スプリント 5: WiX 拡張機能の使用

最も基本的な意味で、WiX は Windows インストーラー用のパッケージを作成するためのビルド ツールのセットです。しかし、WiX はそれだけではありません。その 1 つは、Windows インストーラーに組み込まれていない機能を追加することです。WiX は、その追加機能を*WiX* 拡張機能にパッケージ化します。名前から推測できるように、WiX 拡張機能は WiX に組み込まれていないため、使用するにはいくつかの追加手順を実行する必要があります。(心配しないでください。WiX を作成した賢くて面白い人々は、パッケージ化を楽しむための WiX 拡張機能も多数作成しました。)

今回のスプリントのタスクは次のとおりです。

- WiX拡張機能が必要
- プロジェクトにWiX拡張機能を追加する
- コードにWiX拡張名前空間を追加する
- WiX拡張機能の呼び出し
- WiX拡張機能の結果を使用する

今回は、前回のスプリントと同様に起動条件を扱っているため、スプリントの終了基準も同様です。起動条件をテストして、次のことを確認します。

1. 目標を達成しました（つまり、要件を満たしました）。
2. 誤って他のものを壊したわけではありません。

この章のページ

‹ スプリント 5: WiX 拡張機能の使用　上　スプリント 5: WiX 拡張機能が必要 ›

スプリント 5: WiX 拡張機能が必要

開発チームは本当に忙しくしています。もちろん、アプリのコードを書くためではなく、多くの「市場調査」を行ってきました。これは主に、上級管理職が最新のガジェットを試用するための口実のようですが、当然ながら経費になります。これらのガジェットへの直接的な対応として、開発チームは、アプリが従来の x64 と最新の人気である Arm64 の両方をサポートすることを決定しました。WiX v4 のリリース ノートをざっと見ると、 WiX が Arm64 をサポートしていることが分かります。したがって、インストーラーで簡単にサポートできるはずです。実際、Windows エコシステムについて詳しい私たちは、Windows 11 では x64 アプリを Arm64 マシンで実行できることを知っています。しかし、Windows 10 は Arm64 マシンで 32 ビット x86 アプリのみをサポートし、64 ビット x64 アプリはサポートしないことも知っています。製品管理チームは、Windows 11 を必須にするか、2025 年にサービスが終了するまで Windows 10 のサポートを継続するかで迷っています。そこで、製品バックログに次の項目を追加しました。インストーラーで一致するプラットフォームを必須にする。つまり、x64 パッケージは x64 マシンにのみインストールでき、Arm64 も同様です。製品マネージャーが、サポートされている Windows 11 Arm64 マシンで x64 インストーラーをブロックしたい理由はよくわかりませんが、製品マネージャーは (間接的に) 費用を負担しているので、彼らの賢明さを疑うのは我々ではありません。

さて、この賢明さに疑問を抱く理由の 1 つは、Arm64 で x64 インストーラーをブロックするのが実際にはかなり難しいことです。Microsoft は、Windows 11 での x64-on-Arm64 エミュレーションを誇りにしており、多くの場所で Arm64 マシンが x64 マシンであるかのように見せかけています。Windows インストーラーには、`Msix64` 「x64 プロセッサで実行されている場合にのみ定義される」という組み込みプロパティがあります。ええ、そうではありません。これは、Arm64 プロセッサの Windows 11 で実行されているときにも設定されます。ご想像のとおり、実際に使用されているプロセッサを識別する必要があるソフトウェアはたくさんあります。そして、もう一度想像してみてください。WiX がそれをカバーしています。WiX には、実行されているマシンの基盤となるプロセッサを検出するコードを含む拡張機能があり、そのコードを使用して不一致をブロックできます。

‹ スプリント 5: WiX 拡張機能の使用 上 スプリント 5: プロジェクトに WiX 拡張機能を追加する ›

スプリント 5: プロジェクトに WiX 拡張機能を追加する

FireGiant の非常に賢くて親切な友人たちの助けにより、必要な WiX 拡張機能は であることがわかりました。実行しているプロセッサの実際のタイプを示すコードを実行する前に、プロジェクトに `WixToolset.Util.wixext` 入る必要があります。 `WixToolset.Util.wixext`

WiX拡張機能はNuGetパッケージです

「WiX 拡張機能は NuGet パッケージである」という話をどこかで聞いたことがあるかもしれませんが、これは正確には正しくありません。WiX 拡張機能は、パッケージをビルドするときに WiX に読み込むように指示するアセンブリにすぎません。WiX 拡張機能には、公開する機能を WiX に指示するコードが含まれており、多くの場合、カスタム アクション コードなどをパッケージに組み込むことができる埋め込みライブラリが含まれています。ただし、これらのアセンブリは、少なくとも WiX 独自の拡張機能については、NuGet パッケージとしてバンドルされており、nuget.org で入手できるため、プロジェクトで使用される可能性のある他の NuGet パッケージと同様に、WiX プロジェクトで参照できます。

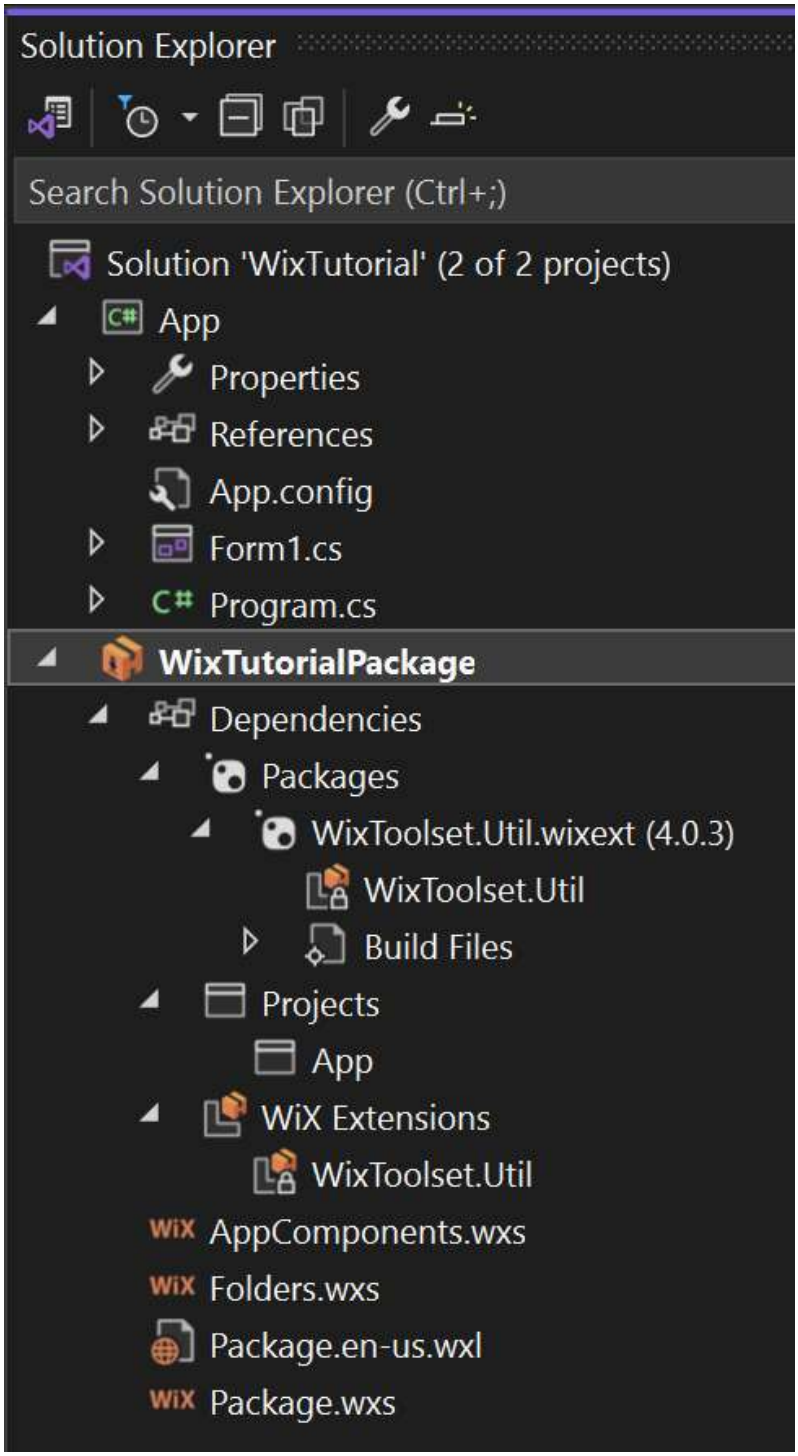
WiX チームが愛情込めて手作りしたすべての WiX 拡張機能は、nuget.org でご覧いただけます。

WiX プロジェクトに WiX 拡張機能を追加する

WiX の拡張機能は nuget.org で NuGet パッケージとして出荷されるため、C# プロジェクトに追加するのとまったく同じ方法で WiX プロジェクトに追加します。

- を右クリックします `WixTutorialPackage` 。
- 選ぶ `Manage NuGet Packages` 。
- `Browse` 左上をクリックします。(プロジェクトに NuGet パッケージがインストールされていないのに、なぜ Visual Studio がそこから起動しないのかと疑問に思う人もいるかもしれません。) (また、「タブ」がボタンやタブではなく、普通のテキストのように見えるのはなぜなのかと疑問に思う人もいるかもしれません。そのような人は、UI の偏屈者と呼ばれることがよくあります。)
- `wixext` 検索ボックスに入力します。
- `WixToolset.Util.wixext` リストで 検索します。
  - メッセージが見つからない、 `WixToolset.Util.wixext` または受信できない場合は、ドロップダウンでまたはが選択されていることを確認してください。 `No packages found nuget.org All Package source`
- `WixToolset.Util.wixext` リストから選択し、 `Install` NuGet パッケージ マネージャー ウィンドウの右側に自動的に表示される詳細パネルをクリックします。
- プロジェクトに `Preview Changes` インストール中であることを示すダイアログが表示される場合があります。(チェックボックスがあるので、この煩わしいダイアログが表示されないように既に設定されている可能性があります。) ダイアログが表示された場合は、 を選択します。 `WixToolset.Util.wixext Don't show this again Apply`

Visual Studio が少し動いて完了すると、ソリューション エクスプローラーのノード `WixToolset.Util.wixext` を見ると、がインストールされていることがわかります `Dependencies` 。



‹ スプリント 5: WiX 拡張機能が必要 上 スプリント 5: コードに WiX 拡張名前空間を追加する ›

スプリント 5: コードに WiX 拡張名前空間を追加する

すでに述べたように、HeatWave から取得する WiX コードと自分で記述する WiX コードはすべて XML で記述されています。また、XML のさまざまな側面の 1 つに、名前空間の概念があります。これは、スプリント 1 で以前指摘した内容の繰り返しです。

> 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。

WiX が「コア」言語に使用する名前空間は です `http://wixtoolset.org/schemas/v4/wxs` 。WiX では、拡張機能が XML 言語要素を追加するときに、独自の一意の名前空間を持つことが求められます。 の名前空間 URI は `WixToolset.Util.wixext` です `http://wixtoolset.org/schemas/v4/wxs/util` 。

したがって、 の要素を使用するには `WixToolset.Util.wixext` 、.wxs ファイルでサポートされている名前空間を追加する必要があります `http://wixtoolset.org/schemas/v4/wxs/util` 。これを行う最も簡単な方法は、ルート要素で名前空間を宣言する `Wix` ことです。

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

名前空間宣言は 3 つの部分に分かれます。

- `xmlns` : 「私は XML 名のスペースです。」
- `util` : これは、この名前空間内の要素に使用するプレフィックスです。
- `http://wixtoolset.org/schemas/v4/wxs/util` : これは名前空間 URI ですが、毎回入力するのは長すぎて面倒なので、代わりにプレフィックスを省略形として使用します。

この機能をパッケージ自体に追加するので、 `Wix` Package.wxs 内の要素を変更します。

スプリント 5: プロジェクトに WiX 拡張機能を追加する 上 スプリント 5: WiX 拡張機能の呼び出し

スプリント5: WiX拡張機能の呼び出し

FireGiant の機知に富んだ魅力的な友人たちは `QueryNativeMachine` 、拡張機能で利用可能な要素を使用する必要があると教えてくれました `WixToolset.Util.wixext` 。非常に簡潔なドキュメント `QueryNativeMachine` には、次のように指定されたプロパティを設定すると書かれています `WIX_NATIVE_MACHINE` 。

> ...マシンのネイティブ アーキテクチャを表すイメージ ファイル マシン値。このプロパティは、Windows 10 バージョン 1511 (TH2) 以降でのみ設定されます。

値は、基盤となるプロセッサを反映する数値です。

> 明らかに、Microsoft の人々は時が経つにつれてますます楽しみを増していたことに注目してください。32 ビット x86 `IMAGE_FILE_MACHINE_I386` プロセッサの値は、予想される 386 ではなく、一見ランダムな (しかし、おそらく根底に合理的な理由がある) 332 です。しかし、64 ビット x64 `IMAGE_FILE_MACHINE_AMD64` プロセッサの値は 34,404 で、16 進数では 0x8664 となり、AMD の 64 ビット アーキテクチャの元の名前であるx86-64に非常に近いです。同様に、Arm64 プロセッサは、値が 43,620 である IMAGE_FILE_MACHINE_ARM64 として報告され、これも 16 進数では 0xAA64 となり、 AArch64に近くなります。残念ながら、Windows インストーラーは 16 進数でのカウント方法を知らないため、面白くない 10 進数値を使用するしかありません。

では、マジック プロセッサ番号を取得するコードをどのように呼び出すのでしょうか。名前空間プレフィックス (`util`) と拡張機能 `QueryNativeMachine` の要素を組み合わせる `WixToolset.Util.wixext` と、答えが得られます。

```
<util:QueryNativeMachine />
```

Package.wxs は次のようになります。

```
<Wix xmlns="http://wixtoolset.org/schemas/v4/wxs"
     xmlns:util="http://wixtoolset.org/schemas/v4/wxs/util">
  <Package Name="WixTutorialPackage" Manufacturer="Edgerock Concepts" Version="1.0.0.0" UpgradeCode="64deef2a-cf9
9-4a0c-be41-5faa802a9502">
    <MajorUpgrade DowngradeErrorMessage="!(loc.DowngradeError)" />

    <MediaTemplate EmbedCab="yes" />

    <util:QueryNativeMachine />

    <Feature Id="Main">
      <ComponentGroupRef Id="AppComponents" />
    </Feature>
  </Package>
</Wix>
```

パッケージをビルドするときに、 `WixToolset.Util.wixext` 拡張機能はオーサリングを解析し `<util:QueryNativeMachine />` 、拡張機能に含まれるコードをいくつか組み込むように WiX に指示します `WixToolset.Util.wixext` 。このようなコードはカスタム アクションと呼ばれ、ファイルのインストール、レジストリへの書き込み、Windows Me で使用されるカタログのインストールなどの重要な処理を実行するために Windows Installer に組み込まれている標準アクションとは対照的です。(すべての標準アクションが同じように重要であるわけではないことに注意してください。)

カスタム アクションは通常、MSI パッケージに埋め込まれ、適切なタイミングで実行されるようにスケジュールする必要があります。当然、WiX 拡張機能は、コード自体を実行するだけでなく、これらすべてを必要に応じて実行します。

> 自分で作成したカスタム アクションと WiX に付属するカスタム アクションを区別するために、後者は (興味深いことに) *WiX* 標準カスタム アクションと呼ばれることがあります。このフレーズの意味は、自由に解釈してください。

カスタム アクション コードは、 `WIX_NATIVE_MACHINE` プロパティを適切なマジック ナンバーに設定します。

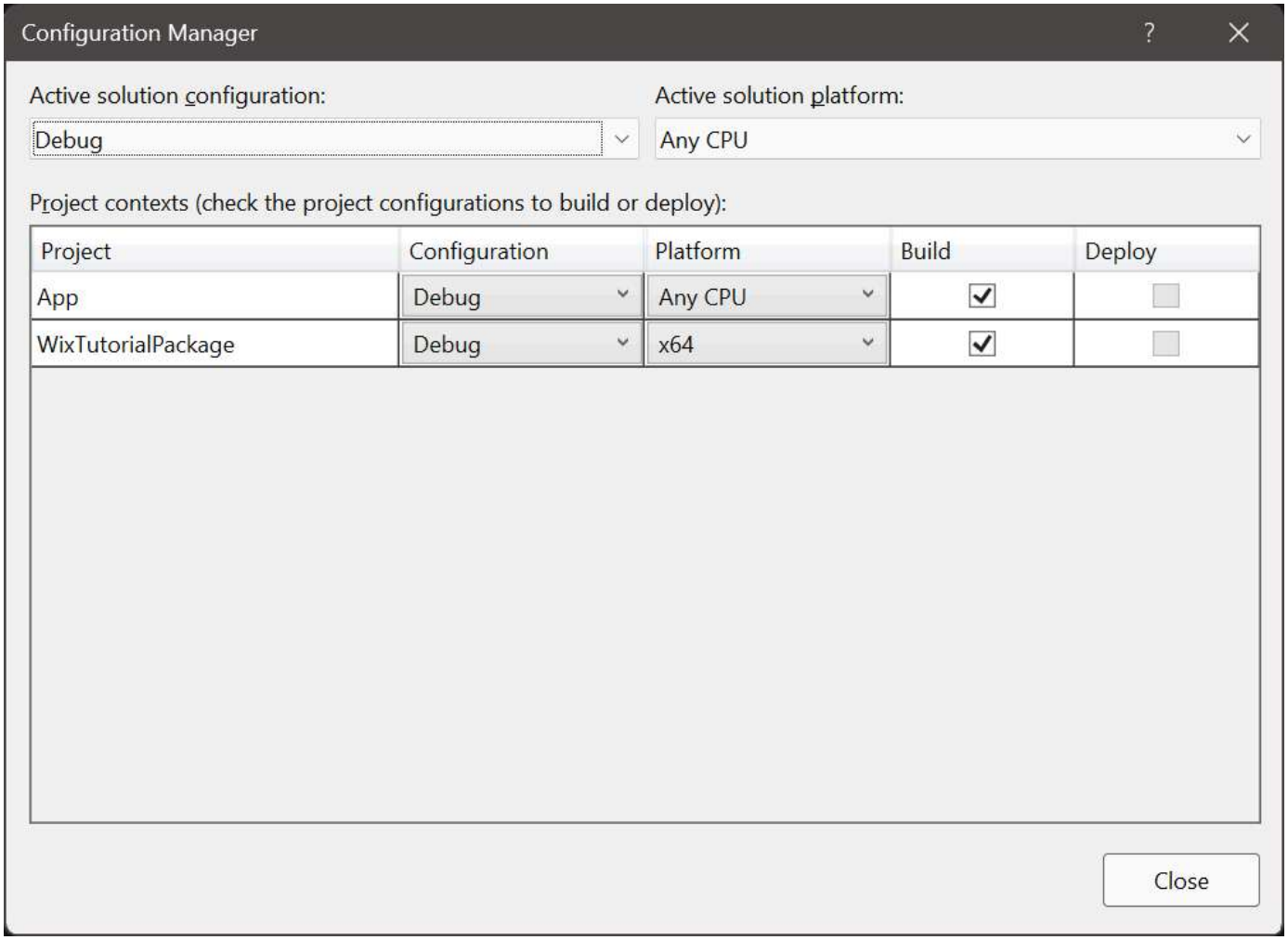
今は、その魔法の数字を何とかして使う必要があります...

‹ スプリント 5: コードに WiX 拡張名前空間を追加する　上　スプリント 5: WiX 拡張機能からの起動条件 ›

スプリント 5: WiX 拡張機能からの起動条件

これで、カスタム アクションによって適切なマジック ナンバーが設定された `WIX_NATIVE_MACHINE` プロパティができました。 `WixToolset.Util.wixext`

経営陣の指示により、私たちの目標は、ユーザーがパッケージを使用するには x64 マシンを実行している必要があることです。考えてみると、パッケージを作成するときにプラットフォームを選択していませんでした。HeatWave はデフォルトで x64 パッケージを作成するように設定されていることがわかりました。Visual Studio 内で `Build` | を選択することで、これを確認できます `Configuration Manager` 。



製品バックログ項目とその概要を覚えておいてください。

> インストーラーで一致するプラットフォームが必要です。つまり、x64 パッケージは x64 マシンにのみインストールでき、Arm64 の場合も同様です。

技術的には、これは (少なくとも) パッケージ (x64 用と Arm64 用) があることを前提としています。今のところ、パッケージは 1 つしかありません。つまり、次の操作を行う必要があります。

1. 現在のパッケージが x64 マシンにのみインストールされることを確認してください。
2. 理論的には、将来の Arm64 パッケージが Arm64 マシンにのみインストールされることを保証します。
3. 将来的には、x64 用に 1 つのパッケージと Arm64 用に別のパッケージを実際に作成します。

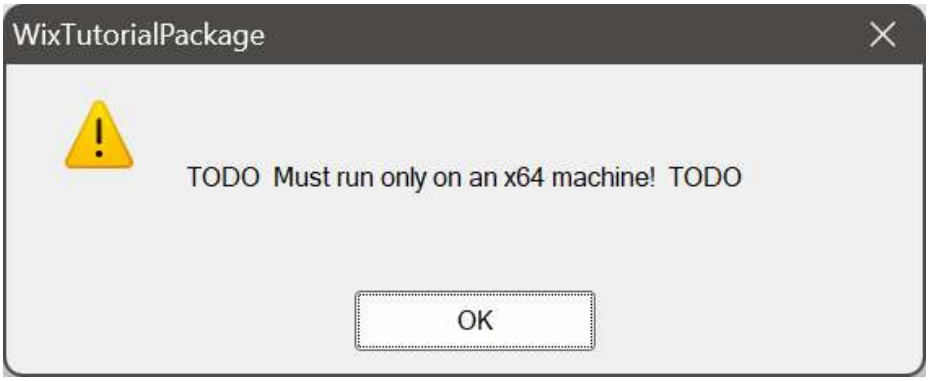
このスプリントでは最初の 2 つに対処します。

スプリント 4 では、組み込みの MSI プロパティを使用して起動条件の式を構築する方法を確認しました。WiX 拡張機能内のコードから設定されたカスタム プロパティを使用してこれを行うのは、ほぼ同じです。

プロパティ値を x64 マジックナンバーと比較すると `WIX_NATIVE_MACHINE` 、起動条件の式を構築できます。

```
<Launch Condition="WIX_NATIVE_MACHINE = 34404" Message="TODO  Must run only on an x64 machine!  TODO" />
```

これを一般的な x64 マシンで試すと、予想どおりインストーラーが実行されます。Arm64 マシンで実行すると、起動条件メッセージが表示されます。



これで完了です。チートをしたいならこれで完了です。ステップ 2 を実行して、Arm64 マシン上の x64 パッケージもブロックする起動条件を記述してみましょう。これが難しすぎる場合は、いつでもチートに頼ることができます。

‹ スプリント 5: WiX 拡張機能の呼び出し　上　スプリント5: より完璧な表現を形成するために ›

スプリント5: より完璧な表現を形成するために

この起動条件はわかっています:

```
<Launch Condition="WIX_NATIVE_MACHINE = 34404" Message="TODO  Must run only on an x64 machine!  TODO" />
```

Arm64 マシン上の現在の x64 パッケージをブロックします。しかし、x64 と Arm64 の両方のパッケージをビルドするときに目標を達成する起動条件 (または複数の起動条件) を思いつくことはできますか? `WIX_NATIVE_MACHINE = 34404` 将来作成する (仮想の) Arm64 パッケージでも、Arm64 マシンで実行すると条件は false になります。

友人の疑似コードをもう一度見てみましょう。次のように動作する条件が必要です。

```
IF IsX64Package == TRUE AND X64Machine == TRUE THEN
  REM OK!
ELSE IF IsArm64Package == TRUE AND Arm64Machine == TRUE THEN
  REM OK!
ELSE
  REM BLOCK!
ENDIF
```

最初のブロックでは、Windows インストーラーが役に立っていることがわかりました `ELSE` 。Arm64 マシンで x64 パッケージを実行することはできますが、その逆はできません。Arm64 パッケージは x64 マシンでは開きません。代わりに、次のようなメッセージが表示されます。

> 残念ながら、MSI はパッケージをまったく実行しないため、エラー メッセージをカスタマイズする方法はありません。ただし、良い点は、MSI がユーザーが選択した言語にローカライズしてくれることです。

つまり、「これが x64 パッケージである場合、マシンも x64 でなければならない。そうでない場合は無視する」という条件を構築するだけでよいのです。うーん... `IF P THEN Q` 聞き覚えがあります... そうです、大学の真理値表の埃をかぶった奥まったところから、それは論理的含意です。含意はまさにここで必要なものです。x64 パッケージでない場合は、マシンのプラットフォームは気にしません。凝った論理記号を使用すると、式は次のようになります。

`IsX64Package` ⇒ `X64Machine`

> あなたもこれを `IsX64Package` っとして学んだことがあるかもしれません `X64Machine` 。これは私の個人的なお気に入りです。

ほとんどの言語には含意演算子はありませんが、もちろん、universe ではより一般的な演算子から含意演算子を構築できます。

`( ¬ IsX64Package) ∨ X64Machine`

または、ほぼ英語に近い Windows インストーラーの式構文では次のようになります。

`(NOT IsX64Package) OR X64Machine`

しかし、信じられないかもしれませんが、MSI には論理含意演算子があります。式構文のルーツが BASIC (または Basic) にあるため、MSI は `IMP` 演算子を実装しています。

`IsX64Package IMP X64Machine`

> この `IMP` 演算子は少し難解ですが、 `(NOT IsX64Package) OR X64Machine` 条件が何を達成しようとしているのかは明らかに分かりません。より明確ですが難解な演算子を優先しましょう。また、MSI が提供するand演算子を `IMP` 使用する方法にも注目しましょう。 `XOR EQV`

疑似コード `X64Machine` は次のものと同等です `WIX_NATIVE_MACHINE = 34404`：

`IsX64Package IMP WIX_NATIVE_MACHINE = 34404`

を置き換えるにはどうしたらよいでしょうか `IsX64Package` 。パッケージがサポートするプラットフォームを決定する組み込みプロパティがあると思われるかもしれません。残念ながら、その考えは間違っています。しかし、心配はいりません。WiX が対応しています。詳細には立ち入りませんが、WiXプリプロセッサには、 `BUILDARCH` "このパッケージがコンパイルされるプラットフォーム (x86、x64、arm64)" を含む という名前の組み込み変数があります。

> プリプロセッサについては、今後のスプリントでさらに詳しく説明します。プリプロセッサによって、難解な論理演算子の記憶を掘り起こす必要が完全になくなると信じられますか? 確かにそうですが、その楽しさをすべて手放すつもりはないですよね?

最終的に次の状態になります:

```
"$(sys.BUILDARCH)" ~= "x64" IMP WIX_NATIVE_MACHINE = 34404
```

文字列を二重引用符で囲みます (そうしないと、MSI はそれらをプロパティとして検索しようとします)。また、 `~=` MSI が大文字と小文字を区別しない文字列比較を行う演算子も使用します。

> WiX のドキュメントでは小文字 `x64` (および `x86` と `arm64`) を使用しており、WiX ソース コードを簡単に確認すると、実際にドキュメントが正しいことが確認できます。ただし `~=` 、 演算子を使用すると、100% 確信する必要がなくなります。残念ながら、MSI は大文字 `~` と小文字を区別しない文字列比較を意味するためにプレフィックスを使用することを選択しました。これは、 の意味に慣れている場合は少し混乱します `NOT` 。MSI の BASIC のような式方言では、not-equal は と綴られることに注意してください `<>` 。

残りは、Package.wxs で起動条件を準備するだけです。

```
<util:QueryNativeMachine />

<Launch
  Condition='"$(sys.BUILDARCH)" ~= "x64" IMP WIX_NATIVE_MACHINE = 34404'
  Message="TODO  This package can run only on $(sys.BUILDARCH) machines!  TODO" />
```

‹ スプリント 5: WiX 拡張機能からの起動条件 上 スプリント 5: テスト ›

スプリント5: テスト

スプリント 4 と同様に、起動条件をテストしており、起動条件は早期に評価されるため、少し危険を冒しても、テストに Windows Sandbox を使用しなくてもよいことがわかっています。

スプリント 4 の起動条件でも、同じ種類の問題があります。x64 マシン上の x64 パッケージで通常の動作が壊れていないことをテストすることはできます (実際、すでにテスト済みです)。しかし、Arm64 マシン上の x64 パッケージをブロックしたことを確認するのは、私たちのような低レベルの人間は Arm64 マシンには不適格であるため、少し困難です。実行できることは 2 つあります。

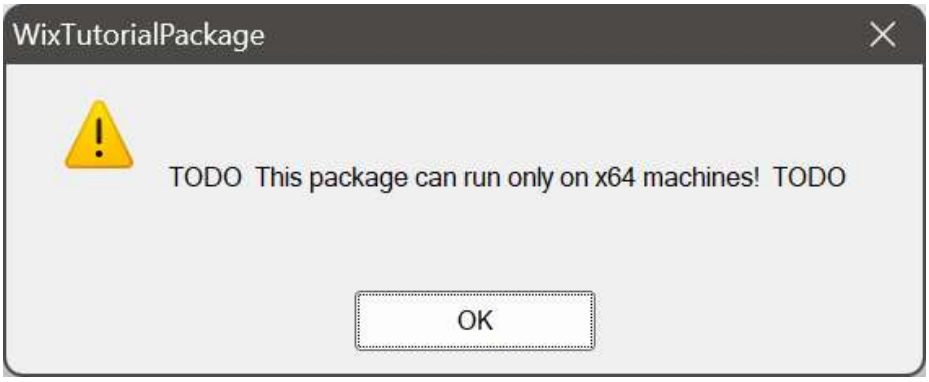
1. カンニング。結局、私たちはそれがとても上手になったんです。
2. 上級者の 1 人に Arm64 デバイスの 1 つを借りるよう罪悪感を抱かせてみてください。

まずは不正行為から始めましょう。

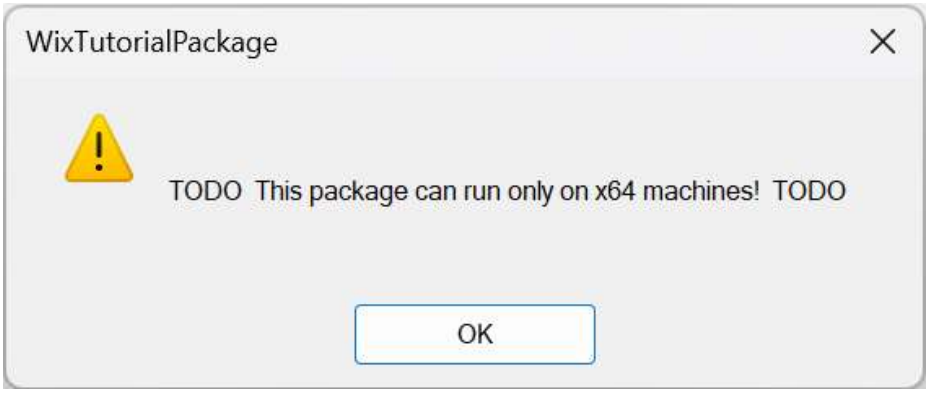
現在の起動条件は次のとおりです。

```
<Launch
  Condition='"$(sys.BUILDARCH)" ~= "x64" IMP WIX_NATIVE_MACHINE = 34404'
  Message="TODO  This package can run only on $(sys.BUILDARCH) machines!  TODO" />
```

`34404` を に変更するだけで済みます `43620` 。これは Arm64 マジックナンバーです。これにより `WIX_NATIVE_MACHINE` 、実際の Arm64 マシンで返されるものを模倣できます。結果は、予想どおり、まさに私たちが期待していたとおりです。



そして、人間味あふれる上級開発者の一人に少し悲しそうな目で見つめられ、何も壊さないと心に誓った後、私たちは不正テストを元に戻し、本物の Arm64 デバイスでパッケージを試す機会を得ました。結果は興味深いものでした。



ミームにあるように、企業はこれら 2 つのエラー メッセージの違いを見つけてほしいと思っています。実際には、これらは同じものです。

スプリント5: より完璧な表現を形成するために

上

スプリント 5 の振り返り ›

スプリント 5 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与えます。スプリント 5 では、どうでしたか?

何がうまくいきましたか?

WiX 拡張機能の使用を開始し、利用可能なセットアップ ツールに関する知識が広がりました。

もっとうまくいく方法は何でしょうか?

私たちは、仕事に必要なすべてのものが揃っていないことの影響をまだ経験しています。時にはごまかしてやり過ごすこともできますが、それは私たちが行っている仕事を適切にテストする能力に影響を与えています。適切なテストがなければ、大量の技術的負債が蓄積され、それが私たちに跳ね返ってきて、不都合な時間に長時間働き、燃え尽きてしまう夜や週末を過ごすことになります。それを避けましょう。

次のスプリントで何を改善しますか?

私たちは自ら主張し、適切なテストを行うために必要なソフトウェアとハードウェアにアクセスする必要があります。強い言葉で書かれた電子メールを送る時期です...

‹ スプリント 5: テスト 上 製品バックログ ›