Json.NET and the Future: .NET Core and .NET 5

WRAP UP

Xavier Morera

HELPING DEVELOPERS UNDERSTAND SEARCH & BIG DATA

@xmorera www.xaviermorera.com

{JSON}

The Case for JSON & Json.NET

Serialization Fundamentals

Settings & Attributes

Custom Serialization

Performance Tips

LINQ to JSON

JSON & XML

Binary JSON (BSON)

Json.NET Schema

.NET

.NET Framework

.NET

.NET Framework

Windows

.NET Core

Windows, Linux, and MacOS

.NET

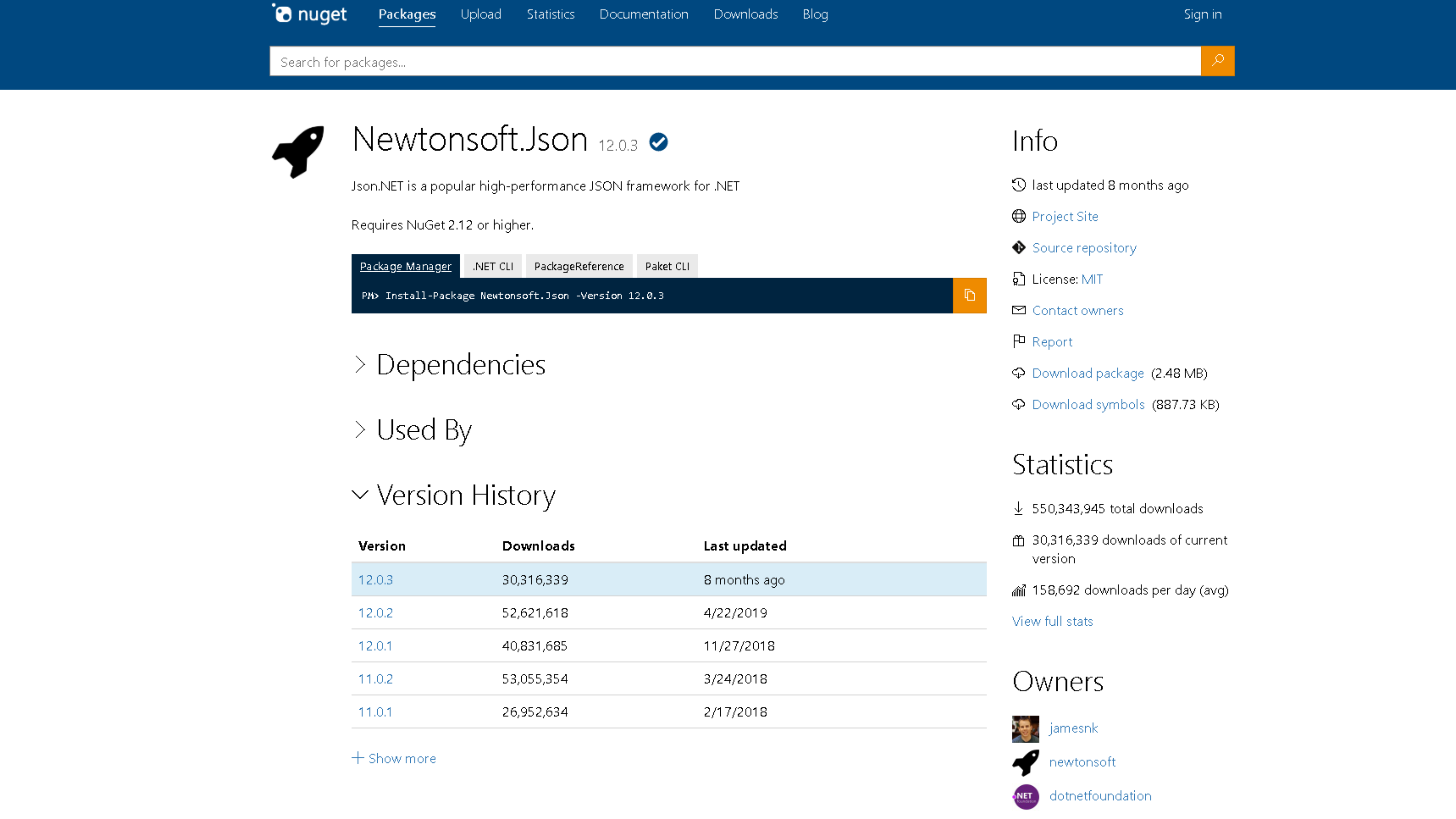
.NET 5

.NET Framework + .NET Core

Windows, Linux, MacOS, Android, IOS...

Demo

Json.NET and the Future: .NET Core and .NET 5



Newtonsoft.Json 12.0.3

Json.NET is a popular high-performance JSON framework for .NET

Requires NuGet 2.12 or higher.

```
PM> Install-Package Newtonsoft.Json -Version 12.0.3
```

Dependencies

Used By

Version History

Version	Downloads	Last updated
12.0.3	30,316,339	8 months ago
12.0.2	52,621,618	4/22/2019
12.0.1	40,831,685	11/27/2018
11.0.2	53,055,354	3/24/2018
11.0.1	26,952,634	2/17/2018

+ Show more

Info

- last updated 8 months ago
- Project Site
- Source repository
- License: MIT
- Contact owners
- Report
- Download package (2.48 MB)
- Download symbols (887.73 KB)

Statistics

- 550,343,945 total downloads
- 30,316,339 downloads of current version
- 158,692 downloads per day (avg)

View full stats

Owners

- jamesnk
- newtonsoft
- dotnetfoundation

Version

.NET Core 3.1

Search

System.Text.Json Namespace

The System.Text.Json namespace provides high-performance, low-allocating, and standards-compliant capabilities to process JavaScript Object Notation (JSON), which includes serializing objects to JSON text and deserializing JSON text to objects, with UTF-8 support built-in. It also provides types to read and write JSON text encoded as UTF-8, and to create an in-memory document object model (DOM) for random access of the JSON elements within a structured view of the data.

Thank you.

In this article

Classes

JsonDocument	Provides a mechanism for examining the structural content of a JSON value without automatically instantiating data values.
JsonException	Defines a custom exception object that is thrown when invalid JSON text is encountered, when the defined maximum depth is passed, or the JSON text is not compatible with the type of a property on an object.
JsonNamingPolicy	Determines the naming policy used to convert a string-based name to another format, such as a camel-casing format.
JsonSerializer	Provides functionality to serialize objects or value types to JSON and to deserialize JSON into objects or value types.
JsonSerializerOptions	Provides options to be used with JsonSerializer.
Utf8JsonWriter	Provides a high-performance API for forward-only, non-cached writing of UTF-8 encoded JSON text.

Structs

JsonDocumentOptions	Provides the ability for the user to define custom behavior when parsing JSON to create a JsonDocument.
JsonElement	Represents a specific JSON value within a JsonDocument.
JsonElement.ArrayEnumerator	Represents an enumerator for the contents of a JSON array.
JsonElement.ObjectEnumerator	Represents an enumerator for the properties of a JSON object.
JsonEncodedText	Provides methods to transform UTF-8 or UTF-16 encoded text into a form that is suitable for

JSON

JSON

System.Text.Json

Released in 2019 with .NET Core 3.0

Provide high-performance JSON APIs

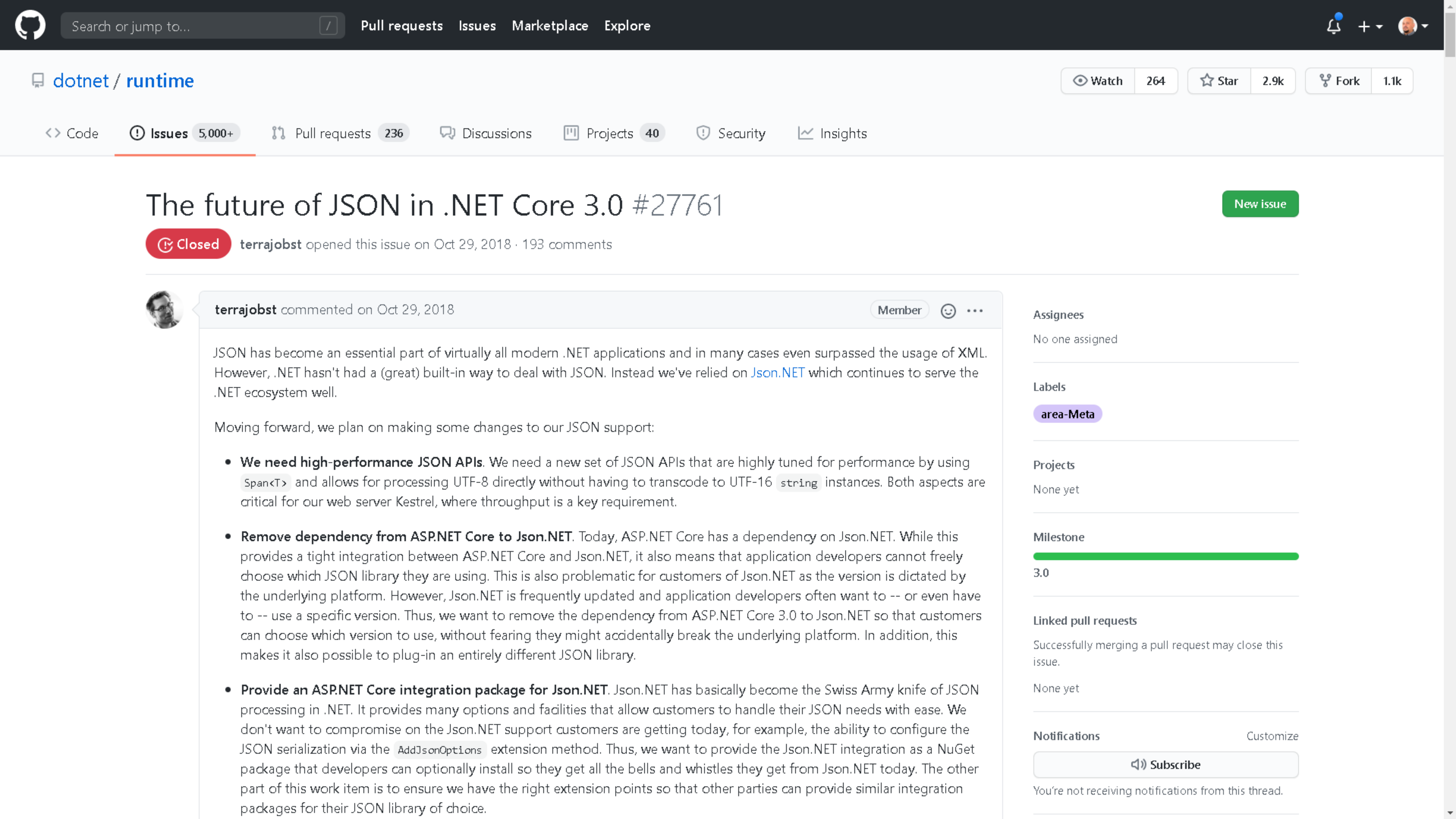
- Use of new functionality, like Span<T>
- Use UTF-8 instead of UTF-16

Remove Json.NET dependency

- From ASP.NET Core
- Moved to a separate package

Read for further information

https://github.com/dotnet/runtime/issues/27761



dotnet / runtime

The future of JSON in .NET Core 3.0 #27761

Closed terrajobst opened this issue on Oct 29, 2018 · 193 comments

terrajobst commented on Oct 29, 2018 Member

JSON has become an essential part of virtually all modern .NET applications and in many cases even surpassed the usage of XML. However, .NET hasn't had a (great) built-in way to deal with JSON. Instead we've relied on Json.NET which continues to serve the .NET ecosystem well.

Moving forward, we plan on making some changes to our JSON support:

- **We need high-performance JSON APIs**. We need a new set of JSON APIs that are highly tuned for performance by using `Span<T>` and allows for processing UTF-8 directly without having to transcode to UTF-16 `string` instances. Both aspects are critical for our web server Kestrel, where throughput is a key requirement.
- **Remove dependency from ASP.NET Core to Json.NET**. Today, ASP.NET Core has a dependency on Json.NET. While this provides a tight integration between ASP.NET Core and Json.NET, it also means that application developers cannot freely choose which JSON library they are using. This is also problematic for customers of Json.NET as the version is dictated by the underlying platform. However, Json.NET is frequently updated and application developers often want to -- or even have to -- use a specific version. Thus, we want to remove the dependency from ASP.NET Core 3.0 to Json.NET so that customers can choose which version to use, without fearing they might accidentally break the underlying platform. In addition, this makes it also possible to plug-in an entirely different JSON library.
- **Provide an ASP.NET Core integration package for Json.NET**. Json.NET has basically become the Swiss Army knife of JSON processing in .NET. It provides many options and facilities that allow customers to handle their JSON needs with ease. We don't want to compromise on the Json.NET support customers are getting today, for example, the ability to configure the JSON serialization via the `AddJsonOptions` extension method. Thus, we want to provide the Json.NET integration as a NuGet package that developers can optionally install so they get all the bells and whistles they get from Json.NET today. The other part of this work item is to ensure we have the right extension points so that other parties can provide similar integration packages for their JSON library of choice.

Assignees
No one assigned

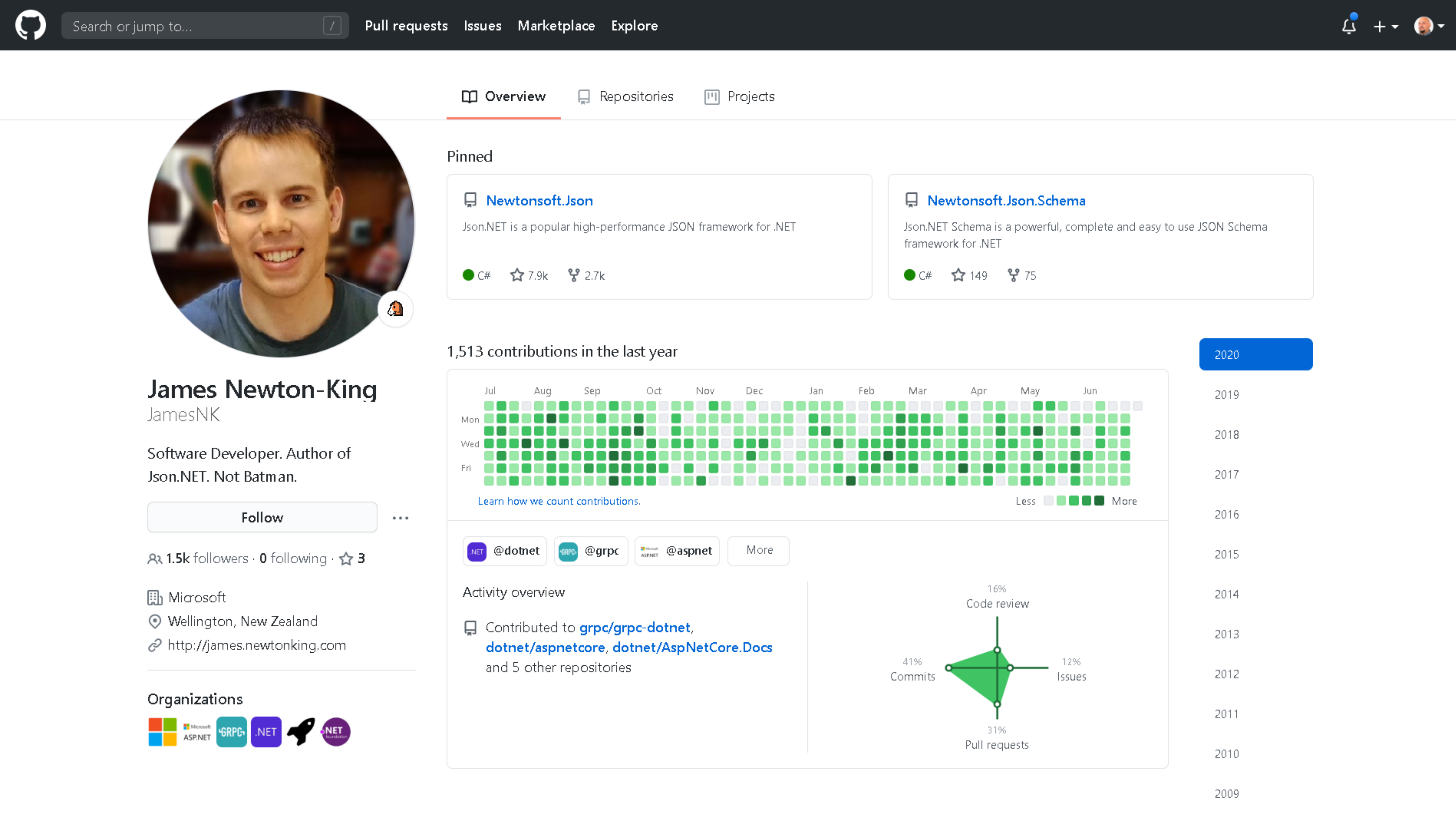
Labels
area-Meta

Projects
None yet

Milestone
3.0

Linked pull requests
Successfully merging a pull request may close this issue.
None yet

Notifications Customize
Subscribe
You're not receiving notifications from this thread.



Pinned

Newtonsoft.Json
Json.NET is a popular high-performance JSON framework for .NET
C# ☆ 7.9k 2.7k

Newtonsoft.Json.Schema
Json.NET Schema is a powerful, complete and easy to use JSON Schema framework for .NET
C# ☆ 149 75

1,513 contributions in the last year

Learn how we count contributions.

Less ⬜ 🟩 🟩 🟩 🟩 More

2020
2019
2018
2017
2016
2015
2014
2013
2012
2011
2010
2009

@dotnet @grpc @aspnet More

Activity overview

Contributed to grpc/grpc-dotnet, dotnet/aspnetcore, dotnet/AspNetCore.Docs and 5 other repositories

James Newton-King
JamesNK

Software Developer. Author of Json.NET. Not Batman.

Follow

1.5k followers · 0 following · ☆ 3

Microsoft
Wellington, New Zealand
http://james.newtonking.com

Organizations

Json.NET was created over 10 years ago, and since then it has added a wide range of features aimed to help developers work with JSON in .NET. In that time Json.NET has also become far and away NuGet's most depended on and downloaded package, and is the go-to library for JSON support in .NET. Unfortunately, Json.NET's wealth of features and popularity works against making major changes to it. Supporting new technologies like Span<T> would require fundamental breaking changes to the library and would disrupt existing applications and libraries that depend on it.

James Newton-King says...

Going forward Json.NET will continue to be worked on and invested in, both addressing known issues today and supporting new platforms in the future. Json.NET has always existed alongside other JSON libraries for .NET, and there will be nothing to prevent you using one or more together, depending on whether you need the performance of the new JSON APIs or the large feature set of Json.NET.

James Newton-King says…

So What Should I Do?

Json.NET

System.Text.Json

How to migrate from Newtonsoft.Json to System.Text.Json

01/10/2020 • 32 minutes to read •

This article shows how to migrate from Newtonsoft.Json to System.Text.Json.

The `System.Text.Json` namespace provides functionality for serializing to and deserializing from JavaScript Object Notation (JSON). The `System.Text.Json` library is included in the .NET Core 3.0 shared framework. For other target frameworks, install the System.Text.Json NuGet package. The package supports:

- .NET Standard 2.0 and later versions
- .NET Framework 4.7.2 and later versions
- .NET Core 2.0, 2.1, and 2.2

`System.Text.Json` focuses primarily on performance, security, and standards compliance. It has some key differences in default behavior and doesn't aim to have feature parity with `Newtonsoft.Json`. For some scenarios, `System.Text.Json` has no built-in functionality, but there are recommended workarounds. For other scenarios, workarounds are impractical. If your application depends on a missing feature, consider filing an issue to find out if support for your scenario can be added.

Most of this article is about how to use the JsonSerializer API, but it also includes guidance on how to use the JsonDocument (which represents the Document Object Model or DOM), Utf8JsonReader, and Utf8JsonWriter types.

Table of differences between Newtonsoft.Json and System.Text.Json

The following table lists `Newtonsoft.Json` features and `System.Text.Json` equivalents. The equivalents fall into the following categories:

Thank you.

In this article

Table of differences between Newtonsoft.Json and System.Text.Json

The following table lists `Newtonsoft.Json` features and `System.Text.Json` equivalents. The equivalents fall into the following categories:

- Supported by built-in functionality. Getting similar behavior from `System.Text.Json` may require the use of an attribute or global option.
- Not supported, workaround is possible. The workarounds are custom converters, which may not provide complete parity with `Newtonsoft.Json` functionality. For some of these, sample code is provided as examples. If you rely on these `Newtonsoft.Json` features, migration will require modifications to your .NET object models or other code changes.
- Not supported, workaround is not practical or possible. If you rely on these `Newtonsoft.Json` features, migration will not be possible without significant changes.

Newtonsoft.Json feature	**System.Text.Json equivalent**
Case-insensitive deserialization by default	✓ PropertyNameCaseInsensitive global setting
Camel-case property names	✓ PropertyNamingPolicy global setting
Minimal character escaping	✓ Strict character escaping, configurable
`NullValueHandling.Ignore` global setting	✓ IgnoreNullValues global option
Allow comments	✓ ReadCommentHandling global setting
Allow trailing commas	✓ AllowTrailingCommas global setting
Custom converter registration	✓ Order of precedence differs
No maximum depth by default	✓ Default maximum depth 64, configurable
Support for a broad range of types	⚠ Some types require custom converters

Demo

Using System.Text.Json

Final Takeaway

JSON
- Data interchange format

JavaScript
- Supported natively

Key/value pairs, arrays, and objects

Json.NET

Serialization and deserialization

JsonSerializer and JsonConvert

Final Takeaway

Types

Dates are hard

JsonSerializerSettings or attributes

Conditional serialization

Custom JsonConverter

Callbacks

Final Takeaway

ITraceWriter

Performance

LINQ To JSON

XML <-> JSON

BSON

Json.NET Schema

Final Takeaway

.NET Core

.NET 5

System.Text.Json

Thanks for watching!

“What you learn is yours for life.”

@xmorera

www.xaviermorera.com

Big Data Inc.

www.bigdatainc.org