OpenApi Doc zu Proxy

Kurzantwort

- NSwag bekommst du als Windows-GUI (NSwagStudio), als CLI und als NuGet/MSBuild-Integration. Download/Installationswege siehe unten.
- „Lists statt Arrays“ stellst du über die Generator-Settings ein (arrayType/arrayInstanceType/responseArrayType/parameterArrayType). Beispiel-nswag.json ist weiter unten.

Wo bekomme ich NSwag her?

- NSwagStudio (GUI + CLI inklusive): auf GitHub unter “Releases/Downloads” – MSI oder ZIP (xcopy). (github.com)
- NSwag.MSBuild (für automatische Generierung beim Build): NuGet-Paket NSwag.MSBuild in dein Projekt einbinden. (nuget.org, github.com)
- Alternativ: CLI als .NET-Tool/ZIP; allgemeine Befehle/Kommandos siehe Doku. (github.com)
- Offizielle Einstiegshilfe von Microsoft (Codegen-Optionen, NSwagStudio, NuGet, CLI): (learn.microsoft.com)

Beispiel: C#-Client mit List<T> statt T[] Variante A: nswag.json (empfohlen, reproduzierbar)

- Lege neben dein Projekt eine nswag.json und rufe sie via NSwagStudio („Generate Outputs“) oder per CLI/MSBuild auf.

Beispieldatei (wichtige Stellen markiert)

```
{
  "documentGenerator": {
    "fromDocument": {
      "url": "https://example.local/swagger/v1/swagger.json",
      "output": null,
      "newLineBehavior": "Auto"
    }
  },
  "codeGenerators": {
    "openApiToCSharpClient": {
      "namespace": "MyCompany.MyApiClient",
```

```
      "className": "{controller}Client",
      "generateClientClasses": true,
      "generateClientInterfaces": true,
      "injectHttpClient": true,
      "disposeHttpClient": false,
      "useBaseUrl": false,

      // WICHTIG: Lists statt Arrays
      "arrayType": "System.Collections.Generic.List",
      "arrayInstanceType": "System.Collections.Generic.List",
      "responseArrayType": "System.Collections.Generic.List",
      "parameterArrayType": "System.Collections.Generic.IEnumerable",

      // weitere sinnvolle Defaults für .NET Framework 4.8
      "jsonLibrary": "NewtonsoftJson",
      "classStyle": "Poco",
      "generateDefaultValues": true,
      "generateDataAnnotations": true,

      "output": "Generated\\MyApiClient.g.cs",
      "newLineBehavior": "Auto"
    }
  }
}
```

- Die Settings arrayType/arrayInstanceType/responseArrayType/parameterArrayType sind offiziell Teil der NSwag/NJsonSchema-Generator-Konfiguration. In der „NSwag Configuration Document“-Doku findest du diese Felder (dort z. T. mit anderen Beispieltpyen); du setzt sie einfach auf List bzw. IEnumerable wie oben. (github.com, stackoverflow.com)
- Tipp: Für Methodenparameter ist IEnumerable<T> flexibel, Rückgaben/DTO-Eigenschaften als List<T> erfüllen „List statt Array“.

Aufruf

- Per CLI (aus Projektordner):
  - nswag run .\nswag.json
- Per MSBuild automatisch (z. B. nach dem Build):

```
<!-- In der .csproj des .NET Framework 4.8-Projekts -->
<ItemGroup>
```

```
  <PackageReference Include="NSwag.MSBuild" Version="14.4.0">
    <PrivateAssets>all</PrivateAssets>
  </PackageReference>
</ItemGroup>

<Target Name="NSwag" AfterTargets="Build">
  <Exec Command="$(NSwagExe) run nswag.json" />
</Target>
```

- NSwag.MSBuild stellt die Variable $(NSwagExe) bereit, die das passende nswag-Tool aufruft. (github.com)

Variante B: Direkt über die CLI ohne nswag.json

- Du kannst die wichtigsten Optionen auch als Schalter setzen, z. B.:

```
nswag openapi2csclient ^
  /input:https://example.local/swagger/v1/swagger.json ^
  /output:Generated\Client.g.cs ^
  /namespace:MyCompany.MyApiClient ^
  /ArrayType:"System.Collections.Generic.List" ^
  /ArrayInstanceType:"System.Collections.Generic.List" ^
  /ResponseArrayType:"System.Collections.Generic.List" ^
  /ParameterArrayType:"System.Collections.Generic.IEnumerable"
```

- Die verfügbaren Kommandos/Parameter sind in der CommandLine-Doku beschrieben; die oben gezeigten Typ-Optionen entsprechen den nswag.json-Feldern aus der Config-Doku. (github.com)

Variante C: NSwagStudio (GUI)

- Swagger-URL eintragen, Tab „CSharp Client“ öffnen und bei „Array Type“/„Array Instance Type“ List eintragen; generieren. NSwagStudio kann die Konfiguration als nswag.json speichern. (github.com)

Server-Stubs (falls du „API“ im Sinne von Controller-Gerüst meintest)

- NSwag kann auch Controller aus einem OpenAPI-Dokument erzeugen (ASP.NET Web API 2 für .NET Framework). In der Config nutzt du statt openApiToCSharpClient den Generator openApiToCSharpController, und kannst arrayType etc. identisch setzen; für klassisches Web API ist das aspNetNamespace typischerweise "System.Web.Http". (github.com)

Kompatibilität .NET Framework 4.8

- Die Kernpakete (NSwag.Core/NSwag.Generation/NSwag.AspNetCore) und NSwag.MSBuild unterstützen .NET Framework ≥4.6.2 – damit funktioniert 4.8 problemlos. (www-1.nuget.org, feed.nuget.org, packages.nuget.org)

Zusätzliche Hinweise

- Serializer: Für .NET Framework ist Newtonsoft.Json die pragmatische Wahl. Du kannst über jsonLibrary zwischen NewtonsoftJson und System.Text.Json wechseln (siehe Microsoft-Blogbeitrag mit Beispiel). (devblogs.microsoft.com)
- Best Practices: Auch wenn du List<T> erzwingst, sind für Parameter IEnumerable<T> und für öffentliche APIs häufig IReadOnlyList<T>/ICollection<T> sinnvoller. Wenn du aber explizit „List überall“ willst, setze responseArrayType ebenfalls auf List wie oben.

Wenn du mir deine Swagger/OpenAPI-URL gibst, kann ich dir eine passende nswag.json mit genau den richtigen Settings für dein Projekt (.NET 4.8, Namespace, Dateinamen, usw.) vorbereiten.

Revision #2
Created 28 August 2025 06:25:48 by Stefan Mechler
Updated 28 August 2025 06:28:48 by Stefan Mechler