BASTA!
.NET, WINDOWS, VISUAL STUDIO

Rainer Stropek | software architects gmbh

C# Powerworkshop

C# - Gegenwart und Zukunft

Die fünfte Version von C# ist da. Zeit, sich intensiv damit auseinanderzusetzen und einen Blick in die Zukunft zu werfen. Rainer Stropek bietet auch dieses Jahr wieder geballtes C#-Wissen in diesem ganztägigen Workshop an. Der Schwerpunkt sind die **Neuerungen von C# 5 hinsichtlich asynchroner und paralleler Programmierung**. Rainer wiederholt zu Beginn die **Grundlagen der parallelen Programmierung mit .NET** (und wird dabei viele nützliche Tipps weitergeben). Danach geht er auf die Anwendung dieser Basics in C# 5 mit async/await ein. Wir kratzen nicht nur an der Oberfläche, sondern gehen wirklich ins Detail. Am Nachmittag wird Rainer einen **Ausblick** auf die Zukunft von C# geben und zeigen, was **Projekte wie "Roslyn"** an Änderungen für C#-Entwickler bringen werden.

Agenda

- Vormittag
  - Block 1 – TPL Grundlagen (.NET 4)
    - Arbeiten mit Tasks
    - Die `Parallel`-Klasse
  - Block 2 – TPL Advanced (.NET 4 & 4.5)
    - Parallel LINQ
    - Collections für parallele Programmierung
    - TPL Dataflow Library
- Nachmittag
  - Block 3 – async/await (C# 5)
    - C# Spracherweiterungen async/await
    - Beispiele
  - Block 4 – C# und .NET Zukunft
    - Modularisierung durch Nuget
    - Roslyn

Async Programming in C# (.NET 4.5/C# 5)

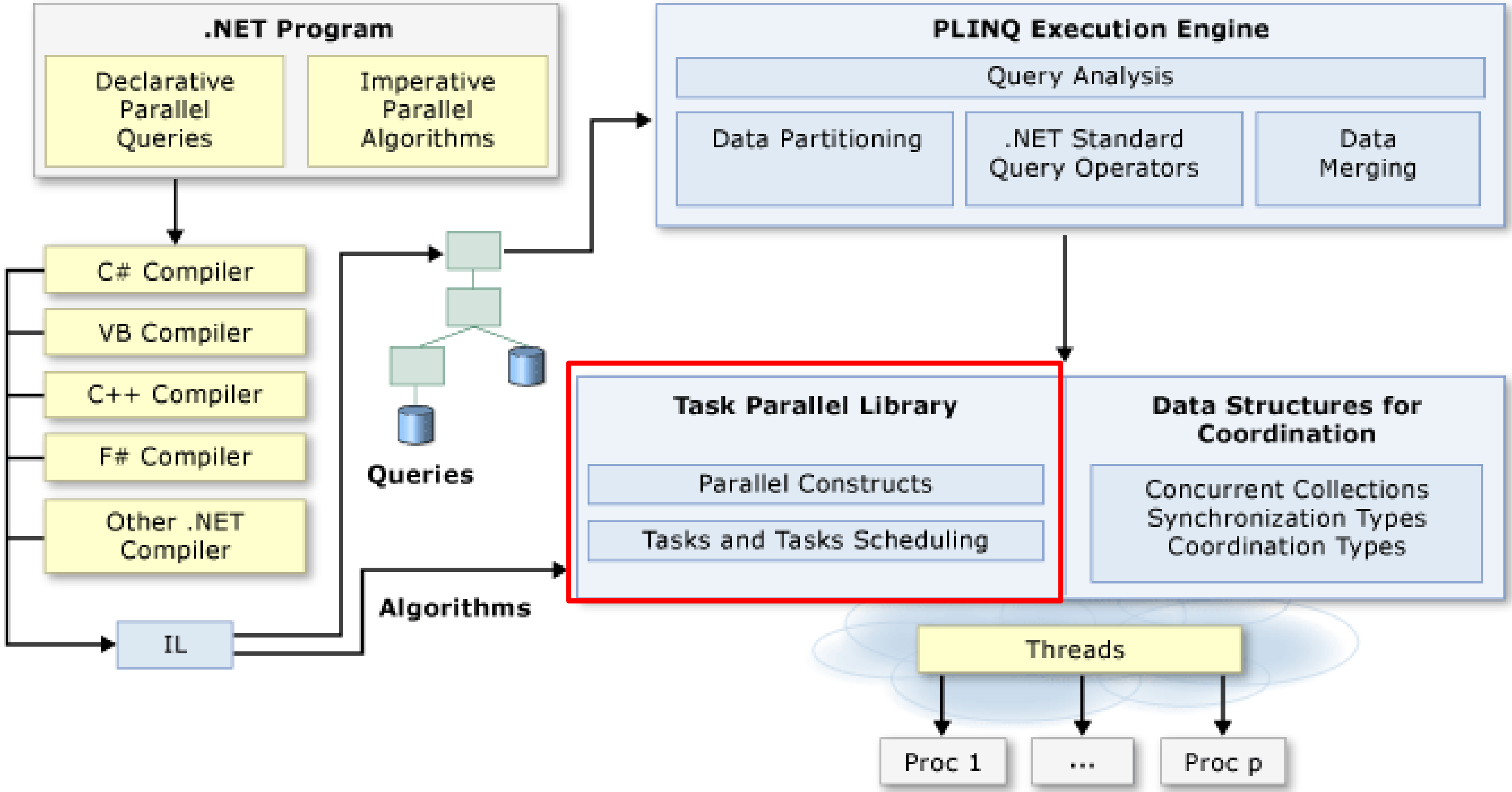
ASYNC/PARALLEL PROGRAMMING

Goals

- Understand Tasks → foundation for `async/await`
- Take a close look at C# 4.5‘s stars `async/await`
- Present enhancements in .NET 4.5 BCL: TPL Dataflow Library

Recommended Reading

- Joseph Albahari, Threading in C#
  (from his O‘Reilly book C# 4.0 in a Nutshell)
- Patterns of Parallel Programming
- Task-based Asynchronous Pattern
- A technical introduction to the Async CTP
- Using Async for File Access
- Async Performance: Understanding the Costs of Async and Await (MSDN Magazine)



See also Patterns of Parallel Programming

Multithreading

Pre .NET 4

- `System.Threading` Namespace
- `Thread` Klasse
- `ThreadPool` Klasse

.NET 4

- `System.Threading.Tasks` Namespace
- `Task` und `Task<TResult>` Klassen
- `TaskFactory` Klasse
- **`Parallel`** Klasse

Kurzer Überblick über Tasks

- Starten
  - `Parallel.Invoke(…)`
  - `Task.Factory.StartNew(…)`
- Warten
  - `myTask.Wait()`
  - `Task.WaitAll`
  - `Task.WaitAny`
  - `Task.Factory.ContinueWhenAll(…)`
  - `Task.Factory.ContinueWhenAny(…)`
- Verknüpfen

  -
    ```
    Task.Factory.StartNew(…,
        TaskCreationOptions.AttachedToParent);
    ```
  - `Task.ContinueWith(…)`
- Abbrechen
  - Cancellation Tokens

MSDN

```
private static void DoSomething()
{
   Action<Action> measure = (body) =>
   {
      var startTime = DateTime.Now;
      body();
      Console.WriteLine("{0} {1}",
         Thread.CurrentThread.ManagedThreadId,
         DateTime.Now - startTime);
   };

   Action calcProcess = () =>
      { for (int i = 0; i < 100000000; i++);};

   measure(() =>
      Task.WaitAll(Enumerable.Range(0, 10)
         .Select(i => Task.Run(() => measure(calcProcess)))
         .ToArray()));
}
```

This process will run in parallel

Note that we use the new `Task.Run` function here; previously you had to use `Task.Factory.StartNew`

```
Action<Action> measure = (body) => {
    var startTime = DateTime.Now;
    body();
    Console.WriteLine("{0} {1}",
      Thread.CurrentThread.ManagedThreadId,
      DateTime.Now - startTime);
};

Action calcProcess = () =>
   { for (int i = 0; i < 350000000; i++);};
Action ioProcess = () =>
   { Thread.Sleep(1000); };

// ThreadPool.SetMinThreads(5, 5);
measure(() =>{
    Task.WaitAll(Enumerable.Range(0, 10)
        .Select(i => Task.Run(() => measure(ioProcess)))
        .ToArray());
});
```

Note that this task is not compute-bound

```
Action<Action> measure = (body) =>{
   var startTime = DateTime.Now;
   body();
   Console.WriteLine("{0} {1}", Thread.CurrentThread.ManagedThreadId,
                     DateTime.Now - startTime);
};

Action calcProcess = () => { for (int i = 0; i < 350000000; i++);};
Action ioProcess = () => { Thread.Sleep(1000); };

ThreadPool.SetMinThreads(5, 5);
measure(() => Enumerable.Range(0, 10)
   .AsParallel()
   .WithDegreeOfParallelism(5)
   .ForAll(i => measure(ioProcess)));
```

```
private static void DoSomethingElse()
{
   Func<int, int> longRunningFunc = (prevResult) =>
      {
         Thread.Sleep(1000);
         return prevResult + 42;
      };
```

Concat tasks using ContinueWith

```
   var task   Task.Run(() => longRunningFunc(0))
      .ContinueWith(t => longRunningFunc(t.Result))
      .ContinueWith(t => longRunningFunc(t.Result));
   task.Wait();
   Console.WriteLine(task.Result);
}
```

Wait for completion of a task.

Schleifen - Parallel.For

```
var source = new double[Program.Size];
var destination = new double[Program.Size];

Console.WriteLine(MeasuringTools.Measure(() => {
        for (int i = 0; i < Program.Size; i++) {
            source[i] = (double)i;
        }

        for (int i = 0; i < Program.Size; i++) {
            destination[i] = Math.Pow(source[i], 2);
        }
    }));

Console.WriteLine(MeasuringTools.Measure(() => {
        Parallel.For(0, Program.Size, (i) => source[i] = (double)i);
        Parallel.For(0, Program.Size,
            (i) => destination[i] = Math.Pow(source[i], 2));
    }));
```

Schleifen - `Parallel.For`

- Unterstützung für Exception Handling
- `Break` und `Stop` Operationen
  - `Stop`: Keine weiteren Iterationen
  - `Break`: Keine Iterationen nach dem aktuellen Index mehr
  - Siehe dazu auch `ParallelLoopResult`
- `Int32` und `Int64` Laufvariablen
- Konfigurationsmöglichkeiten (z.B. Anzahl an Threads)
- Schachtelbar
  - Geteilte Threading-Ressourcen
- Effizientes Load Balancing
- U.v.m.

Nicht selbst entwickeln!

Schleifen - `Parallel.ForEach`

```
Console.WriteLine(
    "Serieller Durchlauf mit foreach: {0}",
    MeasuringTools.Measure(() =>
    {
        double sumOfSquares = 0;
        foreach (var square in Enumerable.Range(0, Program.Size).Select(
            i => Math.Pow(i, 2)))
        {
            sumOfSquares += square;
        }
    }));

Console.WriteLine(
    "Paralleler Durchlauf mit foreach: {0}",
    MeasuringTools.Measure(() =>
    {
        double sumOfSquares = 0;
        Parallel.ForEach(Enumerable.Range(0, Program.Size)
            .Select(i => Math.Pow(i, 2)), square => sumOfSquares += square);
    }));
```

Hoher Aufwand für abgesicherten Zugriff auf MoveNext/Current
→ Parallele Version oft langsamer

Von LINQ zu PLINQ

LINQ

```
var result = source
     .Where(…)
     .Select(…)
```

PLINQ

```
var result = source
     .AsParallel()
     .Where(…)
     .Select(…)
```

Aus `IEnumerable` wird `ParallelQuery`

Tipp: `AsOrdered()` erhält die Sortierreihenfolge

Excursus - PLINQ

- Use `.AsParallel` to execute LINQ query in parallel
- Be careful if you care about ordering
  - Use `.AsOrdered` if necessary
- Use `.WithDegreeOfParallelism` in case of IO-bound tasks
- Use `.WithCancellation` to enable cancelling

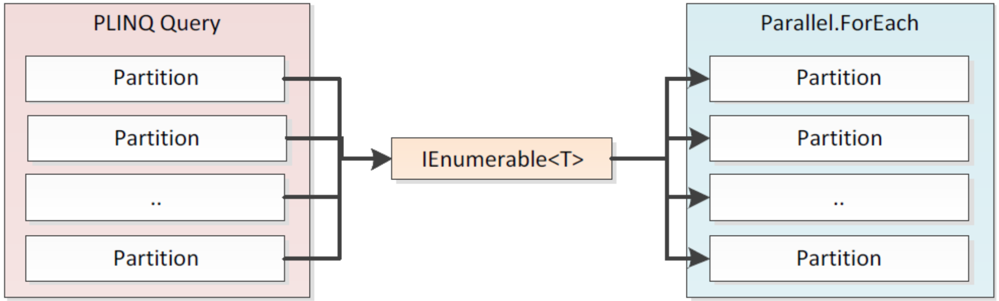
Performancetipps für PLINQ

- Allokieren von Speicher in parallelem Lambdaausdruck vermeiden
  - Sonst kann Speicher + GC zum Engpass werden
  - Wenn am Server: Server GC
- False Sharing vermeiden
- Bei zu kurzen Delegates ist Koordinationsaufwand für Parallelisierung oft höher als Performancegewinn
  - → Expensive Delegates
  - Generell: Auf richtige Granularität der Delegates achten
- `AsParallel()` kann an jeder Stelle im LINQ Query stehen
  - → Teilweise serielle, teilweise parallele Ausführung möglich
- Über `Environment.ProcessorCount` kann Anzahl an Kernen ermittelt werden
- Messen, Messen, Messen!

Was läuft hier falsch? (Code)

```
var result = new List<double>();
Console.WriteLine(
    "Paralleler Durchlauf mit Parallel.ForEach: {0}",
    MeasuringTools.Measure(() =>
    {
        Parallel.ForEach(
            source.AsParallel(),
            i =>
            {
                if (i % 2 == 0)
                {
                    lock (result)
                    {
                        result.Add(
                    }
                }
            });
    }));
```

Parallel.ForEach verwendet `IEnumerable<T>` → unnötige Merge-Schritte

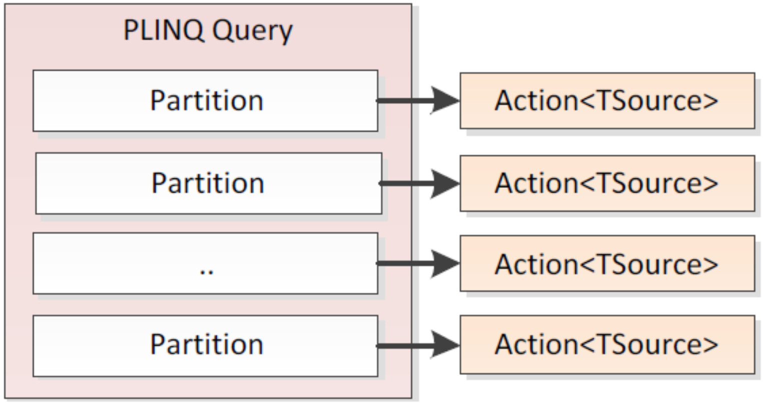


Was läuft hier falsch? (Code)

```
Console.WriteLine(
    "Paralleler Durchlauf mit Parallel.ForAll: {0}",
    MeasuringTools.Measure(() =>
    {
        source.AsParallel().ForAll(
            i =>
            {
                if (i % 2 == 0)
                {
                    lock (result)
                    {
                        result.Add(i);
                    }
                }
            });
    }));
```

✓

Lock-free Collection wäre überlegenswert!



Was läuft hier falsch? (Code)

```
Console.WriteLine(
    "Serielles Lesen: {0}",
    MeasuringTools.Measure(() =>
    {
        foreach (var url in urls)
        {
            var request = WebRequest.Create(url);
            using (var response = request.GetResponse())
            {
                using (var stream = response.GetResponseStream())
                {
                    var content = new byte[1024];
                    while (stream.Read(content, 0, 1024) != 0) ;
                }
            }
        }
    }));
```

Optimal für Parallelisierung selbst bei einem Core (IO-Bound Waits)

Was läuft hier falsch? (Code)

```
Console.WriteLine(
    "Paralleles Lesen: {0}",
    MeasuringTools.Measure(() =>
    {
        Parallel.ForEach(urls, url =>
        {
            var request = WebRequest.Create(url);
            using (var response = request.GetResponse())
            {
                using (var stream = response.GetResponseStream())
                {
                    var content = new byte[1024];
                    while (stream.Read(content, 0, 1024) != 0) ;
                }
            }
        });
    }));
```

Anzahl Threads = Anzahl Cores; könnte mehr sein, da IO-Bound waits

```
Parallel.ForEach(
  urls,
  new ParallelOptions() { MaxDegreeOfParallelism = urls.Length },
  url => { … });
```

Was läuft hier falsch? (Code)

```
Console.WriteLine(
    "Paralleles Lesen: {0}",
    MeasuringTools.Measure(() =>
    {
        urls.AsParallel().WithDegreeOfParallelism(urls.Length)
            .Select(url => WebRequest.Create(url))
            .Select(request => request.GetResponse())
            .Select(response => new {
                Response = response,
                Stream = response.GetResponseStream() })
            .ForAll(stream =>
                {
                    var content = new byte[1024];
                    while (stream.Stream.Read(content, 0, 1024) != 0) ;
                    stream.Stream.Dispose();
                    stream.Response.Close();
                });
    }));
```

✓

OK für Client, tödlich für Server!
Wenn Anzahl gleichzeitiger User wichtig ist sind andere Lösungen vorzuziehen.

Thread Synchronisation

- Use C# `lock` statement to control access to shared variables
  - Under the hoods `Monitor.Enter` and `Monitor.Exit` is used
  - Quite fast, usually fast enough
  - Only care for lock-free algorithms if really necessary
- Note that a thread can lock the same object in a nested fashion

```
// Source: C# 4.0 in a Nutshell, O‘Reilly Media
class ThreadSafe
{
 static readonly object _locker = new object();
 static int _val1, _val2;

 static void Go()
 {
  lock (_locker)
  {
   if (_val2 != 0) Console.WriteLine (_val1 / _val2);
   _val2 = 0;
  }
 }
}

// This is what happens behind the scenes
bool lockTaken = false;
try
{
 Monitor.Enter(_locker, ref lockTaken);
 // Do your stuff...
}
finally
{
  if (lockTaken) Monitor.Exit(_locker);
}
```

```
// Provide a factory for instances of the Random class per thread
var tlr = new ThreadLocal<Random>(
    () => new Random(Guid.NewGuid().GetHashCode()));

var watch = Stopwatch.StartNew();

var tasks =
    // Run 10 tasks in parallel
    Enumerable.Range(0, 10)
        .Select(_ => Task.Run(() =>
            // Create a lot of randoms between 0 and 9 and calculate
            // the sum
            Enumerable.Range(0, 1000000)
                .Select(__ => tlr.Value.Next(10))
                .Sum()))
        .ToArray();
Task.WaitAll(tasks);

// Calculate the total
Console.WriteLine(tasks.Aggregate<Task<int>, int>(
    0, (agg, val) => agg + val.Result));

Console.WriteLine(watch.Elapsed);

watch = Stopwatch.StartNew();
```

Do you think this is a good solution?

```
// Provide a factory for instances of the Random class per thread
var tlr = new ThreadLocal<Random>(
    () => new Random(Guid.NewGuid().GetHashCode()));

var watch = Stopwatch.StartNew();

Console.WriteLine(
    ParallelEnumerable.Range(0, 10000000)
        .Select(_ => tlr.Value.Next(10))
        .Sum());

Console.WriteLine(watch.Elapsed);
```

Prefer PLINQ over TPL because it automatically breaks the workload into packages.

Alternatives For `lock`

- `Mutex`
- `Semaphore(Slim)`
- `ReaderWriterLock(Slim)`
- Not covered here in details

Thread Synchronization

- `AutoResetEvent`
  - Unblocks a thread once when it receives a signal from another thread
- `ManualResetEvent(Slim)`
  - Like a door, opens and closes again
- `CountdownEvent`
  - New in .NET 4
  - Unblocks if a certain number of signals have been received
- `Barrier` class
  - New in .NET 4
  - Not covered here
- `Wait` and `Pulse`
  - Not covered here

```
private static void DownloadSomeTextSync()
{
   using (var client = new WebClient())
   {
      Console.WriteLine(
         client.DownloadString(new Uri(string.Format(
            "http://{0}",
            (Dns.GetHostAddresses("www.basta.net"))[0]))));
   }
}
```

Synchronous version of the code;
would block UI thread

```
private static void DownloadSomeText()
{
    var finishedEvent = new AutoResetEvent(false);

    // Notice the IAsyncResult-pattern here
    Dns.BeginGetHostAddresses("www.basta.net", GetHostEntryFinished,
        finishedEvent);
    finishedEvent.WaitOne();
}

private static void GetHostEntryFinished(IAsyncResult result)
{
    var hostEntry = Dns.EndGetHostAddresses(result);
    using (var client = new WebClient())
    {
        // Notice the Event-based asynchronous pattern here
        client.DownloadStringCompleted += (s, e) =>
        {
            Console.WriteLine(e.Result);
            ((AutoResetEvent)result.AsyncState).Set();
        };
        client.DownloadStringAsync(new Uri(string.Format(
            "http://{0}",
            hostEntry[0].ToString())));
    }
}
```

Notice that control flow is not clear any more.

```
private static void DownloadSomeText()
{
    var finishedEvent = new AutoResetEvent(false);

    // Notice the IAsyncResult-pattern here
    Dns.BeginGetHostAddresses(
        "www.basta.net",
        (result) =>
        {
            var hostEntry = Dns.EndGetHostAddresses(result);
            using (var client = new WebClient())
            {
                // Notice the Event-based asynchronous pattern here
                client.DownloadStringCompleted += (s, e) =>
                {
                    Console.WriteLine(e.Result);
                    ((AutoResetEvent)result.AsyncState).Set();
                };
                client.DownloadStringAsync(new Uri(string.Format(
                    "http://{0}",
                    hostEntry[0].ToString())));
            }
        },
        finishedEvent);
    finishedEvent.WaitOne();
}
```

Notice how lambda expression can make control flow clearer

Notice the use of the new Task Async Pattern APIs in .NET 4.5 here

```
private static void DownloadSomeTextUsingTask()
{
    Dns.GetHostAddressesAsync("www.basta.net")
        .ContinueWith(t =>
        {
            using (var client = new WebClient())
            {
                return client.DownloadStringTaskAsync(new Uri(string.Format(
                        "http://{0}",
                        t.Result[0].ToString())));
            }
        })
        .ContinueWith(t2 => Console.WriteLine(t2.Unwrap().Result))
        .Wait();
}
```

Notice the use of lambda expressions all over the methods

Notice how code has become shorter and more readable

Rules For Async Method Signatures

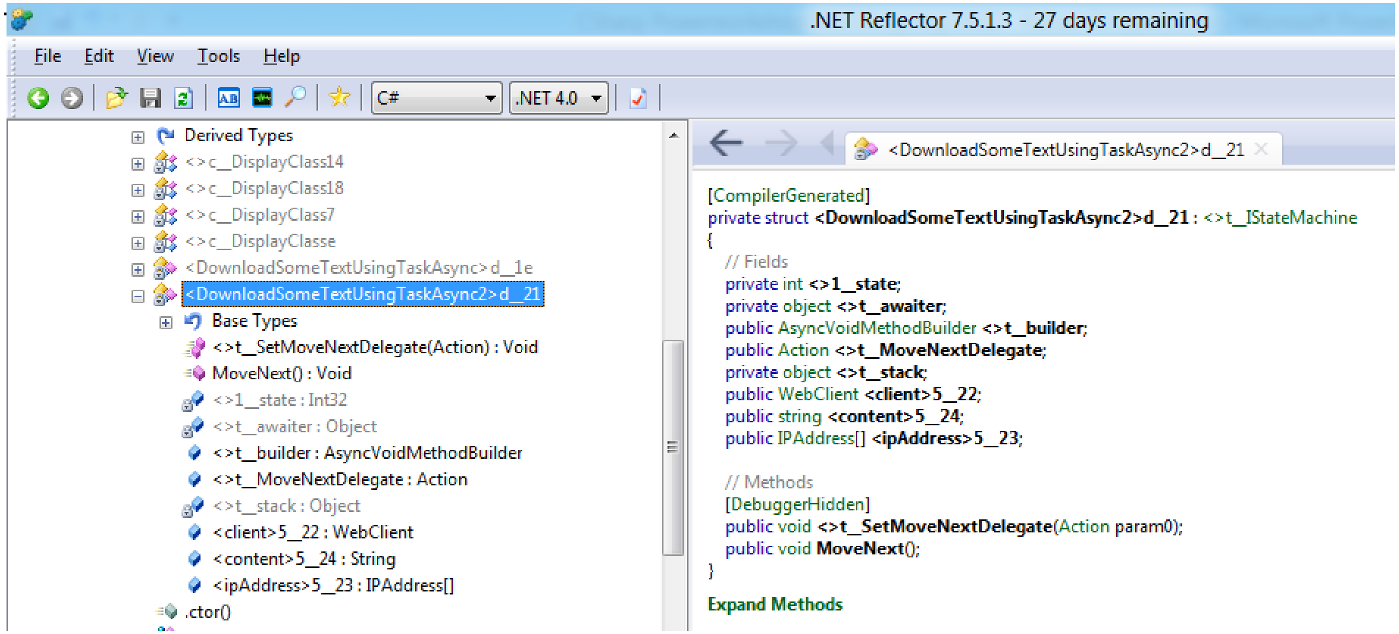
- Method name ends with `Async`
- Return value
  - `Task` if sync version has return type `void`
  - `Task<T>` if sync version has return type `T`
- Avoid `out` and `ref` parameters
  - Use e.g. `Task<Tuple<T1, T2, …>>` instead

```
// Synchronous version
private static void DownloadSomeTextSync()
{
    using (var client = new WebClient())
    {
        Console.WriteLine(
            client.DownloadString(new Uri(string.Format(
                "http://{0}",
                (Dns.GetHostAddresses("www.basta.net"))[0]))));
    }
}
```

Notice how similar the sync and async versions are!

```
// Asynchronous version
private static async void DownloadSomeTextUsingTaskAsync()
{
    using (var client = new WebClient())
    {
        Console.WriteLine(
            await client.DownloadStringTaskAsync(new Uri(string.Format(
                "http://{0}",
                (await Dns.GetHostAddressesAsync("www.basta.net"))[0]))));
    }
}
```

```
private static async void DownloadSomeTextUsingTaskAsync2()
{
    using (var client = new WebClient())
    {
        try
        {
            var ipAddress = await Dns.GetHostAddressesAsync("www.basta.net");
            var content = await client.DownloadStringTaskAsync(
                new Uri(string.Format("htt://{0}", ipAddress[0])));
            Console.WriteLine(content);
        }
        catch (Exception)
        {
            Conso
        }
    }
}
```


.NET Reflector 7.5.1.3 - 27 days remaining
File
Edit
View
Tools
Help
C#
.NET 4.0
<DownloadSomeTextUsingTaskAsync2>d__21

```
Derived Types
<>c__DisplayClass14
<>c__DisplayClass18
<>c__DisplayClass7
<>c__DisplayClasse
<DownloadSomeTextUsingTaskAsync>d__1e
<DownloadSomeTextUsingTaskAsync2>d__21
    Base Types
    <>t__SetMoveNextDelegate(Action) : Void
    MoveNext() : Void
    <>1__state : Int32
    <>t__awaiter : Object
    <>t__builder : AsyncVoidMethodBuilder
    <>t__MoveNextDelegate : Action
    <>t__stack : Object
    <client>5__22 : WebClient
    <content>5__24 : String
    <ipAddress>5__23 : IPAddress[]
.ctor()
```

```
[CompilerGenerated]
private struct <DownloadSomeTextUsingTaskAsync2>d__21 : <>t__IStateMachine
{
    // Fields
    private int <>1__state;
    private object <>t__awaiter;
    public AsyncVoidMethodBuilder <>t__builder;
    public Action <>t__MoveNextDelegate;
    private object <>t__stack;
    public WebClient <client>5__22;
    public string <content>5__24;
    public IPAddress[] <ipAddress>5__23;

    // Methods
    [DebuggerHidden]
    public void <>t__SetMoveNextDelegate(Action param0);
    public void MoveNext();
}
```

Expand Methods

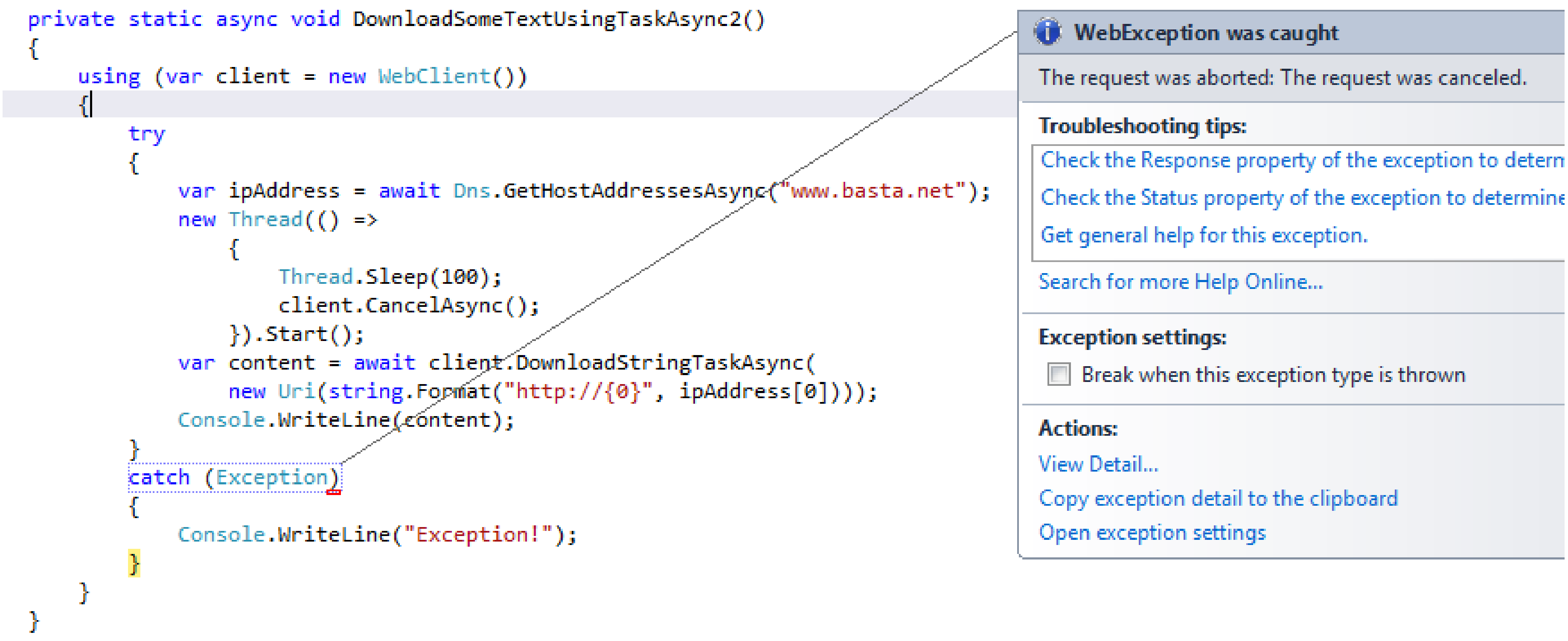
Let's check the generated code and debug the async code

Guidelines for `async/await`

- If `Task` ended in `Canceled` state, `OperationCanceledException` will be thrown

```
private async static void CancelTask()
{
    try
    {
        var cancelSource = new CancellationTokenSource();
        var result = await DoSomethingCancelledAsync(cancelSource.Token);
        Console.WriteLine(result);
    }
    catch (OperationCanceledException)
    {
        Console.WriteLine("Cancelled!");
    }
}

private static Task<int> DoSomethingCancelledAsync(CancellationToken token)
{
    // For demo purposes we ignore token and always return a cancelled task
    var result = new TaskCompletionSource<int>();
    result.SetCanceled();
    return result.Task;
}
```

Note usage of
TaskCompletionSource<T> here



```
private static async void DownloadSomeTextUsingTaskAsync2()
{
    using (var client = new WebClient())
    {
        try
        {
            var ipAddress = await Dns.GetHostAddressesAsync("www.basta.net");
            new Thread(() =>
                {
                    Thread.Sleep(100);
                    client.CancelAsync();
                }).Start();
            var content = await client.DownloadStringTaskAsync(
                new Uri(string.Format("http://{0}", ipAddress[0])));
            Console.WriteLine(content);
        }
        catch (Exception)
        {
            Console.WriteLine("Exception!");
        }
    }
}
```

WebException was caught

The request was aborted: The request was canceled.

Troubleshooting tips:

Check the Response property of the exception to detern

Check the Status property of the exception to determine

Get general help for this exception.

Search for more Help Online...

Exception settings:

□ Break when this exception type is thrown

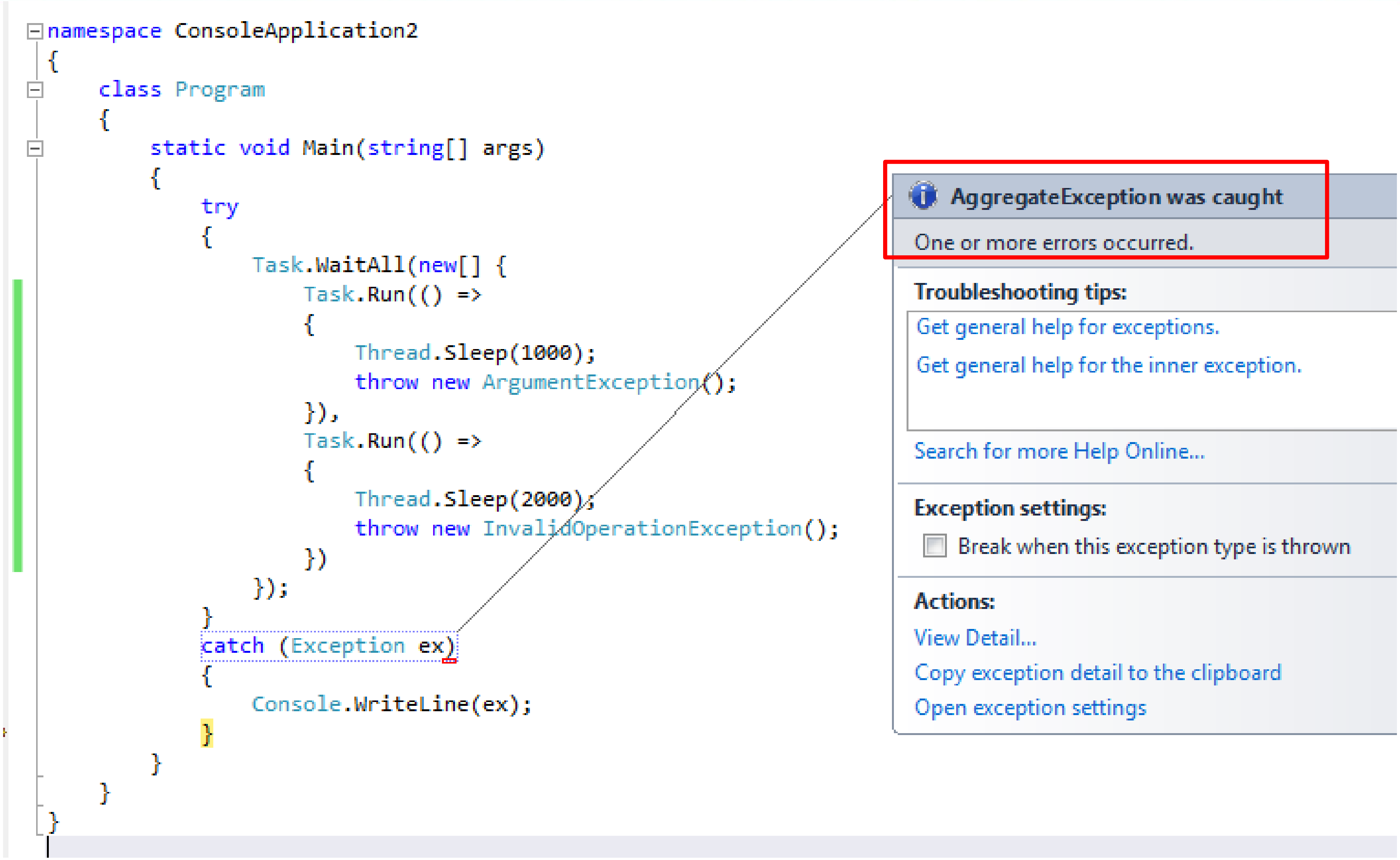
Actions:

View Detail...

Copy exception detail to the clipboard

Open exception settings

Note that async API of `WebClient` uses existing cancellation logic instead of `CancellationTokenSource`



```
namespace ConsoleApplication2
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                Task.WaitAll(new[] {
                    Task.Run(() =>
                    {
                        Thread.Sleep(1000);
                        throw new ArgumentException();
                    }),
                    Task.Run(() =>
                    {
                        Thread.Sleep(2000);
                        throw new InvalidOperationException();
                    })
                });
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex);
            }
        }
    }
}
```

AggregateException was caught

One or more errors occurred.

Troubleshooting tips:

Get general help for exceptions.

Get general help for the inner exception.

Search for more Help Online...

Exception settings:

□ Break when this exception type is thrown

Actions:

View Detail...

Copy exception detail to the clipboard

Open exception settings

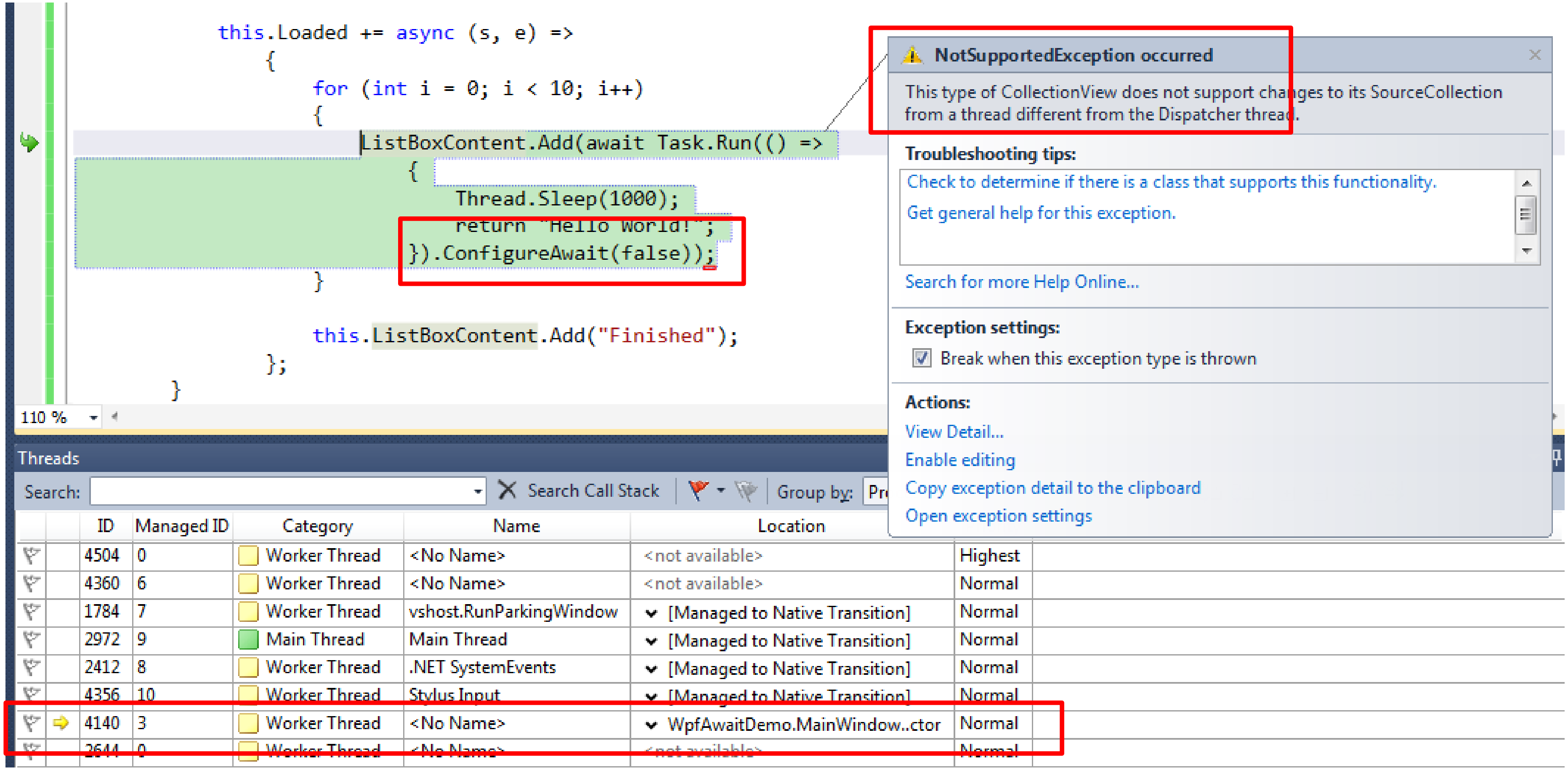
Guidelines for `async/await`

- Caller runs in parallel to awaited methods
- Async methods sometimes do not run async (e.g. if task is already completed when `async` is reached)

Guidelines for `async/await` (UI Layer)

- `async/await` use `SynchronizationContext` to execute the awaiting method → UI thread in case of UI layer
- Use `Task.ConfigureAwait` to disable this behavior
  - E.g. inside library to enhance performance

```
public partial class MainWindow : Window
{
public MainWindow()
{
    this.DataContext = this;
    this.ListBoxContent = new ObservableCollection<string>();
    this.InitializeComponent();
    this.ListBoxContent.Add("Started");

    this.Loaded += async (s, e) =>
        {
            for (int i = 0; i < 10; i++)
            {
                ListBoxContent.Add(await Task.Run(() =>
                    {
                        Thread.Sleep(1000);
                        return "Hello World!";
                    }));
            }

            this.ListBoxContent.Add("Finished");
        };
}

public ObservableCollection<string> ListBoxContent { get; private set; }
```



```
        this.Loaded += async (s, e) =>
            {
                for (int i = 0; i < 10; i++)
                {
                    ListBoxContent.Add(await Task.Run(() =>
                        {
                            Thread.Sleep(1000);
                            return "Hello World!";
                        }).ConfigureAwait(false));
                }

                this.ListBoxContent.Add("Finished");
            };
    }
```

NotSupportedException occurred

This type of CollectionView does not support changes to its SourceCollection from a thread different from the Dispatcher thread.

Troubleshooting tips:

Check to determine if there is a class that supports this functionality.

Get general help for this exception.

Search for more Help Online...

Exception settings:

☑ Break when this exception type is thrown

Actions:

View Detail...

Enable editing

Copy exception detail to the clipboard

Open exception settings

		ID	Managed ID	Category	Name	Location	
		4504	0	Worker Thread	<No Name>	<not available>	Highest
		4360	6	Worker Thread	<No Name>	<not available>	Normal
		1784	7	Worker Thread	vshost.RunParkingWindow	[Managed to Native Transition]	Normal
		2972	9	Main Thread	Main Thread	[Managed to Native Transition]	Normal
		2412	8	Worker Thread	.NET SystemEvents	[Managed to Native Transition]	Normal
		4356	10	Worker Thread	Stylus Input	[Managed to Native Transition]	Normal
	⇨	4140	3	Worker Thread	<No Name>	WpfAwaitDemo.MainWindow..ctor	Normal
		2644	0	Worker Thread	<No Name>	<not available>	Normal

Guidelines For Implementing Methods Ready For `async/await`

- Return `Task/Task<T>`
- Use postfix `Async`
- If method support cancelling, add parameter of type `System.Threading.CancellationToken`
- If method support progress reporting, add `IProgress<T>` parameter
- Only perform very limited work before returning to the caller (e.g. check arguments)
- Directly throw exception only in case of *usage* errors

```
public class Program : IProgress<int>
{
    static void Main(string[] args)
    {
        var finished = new AutoResetEvent(false);
        PerformCalculation(finished);
        finished.WaitOne();
    }

    private static async void PerformCalculation(AutoResetEvent finished)
    {
        Console.WriteLine(await CalculateValueAsync(
            42,
            CancellationToken.None,
            new Program()));
        finished.Set();
    }

    public void Report(int value)
    {
        Console.WriteLine("Progress: {0}", value);
    }
    private static Task<int> CalculateValueAsync(
        int startingValue,
        CancellationToken cancellationToken,
        IProgress<int> progress)
    {
        if (startingValue < 0)
        {
            // Usage error
            throw new ArgumentOutOfRangeException("startingValue");
        }

        return Task.Run(() =>
            {
                int result = startingValue;
                for (int outer = 0; outer < 10; outer++)
                {
                    cancellationToken.ThrowIfCancellationRequested();

                    // Do some calculation
                    Thread.Sleep(500);
                    result += 42;

                    progress.Report(outer + 1);
                }

                return result;
            });
    }
}
```

Note that this pattern is good for compute-bound jobs

```
private static async void PerformCalculation(AutoResetEvent finished)
{
    try
    {
        var cts = new CancellationTokenSource();
        Task.Run(() =>
            {
                Thread.Sleep(3000);
                cts.Cancel();
            });
        var result = await CalculateValueAsync(
            42,
            cts.Token,
            new Program());
    }
    catch (OperationCanceledException)
    {
        Console.WriteLine("Cancelled!");
    }

    finished.Set();
}
```

Note cancellation and handling of `OperationCanceledException`.

```
private static Task<int> CalculateValueAsync(
    int startingValue,
    CancellationToken cancellationToken,
    IProgress<int> progress)
{
    if (startingValue < 0)
    {
        // By definition the result has to be 0 if startingValue < 0
        return Task.FromResult(0);
    }

    return Task.Run(() =>
        {
            […]
        });
}
```

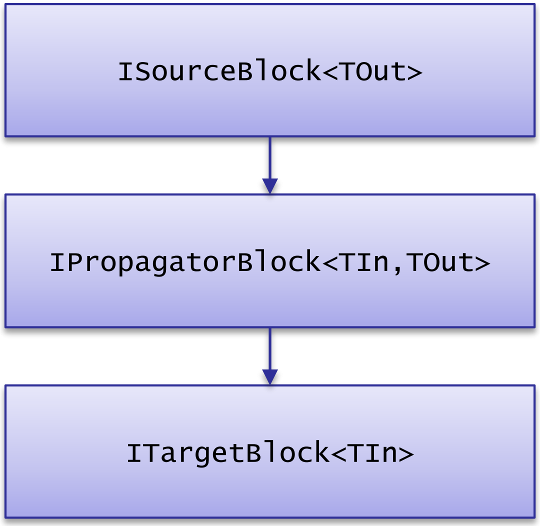
Note that you could use `TaskCompletionSource` instead

Note how `Task.FromResult` is used to return a pseudo-task

Overview

- System.Threading.Tasks.Dataflow
  - You need to install the Microsoft.Tpl.Dataflow NuGet package to get it
- For parallelizing applications with high throughput and low latency

Sources and Targets



- Sources, Propagators, and Targets
- Use `LinkTo` method to connect
  - Optional filtering
- Use `Complete` method after completing work
- Message passing
  - `Post/SendAsync` to send
  - `Receive/ReceiveAsync / TryReceive` to receive

Buffering Blocks

```
// Create a BufferBlock<int> object.
var bufferBlock = new BufferBlock<int>();

// Post several messages to the block.
for (int i = 0; i < 3; i++)
{
   bufferBlock.Post(i);
}

// Receive the messages back from the block.
for (int i = 0; i < 3; i++)
{
   Console.WriteLine(bufferBlock.Receive());
}

/* Output:
   0
   1
   2
 */
```

- BufferBlock<T>
- BroadcastBlock<T>
- WriteOnceBlock<T>

Execution Blocks

```
// Create an ActionBlock<int> object that prints values
// to the console.
var actionBlock = new ActionBlock<int>(n => Console.WriteLine(n));

// Post several messages to the block.
for (int i = 0; i < 3; i++)
{
   actionBlock.Post(i * 10);
}

// Set the block to the completed state and wait for all
// tasks to finish.
actionBlock.Complete();
actionBlock.Completion.Wait();

/* Output:
   0
   10
   20
 */
```

- ActionBlock<T>
- TransformBlock<T>
- TransformManyBlock <T>

Grouping Blocks

```
// Create a BatchBlock<int> object that holds ten
// elements per batch.
var batchBlock = new BatchBlock<int>(10);

// Post several values to the block.
for (int i = 0; i < 13; i++)
{
   batchBlock.Post(i);
}
// Set the block to the completed state. This causes
// the block to propagate out any any remaining
// values as a final batch.
batchBlock.Complete();

// Print the sum of both batches.

Console.WriteLine("The sum of the elements in batch 1 is {0}.",
   batchBlock.Receive().Sum());

Console.WriteLine("The sum of the elements in batch 2 is {0}.",
   batchBlock.Receive().Sum());

/* Output:
   The sum of the elements in batch 1 is 45.
   The sum of the elements in batch 2 is 33.
 */
```

- BatchBlock<T>
- JoinBlock<T>
- BatchedJoinBlock<T>

Die Zukunft

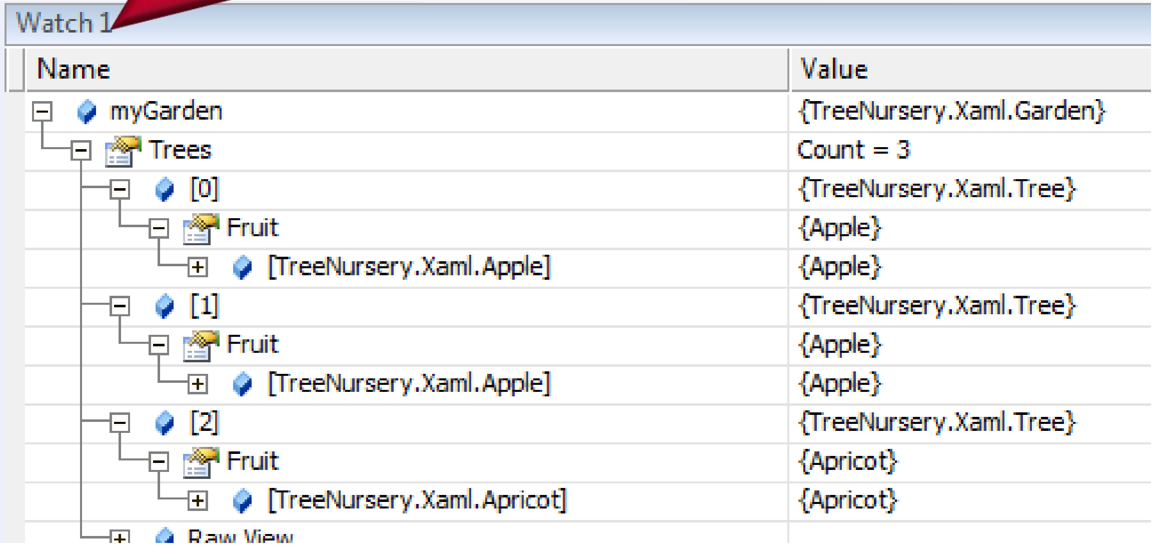
ROSLYN

Von Text zum Baum

```
<Garden xmlns="clr-namespace:TreeNursery.Xaml;assembly=TreeNursery">
    <Garden.Trees>
        <Tree>
            <Tree.Fruit>
                <Apple />
            </Tree.Fruit>
        </Tree>
        <Tree>
            <Tree.Fruit>
                <Apple />
            </Tree.Fruit>
        </Tree>
        <Tree>
            <Tree.Fruit>
                <Apricot />
            </Tree.Fruit>
        </Tree>
    </Garden.Trees>
</Garden>
```

Parser

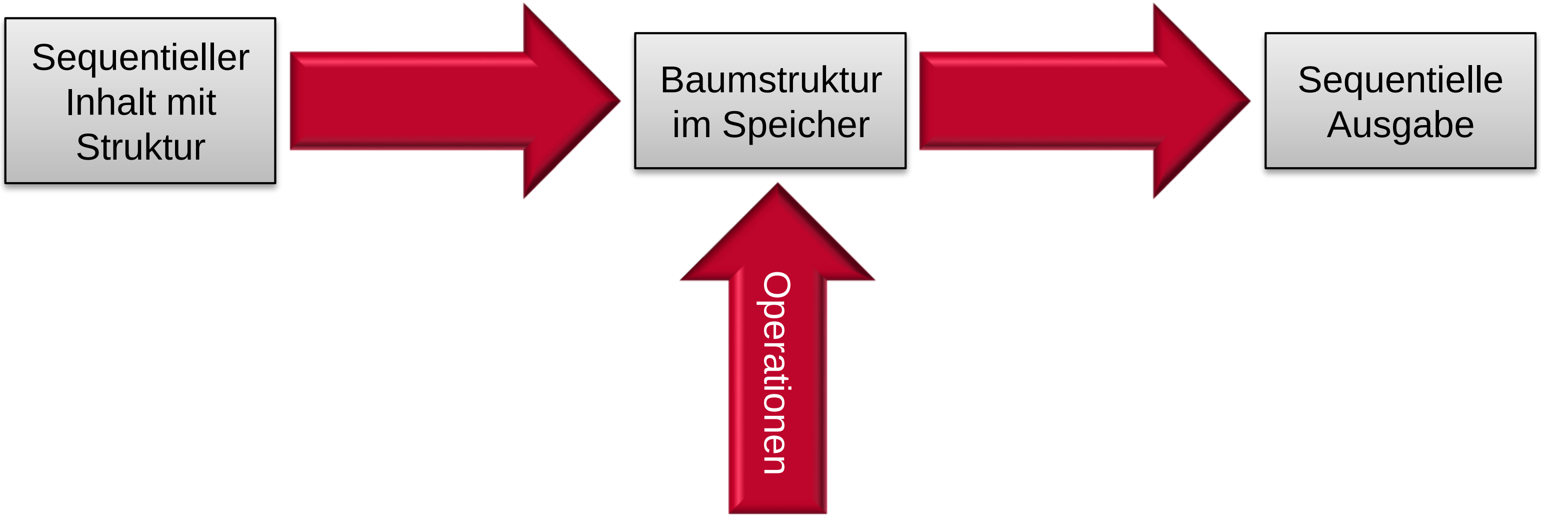
XAML → Objekt-baum im Speicher



Watch 1

Name	Value
⊟ myGarden	{TreeNursery.Xaml.Garden}
⊟ Trees	Count = 3
⊟ [0]	{TreeNursery.Xaml.Tree}
⊟ Fruit	{Apple}
⊞ [TreeNursery.Xaml.Apple]	{Apple}
⊟ [1]	{TreeNursery.Xaml.Tree}
⊟ Fruit	{Apple}
⊞ [TreeNursery.Xaml.Apple]	{Apple}
⊟ [2]	{TreeNursery.Xaml.Tree}
⊟ Fruit	{Apricot}
⊞ [TreeNursery.Xaml.Apricot]	{Apricot}
⊞ Raw View	

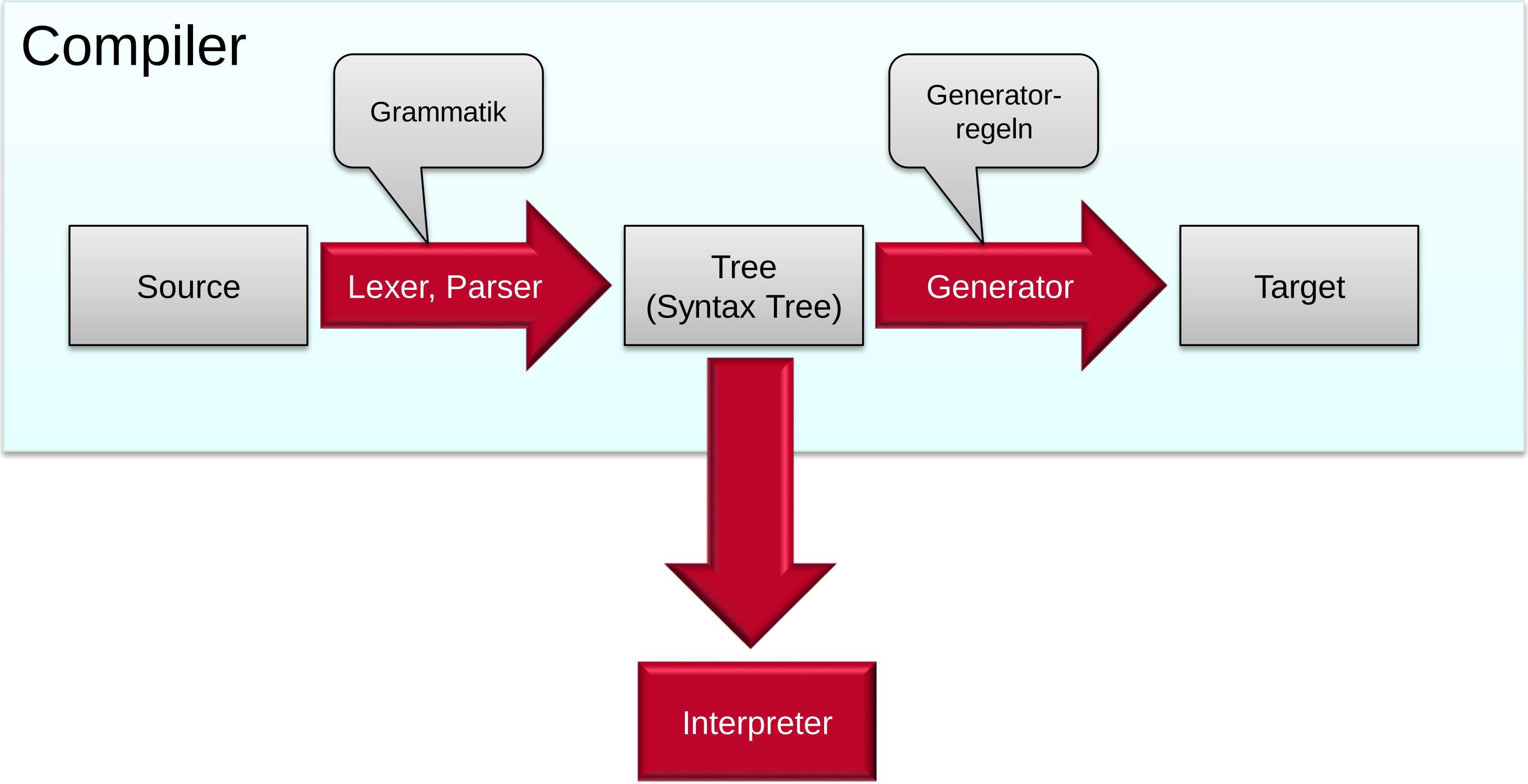
Von Text zum Baum



Einige Beispiele

- Lexer/Parser
  - XML in DOM
  - SQL in Execution Plan
- Compiler bzw. Lexer/Parser/Generator
  - C# in IL
  - FetchXML in SQL (MS CRM)
- Interpreter
  - SQL Server Execution Plan
- Compiler-Compiler
  - ANTLR
  - Coco/R

Wichtige Begriffe



Compiler

Grammatik

Generator-regeln

Source

Lexer, Parser

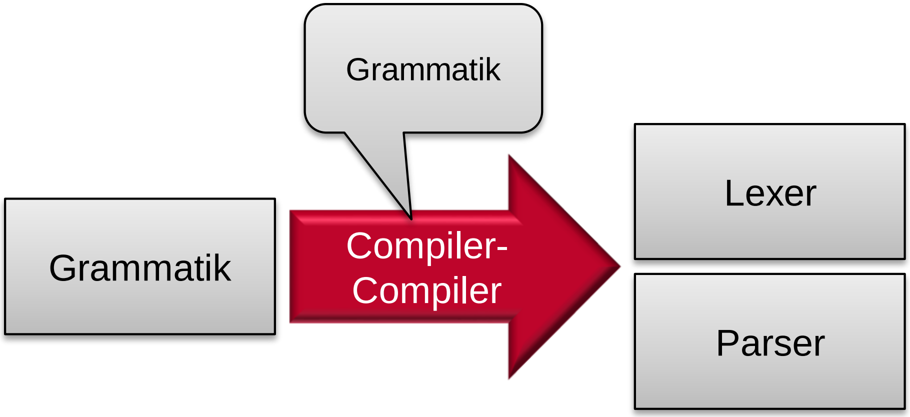
Tree (Syntax Tree)

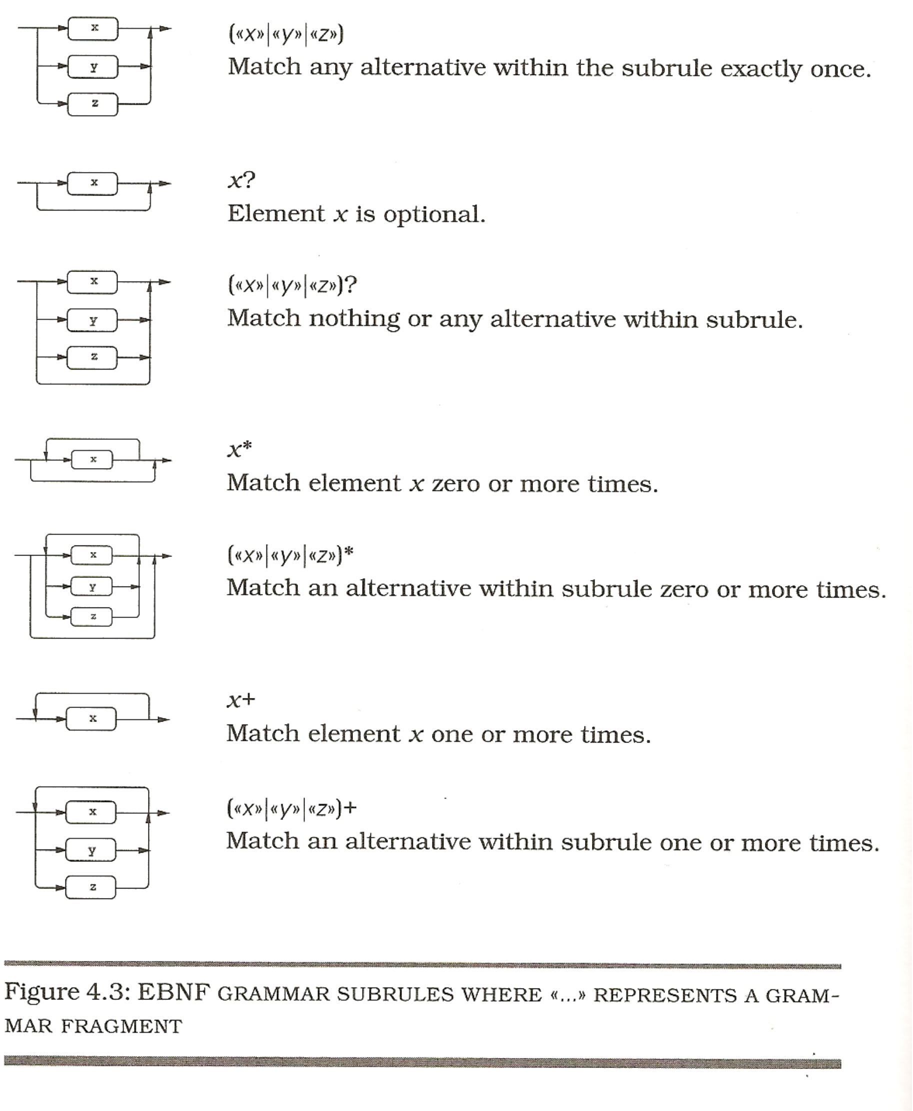
Generator

Target

Interpreter

Wichtige Begriffe





(«x»|«y»|«z»)
Match any alternative within the subrule exactly once.

x?
Element *x* is optional.

(«x»|«y»|«z»)?
Match nothing or any alternative within subrule.

*x**
Match element *x* zero or more times.

(«x»|«y»|«z»)*
Match an alternative within subrule zero or more times.

x+
Match element *x* one or more times.

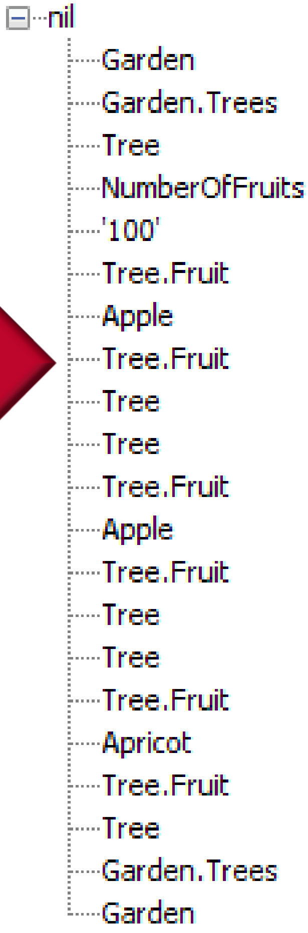
(«x»|«y»|«z»)+
Match an alternative within subrule one or more times.

Figure 4.3: EBNF GRAMMAR SUBRULES WHERE «...» REPRESENTS A GRAMMAR FRAGMENT

Quelle: The Definitive ANTLR Reference, Terence Parr

Praktisches Beispiel

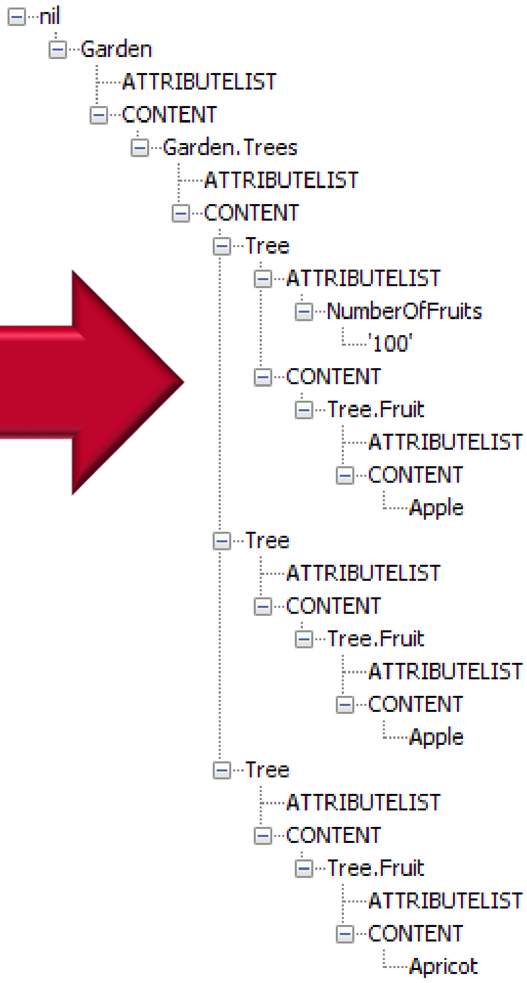
```
grammar XmlLanguage2;
options { output = AST; }

// PARSER ----------------------------------
xmlDocument : node;
node
    : '<'! ELEMENTNAME attributeList '>'!
        ( node )*
      '</'! ELEMENTNAME '>'!
    | '<'! ELEMENTNAME '/>'!;

attributeList : attribute*;
attribute : ELEMENTNAME '='! LITERAL;

// LEXER -----------------------------------
ELEMENTNAME
    : IDENTIFIER ( '.' IDENTIFIER )?;
LITERAL
    : '\'' ( ~'\'' )* '\'';
fragment IDENTIFIER
    : ( 'a'..'z' | 'A'..'Z' | '_' ) ( 'a'..'z' | 'A'..'Z' | '0'..'9' )*;
NEWLINE
    : ('\r'? '\n')+ { $channel = HIDDEN; };
WHITESPACE
    : ( '\t' | ' ' )+ { $channel = HIDDEN; } ;
```



```
⊟ nil
    Garden
    Garden.Trees
    Tree
    NumberOfFruits
    '100'
    Tree.Fruit
    Apple
    Tree.Fruit
    Tree
    Tree
    Tree.Fruit
    Apple
    Tree.Fruit
    Tree
    Tree
    Tree.Fruit
    Apricot
    Tree.Fruit
    Tree
    Garden.Trees
    Garden
```

Praktisches Beispiel

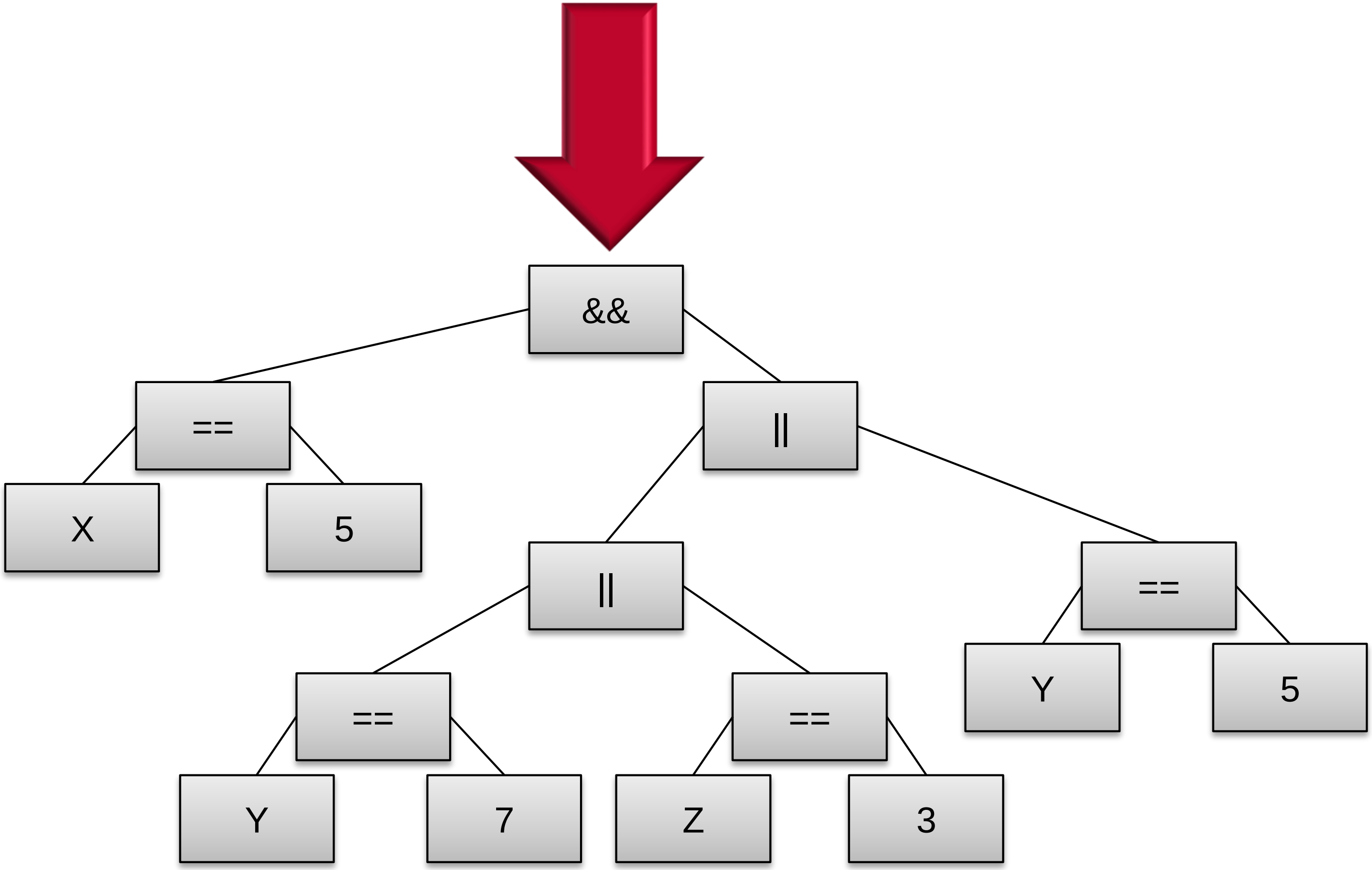
```
grammar XmlLanguage;
options { output = AST; }
tokens {
    NODE = 'Node';
    ATTRIBUTELIST = 'AttributeList';
    ATTRIBUTE = 'Attribute';
    CONTENT = 'CONTENT';
}

// PARSER ----------------------------------
xmlDocument
    : node;
node
    : '<' start=ELEMENTNAME attributeList '>' ( node )*
      '</' end=ELEMENTNAME '>'
        -> ^( NODE [$start] attributeList ^( CONTENT node* ) )
    | '<' tag=ELEMENTNAME '/>'
        -> ^( NODE [$tag] );
attributeList
    : attribute*
        -> ^( ATTRIBUTELIST attribute* );
attribute
    : attribName=ELEMENTNAME '=' LITERAL
        -> ^( ATTRIBUTE [$attribName] LITERAL );

// LEXER ------------------------------------
[…]
```



```
nil
  Garden
    ATTRIBUTELIST
    CONTENT
      Garden.Trees
        ATTRIBUTELIST
        CONTENT
          Tree
            ATTRIBUTELIST
              NumberOfFruits
                '100'
            CONTENT
              Tree.Fruit
                ATTRIBUTELIST
                CONTENT
                  Apple
          Tree
            ATTRIBUTELIST
            CONTENT
              Tree.Fruit
                ATTRIBUTELIST
                CONTENT
                  Apple
          Tree
            ATTRIBUTELIST
            CONTENT
              Tree.Fruit
                ATTRIBUTELIST
                CONTENT
                  Apricot
```

Wo ist der Baum?

```
X=5 And ( Y=7 Or Z=3 Or Y=5 )
```

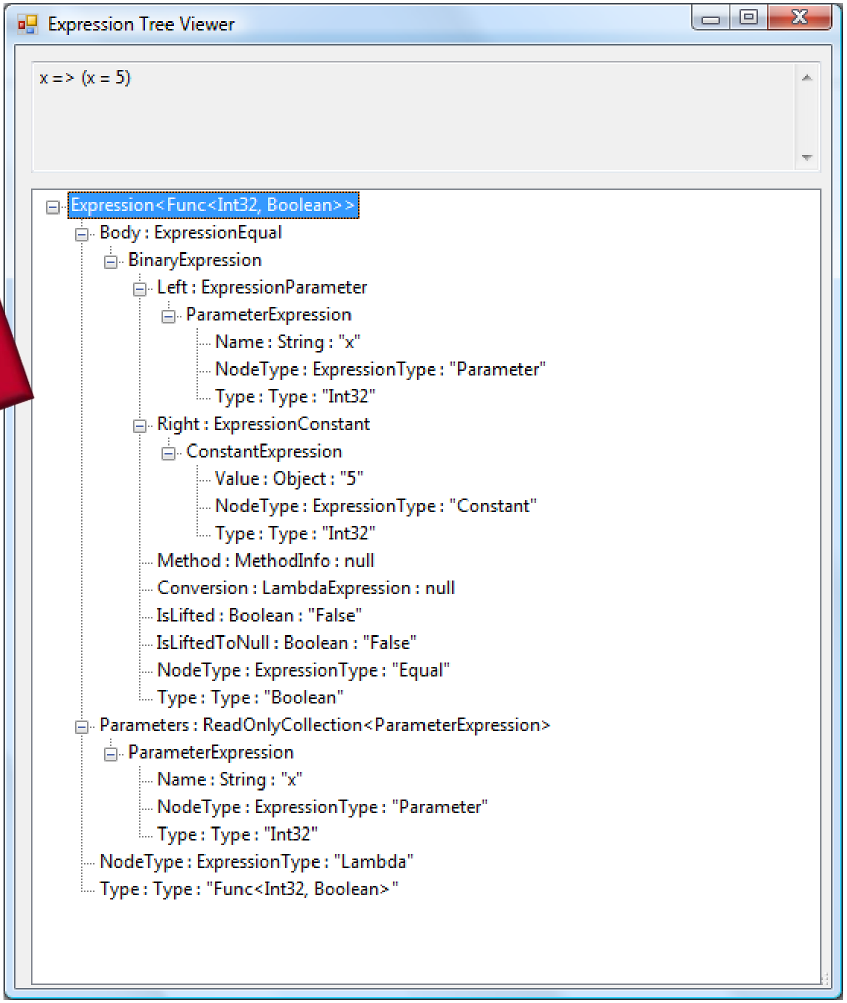


Microsoft Expression Trees

AST IN C#

ExpressionTrees in C#

```
Func<int, bool> f =
    (x) => x==5;

Expression<Func<int, bool>> ex =
    (x) => x == 5;
```



Expression Tree Viewer

```
x => (x = 5)
```

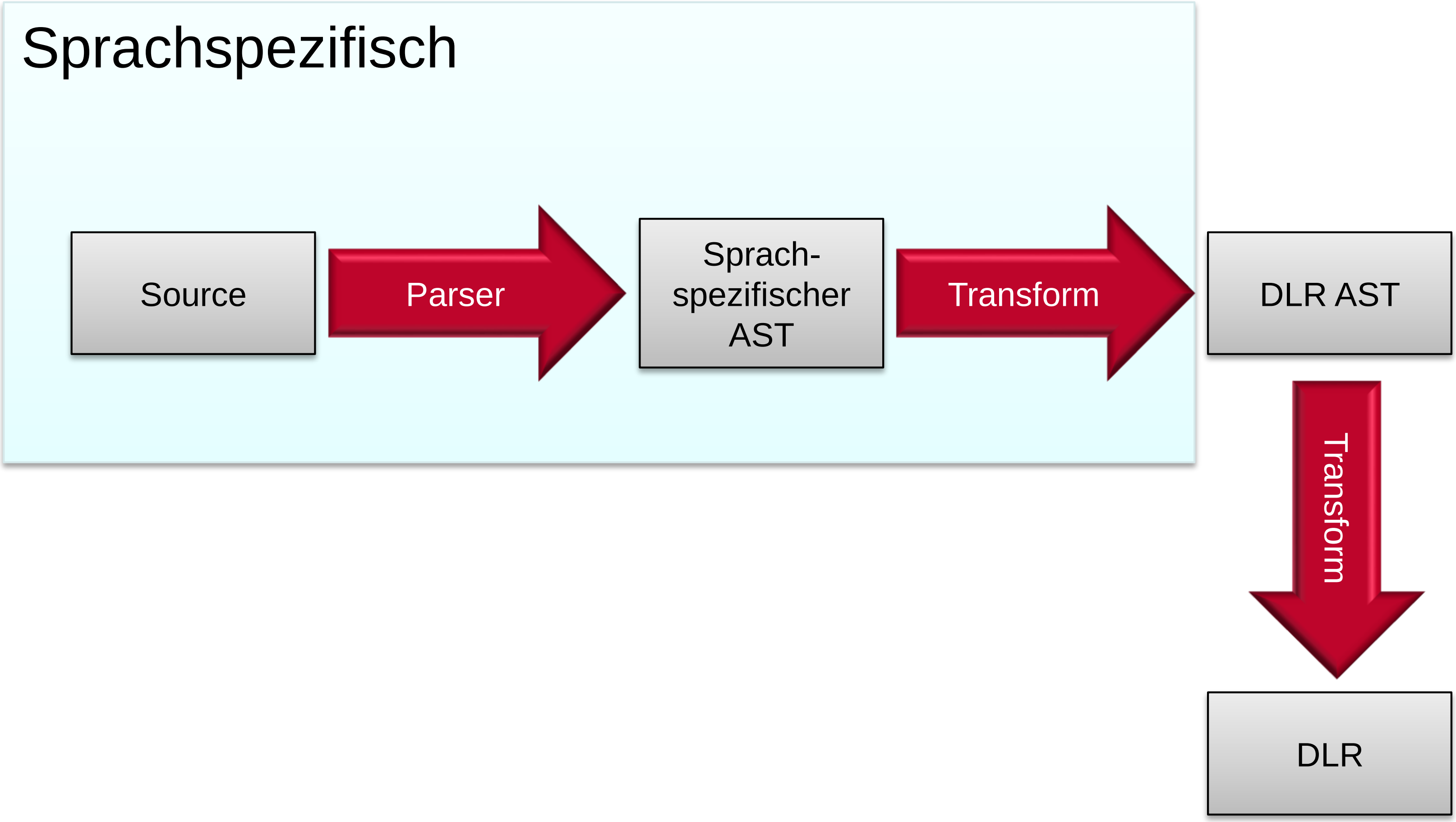
```
Expression<Func<Int32, Boolean>>
    Body : ExpressionEqual
        BinaryExpression
            Left : ExpressionParameter
                ParameterExpression
                    Name : String : "x"
                    NodeType : ExpressionType : "Parameter"
                    Type : Type : "Int32"
            Right : ExpressionConstant
                ConstantExpression
                    Value : Object : "5"
                    NodeType : ExpressionType : "Constant"
                    Type : Type : "Int32"
            Method : MethodInfo : null
            Conversion : LambdaExpression : null
            IsLifted : Boolean : "False"
            IsLiftedToNull : Boolean : "False"
            NodeType : ExpressionType : "Equal"
            Type : Type : "Boolean"
    Parameters : ReadOnlyCollection<ParameterExpression>
        ParameterExpression
            Name : String : "x"
            NodeType : ExpressionType : "Parameter"
            Type : Type : "Int32"
    NodeType : ExpressionType : "Lambda"
    Type : Type : "Func<Int32, Boolean>"
```

Expression Trees in C#

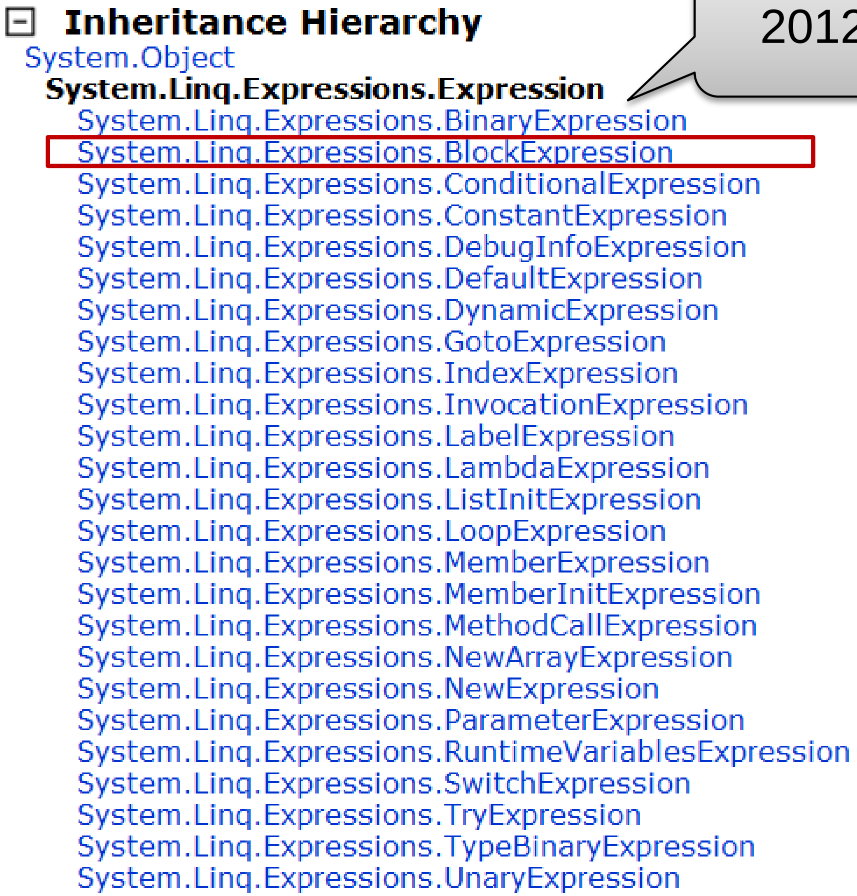
```
private static void Main(string[] args)
{
    Func<int, bool> f;
    Expression<Func<int, bool>> ex;
    […]

    ex = Expression.Lambda<Func<int, bool>>(
        Expression.Equal(
            CS$0$0000 = Expression.Parameter(typeof(int), "x"),
            Expression.Constant((int) 5, typeof(int))
        ),
        new ParameterExpression[] { CS$0$0000 });

     return;
}
```

Compiler bietet Zugriff auf den Syntax Tree zur Laufzeit

AST in DLR


Sprachspezifisch
Source
Parser
Sprach-spezifischer AST
Transform
DLR AST
Transform
DLR

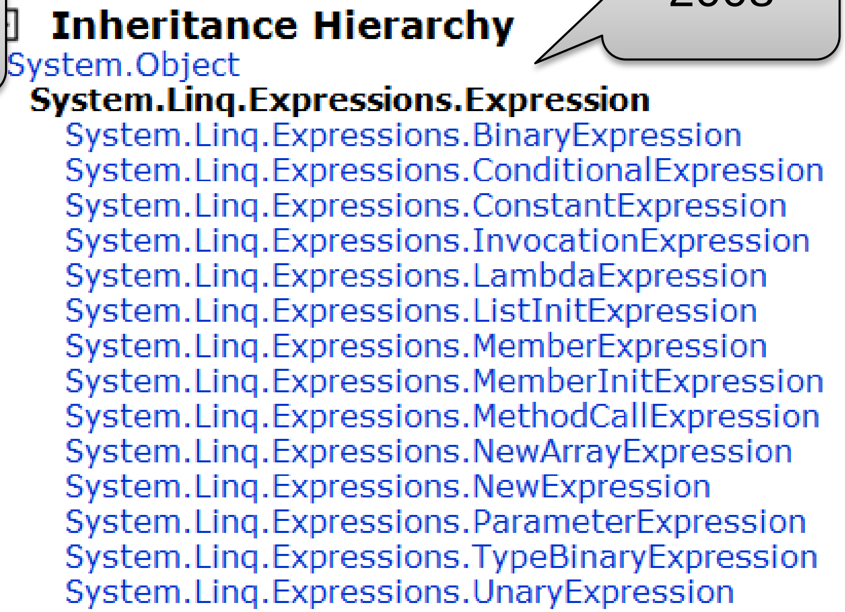
ExpressionTrees in C#



2012

Inheritance Hierarchy

- System.Object
  - **System.Linq.Expressions.Expression**
    - System.Linq.Expressions.BinaryExpression
    - System.Linq.Expressions.BlockExpression
    - System.Linq.Expressions.ConditionalExpression
    - System.Linq.Expressions.ConstantExpression
    - System.Linq.Expressions.DebugInfoExpression
    - System.Linq.Expressions.DefaultExpression
    - System.Linq.Expressions.DynamicExpression
    - System.Linq.Expressions.GotoExpression
    - System.Linq.Expressions.IndexExpression
    - System.Linq.Expressions.InvocationExpression
    - System.Linq.Expressions.LabelExpression
    - System.Linq.Expressions.LambdaExpression
    - System.Linq.Expressions.ListInitExpression
    - System.Linq.Expressions.LoopExpression
    - System.Linq.Expressions.MemberExpression
    - System.Linq.Expressions.MemberInitExpression
    - System.Linq.Expressions.MethodCallExpression
    - System.Linq.Expressions.NewArrayExpression
    - System.Linq.Expressions.NewExpression
    - System.Linq.Expressions.ParameterExpression
    - System.Linq.Expressions.RuntimeVariablesExpression
    - System.Linq.Expressions.SwitchExpression
    - System.Linq.Expressions.TryExpression
    - System.Linq.Expressions.TypeBinaryExpression
    - System.Linq.Expressions.UnaryExpression



2008

Inheritance Hierarchy

- System.Object
  - **System.Linq.Expressions.Expression**
    - System.Linq.Expressions.BinaryExpression
    - System.Linq.Expressions.ConditionalExpression
    - System.Linq.Expressions.ConstantExpression
    - System.Linq.Expressions.InvocationExpression
    - System.Linq.Expressions.LambdaExpression
    - System.Linq.Expressions.ListInitExpression
    - System.Linq.Expressions.MemberExpression
    - System.Linq.Expressions.MemberInitExpression
    - System.Linq.Expressions.MethodCallExpression
    - System.Linq.Expressions.NewArrayExpression
    - System.Linq.Expressions.NewExpression
    - System.Linq.Expressions.ParameterExpression
    - System.Linq.Expressions.TypeBinaryExpression
    - System.Linq.Expressions.UnaryExpression

Pythondatei ausführen

```
// Execute the script and give it access the the ERP's API
var engine = Python.CreateEngine();
var scope = engine.CreateScope();
scope.SetVariable("Context", context);
var script = engine.CreateScriptSourceFromString(scriptSource);
script.Execute(scope);
```

Pythondatei ausführen

```
var engine = Python.CreateEngine();
using (var stream = new ScriptOutputStream( s => {
        this.AppendToScriptOutput(s);
        App.Current.Dispatcher.BeginInvoke(
            new Action(() => this.OnPropertyChanged("ScriptOutput")));
    }, Encoding.UTF8))
{
    engine.Runtime.IO.SetOutput(stream, Encoding.UTF8);
    var scriptSource = engine.CreateScriptSourceFromFile("SampleScript01.py");
    try
    {
        scriptSource.Execute();
    }
    catch (SyntaxErrorException e)
    {
        this.AppendToScriptOutput("Syntax error (line {0}, column {1}): {2}",
            e.Line, e.Column, e.Message);
        App.Current.Dispatcher.BeginInvoke(
            new Action(() => this.OnPropertyChanged("ScriptOutput")));
    }
}
```

Exkurs: ScriptOutputStream

```
public sealed class ScriptOutputStream : Stream
{
    public ScriptOutputStream(Action<string> write, Encoding encoding)
    {
        […]
        chunks = new BlockingCollection<byte[]>();
        this.processingTask = Task.Factory.StartNew(() => {
                foreach (var chunk in chunks.GetConsumingEnumerable()) {
                    write(this.encoding.GetString(chunk));
                }
            }, TaskCreationOptions.LongRunning);
    }
    public override void Write(byte[] buffer, int offset, int count)
    {
        var chunk = new byte[count];
        Buffer.BlockCopy(buffer, offset, chunk, 0, count);
        this.chunks.Add(chunk);
    }
    public override void Close()
    {
        this.chunks.CompleteAdding();
        try { this.processingTask.Wait(); }
        finally { base.Close(); }
    }
    […]
}
```

Beispielscript in Python

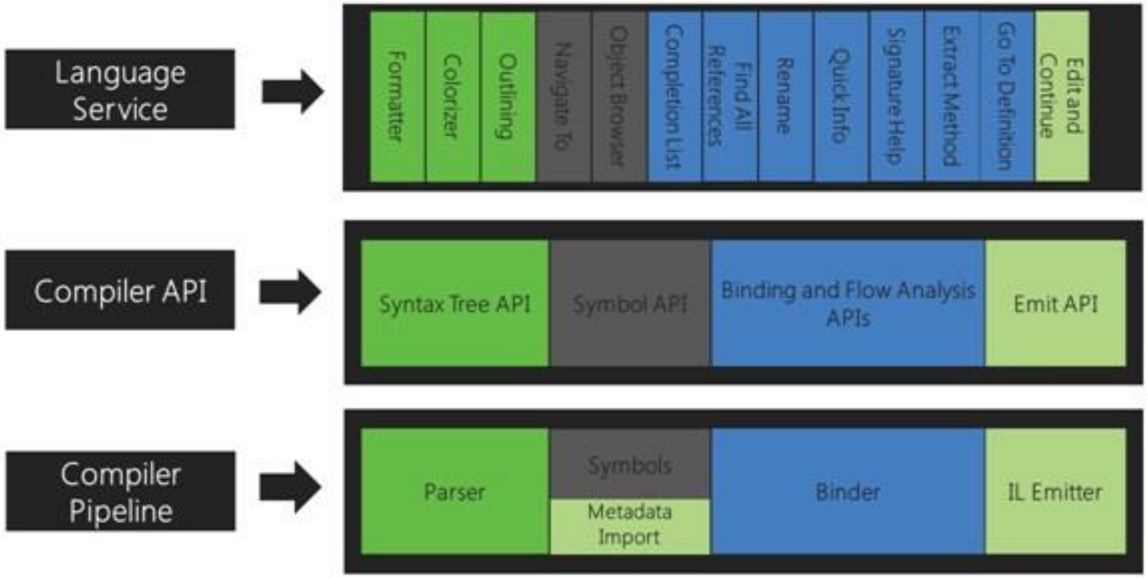
```
import clr
clr.AddReference("mscorlib")

from System.Threading import Thread

for i in range(0, 10):
  print str(i+1)
  Thread.Sleep(500)

print "Done!"
```

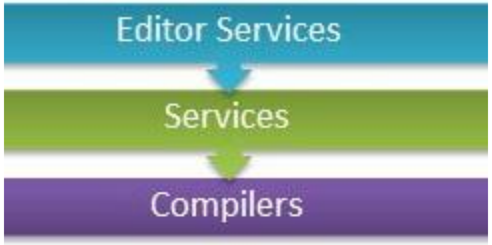
Referenzen auf Assemblies

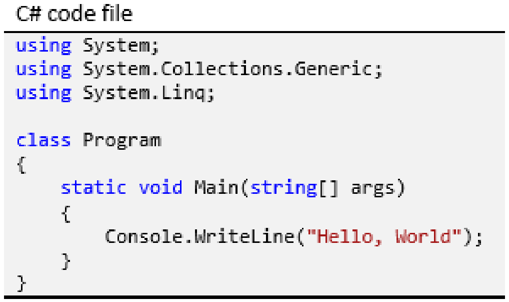
~using

Methode aus dem .NET Framework

Roslyn Architektur

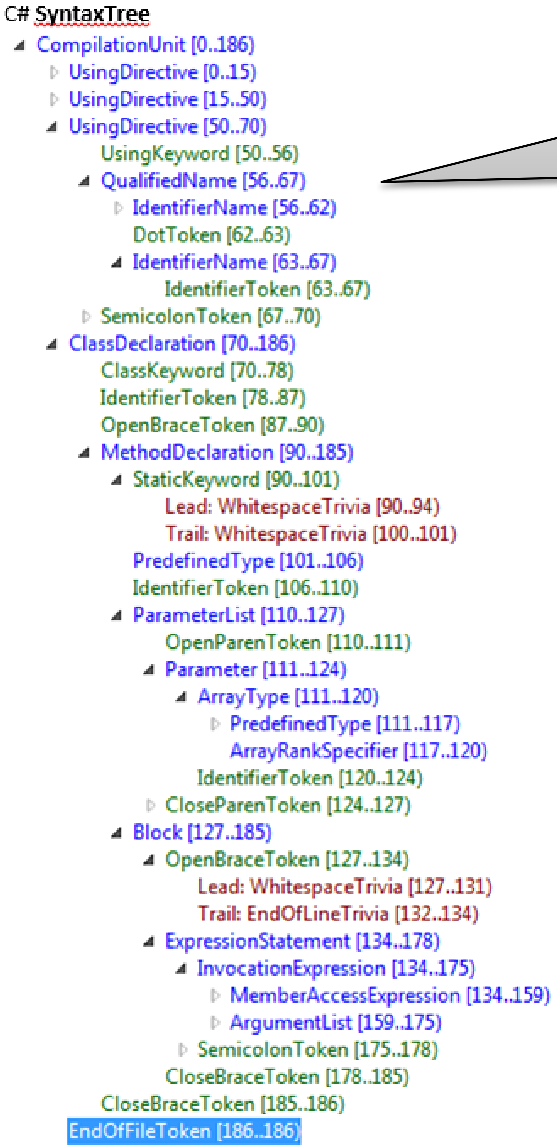






C# code file

```
using System;
using System.Collections.Generic;
using System.Linq;

class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Hello, World");
    }
}
```



C# SyntaxTree

```
◢ CompilationUnit [0..186)
    ▷ UsingDirective [0..15)
    ▷ UsingDirective [15..50)
    ◢ UsingDirective [50..70)
        UsingKeyword [50..56)
      ◢ QualifiedName [56..67)
          ▷ IdentifierName [56..62)
            DotToken [62..63)
          ◢ IdentifierName [63..67)
                IdentifierToken [63..67)
      ▷ SemicolonToken [67..70)
    ◢ ClassDeclaration [70..186)
        ClassKeyword [70..78)
        IdentifierToken [78..87)
        OpenBraceToken [87..90)
      ◢ MethodDeclaration [90..185)
          ◢ StaticKeyword [90..101)
                Lead: WhitespaceTrivia [90..94)
                Trail: WhitespaceTrivia [100..101)
            PredefinedType [101..106)
            IdentifierToken [106..110)
          ◢ ParameterList [110..127)
                OpenParenToken [110..111)
              ◢ Parameter [111..124)
                  ◢ ArrayType [111..120)
                      ▷ PredefinedType [111..117)
                        ArrayRankSpecifier [117..120)
                    IdentifierToken [120..124)
              ▷ CloseParenToken [124..127)
          ◢ Block [127..185)
              ◢ OpenBraceToken [127..134)
                    Lead: WhitespaceTrivia [127..131)
                    Trail: EndOfLineTrivia [132..134)
              ◢ ExpressionStatement [134..178)
                  ◢ InvocationExpression [134..175)
                      ▷ MemberAccessExpression [134..159)
                      ▷ ArgumentList [159..175)
                  ▷ SemicolonToken [175..178)
                CloseBraceToken [178..185)
        CloseBraceToken [185..186)
    EndOfFileToken [186..186)
```

Roslyn Syntax Tree
(roundtrippable)