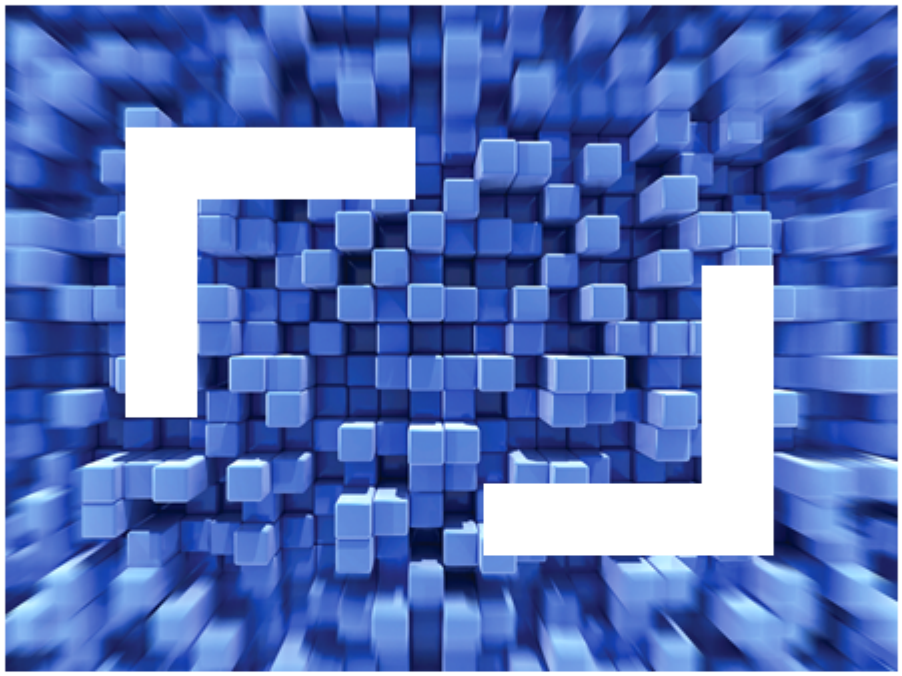

SilkTest 2011
SilkTest Recorder ヘルプ

Borland®
(A MICRO FOCUS COMPANY)

MICRO® FOCUS

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

2011-10-07

目次

SilkTest Recorder クイック スタート チュートリアル

このチュートリアルでは、Web サイトのテストに SilkTest Recorder を使用できるよう、ステップごとに導入手順を説明します。 SilkTest Recorder は動的オブジェクト解決を使用して、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 ケースの記録、再生を行います。

SilkTest Recorder は次の種類のアプリケーションのテストを記録/再生できます：

- Adobe Flex
- Java AWT/Swing
- Java SWT
- Rumba
- SAP
- Silverlight
- Windows API ベースのクライアント／サーバー（Win32）
- Windows Forms
- Windows Presentation Foundation (WPF)
- xBrowser (Web アプリケーション)

説明をより簡潔にするため、本ガイドでは、SilkTest Recorder がすでにインストールされており、サンプルの Insurance Company (保険会社) Web サイト (*http://demo.borland.com/InsuranceWebExtJS/*) を使用することを前提としています。

SilkTest Recorder の起動

- **スタート** > **すべてのプログラム** > **Silk** > **SilkTest <バージョン>** > **SilkTest Recorder** を選択します。
  SilkTest Recorder が開き、SilkTest Open Agent アイコン が、システム トレイに表示されます。

Insurance Company Web サイトのテスト ケースを記録する

SilkTest Recorder がどのようにしてテストを作成するのかを確認するために、Insurance Company サンプル Web サイトのテスト ケースを記録します。

1. 次のいずれか 1 つのステップを行います：
   - ツールバーの ● をクリックします。
   - **記録** > **記録の開始** を選択します。

   新規アプリケーション構成 ウィザードが開きます。
2. **Web サイト テスト構成** をダブルクリックします。
   新規 Web サイト構成 ページが開きます。
3. **ブラウザの種類** リスト ボックスから、**Internet Explorer** を選択します。
   Firefox はテストの再生には使用することができますが、記録には使えません。
4. 次のいずれか 1 つのステップを行います：

- **既存のブラウザを使用する**：テストを構成する際に、すでに開いているブラウザを使用する場合には、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。

  このチュートリアルでは、開いているブラウザをすべて閉じてから、**新しいブラウザを開始する** をクリックして、*http://demo.borland.com/InsuranceWebExtJS/* を指定します。

5. **終了** をクリックします。
   Web サイトが開きます。 SilkTest Recorder は基本状態を作成し、記録を開始します。
6. Insurance Company Web サイトでは、次のステップのいずれかを行います：
   a) **Select a Service or login** リスト ボックスから、**Auto Quote** を選択します。
   Automobile Instant Quote ページが開きます。
   b) 郵便番号と電子メール アドレスを適切なテキスト ボックスに入力し、自動車タイプをクリックして、**Next** をクリックします。
   c) 年齢を指定し、性別と運転履歴タイプをクリックして、**Next** をクリックします。
   d) 製造年、車種、モデルを指定し、財務情報タイプをクリックして、**Next** をクリックします。
   指定した情報の概要が現れます。
   e) 指定した **Zip Code** をポイントし、Ctrl+Alt を押して、スクリプトに検証を追加します。
   表示されたどの情報に対しても、検証を追加することができます。
   プロパティの検証 ダイアログ ボックスが開きます。
   f) **textContents** チェック ボックスをオンにし、**OK** をクリックします。
   検証操作が、郵便番号テキストに対するスクリプトに追加されます。

   各ステップに相当する操作が記録されました。
7. Recorder で、次のステップのいずれかを行います：

   - ツールバーの  をクリックします。
   - **記録** > **記録の停止** を選択します。
8. **ファイル** > **保管** を選択します。
   a) テストを保管したい場所に移動します。
   b) **ファイル名** テキスト ボックスに、テストの名前を入力してから、**保存** をクリックします。
   たとえば、ZipTest と入力します。

テストが期待通りの動作をするか確認するためにテストを再生します。 必要な場合には変更をするために、テストを編集することも可能です。

Insurance Company Web サイトのテスト ケースを再生する

テストが期待どおりの動作をするか確認するためにテストを再生します。

1. メイン ウィンドウで、**再生速度** リスト ボックスから、テストを再生するために使用する速度を選択します。

   - **速い**：このオプションを選択すると、テストを最も速い速度で再生します。スクリプトは実際の速度で実行されます。 その他の速度は、遅延機構を使用するため、テストを確認しながら実行することができます。
   - **普通**：通常、このオプションを選択すると、各操作を確認できるようになります。
   - **遅い**：各操作が確認できることを確実にするためには、このオプションを選択します。

   再生速度 リスト ボックスの後ろにある **再生時間** の時間を確認して、テストの速度を決定できます。

2. テスト全体を再生するには、ツールバーの  をクリックするか、または **再生** > **すべて再生** を選択します。
   SilkTest Recorder によってテストが再生されます。
3. エラーが発生した場合は、次のステップのいずれかを行います：
   - **再試行** をクリックして現在の操作を再生します。
   - **停止** をクリックしてテストを終了します。
   - **スキップ** をクリックしてテストの次の操作に進めます。

SilkTest Classic、Silk4NET、または Silk4J で使用するためにテストをエクスポートします。 または、必要に応じて変更をするために、テストを編集します。

テスト ケースまたはプロジェクトをエクスポートする

以下の手順に従って、テスト ケースまたはプロジェクトを SilkTest Classic、Silk4NET、または Silk4J にエクスポートします。

Silk4J へのプロジェクトのエクスポート

テストの主要 GUI として Silk4J を使用したり、テスト メソッドを整理したりするために、プロジェクトをエクスポートします。

1. **ファイル** > **エクスポート** を選択します。
   エクスポート ウィザードが開きます。
2. **Silk4J プロジェクトとしてエクスポート** をダブルクリックします。
   Silk4J プロジェクトとしてエクスポート ページが開きます。
3. **プロジェクトの場所** テキスト ボックスに、プロジェクトをエクスポートする場所を指定します。

   任意：  をクリックし、使用するフォルダに移動します。
4. **プロジェクト名** テキスト ボックスに、プロジェクト名を指定します。
   たとえば、Web Sample Project と入力します。
5. **パッケージ** テキスト ボックスに、パッケージ名を指定します。
   たとえば、com.example と入力します。
6. **テスト クラス** テキスト ボックスには、テストが属するクラス名を指定します。
   たとえば、AutoTests と入力します。
7. **テスト メソッド** テキスト ボックスに、テスト メソッドの名前を指定します。
   たとえば、TestAutoInput と入力します。
8. **ファイル エンコード** リスト ボックスから、使用するファイル エンコードの種類を選択します。
9. **終了** をクリックします。

   SilkTest Recorder は、プロジェクトを作成し、Silk4J にエクスポートします。

プロジェクトを Silk4J にインポートします。 新しいプロジェクトは、基本状態やテスト メソッドも含んでおり、いつでもテストできます。 プロジェクトのインポートに関する詳細については、『*Silk4J* ユーザー ガイド』を参照してください。

Silk4J へのテスト メソッドのエクスポート

テストの主要 GUI として Silk4J を使用したり、クリップボードに JTF スクリプトをコピーしたりするために、テスト メソッドをエクスポートします。

1. **ファイル** > **エクスポート** を選択します。
   エクスポート ウィザードが開きます。
2. **JTF スクリプトとしてエクスポート** をダブルクリックします。

JTF スクリプトとしてエクスポート ページが開きます。

3. **エクスポート先** 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。
   - **クリップボード**：JTF スクリプトをクリップボードにコピーします。 既存の JTF スクリプトにスクリプトをコピーして貼り付ける場合にこのオプションを選択します。
   - **JTF スクリプト**：スクリプトを Silk4J にエクスポートします。 新しいスクリプトを作成したり既存の JTF スクリプトを上書きする場合にこのオプションを選択します。
4. **テスト メソッド** テキスト ボックスに、テスト メソッドの名前を指定します。
   たとえば、TestAutoInput と入力します。
5. **パッケージ** テキスト ボックスに、パッケージ名を指定します。
   たとえば、com.example と入力します。
6. **テスト クラス** テキスト ボックスには、テストが属するクラス名を指定します。
   たとえば、AutoTests と入力します。
7. **ソース フォルダ** テキスト ボックスには、テストをエクスポートする場所を指定します。

   任意：… をクリックし、使用するフォルダに移動します。
8. **ファイル エンコード** リスト ボックスから、使用するファイル エンコードの種類を選択します。
9. エクスポートしたスクリプトに基本状態を含めるには、**基本状態を使用する** チェック ボックスをオンにします。

   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。

   JTF にエクスポートする場合は、silk4J.settings というファイルが基本状態用に別途作成されます。 クリップボードにエクスポートする場合は、Before メソッドが基本状態を含みます。
10. **終了** をクリックします。

    SilkTest Recorder は、Java 言語を使用するスクリプトを作成し、Silk4J またはクリップボードにエクスポートします。

SilkTest Classic へのプロジェクトのエクスポート

主要 GUI として SilkTest Classic を使用したり、テスト ケースをプロジェクトにグループ化するため、または SilkCentral Test Manager などの他の製品とデータを共有するために、プロジェクトをエクスポートします。

1. **ファイル** > **エクスポート** を選択します。
   エクスポート ウィザードが開きます。
2. **SilkTest プロジェクトとしてエクスポート** をダブルクリックします。
   SilkTest プロジェクトとしてエクスポート ページが開きます。
3. **プロジェクトの場所** テキスト ボックスに、プロジェクトをエクスポートする場所を指定します。

   任意：… をクリックし、使用するフォルダに移動します。
4. **プロジェクト名** テキスト ボックスに、プロジェクト名を指定します。
   たとえば、Web Sample Project と入力します。
5. **4Test スクリプト** テキスト ボックスに、スクリプト ファイルの名前を指定します。
   たとえば、AutoTests.t と入力します。

   任意：… をクリックし、使用するフォルダに移動します。
6. **テスト ケース** テキスト ボックスには、テスト ケースの名前を指定します。
   たとえば、testAutoInput と入力します。
7. テスト ケースをエクスポートしたあとに SilkTest Classic を開始するには、**エクスポートしたプロジェクトを SilkTest で開く** チェック ボックスをオンにします。

8. **終了** をクリックします。

   SilkTest Recorder は、4Test 言語を使用するスクリプトとリカバリ ファイルを含んだプロジェクトを作成し、プロジェクトを SilkTest Classic にエクスポートします。

SilkTest Classic を使用してエクスポートしたプロジェクトで作業することができます。 新しいプロジェクトは、基本状態やテスト ケースを含んでおり、テストの準備が整っています。

SilkTest Classic へのテスト ケースのエクスポート

主要 GUI として SilkTest Classic を使用したり、クリップボードにスクリプトをコピーしたりするために、テスト ケースをエクスポートします。

1. **ファイル** > **エクスポート** を選択します。
   エクスポート ウィザードが開きます。
2. **4Test スクリプトとしてエクスポート** をダブルクリックします。
   4Test スクリプトとしてエクスポート ページが開きます。
3. **エクスポート先** 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。
   - **クリップボード**：スクリプトをクリップボードにコピーします。 既存の 4Test スクリプトにスクリプトをコピーして貼り付ける場合にこのオプションを選択します。
   - **4Test スクリプト**：スクリプトを SilkTest Classic にエクスポートします。 新しいスクリプトを作成したり既存のスクリプトを上書きする場合にこのオプションを選択します。
4. **テスト ケース** テキスト ボックスには、テスト ケースの名前を指定します。
   たとえば、testAutoInput と入力します。
5. **4Test スクリプト** テキスト ボックスに、スクリプト ファイルの名前を指定します。
   たとえば、AutoTests.t と入力します。

   任意：  をクリックし、使用するフォルダに移動します。
6. テスト ケースをエクスポートしたあとに SilkTest Classic を開始するには、**エクスポートしたスクリプトを SilkTest で開く** チェック ボックスをオンにします。
7. **終了** をクリックします。

   SilkTest Recorder は、4Test 言語を使用するスクリプトを作成し、SilkTest Classic またはクリップボードにエクスポートします。

Visual Studio への Silk4NET プロジェクトのエクスポート

1. SilkTest Recorder のメニュー バーで、**ファイル** > **エクスポート** を選択します。
   エクスポート ウィザードが開きます。
2. **Silk4NET プロジェクトとしてエクスポート** をダブルクリックします。
   Silk4NET プロジェクトとしてエクスポート ページが開きます。
3. **プログラム言語** リスト ボックスから、プロジェクトで Visual Basic .NET または C# のいずれを使用するかを指定します。
4. **プロジェクトの場所** テキスト ボックスに、プロジェクトをエクスポートする場所を指定します。
   省略可能：使用するフォルダをクリックして移動します。
5. **プロジェクト名** テキスト ボックスに、プロジェクト名を指定します。
   たとえば、Visual Basic .NET Sample Project と入力します。
6. **名前空間** テキスト ボックスに、プロジェクトのコンテナ名を指定します。
7. **テスト クラス** テキスト ボックスには、テストが属するクラス名を指定します。
   たとえば、AutoTests と入力します。
8. **テスト メソッド** テキスト ボックスに、テスト メソッドの名前を指定します。

たとえば、TestAutoInput と入力します。

9. **終了** をクリックします。
   SilkTest Recorder によって、記録したテストを含む新しいプロジェクトが作成され、指定された場所にエクスポートされます。 この場所から、Visual Studio でプロジェクトを開くことができます。

記録したテスト ステップの Visual Studio へのエクスポート

1. SilkTest Recorder のメニュー バーで、**ファイル** > **エクスポート**を選択します。
   エクスポート ウィザードが開きます。
2. **NTF スクリプトとしてエクスポート** をダブルクリックします。
   NTF スクリプトとしてエクスポート ページが開きます。
3. **エクスポート先** 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。
   - **クリップボード**: 記録したテスト ステップをクリップボードにコピーします。 記録したテスト ステップを Visual Studio プロジェクトの既存の Silk4NET テストにコピーして貼り付ける場合に、このオプションを選択します。

     注: このオプションを選択する場合、ソースの場所または基本状態を指定する必要はありません。
   - **NTF スクリプト**: 記録したテスト ステップをテストとしてエクスポートして、既存の Visual Studio プロジェクトに追加できます。 新しいテストを作成するか、既存のテストを上書きする場合に、このオプションを選択します。
4. **プログラム言語** リスト ボックスから、テストで Visual Basic .NET または C# のいずれを使用するかを指定します。
5. **テスト メソッド** テキスト ボックスに、テスト メソッドの名前を指定します。
   たとえば、TestAutoInput と入力します。
6. **名前空間** ボックスに、テストのコンテナ名を指定します。
7. **テスト クラス** ボックスに、テストが属するクラス名を指定します。
   たとえば、AutoTests と入力します。
8. **ソース・フォルダー** ボックスに、テストをエクスポートする場所を指定します。
   使用するフォルダをクリックして移動することもできます。
9. エクスポートしたテストに基本状態を含めるには、**基本状態を使用する** チェック ボックスをオンにします。
   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。
10. **終了** をクリックします。
    Recorder により Silk4NET テストが作成され、指定の場所またはクリップボードにエクスポートされます。
11. エクスポートした Silk4NET プロジェクトを Visual Studio プロジェクトに追加します。 Visual Studio のメニュー バーから **プロジェクト** > **既存の項目の追加** を選択し、エクスポートしたテストを選択します。

Insurance Company Web サイトのテスト ケースを変更する

テスト ケースを記録した後で、このステップを実行してテストを手動で変更します。 たとえば、追加の操作を記録したり、記録したステップの順序を変更したり、特定の操作のパラメータを変更したりできます。

1. 変更したいテストを開きます。
2. テストしている Web ページで、最初に記録を行った時の最後のウィンドウに移動します。
   たとえば、既存のテストが複数の Web ページを含んでいる場合には、テストの最後のページに移動します。

   ここでは、既存のテストの記録が **Automobile Instant Quote** の概要ページで終了していますが、**ホーム** ページに戻って記録を開始します。つまり、記録を開始すると基本状態によって自動的に戻ります。
3. 変更を記録するには、次のステップを実行します：
   a) ツールバーの  をクリックします。
   b) **Select a Service or login** リスト ボックスから、**Agent Lookup** を選択します。
   Find an Insurance Co. Agent ページが開きます。
   c) ツールバーの  をクリックします。
4. 既存の操作、パラメータ、またはロケーター文字列を変更するには、次のステップを行います：
   a) 操作グリッド内の変更したい行をクリックします。
   たとえば、メール アドレスのパラメータに対する **SetText** 操作を含む行をクリックします。
   操作の詳細 タブには、**ロケーター**、**操作**、および **パラメータ** が表示されます。
   b) ロケーターを変更するには、**ロケーター** テキスト ボックスに文字列を入力します。

   ロケーター文字列によってテストしたいオブジェクトが識別されます。
   c) **ロケーターの検証** をクリックします。
   SilkTest Recorder は新しいロケーターを検証します。 文字列が有効でない場合やオブジェクトが見つからない場合には、エラーが表示されます。
   d) 操作を変更するには、**操作** リストから操作を選択します。
   e) パラメータを変更するには、適切なテキスト ボックスにパラメータの値を入力します。
   たとえば、**SetText** 操作の **text** パラメータを "tutorial@yourcompany.com" に変更します。

   行った変更は直ちに操作グリッドに反映されます。
5. 操作を手動で追加するには、次のステップを行います：

   大抵の場合、テストに追加したい操作を記録します。 しかし、既存の操作を手動でコピー＆ペーストして操作を追加してから、変更することもできます。
   a) **操作** リストの操作をクリックします。

   

   注: Ctrl または Shift を押してからコピーする操作をクリックすると、複数の操作を挿入することができます。
   b) 次のいずれか 1 つのステップを行います：

   - **編集** > **選択した操作のコピー** を選択してから、**編集** > **操作の貼り付け** を選択します。
   - Ctrl+C を押してから、Ctrl+V を押します。

   新しい操作が選択した操作の下に表示されます。
   c) 操作の詳細ビューで、ロケーター、操作名、またはパラメータを必要に応じて変更し、要求に合致するように操作を変更します。
6. **ファイル** > **保管** を選択します。

テストが期待通りの動作をするか確認するためにテストを再生します。

SilkTest Recorder の起動

- **スタート** > **すべてのプログラム** > **Silk** > **SilkTest <バージョン>** > **SilkTest Recorder** を選択します。
  SilkTest Recorder が開き、SilkTest Open Agent アイコン が、システム トレイに表示されます。

テスト ケースを再生する

テストが期待どおりの動作をするか確認するためにテストを再生します。

1. 次のいずれか 1 つを選んでください：

   - テスト全体を再生するには、ツールバーの  をクリックするか、または **再生** > **すべて再生** を選択します。
   - [操作] グリッドで選択した操作とそれに続くすべての操作を再生するには、**再生** > **選択箇所から再生** を選択します。
   - [操作] グリッドで選択した操作を再生するには、**再生** > **選択箇所の再生** を選択します。

     複数の操作を選択するには、操作をクリックするときに Ctrl または Shift を押します。

     注: テストを記録するときに別のブラウザのウィンドウが開いていた場合には、テストを再生するときにも常に別のブラウザのウィンドウが開いていなければなりません。 たとえば、.// BrowserApplication[3] というロケーター文字列は、テストが記録されたときに 3 つのブラウザのウィンドウが開かれていたことを意味します。

   SilkTest Recorder によってテストが再生されます。
2. エラーが発生した場合は、次のステップのいずれかを行います：

   - **再試行** をクリックして現在の操作を再生します。
   - **停止** をクリックしてテストを終了します。
   - **スキップ** をクリックしてテストの次の操作に進めます。

SilkTest Classic、Silk4NET、または Silk4J で使用するためにテストをエクスポートします。 または、必要に応じて変更をするために、テストを編集します。

テスト ケースまたはプロジェクトをエクスポートする

以下の手順に従って、テスト ケースまたはプロジェクトを SilkTest Classic、Silk4NET、または Silk4J にエクスポートします。

SilkTest Classic へのテスト ケースのエクスポート

主要 GUI として SilkTest Classic を使用したり、クリップボードにスクリプトをコピーしたりするために、テスト ケースをエクスポートします。

1. **ファイル** > **エクスポート** を選択します。
   エクスポート ウィザードが開きます。
2. **4Test スクリプトとしてエクスポート** をダブルクリックします。
   4Test スクリプトとしてエクスポート ページが開きます。
3. **エクスポート先** 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。
   - **クリップボード**：スクリプトをクリップボードにコピーします。 既存の 4Test スクリプトにスクリプトをコピーして貼り付ける場合にこのオプションを選択します。
   - **4Test スクリプト**：スクリプトを SilkTest Classic にエクスポートします。 新しいスクリプトを作成したり既存のスクリプトを上書きする場合にこのオプションを選択します。
4. **テスト ケース** テキスト ボックスには、テスト ケースの名前を指定します。
   たとえば、testAutoInput と入力します。
5. **4Test スクリプト** テキスト ボックスに、スクリプト ファイルの名前を指定します。
   たとえば、AutoTests.t と入力します。

   任意：  をクリックし、使用するフォルダに移動します。
6. テスト ケースをエクスポートしたあとに SilkTest Classic を開始するには、**エクスポートしたスクリプトを SilkTest で開く** チェック ボックスをオンにします。
7. **終了** をクリックします。

   SilkTest Recorder は、4Test 言語を使用するスクリプトを作成し、SilkTest Classic またはクリップボードにエクスポートします。

Silk4J へのテスト メソッドのエクスポート

テストの主要 GUI として Silk4J を使用したり、クリップボードに JTF スクリプトをコピーしたりするために、テスト メソッドをエクスポートします。

1. **ファイル** > **エクスポート** を選択します。
   エクスポート ウィザードが開きます。
2. **JTF スクリプトとしてエクスポート** をダブルクリックします。
   JTF スクリプトとしてエクスポート ページが開きます。
3. **エクスポート先** 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。
   - **クリップボード**：JTF スクリプトをクリップボードにコピーします。 既存の JTF スクリプトにスクリプトをコピーして貼り付ける場合にこのオプションを選択します。
   - **JTF スクリプト**：スクリプトを Silk4J にエクスポートします。 新しいスクリプトを作成したり既存の JTF スクリプトを上書きする場合にこのオプションを選択します。

4. **テスト メソッド** テキスト ボックスに、テスト メソッドの名前を指定します。
   たとえば、TestAutoInput と入力します。
5. **パッケージ** テキスト ボックスに、パッケージ名を指定します。
   たとえば、com.example と入力します。
6. **テスト クラス** テキスト ボックスには、テストが属するクラス名を指定します。
   たとえば、AutoTests と入力します。
7. **ソース フォルダ** テキスト ボックスには、テストをエクスポートする場所を指定します。

   任意：  をクリックし、使用するフォルダに移動します。
8. **ファイル エンコード** リスト ボックスから、使用するファイル エンコードの種類を選択します。
9. エクスポートしたスクリプトに基本状態を含めるには、**基本状態を使用する** チェック ボックスをオンにします。

   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。

   JTF にエクスポートする場合は、silk4J.settings というファイルが基本状態用に別途作成されます。 クリップボードにエクスポートする場合は、Before メソッドが基本状態を含みます。
10. **終了** をクリックします。

    SilkTest Recorder は、Java 言語を使用するスクリプトを作成し、Silk4J またはクリップボードにエクスポートします。

記録したテスト ステップの Visual Studio へのエクスポート

1. SilkTest Recorder のメニュー バーで、**ファイル** > **エクスポート**を選択します。
   エクスポート ウィザードが開きます。
2. **NTF スクリプトとしてエクスポート** をダブルクリックします。
   NTF スクリプトとしてエクスポート ページが開きます。
3. **エクスポート先** 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。
   - **クリップボード**：記録したテスト ステップをクリップボードにコピーします。 記録したテスト ステップを Visual Studio プロジェクトの既存の Silk4NET テストにコピーして貼り付ける場合に、このオプションを選択します。

     注: このオプションを選択する場合、ソースの場所または基本状態を指定する必要はありません。
   - **NTF スクリプト**：記録したテスト ステップをテストとしてエクスポートして、既存の Visual Studio プロジェクトに追加できます。 新しいテストを作成するか、既存のテストを上書きする場合に、このオプションを選択します。
4. **プログラム言語** リスト ボックスから、テストで Visual Basic .NET または C# のいずれを使用するかを指定します。
5. **テスト メソッド** テキスト ボックスに、テスト メソッドの名前を指定します。
   たとえば、TestAutoInput と入力します。
6. **名前空間** ボックスに、テストのコンテナ名を指定します。
7. **テスト クラス** ボックスに、テストが属するクラス名を指定します。
   たとえば、AutoTests と入力します。
8. **ソース・フォルダー** ボックスに、テストをエクスポートする場所を指定します。

   使用するフォルダをクリックして移動することもできます。

9. エクスポートしたテストに基本状態を含めるには、**基本状態を使用する** チェック ボックスをオンにします。

   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。

10. **終了** をクリックします。
    Recorder により Silk4NET テストが作成され、指定の場所またはクリップボードにエクスポートされます。

11. エクスポートした Silk4NET プロジェクトを Visual Studio プロジェクトに追加します。 Visual Studio のメニュー バーから **プロジェクト** > **既存の項目の追加** を選択し、エクスポートしたテストを選択します。

SilkTest Classic へのプロジェクトのエクスポート

主要 GUI として SilkTest Classic を使用したり、テスト ケースをプロジェクトにグループ化するため、または SilkCentral Test Manager などの他の製品とデータを共有するために、プロジェクトをエクスポートします。

1. **ファイル** > **エクスポート** を選択します。
   エクスポート ウィザードが開きます。
2. **SilkTest プロジェクトとしてエクスポート** をダブルクリックします。
   SilkTest プロジェクトとしてエクスポート ページが開きます。
3. **プロジェクトの場所** テキスト ボックスに、プロジェクトをエクスポートする場所を指定します。

   任意：  をクリックし、使用するフォルダに移動します。
4. **プロジェクト名** テキスト ボックスに、プロジェクト名を指定します。
   たとえば、Web Sample Project と入力します。
5. **4Test スクリプト** テキスト ボックスに、スクリプト ファイルの名前を指定します。
   たとえば、AutoTests.t と入力します。

   任意：  をクリックし、使用するフォルダに移動します。
6. **テスト ケース** テキスト ボックスには、テスト ケースの名前を指定します。
   たとえば、testAutoInput と入力します。
7. テスト ケースをエクスポートしたあとに SilkTest Classic を開始するには、**エクスポートしたプロジェクトを SilkTest で開く** チェック ボックスをオンにします。
8. **終了** をクリックします。

   SilkTest Recorder は、4Test 言語を使用するスクリプトとリカバリ ファイルを含んだプロジェクトを作成し、プロジェクトを SilkTest Classic にエクスポートします。

SilkTest Classic を使用してエクスポートしたプロジェクトで作業することができます。 新しいプロジェクトは、基本状態やテスト ケースを含んでおり、テストの準備が整っています。

Silk4J へのプロジェクトのエクスポート

テストの主要 GUI として Silk4J を使用したり、テスト メソッドを整理したりするために、プロジェクトをエクスポートします。

1. **ファイル** > **エクスポート** を選択します。
   エクスポート ウィザードが開きます。
2. **Silk4J プロジェクトとしてエクスポート** をダブルクリックします。

Silk4J プロジェクトとしてエクスポート ページが開きます。

3. **プロジェクトの場所** テキスト ボックスに、プロジェクトをエクスポートする場所を指定します。

   任意： をクリックし、使用するフォルダに移動します。
4. **プロジェクト名** テキスト ボックスに、プロジェクト名を指定します。
   たとえば、Web Sample Project と入力します。
5. **パッケージ** テキスト ボックスに、パッケージ名を指定します。
   たとえば、com.example と入力します。
6. **テスト クラス** テキスト ボックスには、テストが属するクラス名を指定します。
   たとえば、AutoTests と入力します。
7. **テスト メソッド** テキスト ボックスに、テスト メソッドの名前を指定します。
   たとえば、TestAutoInput と入力します。
8. **ファイル エンコード** リスト ボックスから、使用するファイル エンコードの種類を選択します。
9. **終了** をクリックします。

   SilkTest Recorder は、プロジェクトを作成し、Silk4J にエクスポートします。

プロジェクトを Silk4J にインポートします。 新しいプロジェクトは、基本状態やテスト メソッドも含んでおり、いつでもテストできます。 プロジェクトのインポートに関する詳細については、『*Silk4J* ユーザー ガイド』を参照してください。

Visual Studio への Silk4NET プロジェクトのエクスポート

1. SilkTest Recorder のメニュー バーで、**ファイル** > **エクスポート** を選択します。
   エクスポート ウィザードが開きます。
2. **Silk4NET プロジェクトとしてエクスポート** をダブルクリックします。
   Silk4NET プロジェクトとしてエクスポート ページが開きます。
3. **プログラム言語** リスト ボックスから､プロジェクトで Visual Basic .NET または C# のいずれを使用するかを指定します。
4. **プロジェクトの場所** テキスト ボックスに、プロジェクトをエクスポートする場所を指定します。

   省略可能：使用するフォルダをクリックして移動します。
5. **プロジェクト名** テキスト ボックスに、プロジェクト名を指定します。
   たとえば、Visual Basic .NET Sample Project と入力します。
6. **名前空間** テキスト ボックスに、プロジェクトのコンテナ名を指定します。
7. **テスト クラス** テキスト ボックスには、テストが属するクラス名を指定します。
   たとえば、AutoTests と入力します。
8. **テスト メソッド** テキスト ボックスに、テスト メソッドの名前を指定します。
   たとえば、TestAutoInput と入力します。
9. **終了** をクリックします。
   SilkTest Recorder によって、記録したテストを含む新しいプロジェクトが作成され、指定された場所にエクスポートされます。 この場所から、Visual Studio でプロジェクトを開くことができます。

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。

カスタム属性の設定

SilkTest Workbench 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、SilkTest Workbench 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. **設定** > **スクリプト オプション** 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、SilkTest Workbench 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。

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. **設定** > **スクリプト オプション** 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. **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. **設定** > **スクリプト オプション** 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_XBROWSER_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. **設定** > **スクリプト オプション** 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。

概念

このセクションでは、SilkTest Recorder の概念的な概要のトピックを説明します。

Open Agent のポート番号

Open Agent が起動すると、SilkTest Workbench、SilkTest Classic、SilkTest Recorder、Silk4J、Silk4NET、およびテストするアプリケーションに対して、使用可能なポートがランダムに割り当てられます。 ポート番号は Information Service に登録されます。 SilkTest Workbench、SilkTest Classic、SilkTest Recorder、Silk4NET、または Silk4J は、Open Agent に接続するために使用するポートを決定するために Information Service に接続します。 Information Service は適切なポートと通信し、SilkTest Workbench、SilkTest Classic、SilkTest Recorder、Silk4NET、または Silk4J はそのポートに接続します。 通信は、エージェントと SilkTest Workbench、SilkTest Classic、SilkTest Recorder、Silk4NET、または Silk4J との間で直接行われます。

デフォルトでは、ポート 22901 を使用して Open Agent は Information Service と通信します。 デフォルト ポートが利用可能でない場合に機能する代替ポートとして、Information Service の追加のポートを構成できます。 デフォルトでは、Information Service は、代替ポートとして 2966、11998、および 11999 を使用します。

大抵の場合、手動でポート番号を設定する必要はありません。 しかし、ポート番号が競合したり、ファイアウォールとの問題があったりした場合には、そのマシンや Information Service に対してポート番号を設定する必要があります。 各マシンごとに異なるポート番号を使用することも、すべてのマシンに対して同じ番号を使用することも可能です。

Information Service に接続するためにクライアントが使用するポートの構成

このタスクを開始する前に、SilkTest Open Agent を停止します。

大抵の場合、手動でポート番号を設定する必要はありません。 Information Service はポート構成を自動的に処理します。 エージェントとの接続には、Information Service のデフォルトのポートを使用します。 これにより、Information Service によって、エージェントが使用するポートに通信が転送されます。

Information Service のデフォルトのポートは 22901 です。 デフォルトのポートが使用可能であれば、ポート番号を指定せずに単純に hostname だけを入力できます。 ポート番号を指定する場合には、Information Service のデフォルトのポートまたは追加したポートの 1 つと一致していることを確認ください。 間違ったポートが指定されていると、通信に失敗します。

必要に応じて、Information Service に接続するためにすべてのクライアントが使用するポート番号を変更できます。

1. infoservice.properties.sample ファイルに移動し、開きます。

   このファイルは、C:¥Documents and Settings¥All Users¥Application Data¥Silk¥SilkTest¥conf にあります。ここで、「C:¥Documents and Settings¥All Users」は、Windows システムにおいてデフォルトで設定されている環境変数 ALLUSERSPROFILE の値です。

   このファイルには、コメントとサンプルの代替ポート設定が含まれています。
2. 代替ポートの値を変更します。

   大抵の場合、ファイアウォールとの問題を避けるために、特定のポートに通信を強制するように Information Service ポートの設定を構成します。

   ポート番号は、1 から 65535 の間の任意の数値を指定できます。

- infoservice.default.port : Information Service が実行されているデフォルト ポートです。 デフォルトでは、このポートは 22901 に設定されています。
- infoservice.additional.ports : デフォルト ポートが利用可能でない場合に Information Service が実行されるポートのカンマ区切りのリストです。 デフォルトでは、このポートは、代替ポートとして 2966、11998、および 11999 が設定されています。

3. ファイルを infoservice.properties という名前で保存します。
4. SilkTest Recorder を使用している場合は、Information Service のポートを変更した際に、SilkTest Recorder **グローバル設定** ダイアログ ボックスで新しいポート番号を指定してください。
5. SilkTest Open Agent、SilkTest Classic、SilkTest Workbench、SilkTest Recorder、Silk4J、Silk4NET、およびテストするアプリケーションを再起動します。

SilkTest Workbench、SilkTest Classic、Silk4J、Silk4NET、またはテストするアプリケーションが Open Agent に接続するポートの構成

このタスクを開始する前に、SilkTest Open Agent を停止します。

大抵の場合、手動でポート番号を設定する必要はありません。 Information Service はポート構成を自動的に処理します。 エージェントとの接続には、Information Service のデフォルトのポートを使用します。 これにより、Information Service によって、エージェントが使用するポートに通信が転送されます。

必要に応じて、SilkTest Workbench、SilkTest Classic、Silk4J、Silk4NET、またはテストするアプリケーションが Open Agent に接続するポート番号を変更します。

1. agent.properties.sample ファイルに移動し、開きます。
   デフォルトで、このファイルは %APPDATA%¥Silk¥SilkTest¥conf にあります。これは、通常 C:¥Documents and Settings¥<ユーザー名>¥Application Data¥Silk¥SilkTest¥conf です。ここで、<ユーザー名> は現在のユーザー名です。
2. 代替ポートの値を変更します。
   大抵の場合、ポートの競合を解決するためにポートの設定を構成します。

   注: 各ポート番号は一意でなければなりません。 エージェントのポート番号が Information Service のポート設定とは異なることを確認してください。

   ポート番号は、1 から 65535 の間の任意の数値を指定できます。

   ポートの設定には次のものがあります :

   - agent.vtadapter.port : テストの実行時に、SilkTest Workbench と Open Agent 間の通信を制御します。
   - agent.xpmodule.port : テストの実行時に、SilkTest Classic とエージェント間の通信を制御します。
   - agent.autcommunication.port : Open Agent とテストするアプリケーション間の通信を制御します。
   - agent.rmi.port : Open Agent と Silk4J 間の通信を制御します。
   - agent.ntfadapter.port : Open Agent と Silk4NET 間の通信を制御します。
   - java.rmi.server.hostname : Open Agent と環境内に構成された任意の仮想マシン間の通信を制御します。

   注: Adobe Flex のテスト時に使用されるポートは、この構成ファイルでは制御できません。 Flex アプリケーションのテストで割り当てられるポート番号は、6000 から始まり、各 Flex アプリケーションがテストされる度に 1 ずつ増加していきます。 Flex テスト用に開始ポートを構成することはできません。
3. ファイルを agent.properties という名前で保存します。

4. SilkTest Open Agent、SilkTest Classic、SilkTest Workbench、SilkTest Recorder、Silk4J、Silk4NET、およびテストするアプリケーションを再起動します。

SilkTest Classic、Silk4J、または Silk4NET が SilkTest Recorder に接続するために使用するポートの構成

このタスクを開始する前に、SilkTest Open Agent を停止します。

大抵の場合、手動でポート番号を設定する必要はありません。 Information Service はポート構成を自動的に処理します。 エージェントとの接続には、Information Service のデフォルトのポートを使用します。 これにより、Information Service によって、エージェントが使用するポートに通信が転送されます。

必要に応じて、SilkTest Classic、Silk4J、または Silk4NET が SilkTest Recorder に接続するために使用するポート番号を変更します。

1. recorder.properties.sample ファイルに移動し、開きます。
   デフォルトで、このファイルは %APPDATA%¥Silk¥SilkTest¥conf にあります。これは、通常 C:¥Documents and Settings¥<ユーザー名>¥Application Data¥Silk¥SilkTest¥conf です。ここで、<ユーザー名> は現在のユーザー名です。
2. recorder.api.rmi.port を、使用するポートに変更します。
   ポート番号は、1 から 65535 の間の任意の数値を指定できます。

   注: 各ポート番号は一意でなければなりません。 エージェントのポート番号が Recorder や Information Service のポート設定とは異なることを確認してください。
3. ファイルを recorder.properties という名前で保存します。
4. SilkTest Open Agent、SilkTest Classic、SilkTest Workbench、SilkTest Recorder、Silk4J、Silk4NET、およびテストするアプリケーションを再起動します。

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 AWT/Swing テクノロジ ドメインにおける priorLabel の決定方法

Java AWT/Swing テクノロジ ドメインにおいて priorLabel を決定する場合、同じウィンドウ内のすべてのラベルとグループが対象のコントロールとみなされます。 以下の条件に従って、コントロールが決定されます。

- コントロールの上または左側にあるラベル、およびコントロールを囲むグループが priorLabel の候補とみなされます。
- コントロールの親が JViewPort または ScrollPane の場合、コントロールを含むウィンドウを親とし、その外側は無関係であるとみなします。
- 最も単純なケースでは、コントロールに最も近いラベルが priorLabel として使用されます。
- コントロールからの距離が等しい 2 つのラベルが存在し、一方がコントロールの左側にあり、他方が上にある場合、左側のものが優先されます。
- 適切なラベルが見つからない場合は、最も近いグループのキャプションが使用されます。

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、Silk4J 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。

Silk4J 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 Silk4J 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	SilkTest 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: Firefox と Internet Explorer では、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。

Win32 テクノロジ ドメインにおける priorLabel の決定方法

Win32 テクノロジ ドメインにおいて priorLabel を決定する場合、同じウィンドウ内のすべてのラベルとグループが対象のコントロールとみなされます。 以下の条件に従って、コントロールが決定されます。

- コントロールの上または左側にあるラベル、およびコントロールを囲むグループが priorLabel の候補とみなされます。
- 最も単純なケースでは、コントロールに最も近いラベルが priorLabel として使用されます。
- コントロールからの距離が等しい 2 つのラベルが存在する場合、次の条件に基づいて priorLabel が決定されます。
  - 一方のラベルがコントロールの左側にあり、他方が上にある場合、左側のものが優先されます。
  - 両方のラベルがコントロールの左側にある場合、上にあるものが優先されます。
  - 両方のラベルがコントロールの上にある場合、左側のものが優先されます。
- 最も近いコントロールがグループ コントロールである場合、まずグループ内のすべてのラベルが上記の規則に従って決定されます。 グループ内に適切なラベルが見つからない場合は、グループのキャプションが priorLabel として使用されます。

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、SilkTest 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']"

Silk4J は、*automationId*、*name*、*caption*、または *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、Silk4J 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>
```

索引

J

O

P

R

S

W

X

え

か

き

く

さ

そ

て

と

ろ