{
    T Current { get; }
    bool MoveNext();
    void Dispose();
}
```

1. Get an enumerator out

IEnumerable for PULL based sequences

```
public interface IEnumerable<T>
{
    IEnumerator<T> GetEnumerator();
}

public interface IEnumerator<T> : IDisposable
{
    T Current { get;
    bool MoveNext();
    void Dispose();
}
```

2. Call MoveNext to load the next value

true - A value is present
false - no more values
Exception - an error occurred

IEnumerable for PULL based sequences

```
public interface IEnumerable<T>
{
    IEnumerator<T> GetEnumerator();
}

public interface IEnumerator<T> : IDisposable
{
    T Current { get; }
    bool MoveN
```

3. Get the value from Current and do some work

IEnumerable for PULL based sequences

```
public interface IEnumerable<T>
{
    IEnumerator<T> GetEnumerator();
}

public interface IEnumerator<T> : IDisposable
{
    T Current { get; }
    bool MoveNext();
    void Dispose();
}
```

4. Call MoveNext again or Dispose to cancel

IEnumerable for PULL based sequences

```
IEnumerable<string> Directory.EnumerateFiles(string path);
```

```
var myFiles =
    Directory.EnumerateFiles(@"\\networkshare\folder").
    Where(f => File.ReadAllText(f).Contains("orion"));

foreach(var path in myFiles) {
    Console.WriteLine("Matched File {0}", path);
}
```

IObservable for PUSH based sequences

```
public interface IObservable<T>
{
    IDisposable Subscribe(IObserver<T> observer);
}

public interface IObserver<T>
{
    void OnNext(T value);
    void OnError(Exception exception);
    void OnCompleted();
}
```

IObservable for PUSH based sequences

```
public interface IObservable<T>
{
    IDisposable Subscribe(IObserver<T> observer);
}

public interface IObserver<
{
    void OnNext(T value);
    void OnError(Exception exception);
    void OnCompleted();
}
```

1. Put your observer in

IObservable for PUSH based sequences

```
public interface IObse
{
    IDisposable Subscr
}

public interface IObse
{
    void OnNext(T value);
    void OnError(Exception exception);
    void OnCompleted();
}
```

2. When something happens
it calls the observer

OnNext - A value is present
OnCompleted - no more values
OnError - an exception

IObservable for PUSH based sequences

```
public interface IObservable<T>
{
    IDisposable Subscribe(IObserver<T> observer);
}

public interface IObserver<T>
{
    void OnNext(T value);
    void OnError(Exception exception);
}
```

3. Use the value in your OnNext to do some work

IObservable for PUSH based sequences

```
public interface IObservable<T>
{
    IDisposable Subscribe(IObserver<T> observer);
}
```

4. Dispose the disposable to cancel

```
{
    void OnNext(T value);
    void OnError(Exception exception);
    void OnCompleted();
}
```

IObservable for PUSH based sequences

```
IObservable<string> EnumerateFilesAsync(string path);
```

```
var myFiles =
    EnumerateFilesAsync(@"\\networkshare\folder").
    Where(f => File.ReadAllText(f).Contains("orion"));

myFiles.Subscribe(path => {
    Console.WriteLine("Matched File {0}", path);
});
```

Unified Interface for Collections

Pull

```
IEnumerable<T>
```

Push

```
IObservable<T>
```

LINQ Standard Query Operators

Pull

```
IEnumerable<Company> companies;

var managers = companies
    .SelectMany(company => employees)
    .Where(employee => employee.IsManager)
    .OrderByDescending(employee => employee.Salary);
```

Push

```
IObservable<Company> companies;

var managers = companies
    .SelectMany(company => employees)
    .Where(employee => employee.IsManager)
    .OrderByDescending(employee => employee.Salary);
```

Query Syntax

Pull

```
IEnumerable<Company> companies;

var managers = from company in companies
    from employee in company
    where employee.IsManager
    orderby employee.Salary descending
```

Push

```
IObservable<Company> companies;

var managers = from company in companies
    from employee in company
    where employee.IsManager
    orderby employee.Salary descending
```

Controlling Concurrency

- You always pull from an IEnumerable on ***your*** thread

- On observable will push on ***its*** thread

- We may need to redirect to another thread (e.g. Dispatcher thread)

Controlling Concurrency

```
public interface IScheduler
{
    IDisposable Schedule(Action action);
    IDisposable Schedule(Action action, TimeSpan dueTime);
}
```

```
companies
    .Where(c => c.IsNotBankrupt)
    .ObserveOn(Scheduler.Dispatcher)
    .Subscribe(c => { })
```

DEMO


SUPERCODER 2000
Air cooled coding keyboard for professional use.
Done
0
1

QUESTIONS?

orion.edwards@gmail.com
twitter: @borland