Borland
A MICRO FOCUS COMPANY

Silk Test 13.5

Silk4NET ユーザー ガイド

Micro Focus
575 Anton Blvd., Suite 510
Costa Mesa, CA 92626

2012-09-20

目次

Silk4NET

Silk4NET は Microsoft Visual Studio に対応した Silk Test プラグインです。 Silk4NET を使用すると、機能テスト、回帰テスト、およびローカリゼーション テストの作成と管理を Visual Studio で直接、効率的に実行できます。 Silk4NET で実行できる Visual Studio 内の作業は、以下のとおりです。

- Visual Basic .NET を使用してテストを開発します。
- C# を使用してテストを開発します。
- Microsoft テスト環境内のテスト計画の一環としてテストを実行できます。
- ビルド プロセスの一環としてテストを実行できます。
- テスト結果を表示します。

Silk4NET では、AJAX と Web 2.0、RCP、WPF、Windows Forms と Win32 など、広範囲にわたるアプリケーション テクノロジがサポートされています。 Silk4NET は、複雑なテストに適用した場合にも自動化の利点が実現されるように設計されており、開発者が使い慣れた環境に直接、テストの自動化機能を追加するため、テスト アプリケーションに加えられた変更に簡単に対処できます。

また、Silk4NET の強力なテスト フレームワークを使用すると、複数のテスト プロジェクトに対してテストの再利用が促進されるため、より高い投資収益率 (ROI) を達成できます。 テスト スイートの構築と保守に要する時間が短縮されるため、QA 担当者はテスト範囲を拡張し、アプリケーション品質を最適化することができます。

Silk Test 製品スイート

Silk Test 製品スイートには、以下のコンポーネントが含まれています。

- Silk Test Workbench：Silk Test Workbench は、新しいネイティブ品質テスト環境です。上級者用の .NET スクリプトと、テストのアクセス可能性を高める革新的なストーリーボードベースのビジュアル テストが提供されます。
- Silk4NET：Silk4NET Visual Studio プラグインを使用すると、Visual Studio で直接 Visual Basic または C# のテスト スクリプトを作成できます。
- Silk4J：Silk4J Eclipse プラグインを使用すると、Eclipse 環境で直接 Java ベースのテスト スクリプトを作成できます。
- Silk Test Recorder：Silk Test Recorder を使用すると、GUI を使用してテストを記録および再生し、それらのテストを Silk Test Classic、Silk4J、または Silk4NET にエクスポートできます。
- Silk Test Classic：Silk Test Classic は、従来の 4Test Silk Test 製品です。
- Silk Test Agent：Silk Test Agent は、テストのコマンドを GUI 固有のコマンドに変換するソフトウェア プロセスです。 つまり、テストするアプリケーションをエージェントが動かし、監視しています。ホストマシン上で 1 つのエージェントをローカルに実行できます。 ネットワーク環境では、任意の数のエージェントをリモート マシン上で実行できます。

インストールする製品スイートによって、使用できるコンポーネントが決まります。 すべてのコンポーネントをインストールするには、完全インストール オプションを選択します。 Silk Test Classic を除くすべてのコンポーネントをインストールするには、標準インストール オプションを選択します。

製品通知サービス

製品通知サービスはシステム トレイで実行されるアプリケーションで、 Silk Test の更新が入手可能になった場合にそれを知らせます。 また、更新ページに移動可能なリンクも提供します。

サービスの実行

システム トレイで、更新通知アイコンをクリックすると、製品通知サービス アプリケーションが起動します。

インストール済みバージョン	現在インストールされている Silk Test アプリケーションのバージョン番号を提供します。
更新バージョン	利用可能な場合、次回のマイナー更新のリンクとバージョン番号を提供します。
新しいバージョン	利用可能な場合、次回のフル リリースのリンクとバージョン番号を提供します。
設定	**設定** ボタンをクリックして、 **設定** ウィンドウを開きます。 通知サービスで更新をチェックするかどうかと、その頻度を選択します。

Silk4NET の新機能

このセクションでは、Silk4NET に対して行われた重要な改善と変更を示します。

Silk4NET での記録

Silk Test Recorder を開始しなくても、Visual Studio 内で Silk4NET テストを 記録できます。 新しい Silk4NET プロジェクトを作成するか、新しい Silk4NET テストを Visual Studio の既存の Silk4NET プロジェクトに追加する場合は、テストを選択して記録できます。 記録が終了すると、記録されたテストは自動的に解析されて、新しいスクリプト ファイルに格納されます。

TrueLog と Silk4NET

Silk4NET を操作するときに、TrueLog を使用して、Silk4NET テストの実行中にビジュアル実行ログを作成できるようになりました。 TrueLog ファイルは、Silk4NET テストが実行されたプロセスの作業ディレクトリに作成されます。 Silk4NET テストの実行が完了すると、新しい **再生完了** ダイアログ ボックスが開き、完了したテストの TrueLog を確認することを選択できます。

Silk4NET の [オプション] ユーザー インターフェイス

Visual Studio で Silk4NET プロジェクトのオプションを設定できるようになりました。 たとえば、テスト中のアプリケーションの基本状態やアプリケーション構成その他のオプションを設定できます。
Silk4NET のオプションはすべて、プロジェクトに固有の構成ファイルに格納されます。

Silk4NET の Locator Spy

Silk4NET の **Locator Spy** を使用してアプリケーションのコントロールのロケーターをキャプチャできるようになりました。 キャプチャしたロケーターはテスト スクリプトに貼り付けることができます。 また、テスト スクリプトでロケーターの属性を手動で編集し、変更を **Locator Spy** で検証することができます。 **Locator Spy** を使用することで、テスト メソッド内のロケーターが有効であることが保障されます。

Silk4NET ヘルプの改善

Silk4NET ヘルプ に、アプリケーションの機能をテストする際に役立つ情報が追加されました。 *Silk4NET* ヘルプ に追加された情報は、以下のとおりです。

- 言語リファレンスがヘルプに追加されました。 このリファレンスでは、アプリケーションのテストに使用できる Visual Basic .NET および Visual C# のクラスおよびメソッドについて説明しています。
- キーワードおよび検索機能が追加されました。これにより、必要な内容を簡単に見つけることができます。

- Silk4NET のワークフローに従うようにヘルプの内容が再構成されました。

Windows Internet Explorer のサポート

Silk Test では、以下のバージョンで実行されるアプリケーションの記録と再生サポートが追加されました。

- Windows Internet Explorer 10

Mozilla Firefox のサポート

Silk Test では、以下のバージョンで実行されるアプリケーションの再生サポートが追加されました。

- Mozilla Firefox 12
- Mozilla Firefox 13
- Mozilla Firefox 14

Google Chrome のサポート

Silk Test では、以下のバージョンで実行されるアプリケーションの再生サポートが追加されました。

- Google Chrome 19
- Google Chrome 20
- Google Chrome 21

64 ビット Windows Internet Explorer バージョンのサポート

Silk Test では現在 64 ビット バージョンの Windows Internet Explorer をサポートしています。

Adobe Flex 4.6 のサポート

Silk Test では Adobe Flex4.6 アプリケーションがサポートされています。

Microsoft .NET Framework 4.5 のサポート

Silk Test で、Microsoft .NET Framework 4.5 で開発されたアプリケーションまたは Microsoft .NET Framework 4.5 で実行されるアプリケーションがサポートされるようになりました。

ブラウザの便利な切り替え

アプリケーション構成に、Web を使用するテストのための新しい設定が追加されました。この設定を使用すると、インストールされたブラウザの中からテストの再生に使用するものを簡単に選択できます。

Micro Focus へのお問い合わせ

Micro Focus は、世界的規模のテクニカル サポートおよびコンサルティング サービスを提供します。 すべての顧客のビジネスを成功に導くために、信頼できるサービスをタイムリーに提供するように Micro Focus はワールドワイドのサポート体制を整えています。

保守およびサポート契約を結んだすべてのお客様、および製品を評価中のお客様は、カスタマ サポートを受けることができます。 弊社の熟練したスタッフが、可能な限り迅速に専門家としてお客様の質問にお答えします。

http://supportline.microfocus.com/assistedservices.asp にアクセスするか、またはメールを supportline@microfocus.com に送信して、Micro Focus SupportLine と直接連絡できます。

また、*http://supportline.microfocus.com* の Micro Focus SupportLine では、 最新のサポートに関するニュースや、さまざまなサポート情報を得ることができます。このサイトに初めてアクセスした場合は、ユーザー登録が必要な場合があります。

Micro Focus SupportLine で必要な情報

Micro Focus SupportLine に連絡する際、可能な場合は以下の情報も提供してください。 詳細な情報をご提供いただければ、Micro Focus SupportLine はより効果的なサポートが可能になります。

- 問題の原因と考えられるすべての製品の名前とバージョン番号。
- コンピュータのメーカーと機種。
- オペレーティング システム名とバージョン、プロセッサ、メモリの詳細などのシステム情報。
- 問題を再現する手順などの、問題の詳細な説明。
- 関係するエラー メッセージの正確な表現。
- シリアル番号。

これらの番号を見つけるには、Micro Focus から受信した Electronic Product Delivery Notice 電子メールの件名および本文を参照してください。

Silk4NET 入門ガイド

Silk4NET を使用するには、次の操作を実行します。

1. Silk4NET プロジェクトを作成します。
2. プロジェクトに Silk4NET テストを追加します。 プロジェクトには記録されたテストおよび、手動でスクリプトを作成したテストを含めることができます。
3. テストを実行します。
4. テスト結果を分析します。

基本状態

アプリケーションの基本状態とは、各テスト ケースの実行開始前にアプリケーションに想定される既知の安定した状態です。アプリケーションは、各テスト ケースの実行が終了したあとに基本状態に戻る場合もあります。 大抵の場合この状態は、アプリケーションを最初に起動したときの状態になります。

Web アプリケーション構成や標準構成用のクラスを作成するとき、Silk4NET は自動的に基本状態を作成します。

基本状態はテストの整合性を保障するための重要な一因です。 各テスト ケースが安定した基本状態から開始することができることを保障することによって、あるテスト ケースのエラーによって、後続のテスト ケースが失敗しないことを保障することができます。

Silk4NET は、次の段階の間に、アプリケーションがその基本状態にあることを自動的に保障します。

- テストの実行前
- テストの実行中
- テストが成功裏に完了した後

注: Silk4NET は、基本状態とすべての Silk4NET オプションを config.silk4net 構成ファイルに保存します。 Silk4NET は、Silk4NET プロジェクトごとにこのようなファイルを作成します。

基本状態を変更する

必要に応じて、基本状態の実行可能ファイルの場所、作業ディレクトリ、ロケーター、URL を変更できます。 たとえば、テスト用の Web サイト上で以前にテストしていたものを、本番の Web サイトに対してテストを行いたい場合には、基本状態の URL を変更すれば、新しい環境でテストが実行されるようになります。

1. **Silk4NET** の隣にあるドロップダウン矢印をクリックして、**アプリケーション構成の編集** を選択します。 **アプリケーション構成の編集** ダイアログ ボックスが開き、既存のアプリケーション構成がリストされます。
2. **基本状態の編集** をクリックします。 **基本状態の編集** ダイアログ ボックスが開きます。
3. **実行可能ファイル** テキスト ボックスに、テストするアプリケーションの実行可能ファイルの名前とファイルへのパスを入力します。
   たとえば、Internet Explorer を指定する場合には、「C:¥Program Files¥Internet Explorer ¥IEXPLORE.EXE」と入力します。
4. 実行可能ファイルと組み合わせてコマンド ライン パターンを使用するには、**コマンド ライン引数** テキスト ボックスにコマンド ライン パターンを入力します。
5. テストするアプリケーションがディレクトリに依存する場合は、**作業ディレクトリ** テキスト ボックスにディレクトリを指定します。
   たとえば、Java アプリケーションを起動するバッチ ファイルを使用する場合には、バッチ ファイルは JAR ファイルを相対パスで参照することができます。 この場合、正しく相対パスが処理されるように作業ディレクトリを指定します。
6. **ロケーター** テキスト ボックスに、アプリケーションのメイン ウィンドウを識別する XPath ロケーター文字列を入力します。
7. Web サイトをテストする場合は、**URL** テキスト ボックスに、テストを開始するときに起動する Web ページの Web アドレスを入力します。
8. **OK** をクリックします。

アプリケーション構成

アプリケーション構成は、テストするアプリケーションに Silk4NET が接続する方法を定義します。 Silk4NET は、基本状態を作成するときに、アプリケーション構成を自動的に作成します。 しかし、アプリケーション構成を追加したり、変更や削除をすることが必要になる場合があります。 たとえば、データベースを変更するアプリケーションをテストしているときに、データベースの内容を確認するためにデータベースのビューアー ツールを使用する場合には、そのデータベースのビューアー ツール用のアプリケーション構成を追加する必要があります。

アプリケーション構成には次のものが含まれます：

- 実行可能ファイル パターン

  このパターンに一致するすべてのプロセスは、テストに対して有効化されます。 たとえば、メモ帳 (Notepad) の実行可能パターンは、¥*notepad.exe です。 実行可能ファイルの名前が notepad.exe で、任意のディレクトリに置かれているプロセスはすべて有効化されます。
- コマンド ライン パターン

  コマンドライン パターンは、テストを行うために有効化されるプロセスの制約に使用される補足パターンで、コマンドライン引数の一部 (実行可能ファイル名の後ろ部分) をマッピングすることにより行います。 コマンド ライン パターンを含むアプリケーション構成では、実行可能パターンとコマンド ライン パターンの両方に一致するプロセスのみが、テストに対して有効化されます。 コマンド ライン パターンが定義されていない場合は、指定された実行可能ファイル パターンを持つすべてのプロセスが有効化されます。 コマンド ラインの使用は、Java アプリケーションに対して特に有益です。これは、ほとんどの Java プログラムが javaw.exe を使用して実行されるためです。 つまり、典型的な Java アプリケーションに対してアプリケーション構成を作成する場合、実行可能パターンには *¥javaw.exe が使用され、このパターンはすべての Java プロセスに一致します。 このような場合、コマンド ライン パターンを使用して、該当するアプリケーションのみがテストに対して有効化されるようにします。 たとえば、アプリケーションのコマンド ラインが **com.example.MyMainClass** で終わる場合には、コマンド ライン パターンに ***com.example.MyMainClass** を使用します。

アプリケーション構成を変更する

アプリケーション構成は、テストするアプリケーションに Silk4NET が接続する方法を定義します。 Silk4NET は、基本状態を作成するときに、アプリケーション構成を自動的に作成します。 しかし、アプリケーション構成を追加したり、変更や削除をすることが必要になる場合があります。 たとえば、データベースを変更するアプリケーションをテストしているときに、データベースの内容を確認するためにデータベースのビューアー ツールを使用する場合には、そのデータベースのビューアー ツール用のアプリケーション構成を追加する必要があります。

1. **Silk4NET** の隣にあるドロップダウン矢印をクリックして、**アプリケーション構成の編集** を選択します。 **アプリケーション構成の編集** ダイアログ ボックスが開き、既存のアプリケーション構成がリストされます。
2. アプリケーション構成をさらに追加するには、**追加** をクリックします。

   新規アプリケーション構成 ウィザードが開きます。
   a) テストするアプリケーションの種類を選択して **次へ** をクリックします。
   b) テストしたいアプリケーションに対応する構成をクリックします。

      テストするアプリケーションがリストに表示されない場合は、**キャプションを持たないプロセスを表示しない** チェック ボックスをオフにします。 このオプションはデフォルトでオンになっており、キャプションが付いているアプリケーションのみをフィルタ処理するために使用します。
   c) Web アプリケーションのテストを選択した場合は、既存のブラウザ インスタンスを使用するのか新たにインスタンスを起動するのかを選択します。

d) **終了** をクリックします。 アプリケーションの実行可能なパターンが、**アプリケーション構成の編集** ダイアログ ボックスのアプリケーション構成のリストに追加されます。

3. アプリケーション構成を削除するには、該当するアプリケーション構成の隣にある **削除** をクリックします。
4. アプリケーション構成を編集するには、**基本状態の編集** をクリックします。 **基本状態の編集** ダイアログ ボックスが開きます。
5. **OK** をクリックします。

アプリケーション構成エラー

プログラムをアプリケーションにアタッチできない場合、以下のエラー メッセージが表示されます。
アプリケーション <Application Name> にアタッチするのに失敗しました。 詳細については、ヘルプを参照してください。

この場合、以下の表に示されている 1 つ以上の問題が原因である可能性があります。

問題	原因	解決策
タイムアウト	• システムが遅すぎます。 • システムのメモリ サイズが小さすぎます。	速いシステムを使用するか、現在使用しているシステムのメモリ使用量を減らします。
ユーザー アカウント制御 (UAC) の失敗	システムの管理者権限がありません。	管理者権限を持つユーザー アカウントでログインします。
コマンド ライン パターン	コマンド ライン パターンが固有すぎます。 この問題は特に Java の場合に発生します。 再生が意図したとおりに機能しないことがあります。	パターンから不明瞭なコマンドを削除します。

Silk4NET プロジェクトの操作

このセクションでは、Silk4NET プロジェクトの使用方法について説明します。

Silk4NET プロジェクトには、Silk4NET を使用してアプリケーションの機能をテストするために必要なリソースがすべて含まれています。

Silk4NET プロジェクトの作成

1. **Silk4NET** > **新しいプロジェクト** または **ファイル** > **新規作成** > **プロジェクト** をクリックします。 **新しいプロジェクト** ダイアログ ボックスが表示されます。
2. **インストールされたテンプレート** で、**Visual Basic** または **Visual C#** をクリックし、**Silk4NET プロジェクト**をダブルクリックします。 **Silk4NET テストの作成** ダイアログ ボックスが開きます。
3. 次のいずれかのオプション ボタンをクリックして、Silk4NET テストの作成方法を選択します。

Silk4NET テストの記録	テスト対象アプリケーションに対する操作および検証を記録し、記録されたオートメーション ステートメントを含む新しいテストを生成します。
空の Silk4NET テストの作成	オートメーション ステートメントを後で入力できる空のテストを作成します。

4. **OK** をクリックします。 空の Silk4NET テストを作成するように選択した場合は、Silk4NET プロジェクトを含む新しいソリューションが作成されます。 また、このプロジェクトには、言語固有の以下のファイル名を使用して、Silk4NET テストも作成されます。

   - UnitTest1.vb
   - UnitTest1.cs

5. 新しい Silk4NET テストを記録するように選択した場合は、**新規アプリケーション構成** ウィザードが開きます。 テストするアプリケーションのタイプを選択します。

   - 標準アプリケーションをテストする場合は、**標準テスト構成** をクリックします。
   - Web アプリケーションをテストする場合は、**Web サイト テスト構成** をクリックします。

6. **次へ** をクリックします。
7. 標準アプリケーションをテストするように選択した場合は、**新規標準構成** ダイアログ ボックスが開きます。 テストしたいアプリケーションに対応する構成をクリックします。

   テストするアプリケーションがリストに表示されない場合は、**キャプションを持たないプロセスを表示しない** チェック ボックスをオフにします。 このオプションはデフォルトでオンになっており、キャプションが付いているアプリケーションのみをフィルタ処理するために使用します。
8. Web アプリケーションをテストするように選択した場合は、**新規 Web サイト構成** ダイアログ ボックスが開きます。**ブラウザの種類** リスト ボックスから、テストするブラウザの種類を選択します。

   a) **ブラウザ インスタンス** セクションで、次のオプション ボタンをクリックします。

既存のブラウザを使用する	すでに開いているブラウザを使用する場合は、このオプション ボタンをクリックします。 たとえば、テストしたい Web ページがすでにブラウザ上に表示されている場合などに、このオプションを使用します。
新しいブラウザを開始する	テストを構成する際に、新しいブラウザ インスタンスを開始する場合には、このオプション ボタンをクリックします。 次に、**ブラウズする URL** テキスト ボックスで、開く Web ページを指定します。

9. **終了** をクリックします。 テスト メソッドの再生で Google Chrome の既存のインスタンスを選択した場合は、Silk Test Recorder がオートメーション サポートが含まれているかどうかをチェックします。

オートメーション サポートが含まれていない場合は、Silk Test Recorder が Google Chrome を再起動する必要があることを通知します。 アプリケーションと **記録中** ダイアログ ボックスが開きます。

注: ソリューション エクスプローラーのコンテキスト メニューを使って Silk4NET プロジェクトを既存のソリューションに追加することもできます。

Silk4NET テストの操作

Silk4NET テストの使用方法について説明します。

AUT に対して行われたユーザーの操作を記録するか、Visual Basic または Visual C# のテスト クラスおよびメソッドのスクリプトを手動で記述することで、Silk4NET テストを新規に作成できます。

プロジェクトへの Silk4NET テストの追加

既存の Silk4NET またはテスト プロジェクトにのみ Silk4NET テストを追加できます。 Silk4NET またはテスト プロジェクトが存在しない場合は、 Silk4NET テストを作成する前に Silk4NET またはテスト プロジェクトを作成してください。

1. **Silk4NET** > **新しいテスト** または **プロジェクト** > **新しい項目の追加** をクリックします。

   注: ソリューションに複数の Silk4NET プロジェクトが存在する場合、**プロジェクトの選択** でリストから新しいテストを追加するプロジェクトを選択します。

   新しい項目の追加 ダイアログ ボックスが開きます。
2. **インストールされたテンプレート** で次のいずれかをクリックします。
   - プロジェクトが Visual Basic プロジェクトの場合は、**共通項目** > **Silk4NET テスト** をクリックします。
   - プロジェクトが Visual C# プロジェクトの場合は、**Visual C# アイテム** > **Silk4NET テスト** をクリックします。

   Silk4NET テストの作成 ダイアログ ボックスが開きます。
3. 次のいずれかのオプション ボタンをクリックして、Silk4NET テストの作成方法を選択します。

Silk4NET テストの記録	テスト対象アプリケーションに対する操作および検証を記録し、記録されたオートメーション ステートメントを含む新しいテストを生成します。
空の Silk4NET テストの作成	オートメーション ステートメントを後で入力できる空のテストを作成します。

4. **OK** をクリックします。 空の Silk4NET テストを作成するように選択した場合は、Silk4NET プロジェクトを含む新しいソリューションが作成されます。 また、このプロジェクトには、言語固有の以下のファイル名を使用して、Silk4NET テストも作成されます。
   - UnitTest1.vb
   - UnitTest1.cs
5. 新しい Silk4NET テストを記録するように選択した場合は、**新規アプリケーション構成** ウィザードが開きます。 テストするアプリケーションのタイプを選択します。
   - 標準アプリケーションをテストする場合は、**標準テスト構成** をクリックします。
   - Web アプリケーションをテストする場合は、**Web サイト テスト構成** をクリックします。
6. **次へ** をクリックします。
7. 標準アプリケーションをテストするように選択した場合は、**新規標準構成** ダイアログ ボックスが開きます。 テストしたいアプリケーションに対応する構成をクリックします。

   テストするアプリケーションがリストに表示されない場合は、**キャプションを持たないプロセスを表示しない** チェック ボックスをオフにします。 このオプションはデフォルトでオンになっており、キャプションが付いているアプリケーションのみをフィルタ処理するために使用します。
8. Web アプリケーションをテストするように選択した場合は、**新規 Web サイト構成** ダイアログ ボックスが開きます。**ブラウザの種類** リスト ボックスから、テストするブラウザの種類を選択します。

a) **ブラウザ インスタンス** セクションで、次のオプション ボタンをクリックします。

既存のブラウザを使用する	すでに開いているブラウザを使用する場合は、このオプション ボタンをクリックします。 たとえば、テストしたい Web ページがすでにブラウザ上に表示されている場合などに、このオプションを使用します。
新しいブラウザを開始する	テストを構成する際に、新しいブラウザ インスタンスを開始する場合には、このオプション ボタンをクリックします。 次に、**ブラウズする URL** テキスト ボックスで、開く Web ページを指定します。

9. **終了** をクリックします。 テスト メソッドの再生で Google Chrome の既存のインスタンスを選択した場合は、Silk Test Recorder がオートメーション サポートが含まれているかどうかをチェックします。 オートメーション サポートが含まれていない場合は、Silk Test Recorder が Google Chrome を再起動する必要があることを通知します。 アプリケーションと **記録中** ダイアログ ボックスが開きます。

テストを記録するように選択した場合は、記録したテストがプロジェクトに追加されます。 空のテストを追加するように選択した場合は、空の Silk4NET テストがプロジェクトに追加されます。

注: ソリューション エクスプローラーのコンテキスト メニューを使って Silk4NET テストを Silk4NET またはテスト プロジェクトに追加することもできます。

Silk4NET テストの記録

1. **Silk4NET** > **新しいテスト** または **プロジェクト** > **新しい項目の追加** をクリックします。

   注: ソリューションに複数の Silk4NET プロジェクトが存在する場合、**プロジェクトの選択** でリストから新しいテストを追加するプロジェクトを選択します。

   新しい項目の追加 ダイアログ ボックスが開きます。
2. **インストールされたテンプレート** で次のいずれかをクリックします。
   - プロジェクトが Visual Basic プロジェクトの場合は、**共通項目** > **Silk4NET テスト** をクリックします。
   - プロジェクトが Visual C# プロジェクトの場合は、**Visual C# アイテム** > **Silk4NET テスト** をクリックします。

   Silk4NET テストの作成 ダイアログ ボックスが開きます。
3. **Silk4NET テストの記録** をクリックします。

Silk4NET テストの記録	テスト対象アプリケーションに対する操作および検証を記録し、記録されたオートメーション ステートメントを含む新しいテストを生成します。

   新規アプリケーション構成 ウィザードが開きます。
4. テストするアプリケーションのタイプを選択します。
   - 標準アプリケーションをテストする場合は、**標準テスト構成** をクリックします。
   - Web アプリケーションをテストする場合は、**Web サイト テスト構成** をクリックします。
5. **次へ** をクリックします。
6. 標準アプリケーションをテストするように選択した場合は、**新規標準構成** ダイアログ ボックスが開きます。 テストしたいアプリケーションに対応する構成をクリックします。

   テストするアプリケーションがリストに表示されない場合は、**キャプションを持たないプロセスを表示しない** チェック ボックスをオフにします。 このオプションはデフォルトでオンになっており、キャプションが付いているアプリケーションのみをフィルタ処理するために使用します。
7. Web アプリケーションをテストするように選択した場合は、**新規 Web サイト構成** ダイアログ ボックスが開きます。**ブラウザの種類** リスト ボックスから、テストするブラウザの種類を選択します。

   a) **ブラウザ インスタンス** セクションで、次のオプション ボタンをクリックします。

既存のブラウザを使用する すでに開いているブラウザを使用する場合は、このオプション ボタンをクリックします。 たとえば、テストしたい Web ページがすでにブラウザ上に表示されている場合などに、このオプションを使用します。

新しいブラウザを開始する テストを構成する際に、新しいブラウザ インスタンスを開始する場合には、このオプション ボタンをクリックします。 次に、**ブラウズする URL** テキスト ボックスで、開く Web ページを指定します。

8. **終了** をクリックします。 テスト メソッドの再生で Google Chrome の既存のインスタンスを選択した場合は、Silk Test Recorder がオートメーション サポートが含まれているかどうかをチェックします。 オートメーション サポートが含まれていない場合は、Silk Test Recorder が Google Chrome を再起動する必要があることを通知します。 アプリケーションと **記録中** ダイアログ ボックスが開きます。
9. 記録を行う、テスト対象アプリケーションとの対話を実行します。

   詳細については、『*Silk Test Recorder* ヘルプ』を参照してください。
10. 記録が終了したら、**記録** ダイアログ ボックスの **記録の停止** をクリックします。

記録された対話は、プロジェクトにファイルとして追加されます。 生成されたファイルのデフォルトのファイル名は、プロジェクトのデフォルト プログラミング言語に応じて、UnitTest<Index>.cs または UnitTest<Index>.vb になります。 たとえば、Visual Basic プロジェクトの初回テストを記録している場合、生成されたファイルの名前は UnitTest1.vb になります。

注: 新しいプロジェクトを作成し、そこに新しいテスト記録することもできます。

Silk4NET テストの手動作成

1. Silk4NET テスをプロジェクトに追加します。
2. オプション : 特定のアプリケーション テクノロジのコントロールのサポートを追加するには、以下の例に示すように、アプリケーション テクノロジの名前空間を参照するテストの先頭にインポート ステートメントを含める必要があります。

```
'Visual Basic .NET
Imports SilkTest.Ntf.Wpf
Imports SilkTest.Ntf.XBrowser
Imports SilkTest.Ntf.Win32
```

```
//C#
using SilkTest.Ntf.Wpf;
using SilkTest.Ntf.XBrowser;
using SilkTest.Ntf.Win32;
```

3. テスト アプリケーションの基本状態を構成します。 例 :

```
'Visual Basic .NET
Dim baseState = New BaseState("C:¥¥Windows¥¥system32¥¥notepad.exe",
"/Window[@caption='Untitled - Notepad']")
baseState.WorkingDirectory = "%USERPROFILE%"
baseState.Execute()
```

```
//C#
BaseState baseState = new BaseState("C:¥¥Windows¥¥system32¥¥notepad.exe",
 "/Window[@caption='Untitled - Notepad']");
baseState.WorkingDirectory = "%USERPROFILE%"; baseState.Execute();
```

注: 基本状態を使用すると、テストするアプリケーションがフォアグラウンドで実行中であることを保証できます。 これにより、テストが常に同じアプリケーション状態で開始されることが保証され、信頼性が高まります。 基本状態を使用するには、メイン ウィンドウの外観、およびテストするアプリケーションが実行されていない場合のアプリケーションの起動方法を指定する必要が

あります。 基本状態の作成は任意です。 ただし、ベスト プラクティスとして、基本状態を作成することをお勧めします。

4. テスト アプリケーションの目的の機能をテストするクラスとメソッドを追加します。

Locator Spy を使用してロケーターをテスト メソッドに追加する

Locator Spy を使用してロケーターを手動でキャプチャし、テスト メソッドにロケーターをコピーします。 たとえば、**Locator Spy** を使って、GUI オブジェクトのキャプションや XPath ロケーター文字列を識別できます。 そして、関係するロケーター文字列や属性をスクリプト内のテストメソッドにコピーします。

1. 変更したいテスト クラスを開きます。
2. **Silk4NET** > **ロケーターの記録** をクリックします。 **Locator Spy** が開きます。
3. 記録するオブジェクトの上にマウスを移動します。 関連するロケーター文字列は、**選択済みロケーター** テキスト ボックスに表示されます。
4. **Ctrl+Alt** を押してオブジェクトをキャプチャします。

   注: スクリプト オプション ダイアログ ボックスの **全般記録オプション** ページで別の記録停止キー操作を指定した場合は、**Ctrl+Shift** を押してオブジェクトをキャプチャします。
5. オプション：**追加のロケーター属性の表示** をクリックすると、関係する属性のすべてが **ロケーター属性** テーブルに表示されます。
6. 省略可能*:* **ロケーター属性** テーブルで、記録したロケーター属性を他のロケーター属性と置き換えることができます。

   たとえば、記録したロケーターが次のような場合を考えます。

   ```
   /Window[@caption='MyApp']//Control[@id='table1']
   ```

   キャプション Files が **ロケーター属性** テーブルにリストされている場合、次のように手動でロケーターを変更することができます。

   ```
   /Window[@caption='MyApp']//Control[@caption='Files']
   ```

   新しいロケーターは、**選択済みロケーター** テキスト ボックスに表示されます。
7. ロケーターをコピーするには、**ロケーターをクリップボードにコピー** をクリックします。

   選択済みロケーター テキスト ボックスで、コピーするロケーター文字列の位置をマークし、マークしたテキストを右クリックして **コピー** をクリックすることもできます。
8. スクリプト内で、記録したロケーターを貼り付ける位置にカーソルを置きます。

   たとえば、スクリプト内の Find メソッドの該当するパラメータにカーソルを置きます。

   ロケーターを貼り付けるテスト メソッドでは、ロケーターをパラメータとして受け取れるメソッドを使用する必要があります。 **Locator Spy** を使用することで、クエリ文字列が正しいことが保障されます。
9. ロケーターをテスト ケースまたはクリップボードにコピーします。
10. **閉じる** をクリックします。

Silk4NET テストの実行

1. Visual Studio で **ファイル** > **開く** > **プロジェクト/ソリューション** をクリックします。 **プロジェクトを開く** ダイアログ ボックスが表示されます。
2. 実行するテストを含むプロジェクトのプロジェクト ファイルを参照します。

   プロジェクト ファイルは .sln ファイルです (Silk4NETProject1.sln など)。

3. プロジェクト ファイルをダブルクリックします。 プロジェクト ファイルに含まれているテスト ファイルが Visual Studio のメイン ウィンドウに開きます。
4. ツールバーで **ソリューションのテストをすべて実行** をクリックします。 開いているプロジェクトのすべてのテストが実行されます。実行が完了すると、 **再生完了** ダイアログ ボックスが開きます。
5. TrueLog を使用してテスト結果を調べるには、**結果の検討** をクリックします。ダイアログ ボックスを閉じるには、**OK** をクリックします。

テスト結果の分析

1. Silk4NET テストを実行します。 実行が完了すると、 **再生完了** ダイアログ ボックスが開きます。
2. **結果の検討** をクリックし、TrueLog を使用してテスト結果を調べます。 Silk TrueLog Explorer が開きます。
3. Silk TrueLog Explorer で結果をクリックします。
   テストに失敗すると、Silk TrueLog Explorer によってスクリーンショットがキャプチャされます。

TrueLog を使用したビジュアル実行ログ

'TrueLog は、ビジュアルな検証を通じてテスト ケースの失敗の根本的な原因の分析を単純化するための強力なテクノロジです。 テストの結果は、TrueLog Explorer で検証できます。 テストの実行中にエラーが発生すると、TrueLog はそのエラーが発生したスクリプトの行を簡単に特定し、問題を解決できるようにします。

注: TrueLog は、スクリプトに対して単一のローカル エージェントまたはリモート エージェントのみをサポートしています。たとえば、1 つのマシンでアプリケーションをテストし、そのアプリケーションが別のマシンのデータベースにデータを書き込む場合のように、複数のエージェントを使用する場合は、スクリプトで使用された最初のエージェントに対してのみ TrueLog が書き出されます。 リモート エージェントを使用する場合は、リモート マシンにも TrueLog ファイルが書き出されます。

TrueLog Explorer の詳細については、 **スタート** > **プログラム** > **Silk** > **Silk Test** > **ドキュメント** にある *Silk TrueLog Explorer* ユーザー ガイド を参照してください。

Silk4NET で TrueLog を有効にして、Silk4NET テストの実行中にビジュアル実行ログを作成できます。 TrueLog ファイルは、Silk4NET テストが実行されたプロセスの作業ディレクトリに作成されます。

デフォルトの設定では、スクリプトでエラーが発生した場合にのみスクリーンショットが作成され、エラーの発生したテスト ケースのログのみが作成されます。

TrueLog の有効化

新しい Silk4NET スクリプトでは、TrueLog がデフォルトで有効になっています。 Visual Studio Unit Testing Framework を使用している既存の Silk4NET スクリプトで TrueLog を有効にするには、スクリプト内のすべてのテスト クラスの TestClass 属性を SilkTestClass 属性で置き換える必要があります。

TrueLog を有効にするには、以下を実行します。

1. TrueLog を有効にするテスト クラスが含まれたスクリプトを開きます。
2. テスト クラスに SilkTestClass 属性を追加します。

ディレクトリの TestResults サブディレクトリに TrueLog が作成されます。このディレクトリには、Visual Studio ソリューション ファイルおよび Visual Studio Unit Testing Framework の結果が格納されています。 Visual Studio ソリューション ファイルには、Silk4NET スクリプトが格納されています。 Silk4NET テストの実行が完了したら、ダイアログ ボックスが開きます。**結果の検討** をクリックすると、完了したテストの TrueLog を確認できます。

使用例

Visual Basic スクリプト内のクラスに対して TrueLog を有効にするには、以下のコードを使用します。

```
<SilkTestClass()> Public Class MyTestClass
  <TestMethod()> Public Sub MyTest()
    ' my test code
  End Sub
End Sub
```

C# スクリプト内のクラスに対して TrueLog を有効にするには、以下のコードを使用します。

```
[SilkTestClass]
public class MyTestClass {
  [TestMethod]
  public void MyTest() {
```

```
    // my test code
  }
}
```

TrueLog で非 ASCII 文字が正しく表示されない理由

TrueLog Explorer は MBCS ベースのアプリケーションであるため、正しく表示するには、すべての文字列が MBCS 形式でエンコードされている必要があります。 TrueLog Explorer でデータを表示およびカスタマイズすると、データが表示される前に、多数の文字列変換処理が発生することがあります。

UTF-8 でエンコードされた Web サイトをテストする場合は、文字列を含むデータをアクティブな Windows システム コード ページに変換できないことがあります。 このような場合、TrueLog Explorer は変換できない非 ASCII 文字列を、構成可能な置換文字 (通常は「?」) で置き換えます。

TrueLog Explorer で非 ASCII 文字列を正確に表示するには、システム コード ページに適切な言語 (日本語など) を設定します。

Team Foundation Server での Silk4NET の使用

このセクションでは、Visual Studio Team Foundation Server (TFS) を使用して Silk4NET テストを実行する方法について説明します。

新機能、サポート対象のプラットフォームとバージョン、既知の問題、および回避策の詳細については、『*Silk Test* リリース ノート』(*http://supportline.microfocus.com/productdoc.aspx* で入手可能) を参照してください。

Silk4NET テストの TFS での実行

注: TFS または Visual Studio の機能についての説明など、このタスクのステップの詳細な情報については、それぞれの製品のドキュメントを参照してください。

Silk4NET テストを実行するために、TFS を使用することができます。

1. Visual Studio で **チーム エクスプローラー ビュー** を開き、 **チーム プロジェクトに接続** をクリックして TFS に接続します。
2. **チーム エクスプローラー ビュー** で、 Silk4NET プロジェクトを TFS に追加します。
3. 省略可能*:* さらに、 Silk4NET テストを Silk4NET プロジェクトに追加します。
4. **チーム エクスプローラー ビュー** で、テストを TFS にチェックインします。
5. Silk4NET プロジェクトを含んだソリューションを右クリックし、 **追加** > **新しい項目** > **テストの設定** > **テストの設定** をクリックして新しいテスト設定ファイルを作成します。
6. テストの実行に使用するテスト コントローラを設定します。
7. **チーム エクスプローラー ビュー** で、新しいビルド定義を作成します。
8. テスト設定ファイルをビルド定義に追加します。
9. Silk4NET テスト プロジェクト アセンブリの自動テストがビルド後に実行されるように、ビルド定義を設定します。
10. 方法*:* テスト エージェントを設定して、デスクトップと対話するテストを実行する の手順に従って、Silk4NET と AUT 間の対話を有効化します。
11. **チーム エクスプローラー ビュー** で、ビルド定義を実行して Silk4NET テストを実行します。
12. 省略可能*:* TrueLog ファイルを分析します。

TFS で実行した Silk4NET テストの TrueLog ファイルの場所

TFS で Silk4NET テストを実行する場合、実行が完了した後に **再生完了** ダイアログ ボックスは表示されず、そのテストの TrueLog ファイルはローカル マシンに出力されません。 TFS で実行した Silk4NET テストの結果を分析するために生成した TrueLog ファイルの場所は次のようになります。

1. TFS がテストを実行したマシンのテスト エージェントに接続します。
2. TrueLog ファイルにアクセスするには、デフォルトの場所 <システム ドライブ>/Builds に移動します。

   たとえば、 C:/Builds になります。

3. *Builds* ディレクトリがテスト エージェントのデフォルトの場所に存在しない場合、デフォルト値が変更されています。 次の手順に従って、TrueLog ファイルが出力される場所を特定します。
   a) TFS を実行するマシンに移動します。
   b) **スタート** > **すべてのプログラム** > **Microsoft Visual Studio Team Foundation Server <バージョン番号>** > **Team Foundation Server 管理コンソール** をクリックします。 **Team Foundation Server 管理コンソール** が開きます。
   c) **ビルド構成** をクリックします。
   d) ビルド エージェント マシンの名前の下の **プロパティ** をクリックします。 [ビルド エージェント プロパティ] ダイアログ ボックスが開きます。 TrueLog ファイルが出力されるディレクトリが **作業ディレクトリ** テキスト フィールドで定義されています。

スクリプト オプションの設定

記録、ブラウザ、カスタム属性、無視するクラス、同期、および再生モードに関するスクリプト オプションを指定します。

注: Silk4NET のプロジェクトごとに、Silk4NET によって config.Silk4net 構成ファイルが作成されます。 Silk4NET によって、テスト対象のアプリケーションの基本状態とすべてのオプションがこのファイルに保存されます。 その後、オプションが再生中に使用されます。

TrueLog オプションの設定

ビットマップをキャプチャして Silk4NET の情報を記録するように TrueLog を有効化します。

ビットマップとコントロールを TrueLog に記録すると Silk4NET のパフォーマンスに悪影響が出ることがあります。 ビットマップをキャプチャして情報を記録すると TrueLog ファイルが大きくなることがあるので、エラーを含むテスト ケースのみを記録するように、詳細情報が必要なテスト ケース用に TrueLog オプションを調整できます。

テストの結果は、TrueLog Explorer で検証できます。 TrueLog Explorer の詳細については、**スタート** > **プログラム** > **Silk** > **Silk Test** > **ドキュメント** にある *Silk TrueLog Explorer* ユーザー ガイド を参照してください。

TrueLog を有効にして、TrueLog が Silk4NET 用に収集する情報をカスタマイズするには、次の手順に従います。

1. **Silk4NET** の隣にあるドロップダウン矢印をクリックし、**オプションの編集** を選択します。 **スクリプト オプション** ダイアログ ボックスが開きます。
2. **TrueLog** タブをクリックします。
3. **ログ** 領域で **TrueLog の有効化** チェック ボックスをオンにします。
   - 正常なものとエラーになったものを両方とも含めて、すべてのテスト ケースのアクティビティを記録するには、**すべてのテストケース** をクリックします。 これはデフォルトの設定です。
   - エラーが発生したテスト ケースのみのアクティビティを記録するには、**エラーのあるテストケース** をクリックします。
4. **TrueLog ファイル** フィールドで TrueLog ファイルのパスと名前を入力するか、**参照** をクリックしてファイルを選択します。

   このパスは、エージェントを実行しているマシンの相対パスです。 デフォルト パスは Silk4J プロジェクト フォルダのパスであり、デフォルトの名前はスイート クラスの名前に .xlg 接尾辞が付いたものになります。

   注: ローカル パスまたはリモート パスをこのフィールドに指定する場合は、スクリプトの実行時までパスを検証できません。
5. **スクリーンショット モード** を選択します。

   デフォルトは **なし** です。
6. 任意：**遅延** を設定します。

   この遅延により、ビットマップが取られる前に Windows がアプリケーション ウィンドウを描画する時間を確保できます。 キャプチャされたビットマップでアプリケーションが適切に描画されない場合は、遅延時間を増やしてください。
7. **OK** をクリックします。

記録オプションの設定

記録を一時停止するためのショートカット キーの組み合わせを設定したり、絶対値による指定やマウスの移動操作が記録されるかどうかを指定したりします。

注: 以下の設定はすべて任意です。 テスト メソッドの品質が向上する場合に、これらの設定を変更してください。

1. **Silk4NET** の隣にあるドロップダウン矢印をクリックし、**オプションの編集** を選択します。 **スクリプト オプション** ダイアログ ボックスが開きます。
2. **記録** タブをクリックします。
3. 記録の一時停止に使用するショートカット キーの組み合わせとして Ctrl+Shift を設定するには、**OPT_ALTERNATE_RECORD_BREAK** チェック ボックスをオンにします。
   デフォルトのショートカット キーの組み合わせは、Ctrl+Alt です。
4. スクロール イベントの絶対値を記録するには、**OPT_RECORD_SCROLLBAR_ABSOLUT** チェック ボックスをオンにします。
5. マウスの移動操作を記録するには、**OPT_RECORD_MOUSEMOVES** チェック ボックスをオンにします。
6. マウスの移動操作を記録する場合は、**OPT_RECORD_MOUSEMOVE_DELAY** テキスト ボックスで、MouseMove 操作を記録する前に必要なマウスの静止時間をミリ秒で指定します。
   デフォルト値は、200 に設定されています。
7. Click 操作ではなく TextClick 操作を記録するには、通常 TextClick 操作が Click 操作よりも望ましいオブジェクトで **OPT_RECORD_TEXT_CLICK** チェック ボックスをオンにします。
8. **OK** をクリックします。

ブラウザの記録オプションの設定

記録中に無視するブラウザの属性や DOM 関数の代わりに、ユーザーの入力そのものを記録するかどうかを指定します。

注: 以下の設定はすべて任意です。 テスト メソッドの品質が向上する場合に、これらの設定を変更してください。

1. **Silk4NET** の隣にあるドロップダウン矢印をクリックし、**オプションの編集** を選択します。 **スクリプト オプション** ダイアログ ボックスが開きます。
2. **ブラウザ** タブをクリックします。
3. **ロケーター属性名除外リスト** グリッドで、記録中に無視する属性名を入力します。
   たとえば、height という名前の属性を記録しない場合には、height 属性名をグリッドに追加します。
   複数の属性名を指定する場合にはカンマで区切ります。
4. **ロケーター属性値除外リスト** グリッドで、記録中に無視する属性値を入力します。
   たとえば、x-auto という値を持つ属性を記録しない場合には、x-auto をグリッドに追加します。
   複数の属性値を指定する場合にはカンマで区切ります。
5. DOM 関数の代わりにユーザーの入力そのものを記録するには、**OPT_XBROWSER_RECORD_LOWLEVEL** チェック ボックスをオンにします。
   たとえば、DomClick の代わりに Click、SetText の代わりに TypeKeys を記録するには、このチェック ボックスをオンにします。
   アプリケーションでプラグインまたは AJAX を使用している場合は、ユーザーの入力そのものを使用します。 アプリケーションでプラグインまたは AJAX を使用していない場合は、再生中にブラウザにフォーカスを設定したりブラウザをアクティブにしたりする必要がない高レベル DOM 関数を使用することをお勧めします。 テストで DOM 関数を使用すると、より高速になり、信頼性も高まります。

6. **OK** をクリックします。

カスタム属性の設定

Silk4NET には、ロケーターが記録時に一意となり、メンテナンスが容易になるようにする、高度なロケーター生成メカニズムが備えられています。 使用するアプリケーションやフレームワークに応じて、最適な結果を得るためにデフォルト設定を変更できます。 それぞれのテクノロジで使用できる任意のプロパティ（整数や倍精度の数値、文字列、項目識別子、列挙値）を、カスタム属性として使用できます。

頻繁には変更されない属性を利用して、適切に定義されたロケーターでは、メンテナンス作業が少なく抑えられます。 カスタム属性を使用すると、caption や index などの他の属性を使用するよりも高い信頼性を得ることができます。これは、caption はアプリケーションを他の言語に翻訳した場合に変更され、index は他のオブジェクトが追加されると変更される可能性があるためです。

xBrowser、WPF、Java SWT、Swing アプリケーションでは、任意のプロパティ（*myCustomProperty* を定義する WPFButton など）を取得し、カスタム属性として使用することもできます。 最適な結果を得るために、テストで利用する要素にカスタム オートメーション ID を追加します。 Web アプリケーションでは、操作する要素に などの属性を追加できます。 また、Java SWT では、GUI を実装する開発者が属性（testAutomationId など）をウィジェットに対して定義することによって、アプリケーション内でそのウィジェットを一意に識別できます。 テスト担当者は、その属性をカスタム属性（この場合は testAutomationId）のリストに追加し、その一意の ID によってコントロールを識別できます。 この手法によって、ロケーターの変更に伴うメンテナンス作業を回避することができます。

caption のように、複数のオブジェクトで同じ属性値が共有されている場合、Silk4NET は、複数の利用可能な属性を "and" 操作で結合してロケーターを一意にするよう試み、一致したオブジェクトのリストを単一のオブジェクトになるまで絞り込んでいきます。 それができなくなった場合には、索引を付加します。つまり、ロケーターは caption が *xyz* である *n* 番め のオブジェクトを探すことを意味します。

複数のオブジェクトに同じカスタム属性の値が割り当てられた場合は、そのカスタム属性を呼び出したときにその値を持つすべてのオブジェクトが返されます。 たとえば、一意の ID として loginName を 2 つの異なるテキスト フィールドに割り当てた場合は、loginName 属性を呼び出したときに、両方のフィールドが返されます。

1. **Silk4NET** の隣にあるドロップダウン矢印をクリックし、**オプションの編集** を選択します。 **スクリプト オプション** ダイアログ ボックスが開きます。
2. **カスタム属性** タブを選択します。
3. **テクノロジ ドメインを選択します** リスト ボックスから、テストするアプリケーションのテクノロジ ドメインを選択します。

   注: Flex または Windows API ベースのクライアント／サーバー（Win32）アプリケーションには、カスタム属性を設定できません。
4. 使用する属性をリストに追加します。

   カスタム属性が利用可能な場合は、ロケーター生成プログラムは、他の属性の前にそれらの属性を使用します。 リストの順番は、ロケーター生成プログラムが使用する属性の優先順位を表しています。 指定した属性が選択したオブジェクトに対して利用可能ではなかった場合には、Silk4NET はテストしているアプリケーションのデフォルトの属性を使用します。

   複数の属性名を指定する場合にはカンマで区切ります。

   注: Web アプリケーションにカスタム属性を含めるためには、HTML タグとして追加します。 たとえば、bcauid という属性を追加するには、<input type='button' bcauid='abc' value='click me' /> と入力します。

   注: Java SWT アプリケーションにカスタム属性を含めるためには、org.swt.widgets.Widget.setData(String *key*, Object *value*) メソッドを使用します。

注: Swing アプリケーションにカスタム属性を含めるためには、SetClientProperty("propertyName", "propertyValue") メソッドを使用します。

5. **OK** をクリックします。

無視するクラスの設定

記録や再生中に無視したいクラスの名前を指定します。

1. **Silk4NET** の隣にあるドロップダウン矢印をクリックし、**オプションの編集** を選択します。 **スクリプト オプション** ダイアログ ボックスが開きます。
2. **無視するクラス** タブをクリックします。
3. **無視するクラス** グリッドで、記録や再生中に無視するクラスの名前を入力します。
   複数のクラス名を指定する場合にはカンマで区切ります。
4. **OK** をクリックします。

記録/再生の対象とする WPF クラスの設定

記録や再生の対象にしたい WPF クラスの名前を指定します。 たとえば、*MyGrid* というカスタム クラスが WPF Grid クラスから継承された場合、*MyGrid* カスタム クラスのオブジェクトは記録や再生に使用できません。 Grid クラスはレイアウト目的のためにのみ存在し、機能テストとは無関係であるため、Grid オブジェクトは記録や再生に使用できません。 この結果、Grid オブジェクトはデフォルトでは公開されません。 機能テストに無関係なクラスに基づいたカスタム クラスを使用するには、カスタム クラス (この場合は *MyGrid*) を **OPT_WPF_CUSTOM_CLASSES** オプションに追加します。 これによって、記録、再生、検索、プロパティの検証など、すべてのサポートされる操作を指定したクラスに対して実行できるようになります。

1. **Silk4NET** の隣にあるドロップダウン矢印をクリックし、**オプションの編集** を選択します。 **スクリプト オプション** ダイアログ ボックスが開きます。
2. **WPF** タブをクリックします。
3. **カスタム WPF クラス名** グリッドで、記録や再生中に公開するクラスの名前を入力します。
   複数のクラス名を指定する場合にはカンマで区切ります。
4. **OK** をクリックします。

同期オプションの設定

Web アプリケーションの同期およびタイムアウトの値を指定します。

注: 以下の設定はすべて任意です。 テスト メソッドの品質が向上する場合に、これらの設定を変更してください。

1. **Silk4NET** の隣にあるドロップダウン矢印をクリックし、**オプションの編集** を選択します。 **スクリプト オプション** ダイアログ ボックスが開きます。
2. **同期** タブを選択します。
3. Web アプリケーションを準備完了状態にするための同期アルゴリズムを指定するには、**OPT_XBROWSER_SYNC_MODE** リスト ボックスからオプションを選択します。
   同期アルゴリズムは、呼び出しが可能になる状態までの待機時間を設定します。 デフォルト値は、**AJAX** に設定されています。
4. **同期除外リスト** テキスト ボックスに、除外するサービスまたは Web ページの URL 全体あるいは URL の一部を入力します。
   AJAX フレームワークやブラウザによっては、サーバーから非同期にデータを取得するために、特殊な HTTP 要求を継続して出し続けるものがあります。 これらの要求により、指定した同期タイムアウトの

期限が切れるまで同期がハングすることがあります。 この状態を回避するには、HTML 同期モードを使用するか、問題が発生する要求の URL を **同期除外リスト** 設定で指定します。

たとえば、クライアントからデータをポーリングすることによってサーバー時間を表示するウィジェットを Web アプリケーションで使用する場合は、このウィジェットのトラフィックが永続的にサーバーに送信されます。 このサービスを同期から除外するには、サービス URL を判別し、除外リストに入力します。

たとえば、以下のように入力します。

- http://example.com/syncsample/timeService
- timeService
- UICallBackServiceHandler

複数のエントリをカンマで区切って指定します。

注: アプリケーションで 1 つのサービスのみが使用されている場合、そのサービスでテストを無効にするには、サービス URL を除外リストに追加するのではなく、HTML 同期モードを使用する必要があります。

5. オブジェクトが準備完了状態になるまでの最大待機時間を指定するには、**OPT_SYNC_TIMEOUT** テキスト ボックスにミリ秒で値を指定します。

   デフォルト値は、**300000** に設定されています。
6. 再生中にオブジェクトが解決されるまでの最大待機時間を指定するには、**OPT_WAIT_RESOLVE_OBJDEF** テキスト ボックスにミリ秒で値を入力します。

   デフォルト値は、**5000** に設定されています。
7. エージェントがオブジェクトの解決を再試行するまでの最大待機時間を指定するには、**OPT_WAIT_RESOLVE_OBJDEF_RETRY** テキスト ボックスにミリ秒で値を入力します。

   デフォルト値は、**500** に設定されています。
8. **OK** をクリックします。

再生オプションの設定

テストするオブジェクトがアクティブであることを確実にしたいかどうかや、デフォルトの再生モードを上書きしたいかどうかを指定します。 再生モードは、コントロールがマウスやキーボードによって再生されるか、API で再生されるかを定義します。 デフォルト モードを使用すると、最も信頼できる結果が得られます。 他のモードを選択した場合は、すべてのコントロールが選択したモードを使用します。

1. **Silk4NET** の隣にあるドロップダウン矢印をクリックし、**オプションの編集** を選択します。 **スクリプト オプション** ダイアログ ボックスが開きます。
2. **再生** タブを選択します。 **再生オプション** ページが表示されます。
3. **OPT_REPLAY_MODE** リスト ボックスから、以下のいずれかのオプションを選択します。
   - **デフォルト**：このモードを使用すると、最も信頼できる結果が得られます。 デフォルトでは、各コントロールそれぞれが、マウスやキーボード (低レベル)、あるいは API (高レベル) モードのどちらかを使用します。 デフォルト モードを使用すると、各コントロールがコントロールの種類に応じて適切なモードが使用されます。
   - **高レベル**：このモードを使用すると、API を使用して各コントロールが再生されます。
   - **低レベル**：このモードを使用すると、マウスやキーボードを使用して各コントロールが再生されます。
4. テストするオブジェクトがアクティブであることを確実にするには、**OPT_ENSURE_ACTIVE_OBJDEF** チェック ボックスをオンにします。
5. **OK** をクリックします。

Silk4NET サンプル テスト

Silk4NET サンプル テストは、Visual Studio ソリューションにパッケージ化されており、開いて表示したり、Silk Test サンプル アプリケーションに対して実行したりできます。

サンプル アプリケーションを *http://supportline.microfocus.com/websync/SilkTest.aspx* からダウンロードします。 サンプル アプリケーションをインストールしたら、**スタート** > **プログラム** > **Silk** > **Silk Test** > **Samples** > **Silk4NET Samples** をクリックして、使用している Visual Studio のバージョンに対応するフォルダを選択し、Silk4NET サンプル テストの Visual Studio ソリューション ファイル (Silk4NET Samples.sln) を開きます。

インストールした Silk4NET サンプル アプリケーションに加え、一連の Silk4NET サンプル テストには、Web ベースの以下の Silk Test サンプル アプリケーション用のテストがいくつか含まれています。

Insurance Co. Web サイト	*http://demo.borland.com/InsuranceWebExtJS/*
Green Mountain Outpost Web	*http://demo.borland.com/gmopost/*

動的オブジェクト解決

動的オブジェクト解決により、オブジェクトを検索し識別する XPath クエリを使用した、テスト メソッドの記述が可能になります。 動的オブジェクト解決は、テスト メソッド内でオブジェクトを識別するために、Find メソッド、または FindAll メソッドを使用します。 たとえば、以下のクエリーは、特定のウィンドウの子である、キャプションが「ok」である最初のボタンを見つけます。

```
Dim okButton = window.Find("//PushButton[@caption=ok]")
```

動的オブジェクト解決がうまく機能するテスト環境の種類の例は以下のとおりです。

- GUI が変更中であるアプリケーション環境

  たとえば、メニューやダイアログ名が変更される可能性があるメニューに属するダイアログの **Check Me** チェック ボックスをテストする場合、動的オブジェクト解決を使用すると、メニューやダイアログ名に依存せずにチェック ボックスをテストすることができます。 この場合、正しいコンポーネントをテストしたことを保障するために、チェック ボックス名、ダイアログ名、メニュー名を検証することができます。
- 動的なテーブルやテキストを含んだ Web アプリケーション

  たとえば、Web ページ上のある項目をユーザーがポイントするときだけ表示されるテーブルをテストするために動的オブジェクト解決を使用すると、テーブルを表示するためにページのどの部分をクリックする必要があるかに関係なく、テーブルを見つけて利用するテスト メソッドを作成できます。
- ビューを使用する Eclipse 環境

  たとえば、ビュー コンポーネントを使用する Eclipse 環境をテストするために動的オブジェクト解決を使用すると、そのビューより前に開く必要のあるオブジェクトの階層に関係なく、ビューを識別することができます。

動的オブジェクト解決を利用する利点

動的オブジェクト解決を利用する利点は次の通りです：

- 動的オブジェクト解決は、W3C（World Wide Web Consortium）によって定義された共通の XML ベース言語である XPath クエリ言語のサブセットを使用します。
- 動的オブジェクト解決は、テストしているアプリケーションのオブジェクトのリポジトリというよりは、単一のオブジェクトを必要とします。 XPath クエリを使用すると、テスト ケースにおいて、Find コマンド (サポートされている XPath 構成子を続ける) を使用して、オブジェクトを見つけることができます。

XPath の基本概念

Silk4NET は、XPath クエリ言語のサブセットをサポートしています。 XPath に関する追加の情報については、*http://www.w3.org/TR/xpath20/* を参照してください。

基本概念

XPath 式は 現在のコンテキスト、つまり、Find メソッドを呼び出したオブジェクトの階層上における位置に依存します。 ファイル システムと同じように、すべての XPath 式は、この位置に依存します。 例：

- "//Shell" は、現在のオブジェクトに相対的なすべての階層にあるすべての Shell を見つけます。
- "Shell" は、現在のオブジェクトの直下の子であるすべての Shell を見つけます。

さらに、ある XPath 式は、コンテキストの影響を受けます。 たとえば、myWindow.Find(xPath) は、myWindow が現在のコンテキストとなります。

サポートする XPath のサブセット

Silk4NET は、XPath クエリ言語のサブセットをサポートしています。 FindAll または Find コマンド (サポートする構成子を続ける) を使用して、テスト ケースを作成します。

以下の表には、Silk4NET がサポートする構成子が一覧されています。

サポートする XPath 構成子	サンプル	説明
属性	MenuItem[@caption='abc']	現在のコンテキストの子のオブジェクト定義内で、指定した caption 属性を持つすべての MenuItem を見つけます。 次の属性がサポートされています : caption（caption index なし）、priorlabel（index なし）、windowid。
索引	MenuItem[1]	現在のコンテキストの子のうち、最初の MenuItem を見つけます。 XPath 内での索引は 1 から始まります。
論理演算子 : and、or、not、=、!=	MenuItem[not(@caption='a' or @windowid!='b') and @priorlabel='p']	
.	TestApplication.Find("//Dialog[@caption='Check Box']/./.")	Find コマンドが実行されるコンテキストを見つけます。 たとえば、このサンプルは、TestApplication.Find("//Dialog[@caption='Check Box']") と同じ意味になります。
..	Desktop.Find("//PushButton[@caption='Previous']/../PushButton[@caption='Ok']")	オブジェクトの親を見つけます。 たとえば、このサンプルは、 caption が "Previous" である PushButton と同列にある caption が "Ok" である PushButton を見つけます。
/	/Shell	現在のオブジェクトの直下の子であるすべての Shell を見つけます。 **注:**「/Shell」は、「Shell」と同じです。
/	/Shell/MenuItem	現在のオブジェクトの子であるすべての MenuItem を見つけます。
//	//Shell	現在のオブジェクトに相対的なすべての階層にあるすべての Shell を見つけます。
//	//Shell//MenuItem	現在のオブジェクトの直下の子である Shell の直接あるいは間接的な子であるすべての MenuItem を見つけます。
//	//MenuItem	現在のコンテキストの直接あるいは間接的な子であるすべての MenuItem を見つけます。

サポートする XPath 構成子	サンプル	説明
*	*[@caption='c']	現在のコンテキストの直下の子であり、かつ指定した caption を持つすべてのオブジェクトを見つけます。
*	//MenuItem/*/Shell	MenuItem の孫であるすべての Shell を見つけます。

以下の表には、Silk4NET がサポートしていない XPath 構成子が一覧されています。

サポートしない XPath 構成子	例
右辺、左辺ともに属性を指定して比較する。	PushButton[@caption = @windowid]
属性名を右辺に指定することはサポートされません。 属性名は左辺に指定する必要があります。	PushButton['abc' = @caption]
複数の XPath 式を 'and' あるいは 'or' で結合する。	PushButton [@caption = 'abc'] or .// Checkbox
複数の属性をかぎ括弧で指定する。	PushButton[@caption = 'abc] [@windowid = '123']
	(代わりに PushButton [@caption = 'abc and @windowid = '123'] を使用)
複数の索引をかぎ括弧で指定する。	PushButton[1][2]
クラスあるいはクラス名の一部にワイルドカードを含むクラス ワイルドカードを明示的に指定しない構成子。	//[@caption = 'abc']
	(代わりに //*[@caption = 'abc'] を使用)
	"//*Button[@caption='abc']"

XPath のサンプル

以下の表示では、サンプルの XPath クエリの一覧を提供し、それぞれのクエリの意味を説明します。

XPath 文字列	説明
desktop.Find("/Shell[@caption='SWT Test Application']")	指定した caption を持つ最初のトップ レベルの Shell を見つけます。
desktop.Find("//MenuItem[@caption='Control']")	指定した caption を持つ任意の階層における MenuItem を見つけます。
myShell.Find("//MenuItem[@caption! ='Control']")	指定した caption を持たない myShell の任意の子階層における MenuItem を見つけます。
myShell.Find("Menu[@caption='Control']/ MenuItem[@caption!='Control']")	指定した Menu を親とし、さらに myShell をその親とする、指定した MenuItem を見つけます。
myShell.Find("//MenuItem[@caption='Control' and @windowid='20']")	指定した caption と windowid を持つ myWindow の任意の子階層における MenuItem を見つけます。
myShell.Find("//MenuItem[@caption='Control' or @windowid='20']")	指定した caption または windowid を持つ myWindow の任意の子階層における MenuItem を見つけます。

XPath 文字列	説明
desktop.FindAll("/Shell[2]/*/PushButton")	親として 2 番目のトップ レベルの Shell を持つ任意の親を持つすべての PushButton を見つけます。
desktop.FindAll("/Shell[2]//PushButton")	直接あるいは間接的な親として 2 番めの Shell を使用するすべての PushButton を見つけます。
myBrowser.Find("//FlexApplication[1]//FlexButton[@caption='ok']")	指定したブラウザ内の最初の FlexApplication 内にある最初の FlexButton を見つけます。
myBrowser.FindAll("//td[@class='abc*']//a[@class='xyz']")	値 abc* の属性 class を持つ td 要素の直接あるいは間接的な子である、値 xyz の属性 class を持つすべてのリンク要素を見つけます。

XPath のパフォーマンス問題のトラブルシューティング

複雑なオブジェクト構造を持つアプリケーションをテストする場合、パフォーマンスの問題やスクリプトの信頼性に関する問題が発生する場合があります。 このトピックでは、 記録中に Silk4NET が自動的に生成したロケーターとは異なるロケーターを使用することによって、スクリプトのパフォーマンスを改善させる方法について説明します。

注: 一般に、複雑なロケーターを使用することは推奨しません。複雑なロケーターを使用すると、テストの信頼性を損なう可能性があります。 複雑なロケーターは、テスト アプリケーションの構造をほんの少し変更しただけで機能しなくなってしまう可能性があります。 それにもかかわらず、スクリプトのパフォーマンスが要求を満たしていない場合には、より固有のロケーターを使用することによってテストのパフォーマンスを向上できる可能性があります。

例として、MyApplication アプリケーションの要素ツリーを以下に示します。

```
Root
  Node id=1
    Leaf id=2
    Leaf id=3
    Leaf id=4
    Leaf id=5
  Node id=6
    Node id=7
      Leaf id=8
      Leaf id=9
    Node id=9
      Leaf id=10
```

以下の最適化手法のいくつかを使用して、スクリプトのパフォーマンスを改善させることができます。

- 複雑なオブジェクト構造内の要素を特定したい場合は、オブジェクト構造全体ではなく、その特定の部分だけを検索するようにします。 たとえば、サンプル ツリーの識別子 4 を持つ要素を検索する場合に Root.Find("//Leaf[@id='4']") というクエリーを使用している場合、 Root.Find("/Node[@id='1']/Leaf[@id='4']") というクエリーで置き換えます。 最初のクエリーでは、識別子 4 を持つリーフが、アプリケーションの要素ツリー全体から検索されます。 最初のリーフが見つかった時点で返されます。 2 番目のクエリーでは、識別子 1 を持つノードと識別子 6 を持つノードがある最初のレベルのノードがまず検索された後、識別子 4 を持つすべてのリーフが識別子 1 を持つノードのサブツリー内から検索されます。
- 同じ階層内の複数の項目を特定したい場合は、まずは階層を特定してからループ内で項目を特定します。 Root.FindAll("/Node[@id='1']/Leaf") というクエリーを使用している場合、次のようなループで置き換えます。

```
Public Sub Main()
  Dim node As TestObject
```

```
  node = _desktop.Find("//Node[@id='1']")
  For i As Integer = 1 To 4 Step 1
    node.Find("/Leaf[@id='"+i+"']")
  Next

End Sub
```

Locator Spy

Locator Spy を使用すると、GUI オブジェクトのキャプションや XPath ロケーター文字列を識別できます。 そして、関係する XPath ロケーター文字列や属性を、スクリプト内のメソッドにコピーできます。 また、テスト スクリプトで XPath ロケーター文字列の属性を手動で編集し、変更を **Locator Spy** で検証することができます。 **Locator Spy** を使用することで、XPath クエリー文字列が正しいことが保障されます。

サポートする属性の種類

ロケーターが作成されるとき、属性の種類はアプリケーションが使用するテクノロジ ドメインに基づいて自動的に割り当てられます。 属性の種類と値によって、ロケーターがテスト内のオブジェクトを識別する方法が決定されます。 必要に応じて、以下のいずれかの方法を使用して属性の種類を変更できます。

- 他の属性の種類と値を手動で入力する。
- **推奨属性リスト** の値を変更して、デフォルトの属性の種類に対して別の設定を指定する。

Adobe Flex アプリケーションの属性

ロケーターが作成されるとき、属性の種類はアプリケーションが使用するテクノロジ ドメインに基づいて自動的に割り当てられます。 属性の種類と値によって、ロケーターがテスト内のオブジェクトを識別する方法が決定されます。

Flex アプリケーションがサポートする属性は次のとおりです。

- automationName
- caption（automationName と同様）
- automationClassName（FlexButton など）
- className（実装クラスの完全修飾名。例：mx.controls.Button）
- automationIndex（FlexAutomation のビューでのコントロールのインデックス。例：index:1）
- index（automationIndex と同様。ただし、接頭辞はなし。例：1）
- id (コントロールの ID)
- windowId (id と同様)
- label（コントロールのラベル）
- すべての動的ロケーター属性

注: 属性の名前は、大文字小文字が区別されます。 ロケーター属性は、ワイルドカード ? および * をサポートしています。

動的ロケーター属性の詳細については、「動的ロケーター属性」を参照してください。

Java AWT/Swing アプリケーションの属性

ロケーターが作成されるとき、属性の種類はアプリケーションが使用するテクノロジ ドメインに基づいて自動的に割り当てられます。 属性の種類と値によって、ロケーターがテスト内のオブジェクトを識別する方法が決定されます。

Java AWT/Swing でサポートされる属性には以下のものがあります。

- caption
- priorlabel : 隣接するラベル フィールドのテキストによってテキスト入力フィールドを識別します。 通常、フォームのすべての入力フィールドに、入力の目的を説明するラベルがあります。 caption のないコントロールの場合、自動的に属性 **priorlabel** がロケーターに使用されます。 コントロールの **priorlabel** 値 (テキスト入力フィールドなど) には、コントロールの左側または上にある最も近いラベルの caption が使用されます。
- name
- accessibleName
- *Swing のみ* : すべてのカスタム オブジェクトの定義属性は、ウィジェットに SetClientProperty("propertyName", "propertyValue") で設定されます。

注: 属性の名前は、大文字小文字が区別されます。 ロケーター属性は、ワイルドカード ? および * をサポートしています。

Java SWT アプリケーションの属性

ロケーターが作成されるとき、属性の種類はアプリケーションが使用するテクノロジ ドメインに基づいて自動的に割り当てられます。 属性の種類と値によって、ロケーターがテスト内のオブジェクトを識別する方法が決定されます。

Java SWT がサポートする属性は次のとおりです。

- caption
- すべてのカスタム オブジェクト定義属性

注: 属性の名前は、大文字小文字が区別されます。 ロケーター属性は、ワイルドカード ? および * をサポートしています。

MSUIA アプリケーションの属性

注: MSUIA は廃止されます。 新しいテストでは、WPF テクノロジ ドメインを使用してください。

ロケーターが作成されるとき、属性の種類はアプリケーションが使用するテクノロジ ドメインに基づいて自動的に割り当てられます。 属性の種類と値によって、ロケーターがテスト内のオブジェクトを識別する方法が決定されます。

MSUIA アプリケーションがサポートする属性は次のとおりです。

- acceleratorkey
- accesskey
- automationid
- classname
- controltype
- frameworkid
- haskeyboardfocus
- helptext
- isenabled
- isoffscreen
- ispassword
- caption
- name
- nativewindowhandle
- orientation

注: 属性の名前は、大文字小文字が区別されます。 ロケーター属性は、ワイルドカード ? および * をサポートしています。

Rumba コントロールを識別するためのロケーター属性

ロケーターが作成されるとき、属性の種類はアプリケーションが使用するテクノロジ ドメインに基づいて自動的に割り当てられます。 属性の種類と値によって、ロケーターがテスト内のオブジェクトを識別する方法が決定されます。 サポートされている属性は次のとおりです。

caption	コントロールが表示するテキスト。
priorlabel	フォームの入力フィールドには通常入力の目的を説明するラベルがあるため、**priorlabel** の目的は隣接するラベル フィールド **RumbaLabel** のテキストによってテキスト入力フィールド **RumbaTextField** を識別することです。 テキスト フィールドの同じ行の直前にラベルがない場合、または右側のラベルが左側のラベルよりテキスト フィールドに近い場合、テキスト フィールドの右側にあるラベルが使用されます。
StartRow	この属性は記録されていませんが、手動でロケーターに追加することができます。**StartRow** を使用して、この行で始まるテキスト入力フィールド、**RumbaTextField** を識別します。
StartColumn	この属性は記録されていませんが、手動でロケーターに追加することができます。**StartColumn** を使用して、この列で始まるテキスト入力フィールド、**RumbaTextField** を識別します。
すべての動的ロケーター属性。	動的ロケーター属性の詳細については、「動的ロケーター属性」を参照してください。

注: 属性の名前は、大文字小文字が区別されます。 ロケーター属性は、ワイルドカード ? および * をサポートしています。

SAP アプリケーションの属性

ロケーターが作成されるとき、属性の種類はアプリケーションが使用するテクノロジ ドメインに基づいて自動的に割り当てられます。 属性の種類と値によって、ロケーターがテスト内のオブジェクトを識別する方法が決定されます。

SAP がサポートする属性は次のとおりです。

- automationId
- caption

注: 属性の名前は、大文字小文字が区別されます。 ロケーター属性は、ワイルドカード ? および * をサポートしています。

Silverlight コントロールを識別するためのロケーター属性

Silverlight コントロールでサポートされているロケーター属性は次のとおりです。

- *automationId*
- *caption*
- *className*
- *name*
- すべての動的ロケーター属性

注: 属性の名前は、大文字小文字が区別されます。 ロケーター属性は、ワイルドカード ? および * をサポートしています。

動的ロケーター属性の詳細については、「動的ロケーター属性」を参照してください。

Silverlight スクリプト内のコンポーネントを識別するために、*automationId*、*caption*、*className*、*name*、または任意の動的ロケーター属性を指定できます。 *automationId* はアプリケーション開発者が設定します。 たとえば、*automationId* を持つロケーターは、以下のようになります : //SLButton[@automationId="okButton"]

automationId は一般に非常に有用で安定した属性であるため、使用することを推奨します。

属性の種類	説明	例
automationId	テスト対象アプリケーションの開発者によって設定される識別子。 Visual Studio デザイナは、デザイナ上で作成されたすべてのコントロールに自動的に *automationId* を割り当てます。 アプリケーション開発者は、アプリケーションのコード上でコントロールを識別するために、この ID を使用します。	//SLButton[@automationId="okButton"]
caption	コントロールが表示するテキスト。 複数の言語にローカライズされたアプリケーションをテストする場合、*caption* の代わりに *automationId* や *name* 属性を使用することを推奨します。	//SLButton[@caption="Ok"]
className	Silverlight コントロールの .NET 単純クラス名 (名前空間なし)。 *className* 属性を使用すると、Silk4NET が解決する標準 Silverlight コントロールから派生したカスタム コントロールを識別するのに役立ちます。	//SLButton[@className='MyCustomButton']
name	コントロールの名前。 テスト対象アプリケーションの開発者によって設定されます。	//SLButton[@name="okButton"]

注目: XAML コードの *name* 属性は、ロケーター属性 *name* ではなく、ロケーター属性 *automationId* にマップされます。

Silk4NET は、*automationId*、*name*、*caption*、*className* 属性をこの表に示した順番に使用して Silverlight コントロールのロケーターを記録時に作成します。 たとえば、コントロールが *automationId* と *name* を持つ場合、*automationId* が固有の場合は Silk4NET がロケーターを作成する際に使用されます。

以下の表は、アプリケーション開発者がテキスト「Ok」を持つ Silverlight ボタンをアプリケーションの XAML コードに定義する方法を示しています。

オブジェクトの XAML コード	Silk Test からオブジェクトを検索するためのロケーター
<Button>Ok</Button>	//SLButton[@caption="Ok"]
<Button Name="okButton">Ok</Button>	//SLButton[@automationId="okButton"]
<Button AutomationProperties.AutomationId="okButton">Ok</Button>	//SLButton[@automationId="okButton"]
<Button AutomationProperties.Name="okButton">Ok</Button>	//SLButton[@name="okButton"]

Web アプリケーションの属性

ロケーターが作成されるとき、属性の種類はアプリケーションが使用するテクノロジ ドメインに基づいて自動的に割り当てられます。 属性の種類と値によって、ロケーターがテスト内のオブジェクトを識別する方法が決定されます。

Web アプリケーションがサポートする属性は次のとおりです。

- caption (次のワイルドカードをサポート：? および *)
- すべての DOM 属性 (次のワイルドカードをサポート：? および *)

注: 各ブラウザによって、空のスペースの処理に違いがあります。 この結果、「textContent」および「innerText」属性は正規化されています。 空のスペースのあとに別の空のスペースが続く場合、空のスペースはスキップされるか、または 1 文字の空白で置き換えられます。 空のスペースとは、検出されたスペース、キャリッジ リターン、改行、タブのことです。 また、このような値に一致するものも正規化されます。 例：

```
<a>abc
abc</a>
```

以下のロケーターを使用します。

```
//A[@innerText='abc  abc']
```

Windows API ベースのクライアント/サーバー アプリケーションの属性

ロケーターが作成されるとき、属性の種類はアプリケーションが使用するテクノロジ ドメインに基づいて自動的に割り当てられます。 属性の種類と値によって、ロケーターがテスト内のオブジェクトを識別する方法が決定されます。

Windows API ベースのクライアント/サーバー アプリケーションがサポートする属性は次のとおりです。

- caption
- windowid
- priorlabel : 隣接するラベル フィールドのテキストによってテキスト入力フィールドを識別します。 通常、フォームのすべての入力フィールドに、入力の目的を説明するラベルがあります。 caption のないコントロールの場合、自動的に属性 **priorlabel** がロケーターに使用されます。 コントロールの **priorlabel** 値 (テキスト ボックスなど) には、コントロールの左側または上にある最も近いラベルの caption が使用されます。

注: 属性の名前は、大文字小文字が区別されます。 ロケーター属性は、ワイルドカード ? および * をサポートしています。

Windows Forms アプリケーションの属性

ロケーターが作成されるとき、属性の種類はアプリケーションが使用するテクノロジ ドメインに基づいて自動的に割り当てられます。 属性の種類と値によって、ロケーターがテスト内のオブジェクトを識別する方法が決定されます。

Windows Forms アプリケーションがサポートする属性は次のとおりです。

- automationid
- caption
- windowid
- priorlabel（caption のないコントロールの場合、自動的に priorlabel が caption として使用されます。 caption のあるコントロールの場合、caption を使う方が簡単な場合があります。）

注: 属性の名前は、大文字小文字が区別されます。 ロケーター属性は、ワイルドカード ? および * をサポートしています。

Windows Presentation Foundation (WPF) アプリケーションの属性

WPF アプリケーションがサポートする属性は次のとおりです。

- *automationId*
- *caption*
- *className*
- *name*
- すべての動的ロケーター属性。

注: 属性の名前は、大文字小文字が区別されます。 ロケーター属性は、ワイルドカード ? および * をサポートしています。

動的ロケーター属性の詳細については、「動的ロケーター属性」を参照してください。

オブジェクト解決

WPF スクリプト内のコンポーネントを識別するために、*automationId*、*caption*、*className*、あるいは *name* を指定できます。 アプリケーション中の要素に指定された name が利用可能な場合、ロケーターの *automationId* 属性として使用されます。 この結果、多くのオブジェクトは、この属性のみを使用して一意に識別できます。 たとえば、*automationId* を持つロケーターは、以下のようになります：// WPFButton[@automationId='okButton']"

automationId や他の属性を定義した場合、再生中に *automationId* だけが使用されます。 *automationId* が定義されていない場合には、コンポーネントを解決するのに *name* が使用されます。 *name* も *automationId* もどちらも定義されていない場合には、*caption* 値が使用されます。 caption が定義されていない場合は、*className* が使用されます。 *automationId* は非常に役立つプロパティであるため、使用することを推奨します。

属性の種類	説明	例
automationId	テスト アプリケーションの開発者によって提供された ID	//WPFButton[@automationId='okButton']"
name	コントロールの名前。 Visual Studio デザイナは、デザイナ上で作成されたすべてのコントロールに自動的に名前を割り当てます。 アプリケーション開発者は、アプリケーションのコード上でコントロールを識別するために、この名前を使用します。	//WPFButton[@name='okButton']"
caption	コントロールが表示するテキスト。 複数の言語にローカライズされたアプリケーションをテストする場合、caption の代わりに automationId や name 属性を使用することを推奨します。	//WPFButton[@automationId='Ok']"
className	WPF の .NET 単純クラス名 (名前空間なし)。 クラス名属性を使用すると、Silk4NET が解決する標準 WPF コントロールから派生したカスタム コントロールを識別するのに役立ちます。	//WPFButton[@className='MyCustomButton']"

Silk4NET は、*automationId*、*name*、*caption*、*className* 属性をこの表に示した順番に使用して WPF コントロールのロケーターを記録時に作成します。 たとえば、コントロールが *automationId* と *name* を持つ場合、Silk4NET がロケーターを作成する際には *automationId* が使用されます。

以下の例では、アプリケーション開発者がアプリケーションの WPF ボタンに対して *name* と *automationId* を XAML コードに定義する方法を示します。

```
<Button  Name="okButton"  AutomationProperties.AutomationId="okButton"
Click="okButton_Click">Ok</Button>
```

動的ロケーター属性

再生中にコントロールを識別するために、事前に定義されたロケーター属性のセット (*caption* や *automationId* など。テクノロジ ドメインに依存します) をロケーターに使用できます。 しかし、動的プロパティを含む、コントロールのすべての属性をロケーター属性として使用することもできます。 特定のコントロールで使用可能なプロパティのリストを取得するには、GetPropertyList メソッドを使用します。 返されたプロパティはすべて、ロケーターを使用してコントロールを識別するのに使用できます。

注: 特定のプロパティの実際の値を取得するには、GetProperty メソッドを使用します。 この値はロケーターで使用できます。

例

WPF アプリケーションのダイアログ ボックスにあるボタンを識別する場合、以下のように入力します。

```
Dim  button  =  dialog.Find("//WPFButton[@IsDefault=true]")
```

または

```
Dim  button  =  dialog.WPFButton("@IsDefault=true")
```

これが機能するのは、Silk Test Workbench により WPF ボタン コントロールの IsDefault というプロパティが公開されるためです。

例

WPF アプリケーションのフォント サイズ 12 のボタンを識別する場合、以下のように入力します。

```
Dim  button  =  dialog.Find("//WPFButton[@FontSize=12]")
```

または

```
Dim  button  =  dialog.WPFButton("@FontSize=12")
```

これが機能するのは、テスト対象アプリケーションの基になるコントロール (この場合、WPF ボタン) が FontSize というプロパティを持つためです。

テストの拡張

このセクションでは、テストの拡張方法について説明します。

Windows DLL の呼び出し

このセクションでは、DLL を呼び出す方法について説明します。 DLL は Open Agent のプロセス内から、または AUT (テスト対象アプリケーション) から呼び出すことができます。 これにより、テスト スクリプト内の既存のネイティブ DLL を再利用できます。

Open Agent 内の DLL 呼び出しは通常、AUT 内の UI コントロールと対話しないグローバル関数を呼び出す場合に使用されます。

AUT 内の DLL 呼び出しは通常、アプリケーションの UI コントロールと対話する関数を呼び出す場合に使用されます。 これにより、Silk4NET は再生中に DLL 呼び出しを自動的に同期できます。

注: 32 ビット アプリケーションでは 32 ビット DLL を、64 ビット アプリケーションでは 64 ビット DLL を呼び出すことができます。 Open Agent は 32 ビットと 64 ビットの両方の DLL を実行できます。

注: .NET Framework では、P/Invoke という DLL 呼び出しも組み込みでサポートされています。 P/Invoke を Visual Basic スクリプト内で使用すると、このスクリプトを実行するプロセス内で DLL 関数を呼び出すことができます。 ただし、AUT では Silk Test Workbench を使用して DLL 関数を呼び出すことができる一方で、自動同期は行われません。

注: DLL を呼び出すには、C インターフェイスを使用する必要があります。 同様に .dll というファイル拡張子の付いた .NET アセンブリを呼び出す場合は、DLL 呼び出し機能を使用しないで、.NET スクリプト内でアセンブリへの参照を追加します。

スクリプトからの Windows DLL の呼び出し

DLL の宣言を開始するには、DLL 属性を持つインターフェイスを使用します。 宣言の構文は次のとおりです。

dllname	スクリプトから呼び出す関数が含まれた DLL ファイルの完全パスの名前。 DLL パス内の環境変数は自動的に解決されます。 パス内のバックスラッシュは 2 重 (¥¥) にする必要はありません。単一のバックスラッシュ (¥) を使用してください。
DllInterfaceName	スクリプト内で DLL と対話するために使用される識別子。
FunctionDeclaration	呼び出そうとしている DLL 関数の関数宣言。

DLL 関数の宣言構文

DLL 関数の宣言は、一般に以下の形式を取ります。

戻り値のない関数の場合、宣言の形式は以下のとおりです、

return-type	戻り値のデータ型。
function-name	関数の名前。
arg-list	関数に渡される引数のリスト。

リストは以下のように指定します。

data-type	引数のデータ型。
identifier	引数の名前。

DLL 関数への引数の受け渡し

DLL 関数は C で記述されているため、これらの関数に渡す引数には適切な C データ型を指定する必要があります。 次のデータ型がサポートされます。

次のデータ型を持つ引数または戻り値には、このデータ型を使用します。

- int
- INT
- long
- LONG
- DWORD
- BOOL
- WPARAM
- HWND

の 型は、4 バイト値を持つすべての DLL 引数に対して有効です。

C データ型 long および int64 を持つ引数または戻り値には、このデータ型を使用します。 の 型は、8 バイト値を持つすべての DLL 引数に対して有効です。

C データ型 short および WORD を持つ引数または戻り値には、このデータ型を使用します。 の型は、2 バイト値を持つすべての DLL 引数に対して有効です。

C データ型 bool を持つ引数または戻り値には、このデータ型を使用します。

String C で String となる引数または戻り値には、このデータ型を使用します。

C データ型 double を持つ引数または戻り値には、このデータ型を使用します。

C データ型 RECT を持つ引数には、このデータ型を使用します。 は戻り値として使用できません。

C データ型 POINT を持つ引数には、このデータ型を使用します。 POINT は戻り値として使用できません。

C データ型 HWND を持つ引数には、このデータ型を使用します。 TestObject は戻り値として使用できませんが、戻り値型として Integer を持つ HWND を戻す DLL 関数を宣言できます。

注: 渡された TestObject は インターフェイスを実装して、DLL 関数に渡される TestObject のウィンドウ ハンドルを Silk4NET が 判別できるようにする必要があります。 そうしないと、この DLL 関数を呼び出すときに、例外がスローされます。

List ユーザー定義の C 構造体の配列には、このデータ型を使用します。 List は戻り値として使用できません。

注: List をパラメーターとして使用する場合は、渡されるリストに、戻される内容を保持できるだけのサイズを確保する必要があります。

注: C 構造体は List で表すことができます。この場合、すべてのリスト要素は構造体のメンバに対応しています。 最初の構造体メンバは、リスト内の最初の要素で表されます。2 番目の構造体メンバは、リスト内の 2 番目の要素で表されます (以下同様)。

注: DLL 関数に渡す引数の前には、いずれかの データ型を配置する必要があります。

DLL 関数への文字列引数の受け渡し

DLL 関数に渡している文字列、または DLL 関数から戻される文字列は、デフォルトでは Unicode Strings として処理されます。 DLL 関数に ANSI String 引数が必要な場合は、DllFunctionOptions 属性の CharacterSet プロパティーを使用します。

例

```
<Dll( "user32.dll" )> Public Interface IUserDll32Functions
  <DllFunctionOptions(CharacterSet:=CharacterSet.Ansi)> Function
SendMessageA( _
     ByVal obj As TestObject, ByVal message As Integer , ByVal wParam
As Integer , ByRef lParam As String ) As Integer
End Interface
```

DLL 呼び出しから String を ByRef 引数 として戻した場合、String のサイズが 256 文字以下であれば、デフォルトの動作に従います。 戻される String が 256 文字を超えている場合は、作成された String を保持できるだけの長さを持つ、Visual Basic String を渡します。

例

1024 個の空白文字を含む String を作成するには、以下のコードを使用します。

```
Dim longEmptyString = New String ( " "c , 1024 )
```

この String を ByRef 引数として DLL 関数に渡します。すると、この DLL 関数は最大 1024 文字の String を戻します。

関数の戻り値として DLL から String が戻される場合、DLL は DLL 関数 FreeDllMemory を実装し、DLL 関数から戻される C String ポインターを受け入れて、以前に割り当てられたメモリーを解放する必要があります。 このような関数が存在しない場合、メモリーはリークされます。

DLL 名のエイリアス設定

DLL 関数に、Visual Basic の予約語と同じ名前が付いている場合、または DLL 関数に名前でなく序数が付いている場合は、宣言内でこの関数の名前を変更し、エイリアス ステートメントを使用して、宣言した名前と実際の名前をマッピングする必要があります。

例

たとえば、Exit ステートメントは Visual Basic コンパイラーで予約されています。 したがって、関数 exit を呼び出すには、次のようにその関数を別の名前で宣言し、エイリアス ステートメントを追加する必要があります。

```
<Dll("mydll.dll")> Public Interface IMyDllFunctions
  <DllFunctionOptions(Alias:="exit")> Sub MyExit()
End Interface
```

DLL 関数呼び出しの表記規則

DLL 関数を呼び出す場合は、次に示す呼び出し規則がサポートされています。

- __stdcall
- __cdecl

DLL 関数を呼び出す場合は、*__stdcall* 呼び出し規則がデフォルトで使用されます。 この呼び出し規則は、すべての Windows API DLL 関数で使用されます。

DLL 関数の呼び出し規則を変更するには、DllFunctionOptions 属性の CallingConvention プロパティーを使用します。

例

次のコード例では、*__decl* 呼び出し規則を使用して DLL 関数を宣言します。

```
<Dll("msvcrt.dll")> Public Interface IMsVisualCRuntime
  <DllFunctionOptions(CallingConvention:=CallingConvention.Cdecl)>
Function cos(ByVal input As Double) As Double
End Interface
```

Windows ユーザー補助

このセクションでは、Windows ユーザー補助 (ユーザー補助) を使用して、クラス レベルでオブジェクトを簡単に認識する方法について説明します。

ユーザー補助の使用

Win32 では、ジェネリック コントロールとして認識されるコントロールにユーザー補助 サポートが使用されます。 Win32 は、コントロールを特定すると、ユーザー補助オブジェクトをコントロールのすべてのユーザー補助の子とともに取得しようとします。

ユーザー補助によって返されるオブジェクトは、AccessibleControl、Button、CheckBox のいずれかのクラスになります。 Button および Checkbox は、そのクラス用に定義されたメソッドとプロパティの通常セットをサポートするので個別に扱われます。 ユーザー補助によって返されるすべてのジェネリック オブジェクトの場合、クラスは AccessibleControl です。

例

ユーザー補助が有効になる前、アプリケーションのコントロール階層が次のようになっていたとします。

- コントロール
  - コントロール
- ボタン

ユーザー補助を有効にすると、階層は次のように変わります。

- コントロール
  - コントロール
    - ユーザー補助コントロール
    - ユーザー補助コントロール
      - ボタン
- ボタン

ユーザー補助の有効化

Win32 アプリケーションをテストしているときに、 でオブジェクトを認識できない場合は、最初にユーザー補助を有効にする必要があります。 ユーザー補助は、オブジェクトの認識機能をクラス レベルで強化するためのものです。

のユーザー補助を有効にするには、以下の手順を実行します。

1. をクリックします。 ダイアログ ボックスが表示されます。
2. **詳細設定** をクリックします。
3. **Microsoft ユーザー補助を使用する** オプションを選択します。 ユーザー補助が有効になります。

動的呼び出し

動的呼び出しを使用すると、テスト対象アプリケーション内のコントロールの実際のインスタンスに関して、メソッドの呼び出し、プロパティーの取得、またはプロパティーの設定を直接実行できます。 また、このコントロールの API で使用できないメソッドおよびプロパティーも呼び出すことができます。 動的呼び出しは、作業しているカスタム コントロールと対話するために必要な機能が、Silk Test Workbench API を通して公開されていない場合に特に便利です。

注: 動的呼び出しは、スクリプトで使用できます。 ビジュアル テストでは使用できません。

オブジェクトの動的メソッドは Invoke メソッドを使用して呼び出します。 コントロールでサポートされている動的メソッドのリストを取得するには、GetDynamicMethodList メソッドを使用します。

オブジェクトの複数の動的メソッドは InvokeMethods メソッドを使用して呼び出します。 コントロールでサポートされている動的メソッドのリストを取得するには、GetDynamicMethodList メソッドを使用します。

動的プロパティの取得には GetProperty メソッドを、動的プロパティの設定には SetProperty メソッドを使用します。 コントロールでサポートされている動的プロパティのリストを取得するには、GetPropertyList メソッドを使用します。

たとえば、テスト対象アプリケーション内のコントロールの実際のインスタンスに関して、タイトルを String 型の入力パラメータとして設定する必要がある SetTitle というメソッドを呼び出すには、次のように入力します：

```
control.Invoke("SetTitle", "my  new  title")
```

注: 通常、ほとんどのプロパティは読み取り専用で、設定できません。

注: ほとんどのテクノロジー ドメインでは、メソッドを呼び出してプロパティーを取得する場合、Reflection を使用します。

テキスト解決のサポート

テキスト解決メソッドでは、オブジェクト解決で識別できない、高度にカスタマイズされたコントロールを含むテスト アプリケーションを便利に操作できます。 座標ベースのクリックの代わりにテキスト クリックを使用し、コントロール内に指定されたテキスト文字列をクリックできます。

たとえば、次の表の 2 行目の最初のセルを選択することをシミュレートできます。

CustomerName	FirstOrder	ID	IsActive	CreditCard
Bob Villa	01.01.2008	0	☑	MasterCard
Brian Miller	02.01.2008	1	☐	Visa
Caral Rudd	03.01.2008	2	☑	American Ex...
Dan Rundgren	04.01.2008	3	☐	MasterCard
Devie Yingstein	05.01.2008	4	☑	Visa

セルのテキストを指定すると、次のコード行が生成されます。

テキスト解決メソッドは、次のテクノロジ ドメインでサポートされます。

- Win32
- WPF

- Windows Forms
- Java SWT と Eclipse
- Java AWT/Swing

  注: Java アプレット、および Java バージョンが 1.6.10 以下である Swing アプリケーションの場合、テキスト解決は追加設定なしでサポートされます。 Java バージョンが 1.6.10 以上の Swing アプリケーションの場合は、アプリケーションの起動時に次のコマンドライン要素を追加する必要があります。

  ```
  -Dsun.java2d.d3d=false
  ```

  例：

  ```
  javaw.exe  -Dsun.java2d.d3d=false  -jar  mySwingApplication.jar
  ```
- xBrowser

テキスト解決メソッド

次のメソッドにより、コントロールのテキストを操作できます。

TextCapture	コントロール内のテキストを返します。 子コントロールのテキストも返します。
TextClick	コントロール内の指定テキストをクリックします。
TextRectangle	コントロール内の特定テキストの矩形、またはコントロールの領域を返します。
TextExists	コントロール内またはコントロールの領域内に特定テキストが存在するかどうかを判断します。

テキスト クリックの記録

テキスト クリックの記録を有効にすると、 は、相対座標でクリックを記録するのではなく、TextClick メソッドを記録します。 通常の座標ベースのクリックよりも TextClick 記録の方が結果が良いコントロールには、この方法を使用します。 テキスト クリックが記録されない場合でも、コントロール用にテキスト クリックをスクリプトに挿入できます。

TextClick 操作を記録しない場合は、テキスト クリックの記録をオフに切り替えて通常のクリックを記録できます。

テキスト解決メソッドでは、部分的に一致する単語よりも完全に一致する単語が優先されます。 では、完全に一致する単語の前に部分的に一致する単語が画面に表示されていても、部分的に一致する単語よりも完全に一致した単語の出現が先に解決されます。 完全に一致する単語がない場合は、部分的に一致する単語が画面に表示される順序で使用されます。

> **例**
>
> ユーザー インターフェイスには、テキスト「*the hostname is the name of the host*」が表示されているとします。 画面には「*hostname*」が「*host*」より前に表示されていますが、次のコードでは「*hostname*」ではなく「*host*」がクリックされます。 次のコードでは 2 回目の出現が指定され、単語「*hostname*」の部分文字列「*host*」がクリックされます。

カスタム コントロール

Silk4NET が専用サポートを提供していないカスタム コントロールに対応するカスタム クラスを作成できます。 カスタム クラスを作成すると、以下の利点があります。

- スクリプト のロケーターが効率化されます。
- カスタム コントロールと対話するための再利用可能コードを簡単に記述できます。

例

カスタム タブ コントロールが Silk4NET で汎用クラス Control として認識されるとします。 Silk4NET のカスタム コントロール サポートを使用すると、以下の利点があります。

カスタム コントロール クラス名をロケーターで使用できるため、オブジェクトの認識率が高まります。

複数のオブジェクトが Control として認識されることがあります。 ロケーターには、特定のオブジェクトを識別するためのインデックスが必要です。 たとえば、オブジェクトはロケーター //Control[13] を使用して識別できます。 このコントロールのカスタム クラス (クラス MyTabControl など) を作成する場合は、ロケーター //MyTabControl を使用できます。 カスタム クラスを作成することによって、テスト対象アプリケーションが変更された場合にオブジェクト識別子が変わりやすい、大きな数字のインデックスを使用する必要がなくなります。

スクリプトのコントロールに対して再利用可能な再生操作を実装できます。

カスタム クラスを使用していない場合は、カスタム タブ コントロール内のタブを選択するときに、以下のようなコードを記述できます。

```
tabControl.TextClick("<TabName>")
```

カスタム クラスを使用している場合は、ユーザー インターフェイスでカスタム コントロールを指定するときに生成されるカスタム クラスに以下のコードを追加して、タブ選択動作をメソッドにカプセル化することができます。

```
Public Sub SelectTab(tabText As String)
   TextClick(tabText)
End Sub
```

カスタム クラスの外観は、以下のようになります。

```
Public Class MyTabControl
   Inherits Control

   Public Sub New(objectHandle As ObjectHandle)
      MyBase.New(objectHandle)
   End Sub

   Public Sub SelectTab(tabText As String)
      TextClick(tabText)
   End Sub

End Class
```

新規に作成されたメソッド SelectTab は以下のようにスクリプト内で使用できます。

```
tabControl.SelectTab("<TabName>")
```

カスタム コントロールのサポート

Silk4NET が専用サポートを提供していないカスタム コントロールに対応するカスタム クラスを作成するには：

1. **Silk4NET** > **カスタム コントロールの管理** をクリックします。 **カスタム コントロールの管理** ダイアログ ボックスが開きます。
2. **Silk4NET カスタム コントロール コードのディレクトリ** フィールドで、名前を入力するか **参照** をクリックして カスタム コントロールを保持する ディレクトリ を選択します。
3. 新しいカスタム クラスを作成するテクノロジ ドメインのタブをクリックします。
4. **追加** をクリックします。
5. 次のいずれかをクリックします。
   - **新しいカスタム コントロールの識別** をクリックし、**オブジェクトの識別** ダイアログ ボックスを使ってアプリケーション内のカスタム コントロールを直接選択します。
   - **新しいカスタム コントロールの追加** をクリックし、カスタム コントロールを手動でリストに追加します。

   新しい行がカスタム コントロールのリストに追加されます。
6. **Silk Test 基本クラス** 列で、クラスの取得元となる既存の基本クラスを選択します。
   このクラスは、ご使用のカスタム コントロールのタイプに最も一致率が高くなければなりません。
7. **Silk Test クラス** 列で、 をクリックし、クラスの参照に使用する名前を入力します。
   この名前は、ロケーターに表示されます。 たとえば、//Control[13] でなく //MyTabControl を入力します。

   注: 有効なクラスを追加すると、そのクラスは **Silk Test 基本クラス** リストで使用できるようになります。 追加したクラスは、基本クラスとして再使用できます。
8. **カスタム コントロール クラス名** 列に、マップしているクラスの完全修飾クラス名を入力します。
   たとえば、Infragistics.Win.UltraWinTabControl.UltraTabControl のように入力します。
9. *Win32* アプリケーションの場合のみ： **クラス マッピング** 列で クラス マッピングを有効にすると、カスタム コントロール クラスの名前を標準 Silk Test クラスの名前にマップすることができます。
   カスタム コントロール クラスを標準 Silk Test クラスにマップすると、標準 Silk Test クラスでサポートされる機能をテストで使用することができます。
10. *Win32* アプリケーションの場合のみ : **クラス マッピング** 列で、クラス マッピングを true に設定して、Silk Test クラスの名前を Silk Test 基本クラスの名前に自動的にマップします。
    カスタム クラスを標準クラスにマップすると、標準クラスでサポートされている機能を使用できます。
11. **OK** をクリックします。
12. スクリプトの場合のみ：
    a) クラスに、カスタム コントロール用のカスタム メソッドおよびプロパティを追加します。
    b) スクリプト内で新しいクラスのカスタム メソッドおよびプロパティを使用します。

    注: カスタム メソッドおよびプロパティは記録されません。

注: スクリプト ファイル内のカスタム クラスまたは基本クラスの名前を変更しないでください。 スクリプト内に生成されたクラスを変更した場合、予期しない動作を起こすことがあります。 カスタム クラスにプロパティおよびメソッドを追加する場合にのみスクリプトを使用してください。 それ以外の変更をカスタム クラスに加える場合は **カスタム コントロールの管理** ダイアログ ボックスを使用してください。

カスタム コントロール オプション

Silk4NET > **カスタム コントロールの管理**

Silk4NET カスタム コントロール コードのディレクトリ で、新しいカスタム クラスが生成される ディレクトリ を定義します。

次の **カスタム コントロール** オプションが使用できます。

オプション	説明
Silk Test 基本クラス	自分のクラスの派生元として使用する既存の基本クラスを選択します。 このクラスは、ご使用のカスタム コントロールのタイプに最も一致率が高くなければなりません。
Silk Test クラス	クラスの参照に使用する名前を入力します。 この名前は、ロケーターに表示されます。 たとえば、//Control[13] でなく //MyTabControl を入力します。
カスタム コントロールのクラス名	マッピングされているクラスの完全修飾クラス名を入力します。 たとえば、Infragistics.Win.UltraWinTabControl.UltraTabControl のように入力します。
クラス マッピング	このオプションは Win32 アプリケーションの場合のみ使用できます。 Silk Test クラスの名前を Silk Test 基本クラスの名前にマップする場合に true を選択します。

注: 有効なクラスを追加すると、そのクラスは **Silk Test 基本クラス** リストで使用できるようになります。 追加したクラスは、基本クラスとして再使用できます。

xBrowser のよくある質問

このセクションでは、Web アプリケーションをテストするときに発生することがある質問のコレクションを示します。

要素のテキストに使用されるフォント タイプの確認方法

属性名を「:」で区切ると、DOM 要素の currentStyle 属性のすべての属性にアクセスできます。

Windows Internet Explorer 8 以前	wDomElement.GetProperty("currentStyle:fontName")
Windows Internet Explorer 9 またはそれ以降および Mozilla Firefox などの他のすべてのブラウザ	wDomElement.GetProperty("currentStyle:font-name")

textContents、innerText、および innerHtml の違い

- textContents は、書式設定のみを目的とする要素およびその子要素に含まれるすべてのテキストです。
- innerText は、要素およびその子要素に含まれるすべてのテキストを返します。
- innerHtml は、要素に含まれるすべてのテキスト（html タグも含む）を返します。

以下の html コードについて検討します。

```
<div id="mylinks">
  This is my <b>link collection</b>:
  <ul>
    <li><a href="www.borland.com">Bye bye <b>Borland</b> </a></li>
    <li><a href="www.microfocus.com">Welcome to <b>Micro Focus</b></a></li>
  </ul>
</div>
```

以下の表に、返されるプロパティの詳細を示します。

コード	返される値
browser.DomElement("//div[@id='mylinks']").GetProperty("textContents")	This is my link collection:
browser.DomElement("//div[@id='mylinks']").GetProperty("innerText")	This is my link collection:Bye bye Borland Welcome to Micro Focus
browser.DomElement("//div[@id='mylinks']").GetProperty("innerHtml")	This is my link collection: Bye bye Borland Welcome to Micro Focus

innerText をカスタム クラス属性として構成したが、ロケーターで使用されない

ロケーター文字列に使用する属性には最大長があります。 InnerText は長くなりすぎる傾向があり、ロケーターで使用できない場合があります。 可能な場合は、textContents を代わりに使用してください。

クロス ブラウザ スクリプトの作成時に必要な処置

クロスブラウザ スクリプトを作成する場合は、以下の 1 つまたは複数の問題に遭遇する場合があります。

- 属性値が異なる。 たとえば、 Windows Internet Explorer の色が "# FF0000" として、 Mozilla Firefox の色が "rgb(255,0,0)" として返されます。
- 属性名が異なる。 たとえば、 Windows Internet Explorer 8 以前のバージョンではフォント サイズ属性が "fontSize" と呼ばれ、Windows Internet Explorer 9 以降および Mozilla Firefox などの他のすべてのブラウザでは "font-size" と呼ばれます。
- 一部のフレームワークで 異なる DOM ツリーがレンダリングされることがある

現在使用しているブラウザを確かめるには

BrowserApplication クラスには、ブラウザの種類を返すプロパティ "browsertype" があります。 このプロパティをロケーターに追加することで、どのブラウザに一致させるかを定義できます。

新機能、サポート対象のプラットフォームとバージョン、既知の問題、および回避策の詳細については、『*Silk Test* リリース ノート』(*http://supportline.microfocus.com/productdoc.aspx* で入手可能) を参照してください。

使用例

ブラウザの種類を取得するには、次のコードをロケーターに入力します。

```
browserApplication.GetProperty("browsertype")
```

また、BrowserWindow には、現在のウィンドウのユーザー エージェント文字列を返すメソッド GetUserAgent があります。

安定したクロス ブラウザ テストを実現するために最適なロケーター

組み込みロケーター生成プログラムでは、安定したロケーターの作成が試みられます。 ただし、情報を使用できない場合、高品質のロケーターを生成することは困難です。 この場合、ロケーター生成プログラムでは、階層形式の情報およびインデックスが使用されます。その結果、直接的な記録／再生には適していても、安定した日常的な実行には適さない脆弱なロケーターが生成されます。 さらに、クロス ブラウザ テストでは、いくつかの AJAX フレームワークで異なるブラウザに対して異なる DOM 階層がレンダリングされることがあります。

この問題を回避するには、アプリケーションの UI 要素にカスタム ID を使用します。

アプリケーションのログ出力に正しくないタイムスタンプが含まれる

この方法によって、同期に関して予期しない結果が発生する場合があります。 この問題を回避するには、HTML 同期モードを指定します。

新しいページに移動したあと、スクリプトがハングする

この問題は、AJAX アプリケーションによりブラウザがビジー（サーバー プッシュ／ActiveX コンポーネントの接続が開いている）のままになっている場合に、発生することがあります。 HTML 同期モードを設定してください。 他のトラブルシューティングのヒントについては、「*xBrowser* のページ同期」のトピックを参照してください。

正しくないロケーターが記録されている

マウスを要素上に移動したときに、要素の属性が変更することがあります。 Silk4NET によってこのシナリオの追跡が試行されますが、失敗することがあります。 影響を受ける属性を特定し、それが Silk4NET で無視されるように構成してください。

Windows Internet Explorer で要素を囲む四角形の位置が正しくない

- 拡大率が 100% に設定されていることを確認します。 このようにしないと、四角形が正しく配置されません。
- ブラウザ ウィンドウの上に通知バーが表示されていないことを確認します。 Silk4NET では、通知バーを処理できません。

Link.Select で、Internet Explorer で新しく開いたウィンドウにフォーカスが設定されない

この制限は、ブラウザの構成設定を変更することで修正できます。 新しく開いたウィンドウが常にアクティブ化されるようにオプションを設定します。

DomClick(x, y) が Click(x, y) のように動作しない

アプリケーションで onclick イベントを使用しており、座標を必要とする場合、DomClick メソッドは動作しません。 代わりに、Click を使用します。

FileInputField.DomClick() でダイアログが開かない

代わりに、Click を使用します。

マウス移動設定がオンになっているにもかかわらず、すべての操作が記録されない理由

多くの無用な MoveMouse 操作がスクリプトに影響を及ぼさないように、 Silk4NET では以下の操作が行われます。

- マウスが一定時間静止している場合にのみ、 MoveMouse 操作が記録されます。
- マウスを要素上に移動したあとで操作が行われていることが確認された場合にのみ、MoveMouse 操作が記録されます。 場合によっては、スクリプトに手動操作を追加することが必要となることがあります。

xBrowser API で公開されていない機能が必要な場合の対処方法

ExecuteJavaScript() を使用して、JavaScript コードを Web アプリケーションから直接実行できます。この方法は、ほとんどすべての問題の回避策となります。

ロケーターでクラスとスタイルの属性が使用されない理由

これらの属性は AJAX アプリケーションで頻繁に変更され、ロケーターの安定性が損なわれることがありため、無視リストに含まれています。 ただし、多くの場合、これらの属性を使用してオブジェクトを識別できるため、アプリケーションで使用することに意味がある場合があります。

再生中にダイアログが認識されない

スクリプトを記録するときに、Silk4NET はいくつかのウィンドウを Dialog として認識します。 スクリプトをクロス ブラウザ スクリプトとして使用する場合は、ブラウザによっては Dialog が認識されないため、 Dialog を Window に置き換える必要があります。

たとえば、スクリプトに以下の行があるとします。

```
/BrowserApplication//Dialog//PushButton[@caption='OK']
```

クロス ブラウザ テストを可能にするには、次のように行を書き換えます。

```
/BrowserApplication//Window//PushButton[@caption='OK']
```

WaitForProperty メソッドを呼び出したときにハンドル無効エラーが表示される理由

このトピックでは、WaitForProperty メソッドを呼び出したときに、Silk4NET に「このオブジェクトのハンドルは無効になりました。」というエラー メッセージが表示された場合の対処法について説明します。

このメッセージは、WaitForProperty を呼び出したオブジェクトが何らかの理由で消失していることを示しています。 たとえば、Web アプリケーションで WaitForProperty を呼び出しているときに、何らかの理由でブラウザが 新しいページに移動した場合、以前のページのすべてのオブジェクトは自動的に無効になります。

この問題の原因が、組み込みの同期機能である場合もあります。 たとえば、テスト対象のアプリケーションにショッピング カートが含まれていて、このショッピング カートに品物を追加したとします。 ユーザーは次のページが読み込まれ、ショッピング カートのステータスが 品物がある 状態に変わるまで待機しています。 品物を追加するという操作からの戻り時間が短すぎた場合、最初のページのショッピング カートはステータスが変わるまで待機しますが、その間も新しいページは読み込まれています。したがって、最初のページのショッピング カートは無効になります。 この動作によって、ハンドル無効エラーが発生します。

この問題を回避するには、2 番目のページでのみ有効なオブジェクトが表示されるまで待機してから、ショッピング カートのステータスを確認するようにしてください。 このオブジェクトが有効になるとすぐに、ショッピング カートのステータスを確認できるようになり、2 番目のページで正しく検証されるようになります。

Windows Internet Explorer 10 で Click の記録が異なる理由

Windows Internet Explorer 10 の DomElement で Click を記録し、DomElement が Click の後で破棄された場合、記録動作が予期したとおりにならないことがあります。 別の DomElement が最初の DomElement の下にある場合、Silk Test では、1 つの Click が記録されるのではなく、Click、MouseMove、および ReleaseMouse が記録されます。

予期しない記録動作を回避する方法は、テスト対象のアプリケーションによって異なります。 通常は、記録されたスクリプトから不必要な MouseMove イベントと ReleaseMouse イベントを削除すれば十分です。

索引

T

W

X

あ

お

か

き

ま

ゆ

よ

れ

ろ

わ