A mapping between project.json and csproj properties

03/13/2017 6 minutes to read Contributors all

In this article

The csproj format
Common top-level properties
frameworks
dependencies
runtimes
tools
buildOptions
packOptions
scripts
runtimeOptions
shared
files
testRunner
See Also

By Nate McMaster

During the development of the .NET Core tooling, an important design change was made to no longer support *project.json* files and instead move the .NET Core projects to the MSBuild/csproj format.

This article shows how the settings in *project.json* are represented in the MSBuild/csproj format so you can learn how to use the new format and understand the changes made by the migration tools when you're upgrading your project to the latest version of the tooling.

The csproj format

The new format, *.csproj, is an XML-based format. The following example shows the root node of a .NET Core project using the `Microsoft.NET.Sdk`. For web projects, the SDK used is `Microsoft.NET.Sdk.Web`.

XML Copy

```xml
<Project Sdk="Microsoft.NET.Sdk">
...
```

```
</Project>
```

Common top-level properties

name

JSON

```json
{
  "name": "MyProjectName"
}
```

No longer supported. In csproj, this is determined by the project filename, which is defined by the directory name. For example, `MyProjectName.csproj`.

By default, the project filename also specifies the value of the `<AssemblyName>` and `<PackageId>` properties.

XML

```xml
<PropertyGroup>
  <AssemblyName>MyProjectName</AssemblyName>
  <PackageId>MyProjectName</PackageId>
</PropertyGroup>
```

The `<AssemblyName>` will have a different value than `<PackageId>` if `buildOptions\outputName` property was defined in project.json. For more information, see Other common build options.

version

JSON

```json
{
  "version": "1.0.0-alpha-*"
}
```

Use the `VersionPrefix` and `VersionSuffix` properties:

XML

```xml
<PropertyGroup>
  <VersionPrefix>1.0.0</VersionPrefix>
  <VersionSuffix>alpha</VersionSuffix>
</PropertyGroup>
```

You can also use the `Version` property, but this may override version settings during packaging:

XML

```xml
<PropertyGroup>
  <Version>1.0.0-alpha</Version>
</PropertyGroup>
```

Other common root-level options

JSON

```json
{
  "authors": [ "Anne", "Bob" ],
  "company": "Contoso",
  "language": "en-US",
  "title": "My library",
  "description": "This is my library.\r\nAnd it's really great!",
  "copyright": "Nugetizer 3000",
  "userSecretsId": "xyz123"
}
```

XML

```xml
<PropertyGroup>
  <Authors>Anne;Bob</Authors>
  <Company>Contoso</Company>
  <NeutralLanguage>en-US</NeutralLanguage>
  <AssemblyTitle>My library</AssemblyTitle>
  <Description>This is my library.
And it's really great!</Description>
  <Copyright>Nugetizer 3000</Copyright>
  <UserSecretsId>xyz123</UserSecretsId>
</PropertyGroup>
```

frameworks

One target framework

JSON

```json
{
  "frameworks": {
    "netcoreapp1.0": {}
  }
}
```

XML Copy

```xml
<PropertyGroup>
  <TargetFramework>netcoreapp1.0</TargetFramework>
</PropertyGroup>
```

Multiple target frameworks

JSON Copy

```json
{
  "frameworks": {
    "netcoreapp1.0": {},
    "net451": {}
  }
}
```

Use the `TargetFrameworks` property to define your list of target frameworks. Use semi-colon to separate multiple framework values.

XML Copy

```xml
<PropertyGroup>
  <TargetFrameworks>netcoreapp1.0;net451</TargetFrameworks>
</PropertyGroup>
```

dependencies

ⓘ **Important**

If the dependency is a **project** and not a package, the format is different. For more information, see the dependency type section.

NETStandard.Library metapackage

JSON Copy

```json
{
  "dependencies": {
    "NETStandard.Library": "1.6.0"
  }
}
```

XML Copy

```xml
<PropertyGroup>
  <NetStandardImplicitPackageVersion>1.6.0</NetStandardImplicitPackageVersion>
</PropertyGroup>
```

Microsoft.NETCore.App metapackage

JSON Copy

```json
{
  "dependencies": {
    "Microsoft.NETCore.App": "1.0.0"
  }
}
```

XML Copy

```xml
<PropertyGroup>
  <RuntimeFrameworkVersion>1.0.3</RuntimeFrameworkVersion>
</PropertyGroup>
```

Note that the `<RuntimeFrameworkVersion>` value in the migrated project is determined by the version of the SDK you have installed.

Top-level dependencies

JSON Copy

```json
{
  "dependencies": {
    "Microsoft.AspNetCore": "1.1.0"
  }
}
```

XML Copy

```xml
<ItemGroup>
  <PackageReference Include="Microsoft.AspNetCore" Version="1.1.0" />
</ItemGroup>
```

Per-framework dependencies

JSON

Copy

```json
{
  "framework": {
    "net451": {
      "dependencies": {
        "System.Collections.Immutable": "1.3.1"
      }
    },
    "netstandard1.5": {
      "dependencies": {
        "Newtonsoft.Json": "9.0.1"
      }
    }
  }
}
```

XML

Copy

```xml
<ItemGroup Condition="'$(TargetFramework)'=='net451'">
  <PackageReference Include="System.Collections.Immutable" Version="1.3.1" />
</ItemGroup>

<ItemGroup Condition="'$(TargetFramework)'=='netstandard1.5'">
  <PackageReference Include="Newtonsoft.Json" Version="9.0.1" />
</ItemGroup>
```

imports

JSON

Copy

```json
{
  "dependencies": {
    "YamlDotNet": "4.0.1-pre309"
  },
  "frameworks": {
    "netcoreapp1.0": {
      "imports": [
        "dnxcore50",
        "dotnet"
      ]
    }
  }
}
```

XML

Copy

```xml
<PropertyGroup>
  <PackageTargetFallback>dnxcore50;dotnet</PackageTargetFallback>
</PropertyGroup>
<ItemGroup>
  <PackageReference Include="YamlDotNet" Version="4.0.1-pre309" />
</ItemGroup>
```

dependency type

type: project

JSON Copy

```json
{
  "dependencies": {
    "MyOtherProject": "1.0.0-*",
    "AnotherProject": {
      "type": "project"
    }
  }
}
```

XML Copy

```xml
<ItemGroup>
  <ProjectReference Include="..\MyOtherProject\MyOtherProject.csproj" />
  <ProjectReference Include="..\AnotherProject\AnotherProject.csproj" />
</ItemGroup>
```

Note

This will break the way that `dotnet pack --version-suffix $suffix` determines the dependency version of a project reference.

type: build

JSON Copy

```json
{
  "dependencies": {
    "Microsoft.EntityFrameworkCore.Design": {
      "version": "1.1.0",
      "type": "build"
    }
    }
  }
```

XML

```xml
<ItemGroup>
  <PackageReference Include="Microsoft.EntityFrameworkCore.Design"
Version="1.1.0" PrivateAssets="All" />
</ItemGroup>
```

type: platform

JSON

```json
{
  "dependencies": {
    "Microsoft.NETCore.App": {
      "version": "1.1.0",
      "type": "platform"
    }
  }
}
```

There is no equivalent in csproj.

runtimes

JSON

```json
{
  "runtimes": {
    "win7-x64": {},
    "osx.10.11-x64": {},
    "ubuntu.16.04-x64": {}
  }
}
```

XML

```xml
<PropertyGroup>
  <RuntimeIdentifiers>win7-x64;osx.10.11-x64;ubuntu.16.04-
x64</RuntimeIdentifiers>
</PropertyGroup>
```

Standalone apps (self-contained deployment)

In project.json, defining a `runtimes` section means the app was standalone during build and publish. In MSBuild, all projects are *portable* during build, but can be published as standalone.

```
dotnet publish --framework netcoreapp1.0 --runtime osx.10.11-x64
```

For more information, see Self-contained deployments (SCD).

tools

JSON

```json
{
  "tools": {
    "Microsoft.EntityFrameworkCore.Tools.DotNet": "1.0.0-*"
  }
}
```

XML

```xml
<ItemGroup>
  <DotNetCliToolReference Include="Microsoft.EntityFrameworkCore.Tools.DotNet"
Version="1.0.0" />
</ItemGroup>
```

Note

`imports` on tools are not supported in csproj. Tools that need imports will not work with the new `Microsoft.NET.Sdk`.

buildOptions

See also Files.

emitEntryPoint

JSON

```json
{
  "buildOptions": {
    "emitEntryPoint": true
  }
}
```

XML Copy

```xml
<PropertyGroup>
  <OutputType>Exe</OutputType>
</PropertyGroup>
```

If `emitEntryPoint` was `false`, the value of `OutputType` is converted to `Library`, which is the default value:

JSON Copy

```json
{
  "buildOptions": {
    "emitEntryPoint": false
  }
}
```

XML Copy

```xml
<PropertyGroup>
  <OutputType>Library</OutputType>
  <!-- or, omit altogether. It defaults to 'Library' -->
</PropertyGroup>
```

keyFile

JSON Copy

```json
{
  "buildOptions": {
    "keyFile": "MyKey.snk"
  }
}
```

The `keyFile` element expands to three properties in MSBuild:

XML Copy

```xml
<PropertyGroup>
  <AssemblyOriginatorKeyFile>MyKey.snk</AssemblyOriginatorKeyFile>
  <SignAssembly>true</SignAssembly>
  <PublicSign Condition="'$(OS)' != 'Windows_NT'">true</PublicSign>
</PropertyGroup>
```

Other common build options

JSON

Copy

```json
{
  "buildOptions": {
    "warningsAsErrors": true,
    "nowarn": ["CS0168", "CS0219"],
    "xmlDoc": true,
    "preserveCompilationContext": true,
    "outputName": "Different.AssemblyName",
    "debugType": "portable",
    "allowUnsafe": true,
    "define": ["TEST", "OTHERCONDITION"]
  }
}
```

XML

Copy

```xml
<PropertyGroup>
  <TreatWarningsAsErrors>true</TreatWarningsAsErrors>
  <NoWarn>$(NoWarn);CS0168;CS0219</NoWarn>
  <GenerateDocumentationFile>true</GenerateDocumentationFile>
  <PreserveCompilationContext>true</PreserveCompilationContext>
  <AssemblyName>Different.AssemblyName</AssemblyName>
  <DebugType>portable</DebugType>
  <AllowUnsafeBlocks>true</AllowUnsafeBlocks>
  <DefineConstants>$(DefineConstants);TEST;OTHERCONDITION</DefineConstants>
</PropertyGroup>
```

packOptions

See also Files.

Common pack options

JSON

Copy

```json
{
  "packOptions": {
    "summary": "numl is a machine learning library intended to ease the use of
using standard modeling techniques for both prediction and clustering.",
    "tags": ["machine learning", "framework"],
    "releaseNotes": "Version 0.9.12-beta",
    "iconUrl": "http://numl.net/images/ico.png",
    "projectUrl": "http://numl.net",
    "licenseUrl": "https://raw.githubusercontent.com/sethjuarez/numl/master/LI-
CENSE.md",
    "requireLicenseAcceptance": false,
    "repository": {
      "type": "git",
      "url": "https://raw.githubusercontent.com/sethjuarez/numl"
    },
    "owners": ["Seth Juarez"]
  }
}
```

XML

```xml
<PropertyGroup>
  <!-- summary is not migrated from project.json, but you can use the <Descrip-
tion> property for that if needed. -->
  <PackageTags>machine learning;framework</PackageTags>
  <PackageReleaseNotes>Version 0.9.12-beta</PackageReleaseNotes>
  <PackageIconUrl>http://numl.net/images/ico.png</PackageIconUrl>
  <PackageProjectUrl>http://numl.net</PackageProjectUrl>
  <PackageLicenseUrl>https://raw.githubusercontent.com/sethjuarez/numl/master/LI-
CENSE.md</PackageLicenseUrl>
  <PackageRequireLicenseAcceptance>false</PackageRequireLicenseAcceptance>
  <RepositoryType>git</RepositoryType>

<RepositoryUrl>https://raw.githubusercontent.com/sethjuarez/numl</RepositoryUrl>
  <!-- owners is not supported in MSBuild -->
</PropertyGroup>
```

There is no equivalent for the `owners` element in MSBuild. For `summary`, you can use the MSBuild `<Description>` property, even though the value of `summary` is not migrated automatically to that property, since that property is mapped to the `description` element.

scripts

JSON

```json
{
  "scripts": {
    "precompile": "generateCode.cmd",
    "postpublish": [ "obfuscate.cmd", "removeTempFiles.cmd" ]
  }
}
```

Their equivalent in MSBuild are targets:

XML

```xml
<Target Name="MyPreCompileTarget" BeforeTargets="Build">
  <Exec Command="generateCode.cmd" />
</Target>

<Target Name="MyPostCompileTarget" AfterTargets="Publish">
```

```
  <Exec Command="obfuscate.cmd" />
  <Exec Command="removeTempFiles.cmd" />
</Target>
```

runtimeOptions

JSON

```json
{
  "runtimeOptions": {
    "configProperties": {
      "System.GC.Server": true,
      "System.GC.Concurrent": true,
      "System.GC.RetainVM": true,
      "System.Threading.ThreadPool.MinThreads": 4,
      "System.Threading.ThreadPool.MaxThreads": 25
    }
  }
}
```

All settings in this group, except for the "System.GC.Server" property, are placed into a file called *runtimeconfig.template.json* in the project folder, with options lifted to the root object during the migration process:

JSON

```json
{
  "configProperties": {
    "System.GC.Concurrent": true,
    "System.GC.RetainVM": true,
    "System.Threading.ThreadPool.MinThreads": 4,
    "System.Threading.ThreadPool.MaxThreads": 25
  }
}
```

The "System.GC.Server" property is migrated into the csproj file:

XML

```xml
<PropertyGroup>
  <ServerGarbageCollection>true</ServerGarbageCollection>
</PropertyGroup>
```

However, you can set all those values in the csproj as well as MSBuild properties:

XML

```xml
<PropertyGroup>
  <ServerGarbageCollection>true</ServerGarbageCollection>
  <ConcurrentGarbageCollection>true</ConcurrentGarbageCollection>
  <RetainVMGarbageCollection>true</RetainVMGarbageCollection>
  <ThreadPoolMinThreads>4</ThreadPoolMinThreads>
  <ThreadPoolMaxThreads>25</ThreadPoolMaxThreads>
</PropertyGroup>
```

shared

JSON Copy

```json
{
  "shared": "shared/**/*.cs"
}
```

Not supported in csproj. You must instead create include content files in your *.nuspec* file. For more information, see Including content files.

files

In *project.json*, build and pack could be extended to compile and embed from different folders. In MSBuild, this is done using items. The following example is a common conversion:

JSON Copy

```json
{
  "buildOptions": {
    "compile": {
      "copyToOutput": "notes.txt",
      "include": "../Shared/*.cs",
      "exclude": "../Shared/Not/*.cs"
    },
    "embed": {
      "include": "../Shared/*.resx"
    }
  },
  "packOptions": {
    "include": "Views/",
    "mappings": {
      "some/path/in/project.txt": "in/package.txt"
    }
  },
  "publishOptions": {
    "include": [
      "files/",
      "publishnotes.txt"
    ]
  }
}
```

XML

```xml
<ItemGroup>
  <Compile Include="..\Shared\*.cs" Exclude="..\Shared\Not\*.cs" />
  <EmbeddedResource Include="..\Shared\*.resx" />
  <Content Include="Views\**\*" PackagePath="%(Identity)" />
  <None Include="some/path/in/project.txt" Pack="true" PackagePath="in/package.txt" />

  <None Include="notes.txt" CopyToOutputDirectory="Always" />
  <!-- CopyToOutputDirectory = { Always, PreserveNewest, Never } -->

  <Content Include="files\**\*" CopyToPublishDirectory="PreserveNewest" />
  <None Include="publishnotes.txt" CopyToPublishDirectory="Always" />
  <!-- CopyToPublishDirectory = { Always, PreserveNewest, Never } -->
</ItemGroup>
```

Note

Many of the default globbing patterns are added automatically by the .NET Core SDK. For more information, see Default Compile Item Values.

All MSBuild `ItemGroup` elements support `Include`, `Exclude`, and `Remove`.

Package layout inside the .nupkg can be modified with `PackagePath="path"`.

Except for `Content`, most item groups require explicitly adding `Pack="true"` to be included in the package. `Content` will be put in the *content* folder in a package since the MSBuild `<IncludeContentInPack>` property is set to `true` by default. For more information, see Including content in a package.

`PackagePath="%(Identity)"` is a short way of setting package path to the project-relative file path.

testRunner

xUnit

JSON

```json
{
  "testRunner": "xunit",
  "dependencies": {
    "dotnet-test-xunit": "<any>"
  }
}
```

XML

```xml
<ItemGroup>
  <PackageReference Include="Microsoft.NET.Test.Sdk" Version="15.0.0-*" />
  <PackageReference Include="xunit" Version="2.2.0-*" />
  <PackageReference Include="xunit.runner.visualstudio" Version="2.2.0-*" />
</ItemGroup>
```

MSTest

JSON

```json
{
  "testRunner": "mstest",
  "dependencies": {
    "dotnet-test-mstest": "<any>"
  }
}
```

XML

```xml
<ItemGroup>
  <PackageReference Include="Microsoft.NET.Test.Sdk" Version="15.0.0-*" />
  <PackageReference Include="MSTest.TestAdapter" Version="1.1.12-*" />
  <PackageReference Include="MSTest.TestFramework" Version="1.1.11-*" />
</ItemGroup>
```

See Also

High-level overview of changes in CLI