

SMARTBEAR



TestLeft

開発者向けの機能テスト ツール

クイックスタートガイド

TestLeft 1.0



エクセルソフト株式会社

作成：2016.5.16

目次

目次.....	2
はじめに	3
チュートリアル	4
TestLeft - 基本概念	5
テストの作成	5
テストの実行	6
Web サービスについて.....	7
1. Visual Studio プロジェクトの作成.....	9
2. テスト コードの記述	13
3. テストの実行とテスト結果の表示	20
次のステップ	22
お問合せ先.....	23

はじめに

このガイドブックは、TestLeft (v1.0) ユーザーガイドのクイック スタート チュートリアルに関して説明した部分を抜粋して翻訳したものです。サンプル コードは、C# 部分のみ記載していますので、Visual Basic .NET のコードを参照する場合は、英語のユーザーガイドをご参照ください。

評価版のインストール ファイルの入手は、弊社 Web サイトからお申込みください。

<http://www.xlsoft.com/jp/products/smartbear/testleft.html>

各機能の詳細を説明する最新バージョンの TestLeft ユーザーガイドは、SmartBear 社の下記のサイトで参照できます。

<https://support.smartbear.com/onlinehelp/>

ビデオによるクイックスタートガイドの操作方法も用意していますので、ご覧ください。

<https://support.smartbear.com/viewarticle/79457/>

© 2016 SmartBear Software. All rights reserved.

Translated by XLsoft Corporation

チュートリアル

このチュートリアルは、Visual Studio で TestLeft のテストを作成する手順を学習するのに役立ちます。

このチュートリアルでは、TestLeft コンポーネントを説明し、さらにそれらがどのように相互に働くかを説明します。チュートリアルでは以下の操作を行います:

- TestLeft テスト用に Visual Studio テスト プロジェクトを作成し、設定します。
- テストするアプリケーションでオブジェクト階層を参照します。
- TestLeft テストからテストするアプリケーションのオブジェクトにアクセスし、それらを使ったユーザー アクションをシミュレートします。
- テストを実行し、テスト結果を確認します。

テスト対象のアプリケーションとして、Windows 付属のメモ帳 (Notepad) アプリケーションを使用します。

TestLeft - 基本概念

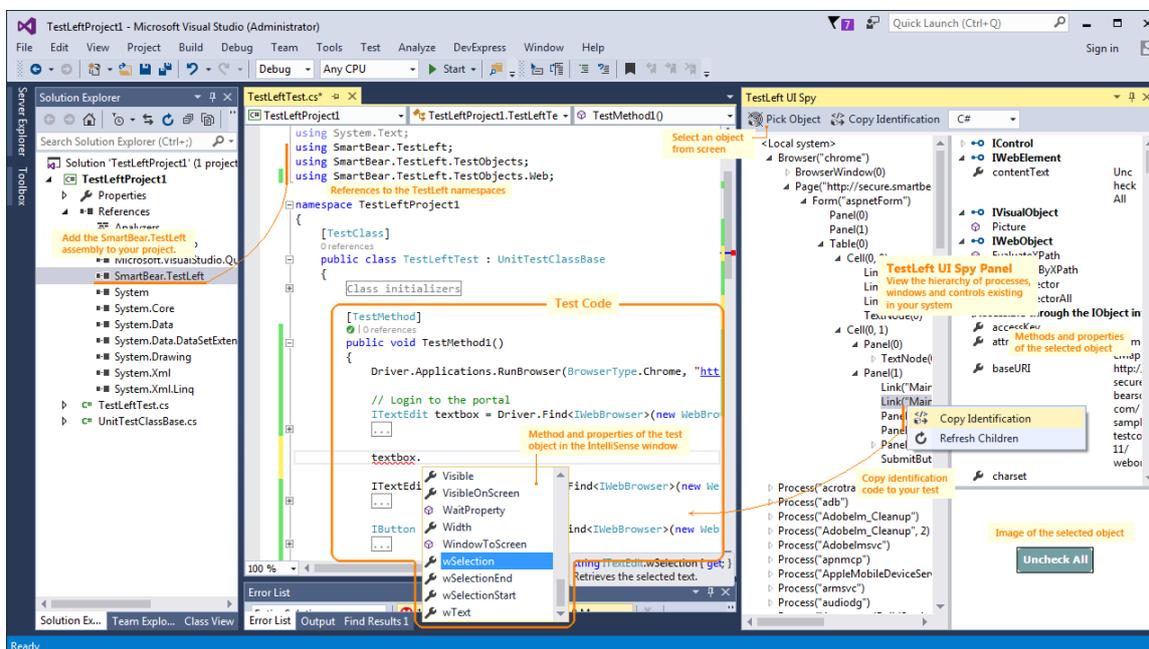
SmartBear TestLeft は、Windows デスクトップおよび Web アプリケーションの機能テストと UI テストを作成および実行するための自動テスト フレームワークです。C#、Visual Basic .NET、その他の任意の .NET 言語でテストを作成することができ、Jenkins のような CI システムや、Visual Studio テスト エクスプローラー、あるいは他の任意の方法でそれらを実行できます。

TestLeft は、TestComplete エンジンで構築され、TestComplete によりサポートされているアプリケーションとテクノロジーの大半をサポートします。

テストの作成

Visual Studio 2013 または 2015 で TestLeft のテストを作成します。TestLeft プロジェクトは、.NET Framework 4.5 またはそれ以降をターゲットにしている必要があります。

TestLeft のインストーラーが Visual Studio に追加する **TestLeft UI Spy** パネルからオブジェクト、メソッド、プロパティを見ることができます。テストコードは、プロジェクトに含まれる **SmartBear.TestLeft.dll** アセンブリにより提供されるインターフェース、クラス、メソッドを使用して記述します。



このパネルとアセンブリに加えて、TestLeft のインストーラーは、MSTest と NUnit 互換のテスト プロジェクトの作成用に定義済みのテンプレートを追加し、Visual Studio を拡張します。これらのテンプレートは、TestLeft のテスト作成をさらに簡単にします。

プロジェクトで以下のことが自動的に取得できます —

- `SmartBear.TestLeft` ライブラリーへの参照。
- ソースファイルの最初で `SmartBear.TestLeft` 名前空間へのリンク。
- 自動的に作成される `Driver` オブジェクト (これは、テスト ランナーと他のテスト関連のアクションの実行でデータ交換のために使用される中核のオブジェクトです)。
- カスタム メッセージを受け取るために自動的に作成される `Log` オブジェクト。

プロジェクト テンプレートに加えて、`TestLeft` インストーラーは、個別の `MSTest` と `JUnit` ベースの `TestLeft` テストを作成するためのテンプレートを追加します。

テンプレートに基づいたプロジェクトをコンパイルした後に、既に社内で使用している `MSTest` または `JUnit` 単体テストフレームワークで実行できるモジュールが得られます。

テスト プロジェクトのために、これらのテンプレートを使用することを推奨しますが、コンソール アプリケーションまたは通常の `WinForm` アプリケーションなど、任意のアプリケーションに `TestLeft` コードを配置することができます。この場合、`TestLeft` テストを作成するための必要なすべてのアクションを手動で実行する必要があります。

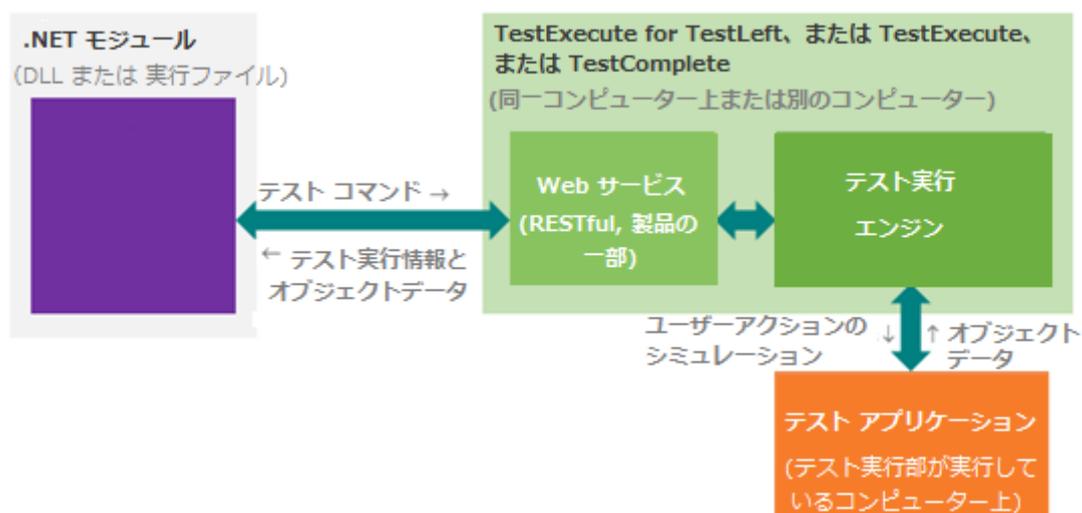
`Visual Studio` でテストコードを記述した後で、それを `DLL` または 実行ファイルにコンパイルします。

テストの実行

テスト コードをコンパイルした後に、任意の方法でそれを実行できます:

- `Visual Studio` テストエクスプローラー
- `Jenkins` または他の任意のビルド自動ツール
- コマンドライン (実行ファイルとしてコードをコンパイルした場合)
- `MSTest` または `Unit` テストランナー
- `TestComplete`
- または任意の他の方法から実行できます。

テストを実行するとき、テスト ランタイム エンジンにユーザーアクションを順番にシミュレートするコマンドを送ります:



テストは、利用中のコンピューターまたは TestLeft がインストールされたネットワーク上の他の任意のコンピューターに存在するランタイム エンジンにコマンドを送ることができます。さらに、テストコードはリモートの TestExecute および TestComplete (これらの最新バージョンは、TestLeft からのコマンドを受け付けます) のインスタンスにコマンドを送ることができます。

Web サービスについて

TestLeft は、RESTful Web サービスを通してテスト ランナーと相互作用します。この Web サービスは、TestLeft の一部であり、TestLeft をインストールすると、自動的にインストールされます。このサービスは、TestExecute と TestComplete の最新バージョンにも含まれています。

サービスは、コンピューター上に存在するアプリケーション、ウィンドウ、コントロールを探索するために必要となります。TestLeft UI Spy パネルで表示されるオブジェクト、メソッド、プロパティ上の情報は、このサービスによって提供されます。

デフォルトでは、このサービスはポート番号 2377 上で動作します (UI Spy は、サービスがこのポートを使用することを意味します)。このポート番号は、変更可能ですが、この場合、UI Spy はシステム上でオブジェクトを認識できません。

テストを作成するための準備

1. TestLeft テストを作成し実行するには、Visual Studio の次のいずれかのバージョンがインストールされている必要があります:

- Microsoft Visual Studio 2015 (Community、 Professional または Enterprise)
- Microsoft Visual Studio 2013 (Professional、 Premium または Ultimate)

TestLeft テストは、Visual Studio IDE 上でのみ作成できます。

2. コンピューターに TestLeft をインストールします。TestLeft のインストール方法については、ヘルプの [Installing TestLeft](#) をご参照ください。
3. Visual Studio で、**TestLeft UI Spy** パネルを開きます。

UI Spy は、TestLeft に含まれている機能です。テストするアプリケーションのオブジェクトの階層を確認でき、テスト用にオブジェクト識別コードを生成するために、これを使用します。

パネルを開くには、Visual Studio のメイン メニューから **[表示]-[その他のウィンドウ]-[TestLeft UI Spy]** を選択します。

必要に応じて、Visual Studio IDE でパネルをドラッグアンドドロップします。

4. 今回のテスト対象となる**メモ帳 (Notepad)** アプリケーションを実行します。このアプリケーションは、サポートしているすべての Windows オペレーティング システムに含まれています。

1. Visual Studio プロジェクトの作成

C# および Visual Basic .NET 言語を使用して、テスト コードを記述できます。また、テスト コードを Visual Studio でサポートしている任意のタイプのプロジェクトに追加できます。

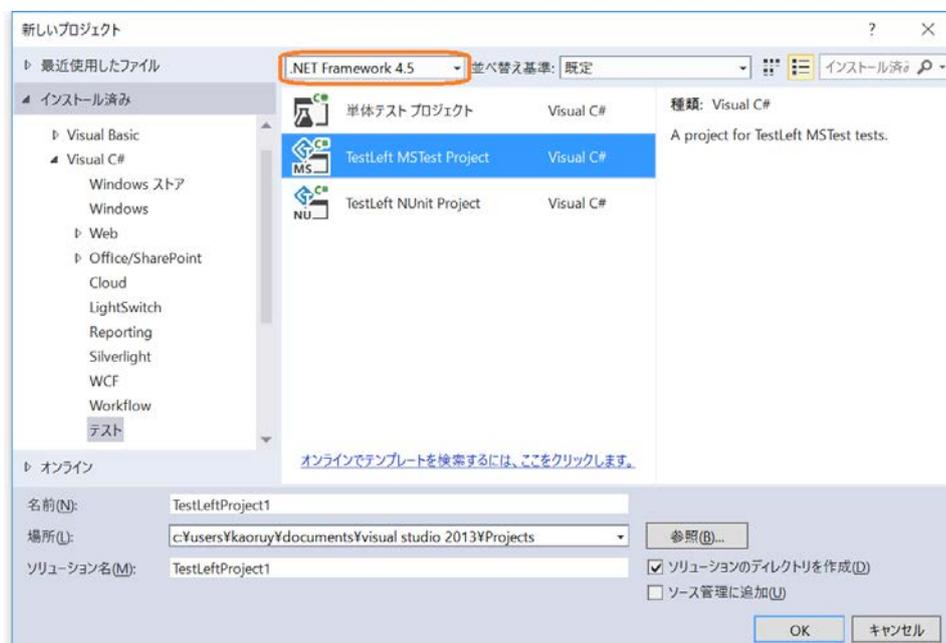
さらに、TestLeft は Visual Studio でテストを作成するためのテンプレートを装備しています。MSTest および NUnit 単体テスト フレームワークを使用することで、これらのテンプレート経由で作成したプロジェクトを実行できます。このように、すでに社内で使用しているテスト フレームワークを使用して、TestLeft のテストを作成し、実行することができます。

このチュートリアルでは、**TestLeft MSTest Project** テンプレートに基づくテスト プロジェクトを作成します:

プロジェクトの作成

既存の Visual Studio ソリューションに新規のプロジェクトを追加するか、新規プロジェクトを作成します。

1. [新しいプロジェクト] ダイアログで、.NET Framework version 4.5 またはそれ以降がターゲットになっているか確認します。

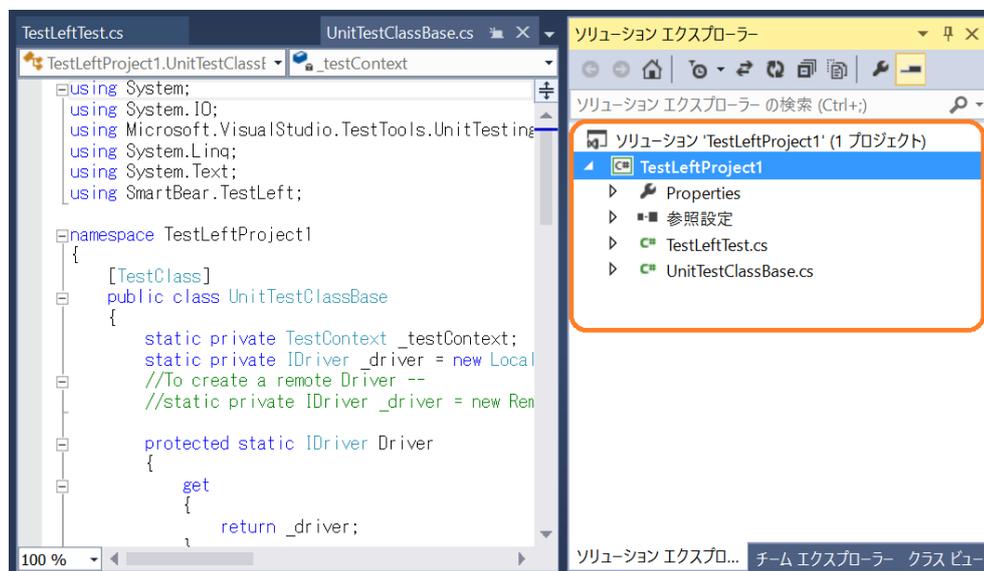


プロジェクト テンプレートの一覧から、[テスト] カテゴリーを選択し、次に [TestLeft MSTest Project] テンプレートを選択します。

必要に応じて、プロジェクト名と場所を指定します。

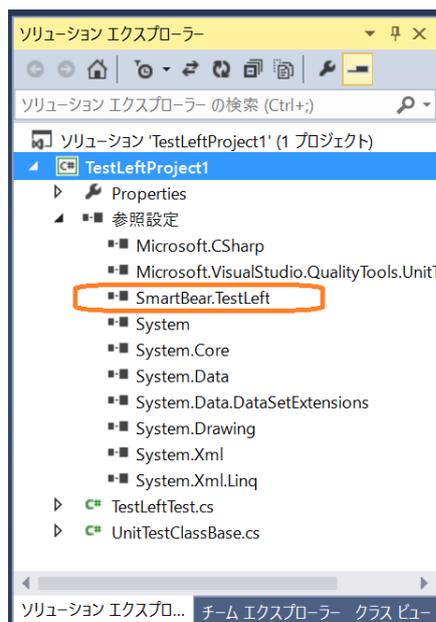
[OK] をクリックします。

2. Visual Studio が新規の TestLeft MSTest プロジェクトを作成し、ソリューションに追加します:



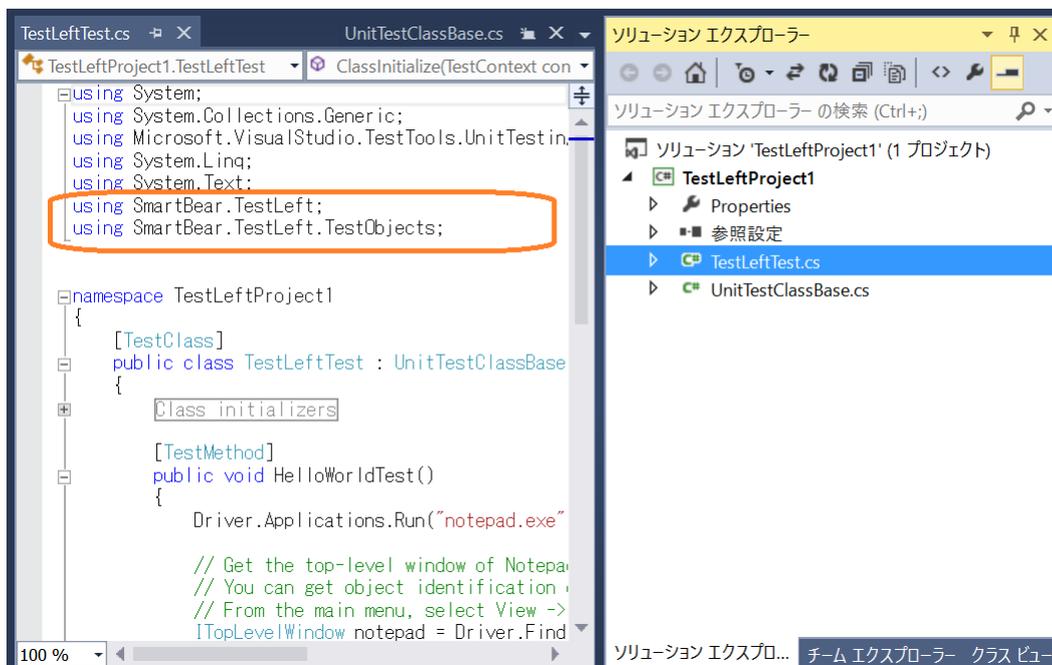
プロジェクトの設定

1. TestLeft でテストを作成するには、Visual Studio プロジェクトに追加された **SmartBear.TestLeft.dll** ライブラリーへの参照設定を行う必要があります。ライブラリーは、TestLeft に付属しており、<TestLeft>¥API¥dotNET¥ フォルダーにあります。
このプロジェクトは、TestLeft MSTest プロジェクト テンプレートをベースにしており、デフォルトで必要な参照がされています。



スクラッチからプロジェクトを作成する場合、参照を手動で追加してください。

- この `SmartBear.TestLeft.dll` ライブラリーは、サポートするすべてのアプリケーション テクノロジーのための基本のテストクラスとインターフェースを提供する名前空間を宣言します。テスト コードを記述するソースファイルで、これらの名前空間がファイルの名前スコープで利用可能になります。このチュートリアルでは、`SmartBear.TestLeft` と `SmartBear.TestLeft.TestObjects` 名前空間が必要になります。`TestLeftTest` ソースファイルを使用する場合、これらはデフォルトで設定されます。



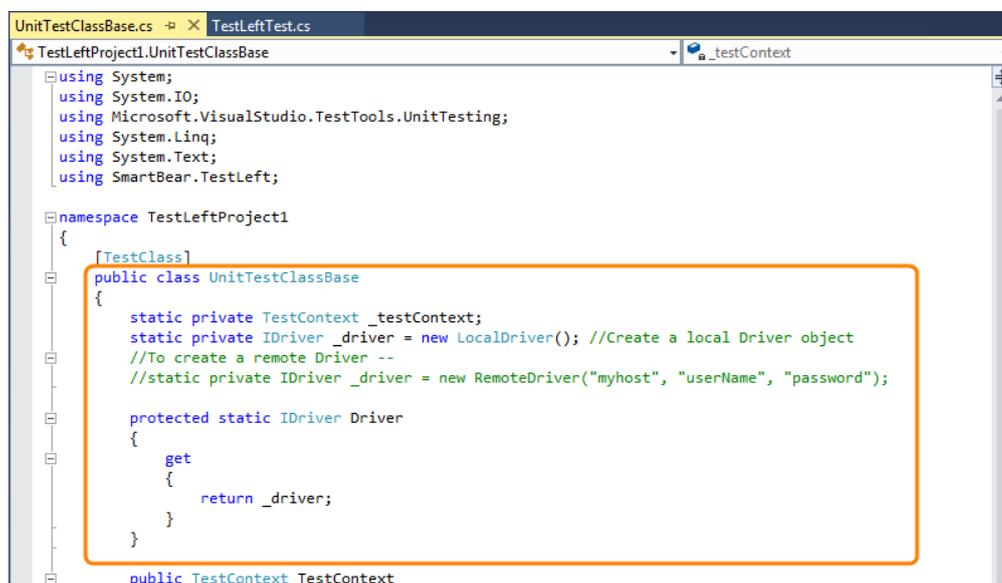
テストモジュールをスクラッチから作成する場合、またはテストで他の `SmartBear.TestLeft` 名前空間が必要な場合、手動で利用可能にしてください。

3. `TestLeft` プロジェクト テンプレートを使用して作成したプロジェクトには、 サンプル `HelloWorldTest` テスト メソッドがあります。このチュートリアルでは不要なため、削除して、スクラッチから新しいテスト メソッドを作成してみましょう。

2. テスト コードの記述

テスト コードを作成するには、`SmartBear.TestLeft.IDriver` インターフェースを使用します。このインターフェースは、TestLeft エンジンへのアクセスを提供します。また、テストするアプリケーションでオブジェクト階層へのアクセスも提供し、様々なテスト関連の操作を実行することができます。このインターフェースの詳細については、[About Driver Objects](#) をご参照ください。

このプロジェクトは、TestLeft MSTest プロジェクト テンプレートから作成されました。テストを作成するために使用するプロジェクトの `UnitTestClassBase` クラスは、必要となるインターフェースを返す `Driver` プロパティを持っています:



```
UnitTestClassBase.cs → TestLeftTest.cs
TestLeftProject1\UnitTestClassBase
- _testContext
using System;
using System.IO;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using System.Linq;
using System.Text;
using SmartBear.TestLeft;

namespace TestLeftProject1
{
    [TestClass]
    public class UnitTestClassBase
    {
        static private TestContext _testContext;
        static private IDriver _driver = new LocalDriver(); //Create a local Driver object
        //To create a remote Driver --
        //static private IDriver _driver = new RemoteDriver("myhost", "userName", "password");

        protected static IDriver Driver
        {
            get
            {
                return _driver;
            }
        }
    }
}

public TestContext TestContext
```

インターフェースを取得するために、`LocalDriver` クラスを使用します。今回はローカル コンピューター上の TestLeft テスト エンジンにアクセスします。

リモート コンピューター上で実行している TestLeft テスト エンジンにアクセスするには、`RemoteDriver` クラスを使用します。

TestLeft テンプレートを使用しないで、スクラッチからテストを作成する場合、コードでインターフェースを手動で取得してください。

ユーザー アクションのシミュレーション

1. このチュートリアルでは、メモ帳 (Notepad) アプリケーションで作業します。テストの最初に メモ帳を実行するコードを作成しましょう:

C#:

```
Driver.Applications.Run("notepad.exe");
```

メモ帳を実行するため、`IDriver.Applications.Run` メソッドを使用します。テストから対象のアプリケーションを実行することに関する詳細は、[Running Applications From Tests](#) をご参照ください。

2. メモ帳のウィンドウにアクセスし、そこにテキスト入力をシミュレートするコードを作成します:

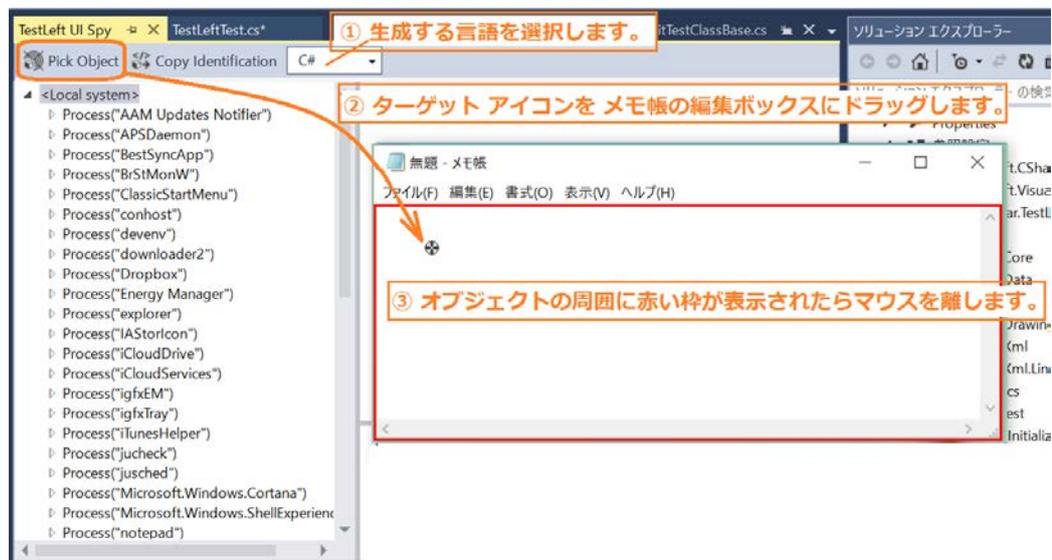
オブジェクトを取得するには

この `Idriver` インターフェースは、テストするアプリケーション内のオブジェクトへのアクセスを取得するために使用できる多数のメソッドを持っています。しかし、必要なオブジェクトを取得するコードを作成するための最も簡単な方法は、[TestLeft UI Spy](#) を使用することです:

- **UI Spy** パネルが隠れている場合、パネルを表示するため、**Visual Studio** のメインメニューから **[表示] - [その他のウィンドウ] - [TestLeft UI Spy]** を選択します。

このパネルは、コンピュータ上で実行しているすべてのアプリケーションの階層と、それらのウィンドウとコントロールの階層を表示します。

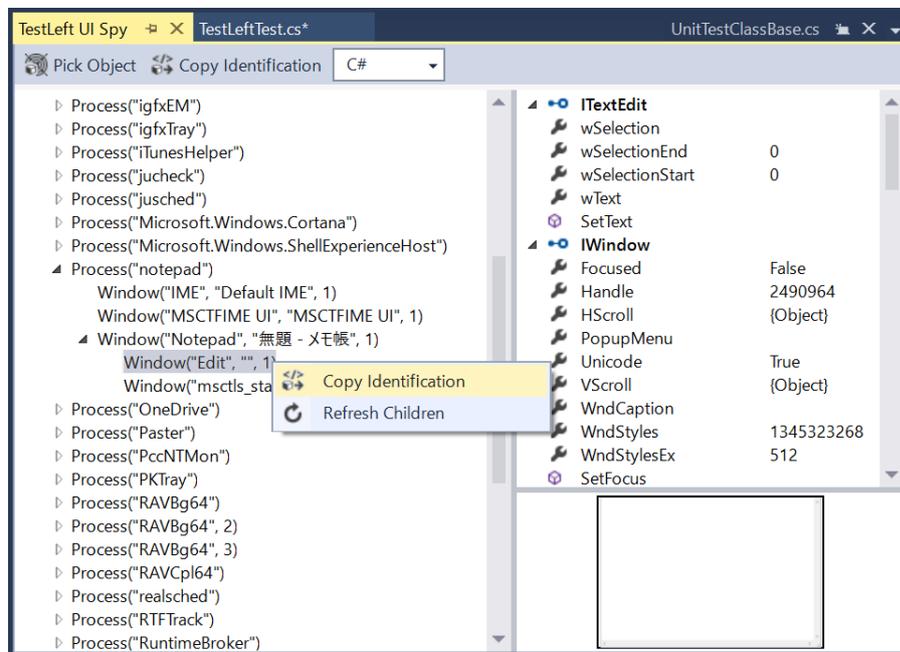
- 画面上にあるメモ帳の編集ボックスを **[Pick Object]** ツールで選択します:



TestLeft UI Spy がそのコントロールをキャプチャーし、オブジェクト ツリーでそれを選択します。

注意: TestLeft が必要なオブジェクトをキャプチャーし、オブジェクトの周りに赤い枠を表示するのに数秒かかることがあります。その場合は少しお待ちください。

- オブジェクトツリーでオブジェクトを右クリックし、 **Copy Identification** をクリックします。



TestLeft UI Spy が、必要なオブジェクトにアクセスするためのコードを生成し、クリップボードにコードをコピーします。

- クリップボードからのコードをテスト メソッドに挿入します:

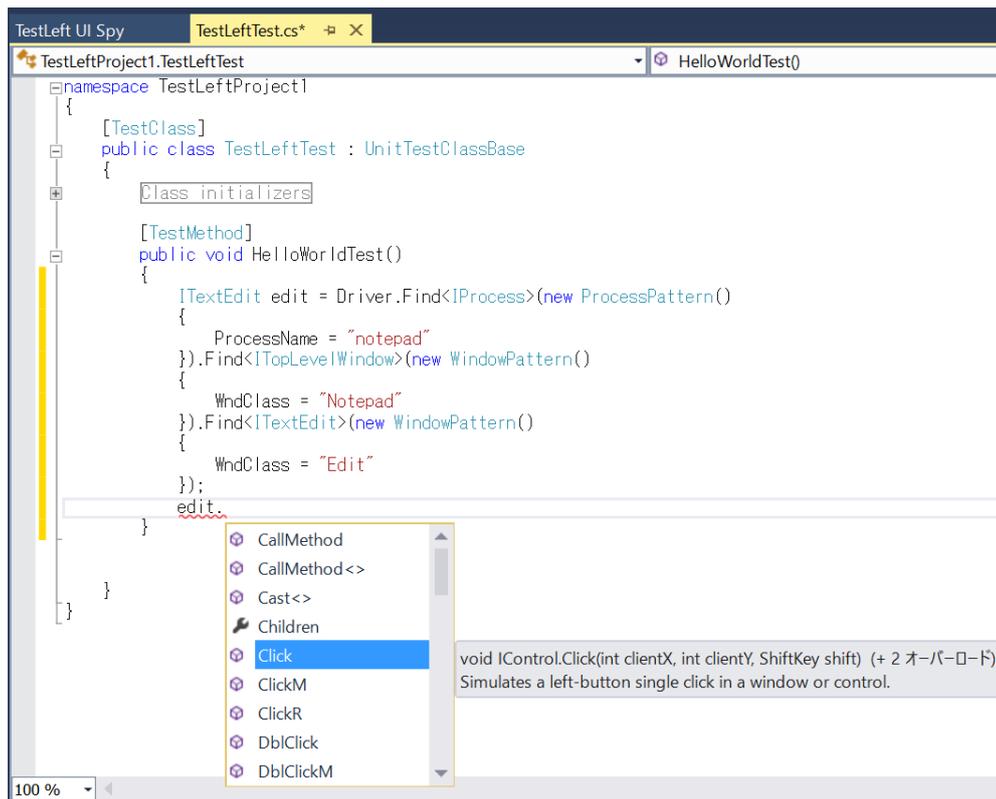
```
C#:
ITextEdit edit = Driver.Find<IProcess>(new ProcessPattern()
{
    ProcessName = "notepad"
}).Find<ITopLevelWindow>(new WindowPattern()
{
    WndClass = "Notepad"
}).Find<ITextEdit>(new WindowPattern()
{
    WndClass = "Edit"
});
```

TestLeft は、テストするアプリケーションのウィンドウまたはコントロールを適切なインターフェースにキャストし、そのウィンドウまたはコントロール上のユーザーアクションをシミュレートするために、インターフェースのメソッドとプロパティを使用します。オブジェクトで作業するために TestLeft が使用するインターフェースは、オブジェクト クラスに依存します。

このチュートリアルでは、TestLeft は、メモ帳の編集ボックスを `ITextEdit` インターフェイスとして認識します。

オブジェクト上でユーザーアクションをシミュレートするには

コントロール上のユーザーアクションをシミュレートするために、オブジェクトのプロパティとメソッドを使用することができます。Visual Studio のインテリセンスで利用可能なプロパティとメソッドの一覧が表示されます。



メモ帳のテキスト エディターでマウス クリックをシミュレートするには、オブジェクトの `Click` メソッドをコールします。テキスト入力をシミュレートするには、オブジェクトの `SetText` メソッドをコールします。

C#:

```
edit.Click();

string inputText = "test";

edit.SetText(inputText);
```

オブジェクト状態のチェック

テストでは、アプリケーションがテスト下で正しく機能しているか確認するために、オブジェクトの状態をチェックすることが必要になることがあります。TestLeft には、期待する値とオブジェクトのプロパティを比較するコードを作成する機能があります。

このチュートリアルでは、テキスト入力が正しくシミュレートされたかをチェックするコードを作成します。期待されるテキストに対して、メモ帳のテキストエディターが含んでいる実際のテキストの情報 (wText プロパティで返される) を比較します。

```
Assert.AreEqual(inputText, edit.wText);
```

アサーションが失敗した場合、テストは停止し、それ以降のすべての操作が実行されません。

テスト結果の保存

TestLeft テストが実行しているとき、TestLeft は、テストするアプリケーションでシミュレートする (マウス クリック、ボタン押下など) すべてのテスト アクションの情報をテスト ログに保存します。テストが失敗した場合、テスト ログを参照し、テスト実行中に何が起こったか、何がテスト失敗の原因かを知ることができます。

TestLeft のテストログにカスタム メッセージをポストすることができます。メモ帳の編集ボックスのスクリーンショットとカスタム警告メッセージを TestLeft のテスト ログにポストしてみましょう:

C#:

```
Driver.Log.Screenshot(edit, "Notepad's edit box screenshot");  
Driver.Log.Warning("A warning message");
```

デフォルトで、TestLeft は一時フォルダーにそのログを格納し、テスト実行が終了したあとにそれを削除します。テスト結果をテスト実行後に表示し、分析するために外部のファイル (MHT または HTML) にエクスポートすることを TestLeft に指示することができます。

注意: TestLeft MSTest プロジェクト テンプレートは、TestLeft ログを自動的にデフォルトの MSTest 結果フォルダーにエクスポートするメソッドを持っています。

テストがパスした場合、MSTest はこのフォルダーをクリアします。このテスト結果は、テストが失敗したときにのみ、見ることができます。

テストがパスしたときこのテスト結果を表示するには、次のいずれかをしてください:

- テスト ログを別の場所にエクスポートします (このチュートリアルで説明するアプローチです)。
— または —
- MSTest がテスト結果をクリアしないようにプロジェクトでテスト設定を指定します。

TestLeft のテスト結果を表示するために、現在のユーザー フォルダーにテスト ログを保管するように命令をテスト メソッドに追加します:

C#:

```
using System.IO;
...

string logPath =
Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments),
DateTime.Now.ToString("MM_dd_yyyy_H_mm_ss"));

Driver.Log.Save(logPath, Log.Format.Html);
```

TestLeft テスト ログにカスタム メッセージをポストしたり、ログをエクスポートすることに関する詳細な情報は、[Working With TestLeft Test Logs](#) セクションをご参照ください。

最終コード

全体的な TestLeftTest クラス コードは、次のようになります:

C#:

```
using System;
using System.Collections.Generic;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using System.Linq;
using System.Text;
using SmartBear.TestLeft;
using SmartBear.TestLeft.TestObjects;
using System.IO;

namespace TestLeftProject1
```

```
{
  [TestClass]
  public class TestLeftTest : UnitTestClassBase
  {
    [ClassInitialize]
    public static void ClassInitialize(TestContext context)
    {
      UnitTestClassBase.InitializeClass(context);
    }

    [TestMethod]
    public void TestMethod1()
    {
      // メモ帳アプリケーションを実行
      Driver.Applications.Run("notepad.exe");

      // メモ帳の編集ボックスを取得
      ITextEdit edit = Driver.Find<IProcess>(new ProcessPattern()
      {
        ProcessName = "notepad"
      }).Find<ITopLevelWindow>(new WindowPattern()
      {
        WndClass = "Notepad"
      }).Find<ITextEdit>(new WindowPattern()
      {
        WndClass = "Edit"
      });

      // メモ帳でマウスクリックをシミュレート
      edit.Click();

      // メモ帳でテキスト入力をシミュレート
      string inputText = "test";
      edit.SetText(inputText);

      // メッセージを TestLeft ログにポスト
      Driver.Log.Screenshot(edit, "Notepad's edit box screenshot");
      Driver.Log.Warning("A warning message");

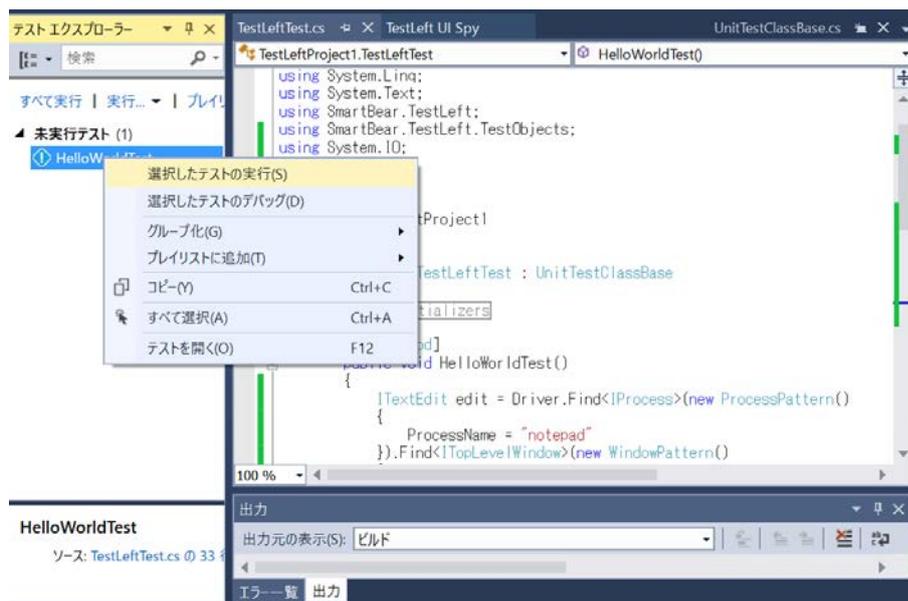
      // メモ帳に含まれるテキストを照合
      Assert.AreEqual(inputText, edit.wText);

      // TestLeft テスト ログを保存
      string logPath =
      Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.MyDocument
      s), DateTime.Now.ToString("MM_dd_yyyy_H_mm_ss"));
      Driver.Log.Save(logPath, Log.Format.Html);
    }

    [ClassCleanup]
    public static void ClassCleanUp()
    {
      UnitTestClassBase.FinalizeClass();
    }
  }
}
```

3. テストの実行とテスト結果の表示

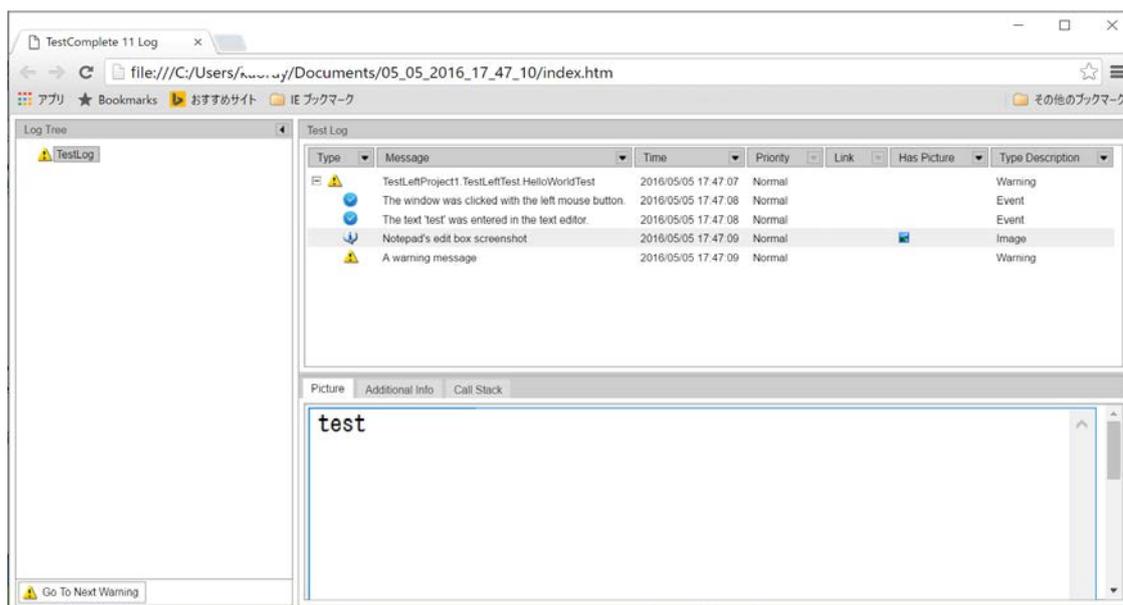
1. テスト プロジェクトをビルドします。
[ビルド]-[<TestLeft プロジェクト> のビルド] を選択して、ビルドします。
2. テスト エクスプローラー パネルを開いて、作成したテストを見つけます。
[テスト]-[ウィンドウ]-[テスト エクスプローラー]の順に選択します。
3. テストを右クリックして、[選択したテストを実行] をクリックします。



4. Visual Studio がテストを実行します。
テスト コードがメモ帳アプリケーションを実行し、編集ボックスの取得、キーボード入力のシミュレート、テスト結果を保存します。
5. テストが終了した後、Visual Studio は、テスト エクスプローラー パネルで MSTest テスト結果を表示します:



6. TestLeft テスト ログを表示するには、テスト ログが保管されているフォルダーに進みます。(今回は、カレント ユーザーの ドキュメント フォルダーになります)。結果のテストログを開きます:



注意: Visual Studio は、TestLeft テスト ログがエラーまたは警告を含んでいるかどうかを追跡しません。テスト コードが正常に実行された場合、テストがパスしたことをレポートします。

次のステップ

テストの作成方法のチュートリアルはこれで終了します。TestLeft を理解する上でお役に立てたと存じます。 TestLeft の詳細については、オンラインヘルプの下記のセクションをご参照ください:

関連トピック

⇒ [Object Identification](#)

テストするオブジェクトを見つける方法を説明します。

⇒ [About TestLeft Test Logs](#)

テスト ログに様々なデータをポストし、テスト ログの保存、ファイルにエクスポートする方法を説明します。

⇒ [Running TestLeft Tests on Remote Computers](#)

リモート コンピューター上で TestLeft を実行する方法を説明します。

⇒ [Running TestLeft Tests by Using Unit Testing Frameworks](#)

様々な単体テストフレームワークを使用して、TestLeft テストを実行する方法を説明します。

お問合せ先

TestLeft またはその他の SmartBear 製品の価格、ライセンスに関するお問合せや、テクニカル サポートのお問合せは、下記の Web サイトからお気軽にお問合せください:

エクセルソフト株式会社

<http://www.xlsoft.com/jp/services/contact.html>