Pacote e restauração do NuGet como destinos do MSBuild

Artigo • 13/03/2024

NuGet 4.0+

Com o formato PackageReference, o NuGet 4.0 e superior pode armazenar todos os metadados de manifesto diretamente dentro de um arquivo de projeto em vez de usar um arquivo `.nuspec` separado.

Com o MSBuild 15.1 ou superior, o NuGet também é um cidadão de primeira classe do MSBuild com os destinos `pack` e `restore`, conforme descrito abaixo. Esses destinos permitem que você trabalhe com o NuGet como faria com qualquer outra tarefa ou destino do MSBuild. Para obter instruções sobre como criar um pacoteNuGet usando o MSBuild, consulte Criar um pacote do NuGet usando o MSBuild. (Para NuGet 3.x e inferiores, use os comandos pack e restore na CLI do NuGet.)

Ordem de build de destino

Como `pack` e `restore` são os destinos do MSBuild, você pode acessá-los para melhorar o fluxo de trabalho. Por exemplo, digamos que você deseja copiar o pacote para um compartilhamento de rede após empacotá-lo. Você pode fazer isso adicionando o seguinte ao seu arquivo de projeto:

XML

```xml
<Target Name="CopyPackage" AfterTargets="Pack">
  <Copy
    SourceFiles="$(OutputPath)..\$(PackageId).$(PackageVersion).nupkg"
    DestinationFolder="\\myshare\packageshare\"
    />
</Target>
```

Da mesma forma, você pode gravar uma tarefa do MSBuild escrever seu próprio destino e consumir propriedades do NuGet na tarefa do MSBuild.

> **Observação**
>
> `$(OutputPath)` é relativo e espera que você esteja executando o comando a partir da raiz do projeto.

pack target

Para projetos .NET que usam o formato `PackageReference` usando `msbuild -t:pack` atrai entradas do arquivo de projeto para usar na criação de um pacote NuGet.

A tabela a seguir descreve as propriedades do MSBuild que podem ser adicionadas a um arquivo do projeto dentro do primeiro nó `<PropertyGroup>`. Você pode fazer essas edições facilmente no Visual Studio 2017 e posterior clicando com o botão direito do mouse no projeto e selecionando **Editar {project_name}** no menu de contexto. Para sua conveniência, a tabela é organizada pela propriedade equivalente em um arquivo .nuspec.

> **Observação**
>
> As propriedades `Owners` e `Summary` de `.nuspec` não são compatíveis com o MSBuild.

Expandir a tabela

Valor do atributo/de nuspec	Propriedade MSBuild	Padrão	Observações
`Id`	`PackageId`	`$(AssemblyName)`	`$(AssemblyName)` de MSBuild
`Version`	`PackageVersion`	Versão	Isso é compatível com o semver, por exemplo `1.0.0`, `1.0.0-beta` ou `1.0.0-beta-00345`. O padrão será `Version` se não estiver definido.
`VersionPrefix`	`VersionPrefix`	empty	Substituir `PackageVersion` configurações `VersionPrefix`
`VersionSuffix`	`VersionSuffix`	empty	Substituir `PackageVersion` configurações `VersionSuffix`
`Authors`	`Authors`	Nome do usuário atual	Uma lista separada por pontos e vírgulas de autores de pacotes, correspondendo aos nomes de perfil em nuget.org. Eles são exibidos na Galeria NuGet no nuget.org e são usados para fazer referência cruzada de pacotes dos mesmos autores.
`Owners`	N/D	Não presente em nuspec	
`Title`	`Title`	`$(PackageId)`	Um título amigável do pacote, geralmente usado em exibições de interface do usuário como em nuget.org e no Gerenciador de Pacotes do Visual Studio.
`Description`	`Description`	"Package Description"	Uma descrição longa para o manifesto do assembly. Se `PackageDescription` não for especificada, essa propriedade também será usada como a descrição do pacote.
`Copyright`	`Copyright`	empty	Detalhes sobre direitos autorais do pacote.
`RequireLicenseAcceptance`	`PackageRequireLicenseAcceptance`	`false`	Um valor booliano que especifica se o cliente deve solicitar que o consumidor aceite a licença do pacote antes de instalá-lo.
`license`	`PackageLicenseExpression`	empty	Corresponde ao `<license type="expression">`. Consulte Empacotando uma expressão de licença ou um arquivo de licença.
`license`	`PackageLicenseFile`	empty	Caminho até um arquivo de licença dentro do pacote, se você estiver usando uma licença personalizada ou uma licença que não tenha recebido um identificador SPDX. Você precisa empacotar explicitamente o arquivo de licença referenciado. Corresponde ao `<license type="file">`. Consulte Empacotando uma expressão de licença ou um arquivo de licença.
`LicenseUrl`	`PackageLicenseUrl`	empty	`PackageLicenseUrl` foi preterido. Em vez disso, use `PackageLicenseExpression` ou `PackageLicenseFile`.
`ProjectUrl`	`PackageProjectUrl`	empty	
`Icon`	`PackageIcon`	empty	Um caminho para uma imagem no pacote a ser usado como um ícone de pacote. Você precisa empacotar explicitamente o arquivo de imagem de ícone referenciado. Para obter mais informações, consulte Empacotando um arquivo de imagem de ícone e iconmetadados.
`IconUrl`	`PackageIconUrl`	empty	`PackageIconUrl` foi preterido em favor de `PackageIcon`. No entanto, para obter a melhor experiência no nível inferior, você deve especificar `PackageIconUrl`, além de `PackageIcon`.
`Readme`	`PackageReadmeFile`	empty	Você precisa empacotar explicitamente o arquivo Leiame referenciado.
`Tags`	`PackageTags`	empty	Uma lista delimitada por ponto e vírgula de marcas que designam o pacote.
`ReleaseNotes`	`PackageReleaseNotes`	empty	Notas de versão do projeto.

Valor do atributo/de nuspec	Propriedade MSBuild	Padrão	Observações
`Repository/Url`	`RepositoryUrl`	empty	URL do repositório usada para clonar ou recuperar código-fonte. Exemplo: *https://github.com/NuGethttps://github.com/NuGet/NuGet.Client.git* .
`Repository/Type`	`RepositoryType`	empty	Tipo de repositório. Exemplos: `git` (padrão), `tfs`.
`Repository/Branch`	`RepositoryBranch`	empty	Informações opcionais de ramificação do repositório. `RepositoryUrl` também deve ser especificado para que essa propriedade seja incluída. Exemplo: *master* (NuGet 4.7.0+).
`Repository/Commit`	`RepositoryCommit`	empty	Confirmação de repositório opcional ou conjunto de alterações para indicar em qual origem o pacote foi criado. `RepositoryUrl` também deve ser especificado para que essa propriedade seja incluída. Exemplo: *0e4d1b598f350b3dc675018d539114d1328189ef* (NuGet 4.7.0+).
`PackageType`	`<PackageType>CustomType1, 1.0.0.0;CustomType2</PackageType>`		Indica o uso pretendido do pacote. Os tipos de pacote usam o mesmo formato que as IDs de pacote e são delimitados por `;`. Os tipos de pacote podem ser versionados anexando um `,` e uma cadeia de caracteres Version. Consulte Definir um tipo de pacote do NuGet (NuGet 3.5.0+).
`Summary`	Sem suporte		

pack target inputs

Expandir a tabela

Propriedade	Descrição
`IsPackable`	Um valor booliano que especifica se o projeto pode ser empacotado. O valor padrão é `true`.
`SuppressDependenciesWhenPacking`	Defina como `true` para suprimir dependências de pacote do pacote NuGetgerado.
`PackageVersion`	Especifica a versão que o pacote resultante terá. Aceita todos os formatos de cadeia de caracteres de versão do NuGet. O padrão é o valor `$(Version)`, ou seja, da propriedade `Version` no projeto.
`PackageId`	Especifica o nome para o pacote resultante. Se não for especificado, a operação `pack` usará como padrão o `AssemblyName` ou o nome do diretório como o nome do pacote.
`PackageDescription`	Uma descrição longa do pacote para exibição de interface do usuário.
`Authors`	Uma lista separada por pontos e vírgulas de autores de pacotes, correspondendo aos nomes de perfil em nuget.org. Eles são exibidos na Galeria NuGet no nuget.org e são usados para fazer referência cruzada de pacotes dos mesmos autores.
`Description`	Uma descrição longa para o manifesto do assembly. Se `PackageDescription` não for especificada, essa propriedade também será usada como a descrição do pacote.
`Copyright`	Detalhes sobre direitos autorais do pacote.
`PackageRequireLicenseAcceptance`	Um valor booliano que especifica se o cliente deve solicitar que o consumidor aceite a licença do pacote antes de instalá-lo. O padrão é `false`.
`DevelopmentDependency`	Um valor booliano que especifica se o pacote está marcado como uma dependência somente de desenvolvimento, que impede que o pacote seja incluído como uma dependência em outros pacotes. Com `PackageReference` (NuGet 4.8+), esse sinalizador também significa que os recursos em tempo de compilação serão excluídos da compilação. Para obter mais informações, confira Suporte do DevelopmentDependency para PackageReference .

Propriedade	Descrição
`PackageLicenseExpression`	Um identificador de licença SPDX ou uma expressão, por exemplo, `Apache-2.0`. Para obter mais informações, consulte Empacotando uma expressão de licença ou um arquivo de licença.
`PackageLicenseFile`	Caminho até um arquivo de licença dentro do pacote, se você estiver usando uma licença personalizada ou uma licença que não tenha recebido um identificador SPDX.
`PackageLicenseUrl`	`PackageLicenseUrl` foi preterido. Em vez disso, use `PackageLicenseExpression` ou `PackageLicenseFile`.
`PackageProjectUrl`	
`PackageIcon`	Especifica o caminho do ícone do pacote, relativo à raiz do pacote. Para obter mais informações, consulte Empacotando um arquivo de imagem de ícone.
`PackageReleaseNotes`	Notas de versão do projeto.
`PackageReadmeFile`	Leiame do pacote.
`PackageTags`	Uma lista delimitada por ponto e vírgula de marcas que designam o pacote.
`PackageOutputPath`	Determina o caminho de saída no qual o pacote empacotado será solto. O padrão é `$(OutputPath)`.
`IncludeSymbols`	Esse valor booliano indica se o pacote deve criar um pacote de símbolos adicionais quando o projeto é empacotado. O formato do pacote de símbolos é controlado pela propriedade `SymbolPackageFormat`. Para obter mais informações, consulte IncludeSymbols.
`IncludeSource`	Esse valor booliano indica se o processo do pacote deve criar um pacote de origem. O pacote de origem contém o código-fonte da biblioteca, bem como arquivos PDB. Os arquivos de origem são colocados no diretório `src/ProjectName`, no arquivo de pacote resultante. Para obter mais informações, consulte IncludeSource.
`PackageType`	
`IsTool`	Especifica se todos os arquivos de saída são copiados para a pasta *tools* em vez da pasta *lib*. Para mais informações, confira IsTool.
`RepositoryUrl`	URL do repositório usada para clonar ou recuperar código-fonte. Exemplo: *https://github.com/NuGethttps://github.com/NuGet/NuGet.Client.git* .
`RepositoryType`	Tipo de repositório. Exemplos: `git` (padrão), `tfs`.
`RepositoryBranch`	Informações opcionais de ramificação do repositório. `RepositoryUrl` também deve ser especificado para que essa propriedade seja incluída. Exemplo: *master* (NuGet 4.7.0+).
`RepositoryCommit`	Confirmação de repositório opcional ou conjunto de alterações para indicar em qual origem o pacote foi criado. `RepositoryUrl` também deve ser especificado para que essa propriedade seja incluída. Exemplo: *0e4d1b598f350b3dc675018d539114d1328189ef* (NuGet 4.7.0+).
`SymbolPackageFormat`	Especifica o formato do pacote de símbolos. Se "symbols.nupkg", um pacote legado de símbolos é criado com uma extensão *.symbols.nupkg* contendo PDBs, DLLs e outros arquivos de saída. Se "snupkg", um pacote de símbolos do snupkg é criado contendo os PDBs portáveis. O padrão é "symbols.nupkg".
`NoPackageAnalysis`	Especifica que `pack` não deve executar a análise de pacote após a compilação do pacote.
`MinClientVersion`	Especifica a versão mínima do cliente NuGet que pode instalar esse pacote, imposta pelo nuget.exe e pelo Gerenciador de Pacotes do Visual Studio.
`IncludeBuildOutput`	Esse valor booliano especifica se os assemblies de saída do build devem ser empacotados ou não no arquivo *.nupkg*.
`IncludeContentInPack`	Esse valor booliano especifica se todos os itens que têm um tipo `Content` são incluídos automaticamente no pacote resultante. O padrão é `true`.
`BuildOutputTargetFolder`	Especifica a pasta para colocar os assemblies de saída. Os assemblies de saída (e outros arquivos de saída) são copiados para suas respectivas pastas de estrutura. Para obter mais informações, consulte Assemblies de saída.

Propriedade	Descrição
`ContentTargetFolders`	Especifica o local padrão para o qual todos os arquivos de conteúdo deverão ir se `PackagePath` não for especificado para eles. O valor padrão é "content;contentFiles". Para obter mais informações, consulte Incluindo conteúdo em um pacote.
`NuspecFile`	Caminho relativo ou absoluto para o arquivo *.nuspec* que está sendo usado para o empacotamento. Se especificado, ele será usado **exclusivamente** para as informações de empacotamento e as informações nos projetos não serão usadas. Para obter mais informações, confira Empacotar usando um .nuspec.
`NuspecBasePath`	O caminho base para o arquivo *.nuspec*. Para obter mais informações, confira Empacotar usando um .nuspec.
`NuspecProperties`	Lista separada por ponto e vírgula de pares chave-valor. Para obter mais informações, confira Empacotar usando um .nuspec.

pack scenarios

Suprimindo dependências

Para suprimir dependências de pacote do pacote gerado NuGet, defina `SuppressDependenciesWhenPacking` como `true`, o qual permitirá ignorar todas as dependências do arquivo nupkg gerado.

`PackageIconUrl`

`PackageIconUrl` é preterido em favor da propriedade PackageIcon. A partir da versão 5.3 do NuGet e do Visual Studio 2019 versão 16.3, `pack` gera o aviso NU5048 se os metadados do pacote especificarem apenas `PackageIconUrl`.

`PackageIcon`

> **Dica**
>
> Para manter a compatibilidade com versões anteriores de clientes e fontes que ainda não oferecem suporte a `PackageIcon`, especifique `PackageIcon` e `PackageIconUrl`. O Visual Studio oferece suporte a `PackageIcon` para pacotes provenientes de uma fonte baseada em pasta.

Empacotando um arquivo de imagem de ícone

Ao empacotar um arquivo de imagem de ícone, use a propriedade `PackageIcon` para especificar o caminho do arquivo de ícone, relativo à raiz do pacote. Além disso, certifique-se de que o arquivo está incluído no pacote. O tamanho do arquivo de imagem é limitado a 1 MB. Os formatos de arquivo com suporte incluem JPEG e PNG. Recomendamos uma resolução de imagem de 128x128.

Por exemplo:

XML

```
<PropertyGroup>
    ...
    <PackageIcon>icon.png</PackageIcon>
    ...
</PropertyGroup>

<ItemGroup>
    ...
    <None Include="images\icon.png" Pack="true" PackagePath="\"/>
```

```
    ...
</ItemGroup>
```

Exemplo de ícone de pacote .

Para o equivalente ao nuspec, dê uma olhada na referência para ícone do nuspec.

PackageReadmeFile

*Há suporte com o **NuGet 5.10.0 versão prévia 2 / SDK do .NET 5.0.300** e superior*

Ao empacotar um arquivo Leiame, você precisa usar a propriedade `PackageReadmeFile` para especificar o caminho do pacote, relativo à raiz do pacote. Além disso, você precisa certificar-se de que o arquivo está incluído no pacote. Os formatos de arquivo suportados incluem apenas Markdown (*.md*).

Por exemplo:

XML

```
<PropertyGroup>
    ...
    <PackageReadmeFile>readme.md</PackageReadmeFile>
    ...
</PropertyGroup>

<ItemGroup>
    ...
    <None Include="docs\readme.md" Pack="true" PackagePath="\"/>
    ...
</ItemGroup>
```

Para o equivalente ao nuspec, dê uma olhada na referência para readme do nuspec.

Assemblies de saída

O `nuget pack` copia arquivos de saída com extensões `.exe`, `.dll`, `.xml`, `.winmd`, `.json` e `.pri`. Os arquivos de saída que são copiados dependem do que o MSBuild fornece do destino `BuiltOutputProjectGroup`.

Há duas propriedades de MSBuild que você pode usar em seu arquivo de projeto ou a linha de comando para controlar onde ficam os assemblies de saída:

- `IncludeBuildOutput`: um valor booliano que determina se os assemblies de saída de build devem ser incluídos no pacote.
- `BuildOutputTargetFolder`: especifica a pasta na qual os assemblies de saída devem ser colocados. Os assemblies de saída (e outros arquivos de saída) são copiados para suas respectivas pastas de estrutura.

Referências de pacote

Consulte Referências de pacote em arquivos de projeto.

Referências de projeto a projeto

As referências projeto a projeto são consideradas como referências de pacote do NuGet. Por exemplo:

XML

```
<ProjectReference Include="..\UwpLibrary2\UwpLibrary2.csproj"/>
```

Você também pode adicionar os seguintes metadados à referência de projeto:

```
XML
<IncludeAssets>
<ExcludeAssets>
<PrivateAssets>
```

Incluindo o conteúdo em um pacote

Para incluir o conteúdo, adicione metadados extras ao item `<Content>` existente. Por padrão, tudo que pertencer ao tipo "Conteúdo" é incluído no pacote, a menos que seja substituído por entradas como a seguinte:

```
XML
<Content Include="..\win7-x64\libuv.txt">
 <Pack>false</Pack>
</Content>
```

Por padrão, tudo é adicionado à raiz da pasta `content` e `contentFiles\any\<target_framework>` dentro de um pacote e preserva a estrutura e pasta relativa, a menos que você especifique um caminho de pacote:

```
XML
<Content Include="..\win7-x64\libuv.txt">
  <Pack>true</Pack>
  <PackagePath>content\myfiles\</PackagePath>
</Content>
```

Se você quiser copiar todo o conteúdo somente para determinadas pastas raiz (em vez de para `content` e `contentFiles`), será possível usar a propriedade MSBuild do `ContentTargetFolders`, que assume como padrão "content;contentFiles", mas pode ser definida como qualquer outro nome de pasta. Observe que só especificar "contentFiles" em `ContentTargetFolders` coloca os arquivos em `contentFiles\any\<target_framework>` ou `contentFiles\<language>\<target_framework>` com base em `buildAction`.

`PackagePath` pode ser um conjunto de caminhos de destino separados por ponto e vírgula. Especificar um caminho de pacote vazio adicionaria o arquivo à raiz do pacote. Por exemplo, o seguinte adiciona `libuv.txt` a `content\myfiles`, `content\samples` e a raiz do pacote:

```
XML
<Content Include="..\win7-x64\libuv.txt">
  <Pack>true</Pack>
  <PackagePath>content\myfiles;content\sample;;</PackagePath>
</Content>
```

Há também uma propriedade do MSBuild, `$(IncludeContentInPack)`, que assume o padrão `true`. Se isso for definido como `false` em qualquer projeto, o conteúdo desse projeto não está incluído no pacote do nuget.

Outros metadados específicos do pacote que pode ser definido em qualquer um dos itens acima `<PackageCopyToOutput>` e `<PackageFlatten>`, que define os valores `CopyToOutput` e `Flatten` na entrada `contentFiles` no nuspec de saída.

> ⓘ **Observação**
>
> Além de itens de Conteúdo, os metadados `<Pack>` e `<PackagePath>` também podem ser definidos em arquivos com uma ação de Compile, EmbeddedResource, ApplicationDefinition, Page, Resource, SplashScreen, DesignData,

DesignDataWithDesignTimeCreateableTypes, CodeAnalysisDictionary, AndroidAsset, AndroidResource, BundleResource ou None.

Para o pacote acrescentar o nome de arquivo ao caminho do pacote ao usar padrões glob, seu caminho de pacote deve terminar com o caractere separador de pasta, caso contrário, o caminho do pacote será tratado como o caminho completo, incluindo o nome do arquivo.

IncludeSymbols

Ao usar o `MSBuild -t:pack -p:IncludeSymbols=true`, os arquivos `.pdb` correspondentes são copiados junto com outros arquivos de saída (`.dll`, `.exe`, `.winmd`, `.xml`, `.json`, `.pri`). Observe que a configuração `IncludeSymbols=true` cria um pacote regular *e* um pacote de símbolos.

IncludeSource

Isso é o mesmo que `IncludeSymbols`, exceto que também copia os arquivos de origem com arquivos `.pdb`. Todos os arquivos do tipo `Compile` são copiadas para `src\<ProjectName>\`, preservando a estrutura de pastas do caminho relativo no pacote resultante. O mesmo também ocorre para arquivos de origem de qualquer `ProjectReference` que tem `TreatAsPackageReference` definido como `false`.

Se um arquivo do tipo Compilação estiver fora da pasta de projeto, ele será adicionado apenas a `src\<ProjectName>\`.

Empacotando uma expressão de licença ou um arquivo de licença

Ao usar uma expressão de licença, use a propriedade `PackageLicenseExpression`. Para obter um exemplo, consulte Exemplo de expressão de licença .

XML

```xml
<PropertyGroup>
    <PackageLicenseExpression>MIT</PackageLicenseExpression>
</PropertyGroup>
```

Para saber mais sobre expressões de licença e licenças aceitas pelo NuGet.org, consulte metadados de licença.

Ao empacotar um arquivo de licença, use a propriedade `PackageLicenseFile` para especificar o caminho do pacote, relativo à raiz do pacote. Além disso, certifique-se de que o arquivo está incluído no pacote. Por exemplo:

XML

```xml
<PropertyGroup>
    <PackageLicenseFile>LICENSE.txt</PackageLicenseFile>
</PropertyGroup>

<ItemGroup>
    <None Include="licenses\LICENSE.txt" Pack="true" PackagePath=""/>
</ItemGroup>
```

Para obter um exemplo, consulte Exemplo de arquivo de licença .

> **Observação**
>
> Somente um dentre `PackageLicenseExpression`, `PackageLicenseFile` e `PackageLicenseUrl` pode ser especificado por vez.

Empacotando um arquivo sem uma extensão

Em alguns cenários, como ao empacotar um arquivo de licença, convém incluir um arquivo sem uma extensão. Por razões históricas, o NuGet e o MSBuild tratam caminhos sem extensão como diretórios.

XML

```xml
<PropertyGroup>
  <TargetFrameworks>netstandard2.0</TargetFrameworks>
  <PackageLicenseFile>LICENSE</PackageLicenseFile>
</PropertyGroup>

<ItemGroup>
  <None Include="LICENSE" Pack="true" PackagePath=""/>
</ItemGroup>
```

Arquivo sem exemplo de extensão .

IsTool

Ao usar `MSBuild -t:pack -p:IsTool=true`, todos os arquivos de saída, conforme especificado no cenário Assemblies de saída, são copiados para a pasta `tools` em vez da pasta `lib`. Observe que isso é diferente de um `DotNetCliTool` que é especificado pela configuração do `PackageType` no arquivo `.csproj`.

Empacotando usando um arquivo `.nuspec`

Embora seja recomendável incluir todas as propriedades que geralmente estão no arquivo `.nuspec` no arquivo de projeto, você pode optar por usar um arquivo `.nuspec` para empacotar seu projeto. Para um projeto que não seja de estilo SDK que usa `PackageReference`, você deve importar `NuGet.Build.Tasks.Pack.targets` para que a tarefa do pacote possa ser executada. Você ainda precisa restaurar o projeto antes de empacotar um arquivo nuspec. (Um projeto no estilo SDK inclui os destinos do pacote por padrão.)

A estrutura de destino do arquivo de projeto é irrelevante e não é usada ao empacotar um arquivo do nuspec. As três propriedades MSBuild a seguir são relevantes para empacotamento usando um `.nuspec`:

1. `NuspecFile`: caminho relativo ou absoluto para o arquivo `.nuspec` que está sendo usado para o empacotamento.
2. `NuspecProperties`: uma lista separada por ponto e vírgula de pares chave/valor. Devido à maneira como a análise de linha de comando do MSBuild funciona, várias propriedades precisam ser especificadas da seguinte maneira: `-p:NuspecProperties="key1=value1;key2=value2"`.
3. `NuspecBasePath`: o caminho base para o arquivo `.nuspec`.

Se estiver usando `dotnet.exe` para empacotar seu projeto, use um comando como o seguinte:

CLI do .NET

```
dotnet pack <path to .csproj file> -p:NuspecFile=<path to nuspec file> -p:NuspecProperties=<> -p:NuspecBasePath=
<Base path>
```

Se estiver usando MSBuild para empacotar seu projeto, use um comando como o seguinte:

cli

```
msbuild -t:pack <path to .csproj file> -p:NuspecFile=<path to nuspec file> -p:NuspecProperties=<> -p:NuspecBasePath=
<Base path>
```

Observe que empacotar um nuspec usando dotnet.exe ou o msbuild também leva à compilação do projeto por padrão. Isso pode ser evitado passando a propriedade `--no-build` para dotnet.exe, que é o equivalente à configuração `<NoBuild>true</NoBuild>` no arquivo de projeto, juntamente com a configuração `<IncludeBuildOutput>false</IncludeBuildOutput>` no arquivo de projeto.

Um exemplo de um arquivo *.csproj* para empacotar um arquivo nuspec é:

XML

```xml
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>netstandard2.0</TargetFramework>
    <NoBuild>true</NoBuild>
    <IncludeBuildOutput>false</IncludeBuildOutput>
    <NuspecFile>PATH_TO_NUSPEC_FILE</NuspecFile>
    <NuspecProperties>add nuspec properties here</NuspecProperties>
    <NuspecBasePath>optional to provide</NuspecBasePath>
  </PropertyGroup>
</Project>
```

Pontos de extensão avançados para criar um pacote personalizado

O destino `pack` fornece dois pontos de extensão que são executados na compilação interna específica da estrutura de destino. Os pontos de extensão oferecem suporte à inclusão de conteúdo específico da estrutura de destino e dos assemblies em um pacote:

- Destino `TargetsForTfmSpecificBuildOutput`: use para arquivos dentro da pasta `lib` ou de uma pasta especificada usando `BuildOutputTargetFolder`.
- Destino `TargetsForTfmSpecificContentInPackage`: use para arquivos fora do `BuildOutputTargetFolder`.

`TargetsForTfmSpecificBuildOutput`

Escreva um destino personalizado e especifique-o como o valor da propriedade `$(TargetsForTfmSpecificBuildOutput)`. Para todos os arquivos que precisam ir para `BuildOutputTargetFolder` (biblioteca por padrão), o destino deve gravar esses arquivos no `BuildOutputInPackage` do ItemGroup e definir os dois valores de metadados a seguir:

- `FinalOutputPath`: o caminho absoluto do arquivo. Se não for fornecido, a Identidade será usada para avaliar o caminho de origem.
- `TargetPath`: (opcional) defina quando o arquivo precisa ir para uma subpasta dentro do `lib\<TargetFramework>`, como assemblies satélite que vão para suas respectivas pastas de cultura. Usa o nome do arquivo como padrão.

Exemplo:

XML

```xml
<PropertyGroup>

<TargetsForTfmSpecificBuildOutput>$(TargetsForTfmSpecificBuildOutput);GetMyPackageFiles</TargetsForTfmSpecificBuildO
utput>
</PropertyGroup>

<Target Name="GetMyPackageFiles">
  <ItemGroup>
    <BuildOutputInPackage Include="$(OutputPath)cs\$(AssemblyName).resources.dll">
        <TargetPath>cs</TargetPath>
    </BuildOutputInPackage>
  </ItemGroup>
</Target>
```

`TargetsForTfmSpecificContentInPackage`

Escreva um destino personalizado e especifique-o como o valor da propriedade `$(TargetsForTfmSpecificContentInPackage)`. Para que todos os arquivos sejam incluídos no pacote, o destino deve gravar esses arquivos no `TfmSpecificPackageFile` do ItemGroup e definir os seguintes metadados opcionais:

- `PackagePath`: caminho onde o arquivo deve ser gerado no pacote. O NuGet emite um aviso se mais de um arquivo for adicionado ao mesmo caminho de pacote.
- `BuildAction`: a ação de compilação a ser atribuída ao arquivo, necessária somente se o caminho do pacote estiver na pasta `contentFiles`. Assume o valor padrão "Nenhum".

Por exemplo,

XML

```
<PropertyGroup>

<TargetsForTfmSpecificContentInPackage>$(TargetsForTfmSpecificContentInPackage);CustomContentTarget</TargetsForTfmSpecificContentInPackage>
</PropertyGroup>

<Target Name="CustomContentTarget">
  <ItemGroup>
    <TfmSpecificPackageFile Include="abc.txt">
      <PackagePath>mycontent/$(TargetFramework)</PackagePath>
    </TfmSpecificPackageFile>
    <TfmSpecificPackageFile Include="Extensions/ext.txt" Condition="'$(TargetFramework)' == 'net46'">
      <PackagePath>net46content</PackagePath>
    </TfmSpecificPackageFile>
  </ItemGroup>
</Target>
```

restore target

`MSBuild -t:restore` (que `nuget restore` e `dotnet restore` usam com projetos do .NET Core), restaura pacotes referenciados no arquivo de projeto da seguinte maneira:

1. Ler todas as referências projeto a projeto
2. Ler as propriedades do projeto para localizar a pasta intermediária e as estruturas de destino
3. Passe dados do MSBuild para NuGet.Build.Tasks.dll
4. Executar restauração
5. Baixar os pacotes
6. Gravar arquivo de ativos, destinos e objetos

O destino `restore` funciona para projetos usando o formato PackageReference. `MSBuild 16.5+` também tem suporte de aceitação para o formato `packages.config`.

> **Observação**
>
> O destino `restore` **não deve ser executado** em combinação com o destino `build`.

Restaurar propriedades

Configurações de restauração adicionais podem vir de propriedades do MSBuild no arquivo de projeto. Os valores também podem ser definidos na linha de comando usando a opção `-p:` (consulte os Exemplos abaixo).

Expandir a tabela

Propriedade	Descrição
`RestoreSources`	Uma lista delimitada por ponto e vírgula de origens de pacote.
`RestorePackagesPath`	Caminho da pasta dos pacotes do usuário.
`RestoreDisableParallel`	Limite os downloads para um de cada vez.
`RestoreConfigFile`	Caminho para um arquivo `Nuget.Config` a ser aplicado.
`RestoreNoHttpCache`	Se verdadeiro, evita o uso de pacotes em cache http. Consulte Gerenciando os pacotes globais e as pastas de cache.
`RestoreIgnoreFailedSources`	Se verdadeiro, ignora origens de pacote com falha ou ausentes.
`RestoreFallbackFolders`	Pastas de fallback, usadas da mesma forma que a pasta de pacotes do usuário é usada.
`RestoreAdditionalProjectSources`	Fontes adicionais a serem usadas durante a restauração.
`RestoreAdditionalProjectFallbackFolders`	Pastas de fallback adicionais a serem usadas durante a restauração.
`RestoreAdditionalProjectFallbackFoldersExcludes`	Exclui pastas de fallback especificadas em `RestoreAdditionalProjectFallbackFolders`
`RestoreTaskAssemblyFile`	Caminho para `NuGet.Build.Tasks.dll`.
`RestoreGraphProjectInput`	Lista separada por ponto e vírgula de projetos a serem restaurados, que devem conter caminhos absolutos.
`RestoreUseSkipNonexistentTargets`	Quando os projetos são coletados através de MSBuild, ele determina se eles são coletados usando a otimização `SkipNonexistentTargets`. Quando não for definido, o padrão será `true`. A consequência é um comportamento rápido quando os destinos de um projeto não podem ser importados.
`MSBuildProjectExtensionsPath`	Pasta de saída que usa `BaseIntermediateOutputPath` e a pasta `obj` como padrão.
`RestoreForce`	Em projetos baseados em PackageReference, força todas as dependências a serem resolvidas, mesmo que a última restauração tenha sido bem-sucedida. Especificar esse sinalizador é semelhante à exclusão do arquivo `project.assets.json`. Isso não ignora o http-cache.
`RestorePackagesWithLockFile`	Opta pelo uso de um arquivo de bloqueio.
`RestoreLockedMode`	Execute a restauração no modo de bloqueio. Isso significa que a restauração não reavaliará as dependências.
`NuGetLockFilePath`	Um local personalizado para o arquivo de bloqueio. O local padrão é ao lado do projeto e é nomeado `packages.lock.json`.
`RestoreForceEvaluate`	Força a restauração para recalcular as dependências e atualizar o arquivo de bloqueio sem qualquer aviso.
`RestorePackagesConfig`	Uma opção de aceitação, que restaura projetos com packages.config. Suporte com `MSBuild -t:restore` apenas.
`RestoreRepositoryPath`	somente packages.config. Especifica o diretório de pacotes no qual os pacotes devem ser restaurados. `SolutionDirectory` será usado se não dor especificado.
`RestoreUseStaticGraphEvaluation`	Uma opção de aceitação para usar a avaliação estática de gráficos do MSBuild em vez da avaliação padrão. A avaliação de gráficos estáticos é um recurso experimental que é significativamente mais rápido para grandes repositórios e soluções.
`RestoreUseLegacyDependencyResolver`	Uma recusa em usar o resolvedor de dependência herdado. A implementação do resolvedor de dependências do NuGet foi reescrita na versão 6.12. Essa opção força o algoritmo anterior a ser usado.

A propriedade `ExcludeRestorePackageImports` é uma propriedade interna usada pelo NuGet. Não deve ser modificado ou definido em nenhum arquivo do MSBuild.

Exemplos

Linha de comando:

```
cli
msbuild -t:restore -p:RestoreConfigFile=<path>
```

Arquivo de projeto:

```
XML
<PropertyGroup>
    <RestoreIgnoreFailedSources>true</RestoreIgnoreFailedSources>
</PropertyGroup>
```

Restaurar saídas

A restauração cria os seguintes arquivos na pasta `obj` de build:

Expandir a tabela

Arquivo	Descrição
`project.assets.json`	Contém o gráfico de dependência de todas as referências de pacote.
`{projectName}.projectFileExtension.nuget.g.props`	Referências a objetos do MSBuild contidos em pacotes
`{projectName}.projectFileExtension.nuget.g.targets`	Referências a destinos do MSBuild contidos em pacotes

Restaurando e compilando com um comando do MSBuild

Devido ao fato de que NuGet pode restaurar pacotes que derrubam destinos e adereços do MSBuild, as avaliações de restauração e compilação são executadas com propriedades globais diferentes. Isso significa que os seguintes terão um comportamento imprevisível e, muitas vezes, incorreto.

```
cli
msbuild -t:restore,build
```

Em vez disso, a abordagem recomendada é:

```
cli
msbuild -t:build -restore
```

A mesma lógica se aplica a outros alvos semelhantes a `build`.

Restaurando projetos PackageReference e packages.config com o MSBuild

Com o MSBuild 16.5+, também há suporte de packages.config para `msbuild -t:restore`.

```
cli
msbuild -t:restore -p:RestorePackagesConfig=true
```

> **Observação**
>
> A restauração `packages.config` só está disponível com `MSBuild 16.5+`, e não com `dotnet.exe`

Restauração com avaliação estática de gráficos do MSBuild

> **Observação**
>
> Com o MSBuild 16.6+, o NuGet adicionou um recurso experimental para usar a avaliação estática de gráficos a partir da linha de comando que melhora significativamente o tempo de restauração para grandes repositórios.

cli

```
msbuild -t:restore -p:RestoreUseStaticGraphEvaluation=true
```

Como alternativa, você pode habilitá-la configurando a propriedade em um arquivo Directory.Build.Props.

XML

```
<Project>
  <PropertyGroup>
    <RestoreUseStaticGraphEvaluation>true</RestoreUseStaticGraphEvaluation>
  </PropertyGroup>
</Project>
```

> **Observação**
>
> A partir do Visual Studio 2019.x e do NuGet 5.x, esse recurso é considerado experimental e de aceitação. Siga NuGet/Home#9803 para obter detalhes sobre quando esse recurso será habilitado por padrão.

A restauração estática do gráfico altera a parte msbuild da restauração, a leitura e a avaliação do projeto, mas não o algoritmo de restauração. O algoritmo de restauração é o mesmo em todas as ferramentas do NuGet (NuGet.exe, MSBuild.exe, dotnet.exe e Visual Studio).

Em pouquíssimos cenários, a restauração de gráfico estático pode se comportar de forma diferente da restauração atual e certos PackageReferences ou ProjectReferences declarados podem estar ausentes.

Para tranquilizar sua mente, como uma verificação única, ao migrar para a restauração de gráfico estático, considere executar:

cli

```
msbuild.exe -t:restore -p:RestoreUseStaticGraphEvaluation=true
msbuild.exe -t:restore
```

O NuGet *não* deve relatar alterações. Se você vir uma discrepância, registre um problema em NuGet/Home .

Substituindo uma biblioteca de um grafo de restauração

Se uma restauração está trazendo o assembly errado, é possível excluir a escolha padrão do pacote e substituí-lo por outro de sua escolha. Primeiro com um `PackageReference` de nível superior, exclua todos os ativos:

XML

```
<PackageReference Include="Newtonsoft.Json" Version="9.0.1">
  <ExcludeAssets>All</ExcludeAssets>
</PackageReference>
```

Em seguida, adicione sua própria referência à cópia local apropriada da DLL:

XML

```
<Reference Include="Newtonsoft.Json.dll" />
```

Comentários

Esta página foi útil? Yes No

Fornecer comentários sobre o produto