

1.– 4. September 2014
in Nürnberg



Herbstcampus

Wissenstransfer
par excellence

Nie wieder ohne

Libs ohne die ich nicht mehr programmieren will

Tim Bourguignon

MATHEMA Software GmbH

▼ Nie wieder ohne

Libs ohne die ich nicht mehr programmieren will

- ▼ **Timothée Bourguignon**
 - ▼ Senior / Lead Developer
 - ▼ Consultant
 - ▼ Trainer

- ▼ **MATHEMA Software GmbH**
 - ▼ Erlangen
 - ▼ www.mathema.de



- ▼ How do corporations think?
 - ▼ Frameworks
 - ▼ Small libs
 - ▼ OpenSource Libs
 - ▼ Self-made
- ▼ OpenSource at Microsoft?
- ▼ Barefoot coding?

- ▼ Too much talk about the big projects while there are world class (small) libraries moving things forward

▼ Autofac

▼ RhinoMocks

▼ ServiceStack

▼ Ninject



timothep
@timothep

I'm searching for those .NET libs you wouldn't want to live without anymore... any idea? (Please RT)

▼ FakeItEasy

▼ Json.Net

▼ NSubstitute

▼ Tinyloc

▼ JsonFx

▼ Moq

▼ NancyFx

▼ RestSharp

- ▼ Autofac
- ▼ AutoPoco
- ▼ CsQuery
- ▼ Dapper
- ▼ EmitMapper
- ▼ FakeItEasy
- ▼ Json.Net
- ▼ JsonFx
- ▼ Moq
- ▼ NancyFx
- ▼ NDatabase
- ▼ NEST
- ▼ Ninject
- ▼ NLog
- ▼ NodaTime
- ▼ Nowin
- ▼ NSubstitute
- ▼ Owin.*
- ▼ Purify
- ▼ RestSharp
- ▼ RhinoMocks
- ▼ ServiceStack
- ▼ SimpleAuthentication
- ▼ SimpleValidator
- ▼ Spring.Rest
- ▼ T4
- ▼ Tinyloc
- ▼ Windsor

- ▼ NodaTime
- ▼ Json.NET & JsonFx
- ▼ RestSharp
- ▼ TinyloC

▼ NodaTime



- ▼ Port of JodaTime (Java)
 - ▼ Author: Jon Skeets
 - ▼ www.nodatime.org
 - ▼ Nuget: nodatime
 - ▼ License Apache 2.0

- ▼ Two classes to rule them all
 - ▼ DateTime
 - ▼ TimeSpan
- ▼ How to express:
 - ▼ A Date only?
 - ▼ „May 21st“
 - ▼ A Time period?
 - ▼ „May 21st – 23rd 2014“
- ▼ Works but...



- ▼ An „instant“ on the global timeline since the Unix epoch
 - ▼ Since „Jan. 1, 1970 Midnight UTC“
- ▼ 10,000 ticks in a millisecond

```
var now = SystemClock.Instance.Now;  
Console.WriteLine(now.Ticks); // 14001007108796083
```

- ▼ Part of a local date/time
 - ▼ Not enough to represent a specific instant in time
 - ▼ For anyone who has had problems mixing date-only and date-time values, the `LocalDate` type will be appreciated

```
var myBirthday = new LocalDate(1983, 04, 19);  
var noon = new LocalTime(12,0,0);
```

▼ One global DateTime

```
var now = SystemClock.Instance.Now;  
var dtzp = DateTimeZoneProviders.Tzdb;  
var berlinTz = dtzp["Europe/Berlin"];  
var berlinNow = new ZonedDateTime(now, berlinTz);  
//2014-05-08T22:42:08 Europe/Berlin (+02)
```

- ▼ “Offset” „+02“ depends on what country you’re in, what calendar you’re using, the DaylightSavingTime etc.

▼ Specific time interval, retains start and end instants

```
// 2014-07-15T06:00:00Z - 2014-07-15T17:00:00Z
var tz = DateTimeZoneProviders.Tzdb.GetSystemDefault();
var localBeginDateTime = LocalDateTime.FromDateTime(
    new DateTime(2014, 07, 15, 8, 0, 0));
ZonedDateTime zonedBeginDateTime =
    localBeginDateTime.InZoneStrictly(tz);
// Same for endTime...
var interval = new Interval( zonedBeginDateTime.ToInstant(),
    zonedEndDateTime.ToInstant());
```

- ▼ Time period defined in terms of fields
- ▼ Inexact in milliseconds (unlike Duration)

```
var fortieth = new LocalDate(2023, 4, 19);  
var today = berlinNow.LocalDateTime.Date;  
  
//P3257D  
Period period = Period.Between(  
    today, fortieth, PeriodUnits.Days);  
  
//P8Y11M  
Period between = Period.Between(  
    today, fortieth, PeriodUnits.YearMonthDay);
```

- ▼ Date and time manipulations are horribly complex
- ▼ Video: „The problem with Time&Timezones“ (Computerphile)
 - ▼ <https://www.youtube.com/watch?v=-5wpm-gesOY>
- ▼ „DateTime“ & „TimeSpan“ do not make it easier to manipulate
- ▼ NodaTime is more verbose, but decouples the concepts for more clarity

▼ **Json.NET & JsonFx**

▼ Json Manipulation Libraries

▼ Json.NET

- ▼ <http://james.newtonking.com/json>
- ▼ Author: Newtonsoft
- ▼ Nuget: Newtonsoft.Json
- ▼ License MIT



▼ JsonFx

- ▼ <https://github.com/jsonfx/jsonfx>
- ▼ Author: Stephen M. McKamey
- ▼ Nuget: JsonFx
- ▼ License MIT



- ▼ Most downloaded on Nuget
- ▼ Jack-of-all-trade
- ▼ Json and .NET hand-in-hand

▼ Just serialize

```
Product product = new Product();
product.Name = "Apple";
product.ExpiryDate = new DateTime(2008, 12, 28);
product.Price = 3.99M;
product.Sizes = new string[] { "Small", "Medium", "Large" };

string output = JsonConvert.SerializeObject(product);

//{
//  "Name": "Apple",
//  "ExpiryDate": "2008-12-28T00:00:00",
//  "Price": 3.99,
//  "Sizes": [ "Small",
//            "Medium",
//            "Large" ]
//}
```

▼ Just deserialize

```
string json = @"{
    'Name': 'Bad Boys',
    'ReleaseDate': '1995-4-7T00:00:00',
    'Genres': [ 'Action', 'Comedy' ]
}";
Movie m = JsonConvert.DeserializeObject<Movie>(json);
string name = m.Name; // Bad Boys
```

▼ JsonSerializerSettings

```
const string json = @"{ ""Date"" : ""09/12/2013"" }";
var obj = JsonConvert.DeserializeObject<MyObject>(json,
    new IsoDateTimeConverter { DateTimeFormat = "dd/MM/yyyy" });
DateTime date = obj.Date;
```

▼ Dynamic Style

```
JObject jobj = JObject.Parse(@"{
    'CPU': 'Intel',
    'Drives': ['DVD read/writer', '500 gigabyte hard drive']
}");

Assert.AreEqual("Intel", jobj["CPU"]);

// JObject of type IEnumerable<KeyValuePair<string, Jtoken>>
```

Simply validate

```
const string schemaJson = @"{
  'description': 'A person', 'type': 'object',
  'properties': {
    'name': {'type': 'string'},
    'hobbies': {
      'type': 'array',
      'items': {'type': 'string'}
    }
  }
}";
JsonSchema schema = JsonSchema.Parse(schemaJson);
JObject person = JObject.Parse(@"{ 'name': 'James',
  'hobbies': ['.NET', 'Reading', 'Xbox', 'LOLCATS' ]}");
Assert.IsTrue(person.IsValid(schema));
```

Schema generation

▶ <http://sixgun.wordpress.com/2012/02/09/using-json-net-to-generate-jsonschema/>

- ▼ Not as good
- ▼ Not as complete
- ▼ One killer feature
 - ▼ Deserialize to „dynamic“

▼ Remember this Json.NET example?

```
JObject jobj = JObject.Parse(@"{
    'CPU': 'Intel', 'Drives': [ 'DVD read/writer',
                                '500 gigabyte hard drive' ] }");

Assert.AreEqual("Intel", jobj["CPU"]);
```

▼ ... here it is with JsonFx

```
var reader = new JsonFx.Json.JsonReader();

dynamic output = reader.Read(@"{ 'CPU': 'Intel',
    'Drives': [ 'DVD read/writer', '500 gigabyte hard drive' ]}");

Assert.AreEqual(output.CPU, "Intel");
Assert.AreEqual(output.Drives[0], "DVD read/writer");
```

- ▼ Json.NET
 - ▼ Solid library
 - ▼ Is defacto the .NET standard

- ▼ JsonFx
 - ▼ (At least) one killer feature

- ▼ Who said Json wasn't made for C#?

▼ RESTSharp

- ▼ Simple REST and HTTP API Client for .NET
 - ▼ <http://restsharp.org/>
 - ▼ Nuget: RestSharp
 - ▼ Author: John Sheehan
 - ▼ Current maintener: Phil Haack (and searching for a replacement)
 - ▼ License Apache 2.0



▼ The standard way using the BCL

```
WebRequest request = WebRequest.Create(url);
WebResponse response = request.GetResponse();
Stream dataStream = response.GetResponseStream();
StreamReader reader = new StreamReader(dataStream);
string responseFromServer = reader.ReadToEnd();
Console.WriteLine(responseFromServer);
reader.Close();
response.Close();
```

▼ Artificially complicated

- ▼ Stream manipulations
- ▼ „Transactions“

- ▼ Specialized objects
 - ▼ RestClient
 - ▼ RestRequest

```
var client = new RestClient("url");  
var request = new RestRequest("/", Method.GET);
```

▼ Manipulate the request parameters

```
// adds to POST or URL querystring based on Method
request.AddParameter("name", "value");

// replaces matching token in request.Resource
request.AddUrlSegment("id", "123");

// add parameters for all properties on an object
request.AddObject(new MyInt{myInt = 42});

// add files to upload (works with compatible verbs)
// may throw a FileNotFoundException
request.AddFile("MyFile", "path");
```

▼ Headers

```
// easily add HTTP Headers
request.AddHeader("header", "value");
```

▼ Synchron

```
var client = new RestClient("http://example.com");  
  
// execute the request  
IRestResponse response = client.Execute(request);  
var content = response.Content; // raw content as string
```

▼ Asynchron

```
// easy async support  
var asyncHandle = client.ExecuteAsync(request, response =>  
    Console.WriteLine(response.Content));  
  
asyncHandle.Abort();
```


▼ Deserialize on the fly

```
var client = new RestClient("url");
var request = new RestRequest("/", Method.GET);

// return content type is sniffed
// but can be explicitly set via RestClient.AddHandler();
IRestResponse<Person> response = client.Execute<Person>(request);
var name = response.Data.Name;
```

▼ With async as well

```
// async with deserialization
var asyncHandle = client.ExecuteAsync<Person>(request, resp =>
    Console.WriteLine(resp.Data.Name));
```

▼ Direct download

```
var client = new RestClient("url");  
var request = new RestRequest("/", Method.GET);  
  
client.DownloadData(request).SaveAs("path");
```

▼ Streaming

```
string tempFile = Path.GetTempFileName();  
using (var writer = File.OpenWrite(tempFile))  
{  
    var client = new RestClient("baseUrl");  
    var request = new RestRequest("Assets/LargeFile.7z");  
    request.ResponseWriter = (responseStream) =>  
        responseStream.CopyTo(writer);  
    var response = client.DownloadData(request);  
}
```

- ▼ For .NET 4.5 (and greater) only
 - ▼ Leaner, RestSharp-y syntax

```
var client = new HttpClient();  
var response = await client.GetAsync("url");  
var content = response.Content;  
var result = await content.ReadAsStringAsync();
```

- ▼ Why RESTSharp when there's HttpClient?
 - ▼ „Very good question! System.Net.HttpClient is only available for .NET 4.5. There's the Portable Class Library (PCL) version, but that is encumbered by silly platform restrictions."
Phill Haack
 - ▼ Those (platform restrictions) are slowly being solved...
- ▼ Still a very solid library
- ▼ Life saver in many occasions

▼ TinyloC

- ▼ „An easy to use, hassle free, Inversion of Control Container for small projects, libraries and beginners alike“
 - ▼ <https://github.com/grumpydev/TinyIoC>
 - ▼ Deployed as a single „.cs“ file (no Nuget)
 - ▼ Author: Steven Robbins (@GrumpyDev)
 - ▼ Supports Windows, Mono, MonoTouch, PocketPC, Windows Phone 7 and MonoDroid

- ▼ Cornerstone of the Nancy Framework (nancyfx.org)

- „If you have an object that interacts with other objects, the responsibility of finding a reference to those objects should be moved outside of the object itself.“

```
public DateTime XDaysFromNow(int days)
{
    var now = DateTime.Now;
    return now.AddDays(days);
}
```



```
public DateTime AddDays(int days, DateTime date)
{
    return date.AddDays(days);
}
// You can still use DateTime.Now in the call,
// but the dependency is now outside
this.AddDays(2, DateTime.Now);
```

- z.B. Factory

▼ Retrieve the container

```
// Lazy singleton as container  
var container = TinyIoCContainer.Current;
```

▼ No registration required for concrete types

```
// Creates an instance of MyConcreteType or make it singleton  
var instance = container.Resolve<MyConcreteType>();  
container.Register<MyConcreteType>().AsSingleton();
```

▼ Explicit autoregister

```
// Register all concrete types and interfaces  
container.AutoRegister();  
  
// Creates an instance of the only type  
// implementing IMyInterface  
var implementation = container.Resolve<IMyInterface>();
```



```
public interface ItoInject {}
public class ToInject: ItoInject{}

public class ToBuild
{
    public ToBuild(ItoInject toInject)
    {
    }
}

// Call
var container = TinyIoCContainer.Current;
container.AutoRegister();
var instance = container.Resolve<ToBuild>();
```

▼ Properties are resolved and instantiated

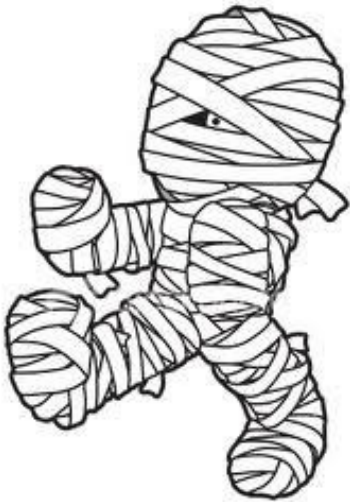
```
internal class TestClassPropertyDependencies
{
    // Will be set if we can resolve and isn't already set
    public ITestInterface Property1 { get; set; }

    // Will be ignored
    public int Property2 { get; set; }

    // Will be set if we can resolve and isn't already set
    public TestClassDefaultCtor ConcreteProperty
        { get; set; }
}

var input = new TestClassPropertyDependencies();
container.BuildUp(input); // Properties are now set
```

- ▼ Often exactly what you need
- ▼ Registration and resolution as simple as it gets
- ▼ No setup required



▼ Wrap Up

▼ NodaTime

- ▼ Paracetamol for your complex time based projects
- ▼ Nothing like what .NET offered until now

▼ Json.NET & JsonFx

- ▼ A solid library tending to become a standard for .NET
- ▼ A small contestant that can have some advantages

▼ RESTSharp

- ▼ A better REST API for .NET
- ▼ Slowly integrated into the Framework

▼ TinyIoC

- ▼ Now you don't have any excuse not to IoC your prototypes

- ▼ Give small libs a chance
- ▼ OpenSource is the key
 - ▼ Producing libs we'll love
 - ▼ Helping the „bigger“ Frameworks improve
- ▼ Where to start?
 - ▼ Blogs, tutorials, videos
 - ▼ Pull Requests
 - ▼ <http://up-for-grabs.net>

QUESTIONS?
SUGGESTIONS?
IDEAS?

THANKS!



- ▼ www.timbourguignon.fr
- ▼ tim.bourguignon@mathema.de
- ▼ Slides: <https://github.com/TimotheP/MarathonWOShoes>