WiX Toolset v4 および v5 チュートリアル

WiX ツールセット (略して WiX) は、Windows のインストーラーを作成するためのビルド ツール、ランタイム ツール、ライブラリのコレクションです。このチュートリアルでは、Visual Studio、 FireGiant HeatWave、GitHubなどの最新のツールを使用した WiX を紹介します。インストーラーを作成する必要があり、そのために WiX を使用したいと考えている方を想定しています。

このチュートリアル全体を 1 ステップずつ実行できます。また、興味のあるさまざまなトピックに飛び込んでいくこともできます。

システム要件

WiX は、.NET 6.0 以降の SDK を使用する場合、MSBuild SDK および .NET ツールとして利用できます。また、Visual Studio 2019 以降で MSBuild を使用する場合、MSBuild SDK としても利用できます。

このチュートリアルでは Visual Studio 2022 を使用していますが、Visual Studio 2019 を使用することもできます。あるいは Visual Studio をまったく使用しないこともできます。ただし、言葉や手順では、Visual Studio 2022 の最新バージョンの使用を前提としている場合があります。また、実際にコードを操作せずに、ただ従うこともできますが、それでは面白くありません。

これまでの話

このチュートリアルのストーリーは次のとおりです。あなたは、インストーラーを必要とするデスクトップ クライアント アプリケーションを作成しているチームで作業している開発者です。あなたのチームは、古き良き Scrum 開発モデルに従っています。各スプリントで、クライアント アプリケーションを更新し、インストーラーにも一致する更新を行います。構築しているクライアント アプリケーションが非常にシンプルであることに驚かないかもしれません。つまり、できる限りシンプルにすることです。結局のところ、焦点はクライアント アプリケーションではなくインストーラーにあります。

クライアント アプリは C# と WinForms で構築されています。構築するには、 `.NET desktop development` Visual Studio にワークロードがインストールされている必要があります。

この章のページ

スプリント 1: チームへようこそ ›

スプリント1: チームへようこそ

最初のスプリントでは、クライアント アプリのインストーラーを作成する準備をします。この最初のスプリントでは、実際の作業はあまり行いません。作業の大部分は、 WiX ソース コードがどのようなものかを調べ、それがどのように組み立てられているかの基礎を固めるためのスパイク(調査と学習に費やされる時間を表すスクラム用語) です。これは、将来的に機能を追加する際に重要になり、このスプリントで忙しくするには十分です。これが私たちのスプリントのテーマです。スプリントのタスクは次のとおりです。

- HeatWaveをインストールします。
- 空のソリューションを作成します。
- MSI パッケージ プロジェクトを作成します。
- Spike: HeatWave が作成したプロジェクトを詳しく見てみましょう。
- ビルドに成功しました。

私たちのスプリントの終了基準は、実際には非常に簡単な基準の 1 つにすぎません。つまり、構築に成功したということです。今後のスプリントでは、より多くの機能が追加され、少し難しくなります。

この章のページ

‹ スプリント 1: チームへようこそ　上　スプリント 1: HeatWave のインストール ›

スプリント1: HeatWaveをインストールする

私たちは Visual Studio でアプリとそのインストーラーを構築しているので、WiX についてすべて認識している Visual Studio 拡張機能が必要です。幸いなことに、FireGiant の親切なスタッフがまさにそれを持っているのです。それはHeatWave です。

HeatWave は Visual Studio Marketplace から入手できます。

- HeatWave for Visual Studio 2022 をここからダウンロードしてください。
- HeatWave for Visual Studio 2019 をここからダウンロードしてください。

ダウンロードした .vsix ファイルをダブルクリックして、HeatWave をインストールします。

‹ スプリント 1: チームへようこそ　上　スプリント 1: 空白のソリューションを作成する ›

スプリント1: 空白のソリューションを作成する

Visual Studio で空のソリューションを作成するには:

1. 選ぶ `File|New|Project` 。
2. `All languages` 言語ドロップダウンで選択します。
3. `Blank Solution` プロジェクト テンプレート リストから選択します(`blank` 検索ボックスに追加して、スクロールする指の負担を軽減できます)。
4. 選ぶ `Next` 。
5. `WixTutorial` ソリューション名として入力します。
6. Visual Studio のデフォルト ディレクトリを使用すること `Location` も、反対意見がある場合は別のディレクトリを使用することもできます。
7. 選ぶ `Create` 。

‹ スプリント 1: HeatWave のインストール

上

スプリント 1: MSI パッケージ プロジェクトを作成する ›

スプリント 1: MSI パッケージ プロジェクトを作成する

HeatWave を使用して Visual Studio で MSI パッケージ プロジェクトを作成するには:

1. ソリューションを右クリックして を選択します `Add|New Project` 。
2. `WiX` 言語ドロップダウンで選択します。
3. `MSI Package (WiX v4)` プロジェクト テンプレート リストで選択します。
4. 選ぶ `Next` 。
5. `WixTutorialPackage` プロジェクト名として入力します。
6. 選ぶ `Create` 。

‹ スプリント 1: 空白のソリューションを作成する　上　スプリント 1 スパイク: プロジェクトの詳細を見る ›

スプリント1スパイク: プロジェクトを探索する

HeatWave は何を提供してくれたのでしょうか? `MSI Package (WiX v4)` HeatWave のテンプレートは 4 つのファイルを提供します。重要度順に並べると、次のようになります。

- パッケージ.wxs
- フォルダ.wxs
- 例コンポーネント.wxs
- パッケージ.en-us.wxl

それらを見てみましょう。

ファイル拡張子

HeatWave が生成したファイルには、次の 2 つの拡張子があります。

- .wxs (WiX**ソースファイル)** 。
- .wxl (WiX**ローカリゼーションファイル)** 。

> WiX では、WiX ソース ファイルに特定の拡張子を使用するように強制されませんが、デフォルトのままにしておけば作業が楽になります。(たとえば、.wxl 拡張子のファイルを提供すると、WiX はそれをローカリゼーション ファイルとして扱いますが、別の拡張子を使用する場合は、スイッチを使用して WiX にローカリゼーション ファイルであることを通知する必要があります。)

‹ スプリント 1: MSI パッケージ プロジェクトを作成する　上　スプリント 1 スパイク: Package.wxs には何が含まれていますか? ›

スプリント 1 スパイク: Package.wxs には何が含まれていますか?

最初に見るファイルは Package.wxs です。

```
<Wix xmlns="http://wixtoolset.org/schemas/v4/wxs">
```

WiX でパッケージを構築するために記述するソース コードは XML です。(昔は、 X WiX の は を表していました XML 。現在では、WiX は単に WiX を表します。) WiX が XML ファイルをロードすると、次の 2 つの結果が生じます。

- すべての XML ファイルにはルート要素があり、WiX ソース ファイルの場合、そのルート要素はWixです。
- すべての XML 要素には名前空間があり、WiX 要素の場合、その名前空間は です http://wixtoolset.org/schemas/v4/wxs 。

ルート要素と名前空間は、すべての *WiX*ソースファイルで同じです。ルート要素と名前空間が異なる他の WiX ファイル タイプもあります。

> XML 名前空間は、Uniform Resource Identifier (URI)であり、 Uniform Resource Locator (URL)のように見えますが、実際には URL ではありません。WiX は、すべて で始まる複数の名前空間を使用します http://wixtoolset.org/schemas/ 。ただし、名前空間名は有効な URL ではありません。WiX スキーマ ドキュメントは、名前空間 URI に一致する URL にあると思われるかもしれませんが、実際には https://wixtoolset.org/docs/schema/にあります。

パッケージ

```
<Package
```

MSI パッケージを記述する WiX 要素は...ちょっと待ってください... MSI パッケージに関するすべてを定義する属性と子要素を持っています Package 。 Package

```
Name="WixTutorialPackage"
Manufacturer="TODO Manufacturer"
Version="1.0.0.0"
UpgradeCode="64deef2a-cf99-4a0c-be41-5faa802a9502">
```

それでは、属性から始めましょう。HeatWave テンプレートは、すべての MSI パッケージに必要な属性を入力します。この場合、必須とは、それらの属性がないと WiX がエラーを起こすことを意味します。

> 細かいことにこだわる人のために言っておくと、属性のうち 3 つは必ず存在しなければならず、いずれかが欠けていると、WiX はエラーをスローします。4 番目の属性が欠けていると、エラーではなく警告が表示されます。したがって、技術的には必須ではありませんが、ご安心ください。警告メッセージは痛烈なので、リスクを冒さない方がよいでしょう。

- 属性 Name はパッケージの名前を設定します。この名前は、他の場所でも使用されますが、 Installed apps リストに表示される名前としても使用されます。HeatWave テンプレートは、 Name 作成したプロジェクトの名前を に設定します。これは妥当なデフォルトですが、表示される製品名はマーケティング タイプに大きく影響されることがよくあります。この属性については後でもう一度説明します。

> この Installed apps リストにはさまざまな名前が付けられてきました。セットアップを長い間行ってきた人なら、 Add/Remove Programs Windows 95 の頃のことを覚えているでしょう。誰かが と言うのを聞いたら ARP 、

> セットアップ界のベテランの相手をしていることがわかります。

- この `Manufacturer` 属性は、インストーラーに含まれるソフトウェアを作成した会社の名前 (通常) を設定します。この文字列は ARP にも表示されます...それだけです。値に魔法のようなものはなく `Manufacturer` 、単なる文字列です。

> 特に大企業では、チームごとに異なる値を指定することが驚くほど一般的です `Manufacturer` 。(名前を挙げずに言うと、現在 `Foo` 、プログラムと機能には `Foo Corporation` 、、、が表示されています `Foo(R) Corporation` 。) 会社の実際の名前が何であるかを確認するには、弁護士に相談することをお勧めします。

- 属性 `Version` はパッケージのバージョンを設定します。パッケージ バージョンは、これまで見てきた他の属性と同様に、通常は ARP に表示されますが、見た目を美しくする以外にもさまざまな目的で使用されます。パッケージ バージョンは、たとえばパッケージ間のアップグレードを管理する上で重要な役割を果たします。パッケージ バージョンについては、今後必ず再度取り上げます。

> Windows インストーラーには、パッケージのバージョンに関する奇妙なルールがあります。まず、高度なセマンティック バージョン管理は期待しないでください。MSI は 90 年代後半 (正確には 1990 年代) に構築されたため、バージョンに関する考え方は同様に古いものです。バージョンは通常、4 つの部分 (つまり 0 から `major.minor.build.patch` 65535 までの範囲) で構成されます。MSI は、パッケージ バージョンに対してさらに厳格なルールを適用します。パッケージ バージョンは**3 つ**の部分 (`major.minor.build` つまり 0 から255 まで `major` の `minor` 範囲) で構成されます。 Windows Installer SDK にはこれらの制限についての説明はありませんが、90 年代にコンピュータに数メガバイトの RAM と数ギガバイトの回転ディスク領域があったときに個々のバイトに細心の注意を払っていたとしたら、2 つの 8 ビット整数 (0..255) と 1 つの 16 ビット整数 (0..65535) を 1 つの 32 ビット整数に圧縮すると、4 つの 16 ビット整数よりも 32 ビットが節約されることを指摘しておく価値があります。

- この `UpgradeCode` 属性は興味深いものです。まず、パッケージのアップグレード コードは GUID (「グローバルに一意の識別子」) であることを認識することが重要です。Windows インストーラーは GUID を頻繁に使用しますが、WiX は GUID をユーザーから隠すためにできる限りのことをします。アップグレード コードは、互いに「関連」しているパッケージを識別します。典型的な関係は、同じソフトウェア製品の異なるバージョンのパッケージです。(前の箇条書きで「パッケージ バージョンについては必ず再検討する」と述べたことを覚えていますか?) 今のところは、HeatWave テンプレートから提供されたアップグレード コード (新しく生成された GUID) を使用します。

> 重要な注意*: GUID* の発音は が一番です `goo'-id` 。これは「goo」という単語が含まれているためです。言葉遊びを好まない人は「gwid」と言うことを好むかもしれません。Windows インストーラーが GUID を使用するのは、ランダムかつ一意に生成するための明確なルールがあるためです。また、一意の識別子としては比較的コンパクトです。人間が独自の一意の識別子を作成しなければならない場合、おそらく長い文字列になりますが、GUID は常に という形式で 38 文字です `{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}` 。確かに、トレードオフとして、これらの重要な一意の識別子は発音しにくく、視覚的にスキャンしにくいという点があります。

メジャーアップグレード

```
<MajorUpgrade
  DowngradeErrorMessage="!(loc.DowngradeError)" />
```

以前、 `UpgradeCode` 属性について話したのを覚えていますか? `MajorUpgrade` 要素はアップグレード コードを結び付け、新しいバージョンをインストールするときに、まず以前のバージョンが削除され、インストールの最後に新しいバージョンが残るようにします。 `DowngradeErrorMessage` 属性を使用すると、ユーザーが製品をダウングレードしようとした場合 (すでにインストールされているバージョンよりも低いバージョンをインストールしようとした場合) に表示されるメッセージを指定できます。構文は、ローカリゼーション文字列 `!(loc.DowngradeError)` への参照です。ローカリゼーション文字列を使用すると、WiX オーサリングを変更することなく、サポートするすべての言語に対して複数のパッケージを作成できます。HeatWave は、ローカリゼーション文字列がぎっしり詰まった独自のファイルを提供します。これについては後で説明します。

> Windows インストーラーは、アップグレードを幅広くサポートしています。 `MajorUpgrade` 要素には、アップグレードの動作を制御するために使用できる一連の属性があります。デフォルトでは、一般的なアップグレードの動作が提供されます。ダウングレードはエラー メッセージでブロックされ、アップグレードするには製品の言語が一致している必要があり、コンポーネントや機能の変更に最大限の柔軟性を提供するためにアップグレードは早期に実行されます。

特徴

```
<Feature Id="Main">
```

MSI では、パッケージ内に機能の階層を設けて、インストールされるものを制御できます。最近では、機能が表示されることはあまりありませんが、MSI では少なくとも 1 つの機能が必要です。HeatWave テンプレートでは、その 1 つの機能が提供されます。

> 機能の典型的な例は、昔 (私たち全員が Y2K を生き延びた直後) の Microsoft Office で、多くのブランチを持つ機能ツリーがありました。最上位レベルでは、Office に付属する多くのアプリのどれをインストールするかを制御できます。たとえば、ハード ドライブが小さく、プレゼンテーションをまったく行わない場合は、PowerPoint をインストールしないという選択をするかもしれません。今日では、ハード ドライブは容量が大きく、オプションが少ないためテストが容易なので、多くの開発者は機能を公開することに煩わされません。

```
<ComponentGroupRef Id="ExampleComponents" />
```

ここでは、最初の XML 親子関係を示します。 `ComponentGroupRef` 要素は要素の子です `Feature` 。

> 親/子という用語に加えて、要素が他の要素の下にある、または他の要素の下にネストされているという話も聞くでしょう。これらはすべて同じです。

WiX (および一般的な XML) では、ネストされた要素は親要素と子要素の間の何らかの関係を示します。この場合、要素は要素 `ComponentGroupRef` によって記述された機能を「埋める」ために使用されます `Feature` 。

では、フィーチャーはどのようにして埋められるのでしょうか? その答えを得るには、コンポーネント、参照、およびグループについて説明する必要があります。

コンポーネント

MSI は、ファイルやレジストリ値などの個々の「リソース」を追跡しません。代わりに、それらを**コンポーネント**にグループ化します。コンポーネントは複数のファイルとレジストリ値 (および他のいくつかのもの) をグループ化できるため、MSI はそれらを 1 つのものとして扱うことができます。ユーザー側で少し記録が必要になりますが、1 つの優れた利点があります。MSI は常にコンポーネントのすべてのリソースを一度にインストールおよびアンインストールするため、関連するリソースを同じコンポーネントに保持しておけば、マシンが必要なものの一部だけを持つ状態になることはありません。

WiX は、XML の親子関係を使用してコンポーネントを組み立てます。や `Component` などの子要素の親として機能する要素があります。 このトピックについては、後ほど詳しく説明します。 `File` `RegistryValue`

参考文献

コードを書くと、最終的には (潜在的に大きな) 単一の .exe ファイルになることもありますが、通常は何らかのロジックやルールを使用してソース コードを複数のファイルに分割します。100 万行のソース コードは、どのように分割しても扱いにくいものですが、その数百万行を数百のファイルに分割すると便利です。

同じことが WiX コードにも当てはまります。MSI パッケージは、ほぼ任意の複雑な製品をインストールできる (潜在的に大きい) 単一ファイルになる場合がありますが、WiX コードを複数のファイルに分割すると、精神的負荷が軽減

されます。

> WiX コードは、XML で書かれており、一般的な手続き型プログラミング言語で書かれていないにもかかわらず、ソース コードです。WiX 言語で書かれたコードと WiX 自体を構成するコードを区別するために、「WiX オーサリング」と呼ばれることがあります。「WiX スクリプト」とは呼ばないでください。スクリプトではないため、コードの意味を誤解することになります。周囲でスクリプトと呼ぶ人がいたら、その人をブロックして、そのネガティブな言葉をあなたの人生から排除してください。

一般的なプログラミング言語では、あるファイルのコードが別のファイルの関数を呼び出すことで、別のソース コード ファイルを接続します。WiX での (非常に大まかな) 同等のことは、他の場所の要素を参照する要素を使用することです。これは、同じソース ファイル内にある場合もあれば、どこかのライブラリに隠れている場合もあります。参照は、WiX に参照先の「もの」を取得して、参照が存在する場所に取り込むように指示します。この場合、はWiX に、同じ を持つ を `ComponentGroupRef` 検索するように指示します。親要素は であるため、指定されたコンポーネント グループ内のすべてのコンポーネントが機能に割り当てられます。(そのコンポーネント グループは、HeatWave が作成した別のファイルにあります。これについては後で説明します。) `ComponentGroup Id Feature`

> `Ref` 一般的に、WiX での参照は、 や のように で終わる名前を持つ要素 `ComponentGroupRef` と `DirectoryRef` 、 や の `Ref` ようにで終わる名前を持つ属性で表現されます。名前が で終わらない場合でも、参照は常に明示的です。たとえば、要素には、他の場所で定義された要素への参照である属性があります。
> `BinaryRef FileRef Ref Component Directory  Directory`

コンポーネントグループとは何でしょうか?

グループ

WiX コードを複数のファイルに分割できることに加えて、WiX 言語には、MSI 自体がサポートする方法を超えた方法でコードを構造化できる要素があります。たとえば、MSI ではコンポーネントを機能にグループ化できますが、それだけです。コンポーネントをグループ化する他のメカニズムはありません。WiX では、コンポーネントをグループ化する方法が多数ある必要があるため、で定義し、 で参照するコンポーネント グループがあります。高度なシナリオでは、コンポーネント グループを使用して多くのコードの重複を回避できます。WiX がコードをビルドすると、それらの参照がすべて、たとえば機能内のコンポーネントのリストにフラット化されます。
`ComponentGroup ComponentGroupRef`

> *WiX* の開発者は柔軟性を重視しているため、WiX には多数のグループ化構造があることはおそらく驚きではないでしょう。

まとめ

```
    </Feature>
  </Package>
</Wix>
```

WiX 言語は *XML* で表現されるため、*XML* のルールに従う必要があります。ここでのルールは、要素を終了タグで閉じる必要があるというものです(構文)。 `</Element>`

‹ スプリント 1 スパイク: プロジェクトを探索する　上　スプリント 1 スパイク: Folders.wxs には何が含まれていますか? ›

スプリント 1 スパイク: Folders.wxs には何が含まれていますか?

Folders.wxs は別の *WiX* ソース ファイルで、名前が示すように、インストーラーで使用されるフォルダーの *WiX* ソース コードが含まれています。すべてのフォルダーを含める必要はありません。通常、複数回使用されるディレクトリは、重複を避けるために一元化できます。

> これは小さなファイルなので、ソース コードを独自のファイルに分離するのはやりすぎのように思えるかもしれません。少なくとも、現時点でファイルに含まれているものに関してはやりすぎです。しかし、インストーラーが複雑になるにつれて、「コンセプト」ごとに分離することは、コードを分割する良い方法の 1 つになります。

```
<Wix xmlns="http://wixtoolset.org/schemas/v4/wxs">
```

Folders.wxs には、Package.wxs と同じルート要素と名前空間があります。これは、すべての WiX ソース ファイルに当てはまります。(他のルート要素と名前空間がいくつかありますが、これについては後で説明します。)

```
<Fragment>
```

Package.wxs に戻ると、メイン要素 (の下 `Wix`) は です `Package` 。これは、WiX にパッケージを作成中であることを伝えます。ここで、メイン要素は です `Fragment` 。これは、WiX に何かを作成中であることを伝えます。フラグメントとは何でしょうか。WiX では、フラグメントはオーサリングを整理する方法にすぎません。ここで、Folders.wxs には、Package.wxs から参照されるオーサリングを含むフラグメントが 1 つあります。

> フラグメント内の何かを参照すると、WiX はそのフラグメント内のすべてのものを含めます。その理由は、 のような 1 つの参照を持ち `ComponentGroupRef` 、そのコンポーネント グループを構成するすべてのものを取得できるからです。つまり、一般にフラグメントには、互いに関連するものだけを含める必要があります。

`Fragment` 1 つの .wxs ファイルに複数の を含めることができます。また、要素 `Fragment` と同じ .wxs ファイルに複数の を含めることもできます `Package` 。

> 技術的には、 `Package` と `Fragment` (および後で説明する他のいくつかの) は、WiX がセクションと呼ぶものを導入します。 などの一部のセクションは `Package` エントリ セクションと呼ばれ、どのようなものが構築されるかを識別するため特別です。一度に構築できるエントリ セクションは 1 つだけです。フラグメント セクションはそれほど特別ではありません。一度に任意の数のフラグメントを構築できます。

```
<StandardDirectory Id="ProgramFiles6432Folder">
```

要素 `StandardDirectory` を使用すると、いわゆる標準ディレクトリの 1 つをパッケージのディレクトリの親ディレクトリとして使用できます。親ディレクトリとして使用できるディレクトリは多数あります。そのほとんどはMSI システム フォルダのプロパティ IDです。残りは WiX 標準ディレクトリです。

- CommonFiles6432フォルダー
- プログラムファイル6432フォルダー
- システム6432フォルダ

これらの ID は MSI には存在しないという点で特別です。代わりに、WiX はパッケージのビット数に基づいて適切なディレクトリを選択します。

> 「ビット数」は実際の単語ではないと主張する人もいますが、そのような人に対しては、私たちは攻撃的な態度で目を丸くします。代わりに「アーキテクチャ」を使用することもできますが、この場合、ディレクトリが 32 ビット コードの Program Files ディレクトリであるか、64 ビット コードの Program Files ディレクトリであるかが唯一の問題です。したがって、「ビット数」は技術的に正しいです (最高の種類の正確さ)。

したがって、 は、 `ProgramFiles6432Folder` パッケージのビット数に基づいて標準の MSI ディレクトリの 1 つに変換されます。32 ビット パッケージでは、は `ProgramFiles6432Folder` として解決され、これは `ProgramFilesFolder` 通常です `C:\Program Files (x86)` 。64 ビット パッケージでは、は `ProgramFiles6432Folder` として解決され `ProgramFiles64Folder` 、これは通常 です `C:\Program Files` 。

HeatWave は、 *x86*、*x64*、または*Arm64*のパッケージを提供しましたが、デフォルトは*x86*なので、別のアーキテクチャを選択するまでは、x86 用の 32 ビット MSI パッケージを使用します。

```
<Directory Id="INSTALLFOLDER"
  Name="!(bind.Property.Manufacturer) !(bind.Property.ProductName)" />
```

この `Directory` 要素は、親要素で識別されるディレクトリの子ディレクトリとして新しいディレクトリを作成します。

> 厳密に言うと、ディレクトリをリストするだけでは、実際にディスク上にディレクトリが作成されるわけではありません。代わりに、MSI はファイルをインストールするために必要に応じてディレクトリを作成します。空のディレクトリを明示的に作成することもできます。

属性 `Id` を使用すると、このディレクトリを参照するときに使用する ID を指定できます。たとえば、このディレクトリにインストールするファイルが多数ある場合は、ディレクトリの ID を取得する `ComponentGroup` 属性を取得し `Directory` 、そのディレクトリをコンポーネント グループ内のすべてのファイルのホームとして使用する 要素を使用できます。

属性 `Name` は、ほとんどの人が推測するように、ディスク上のディレクトリの名前です。通常、これは のような単純な文字列です `My Awesome Product` 。ただし、HeatWave テンプレートでは、単純な文字列からかなり離れたものが提供されました。

Package.wxs の調査で、HeatWave が `Package` いくつかの属性を持つ要素を提供したのを覚えているかもしれません。その中には次のようなものがありました。

```
<Package
  Name="WixTutorialPackage"
  Manufacturer="TODO Manufacturer"
```

WiX では、これらの属性を参照できます (ただし、間接的です)。これらの動作の詳細については後で説明しますが、今のところは、構文が属性 `!(bind.Property.Manufacturer)` の値を取得し `Package/Manufacturer` 、属性 `!(bind.Property.ProductName)` の値を取得すると理解しておきましょう `Package/Name` 。

それで

```
<Directory Id="INSTALLFOLDER"
  Name="!(bind.Property.Manufacturer) !(bind.Property.ProductName)" />
```

は、

```
<Directory Id="INSTALLFOLDER"
  Name="TODO Manufacturer WixTutorialPackage" />
```

> 奇妙な構文を使う価値はありましたか? おそらくそうです。重複を避けることは通常良いことです。 属性 `Manufacturer` と `Name` 属性に加えた変更は、ディレクトリ名に自動的に反映されます。

```
    </StandardDirectory>
  </Fragment>
</Wix>
```

新しいことは何もありません。XML ファイルではすべての要素が閉じられている必要があります。

スプリント 1 スパイク: Package.wxs には何が含まれていますか?

上

スプリント 1 スパイク: ExampleComponents.wxs には何が含まれていますか? ›

スプリント 1 スパイク: ExampleComponents.wxs には何が含まれていますか?

HeatWave のパッケージ テンプレートにはインストール済みファイルが含まれています。これは、ファイルのインストールがインストーラーにとって非常に一般的な作業であるからです。HeatWave は超能力者ではないので、何をインストールしたいのか推測できません。そのため、プレースホルダー ファイルをインストールするだけです。今のところはこれで十分です。すぐに、実際に何かをインストールします。

```
<Wix xmlns="http://wixtoolset.org/schemas/v4/wxs">
  <Fragment>
```

ExampleComponents.wxs はフラグメントを含む別の WiX ソース ファイルです。

```
<ComponentGroup Id="ExampleComponents" Directory="INSTALLFOLDER">
```

以前説明したコンポーネント グループの具体的な例を示します。実際、これは、 `ComponentGroupRef` Package.wxs の要素の下にある要素で参照したのとまったく同じコンポーネント グループです `Feature` 。つまり、すべてのコンポーネント (このプレースホルダーの場合は 1 つ) がグループ化され、親機能の下に 1 つのピースとして取り込まれます。

> WiX グループには、必ずしも MSI 内に相当するものがあるわけではないことに注意してください。 `ComponentGroup` たとえば、 は、どの機能に組み込むかをすぐに気にすることなく、コンポーネントのグループを簡単に構築できるようにするための WiX の概念です。他のコンポーネント グループからコンポーネント グループを構築することもできます。 `ComponentGroupRef` の下でを使用します `ComponentGroup` 。

```
<Component>
  <File Source="ExampleComponents.wxs" />
</Component>
```

グランド コンポーネント グループ HeatWave には、ExampleComponents というコンポーネントが 1 つあります。そのコンポーネントには 1 つのファイルがあります。そのファイルの名前は ExampleComponents.wxs です。そうです、つまり、このパッケージがインストールする唯一のファイルは、パッケージの作成に使用された WiX ソース ファイルのコピーです。これは HeatWave テンプレート デザイナーがちょっと遊んでいるだけなのでしょうか? おそらくそうでしょう。ただし、このファイル (およびコンポーネント) (およびコンポーネント グループ) は単なるプレースホルダーであり、もっと優れたものに置き換えられるのを待っていることを忘れないでください。もうすぐです!

‹ スプリント 1 スパイク: Folders.wxs には何が含まれていますか?

上

スプリント 1 スパイク: Package.en-us.wxl には何が含まれていますか? ›

スプリント 1 スパイク: Package.en-us.wxl には何が含まれていますか?

HeatWave が提供してくれた 4 番目で最後のファイルは、WiX ソース ファイルではない最初のファイルです。これはローカリゼーション ファイルで、WiX ビルド プロセス中に処理され、さまざまな言語のパッケージを作成できるようにします。現時点では、ローカリゼーション ファイルは 1 つあり、それは米国英語です。ローカリゼーション ファイルは、言語が処理できる限りいくつでも使用できます。

```
<!--
This file contains the declaration of all the localizable strings.
-->
```

注目すべき最初の点: XML ファイルにコメントを含めることができます。コメントは以前から便利だったかもしれませんが、今は利用できるものを利用することにします。

```
<WixLocalization xmlns="http://wixtoolset.org/schemas/v4/wxl" Culture="en-US">
```

ローカリゼーション ファイルは WiX ソース ファイルではないため、異なるルート要素と名前空間を使用します。`Culture` 属性は、ローカリゼーションで使用されるカルチャ (言語と地域) を識別します。

> `Culture` は、インターネット標準に準拠する.NET `CultureInfo` クラスのルールに従うカルチャの名前です。

```
<String Id="DowngradeError" Value="A newer version of [ProductName] is already installed." />
```

ローカリゼーション ファイルには、ほとんどの場合 (通常) 一連の文字列が含まれています。各文字列には `WixLocalization/@Culture` 、属性で指定されたカルチャでローカライズされた文字列である ID と値があります。WiX では、ハードコードされた文字列がサポートされている場所であればどこでも、ローカリゼーション文字列を参照できます。この場合、 `MajorUpgrade` 要素のエラー メッセージを提供するために、Package.wxs で DowngradeError 文字列が使用されています。

```
<MajorUpgrade
  DowngradeErrorMessage="!(loc.DowngradeError)" />
```

上で説明したように、 `!(loc.DowngradeError)` 構文は ID によってローカリゼーション文字列を参照する方法です。米国英語のパッケージをビルドする場合、WiX は culture を含むローカリゼーション ファイルの DowngradeError 文字列を使用します。ドイツではドイツ語、日本では日本語、クリンゴン語など、 `en-US` 他の文化でも同様です。 `de-DE ja-JP tlh`

‹ スプリント 1 スパイク: ExampleComponents.wxs には何が含まれていますか?

上

スプリント 1: プロジェクトの構築 ›

スプリント1: プロジェクトの構築

HeatWave を使用して Visual Studio で MSI パッケージ プロジェクトをビルドするには、 `WixTutorialPackage` ソリューション エクスプローラーでプロジェクトを右クリックし、 を選択します `Build` 。

すべてがうまくいけば、出力ウィンドウには次のような内容が表示されます。

```
Build started...
1>------ Build started: Project: WixTutorialPackage, Configuration: Debug x86 ------
1>WixTutorialPackage -> x:\path\to\WixTutorialPackage\bin\x86\Debug\en-US\WixTutorialPackage.msi
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
========== Build started at 9:01 PM and took 01.730 seconds ==========
```

‹ スプリント 1 スパイク: Package.en-us.wxl には何が含まれていますか?　上　スプリント 1 の振り返り ›

スプリント 1 の振り返り

スプリントの振り返りはスプリントを終了し、チームが何がうまくいったか、何がうまくいかなかったか、そしてチームがどのように改善できるかについて正直に話し合う機会を提供します。

何がうまくいきましたか?

インストーラーをすぐに使い始めて、詳細に調べることができました。

もっとうまくいく方法は何でしょうか?

このスプリントでは、実際にはあまり何もしませんでした。特に最初はそれで問題ありませんが、すぐにもっと本格的な作業に取り掛かりましょう。

次のスプリントで何を改善しますか?

アプリのインストーラーを作成しているので、プレースホルダー アプリを用意して実際にインストールする必要があります。これをスプリント 2 のホッパーに入れましょう。

‹ スプリント 1: プロジェクトの構築 上 スプリント 2: アプリのインストール ›