ハンズオン ラボ

.NET Framework 4 における

Windows Workflow Foundation の概要

ラボ バージョン: 1.0.0

最終更新日: 2010 年 9 月 21 日

目次

概要

Windows Workflow Foundation 4 (WF) にようこそ。WF は、ワークフロー対応のアプリケーションをすばやく構築するためにマイクロソフトが提供するプログラミング モデル、エンジン、およびツールです。.NET Framework 4 で今回リリースする WF では、以前のバージョンの開発パラダイムがいくつか変更されています。ワークフローの作成、実行、および管理がこれまで以上に容易になり、多数の新機能が実装されています。

このラボでは、ワークフローの作成、ホスト、および実行の基本について学習します。また、新しいワークフロー デザイナー、式、変数、引数など、.NET Framework 4 と Visual Studio 2010 の新しいワークフロー作成用構成について概説することも目的としています。さらに、基本的な組み込みアクティビティの使い方についても説明します。

前半の演習では、あいさつメッセージをコンソール ウィンドウに表示するワークフロー アプリケーションを作成します。アプリケーションのワークフローの作成にデザイナーと XAML を使用する方法と、C# コードや Visual Basic コードを使用する方法を併せて説明します。

次に、独自の条件に応じて異なるあいさつメッセージを表示するため、If アクティビティを使用する条件ロジックをワークフローに追加します。最後に、例外処理アクティビティを使用して、実行中のワークフローでエラーをハンドルする方法について学習します。

また、このラボでは "テストを先に作成する" 方式を採用しています。基本的には、追加する新機能のテストを作成してから、このテストを成功させるために必要なコードを実装します。

目的

このハンズオン ラボでは、次のことを行う方法について学習します。

- デザイナーと XAML を使用するか、C# コードや Visual Basic コードを使用して、シーケンシャル ワークフローを作成する
- **WorkflowApplication** クラスと **WorkflowInvoker** クラスを使用してシーケンシャル ワークフローを実行およびテストする
- ワークフローに "入力引数" を渡して "出力引数" から受け取る

- "式" と "変数" を使用する
- **WriteLine**、**If**、**TryCatch**、**Catch<T>**、および **Throw** の各アクティビティを使用する
- アクティビティを .xaml ファイルから読み込んで実行する
- アクティビティ デザイナーを作成する
- アプリケーションで **WorkflowDesigner** をホストする

システム要件

このラボには、次のものが必要です。

- Microsoft Visual Studio 2010 Beta 2
- Microsoft .NET Framework 4 Beta 2

セットアップ

> **メモ:** 日本語環境でこのラボを実行する場合は下記の Read Me を参考にして、セットアップを実行してください。
>
> http://msdn.microsoft.com/ja-jp/netframework/ff384798.aspx

依存関係チェッカー (Dependency Checker) を使用すると、このラボの要件がすべて確認されます。すべての要件が正しく構成されていることを確認するには、次の手順を実行します。

> **メモ**: セットアップ手順を実行するには、管理者特権を使ってコマンド ウィンドウからスクリプトを実行する必要があります。

1. トレーニング キットの依存関係チェッカーを以前に実行していなければ、実行します。これを行うには、**%TrainingKitInstallationFolder%\Labs\IntroToWF\Source\Setup** フォルダーの **CheckDependencies.cmd** スクリプトを実行します。前提条件を満たしていなけれ

ば、必要な項目をすべてインストールし (必要に応じて再スキャンし)、ウィザードを完了します。

> **メモ**: 便宜上、このラボで管理するコードの大半は、Visual Studio のコード スニペットとして使用できるようにしています。**CheckDependencies.cmd** ファイルによって Visual Studio インストーラー ファイルが起動し、コード スニペットがインストールされます。ソリューションを作成する際にスニペットが見つからない場合は、Visual Studio 2010 コード スニペット リポジトリにコード スニペットがインストールされていることを確認してください。

演習

このハンズオン ラボは以下の演習から構成されています。

1. あいさつワークフロー
2. ワークフローのリファクタリング
3. CodeActivity
4. XAML による動的ワークフロー
5. ワークフローのテスト
6. WorkflowApplication
7. If/Else ロジックの追加
8. エラー処理
9. アクティビティ デザイナー
10. ホストされたデザイナー

演習の教材

このハンズオン ラボには次の教材が含まれています。

- **Visual Studio ソリューション**: 演習の出発点として使用するため、C# と Visual Basic の Visual Studio ソリューションを演習ごとに用意しています。

行き詰まったら

このハンズオン ラボに付属するソース コードには end フォルダーがあり、各演習を修了すると完成する最終的な Visual Studio ソリューションが含まれています。演習中に支援が必要になった場合は、このソリューションをガイドとして利用できます。

ワークフロー デザイナーを使用してファイルを開く前に、必ずソリューションをビルドしてください。

演習 1: あいさつワークフロー

ワークフローとは、ビジネス プロセスを実施する方法です。ビジネス プロセスの各ステップは、"アクティビティ" によって実装します。

この演習では、Windows Workflow Foundation 4 を使用して、簡単な "あいさつ" を行うビジネス プロセスを作成してテストします。

タスク 1 – 簡単なあいさつワークフロー アプリケーションを作成する

このタスクでは、次のコードに相当する非常に簡単なワークフローを作成します。

C#

```
private static void SayHello()
{
  Console.WriteLine("Hello Workflow 4");
}
```

Visual Basic

```
Private Shared Sub SayHello()
  Console.WriteLine("Hello Workflow 4")
End Sub
```

1. Microsoft Visual Studio 2010 を起動します。[スタート] ボタンをクリックし、[すべてのプログラム]、[Microsoft Visual Studio 2010]、[Microsoft Visual Studio 2010] の順にクリックします。
2. [ファイル]、[新規作成]、[プロジェクト] の順にクリックし、そしてプロジェクトのプロパティを設定していきます。
    a. Visual Studio 2010 の [Visual C#] または [Visual Basic] のいずれかのプロジェクト テンプレート一覧で、[Workflow] (Workflow) プロジェクトを選択します。
    b. [Workflow Console Application] (ワークフロー コンソール アプリケーション) というテンプレートを選択します。
    c. 対象のランタイムに [.NET Framework 4] を選択していることを確認し、「HelloWorkflow」という名前を付けます。

d. **%TrainingKitInstallFolder%\Labs\IntroToWF\Ex1-HelloWorkflow\Begin** フォルダーの **C#** フォルダーもしくは **VB** フォルダーにあるをソリューションの配置場所に指定します。(お好きな言語を選択してください。)

e. ソリューションに「HelloWorkflow」という名前を付け、作成先の場所を設定して、[OK] をクリックします。

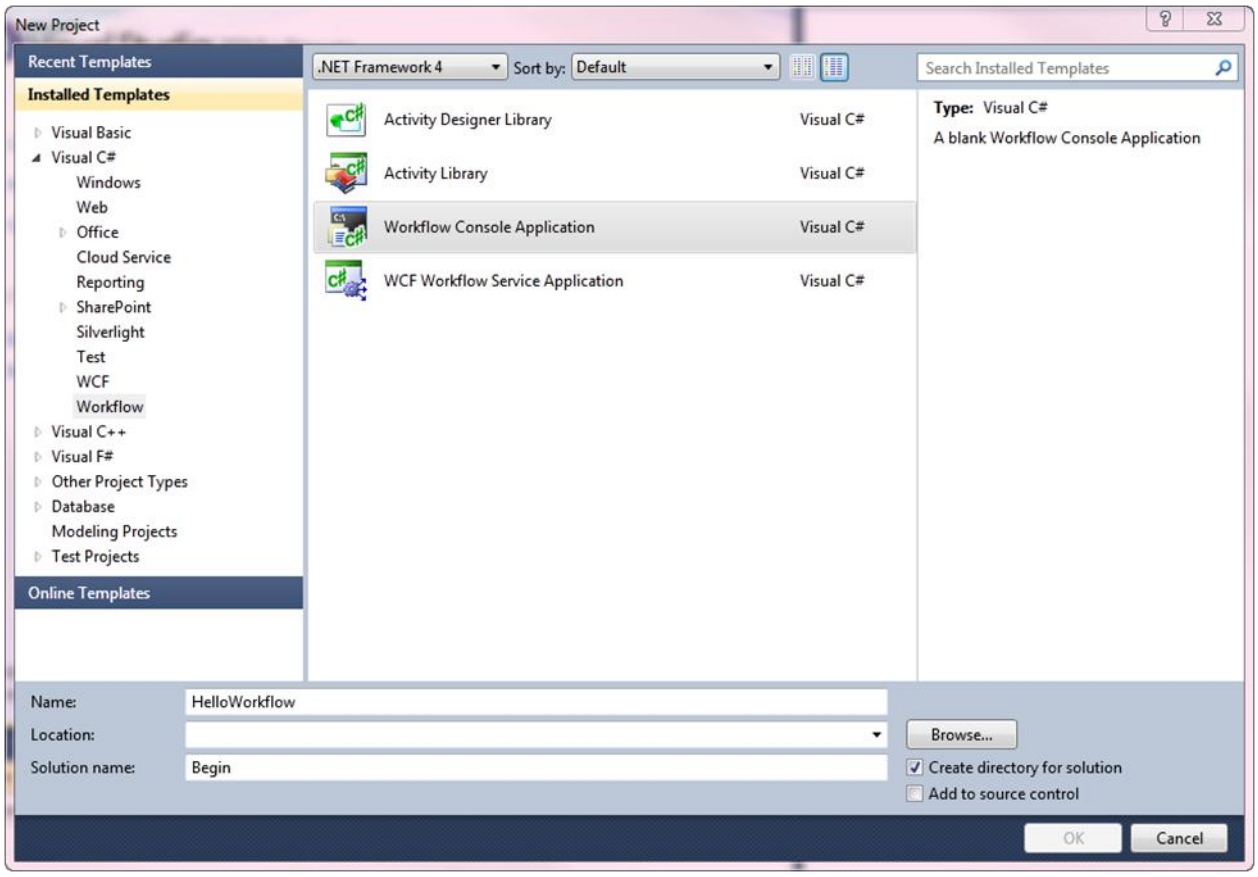

図 1

新しいワークフロー コンソール アプリケーションの作成 (C#)

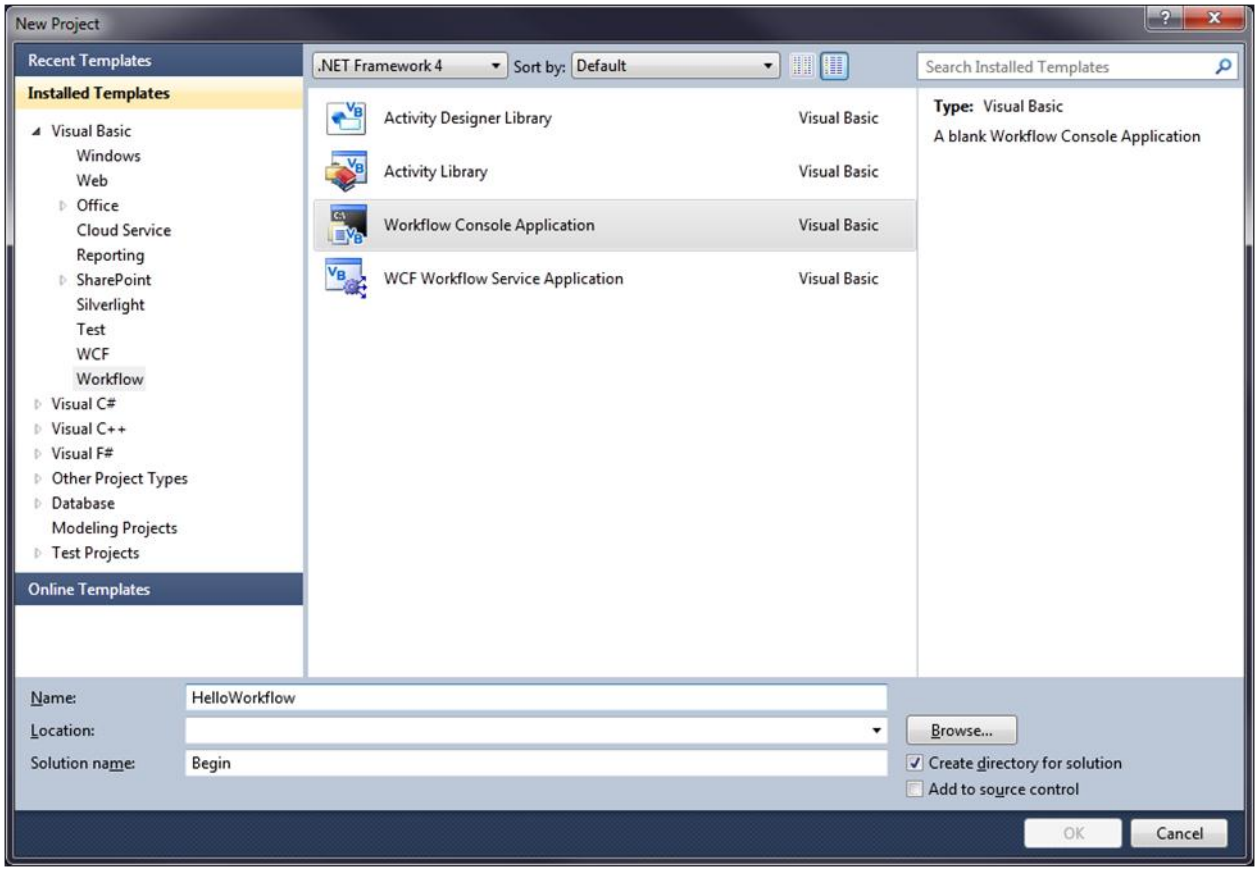

図 2

新しいワークフロー コンソール アプリケーションの作成 (Visual Basic)

3. このビジネス プロセスは 1 ステップのプロセスになるため、**WriteLine** アクティビティを追加するだけで実装できます。ツールボックスから **WriteLine** アクティビティをドラッグし、デザイン画面にドロップします。

図 3

WriteLine アクティビティの追加

> **メモ**: [ツールボックス] ウィンドウが表示されていなければ、[View] (表示) メニューの [Toolbox] (ツールボックス) をクリックします。

4. **WriteLine** の **Text** プロパティに「Hello Workflow 4」を設定します。

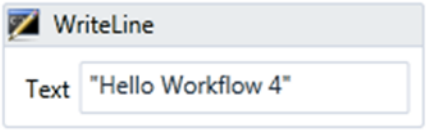

図 4

Text プロパティの設定

> **WriteLine アクティビティ**
>
> **WriteLine** アクティビティは、コンソールにメッセージを表示する単純なアクティビティです。Text プロパティは "式" で、関数の呼び出し結果やオブジェクトのプロパティの評価結果も設定できます。ここでは、式がリテラル文字列なので、文字列を引用符で囲む必要があります。

次の手順

演習 1: 確認

演習 1: 確認

1. **Ctrl** キーを押しながら **F5** キーを押し、ワークフローをビルドしてデバッグなしで実行します。アプリケーションがコンソール ウィンドウで実行され、"Hello Workflow 4" というメッセージが表示されます。

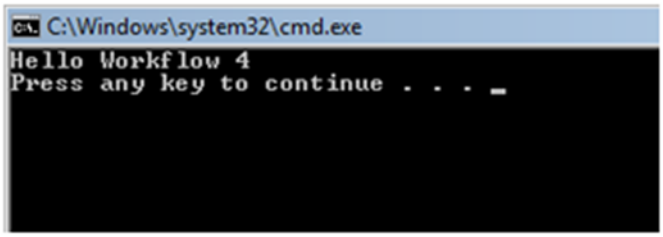

図 5

完成した "あいさつワークフロー" アプリケーション

次の手順

演習 2: ワークフローのリファクタリング

演習 2: ワークフローのリファクタリング

正常に機能しているアプリケーションでも、いくつか改善が必要なこともあります。この演習では、**Workflow1** はあまりわかりやすい名前とは言えないため、**SayHello** に変更します。

タスク 0 – ソリューションを開く

この演習では、演習 1 で作成したソリューションをリファクタリングします。演習 1 を完了していない場合は、次の手順で演習 2 を開始することができます。

1. Microsoft Visual Studio 2010 を管理者として起動します。[スタート] ボタンをクリックし、[すべてのプログラム]、[Microsoft Visual Studio 2010] の順にクリックします。
2. **%TrainingKitInstallFolder%\Labs\IntroToWF\Source\Ex2-RefactoringWorkflow\Begin** フォルダーには、C# と Visual Basic の開始ソリューションがあります。
3. **Ctrl** キー、**Shift** キー、**B** キーを同時に押して、ソリューションをビルドします。

タスク 1 – Workflow1 から SayHello に名前を変更する

1. **Workflow1.xaml** に定義されているワークフローの名前を変更するには、このファイルをワークフロー デザイナーで開いて、デザイナー画面の空白領域をクリックします。その結果、ワークフローのプロパティを変更できるようになります。

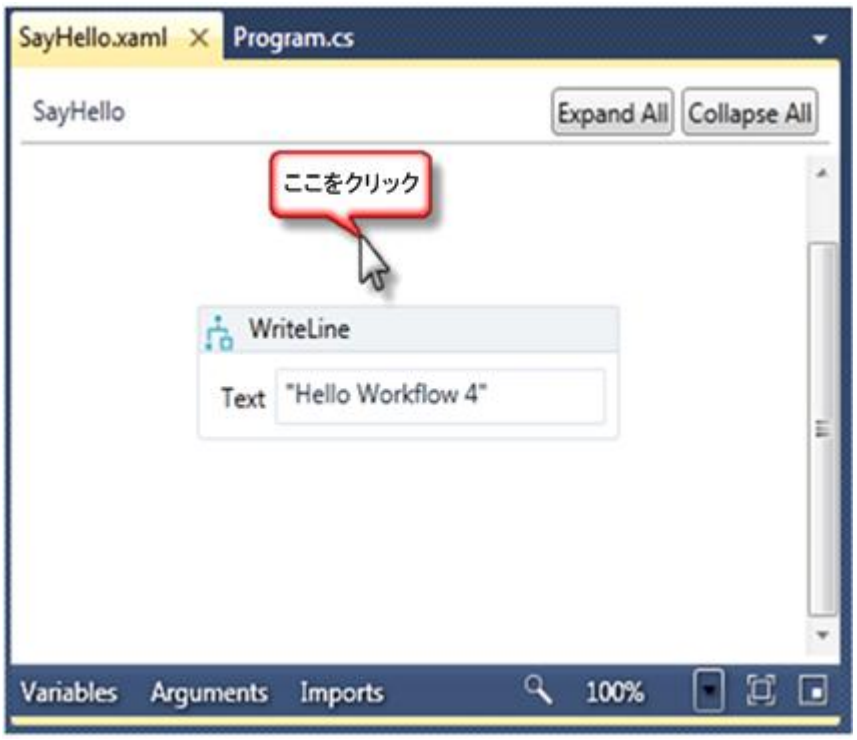

図 6

ワークフローの名前を設定できるように、デザイナー画面の空白領域をクリック

2. [Properties] (プロパティ) ウィンドウで、ワークフローの **Name** プロパティを **HelloWorkflow.SayHello** (C#) または **SayHello** (VB) に設定します。

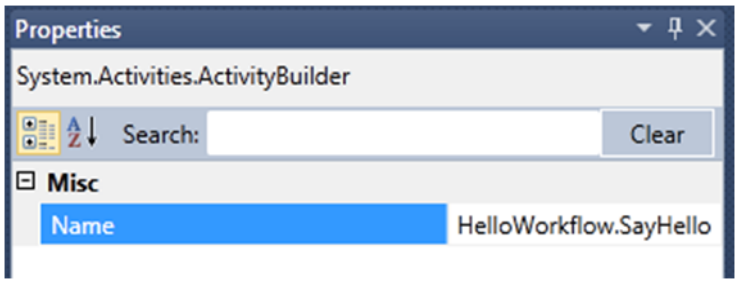

図 7

ワークフローの Name プロパティの設定 (C#)

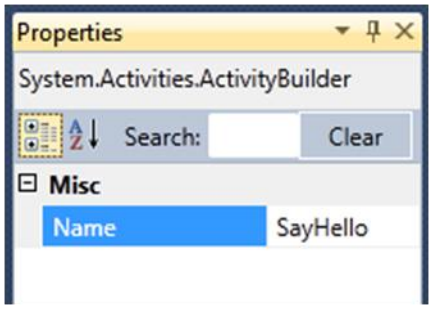

図 8

ワークフローの Name プロパティの設定 (VB)

3. 必ずしも必要ではありませんが、.xaml ファイルの名前を、ファイルに含まれているアクティビティの名前に合わせることをお勧めします。ソリューション エクスプローラーで **Workflow1.xaml** を右クリックし、[Rename] (名前の変更) をクリックして「SayHello.xaml」と入力します。

> **注意**
>
> C# ファイルまたは Visual Basic ファイルの名前を変更すると、クラス名も変更してリファクタリングするかどうかが確認されます。XAML ファイルの名前を変更する場合にはこの確認は行われません。そのため、ワークフロー名も変更するときは、手作業で変更する必要があります。

4. **Ctrl** キー、**Shift** キー、**B** キーを同時に押して、ソリューションをビルドします。ワークフロー名を変更したため、コンパイルに失敗します。

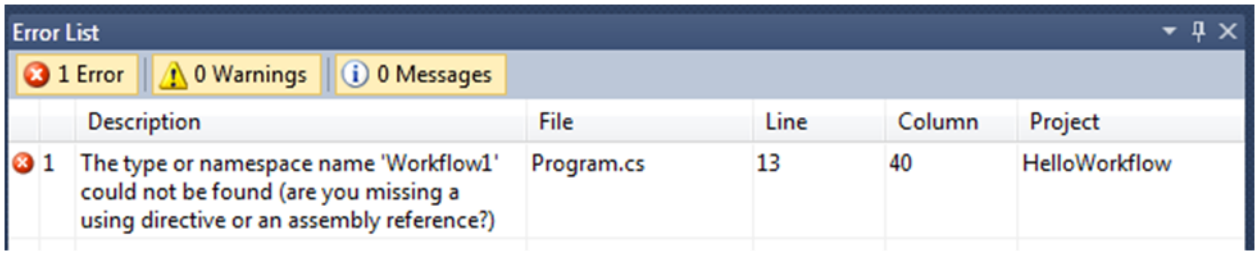

図 9

ワークフローのコンパイルに失敗 (C#)

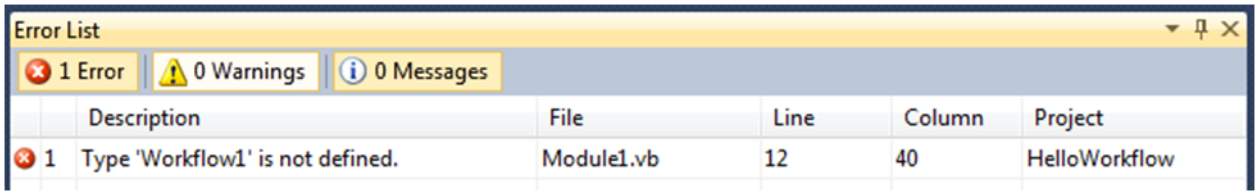

図 10

ワークフローのコンパイルに失敗 (VB)

> **ワークフロー名を変更するとアプリケーションのコンパイルに失敗する理由**

ワークフローは、実際には System.Activities.Activity から継承されるクラスで、XAML で宣言されています。ワークフロー名は、ワークフローを実行するために作成しなければならないクラスの名前です。Name プロパティを設定すると、そのクラス名も変更されます。

タスク 2 – 新しい名前を使用するよう Main メソッドを更新する

WF4 では、"ワークフロー ランタイム" を使用してワークフローが実行されます。ランタイムを呼び出す最も簡単な方法は、**WorkflowInvoker** クラスを使用することです。ワークフロー コンソール アプリケーション テンプレートで使用されているのはこのクラスです。

1. ビルドの問題を解決するには、**Program.cs (C#)** または **Module1.vb (Visual Basic)** を開き、新しい **SayHello** というクラス名を使用するように変更します。これを行うには、次の手順を実行します。
   a. **WorkflowInvoker.Invoke** メソッド呼び出しを探します。
   b. 次のコード (太字箇所) のように、**Workflow1** の代わりに **SayHello** を使用するよう変更します。

C#

```
static void Main(string[] args)
{
  WorkflowInvoker.Invoke(new SayHello());
}
```

Visual Basic

```
Shared Sub Main()
  WorkflowInvoker.Invoke(New SayHello())
End Sub
```

Visual Studio で、型 SayHello が定義されていないと警告される理由

ワークフローを新しい SayHello という名前でコンパイルする前に Main() メソッドを変更すると、型 SayHello が定義されていないと警告されることがあります。これは、XAML で宣言された型が、ソリューションをビルドするまで Visual Studio で認識されないためです。

次の手順

演習 2: 確認

演習 2: 確認

1. **Ctrl** キー、**Shift** キー、**B** キーを同時に押して、ソリューションをビルドします。今度はエラーが発生せずにビルドされます。
2. **Ctrl** キーを押しながら **F5** キーを押してソリューションを実行します。前と同じように、"Hello Workflow 4" というメッセージが表示されます。

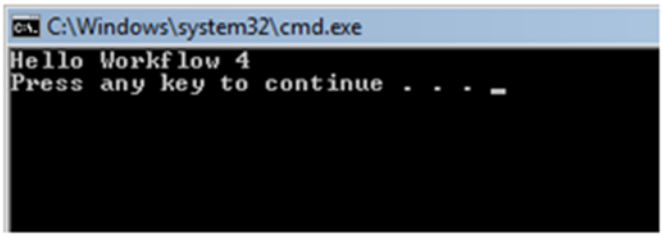

図 11

"Hello Workflow 4" メッセージ

次の手順

演習 3: CodeActivity

演習 3: CodeActivity

メモ: 演習 2 を完了していない場合は、このラボで用意したソリューションを使用できます。**%TrainingKitInstallFolder%\Labs\IntroToWF\Source\Ex3-CodeActivity\Begin** フォルダーには、C# と Visual Basic の開始ソリューションがあります。

既に説明したように、WF4 は、**.xaml** ファイルを編集する "デザイナー" と、アクティビティを呼び出す "ランタイム" から構成されます。ワークフローの作成時には新しい種類のアクティビティを作成していることになります。アクティビティは **System.Activities.Activity** またはそのサブクラスのいずれかから継承するクラスなので、C#、VB、または XAML を使用してワークフローを宣言できます。この演習では、C# や VB でアクティビティを作成して、"あいさつ" ビジネス プロセスを実装します。

アクティビティはビジネス プロセスを実装します。他のアクティビティを呼び出すことでプロセスを実装するアクティビティもあります。たとえば、**SayHello** アクティビティでは、実際は、テキストをコンソールに表示する代わりに **WriteLine** アクティビティを使用してこの処理を行いました。C# でも VB でも、次の例のように **System.Activities.Activity** から継承し、**Implementation** プロパティをオーバーライドすることで、同じ **SayHello** アクティビティを実装できます。

C#

```
public sealed class SayHelloActivity : Activity
{
   WriteLine writeLine = new WriteLine() { Text = "Hello Workflow 4" };

   public SayHelloActivity()
   {
      Implementation = () => { return writeLine; };
   }
}
```

Visual Basic

```
Public NotInheritable Class SayHelloActivity
   Inherits Activity

   Private writeLine As New WriteLine With {.Text = "Hello Workflow 4"}
   Public Sub New()
      Implementation = Function()
                          Return writeLine
                       End Function
   End Sub
End Class
```

他のアクティビティからワークフローを作成する場合、作成元のアクティビティが、演習 1 で **WriteLine** アクティビティを使用して実行したような処理を実行している場合は、この方法が便利です。しかし、内部にビジネス ロジックを実装するアクティビティや、アクティビティではない別のクラスを呼び出して目的の処理を実行するロジックを実装するアクティ

ビティを作成する場合もあります。これを行うには、**System.Activities.CodeActivity** という別の基本クラスから継承し、**Execute** メソッドをオーバーライドします。

タスク 0 – ソリューションを開く

この演習では、演習 2 で作成したソリューションを使用することができます。演習 2 を完了していない場合は、次の手順で演習 3 を開始することができます。

1. Microsoft Visual Studio 2010 を管理者として起動します。[スタート] ボタンをクリックし、[すべてのプログラム]、[Microsoft Visual Studio 2010] の順にクリックします。
2. **%TrainingKitInstallFolder%\Labs\IntroToWF\Source\ Ex3-CodeActivity \Begin** フォルダーには、C# と Visual Basic の開始ソリューションがあります。
3. **Ctrl** キー、**Shift** キー、**B** キーを同時に押して、ソリューションをビルドします。

タスク 1 – SayHelloInCode アクティビティを作成する

このタスクでは、コードでアクティビティを作成します。このアクティビティでは、**Console.WriteLine** を使用してテキストをコンソールに表示します。

1. **HelloWorkflow** プロジェクトを右クリックし、[Add] (追加) をポイントして、[NewItem] (新しい項目) をクリックします。[Workflow] テンプレートから [CodeActivity] (コード アクティビティ) を選択し、 [Name] (名前) ボックスに「SayHelloInCode」と入力します。
2. **Text** プロパティをテンプレートから削除してください。
3. **CodeActivity** は "抽象" クラスです。つまり、このクラスから継承する場合は、**Execute** メソッドをオーバーライドする必要があります。このメソッドに、アクティビティが行う処理を配置します。クラスの既定の実装を次のコードに置き換えます。

(Code Snippet - Introduction to WF4 Lab - SayHelloInCode Class CSharp)

C#

```
public sealed class SayHelloInCode : CodeActivity
{
   protected override void Execute(CodeActivityContext context)
   {
      Console.WriteLine("Hello Workflow 4 in code");
   }
}
```

(Code Snippet - Introduction to WF4 Lab - SayHelloInCode Class VB)

Visual Basic

```
Public NotInheritable Class SayHelloInCode
  Inherits CodeActivity

  Protected Overrides Sub Execute(ByVal context As CodeActivityContext)
    Console.WriteLine("Hello Workflow 4 in code")
  End Sub
End Class
```

タスク 2 – SayHelloInCode を呼び出すよう Main メソッドを更新する

1. **SayHelloInCode** クラスを使用するように、**Program.cs (C#)** または **Module1.vb (Visual Basic)** を変更します。これを行うには、**WorkflowInvoker.Invoke** メソッドを探し、次のコードに置き換えます。

(Code Snippet - Introduction to WF4 Lab - InvokeSayHelloInCode Class CSharp)

C#

```
static void Main(string[] args)
{
  WorkflowInvoker.Invoke(new SayHelloInCode());
}
```

(Code Snippet - Introduction to WF4 Lab - InvokeSayHelloInCode Class VB)

Visual Basic

```
Shared Sub Main(ByVal args() As String)
  WorkflowInvoker.Invoke(New SayHelloInCode())
End Sub
```

次の手順

演習 3: 確認

演習 3: 確認

1. **Ctrl** キーを押しながら **F5** キーを押し、デバッグなしでワークフローを実行します。アプリケーションがコンソール ウィンドウで実行され、"Hello Workflow 4 in code" メッセージが表示されます。

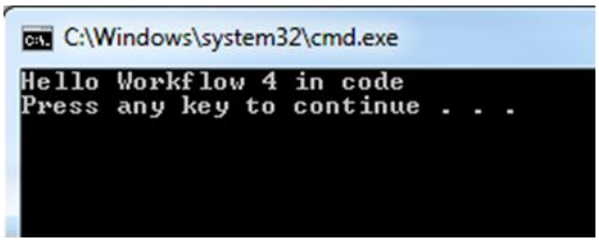

図 12

完成した "あいさつワークフロー" アプリケーション

コード アクティビティを作成する理由

コードでビジネス ロジックを作成することは、特に目新しいことではありません。では、CodeActivity を継承する特殊なクラスをわざわざ作成するのはなぜでしょう。それは、このようにすると、ワークフロー ランタイムを使用するもっと大きなビジネス プロセスにビジネス ロジックを組み込めるようになるためです。このラボの後半で説明するように、この方法を採用すると、拡張性が高く、実行時間の長い業務アプリケーション向けに提供される、スレッド処理やデータ管理のモデルからのメリットも得られます。

次の手順

演習 4: XAML による動的ワークフロー

演習 4: XAML による動的ワークフロー

ここまでは、.xaml ファイル、.cs ファイル、または .vb ファイルでワークフローを作成してきました。これらのファイルは、プロジェクトのアセンブリに含められる型にコンパイルされ、ワークフロー ランタイムによって実行されます。

ソース ファイルの形式は関係ないと思われるでしょうが、.xaml ファイルには、C# または VB でのワークフロー作成に比べて明らかなメリットがあります。

- ワークフロー デザイナーで操作できるのは .xaml ファイルのみです。デザイナーでは、C# または VB で作成したワークフローはサポートされません。
- XAML は、アセンブリにコンパイルしなくても動的に読み込んで実行できます。

動的ワークフローにより、ビジネス ロジックを生成するプログラムや、読み込んで実行するビジネス ロジックを実行時に決定するプログラムといった、興味深い可能性が広がります。

タスク 0 – ソリューションを開く

この演習では、演習 3 で作成したソリューションを使用することができます。演習 3 を完了していない場合は、次の手順で演習 4 を開始することができます。

1. Microsoft Visual Studio 2010 を管理者として起動します。[スタート] ボタンをクリックし、[すべてのプログラム]、[Microsoft Visual Studio 2010] の順にクリックします。
2. **%TrainingKitInstallFolder%\Labs\IntroToWF\Source\Ex4-XAML\Begin** フォルダーには、C# と Visual Basic の開始ソリューションがあります。
3. **Ctrl** キー、**Shift** キー、**B** キーを同時に押して、ソリューションをビルドします。

タスク 1 – SayHello.xaml ファイルのプロパティを変更する

このタスクでは、SayHello.xaml ファイルを読み込んで実行するように HellowWorkflow プログラムを変更します。その後、SayHello.xaml ファイルのテキストを変更し、アプリケーションの次回実行時のメッセージの変化を確認します。

1. ビルドすべきコードではなく、配置すべきコンテンツとして SayHello.xaml を扱うよう、Visual Studio に指定します。これを行うには、次の手順を実行します。
    a. ソリューション エクスプローラーで、**SayHello.xaml** を選択します。

b. [Properties] (プロパティ) ウィンドウで、ビルド アクションを "Content"(コンテンツ) に設定します。

c. SayHello.xaml の [Copy to Output Directory] (出力ディレクトリにコピー) プロパティを "Copy Always"(常にコピーする) に設定します。

d. [Custom Tool] (カスタム ツール) プロパティを空白にします。

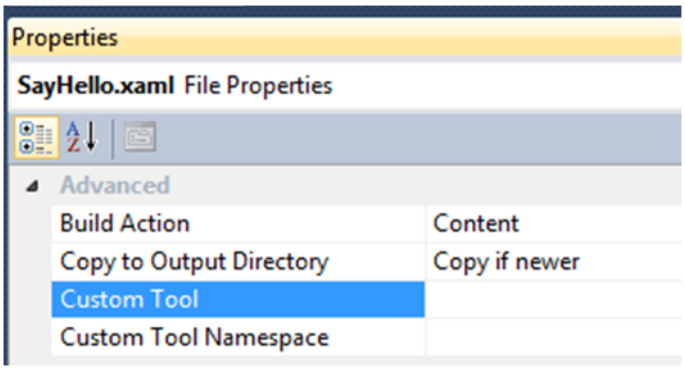

図 13

SayHello.xaml をコンテンツとして扱うためのプロパティの変更

タスク 2 – Main() メソッドを変更して SayHello.xaml ファイルを読み込む

ここまでは、クラスは型にコンパイルされていました。.xaml ファイルからワークフローを呼び出すには、**ActivityXamlServices** を使用して .xaml ファイルをメモリに読み込み、**WorkflowInvoker** から呼び出せる **System.Activities.DynamicActivity** のインスタンスを作成する必要があります。ワークフローの呼び出し時には、.xaml ファイルから参照するすべてのアセンブリにアクセスできなければなりません。

1. ファイルに次の名前空間ディレクティブを追加します。

C#

```
using System.Activities.XamlIntegration;
```

Visual Basic

```
Imports System.Activities.XamlIntegration
```

2. **ActivityXamlServices** を使用するように program.cs を変更し、**Console.ReadKey** の呼び出しも追加します。このようにすると、エクスプローラーでアプリケーション

の処理内容を確認するのが簡単になります。これを行うには、**Main** メソッドの実装を次のコードに置き換えます。

(Code Snippet - Introduction to WF4 Lab - ActivityXamlServices CSharp)

C#

```
static void Main(string[] args)
{
   WorkflowInvoker.Invoke(ActivityXamlServices.Load("SayHello.xaml"));
   Console.ReadKey(false);
}
```

(Code Snippet - Introduction to WF4 Lab - ActivityXamlServices VB)

Visual Basic

```
Shared Sub Main()
   WorkflowInvoker.Invoke(ActivityXamlServices.Load("SayHello.xaml"))
   Console.ReadKey(False)
End Sub
```

次の手順

演習 4: 確認

演習 4: 確認

この確認では、まず HelloWorkflow アプリケーションを実行し、次に SayHello.xaml ファイルを変更して、新しいテキストがコンソール ウィンドウに表示されることを確認します。

1. **Ctrl** キーを押しながら **F5** キーを押し、デバッグなしでワークフローを実行します。アプリケーションがコンソール ウィンドウで実行され、"Hello Workflow 4" というメッセージが表示されます。
2. プロジェクト フォルダーの **Bin\Debug** ディレクトリに移動し、**SayHello.xaml** を探します。

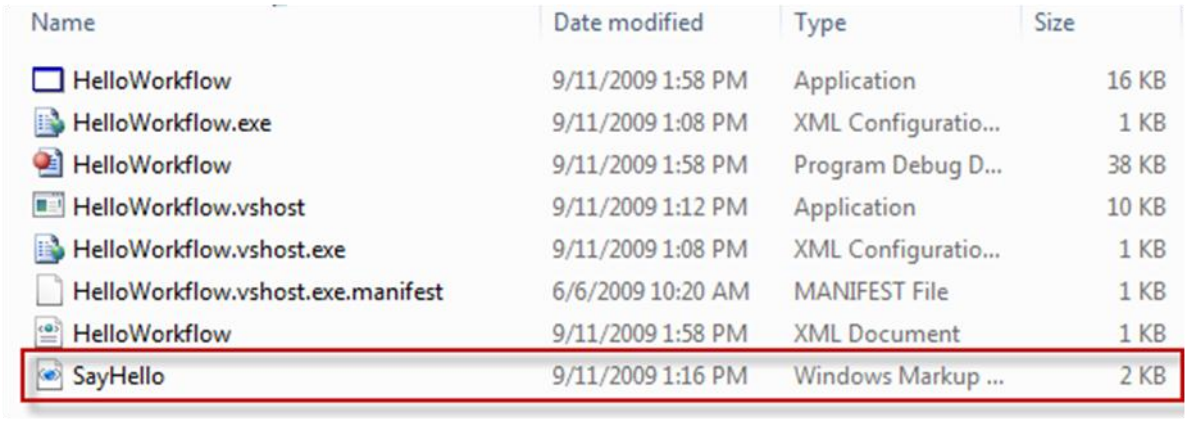

図 14

プロジェクトの Bin\Debug ディレクトリにある SayHello.xaml ファイルの検索

3. **SayHello.xaml** ファイルを右クリックし、[編集] をクリックしてメモ帳で開きます。

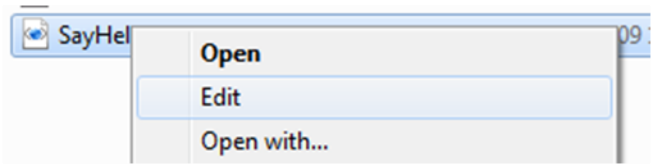

図 15

右クリックして [編集] をクリック

4. メモ帳で、WriteLine アクティビティの Text プロパティを "Hello Workflow 4 XAML" に変更し、ファイルを保存して終了します。

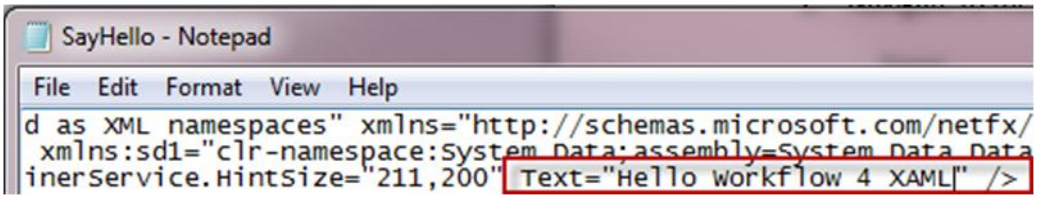

図 16

メモ帳での Text プロパティの変更

5. エクスプローラーで HelloWorkflow.exe を実行すると、今度は "Hello Workflow 4 XAML" と表示されます。任意のキーを押して終了します。

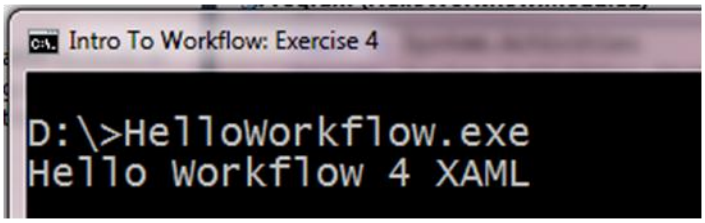

図 17

.xaml ファイルから新しいメッセージを表示する HelloWorkflow.exe メモ帳での Text プロパティの変更

6. Visual Studio にもどり、**SayHello.xaml** を既定の設定に戻してください。既定の設定に戻すには次の手順が必要です。

    a. [Properties] (プロパティ) ウィンドウで、ビルド アクションを "XamlAppDef" に設定します。

    b. SayHello.xaml の [Copy to Output Directory] (出力ディレクトリにコピー) プロパティを "Do not copy"(コピーしない) に設定します。

    c. [Custom Tool] (カスタム ツール) プロパティを ”MSBuild:Compile” にします。

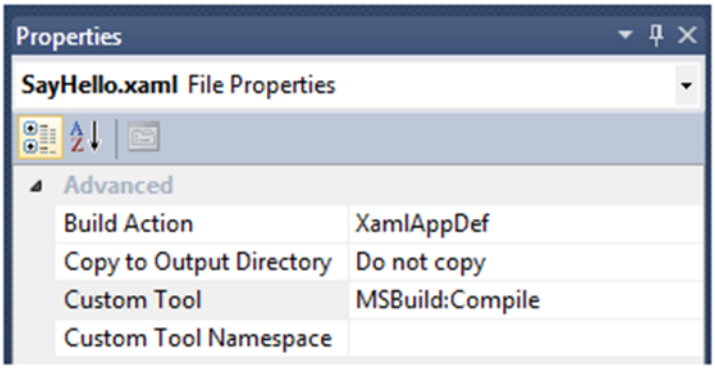

図 18

SayHello.xaml の .xaml ファイルのプロパティを既定値に戻す

注意：

Visual Studio 2010 Beta 2 のバグにより、XamlAppDef が選択肢として表示されない場合があります。表示されない場合は HelloWorkflow プロジェクトを右クリックし、[Add] (追加) をポイントし、[New Item](新しい項目) をクリックします。[Workflow] で [Activitiy] (アクティビティ) を選択し、Activity1.xaml を追加します。この操作に

より、XamlAppDef のビルドアクションが表示されるようになります。

XamlAppDef が選択肢として表示されたら、Activity1.xaml は削除してかまいません。

次の手順

演習 5: ワークフローのテスト

演習 5: ワークフローのテスト

ここまでのところ、アプリケーションではたいして興味深い処理を行っていません。アプリケーションはコンソールで動作しているだけで、入力引数をまったく受け取りません。意味のある処理を実行するアプリケーションでは、入力引数と出力引数を処理する必要があります。また、現在の形式ではアプリケーションを簡単にテストできません。

このタスクでは、引数を使用するよう SayHello アクティビティを変更します。また、WriteLine アクティビティでコンソールにあいさつ文を直接表示するのではなく、あいさつ文を返すことで、他のコンソールを使用しないアプリケーション環境でも使いやすくなるよう準備します。この作業では、"テストを先に作成する" 方式を採用します。まず、失敗するテストを作成してから、テストの成功に必要なコードを追加します。

最終的なアプリケーションは、次のようなコードになります。

C#

```
private static string SayHello(string name)
{
  return "Hello " + name + " from Workflow 4";
}
```

Visual Basic

```
Private Shared Function SayHello(ByVal name As String) As String
  Return "Hello " & name & " from Workflow 4"
End Function
```

タスク 0 – ソリューションを開く

この演習では、演習 4 で作成したソリューションを使用することができます。演習 4 を完了していない場合は、次の手順で演習 5 を開始することができます。

1. Microsoft Visual Studio 2010 を管理者として起動します。[スタート] ボタンをクリックし、[すべてのプログラム]、[Microsoft Visual Studio 2010] の順にクリックします。
2. **%TrainingKitInstallFolder%\Labs\IntroToWF\Source\Ex5-Testing\Begin** フォルダーには、C# と Visual Basic の開始ソリューションがあります。
3. **Ctrl** キー、**Shift** キー、**B** キーを同時に押して、ソリューションをビルドします。

タスク 1 – 単体テスト プロジェクトを作成する

1. まず、ワークフローの単体テストを作成し、ワークフローが正常に動作することを確認します。ソリューション エクスプローラーで HelloWorkflow ソリューションを右クリックして、[Add] (追加) をポイントし、[New Project] (新しいプロジェクト) をクリックします。
    a. [Add New Project] (新しいプロジェクトの追加) ダイアログ ボックスの [Visual C#] または [Visual Basic] プロジェクトのテンプレート一覧で、[Test] (テスト) を選択します。
    b. [Test Project] (テスト プロジェクト) を選択します。
    c. プロジェクトの [Name] (名前) ボックスに「HelloWorkflow.Tests」と入力します。プロジェクトの場所は既定値のままにします。

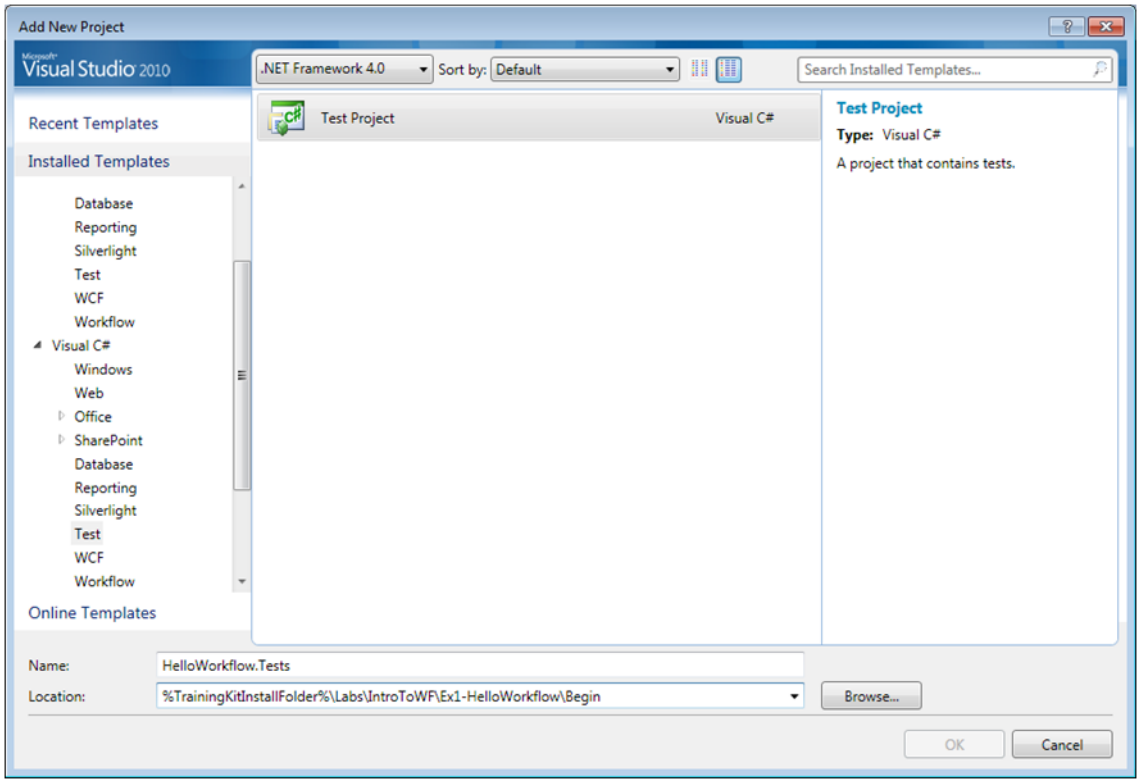

図 19

ソリューションへの新しいテスト プロジェクトの追加 (C#)

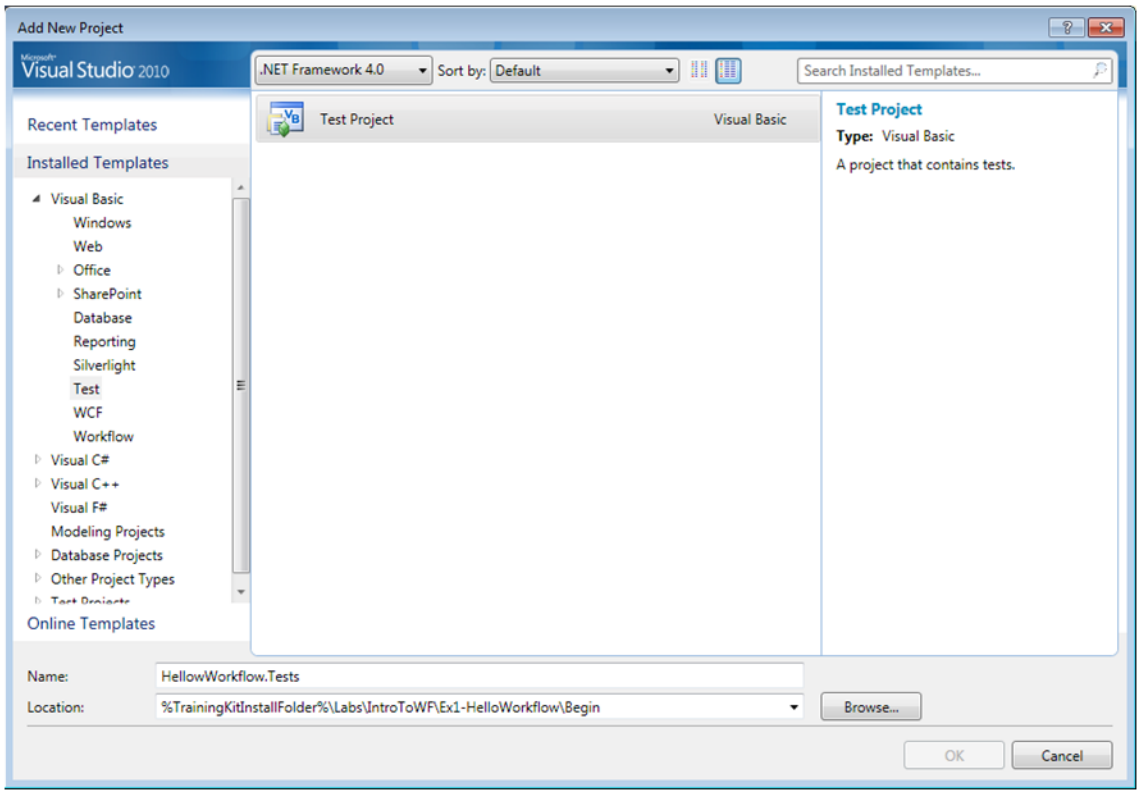

図 20

ソリューションへの新しいテスト プロジェクトの追加 (Visual Basic)

2. **HelloWorkflow.Tests** プロジェクトを右クリックし、[Add Reference] (参照の追加) をクリックします。[Projects] (プロジェクト) タブを使用して、**HelloWorkflow** プロジェクトへのプロジェクト参照を追加します。この手順を繰り返し、今度は [.NET] タブを使用して、**System.Activities** ライブラリへの参照を追加します。
3. **UnitTest1.cs** (C#) または **UnitTest1.vb** (VB) を右クリックし、[Rename] (名前の変更) をクリックして、ファイル名を **SayHelloFixture.cs** (C#) または **SayHelloFixture.vb** (VB) に変更します。**UnitTest1** クラスの名前の変更を確認するメッセージが表示されたら、[はい] をクリックします。

タスク 2 – テストを作成する

このタスクでは、実際にアクティビティの動作を実装する前にテストを作成します。

1. **SayHelloFixture.cs** (C#) ファイルまたは **SayHelloFixture.vb** (VB) ファイルに、次の名前空間ディレクティブを追加します。

C#

```
using System.Activities;
using HelloWorkflow;
```

Visual Basic

```
Imports System.Activities
Imports HelloWorkflow
```

2. ワークフローで正しくあいさつ文が表示されることを確認するテストを作成します。そのためには、**HelloWorkflow.Tests** プロジェクトの **SayHelloFixture.cs** (C#) または **SayHelloFixture.vb** (VB) を開きます。
3. **SayHello** アクティビティではまだ引数を受け取りませんが、受け取ると仮定してこのアクティビティを呼び出すコードを作成します。このようにすると、アクティビティを使用する際にアクティビティへのインターフェイスがどのようになるか考えることができます。Visual Studio から **UserName** プロパティが定義されていないと警告されても、問題ありません。**TestMethod1** を次のコードに置き換えます。

(Code Snippet - Introduction to WF4 Lab - ShouldReturnGreetingImpl CSharp)

C#

```
[TestMethod]
public void ShouldReturnGreetingWithName()
{
   IDictionary<string, object> output;

   output = WorkflowInvoker.Invoke(
      new SayHello()
      {
         UserName = "Test"
      });
   Assert.AreEqual("Hello Test from Workflow 4", output["Greeting"]);
}
```

(Code Snippet - Introduction to WF4 Lab - ShouldReturnGreetingImpl VB)

Visual Basic

```
<TestMethod()> Public Sub ShouldReturnGreetingWithName()
   Dim output = WorkflowInvoker.Invoke(
      New SayHello() With {.UserName = "Test"})
   Assert.AreEqual("Hello Test from Workflow 4", output("Greeting"))
End Sub
```

アクティビティに引数を渡す方法

オブジェクトの初期化を使用すると、アクティビティを作成して引数 (パブリックプロパティ) を初期化できます。また、アクティビティの入力引数名にマップされる入力パラメーターの **Dictionary<string, object>** (C#) または **Dictionary(Of String, Object)** (VB) を渡すこともできます。

出力からデータを取得する方法

出力変数は、**IDictionary<string, object>** (C#) または **IDictionary(Of String, Object)** (VB) オブジェクトで、変数名をキーに使用する出力変数のマップを含みます。

タスク 3 – アプリケーションをコンパイル可能にする

(入力引数と出力引数に関して) このアクティビティのインターフェイスは正常に見えますが、アプリケーションはコンパイルされません。最初の目標は、アプリケーションをコンパイル可能な状態にすることです。

1. **Ctrl** キー、**Shift** キー、**B** キーを同時に押してアプリケーションをビルドすると、コンパイル エラーが発生して失敗します。

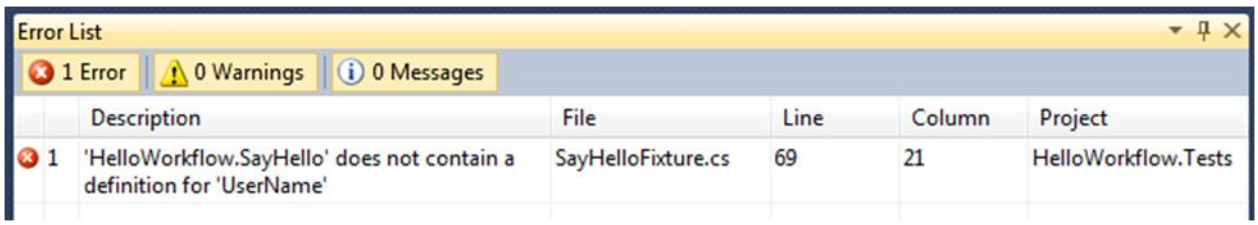

図 21

UserName が未定義 (C#)

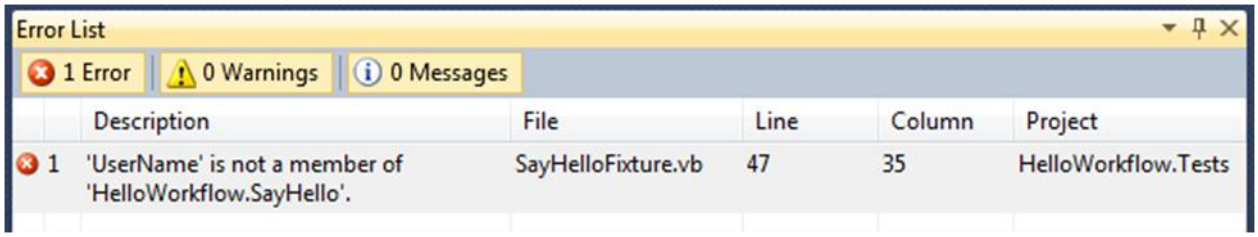

図 22

UserName が未定義 (VB)

2. デザイナーで **SayHello.xaml** を開きます。

3. [Arguments] (引数) をクリックして、引数のペインを開きます。

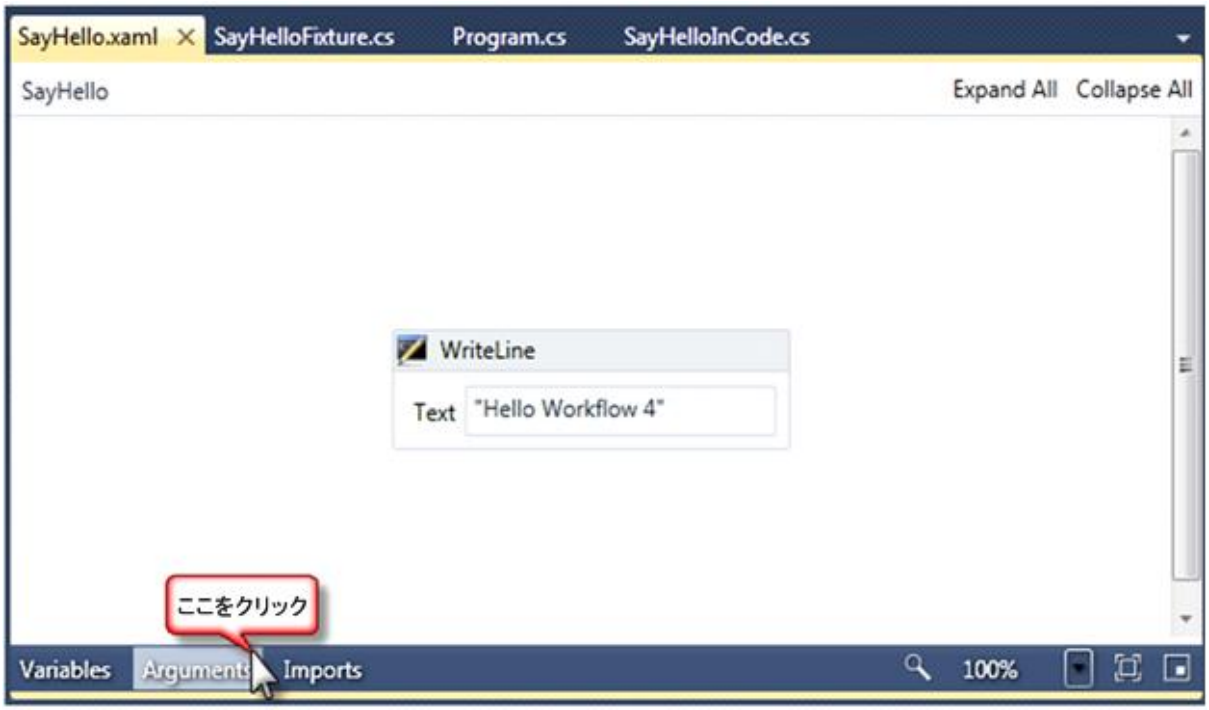

図 23

[Arguments] (引数) をクリックして引数のペインを開く

4. 次のように、**UserName** 引数と **Greeting** 引数を追加します。

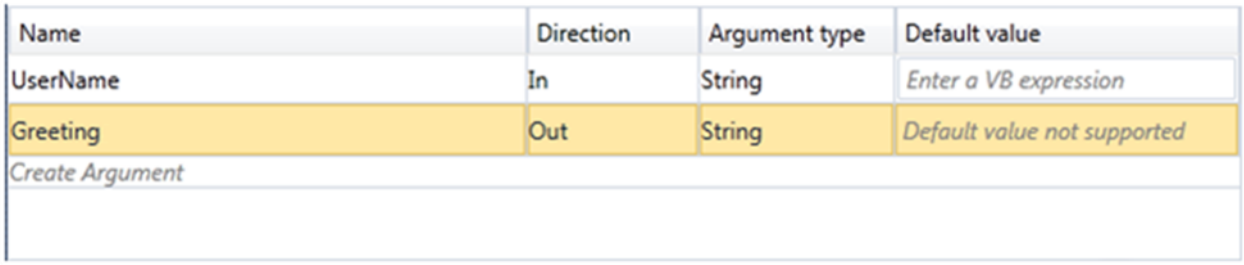

図 24

アクティビティの引数の宣言

> **引数**
>
> **Windows Workflow Foundation** (WF) では、引数はアクティビティに出入りするデータの流れを表します。アクティビティには一連の引数があり、これらの引数によってアクティビティのシグネチャが構成されます。各引数には、入力、出力、または入出力のいずれかの方向を指定します。

5. **Ctrl** キー、**Shift** キー、**B** キーを同時に押してソリューションをビルドすると、今度はエラーが発生せずにビルドされます。

タスク 4 – テストが失敗することを確認する

テストに潜むバグが原因で常にテストが成功することもあるため、テストが失敗することを確認するのは重要です。このタスクでは、テストが実際に失敗することを確認します。

1. **Ctrl** キーを押しながら **R** キーを押し、次に **T** キーを押して、現在のコンテキストで単体テストを実行します。テストは実行されますが、アクティビティから Greeting 出力引数に何も返されないため、失敗します。

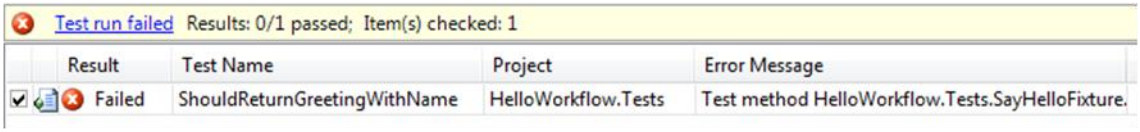

図 25

ShouldReturnGreetingwithName テスト

タスク 5 – テストを成功させる

テストが機能することがわかったので、できる限り簡単な方法でテストを成功させる必要があります。

1. **WriteLine** アクティビティを使用してテキストをコンソールに表示するとテストが成功しないため、**WriteLine** アクティビティを右クリックして [Delete](削除) ( ) をクリックし、このアクティビティを削除します。

2. **Greeting** 出力引数の値を設定する必要があります。値を設定するには、ツールボックスから **Assign** アクティビティをドラッグし、デザイナー画面にドロップします。

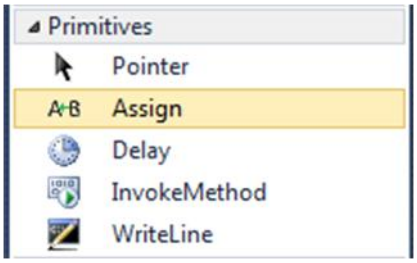

図 26

デザイナー画面への Assign アクティビティのドラッグ

3. **To** プロパティを **Greeting** に設定します。

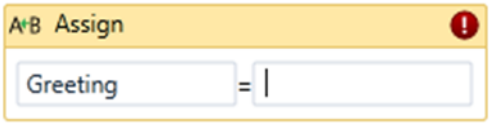

図 27

左のボックスへの「Greeting」の入力

4. デザイナー画面であいさつ文の式を入力することもできますが、ボックスよりも式が長いので、[Properties] (プロパティ) ウィンドウを使用します。Value プロパティの右にあるボタンをクリックし、式エディターを開きます。

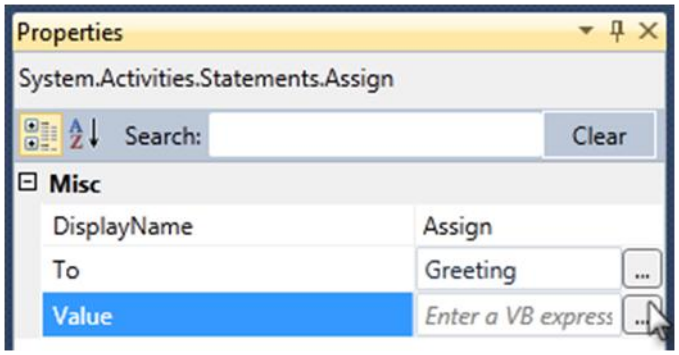

図 28

Value プロパティの右にあるボタンのクリック

5. 式エディターでは、長い式を入力できます。**"Hello " & UserName & " from Workflow 4"** という式を設定します。

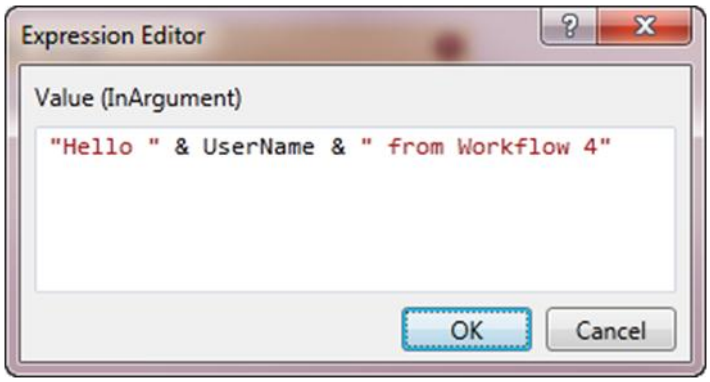

図 29

Greeting 引数を設定する Assign アクティビティ

6. **Ctrl** キー、**Shift** キー、**B** キーを同時に押してソリューションをビルドすると、エラーが発生せずにコンパイルされます。

次の手順

演習 5: 確認

演習 5: 確認

これで、テストを実行して、アクティビティのテストが成功するかどうか確認する準備ができました。

1. テスト ビュー ウィンドウを開きます。そのためには、[Test] (テスト) メニューの [Windows] (ウィンドウ) をポイントし、[Test View] (テスト ビュー) ( ) をクリックします。

2. テスト ビューで、**ShouldReturnGreetingWithName** テストを選択し、[Run Selection] (選択範囲の実行) ( ) をクリックします。

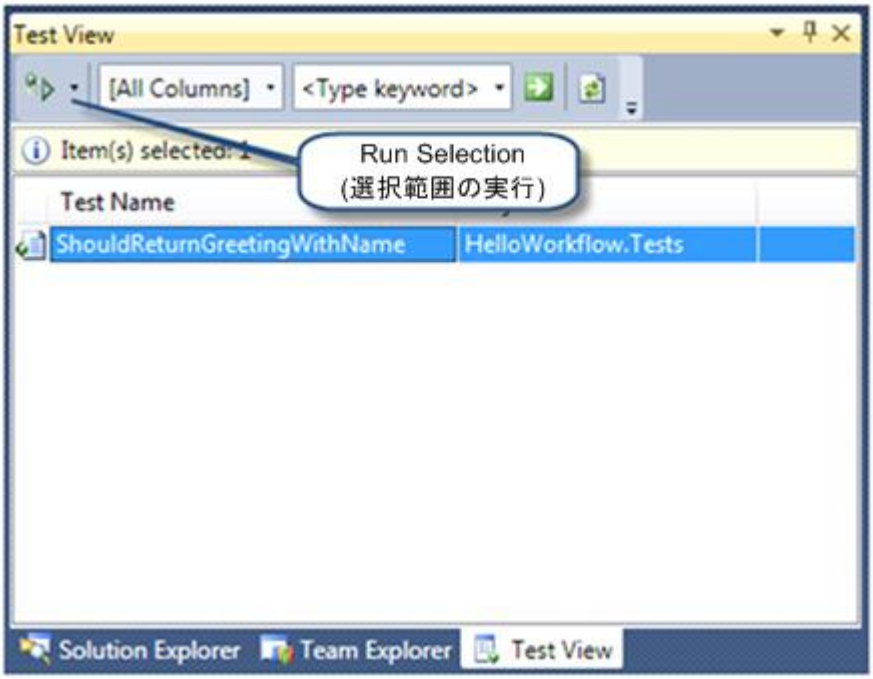

図 30

ShouldReturnGreetingWithName テストの選択と実行

> **注意**
>
> Visual Studio 2010 Beta 2 のバグにより XAML ファイルに対する変更はテスト実行時に自動的に保存されません。テストを実行する前に手動でソリューションをびるどする必要があります。

3. テスト実行が成功することを確認します。

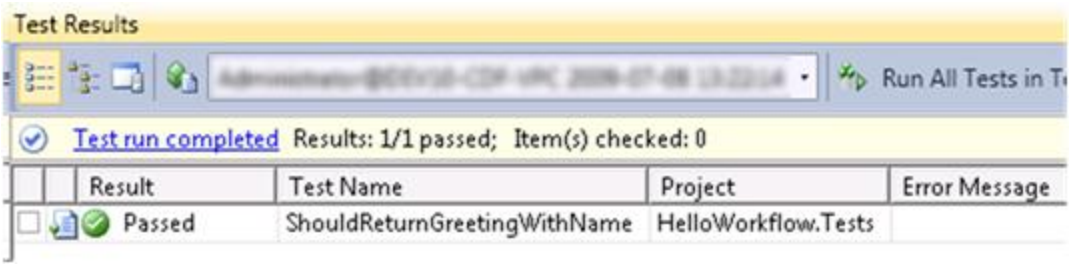

図 31

テストの成功

次の手順

演習 6: WorkflowApplication

演習 6: WorkflowApplication

ラボのここまでの段階では、**WorkflowInvoker** クラスを使用する最も簡単な方法でアクティビティを作成して呼び出すことに重点を置いてきました。**WorkflowInvoker.Invoke** メソッドが簡単なのは、このメソッドでは同期が取られ、呼び出し側と同じスレッドでワークフローが呼び出されるためです。

ワークフローを呼び出す別の方法として、**WorkflowApplication** クラスを使用します。このクラスを使用すると、ワークフローを別のスレッドで実行でき、ワークフローの完了時、アイドル状態への移行時、終了時、またはハンドルされない例外の発生し時に呼び出されるデリゲートを提供できます。このため、ワークフローを使用しない場合よりも簡単に、マルチスレッドのサーバー プログラムやクライアント プログラムを作成できます。

この演習では、ホスト アプリケーションを変更して、SayHello アクティビティを WorkflowApplication で実行し、スレッドの動作を確認します。そのため、次の 2 つの要件をワークフローに追加します。

1. 個人用にカスタマイズしたあいさつ文を返す。
2. ワークフローを呼び出したマネージ スレッド ID を含む、0 でない Int32 値を返す。

"テストを先に作成する" 方式を使用して、ワークフローのスレッド ID に関する新しい要件を確認するテストを作成します。

タスク 0 – ソリューションを開く

この演習では、演習 5 で作成したソリューションを使用することができます。演習 5 を完了していない場合は、次の手順で演習 6 を開始することができます。

1. Microsoft Visual Studio 2010 を管理者として起動します。[スタート] ボタンをクリックし、[すべてのプログラム]、[Microsoft Visual Studio 2010] の順にクリックします。
2. **%TrainingKitInstallFolder%\Labs\IntroToWF\Source\Ex6-WorkflowApplication\Begin** フォルダーには、C# と Visual Basic の開始ソリューションがあります。
3. **Ctrl** キー、**Shift** キー、**B** キーを同時に押して、ソリューションをビルドします。

タスク 1 – ワークフローのスレッド ID が出力引数として返されることを確認するテストを作成する

このタスクでは、ワークフローから WorkflowThread という出力引数に 0 以外の整数値が返されることを確認するテストを作成します。

1. SayHelloFixture.cs を開き、次の名前空間ディレクティブを追加します。

C#

```
using System.Threading;
using System.Diagnostics;
```

Visual Basic

```
Imports System.Threading
Imports System.Diagnostics
```

2. 次のように **ShouldReturnWorkflowThread** テストを追加します。

(コード スニペット: Introduction to WF Lab - ShouldReturnWorkflowThread TestMethod CSharp)

C#

```
/// <summary>
/// ワークフローから出力引数が返されるかどうかの確認
/// 名前: WorkflowThread
/// 型: Int32
```

```
/// 値: 0 以外
/// </summary>
[TestMethod]
public void ShouldReturnWorkflowThread()
{
 var output = WorkflowInvoker.Invoke(
  new SayHello()
  {
   UserName = "Test"
  });

 Assert.IsTrue(output.ContainsKey("WorkflowThread"),
  "SayHello must contain an OutArgument named WorkflowThread");

 // この時点では実際の値がわかりません
 var outarg = output["WorkflowThread"];

 Assert.IsInstanceOfType(outarg, typeof(Int32),
  "WorkflowThread must be of type Int32");

 Assert.AreNotEqual(0, outarg,
  "WorkflowThread must not be zero");

 Debug.WriteLine("Test thread is " +
       Thread.CurrentThread.ManagedThreadId);
 Debug.WriteLine("Workflow thread is " + outarg.ToString());
}
```

(コード スニペット: Introduction to WF Lab - ShouldReturnWorkflowThread TestMethod VB)

Visual Basic

```
''' <summary>
''' ワークフローから出力引数が返されるかどうかの確認
''' 名前: WorkflowThread
''' 型: Int32
''' 値: 0 以外
''' </summary>
''' <remarks></remarks>
<TestMethod()> Public Sub ShouldReturnWorkflowThread()
 Dim output = WorkflowInvoker.Invoke( _
 New SayHello() With {.UserName = "Test"})

 Assert.IsTrue(output.ContainsKey("WorkflowThread"),
  "SayHello must contain an OutArgument named WorkflowThread")

 ' この時点では実際の値がわかりません
 Dim outarg = output("WorkflowThread")

 Assert.IsInstanceOfType(outarg, GetType(Int32),
  "WorkflowThread must be of type Int32")
```

```
 Assert.AreNotEqual(0, output("WorkflowThread"),
  "WorkflowThread must not be zero")

 Debug.WriteLine("Test thread is " & _
  Thread.CurrentThread.ManagedThreadId)
 Debug.WriteLine("Workflow thread is " & outarg.ToString())
End Sub
```

3. **Ctrl** キーを押しながら **R** キーを押し、次に **A** キーを押して、現在のコンテキストですべてのテストを実行します。**WorkflowThread** 出力引数をまだ追加していないので、テストが失敗することがわかります。

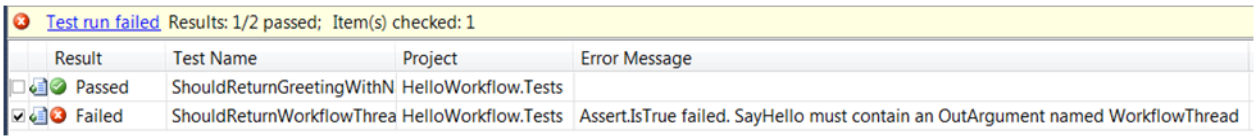

図 32

1 つのテストが成功し、WorkflowThread 引数がないためにもう 1 つのテストが失敗

タスク 2 – WorkflowThread を引数として返す

動作を確認するテストが完成したので、テストが成功するようワークフローを変更します。

1. **SayHello.xaml** を開き、**WorkflowThread** というで **Int32** 型の出力引数を追加します。

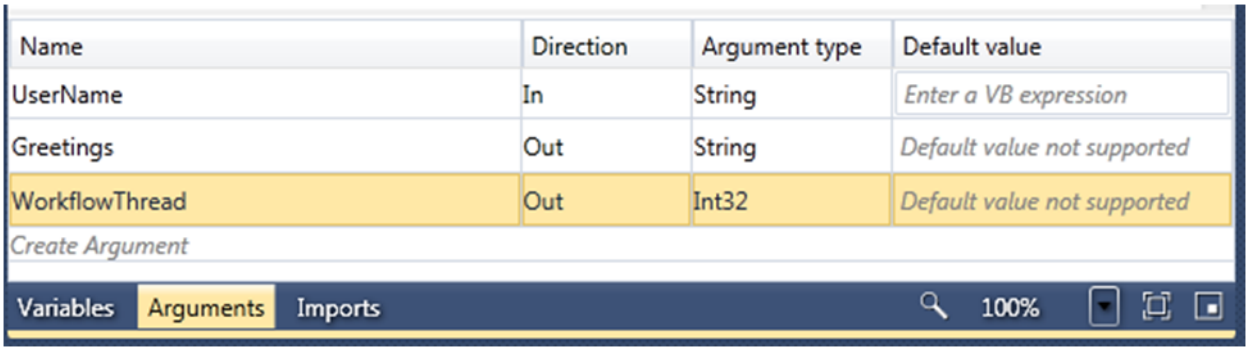

図 33

WorkflowThread 出力引数の追加

ここまで、ワークフローには 1 つしかアクティビティがありませんでした。しかし、ここでは、あいさつ文を割り当てるアクティビティと、ワークフロー スレッドを割り当てるアクティビティの 2 つのアクティビティが必要になりました。2 つの Assign

アクティビティを含む、1 つのアクティビティを使用するようにワークフローを変更する必要があります。そのために使用できるアクティビティはいくつかありますが、まずは最も簡単な、Sequence アクティビティについて説明します。

2. 既にデザイナーに配置されている Assign アクティビティを削除しない限り、デザイナーに Sequence アクティビティをドロップできません。この Assign アクティビティを Sequence アクティビティ内で使用するため、この Assign アクティビティを切り取り、Sequence アクティビティをドロップしてから再度貼り付ける必要があります。 Assign アクティビティを右クリックし、[Cut] (切り取り) をクリックします。

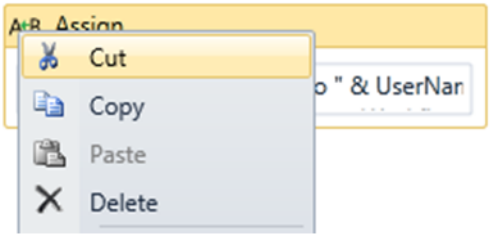

図 34

アクティビティの切り取り

3. **Sequence** アクティビティをドラッグし、デザイナー画面にドロップします。

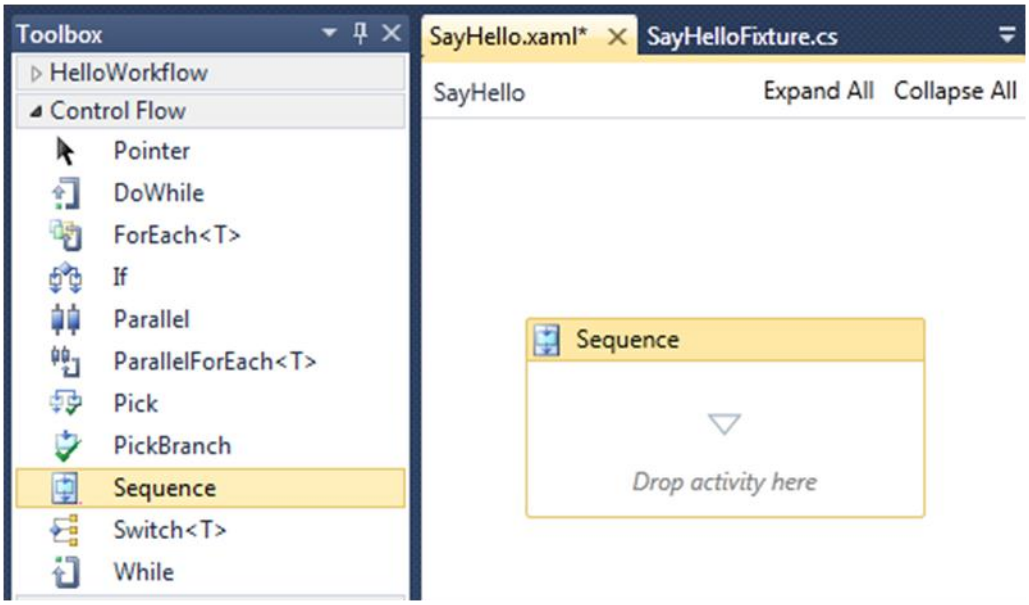

図 35

デザイナーへの Sequence アクティビティのドロップ

4. Sequence アクティビティ内部を右クリックし、[Paste] (貼り付け) をクリックして Assign アクティビティを Sequence アクティビティ内に配置します。

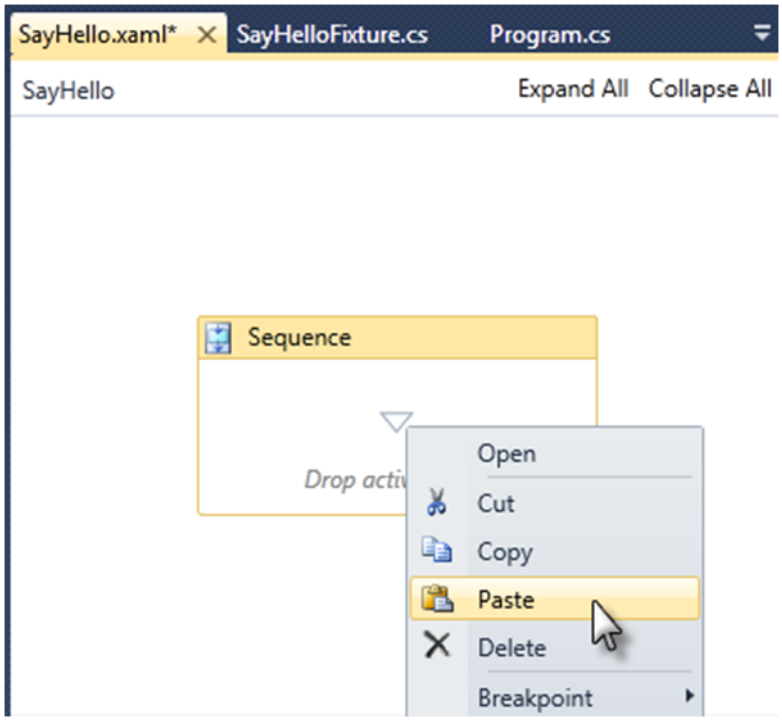

図 36

Sequence アクティビティ内への Assign アクティビティの貼り付け

5. **System.Threading** 名前空間をワークフローにインポートします。[Imports] (インポート) をクリックし、**System.Threading** を追加します。

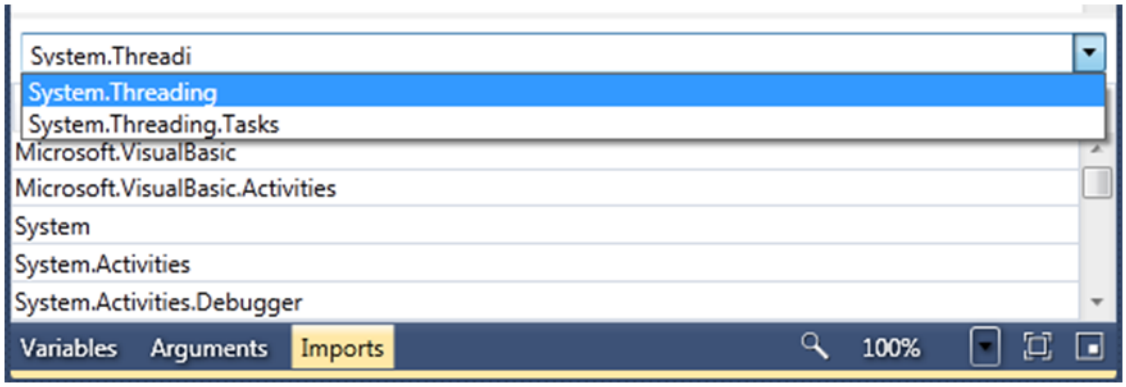

図 37

[Imports] (インポート) のクリックと System.Threading の追加

System.Threading を追加する必要性

特に必要性はありません。すべての C# プロジェクトや VB プロジェクトと同様、この名前空間は省略できます。インポートしなければ、この名前空間のクラスを完全修飾しなければならないだけです (**System.Threading.Thread** というように)。

6. ここで、現在のマネージ スレッド ID を **WorkflowThread** 出力引数に割り当てる必要があります。Sequence アクティビティの最初の Assign アクティビティの下に Assign アクティビティをドロップし、次のようにプロパティを設定します。

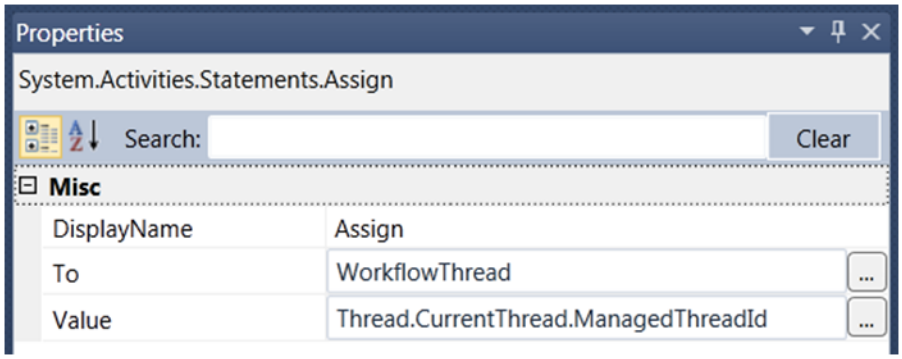

図 38

2 つ目の Assign アクティビティのプロパティの設定

7. **Ctrl** キー、**Shift** キー、**B** キーを同時に押して、ソリューションをリビルドします。

> **注意**
>
> Visual Studio 2010 Beta 2 のバグのため、テストを実行する前にビルドを行う必要があります。ビルドを行わずテストを実施すると、コードの変更が反映されず、正しいテストを実施することができません。

8. **Ctrl** キーを押しながら **R** キーを押し、次に **A** キーを押して、現在のコンテキストでテストを実行します。今度はテストが成功します。ワークフローから返されたスレッドを確認するには、**ShouldReturnWorkflowThread** テストをダブルクリックします。デバッグ出力がテスト結果に表示されます。実際のスレッド ID はコンピューターによって異なりますが、テストのスレッド ID とワークフローのスレッド ID が同じことがわかるでしょう。これは、**WorkflowInvoker** によって、呼び出しスレッドに同期してワークフローが呼び出されるためです。

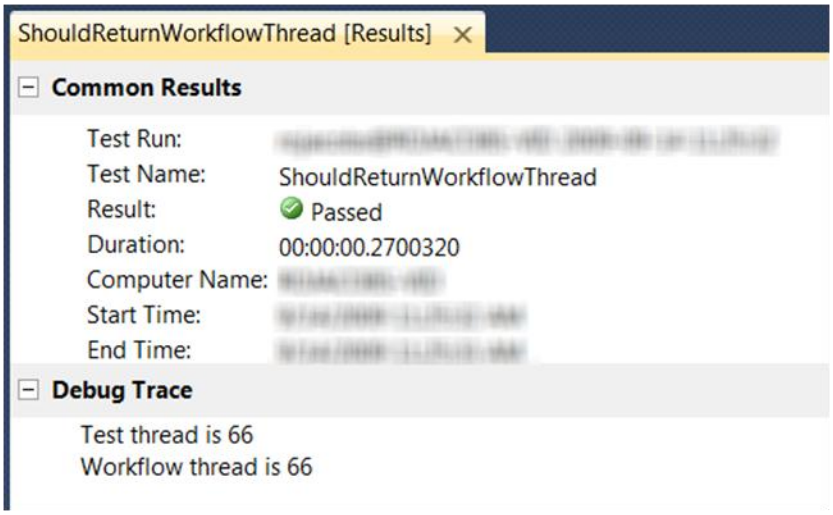

図 39

テスト結果に表示されるデバッグ トレースの出力

タスク 3 – テストを変更して WorkflowApplication を使用する

このプロジェクトのテストは適切ですが、問題点が 1 つあります。このテストでは、返された **WorkflowThread** が 0 でないことは確認されますが、ワークフローが実行されている実際のマネージ スレッド ID が返されたかどうかは確認されません。ワークフローが常に 1 を返しても、テストは成功します。

実際のスレッド ID を確認するには、**WorkflowApplication** を使用してスレッドを呼び出す必要があります。このタスクでは、ワークフローのスレッドを取得するようテストを変更します。そのためには、**WorkflowApplication.Completed** アクションの呼び出し時にスレッドを取得して、その値をワークフローから返された値と比較します。

1. **SayHelloFixture.cs** (C#) ファイルまたは **SayHelloFixture.vb** (VB) ファイルを開き、**ShouldReturnWorkflowThread** テストを探します。このテストを次のコードに置き換えて、**WorkflowApplication** クラスを使用してワークフローを実行します。このコードでは、**WorkflowApplication.Completed** アクションを使用して、出力引数とスレッド ID を取得します。

(Code Snippet - Introduction to WF4 Lab - ShouldReturnWorkflowThread WFApp Test CSharp)

C#

```
/// <summary>
/// Verifies that the workflow returns an Out Argument
```

```
/// Name: WorkflowThread
/// Type: Int32
/// Value: Non-Zero, matches thread used for Completed action
/// </summary>
[TestMethod]
public void ShouldReturnWorkflowThread()
{
  AutoResetEvent sync = new AutoResetEvent(false);
  Int32 actionThreadID = 0;
  IDictionary<string, object> output = null;

  WorkflowApplication workflowApp =
    new WorkflowApplication(
          new SayHello()
          {
            UserName = "Test"
          });

  // Create an Action<T> using a lambda expression
  // To be invoked when the workflow completes
  workflowApp.Completed = (e) =>
    {
      output = e.Outputs;
      actionThreadID = Thread.CurrentThread.ManagedThreadId;

      // Signal the test thread the workflow is done
      sync.Set();
    };

  workflowApp.Run();

  // Wait for the sync event for 1 second
  sync.WaitOne(TimeSpan.FromSeconds(1));

  Assert.IsNotNull(output,
    "output not set, workflow may have timed out");

  Assert.IsTrue(output.ContainsKey("WorkflowThread"),
    "SayHello must contain an OutArgument named WorkflowThread");

  // Don't know for sure what it is yet
  var outarg = output["WorkflowThread"];

  Assert.IsInstanceOfType(outarg, typeof(Int32),
    "WorkflowThread must be of type Int32");

  Assert.AreEqual(actionThreadID, (int)outarg,
    "WorkflowThread should equal actionThreadID");

  Debug.WriteLine("Test thread is " +
            Thread.CurrentThread.ManagedThreadId);
  Debug.WriteLine("Workflow thread is " + outarg.ToString());
}
```

(Code Snippet - Introduction to WF4 Lab – ShouldReturnWorkflowThread WFApp Test VB)

Visual Basic

```
''' <summary>
''' Verifies that the SayHello workflow contains an Out Argument
''' Name: WorkflowThread
''' Type: Int32
''' Value: Non-Zero, matches thread used for Completed action
''' </summary>
''' <remarks></remarks>
<TestMethod()> Public Sub ShouldReturnWorkflowThread()
  Dim sync As New AutoResetEvent(False)
  Dim actionThreadID As Int32 = 0
  Dim output As IDictionary(Of String, Object) = Nothing
  Dim workflowApp As New WorkflowApplication( _
    New SayHello() With {.UserName = "Test"})

  workflowApp.Completed = _
    Function(e)
      output = e.Outputs
      actionThreadID = Thread.CurrentThread.ManagedThreadId

      ' Signal the test thread the workflow is done
      sync.Set()

      ' VB requires a lambda expression to return a value
      ' It is not used with Action(Of T)
      Return Nothing
    End Function

  workflowApp.Run()

  ' Wait for the sync event for 1 second
  sync.WaitOne(TimeSpan.FromSeconds(1))

  Assert.IsNotNull(output, "output not set, workflow may have timed out")

  Assert.IsTrue(output.ContainsKey("WorkflowThread"),
          "SayHello must contain an OutArgument named WorkflowThread")

  ' Don't know for sure what it is yet
  Dim outarg = output("WorkflowThread")

  Assert.IsInstanceOfType(outarg, GetType(Int32),
              "WorkflowThread must be of type Int32")

  Assert.AreEqual(actionThreadID,
          DirectCast(output("WorkflowThread"), Integer),
          "WorkflowThread should equal actionThreadID")


  Debug.WriteLine("Test thread is " & _
          Thread.CurrentThread.ManagedThreadId)
  Debug.WriteLine("Workflow thread is " & outarg.ToString())
End Sub
```

> **イベントではなくデリゲート**
>
> **WorkflowApplication.Completed** など、**WorkflowApplication** のプロパティはイベントではなくデリゲートです。プロパティを扱うには、メソッド、匿名デリゲート、またはラムダ式を用意する必要があります。

次の手順

演習 6: 確認

演習 6: 確認

これで、テストを実行して、新しい要件でアクティビティのテストが成功するかどうか確認する準備ができました。

1. **Ctrl** キー、**Shift** キー、**B** キーを同時に押して、ソリューションをリビルドします。
2. **Ctrl** キーを押しながら **R** キーを押し、次に **A** キーを押して、現在のコンテキストでテストを実行します。
3. テスト実行が成功することを確認します。

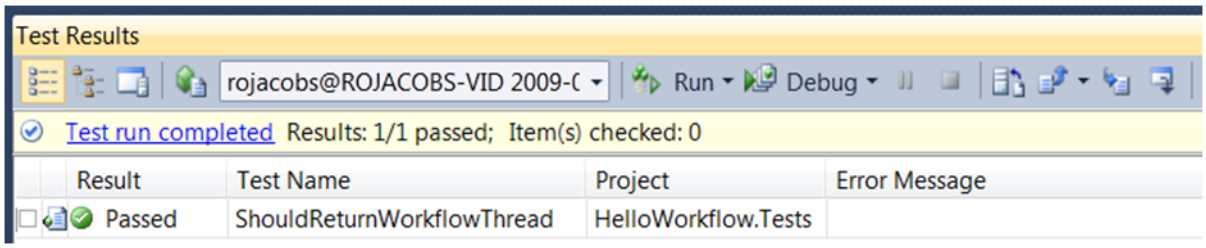

図 40

テストの成功

4. **ShouldReturnWorkflowThread** のテスト結果をダブルクリックして、スレッドに関するデバッグ メッセージを表示します。

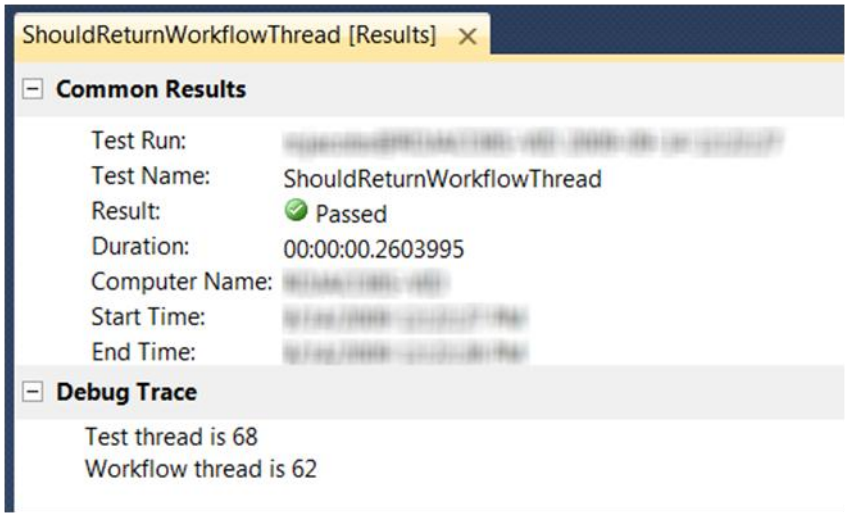

図 41

ワークフローが異なるスレッドで実行されたことを示すテスト結果

次の手順

演習 7: If/Else ロジックの追加

演習 7: If/Else ロジックの追加

前の演習では、独自のあいさつメッセージを使用するように、あいさつワークフロー アプリケーションを強化しました。この演習では、If/Else ロジックをワークフローに追加して、独自の条件に応じて異なるあいさつメッセージを表示します。

この演習でも "テストを先に作成する" 方式を採用します。つまり、先に新しい要件のテストを作成してから、テストの成功に必要なコードを実装します。

タスク 0 – ソリューションを開く

この演習では、演習 6 で作成したソリューションを使用することができます。演習 6 を完了していない場合は、次の手順で演習 7 を開始することができます。

1. Microsoft Visual Studio 2010 を管理者として起動します。[スタート] ボタンをクリックし、[すべてのプログラム]、[Microsoft Visual Studio 2010] の順にクリックします。

2. **%TrainingKitInstallFolder%\Labs\IntroToWF\Source\Ex7-IfElse\Begin** フォルダーには、C# と Visual Basic の開始ソリューションがあります。
3. **Ctrl** キー、**Shift** キー、**B** キーを同時に押して、ソリューションをビルドします。

タスク 1 – 新しい要件のテストを作成する

前の演習でアプリケーションに新しい要件を導入しました。名前の文字数が奇数であれば、あいさつ文の 1 語目を "Greetings" にし、偶数であれば "Hello" にします。たとえば、ワークフローをコードにすれば次のようになります。

C#

```
private static string SayHello(string userName)
{
 string FirstWord = null;

 if (userName.Length % 2 == 0)
  FirstWord = "Hello";
 else
  FirstWord = "Greetings";

 return FirstWord + ", " + userName +" from Workflow 4";
}
```

Visual Basic

```
Private Shared Function SayHello(ByVal userName As String) As String
 Dim FirstWord As String = Nothing

 If userName.Length Mod 2 = 0 Then
  FirstWord = "Hello"
 Else
  FirstWord = "Greetings"
 End If

 Return FirstWord & ", " & userName & " from Workflow 4"
End Function
```

1. 新しい要件を確認するテストを作成します。そのためには、**HelloWorkflow.Tests** プロジェクトの下にある **SayHelloFixture.cs** (C#) ファイルまたは **SayHelloFixture.vb** (VB) ファイルを開き、次のテストを追加します。

(Code Snippet - Introduction to WF4 Lab - ShouldReturnGreetingWithOddLengthName Test CSharp)

C#

```
[TestMethod]
```

```
public void ShouldReturnGreetingWithOddLengthName()
{
   var output = WorkflowInvoker.Invoke(
      new SayHello() { UserName = "Odd" });

   string greeting = output["Greeting"].ToString();

   Assert.AreEqual("Greetings Odd from Workflow 4", greeting);
}
```

(Code Snippet - Introduction to WF4 Lab - ShouldReturnGreetingWithOddLengthName Test VB)

Visual Basic

```
<TestMethod()>
Public Sub ShouldReturnGreetingWithOddLengthName()
   Dim output = WorkflowInvoker.Invoke(
      New SayHello() With {.UserName = "Odd"})
   Assert.AreEqual("Greetings Odd from Workflow 4",
            output("Greeting").ToString())
End Sub
```

2. テスト メソッドを右クリックし、[Run Tests] (テストの実行) () をクリックします。条件に応じて異なるあいさつメッセージを返すようにワークフローを変更していないため、テストは失敗します。

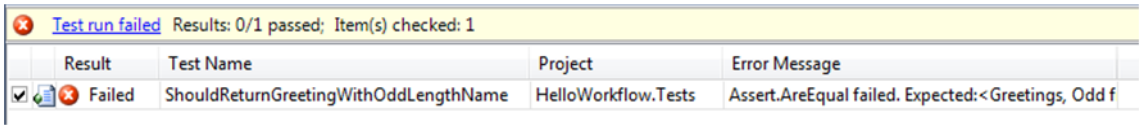

図 42

ShouldReturnGreetingWithOddLengthName テストの失敗

タスク 2 – ワークフローに新しい要件を実装する

1. ワークフロー デザイナーで **SayHello.xaml** を開きます。そのためには、ソリューション エクスプローラーでファイルをダブルクリックします。
2. あいさつメッセージの 1 語目 ("Hello" または "Greetings") を格納する **FirstWord** 変数を追加します。これを行うには、次の手順を実行します。
   a. アクティビティの図形をクリックして **Sequence** アクティビティを選択します。
   b. [Variables] (変数) をクリックします。**Sequence** アクティビティに使用できる変数を示すパネルが表示されます。

c. [Create Variable] (変数の作成) をクリックします。

d. [Name] (名前) ボックスに「FirstWord」と入力します。

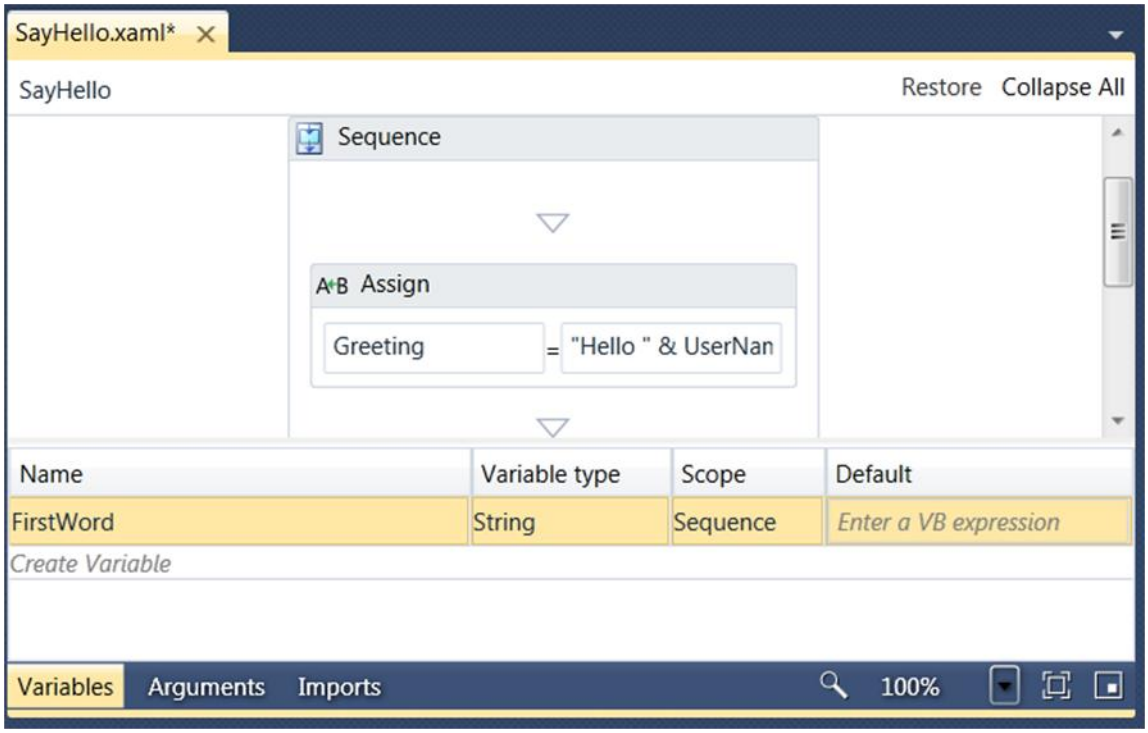

図 43

Sequence アクティビティへの FirstWord 変数 (String 型) の追加

変数

Windows Workflow Foundation (WF) の変数は、データの格納場所を表します。一方、引数はアクティビティに出入りするデータの流れを表します。C# や Visual Basic の場合と同様に、WF の変数にはスコープがあります。アクティビティを選択しないで変数のペインを開いても、変数を追加できません。デザイナーでアクティビティを選択することで、変数のスコープが指定されます。この場合、**FirstWord** 変数は **Sequence** スコープに属します。

3. 次に、**UserName** 引数をテストして、文字数が偶数か奇数か判断する必要があります。そのためには、ツールボックスから **If** アクティビティをドラッグし、**Sequence** アクティビティ内の **Assign** アクティビティの前にドロップします。

4. **If** アクティビティの **DisplayName** プロパティを **Set FirstWord** 値に設定します。そのためには、図形のテキストを選択してインラインで編集するか、プロパティ グリッド (アクティビティを選択して **F4** キーを押すと表示) でテキストを設定します。

> **メモ**: ワークフロー デザイナーでは、DisplayName プロパティを設定することで、図形にわかりやすい名前をつけることができます。

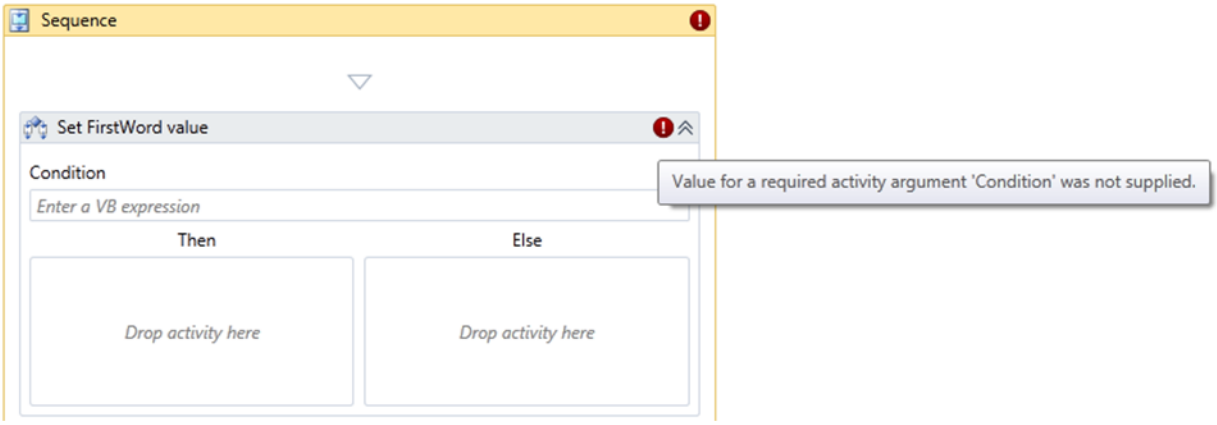

図 44

説明を設定した If アクティビティ

> **注意**
>
> 赤いアイコン ( ) は、現在このワークフローが有効ではないことを警告しています。**If** アクティビティではさらに Condition 引数に式を設定する必要があります。

5. **If** 条件の式を指定します。これを行うには、If アクティビティをダブルクリックして開き、[Condition] (条件) ボックスに以下の式を入力します。この式によって、名前の長さが奇数か偶数かを確認します。

```
Visual Basic
UserName.Length Mod 2 = 0
```

> **式**

"式" はプログラム ステートメントで、リテラル文字列、条件付きステートメント、または複数の文字列を連結したり、メソッドや他のアクティビティを呼び出したりする式を指定できます。アプリケーションが C# で記述されていても、式は Visual Basic 構文を使用して記述します。つまり、大文字と小文字は区別されず、比較は "==" ではなく 1 つの等号を使用し、ブール演算子は "&&" と "||" といった記号ではなく "And" と "Or" といった語句です。

6. 名前の長さが偶数の場合に **FirstWord** 変数の値を更新し、"Hello" というメッセージを表示します。ツールボックスから **Assign** アクティビティをドラッグし、**Then** 領域にドロップします。続いて、[To] (To) ボックス (左側) に「FirstWord」と入力し、[Value] (値) ボックス (右側) に「"Hello "」と入力します (末尾にスペースを含めます)。
7. 今度は名前の長さが奇数の場合に **FirstWord** 変数の値を更新し、"Greetings" というメッセージを表示します。ツールボックスから **Assign** アクティビティをドラッグし、**Else** 領域にドロップします。続いて、[To] (To) ボックス (左側) に「FirstWord」と入力し、[Value] (値) ボックス (右側) に「"Greetings "」と入力します (末尾にスペースを含めます)。

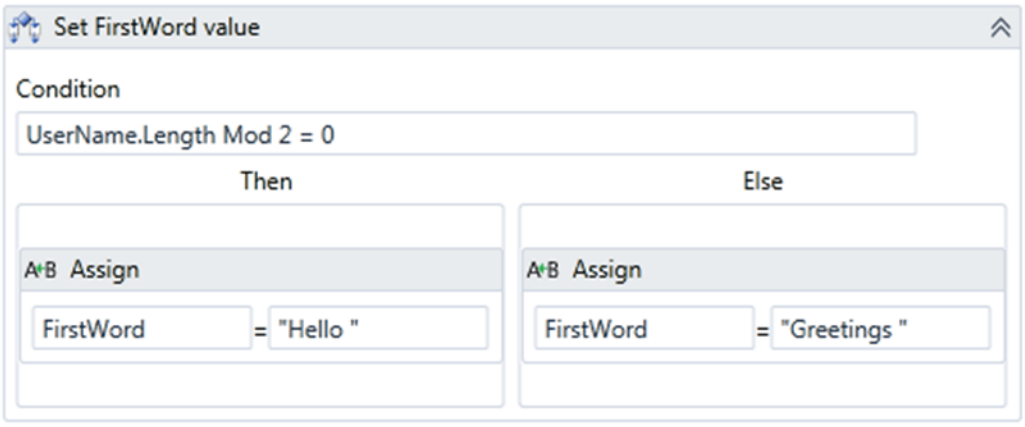

図 45

完成した If アクティビティ

8. 最後の **Assign** アクティビティを変更して、**FirstWord** 変数の値に基づいて表示するあいさつメッセージをカスタマイズします。これを行うには、If アクティビテ

ィの下にある **Assign** アクティビティで、[Value] (値) ボックス (右側) の内容を次の式に置き換えます。

Visual Basic

```
FirstWord & UserName & " from Workflow 4"
```

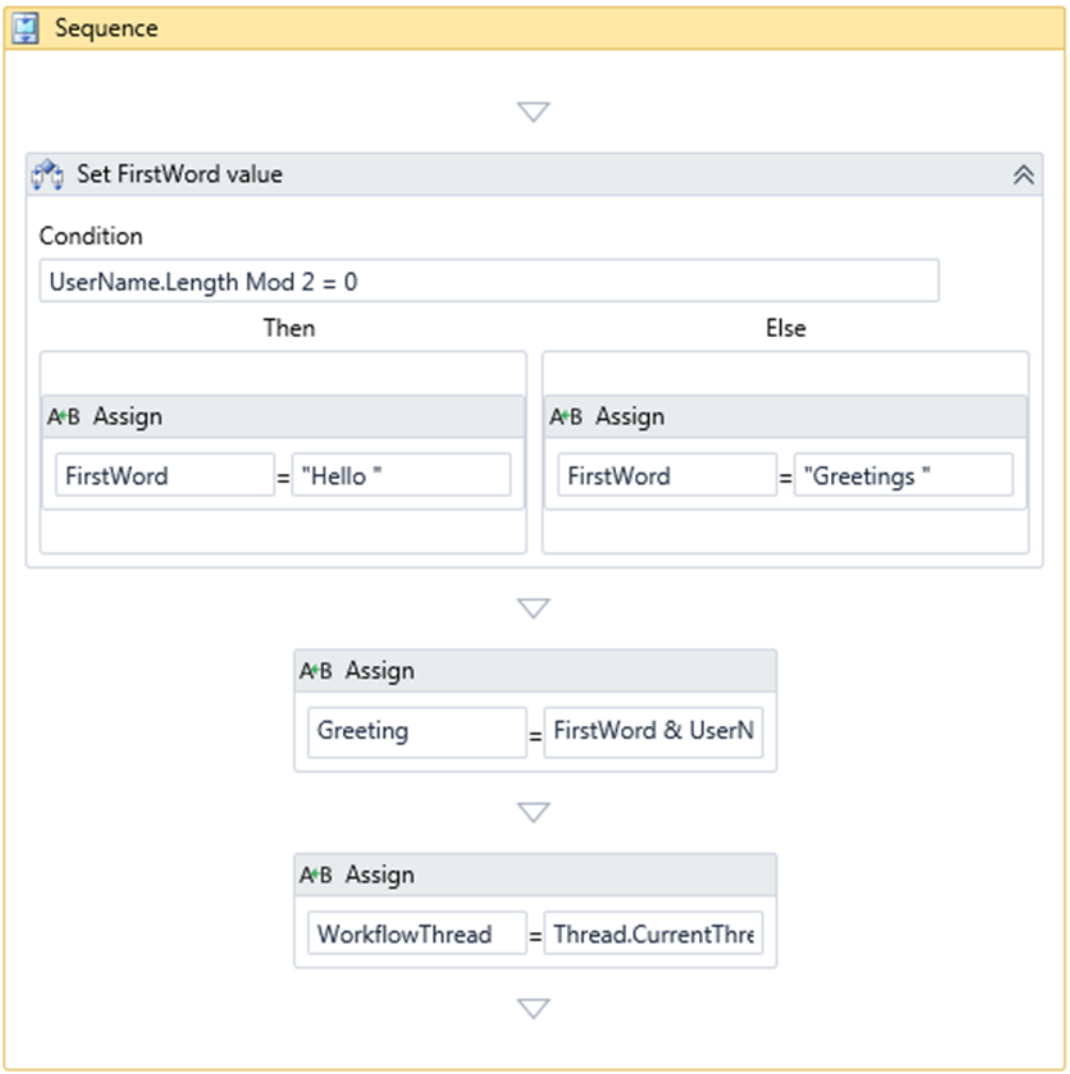

図 46

完成したワークフロー

9. **Ctrl** キー、**Shift** キー、**B** キーを同時に押して、ソリューションをビルドします。

次の手順

演習 7: 確認

演習 7: 確認

演習のすべての手順を正しく実行したことを確認するには、ソリューションのすべてのテストを実行して、これらのテストが成功することを確かめます。

1. [Test] (テスト) メニューの [Run] (実行) をポイントし、[All Tests In Solution] (ソリューションのすべてのテスト) をクリックします。または、**Ctrl** キーを押しながら **R** キーを押し、次に **A** キーを押します。
2. すべてのテストが成功することを確認します。

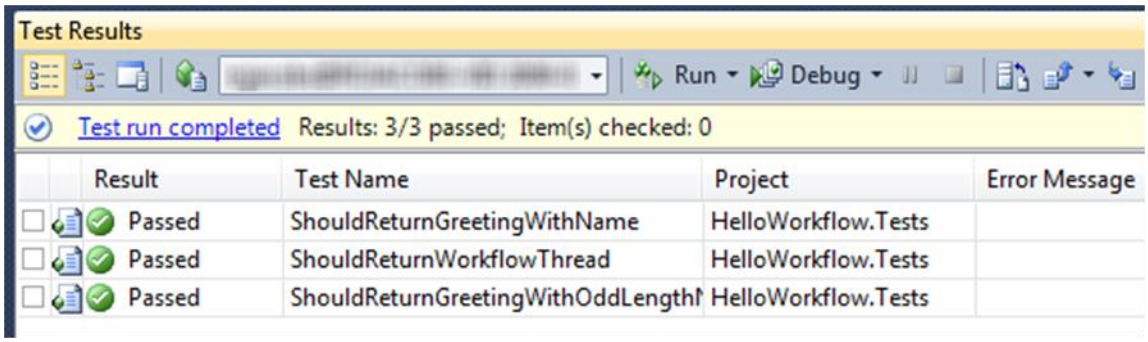

図 47

テストの成功

次の手順

演習 8: エラー処理

演習 8: エラー処理

メモ: 演習 7 を完了していない場合は、このラボで用意したソリューションを使用できます。**%TrainingKitInstallFolder%\Labs\IntroToWF\Source\Ex8-ErrorHandling\Begin** フォルダーには、C# と Visual Basic の開始ソリューションがあります。

この簡単なアプリケーションには潜在的なバグがあることに気付いたかもしれません。UserName 引数をワークフローに渡さないとどうなるでしょう。この演習では、組み込みアクティビティの **TryCatch**、**Catch<T>**、および **Throw** を使用して、エラーをハンドルする機能をワークフローに追加します。

タスク 0 – ソリューションを開く

この演習では、演習 7 で作成したソリューションを使用することができます。演習 7 を完了していない場合は、次の手順で演習 8 を開始することができます。

1. Microsoft Visual Studio 2010 を管理者として起動します。[スタート] ボタンをクリックし、[すべてのプログラム]、[Microsoft Visual Studio 2010] の順にクリックします。
2. **%TrainingKitInstallFolder%\Labs\IntroToWF\Source\Ex8-ErrorHandling\Begin** フォルダーには、C# と Visual Basic の開始ソリューションがあります。
3. **Ctrl** キー、**Shift** キー、**B** キーを同時に押して、ソリューションをビルドします。

タスク 1 – エラーの動作を確認するテストを作成する

空文字列 ("") を渡しても、アプリケーションは正しく機能することに注意してください。ワークフローに名前として null を渡す唯一の方法は、UserName に入力引数を指定せずにワークフローを作成することです。

1. 名前引数をワークフローに渡さない場合に起きることを確認するテストを作成します。そのためには、ソリューション エクスプローラーを使用して、**HelloWorkflow.Tests** プロジェクトの下にある **SayHelloFixture.cs** (C#) ファイルまたは **SayHelloFixture.vb** (Visual Basic) ファイルを開き、次のテストを追加します。

(Code Snippet - Introduction to WF4 Lab - ShouldHandleNullUserName Test CSharp)

C#

```
[TestMethod]
public void ShouldHandleNullUserName()
{
 // 引数を指定しないで呼び出します
 WorkflowInvoker.Invoke(new SayHello());
}
```

(Code Snippet - Introduction to WF4 Lab - ShouldHandleNullUserName Test VB)

Visual Basic

```
<TestMethod()>
Public Sub ShouldHandleNullUserName()

 ' 引数を指定しないで呼び出します
 WorkflowInvoker.Invoke(New SayHello())

End Sub
```

2. ここで、テストを実行します。テスト メソッド名を右クリックし、[Run Tests] (テストの実行) ( ) をクリックします。

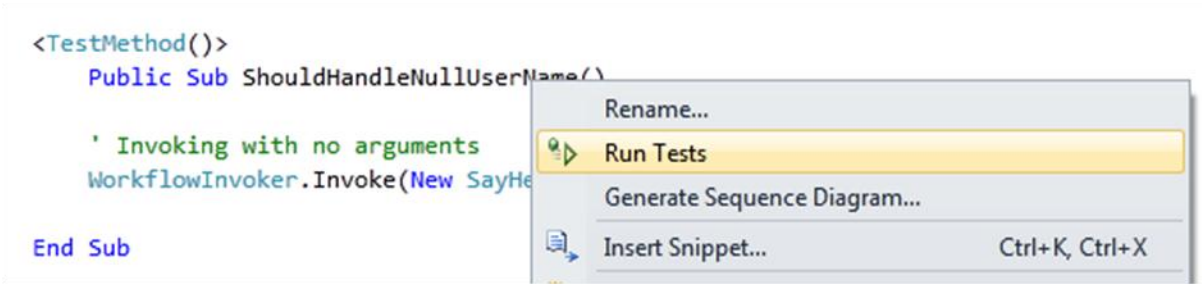

図 48

テストの実行

3. その結果、UserName.Length を使用する式が If アクティビティにあっても、UserName が null のため、**NullReferenceException** が発生してテストが失敗します。

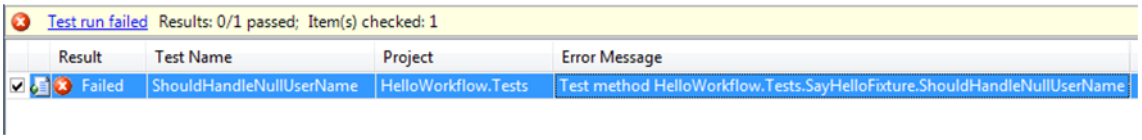

図 49

ShouldHandleNullUserName テストの失敗

テスト結果

HelloWorkflow.Tests.SayHelloFixture.ShouldHandleNullUserName threw exception: System.NullReferenceException: Object reference not set to an instance of an object.(テスト メソッド HelloWorkflow.Tests.SayHelloFixture.ShouldHandleNullUserName が例外をスローしました: System.NullReferenceException: オブジェクト参照がオブジェクト インスタンスに設定されていません。)

タスク 2 – ワークフローに TryCatch アクティビティを追加する

エラーをハンドルする場合、名前引数を使用前に検証することも、単に例外をキャッチして処理することもできます。このタスクでは、例外をキャッチします。

1. **SayHello.xaml** をデザイナーで開き、ツールボックスから **TryCatch** アクティビティをドラッグして Sequence アクティビティ上にドロップします。

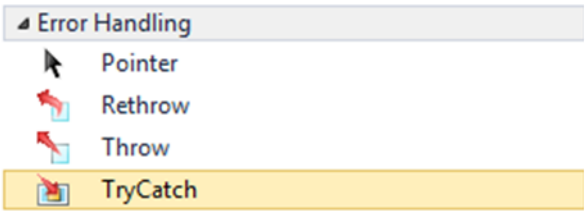

図 50

ツールボックスの TryCatch アクティビティ

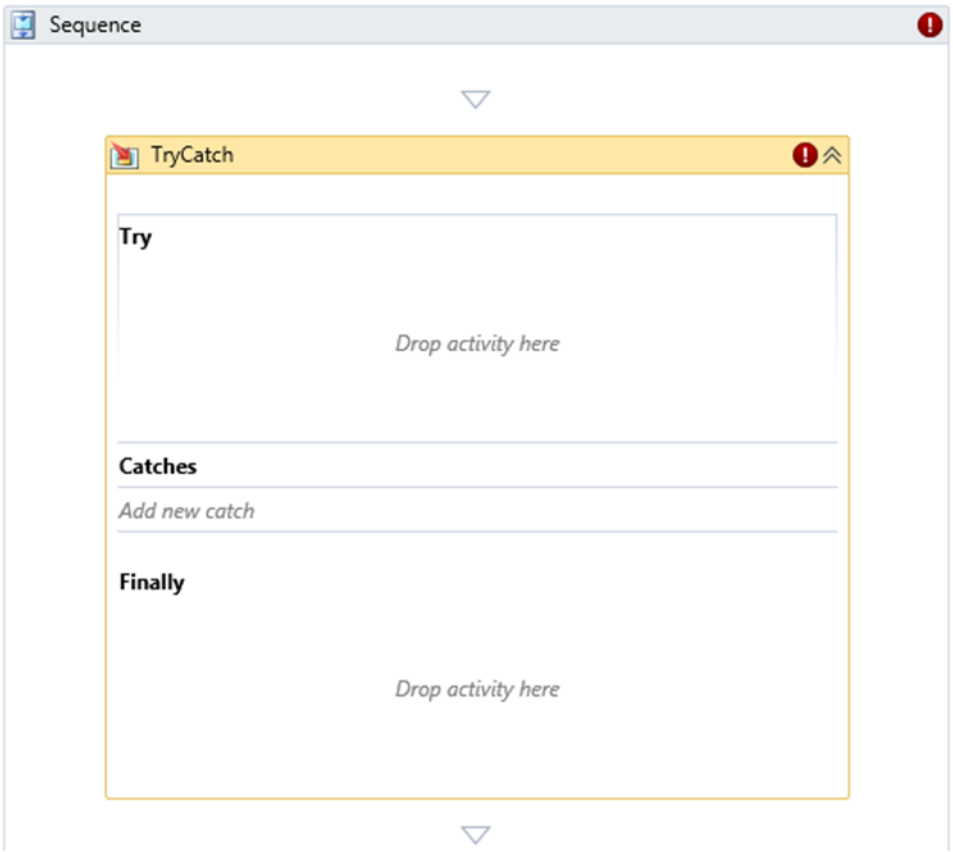

図 51

TryCatch アクティビティの追加

TryCatch アクティビティ

ワークフローでは **TryCatch** アクティビティを使用して、ワークフローの実行中に発生した例外をハンドルできます。これらの例外は、ハンドルすることも、**Throw** アクティビティを使用して再度スローすることもできます。Finally セクションのアクティビティは、**Try** セクションと **Catches** セクションのいずれかが完了すると実行されます。

2. ここで、**If** アクティビティを **Try** ブロック内に移動する必要があります。
   a. If アクティビティを折りたたみ ( )、図を縮小します。
   b. **If** アクティビティを **Try** ブロックにドラッグ アンド ドロップします。

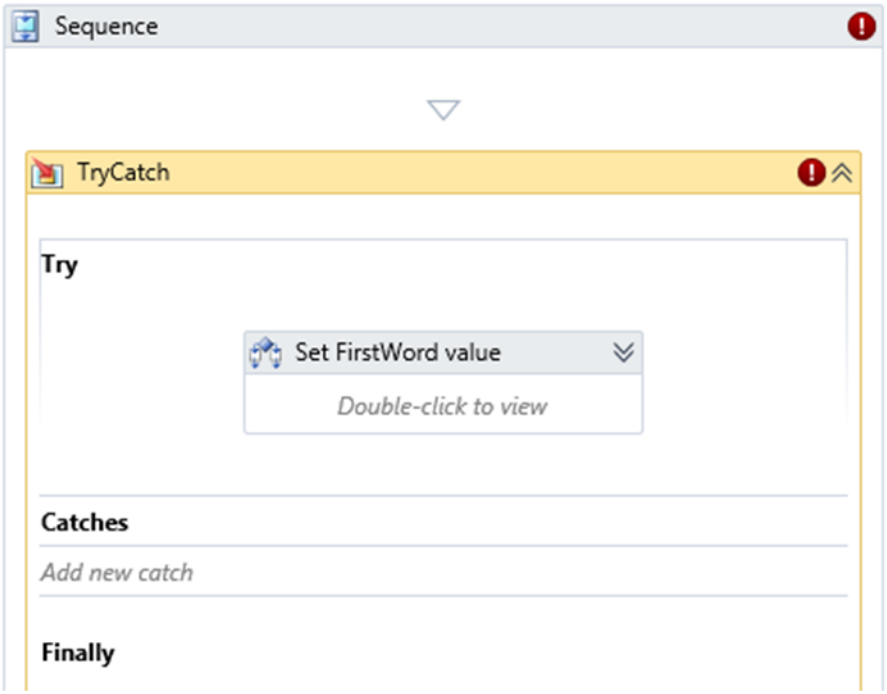

図 52

If アクティビティの移動

3. 次に、**NullReferenceException** をキャッチします。これを行うには、次の手順を実行します。
   a. [Add new catch] (新しい Catch を追加) をクリックします。

b. コンボ ボックスの一覧で [System.NullReferenceExeption] をクリックし、**Enter** キーを押します。

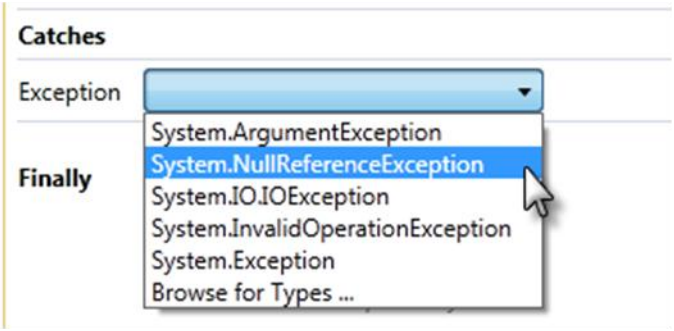

図 53

NullReferenceExeption の選択

4. アクティビティの **Catches** セクションでは、エラーのハンドル方法を決める必要があります。この場合は、キャッチした例外を **ArgumentNullException** に置き換えてスローします。このようにすると、呼び出し側に、例外が発生したのは **UserName** 引数を指定しなかった呼び出し側の過失だと知らせることができます。ツールボックスから **Throw** アクティビティをドラッグし、**Catches** 領域にドロップします。

5. **Throw** アクティビティの式を設定します。これを行うには、Throw アクティビティを選択して、[Properties] (プロパティ) ウィンドウの [Exception] (例外) ボックスに以下の式を入力します。

Visual Basic

```
New ArgumentNullException("UserName")
```

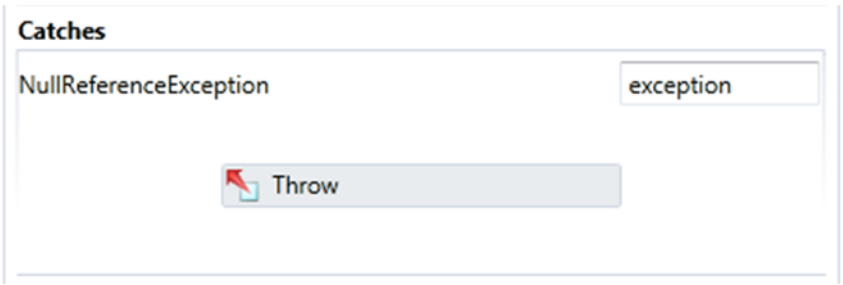

図 54

Throw アクティビティの追加

6. 今度は、この例外を想定して、**ShouldHandleNullUserName** テストを修正します。これを行うには、**SayHelloFixture.cs** を開き、次のコードに示すように、テスト メソッドに **ExpectedException** 注釈を追加します。

C#

```
[TestMethod]
[ExpectedException(typeof(ArgumentNullException))]
public void ShouldHandleNullUserName()
{
 // 引数を指定しないで呼び出します
 WorkflowInvoker.Invoke(new SayHello());
}
```

Visual Basic

```
<TestMethod(), ExpectedException(GetType(ArgumentNullException))>
Public Sub ShouldHandleNullUserName()

   ' 引数を指定しないで呼び出します
   WorkflowInvoker.Invoke(New SayHello())

End Sub
```

次の手順

演習 8: 確認

演習 8: 確認

1. **Ctrl** キー、**Shift** キー、**B** キーを同時に押して、ソリューションをビルドします。
2. [Test] (テスト) メニューの [Run] (実行) をポイントし、[All Tests In Solution] (ソリューションのすべてのテスト) をクリックします。または、**Ctrl** キーを押しながら **R** キーを押し、次に **A** キーを押します。
3. すべてのテストが成功することを確認します。

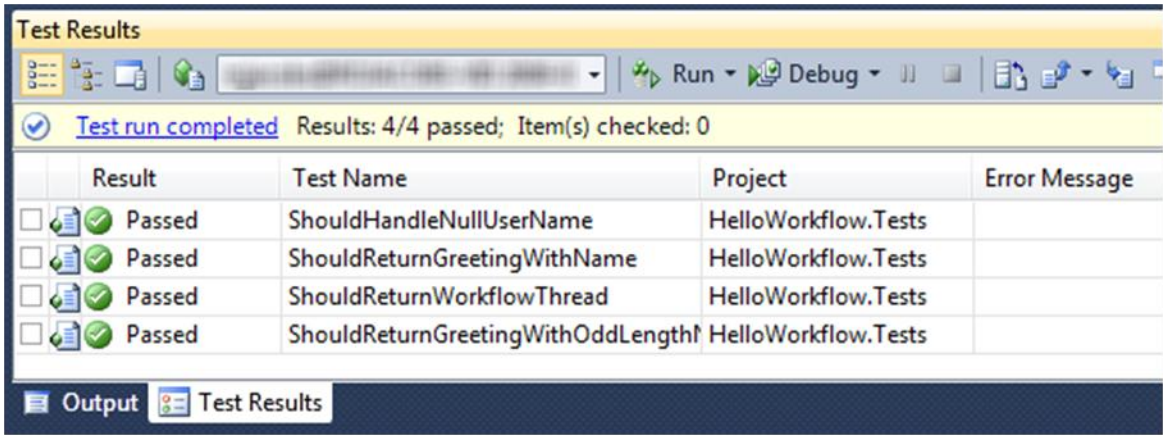

図 55

テストの成功

次の手順

演習 9: アクティビティ デザイナー

演習 9: アクティビティ デザイナー

既に説明したように、Windows Workflow ではコードを使ってカスタム アクティビティを構築できます。基本クラスに応じて、構築できるカスタム アクティビティが数種類あります。

基本クラス	用途
Activity	他のアクティビティから構成されるアクティビティ
CodeActivity	実行の制御を目的とするアクティビティ
AsyncCodeActivity	実行中に非同期処理を実行するアクティビティ
NativeActivity	他のアクティビティを含むか、ワークフロー ランタイムの高度なサービスを必要とするアクティビティ

タスク 0 – ソリューションを開く

この演習では、演習 8 で作成したソリューションを使用することができます。演習 8 を完了していない場合は、次の手順で演習 9 を開始することができます。

1. Microsoft Visual Studio 2010 を管理者として起動します。[スタート] ボタンをクリックし、[すべてのプログラム]、[Microsoft Visual Studio 2010] の順にクリックします。
2. **%TrainingKitInstallFolder%\Labs\IntroToWF\Source\Ex9-ActivityDesigner\Begin** フォルダーには、C# と Visual Basic の開始ソリューションがあります。
3. **Ctrl** キー、**Shift** キー、**B** キーを同時に押して、ソリューションをビルドします。

タスク 1 – カスタム NativeActivity を作成する

このタスクでは、前処理機能と後処理機能を備えた簡単なカスタム アクティビティを作成します。

1. **Ctrl** キー、**Shift** キー、**B** キーを同時に押して、ソリューションをビルドします。
2. ソリューション エクスプローラーで HelloWorkflow ソリューション ファイルを右クリックし、[Add] (追加) をポイントして、[New Project] (新しいプロジェクト) をクリックします。
3. [Workflow] (Workflow) テンプレートを選択し、[Activity Library] (アクティビティ ライブラリ) をクリックします。プロジェクトに「HelloWorkflow.Activities」という名前を付けます。

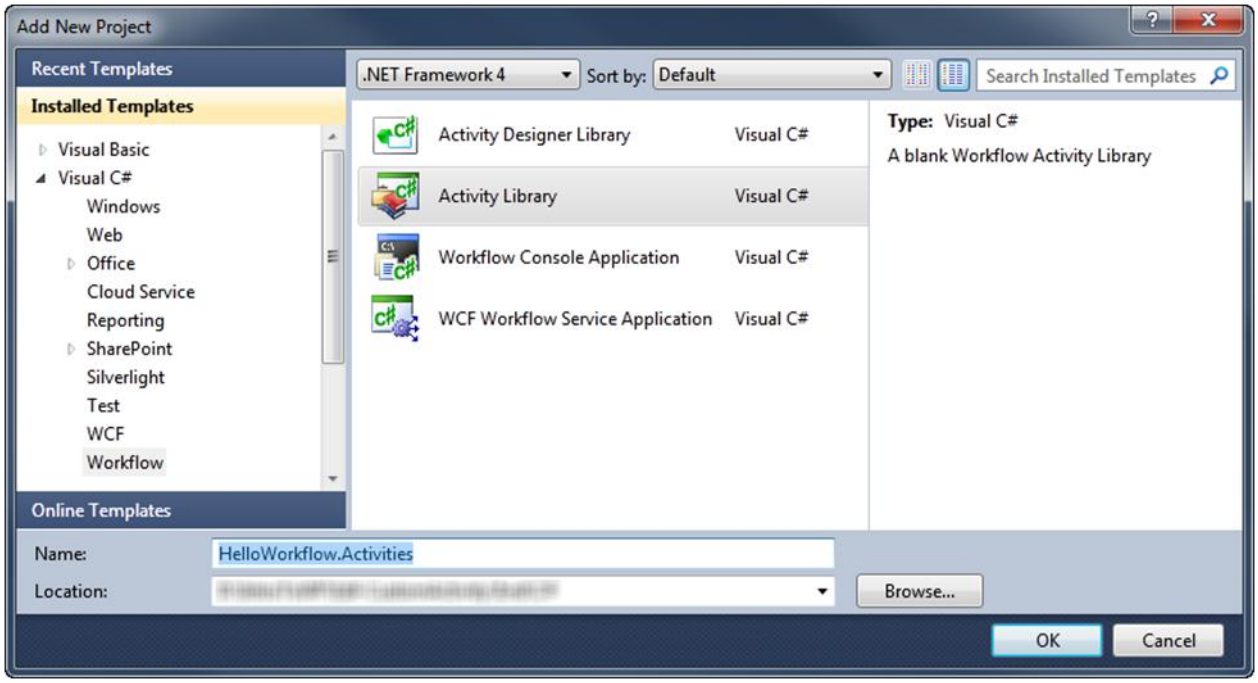

図 56

HelloWorkflow.Activities という新しいアクティビティ ライブラリの追加

4. **Activity1.xaml** を削除します。このラボには必要ありません。
5. ソリューション エクスプローラーで HelloWorkflow.Activities プロジェクトを右クリックし、[Add] (追加) をポイントして、[New Item] (新しい項目) をクリックします (または、**Ctrl** キー、**Shift** キー、**A** キーを同時に押します)。
6. [Workflow] (Workflow) テンプレートで、[Code Activity] (コード アクティビティ) をクリックし、「PrePostSequence」という名前を付けます。

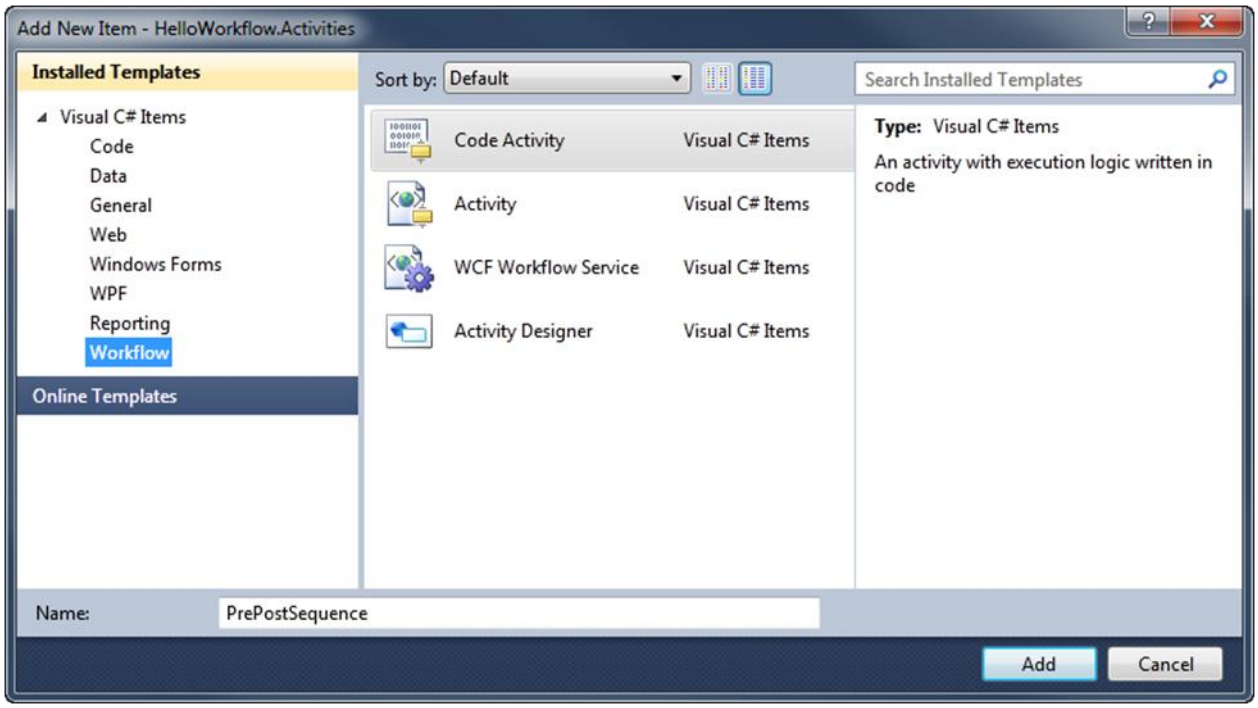

図 57

PrePostSequence という新しいコード アクティビティの追加

7. **PrePostSequence** クラスは、他のアクティビティのコンテナーとして機能します。テンプレートに既に入力されているコードに少し変更を加える必要があります。クラスの内容を削除し、次のコードに置き換えます。

(Code Snippet - Introduction to WF4 Lab – PrePostSequence Class CSharp)

C#

```
public sealed class PrePostSequence : NativeActivity
{
   public Activity Pre { get; set; }
   public Activity Post { get; set; }
   public List<Activity> Activities { get; set; }
```

```
    public PrePostSequence()
    {
        Activities = new List<Activity>();
    }

    protected override void Execute(NativeActivityContext context)
    {
        // アクティビティのスケジュールを順番に設定します
        context.ScheduleActivity(Pre);
        Activities.ForEach((a) => { context.ScheduleActivity(a); });
        context.ScheduleActivity(Post);
    }
}
```

(Code Snippet - Introduction to WF4 Lab – PrePostSequence Class VB)

Visual Basic

```
Public NotInheritable Class PrePostSequence
    Inherits NativeActivity

    Public Property Pre() As Activity
    Public Property Post() As Activity
    Public Property Activities() As List(Of Activity)

    Public Sub New()
        Activities = New List(Of Activity)()
    End Sub

    Protected Overrides Sub Execute(ByVal context As System.Activities.NativeActivityContext)
        ' アクティビティのスケジュールを順番に設定します
        context.ScheduleActivity(Pre)
        For Each Activity In Activities
            context.ScheduleActivity(Activity)
        Next
        context.ScheduleActivity(Post)
    End Sub
End Class
```

8. **Ctrl** キー、**Shift** キー、**B** キーを同時に押して、ソリューションをビルドします。
9. **HelloWorkflow** プロジェクトでは、**HelloWorkflow.Activities** プロジェクトへの参照が必要です。この参照を追加するには ソリューション エクスプローラーで HelloWorkflow プロジェクトを右クリックし、[Add Refference] (参照の追加) を選択します。ダイアログの[Project] (プロジェクト) タブで Hello.Activities を選択し、[OK] をクリックします。

10. デザイナーで **SayHello.xaml** を開きます。ツールボックスに **PrePostSequence** アクティビティが表示されるようになったことがわかります。

> **注意**
>
> Visual Studio 2010 Beta 2 ではカスタム アクティビティがツールボックスに表示されない場合があるという問題があります。表示されない場合は、ソリューションの項目のフォルダーをソリューションエクスプローラー上で削除し、リビルドを行ってください。もし、これで修復しない場合は、ツールボックスに手動で追加してください。

回避策 – カスタム アクティビティがツールボックス上に表示されない場合

1. ツールボックスを右クリックし、追加タブを選択します。
2. タブに **HelloWorkflow.Activities** と名前を付けます。
3. **HelloWorkflow.Activities** グループで右クリックし、[アイテムの選択]を選択します。
4. ツールボックス アイテムの選択ダイアログが表示されたら、**System.Activities components** タブをクリックします。
5. [参照] ボタンをクリックします。
6. **HelloWorkflow.Activities** 直下の **bin\Debug** フォルダーに移動し、**HelloWorkflow.Activities.dll** を選択します。
7. 新しいアクティビティがチェックボックスにチェックが入った状態で表示されます。[OK]をクリックしてダイアログを閉じます。

タスク 2 – カスタム アクティビティをデザインビューにドラッグする

1. このアクティビティを **Finally** ブロックのすぐ下の画面にドラッグします。アクティビティは機能しますが、この段階ではそれほど役立つアクティビティではありません。カスタム デザイナーを作成する必要があります。

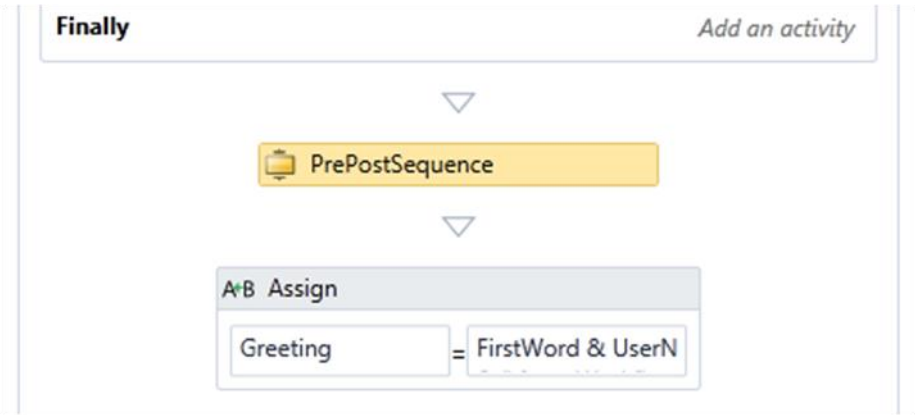

図 58

カスタム デザイナーがない PrePostSequence アクティビティ

2. デザイナー画面から **PrePostSequence** アクティビティを削除します。
3. Visual Studio でアセンブリを閉じることができるよう **SayHello.xaml** を閉じます。

タスク 3 – アクティビティ デザイナーを作成する

このタスクでは、**PrePostSequence** アクティビティ用のカスタム アクティビティ デザイナーを追加します。

1. ソリューション エクスプローラーで HelloWorkflow ソリューション ファイルを右クリックし、[Add] (追加) をポイントして、[New Project] (新しいプロジェクト) をクリックします。
2. [Workflow] (Workflow) テンプレートを選択し、[Activity Designer Library] (アクティビティ デザイナー ライブラリ) をクリックします。プロジェクトに「HelloWorkflow.Activities.Designers」という名前を付けます。

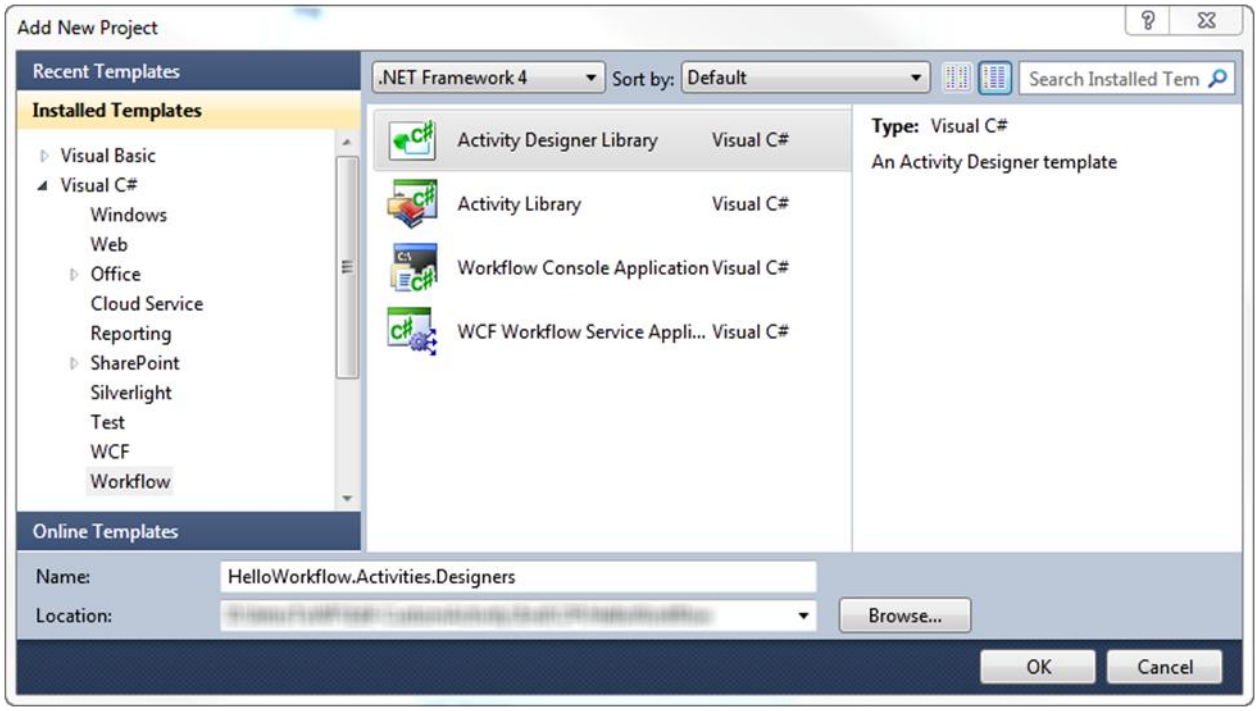

図 59

新しいアクティビティ デザイナー ライブラリの追加

3. **ActivityDesigner1.xaml** を削除します。このファイルは必要ありません。
4. ソリューション エクスプローラーで **HelloWorkflow.Activities.Designers** プロジェクトを右クリックし、[Add] (追加) をポイントして、[New Item] (新しい項目) をクリックします。
5. [Workflow] (Workflow) テンプレートで、[Activity Designer] (アクティビティ デザイナー) をクリックし、「PrePostSequenceDesigner」という名前を付けます。

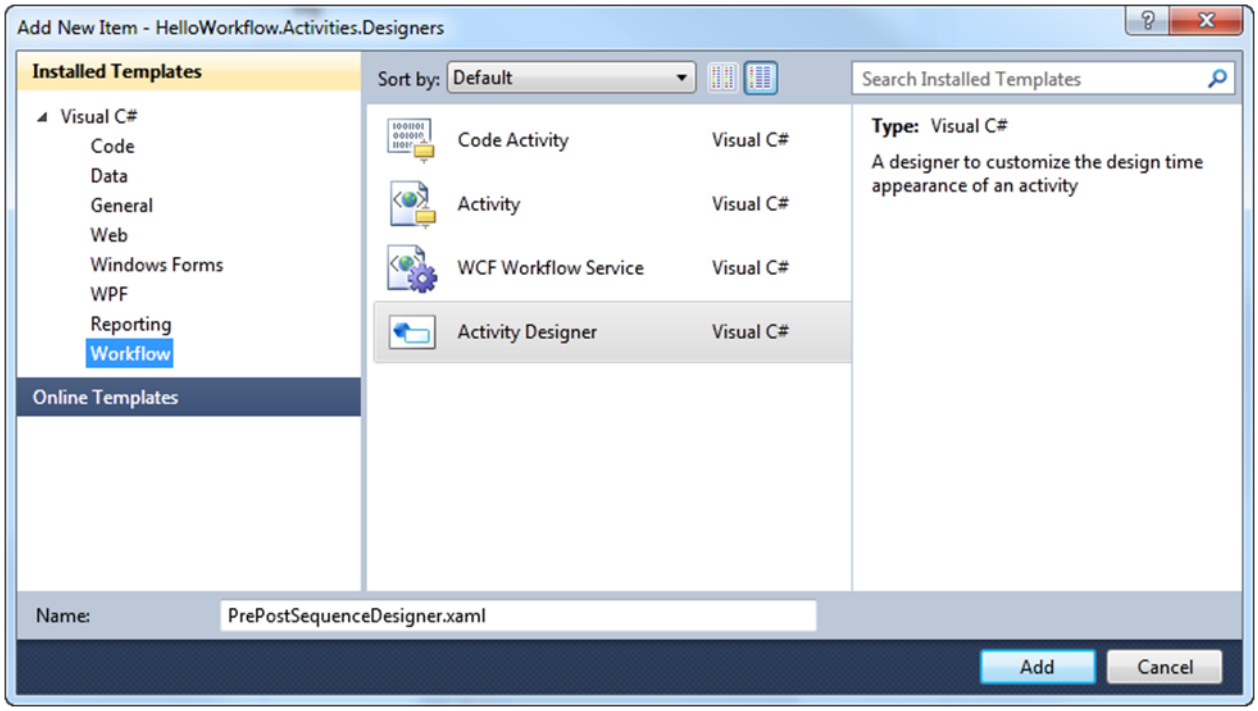

図 60

PrePostSequenceDesigner アクティビティ デザイナーの追加

6. テンプレートの XAML を次のコードに置き換えます。

(Code Snippet - Introduction to WF4 Lab – PrePostSequenceDesigner XAML CSharp)

XAML (C#)

```
<sap:ActivityDesigner x:Class="HelloWorkflow.Activities.Designers.PrePostSequenceDesigner"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:sap="clr-
namespace:System.Activities.Presentation;assembly=System.Activities.Presentation"
  xmlns:sapv="clr-
namespace:System.Activities.Presentation.View;assembly=System.Activities.Presentation">
  <StackPanel>
    <Border BorderBrush="Green" BorderThickness="4" CornerRadius="5">
      <sap:WorkflowItemPresenter Item="{Binding Path=ModelItem.Pre, Mode=TwoWay}"
HintText="Insert Pre Activities Here"/>
    </Border>
    <Border BorderBrush="Red" BorderThickness="4" CornerRadius="5">
      <StackPanel>
        <Border BorderBrush="Black" BorderThickness="2" CornerRadius="5">
          <TextBlock HorizontalAlignment="Center">Activities</TextBlock>
        </Border>
        <sap:WorkflowItemsPresenter Items="{Binding Path=ModelItem.Activities}"
HintText="Insert Activities Here">
          <sap:WorkflowItemsPresenter.SpacerTemplate>
            <DataTemplate>
              <Ellipse Fill="Red" Width="30" Height="30" />
            </DataTemplate>
          </sap:WorkflowItemsPresenter.SpacerTemplate>
```

```
            <sap:WorkflowItemsPresenter.ItemsPanel>
              <ItemsPanelTemplate>
                <StackPanel Orientation="Horizontal"/>
              </ItemsPanelTemplate>
            </sap:WorkflowItemsPresenter.ItemsPanel>
          </sap:WorkflowItemsPresenter>
        </StackPanel>
      </Border>
      <Border BorderBrush="Black" BorderThickness="4" CornerRadius="5">
        <sap:WorkflowItemPresenter Item="{Binding Path=ModelItem.Post, Mode=TwoWay}"
HintText="Insert Post Activities Here"/>
      </Border>
    </StackPanel>
</sap:ActivityDesigner>
```

(Code Snippet - Introduction to WF4 Lab – PrePostSequenceDesigner VB)

XAML (VB)

```
<sap:ActivityDesigner x:Class="PrePostSequenceDesigner"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:sap="clr-
namespace:System.Activities.Presentation;assembly=System.Activities.Presentation"
    xmlns:sapv="clr-
namespace:System.Activities.Presentation.View;assembly=System.Activities.Presentation">
    <StackPanel>
      <Border BorderBrush="Green" BorderThickness="4" CornerRadius="5">
        <sap:WorkflowItemPresenter
          Item="{Binding Path=ModelItem.Pre, Mode=TwoWay}"
           HintText="Insert Pre Activities Here"/>
      </Border>
      <Border BorderBrush="Red" BorderThickness="4" CornerRadius="5">
        <StackPanel>
          <Border BorderBrush="Black"
            BorderThickness="2" CornerRadius="5">
            <TextBlock
              HorizontalAlignment="Center">Activities</TextBlock>
          </Border>
          <sap:WorkflowItemsPresenter
           Items="{Binding Path=ModelItem.Activities}"
           HintText="Insert Activities Here">
            <sap:WorkflowItemsPresenter.SpacerTemplate>
              <DataTemplate>
                <Ellipse Fill="Red" Width="30" Height="30" />
              </DataTemplate>
            </sap:WorkflowItemsPresenter.SpacerTemplate>
            <sap:WorkflowItemsPresenter.ItemsPanel>
              <ItemsPanelTemplate>
                <StackPanel Orientation="Horizontal"/>
              </ItemsPanelTemplate>
            </sap:WorkflowItemsPresenter.ItemsPanel>
          </sap:WorkflowItemsPresenter>
        </StackPanel>
      </Border>
      <Border BorderBrush="Black" BorderThickness="4" CornerRadius="5">
```

```
        <sap:WorkflowItemPresenter Item="{Binding Path=ModelItem.Post, Mode=TwoWay}"
HintText="Insert Post Activities Here"/>
      </Border>
    </StackPanel>
</sap:ActivityDesigner>
```

WorkflowItemPresenter と WorkflowItemsPresenter

このカスタム デザイナーでは、データ バインドを使用して PrePostSequence クラスのプロパティをバインドします。Pre プロパティと Post プロパティは 1 つのアクティビティなので、デザイナーでは WorkflowItemPresenter を使用してこれらのプロパティのデザイナー画面を有効にします。

Activities コレクションでは、WorkflowItemsPresenter を使用して、アクティビティのコレクションを保持できるデザイナー画面を作成します。

7. **Ctrl** キー、**Shift** キー、**B** キーを同時に押して、ソリューションをビルドします。

タスク 4 – アクティビティ デザイナーをアクティビティにリンクする

これで、このクラスのデザイナーが作成されましたが、このデザイナーはタスク 1 で作成したカスタム アクティビティにリンクしない限り使用できません。

1. ソリューション エクスプローラーで HelloWorkflow.Activities プロジェクトを右クリックし、[Add Reference] (参照の追加) をクリックします。
2. [Projects] (プロジェクト) タブで、**HelloWorkflow.Activities.Designers** への参照を追加します。
3. [Add Reference] (参照の追加) を再度選択して、[.NET] タブで次のアセンブリへの参照を追加します。
   - System.Activities.Presentation
   - PresentationFramework
   - PresentationCore
   - WindowsBase
4. **PrePostSequence.cs** (C#) ファイルまたは **PrePostSequence.vb** (VB) ファイルを開き、次の名前空間ディレクティブを追加します。

C#

```
using System.ComponentModel;
using HelloWorkflow.Activities.Designers;
```

Visual Basic

```
Imports System.ComponentModel
Imports HelloWorkflow.Activities.Designers
```

5. **PrePostSequence** クラスに次の属性を追加します。

C#

```
[Designer(typeof(PrePostSequenceDesigner))]
public sealed class PrePostSequence : NativeActivity
```

Visual Basic

```
<Designer(GetType(PrePostSequenceDesigner))>
Public NotInheritable Class PrePostSequence
```

次の手順

演習 9: 確認

演習 9: 確認

この確認では、演習のすべての手順を正しく実行したことを確認するテストを作成します。

1. **Ctrl** キー、**Shift** キー、**B** キーを同時に押して、ソリューションをビルドします。
2. **HelloWorkflow.Tests** プロジェクトには、カスタム アクティビティ プロジェクトへの参照が必要です。ソリューション エクスプローラーで、**HelloWorkflow** プロジェクトを右クリックし、[Add Reference] (参照の追加) をクリックします。[Project] (プロジェクト) タブで **HelloWorkflow.Activities** を選択します。
3. 次のように、**SayHelloFixture.cs** (C#) ファイルまたは **SayHelloFixture.vb** (VB) ファイルに新しいテストを追加します。

(Code Snippet - Introduction to WF4 Lab – ShouldreturnPrePostMessages Test CSharp)

C#

```
[TestMethod]
public void ShouldReturnPrePostMessages()
{
  IDictionary<string, object> output;
```

```
    output = WorkflowInvoker.Invoke(new SayHello()
    {
        UserName = "Test"
    });

    Assert.AreEqual("This is Pre-Sequence", output["PreMessage"]);
    Assert.AreEqual("This is Post-Sequence", output["PostMessage"]);
}
```

(Code Snippet - Introduction to WF4 Lab – ShouldreturnPrePostMessages Test VB)

Visual Basic

```
<TestMethod()>
Public Sub ShouldReturnPrePostMessages()
    Dim output = WorkflowInvoker.Invoke(
        New SayHello() With {.UserName = "Test"})

    Assert.AreEqual("This is Pre-Sequence", output("PreMessage"))
    Assert.AreEqual("This is Post-Sequence", output("PostMessage"))
End Sub
```

4. テスト名 (**ShouldReturnPrePostMessages**) を右クリックし、[Run Tests] (テストの実行) をクリックして、新しいテストを実行します。PrePostSequence をまだ実装していないので、テストは失敗します。
5. ソリューション エクスプローラーで **SayHello.xaml** ファイルをダブルクリックして開きます。
6. **Finally** アクティビティの下に **PrePostSequence** をドロップすると、新しい **PrePostSequence** アクティビティ デザイナーが表示されます。
7. **PrePostSequence** の Pre 領域と Post 領域からのメッセージを保持する 2 つの出力引数を新しく追加します。

Name	Direction	Argument type	Default value
UserName	In	String	*Enter a VB expression*
Greeting	Out	String	*Default value not supported*
WorkflowThread	Out	Int32	*Default value not supported*
PreMessage	Out	String	*Default value not supported*
PostMessage	Out	String	*Default value not supported*
Create Argument			

図 61

PreMessage 出力引数と PostMessage 出力引数の追加

8. [Primitives] (プリミティブ) グループの **Assign** アクティビティを Pre アクティビティの領域にドロップし、次のようにプロパティを設定します。
    a. To: **PreMessage**
    b. Value: **"This is Pre-Sequence"**
9. [Primitives] (プリミティブ) グループの **Assign** アクティビティを Post アクティビティの領域にドロップし、次のようにプロパティを設定します。
    a. To: **PostMessage**
    b. Value: **"This is Post-Sequence"**
10. **PrePostSequence** の下にある 2 つの Assign アクティビティをドラッグし、Activities 領域内にドロップします。これを行うには、アクティビティをドロップするときに、以下の図の赤い点にカーソルを動かします。

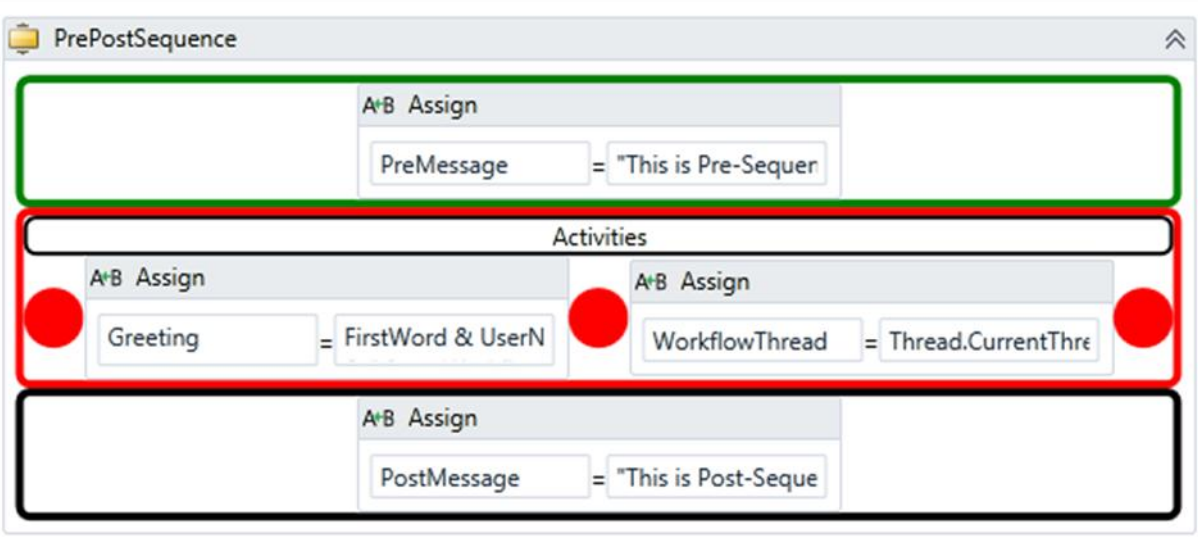

図 62

PrePostSequence を配置して完成したワークフロー

11. **Ctrl** キー、**Shift** キー、**B** キーを同時に押して、ソリューションをビルドします。
12. **Ctrl** キーを押しながら **F5** キーを押し、次に **A** キーを押して、すべてのテストを実行します。今度はすべてのテストが成功します。

次の手順

演習 10: ホストされたデザイナー

演習 10: ホストされたデザイナー

PrePostSequence などのカスタム アクティビティを作成して、Visual Studio を持っていない他のユーザーが使用できるようにするとしたら、どうでしょうか。多くの製品では、エンドユーザーがワークフローをカスタマイズできます。Windows Workflow Foundation 4 を使用すると、非常に簡単にアプリケーションでデザイナーをホストできます。この演習では、デザイナーをホストして、カスタム アクティビティを使用します。

タスク 0 – ソリューションを開く

この演習では、演習 9 で作成したソリューションを使用することができます。演習 9 を完了していない場合は、次の手順で演習 10 を開始することができます。

1. Microsoft Visual Studio 2010 を管理者として起動します。[スタート] ボタンをクリックし、[すべてのプログラム]、[Microsoft Visual Studio 2010] の順にクリックします。
2. **%TrainingKitInstallFolder%\Labs\IntroToWF\Source\Ex10-HostedDesigner\Begin** フォルダーには、C# と Visual Basic の開始ソリューションがあります。
3. **Ctrl** キー、**Shift** キー、**B** キーを同時に押して、ソリューションをビルドします。

タスク 1 – 新しい WPF アプリケーションを追加する

1. ソリューション エクスプローラーで **HelloWorkflow** ソリューションを右クリックし、[Add] (追加) をポイントして、[New Project] (新しいプロジェクト) をクリックします。
2. [Windows] テンプレートで **HelloDesigner** という新しい **WPF** アプリケーションを追加します。
3. **HelloDesigner** プロジェクトをスタートアップ プロジェクトに設定します。
4. [Add Reference] (参照の追加) をクリックします。

- System.Activities.Presentation
- System.Activities.Core.Presentation
- System.Activities

5. [Projects] (プロジェクト) タブで、次のアセンブリを追加します。
   - HelloWorkflow.Activities
   - HelloWorkflow.Activities.Designers
6. **MainWindow.xaml** を開き、次のように変更します。

(Code Snippet - Introduction to WF4 Lab – MainWindow XAML CSharp)

XAML (C#)

```
<Window x:Class="HelloDesigner.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="MainWindow" Height="600" Width="1000">
  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition Height="4*" />
      <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <Grid Name="grid1">
      <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto" />
        <ColumnDefinition Width="4*" />
        <ColumnDefinition Width="Auto" />
      </Grid.ColumnDefinitions>
    </Grid>
    <TextBox Grid.Row="1" Name="textXAML"
         VerticalScrollBarVisibility="Visible" />
  </Grid>
</Window>
```

(Code Snippet - Introduction to WF4 Lab – MainWindow XAML VB)

XAML (VB)

```
<Window x:Class="MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="MainWindow" Height="600" Width="1000">
  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition Height="4*" />
      <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <Grid Name="grid1">
      <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto" />
        <ColumnDefinition Width="4*" />
        <ColumnDefinition Width="Auto" />
      </Grid.ColumnDefinitions>
    </Grid>
```

```
    <TextBox Grid.Row="1" Name="textXAML"
             VerticalScrollBarVisibility="Visible" />
  </Grid>
</Window>
```

7. **MainWindow.xaml.cs** (C#) または **MainWindow.xaml.vb** を開きます。

8. 次の名前空間ディレクティブを追加します。

(Code Snippet - Introduction to WF4 Lab – MainWindow Namespaces CSharp)

C#

```
using System.Activities.Presentation;
using System.Activities.Statements;
using System.Activities.Presentation.Toolbox;
using System.Activities.Core.Presentation;
using System.Activities.Presentation.Metadata;
using System.ComponentModel;
using HelloWorkflow.Activities;
using HelloWorkflow.Activities.Designers;
```

(Code Snippet - Introduction to WF4 Lab – MainWindow Namespaces VB)

Visual Basic

```
Imports System.Activities.Presentation
Imports System.Activities.Statements
Imports System.Activities.Presentation.Toolbox
Imports System.Activities.Core.Presentation
Imports System.Activities.Presentation.Metadata
Imports System.ComponentModel
Imports HelloWorkflow.Activities
Imports HelloWorkflow.Activities.Designers
```

9. 次のように **WorkflowDesigner** 型のフィールドを追加します。

C#

```
public partial class MainWindow : Window
{
    WorkflowDesigner workflowDesigner = new WorkflowDesigner();
```

Visual Basic

```
Class MainWindow
    Private workflowDesigner As WorkflowDesigner = New WorkflowDesigner()
```

10. 次のように、**RegisterMetadata** という新しい関数を作成します。

(Code Snippet - Introduction to WF4 Lab – RegisterMetadata method CSharp)

C#

```
private void RegisterMetadata()
{
   DesignerMetadata metaData = new DesignerMetadata();
   metaData.Register();
   AttributeTableBuilder builder = new AttributeTableBuilder();
   MetadataStore.AddAttributeTable(builder.CreateTable());
}
```

(Code Snippet - Introduction to WF4 Lab – RegisterMetadata method VB)

Visual Basic

```
Public Sub RegisterMetadata()
   Dim metaData As DesignerMetadata = New DesignerMetadata()
   metaData.Register()
   Dim builder As AttributeTableBuilder = New AttributeTableBuilder()
   MetadataStore.AddAttributeTable(builder.CreateTable())
End Sub
```

メモ: この関数では、デザイナーのメタデータ ストアを有効にします。

デザイナーをホストすると、ツールボックスを制御できます。表示されるコントロール、コントロールが表示されるカテゴリ、およびコントロール名さえも選択できます。

(Code Snippet - Introduction to WF4 Lab – CreateToolboxControl method CSharp)

C#

```
private ToolboxControl CreateToolboxControl()
{
   //ToolBoxControl を作成します
   ToolboxControl ctrl = new ToolboxControl();

   //カテゴリ項目のコレクションを作成します
   ToolboxCategory category = new ToolboxCategory("Hello Workflow");

   //toolboxItems を作成します
   ToolboxItemWrapper tool0 = new ToolboxItemWrapper(
      "System.Activities.Statements.Assign",
      "System.Activities, Version=4.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35"
      , null,
      "Assign");
   ToolboxItemWrapper tool1 = new ToolboxItemWrapper(
      "System.Activities.Statements.Sequence",
      "System.Activities, Version=4.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35",
      null,
      "Sequence");
   ToolboxItemWrapper tool2 = new ToolboxItemWrapper(
      "System.Activities.Statements.TryCatch",
      "System.Activities, Version=4.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35",
```

```
        null,
        "Try It"); // 別の名前を使用できます
    ToolboxItemWrapper tool3 = new ToolboxItemWrapper(
        "HelloWorkflow.Activities.PrePostSequence",
        "HelloWorkflow.Activities",
        null,
        "PrePostSequence");

    //toolboxItems をカテゴリに追加します
    category.Add(tool0);
    category.Add(tool1);
    category.Add(tool2);
    category.Add(tool3);

    //カテゴリを ToolBox コントロールに追加します
    ctrl.Categories.Add(category);
    return ctrl;
}
```

(Code Snippet - Introduction to WF4 Lab – CreateToolboxControl method VB)

Visual Basic

```
Private Function CreateToolboxControl() As ToolboxControl
    'ToolBoxControl を作成します
    Dim ctrl As ToolboxControl = New ToolboxControl()

    'カテゴリ項目のコレクションを作成します
    Dim category As ToolboxCategory = New ToolboxCategory("Hello Workflow")

    'toolboxItems を作成します
    Dim tool0 As ToolboxItemWrapper = New ToolboxItemWrapper(
        "System.Activities.Statements.Assign",
        "System.Activities, Version=4.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35",
        Nothing,
        "Assign")
    Dim tool1 As ToolboxItemWrapper = New ToolboxItemWrapper(
        "System.Activities.Statements.Sequence",
        "System.Activities, Version=4.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35",
        Nothing,
        "Sequence")
    Dim tool2 As ToolboxItemWrapper = New ToolboxItemWrapper(
        "System.Activities.Statements.TryCatch",
        "System.Activities, Version=4.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35",
        Nothing,
        "Try It") ' 別の名前を使用できます
    Dim tool3 As ToolboxItemWrapper = New ToolboxItemWrapper(
        "HelloWorkflow.Activities.PrePostSequence",
        "HelloWorkflow.Activities",
        Nothing,
        "PrePostSequence")
```

```
    'toolboxItems をカテゴリに追加します
    category.Add(tool0)
    category.Add(tool1)
    category.Add(tool2)
    category.Add(tool3)

    'カテゴリを ToolBox コントロールに追加します
    ctrl.Categories.Add(category)
    Return ctrl
End Function
```

11. デザイナーをウィンドウに追加する、**AddDesigner** という関数を追加します。

(Code Snippet - Introduction to WF4 Lab – AddDesigner method CSharp)

```
C#
private void AddDesigner()
{
    //WorkflowDesigner クラスのインスタンスを作成します
    this.workflowDesigner = new WorkflowDesigner();

    //グリッドの中央の列に WorkflowDesigner を配置します
    Grid.SetColumn(this.workflowDesigner.View, 1);

    // モデルが変更されたときにワークフローをフラッシュします
    workflowDesigner.ModelChanged += (s, e) =>
    {
        workflowDesigner.Flush();
        textXAML.Text = workflowDesigner.Text;
    };

    //既定値として新しい Sequence を読み込みます
    this.workflowDesigner.Load(new Sequence());

    //WorkflowDesigner をグリッドに追加します
    grid1.Children.Add(this.workflowDesigner.View);

    // プロパティ インスペクターを追加します
    Grid.SetColumn(workflowDesigner.PropertyInspectorView, 2);
    grid1.Children.Add(workflowDesigner.PropertyInspectorView);

    // ツールボックスを追加します
    ToolboxControl tc = CreateToolboxControl();
    Grid.SetColumn(tc, 0);
    grid1.Children.Add(tc);
}
```

(Code Snippet - Introduction to WF4 Lab – AddDesigner method VB)

```
Visual Basic
Private Sub AddDesigner()

    'WorkflowDesigner クラスのインスタンスを作成します
    workflowDesigner = New WorkflowDesigner()

    'グリッドの中央の列に WorkflowDesigner を配置します
    Grid.SetColumn(workflowDesigner.View, 1)

    ' ModelChanged イベント ハンドラーを設定します
    AddHandler workflowDesigner.ModelChanged,
      Function(sender As Object, e As System.EventArgs)
        ' モデルが変更されたときにワークフローをフラッシュします
        workflowDesigner.Flush()
        textXAML.Text = workflowDesigner.Text
        Return Nothing
      End Function

    '既定値として新しい Sequence を読み込みます
    workflowDesigner.Load(New Sequence())

    'WorkflowDesigner をグリッドに追加します
    grid1.Children.Add(workflowDesigner.View)

    ' プロパティ インスペクターを追加します
    Grid.SetColumn(workflowDesigner.PropertyInspectorView, 2)
    grid1.Children.Add(workflowDesigner.PropertyInspectorView)

    ' ツールボックスを追加します
    Dim tc As ToolboxControl = CreateToolboxControl()
    Grid.SetColumn(tc, 0)
    grid1.Children.Add(tc)
End Sub
```

12. MainWindow のコンストラクターを変更して、この演習で追加した関数を呼び出します。

(Code Snippet - Introduction to WF4 Lab – MainWindow method CSharp)

```
C#
public MainWindow()
{
    InitializeComponent();
    RegisterMetadata();
    AddDesigner();
}
```

(Code Snippet - Introduction to WF4 Lab – MainWindow method VB)

```
Visual Basic
Public Sub New()
    ' この呼び出しはデザイナーが必要とします
    InitializeComponent()

    ' InitializeComponent() の呼び出しの後に任意の初期化を追加します
    RegisterMetadata()
    AddDesigner()
End Sub
```

次の手順

演習 10: 確認

演習 10: 確認

すべての手順を正しく実行したことを確認するために、**HelloDesigner** アプリケーションを実行し、ワークフローを変更します。

1. **HelloDesigner** がスタートアップ プロジェクトに設定されていることを確認します。
2. **F5** キーを押して、デバッグ モードでアプリケーションを起動します。
3. デザイン ウィンドウが表示されたら、**PrePostSequence** をデザイナー画面にドロップします。
4. 次の出力結果が表示されます。

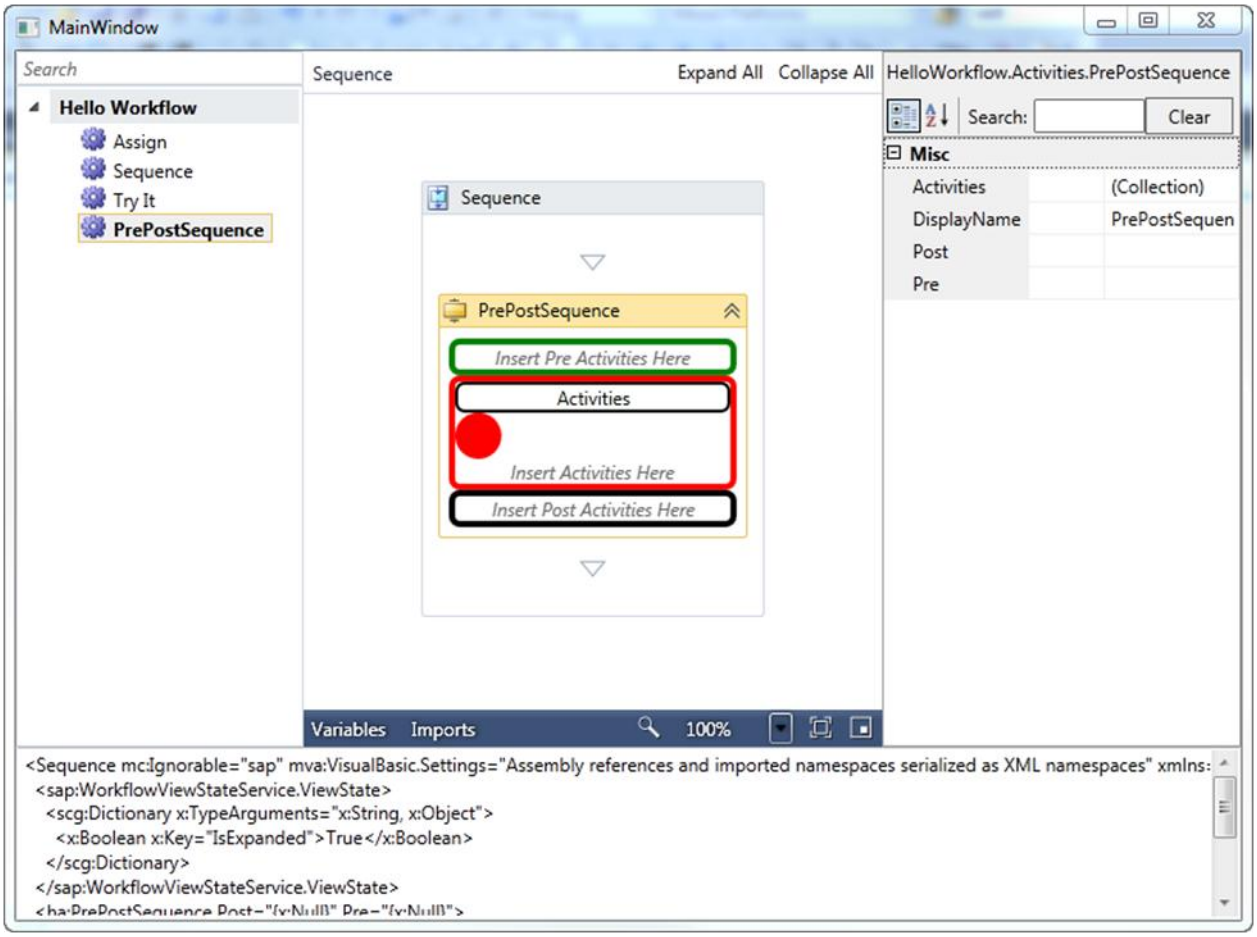

図 63

WorkflowDesigner をホストする HelloDesigner アプリケーション

次の手順

まとめ

まとめ

このハンズオン ラボでは、簡単な Windows Workflow アプリケーションを作成して、Windows Workflow Foundation 4 の基本について学習しました。ここでは、新しい Visual Studio 2010 のワークフロー デザイナーを使用してワークフローを作成する方法を学びました。また、その代替手段として、C# または VB コードを使用してワークフローを作成する方法についても学びました。入力引数と出力引数の使い方を調べ、If アクティビティが備わったワークフローに if/else ロジックを追加する方法を理解しました。最後に例外処理専用のア

クティビティ (TryCatch アクティビティ、Catch<T> アクティビティ、Throw アクティビティなど) を使用してワークフローのエラー状態をハンドルする方法について学習しました。もちろん、Windows Workflow 4 の機能は他にも多数あります。ワークフロー対応アプリケーションを構築する方法の詳細については、フローチャートおよびワークフロー サービスについてのラボを参照することをお勧めします。

フィードバック

このラボおよび新しい Windows Workflow 4 に関する皆様のご意見、ご感想をお寄せください。皆様からフィードバックを参考に、最高の製品に仕上げます。製品の提供まで、もうしばらくお待ちください。皆様のご意見、ご感想は **wfwcfhol@microsoft.com** まで英語でお寄せください。