スプリント4: ブロッキングパッケージ

チームの予算が拡大し続ける中 (上級チーム ミーティング用のエビ料理の予算は、下級チーム ミーティング用のチリケータリングの予算より桁違いに大きい)、金持ちの人たちは収益を増やす方法を模索しています。彼らはまだ詳細を詰めているところですが、噂によると、高価な「究極」エディション (コード名: Project FOMO) をまとめて、「クラシック」エディションを制限する必要があるようです。製品バックログに課されている制限の 1 つは、Windows の Server エディションに製品をインストールできないようにすることです。これらのエディションを作成し、エンタープライズ エディションの制限を緩和する必要がありますが、その作業は後で行います。

したがって、次の方法が必要です:

1. Windows の Server エディションで実行されていることを検知します。
2. そうであれば、メッセージを表示してインストーラを終了します。

嬉しいことに、Windows インストーラーは、その両方を実行できます。MSI には、パッケージが実行されている Windows のエディションを検出できる組み込みプロパティが多数あり、ユーザーが指定した条件が満たされない場合にパッケージをブロックする機能 (起動条件と呼ばれる) もあります。

今回のスプリントのタスクは次のとおりです。

- Windows Server を識別する適切なプロパティを見つけます。
- 適切な条件とユーザーに対する明確なメッセージを使用して起動条件を作成します。

今回のスプリント終了基準はより厳格になっています。意図した動作が機能し、障害が発生していないことを確認する必要があります。(障害の発生を気にしなければ、はるかに簡単ではないでしょうか?)

この章のページ

スプリント4: プロパティについて
スプリント4: 条件と式
スプリント4: ブロックするかしないか
スプリント4: 起動条件の作成
スプリント4: テスト
スプリント4の振り返り

‹ スプリント 4: ブロッキング パッケージ 上 スプリント4: プロパティについて ›

スプリント4: プロパティについて

タスクでプロパティを使用する必要があることを考えると、プロパティとは何かと疑問に思うかもしれません。Windows インストーラーのドキュメントの 1 文の定義では、プロパティは*Windows インストーラーがインストール中に使用するグローバル変数である*とされています。一般的なプログラミング言語の変数と同様に、プロパティから値を取得して、何らかのロジックで使用することができます。また、ビルド時にプロパティ値をハードコード値に設定し、実行時に「計算された」値に設定することもできます。MSI がプロパティ値を設定することもあります。(ネタバレ注意! カスタム アクション コードを含む WiX 拡張機能もプロパティ値を設定できます。)

今のところ、MSI が設定するプロパティに焦点を当てています。MSI には多くの組み込みプロパティがあります。就寝前にちょっとした読み物をお探しの場合は、ここでそれらすべての公式ドキュメントを参照できます。ここには興味深いプロパティがたくさんありますが、最も興味深いのは、オペレーティング システムのプロパティセクションにリストされています。かなり前に廃止された Microsoft 製品のサポートに加えて (BackOffice を覚えているなら、帯状疱疹のワクチンを接種する時期は過ぎているでしょう)、「Windows 2000 以降のオペレーティング システム用」の `MsiNTProductType` プロパティ (!) が次の値に設定されています。

1. Windows 2000 Professional以降
2. Windows 2000 ドメイン コントローラ以降
3. Windows 2000 Server 以降

基礎となる API ドキュメントでは、値が 1 の場合、次のことが明確にされています。

> オペレーティング システムは、Windows 8、Windows 7、Windows Vista、Windows XP Professional、Windows XP Home Edition、または Windows 2000 Professional です。

(以降の OS の Home エディションがカウントされること、および Windows 10 と Windows 11 のエントリが欠落しているのは、ドキュメントの一部が更新されていないことによる単なる副作用であると想定します。)

つまり、 `MsiNTProductType` でない場合は `1` 、何らかの Windows Server 上で実行されているため、インストールをブロックする必要があります。では、これを起動条件にするにはどうすればよいでしょうか。

‹ スプリント 4: ブロッキング パッケージ 上 スプリント4: 条件と式 ›

スプリント4: 条件と式

したがって、組み込み `MsiNTProductType` プロパティが 以外の何か (何でもいい) に設定されているかどうかを検出する必要があります。C、C++、C# などの言語 (または、一般的な中括弧プログラミング言語) では、次の式 `1` になります。

```
MsiNTProductType != 1
```

Windows インストーラーは、典型的なプログラミング言語ではありませんが、それでもそのような表現をサポートしています。

> 一般的に、MSI は条件を幅広くサポートしています。起動条件、コンポーネント条件、機能条件など、後で説明するさまざまな条件が提供されています。(条件をこのように幅広くサポートする理由の 1 つは、MSI が典型的なプログラミング言語ではないことですが、条件をエンジンに組み込むと、**if**ステートメントをシミュレートできます。for ループのサポートは、**まだ**明確ではありません。)

しかし、Windows インストーラーは中括弧プログラミング パラダイムを採用していません。その式構文は、昔ながらの Basic (または、昔の BASIC) や一部の SQL に非常に近いものです。Windows インストーラーSDK で構文の詳細を確認できますが、私たちにとって最も重要な 2 つの違いは、等号演算子が であり `=` 、 ではなく `==` 、不等号演算子が であり `<>` 、 ではないことです `!=` 。

したがって、上記の表現は、MSI 用語では次のようになります。

```
MsiNTProductType <> 1
```

ほとんど。

以前のスプリントで説明したように、WiX 言語は XML で表現されます。XML には多くのルールがあり、一方では XML を、少し冗長ではあるものの優れたマークアップ言語にしていますが、他方では、非常に厄介なものです。そのルールの 1 つは、 `<` と の `>` 文字が特殊であり、要素名を囲むためだけに使用しなければならないというものです。引用符で囲まれた文字列内でも、他の方法では使用できません。代わりに、XML ではエンティティを `<` 使用する必要があります。とのエンティティは、それぞれと `>` です。つまり、は ではなく、 になります。

`< > MsiNTProductType != 1 MsiNTProductType <> 1`

```
MsiNTProductType &lt;&gt; 1
```

‹ スプリント 4: プロパティについて 上 スプリント 4: ブロックするかしないか ›

スプリント4: ブロックするかしないか

私たちの目標は、Windows の Server エディションでパッケージが実行されないようにすることです。Windows インストーラーには起動条件があり、これによって必要なブロッキング動作を実装できますが、起動条件は技術的にはブロッキングではありません。代わりに、起動条件は文字通りに解釈されるものです。つまり、パッケージを起動するために満たす必要がある条件です。条件が満たされない場合は、メッセージが表示されます (パッケージがブロックされます)。したがって、純粋に専門的な観点から言えば、起動条件はブロッキングではなく、ブロッキングではないわけではありません。疑似コードでは、次のように動作します。

```
IF launch_condition = FALSE THEN
  SHOW launch_condition_message
ENDIF
```

起動条件メッセージが起動条件式と一致していないため、少し逆になっています。

しかし、待ってください。必要なブロッキング条件は `MsiNTProductType <> 1` (`MsiNTProductType <> 1` XML では) であることがわかりましたが、これを起動条件に変換するには、その逆が必要です。 の反対は `<>` なので `=` 、必要な起動条件式は です `MsiNTProductType = 1` 。

> はい、MSI には `NOT` 演算子がありますので、そのようにしたい場合は、起動条件を として表現できます `NOT (MsiNTProductType <> 1)` 。これはまったく混乱を招くものではありません。 を使用し、とXML の奇妙さ `=` を避けてください。 `< >`

擬似コードに戻ると、次のようになります。

```
IF (MsiNTProductType = 1) = FALSE THEN
  SHOW launch_condition_message
ENDIF
```

それでは、WiX コードをいくつか書いてみましょう。

‹ スプリント 4: 条件と式　上　スプリント 4: 起動条件の作成 ›

スプリント4: 起動条件の作成

さて、どの条件をチェックする必要があるかがわかったので、WiX コードを記述してみましょう。起動条件は、`Launch` 要素によって表されます。これは通常、 要素の子要素として存在します `Package` 。Visual Studio 内で入力を開始すると、Intellisense が一致する要素を一覧表示します。その中に、 で始まり、 で許可されている `<Launch` 要素が 1 つあります。Intellisense は使用可能な属性も表示するため、次のスニペットを簡単に取得できます。
`L Package`

```
<Launch Condition="" Message="" />
```

> Intellisense を使用して利用可能な属性を見つけるときは注意してください。Intellisense は、親要素の正当な子要素ではない要素をリストしないことを認識できるほど賢いですが、属性についてはそれほど賢くありません。非常に賢い WiX 開発者は、可能な限り最高の WiX 言語を作成するために多くの時間を費やしています。つまり、利用可能な属性の多くに対するデフォルト値は、まさにあなたが望むとおりのものです。それらのデフォルト値、さらにはすべてのデフォルト値を指定して、あなたが何を意味しているかをさらに細かく指定することができます。しかし、それは *WiX* コードが非常に冗長になり、理解しにくくなるという代償を伴います。そうしないでください。非常に賢い WiX 開発者を信頼してください。

必要な条件は であることが分かっており `MsiNTProductType = 1` 、次のようになります。

```
<Launch Condition="MsiNTProductType = 1" Message="Launch Condition Failed Message Goes Here" />
```

起動条件が false の場合、MSI は次のような簡単なメッセージ ボックスをポップアップ表示します。

では、メッセージで何を伝えたいのでしょうか。1 つの方法は、シソーラスを取り出して完璧なメッセージを書こうとすることです。もう 1 つの方法は、マーケティング担当者が、伝える情報量を減らして単語数を増やすために書き直したいと思うだろうと認識し、わざわざ凝ったことを書かないようにすることです。2 番目の効率的な (「怠惰な」) 方法を使用しましょう。

```
<Launch Condition="MsiNTProductType = 1" Message="TODO Wrong OS type! TODO" />
```

> `TODO` コメントは簡単に検索でき、誤ってユーザーに送信されることはありませんよね?

‹ スプリント 4: ブロックするかしないか　上　スプリント4: テスト ›

スプリント4: テスト

起動条件をテストしており、起動条件は UAC プロンプトの前に早期に評価されるため、Windows Sandbox を起動する (最小限の) 手間をかけずに、簡単かつ安全にテストできます。テストするシナリオは 2 つあります。

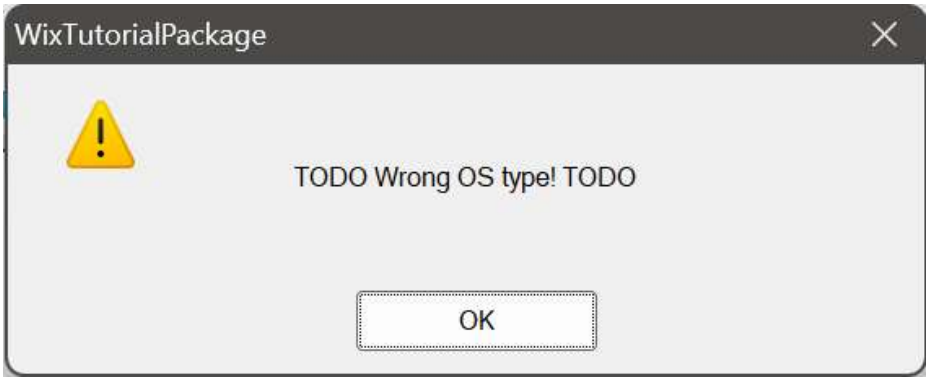
1. Windows Server マシンで実行しているときにパッケージをブロックすることに成功しました。
2. 誤って、Server 以外の Windows マシンへのパッケージのインストールを阻止したわけではありません。

昔、これが「簡単で安全」だと言われていたことを覚えていますか? 実際に、Windows Server マシンをお持ちで、それを操作できるのであれば、それは可能です。誰もがそうであるとは限りません。そのため、シナリオ 1 を正常にテストするには、Windows Server を搭載した仮想マシンを準備する必要があります。これは、特に製品開発サイクルのこの初期段階では、大変な作業のように思えます。おそらく、他の誰かが Windows Server VM をセットアップして、それを「借りる」必要があるでしょう。ごまかす方法はあるのでしょうか? ああ、ごまかす方法はいつでもありますよ、友よ。

面倒なので、Windows インストーラーのドキュメントが正しく、ドキュメントに記載されている の値を信頼できると仮定しましょう `MsiNTProductType` 。 の値は Windows インストーラーに組み込まれているため、変更できません `MsiNTProductType` 。ただし、起動条件の条件式を変更して、Windows Server で実行していることをシミュレートすることはできます。起動条件で Windows Server でのパッケージの起動を防ぐのではなく、起動条件を逆にして、Server 以外 (Home および Professional) での起動を防ぐことができます。

```
<Launch Condition="MsiNTProductType &lt;&gt; 1" Message="TODO Wrong OS type! TODO" />
```

そして、その偽の条件を設定すると、次のようになります。



これで、一時的な成功を宣言します。

偽の起動条件の変更を元に戻します。

```
<Launch Condition="MsiNTProductType = 1" Message="TODO Wrong OS type! TODO" />
```

パッケージをビルドして起動すると、起動条件の失敗メッセージではなく、UAC プロンプトが表示されます。これをキャンセルするだけで、簡単で安全なテストが完了します。

‹ スプリント 4: 起動条件の作成　上　スプリント4の振り返り ›

スプリント4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 4 では、どうでしたか?

何がうまくいきましたか?

私たちは WiX コードを書くことに着手し、実際に書きました。素晴らしいです! Windows インストーラーの内部と、それを使用して基本的な機能を実現する方法について少し学びました。つまり、ユーザーがサポートされている構成で終わるようにし、サポートされていない構成で終わるのを防ぐことです。

もっとうまくいく方法は何でしょうか?

セットアップ作業に関する課題の 1 つが判明しました。それは、たとえ「すべて Windows だけ」であっても、オペレーティング システムの複数のバージョンや種類を扱う必要があることです。通常、小規模な企業でも、退屈なほどすべてを同じにしておくために、1 つのバージョンの Windows に標準化します。しかし、幅広いユーザーの世界では、より多様な Windows のバージョンやエディションをサポートする必要がある可能性があります。セットアップに関しては、通常、少数のオペレーティング システムで開発およびテストできますが、製品がサポートするものに応じて、より多くのオペレーティング システムでテストする準備が必要です。それをどのように行うかを検討する必要があります。

次のスプリントで何を改善しますか?

私たちは、このスプリントで行ったこととその方法について、実はかなり満足しています。「多くのウィンドウ」問題を解決する必要があることはわかっていますが、それに取り組む時間が少しあることを願っています。

‹ スプリント 4: テスト 上 スプリント 5: WiX 拡張機能の使用 ›