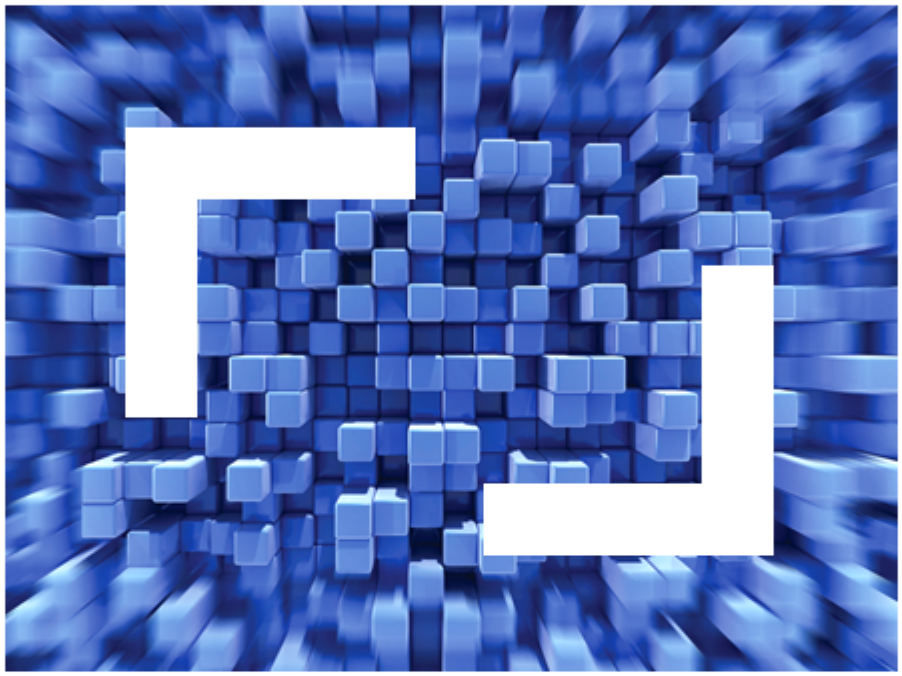


SilkTest 2011
Silk4J ユーザー ガイド

Borland®
(A MICRO FOCUS COMPANY)

MICRO® FOCUS

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

2011-10-07

目次

Silk4J

Silk4J によって、Java プログラム言語を使用して機能テストを作成することができるようになります。 Silk4J は、Java ランタイム ライブラリを提供しており、そのライブラリには Silk4J がテストをサポートするすべてのクラスに対するテスト クラスが含まれています。 このラインタイム ライブラリは、JUnit と互換性があります。つまり、JUnit のインフラストラクチャを利用して Silk4J テストを実行することができます。 また、すべての利用可能な Java ライブラリをテスト ケースで使用することもできます。

Silk4J がサポートするテスト環境は次のとおりです：

- 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)

『*Silk4J* ユーザー ガイド』では、Silk4J についての詳細な情報を提供しています。

注: Silk4J を実行するには、ローカルの管理者権限を持っている必要があります。

さらなる支援、情報、リソースが必要な場合には、次のリンクにアクセスしてください：

- *Technical Publications Web サイト*
- Borland は、戦略的パートナーである Micro Focus 社によって本製品のサポートが行われるよう契約を結びました。 サポートを受ける場合は、カスタマー ケアにアクセスしてください。
- Borland の情報を得るには、*Borland のホーム ページ* にアクセスしてください。

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、Silk4J を使用し、動的オブジェクト解決を用いた Web アプリケーションのテストが行えるよう、導入手順をステップ by ステップで提供します。 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 ケースの記述が可能になります。

重要: このチュートリアルでの作業をスムーズに完了させるには、Java および JUnit の基礎知識が必要となります。

説明をより簡潔にするため、本ガイドでは、Silk4J 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 アプリケーションを使用することを前提としています。

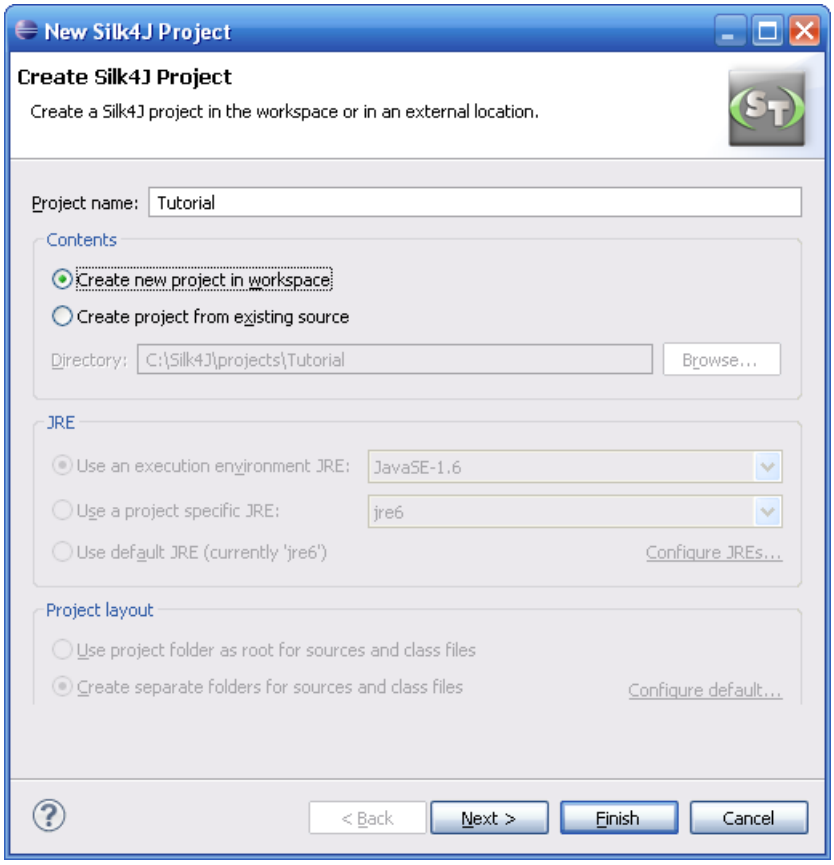
注: Silk4J を実行するには、ローカルの管理者権限を持っている必要があります。

Silk4J プロジェクトを作成する

新規 Silk4J プロジェクト ウィザードを使用して Silk4J プロジェクトを作成する際、このウィザードには、**新規 Java プロジェクト** ウィザードを使用して Java プロジェクトを作成する際に利用できるオプションと同じものが含まれています。 さらに、この Silk4J ウィザードでは、Java プロジェクトを自動的に Silk4J プロジェクトにします。

1. Eclipse ワークスペースで、次のステップのいずれかを行います：
   - **ファイル** > **新規作成** > **Silk4J プロジェクト** を選択します。
   - SilkTest ツールバー アイコン の隣にある、ドロップダウン矢印をクリックし、**新規 Silk4J プロジェクト** を選択します。
   - 既存の Eclipse の場所へ Silk4J をインストールまたは更新した場合には、**ファイル** > **新規** > **その他...** を選択します。 Silk4J フォルダを展開し、**Silk4J プロジェクト** をダブルクリックします。

   新規 Silk4J プロジェクト ウィザードが開きます。
2. **プロジェクト名** テキスト ボックスに、プロジェクトの名前を入力します。
3. 残りのオプションについては、デフォルトの設定をそのまま利用します。



4. **次へ** をクリックし、他の設定を必要に応じて指定します。
5. **終了** をクリックします。
   JRE システム ライブラリと必要な .jar ファイル（silktest-jtf-nodeps.jar と junit.jar）が入った、新しい Silk4J プロジェクトが作成されます。

Insurance Company Web アプリケーションのテスト クラスを作成する

Insurance Company Web アプリケーションのクラスおよびテスト メソッドを作成します。 テクノロジの種類ごとの、クラスの追加やテスト アプリケーションの構成方法の詳細については、『Silk4J ユーザー ガイド』のセクション「テスト メソッドを作成する」内の、「テスト クラスを作成する」を参照してください。

1. パッケージ エクスプローラーで、次のステップのいずれかを行います：
   - プロジェクトの名前をクリックし、 **ファイル** > **新規** > **Silk4J テスト クラス** を選択します。
   - SilkTest ツールバー アイコン の隣にある、ドロップ ダウン矢印をクリックし、**新規 Silk4J テスト クラス** を選択します。
   - 既存の Eclipse の場所に Silk4J をインストールまたは更新した場合は、 **ファイル** > **新規** > **その他** を選択します。 Silk4J フォルダを展開し、**Silk4J テスト クラス** をダブルクリックします。

   新規 Silk4J テスト クラス ダイアログ ボックスが開きます。
2. **ソース フォルダ** テキスト ボックスで、使用するソース ファイルの場所を指定します。
   ソース フォルダ テキスト ボックスは、選択したプロジェクトのソース ファイルの場所で、自動的に埋められています。

   別のソース フォルダを使用するには、**参照...** をクリックし、使用するフォルダまで辿っていきます。

3. **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。
4. **名前** テキスト ボックスには、テスト クラスの名前を指定します。
   たとえば、AutoQuoteInput と入力します。
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、次のように入力します：autoQuote。
6. **終了** 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。
7. **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。
8. **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。
9. 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。
10. **終了** をクリックします。
    Web サイトが開きます。 Silk4J は基本状態を作成し、記録を開始します。
11. 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。
12. **記録の停止** をクリックします。
    基本状態、テスト クラス、テスト メソッド、およびパッケージが作成されます。 新しいクラスのファイルが開きます。

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 アプリケーションのテスト メソッドを作成する

Insurance Company Web アプリケーションにおいて、**Agent Lookup** ページまで遷移するためのテスト メソッドを作成します。

1. パッケージ エクスプローラーで、次のステップのいずれかを行います：
   - プロジェクトの名前をクリックし、 **ファイル** > **新規** > **Silk4J テスト メソッド** を選択します。
   - SilkTest ツールバー アイコン  の隣にある、ドロップダウン矢印をクリックし、**新規 Silk4J テスト メソッド** を選択します。
   - 既存の Eclipse の場所に Silk4J をインストールまたは更新した場合は、 **ファイル** > **新規** > **その他** を選択します。 Silk4J フォルダを展開し、Silk4J **テスト メソッド** をダブルクリックします。

   新規 Silk4J テスト メソッド ダイアログ ボックスが開きます。
2. **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、次のように入力します：realtorPage。
3. **テスト クラス** テキスト ボックスで **参照** をクリックして、先ほど作成した既存のテスト クラスを選択してから、**OK** をクリックします。
   たとえば、**AutoQuoteInput - com.example** を参照してから、**OK** をクリックします。
4. **終了** をクリックします。
   Insurance Company Web サイトが開かれます。
5. Insurance Company Web アプリケーションで、**Automobile Instant Quote** ページにある **Choose One** ドロップダウン リストの上にマウスを移動して、**Agent Lookup** をクリックし、新しいページが開いたら Ctrl+Alt を押します。
   記録中 ダイアログ ボックスの **アクティブ オブジェクト** テキスト ボックスに要素識別子が表示されます。
6. **記録の停止** をクリックします。
   新しいメソッドが既存のクラスに追加されます。
7. **ファイル** > **保管** を選択してメソッドを保存します。
   保存するたびに、Eclipse はソース ファイルをコンパイルします。 何らかの問題がある場合には、Eclipse がそこに赤線で下線を付けます。

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。

テスト メソッドを再生する

パッケージ エクスプローラーで **AutoQuoteInput** クラスを右クリックし、**実行** > **JUnit テスト** を選択します。

テスト実行が終わると、テストの結果が JUnit ビューに表示されます。 テストが成功した場合には、ステータス バーが緑に、テストが失敗した場合には、ステータスバーが赤になります。 失敗したテストをクリックすると、「障害トレース」領域にスタック トレースを表示させることができます。

サンプル プロジェクトとスクリプトをレビューする

Silk4J は、製品をより良く理解するために使用できる、いくつかのサンプル プロジェクトとスクリプトを提供しています。

サンプル アプリケーションのインストール

Silk4J は、チュートリアルで利用したり、サンプルのテストを作成するために、いくつかのサンプル アプリケーションを提供しています。 ダウンロードしたサンプル アプリケーションには、サンプル プロジェクトも含まれています。

1. *http://techpubs.borland.com/silk_gauntlet/SilkTest/* に移動します。
2. Java AWT、Java Swing、Java SWT、Microsoft .NET (Windows Forms と WPF アプリケーションを含む)、Windows ベースのクライアント/サーバー、および xBrowser アプリケーション用のサンプル アプリケーションとプロジェクトをダウンロードするには、**SampleApplications** をクリックします。 SampleApplications.zip ファイルを開くか保存するかを確認するダイアログが開きます。
3. サンプルの zip ファイルをダウンロードしたら、zip ファイルを展開してサンプル アプリケーションをインストールします。

注: インストール可能な Adobe Flex のサンプル アプリケーションは提供されなくなりました。 ただし、*http://demo.borland.com/flex/SilkTest2011/index.html* で Flex のサンプル アプリケーションにアクセスできます。

Silk4J サンプル プロジェクトのインポート

このタスクを実行する前に、サンプル アプリケーションをダウンロードしてインストールする必要があります。

サンプル プロジェクトを使用して、一般的なスクリプトの構成やテスト メソッドの実行を確認します。

1. Eclipse ワークスペース内で、**ファイル** > **インポート** を選択します。
   インポート ウィザードが開きます。
2. メニュー ツリーで、プラス記号（+）をクリックして **一般** フォルダを展開し、**既存プロジェクトをワークスペースへ** を選択します。
3. **次へ** をクリックします。
   プロジェクトのインポート ダイアログ ボックスが開きます。
4. **ルート ディレクトリの選択** テキスト ボックスで、**参照** ボタンをクリックします。
   フォルダの参照 ダイアログ ボックスが開きます。
5. テストするテクノロジの種類に対応したサンプル スクリプトに移動します。

環境	ファイルの場所
Java SWT	Documents and Settings¥All Users¥Shared Documents ¥SilkTest¥samples¥Silk4J¥SWT
Windows Forms	Documents and Settings¥All Users¥Shared Documents ¥SilkTest¥samples¥Silk4J¥Windows Forms

環境	ファイルの場所
Windows Presentation Foundation (WPF)	Documents and Settings¥All Users¥Shared Documents ¥SilkTest¥samples¥Silk4J¥WPF
xBrowser	Documents and Settings¥All Users¥Shared Documents ¥SilkTest¥samples¥Silk4J¥xBrowser

注: Windows Vista および Windows 7.0 の場合、ファイルの場所は、Documents and Settings ¥All Users¥Shared Documents¥SilkTest¥samples¥ ではなく、Users¥Public¥Documents ¥SilkTest¥samples¥Silk4J¥ となります。

6. サンプルのルート ディレクトリを選択してから、**OK** をクリックします。
7. **プロジェクト** セクションで、インポートするプロジェクトを選択してから、**終了** をクリックします。
サンプル プロジェクトが **パッケージ エクスプローラー** ビューに表示されます。

サンプルのテスト メソッドを実行する

Silk4J が提供するサンプルのテスト メソッドを使用して、スクリプトの内容とテストの実行結果を確認します。

1. サンプル プロジェクトに移動します。
2. 次のいずれか 1 つのステップを行います：

注: サンプル スクリプトは、テスト対象アプリケーションの場所へのパスを基本状態の中に保持しています。 SilkTest がデフォルトの場所にインストールされていない場合には、基本状態を編集して、正しい場所を参照するようパスを調整してください。

- プロジェクト内のすべてのテストを実行するには、パッケージ エクスプローラーで、パッケージ名を右クリックします。

たとえば、**com.borland.flex.store** を右クリックします。

- 特定のクラス内のすべてのテスト メソッドを実行するには、パッケージ エクスプローラーで、クラス名を右クリックします。

たとえば、**com.borland.flex.store** パッケージ内で **FlexStoreTest** を右クリックします。

3. **実行** > **JUnit テスト** を選択します。
テスト実行が終わると、テストの結果が JUnit ビューに表示されます。 すべてのテストが成功した場合には、ステータス バーが緑に、 1 つでもテストが失敗した場合には、ステータス バーが赤になります。 失敗したテストをクリックすると、「障害トレース」領域にスタック トレースを表示させることができます。

テスト メソッドを再生する

Eclipse 内から、あるいはコマンド ラインを使用してテスト メソッドを実行します。

Eclipse からテスト メソッドを再生する

1. テストしたいテスト メソッドに移動します。
2. 次のいずれか 1 つのステップを行います：
   - プロジェクト内のすべてのテスト メソッドを再生するには、パッケージ エクスプローラーで、パッケージ名を右クリックします。
   - クラス内のすべてのテスト メソッドを再生するには、パッケージ エクスプローラーで、クラス名を右クリックします。 または、ソース エディタでそのクラスを開き、ソース エディタで右クリックします。
   - 特定のメソッドのみのテストを再生するには、パッケージ エクスプローラーで、メソッド名を右クリックします。 または、ソース エディタでそのクラスを開き、テスト メソッドの名前をクリックして選択してから右クリックします。
3. **実行** > **JUnit テスト** を選択します。
   テスト実行が終わると、テストの結果が JUnit ビューに表示されます。 すべてのテストが成功した場合には、ステータス バーが緑に、 1 つでもテストが失敗した場合には、ステータス バーが赤になります。 失敗したテストをクリックすると、「障害トレース」領域にスタック トレースを表示させることができます。

コマンド ラインからテスト メソッドを再生する

このタスクを実行する前に、JDK の場所を参照できるように PATH 変数を更新する必要があります。 詳細については、次の Sun のドキュメントを参照してください：*http://java.sun.com/j2se/1.5.0/install-windows.html*。

1. CLASSPATH を以下のように設定します。

```
set CLASSPATH=<eclipse_install_directory>¥plugins¥org.junit4_4.3.1¥junit.jar;
%OPEN_AGENT_HOME%¥JTF¥silktest-jtf-nodeps.jar;C:¥myproject¥
```

2. 次のように入力して JUnit テスト メソッドを実行します：

```
java org.junit.runner.JUnitCore <test class name>
```

注: トラブル シューティングの情報については、次の JUnit のドキュメントを参照してください：*http://junit.sourceforge.net/doc/faq/faq.htm#running_1*。

Ant からテスト メソッドを再生する

Apache Ant を使用して Silk4J テストを実行する場合、fork="yes" 属性を指定して JUnit タスクを使用するとテストがハングアップします。 これは、Apache Ant の既知の問題です（*https://issues.apache.org/bugzilla/show_bug.cgi?id=27614*）。 2 種類の回避策があります。 次のいずれか 1 つを選んでください：

- fork="yes" を使用しない

- fork="yes" を使用する場合は、テストが実行される前に Open Agent が起動していることを確認します。 Open Agent は、手動あるいは以下の Ant ターゲットを使って起動できます。

```
<property environment="env" />
<target name="launchOpenAgent">
<echo message="OpenAgent launch as spawned process" />
<exec spawn="true" executable="${env.OPEN_AGENT_HOME}/agent/openAgent.exe" />
<!-- give the agent time to start -->
<sleep seconds="30" />
</target>
```

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。

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. SilkTest ツールバー アイコン 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。

   注: SAP アプリケーションでは、ショートカット キーの組み合わせとして Ctrl+Shift を設定する必要があります。
3. 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。
4. 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。
5. 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。
6. **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. SilkTest ツールバー アイコン 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

Silk4J 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、Silk4J 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. SilkTest ツールバー アイコン  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、Silk4J 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. SilkTest ツールバー アイコン 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. SilkTest ツールバー アイコン 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. **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. SilkTest ツールバー アイコン  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_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. SilkTest ツールバー アイコン  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。

Silk4J の設定を変更する

Silk4J は、Java Runtime Environment（JRE）のバージョン 1.6 以降を必要とします。

デフォルトでは、Silk4J は、Silk4J が起動するときに毎回 JRE のバージョンを確認し、JRE のバージョンが Silk4J と互換性のない場合にエラー メッセージを表示します。

1. エラー メッセージを表示されないようにするには、 **ウィンドウ** > **設定** > **Silk4J** を選択します。
2. **Silk4J** ノードを選択し、**JRE のバージョンに互換性がない場合にエラー メッセージを表示** チェック ボックスのチェックをオフにします。
3. **OK** をクリックします。

テスト環境

Silk4J は、いくつかの種類の環境をサポートします。

Adobe Flex アプリケーション

Silk4J は、Internet Explorer、Firefox、スタンドアロンの Flash Player、および Adone AIR を使用した Adobe Flex アプリケーションのテストを組み込みでサポートしています。

注: Silk4J がサポートする Adobe AIR でのテストは、Flex 4 コンパイラを使用してコンパイルされたアプリケーションのみです。

Silk4J では、Flex 3.x および 4.x アプリケーションにおいて複数のアプリケーション ドメインもサポートされているため、サブアプリケーションをテストできます。 Silk4J では、ロケーター階層ツリーの各サブアプリケーションが、関連するアプリケーション ドメイン コンテキストを持つアプリケーション ツリーとして認識されます。 Flex 4.x サブアプリケーションでは、ロケーター属性テーブルのルート レベルで SparkApplication クラスが使用されます。 Flex 3.x サブアプリケーションでは、FlexApplication クラスが使用されます。

サポートするバージョン、既知の問題、および回避策についての最新の情報は、リリース ノートを確認してください。

サンプル アプリケーション

Silk4J は、Flex アプリケーションのサンプル アプリケーションを提供しています。 サンプル Flex アプリケーションには、以下の URL からアクセスできます。

- *http://demo.borland.com/flex/SilkTest2011/index.html*

サポートするコントロール

Flex のテストで利用可能なコントロールの完全な一覧については、「*API* リファレンス」の com.borland.silktest.jtf.flex パッケージ内でサポートされる Flex クラスの一覧を参照してください。

大抵の Flex アプリケーションは、カスタム コントロールを含んでいます。 カスタム Flex コントロールをテストするために、Silk4J を設定することが可能です。 カスタム コントロールをテストするための Silk4J の設定方法に関する詳細については、「*Flex* カスタム コントロールをテストに含める」を参照してください。

サポートされている属性の一覧については、「サポートする属性の種類」を参照してください。

Adobe Flex アプリケーションのスタイル

Adobe Flex 3.x で開発されたアプリケーションについて、SilkTest Workbench ではスタイルとプロパティを区別しません。 この結果、スタイルはプロパティとして公開されます。 ただし、Adobe Flex 4.x の Spark という接頭辞が付いているすべての新しい Flex コントロール（SparkButton など）では、スタイルがプロパティとして公開されません。 この結果、Flex 4.x コントロールの GetProperty() メソッドおよび GetPropertyList() メソッドでは color や fontSize などのスタイルが返されず、text や name などのプロパティのみが返されます。

GetStyle(string styleName) メソッドは、スタイルの値を文字列として返します。 どのようなスタイルが存在するかを確認するには、Adobe ヘルプ（*http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/package-detail.html*）を参照してください。

スタイルが設定されていない場合は、再生中に StyleNotSetException が発生します。

FlexTree などの Flex 3.x コントロールでは、GetProperty() を使用してスタイルを取得できます。 GetStyle() を使用することもできます。 Flex 3.x コントロールでは、GetProperty() メソッドと GetStyle() メソッドの両方が動作します。

色スタイルの計算

Flex では、色は数値として表されます。 色は、以下の式を使用して計算できます。

```
red*65536 + green*256 + blue
```

例

以下のスクリプト例では、フォント サイズが 12 であるかどうかを検証しています。 16711680 という数値は、255*65536 + 0*256 + 0 という式から算出されます。 この値は赤色を表し、スクリプトは背景色に対してこの色を検証します。

```
Assert.That(control.GetStyle("fontSize"), [Is].EqualTo("12"))
Assert.That(label.GetStyle("backgroundColor"), [Is].EqualTo("16711680"))
```

Adobe Flash Player のセキュリティ制約に対応するための Flex アプリケーションの構成

Adobe Flash Player 10 では、セキュリティ モデルが以前のバージョンから変更されています。 Flash Player を使用するテストを記録する場合、記録は想定どおりに動作します。 ただし、テストを再生する場合は、特定の状況で高レベルのクリックが行われると、予期しない結果が発生します。 たとえば、**ファイル参照** ダイアログ ボックスをプログラムから開くことができません。このシナリオの再生を試みると、セキュリティ制約が原因でテストに失敗します。

このセキュリティ制約を回避するには、ダイアログ ボックスを開くボタンに対して低レベルのクリックを実行します。 低レベルのクリックを作成するには、click メソッドにパラメータを追加します。

たとえば、SparkButton::Click() の代わりに SparkButton::Click(1) を使用します。 パラメータを指定しない Click() は高レベルのクリックとして再生され、パラメータを指定したクリック（ボタンなど）は低レベルのクリックとして再生されます。

1. Flash Player を使用するテストを記録します。
2. Click メソッドに移動して、パラメータを追加します。
   たとえば、**ファイルを開く** ダイアログ ボックスを開くには、以下のように指定します。

   ```
   SparkButton("@caption='Open File Dialog...'").Click(1)
   ```

   テストを再生すると、想定どおりに動作します。

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。

Adobe Flex カスタム コントロールのテスト

Silk4J では、Flex カスタム コントロールのテストがサポートされています。 デフォルトで、Silk4J では、カスタム コントロールの個別のサブコントロールに対する記録および再生のサポートが提供されます。

カスタム コントロールをテストする場合、以下のオプションが存在します。

- 基本サポート

  基本サポートでは、動的呼び出しを使用して、再生中にカスタム コントロールと対話します。 作業量が少なく済むこのアプローチは、テスト アプリケーションにおいて、Silk4J が公開しないカスタム コントロールのプロパティおよびメソッドにアクセスする場合に使用します。 カスタム コントロールの開発者は、コントロールのテストを容易にすることのみを目的としたメソッドおよびプロパティをカスタム コントロールに追加することもできます。 ユーザーは、動的呼び出し機能を使用してこれらのメソッドやプロパティを呼び出すことができます。

  基本サポートには以下のような利点があります。

  - 動的呼び出しでは、テスト アプリケーションのコードを変更する必要がありません。
  - 動的呼び出しを使用することによって、ほとんどのテストのニーズを満たすことができます。

  基本サポートには以下のような短所があります。

  - ロケーターには、具体的なクラス名が組み込まれません (たとえば、Silk4J では「//FlexSpinner」ではなく「//FlexBox」と記録されます)。
  - 記録のサポートが限定されます。
  - Silk4J では、イベントを再生できません。

  例を含む動的呼び出しの詳細については、「*Flex* メソッドの動的呼び出し」を参照してください。
- 高度なサポート

  高度なサポートでは、カスタム コントロールに対して、特定のオートメーション サポートを作成できます。 この追加のオートメーション サポートによって、記録のサポートおよびより強力な再生のサポートが提供されます。 高度なサポートには以下のような利点があります。

  - イベントの記録と再生を含む、高レベルの記録および再生のサポートが提供されます。
  - Silk4J では、カスタム コントロールが他のすべての組み込み Flex コントロールと同様に処理されます。
  - Silk4J API とシームレスに統合できます。
  - Silk4J では、ロケーターで具体的なクラス名が使用されます (たとえば、Silk4J では「//FlexSpinner」と記録されます)。

  高度なサポートには以下のような短所があります。

  - 実装作業が必要です。 テスト アプリケーションを変更し、Open Agent を拡張する必要があります。

テスト アプリケーションでのカスタム コントロールの定義

通常、テスト アプリケーションには、アプリケーションの開発中に追加されたカスタム コントロールがすでに含まれています。 テスト アプリケーションにすでにカスタム コントロールが含まれている場合は、「動的呼び出しを使用して *Flex* カスタム コントロールをテストする」または「オートメーション サポートを使用してカスタム コントロールをテストする」に進んでください。

この手順では、Flex アプリケーション開発者が Flex で Spinner カスタム コントロールを作成する方法を示します。 このトピックで作成する Spinner カスタム コントロールは、カスタム コントロールの実装およびテストのプロセスを説明するために、いくつかのトピックで使用されています。

Spinner カスタム コントロールは、以下のグラフィックに示すように、2 つのボタンと 1 つのテキスト フィールドを含んでいます。

ユーザーは、**Down** をクリックしてテキスト フィールドに表示されている値を 1 減分させ、**Up** をクリックしてテキスト フィールドの値を 1 増分させることができます。

カスタム コントロールには、設定および取得が可能なパブリックの CurrentValue プロパティが用意されています。

1. テスト アプリケーションで、コントロールのレイアウトを定義します。
   たとえば、Spinner コントロール タイプでは、以下のように記述します。

```
<?xml version="1.0" encoding="utf-8"?>
<customcontrols:SpinnerClass xmlns:mx="http://www.adobe.com/2006/mxml"
xmlns:controls="mx.controls.*" xmlns:customcontrols="customcontrols.*">
    <controls:Button id="downButton" label="Down" />
    <controls:TextInput id="text" enabled="false" />
    <controls:Button id="upButton" label="Up"/>
</customcontrols:SpinnerClass>
```

2. カスタム コントロールの実装を定義します。
   たとえば、Spinner コントロール タイプでは、以下のように記述します。

```
package customcontrols
{
    import flash.events.MouseEvent;

    import mx.containers.HBox;
    import mx.controls.Button;
    import mx.controls.TextInput;
    import mx.core.UIComponent;
    import mx.events.FlexEvent;

    [Event(name="increment",        type="customcontrols.SpinnerEvent")]
    [Event(name="decrement",        type="customcontrols.SpinnerEvent")]

    public class SpinnerClass extends HBox
    {
        public var downButton : Button;
        public var upButton : Button;
        public var text : TextInput;
        public var ssss: SpinnerAutomationDelegate;
        private var _lowerBound : int = 0;
        private var _upperBound : int = 5;

        private var _value : int = 0;
        private var _stepSize : int = 1;


        public function SpinnerClass() {
            addEventListener(FlexEvent.CREATION_COMPLETE,
creationCompleteHandler);
        }
```

```
        private function creationCompleteHandler(event:FlexEvent) : void {
            downButton.addEventListener(MouseEvent.CLICK, downButtonClickHandler);
            upButton.addEventListener(MouseEvent.CLICK, upButtonClickHandler);
            updateText();
        }

        private function downButtonClickHandler(event : MouseEvent) : void {
            if(currentValue - stepSize >= lowerBound) {
                currentValue = currentValue - stepSize;
            }
            else {
                currentValue = upperBound - stepSize + currentValue - lowerBound
+ 1;
            }

            var spinnerEvent : SpinnerEvent = new
SpinnerEvent(SpinnerEvent.DECREMENT);
            spinnerEvent.steps = _stepSize;
            dispatchEvent(spinnerEvent);
        }

        private function upButtonClickHandler(event : MouseEvent) : void {
            if(currentValue <= upperBound - stepSize) {
                currentValue = currentValue + stepSize;
            }
            else {
                currentValue = lowerBound + currentValue + stepSize -
upperBound - 1;
            }

            var spinnerEvent : SpinnerEvent = new
SpinnerEvent(SpinnerEvent.INCREMENT);
            spinnerEvent.steps = _stepSize;
            dispatchEvent(spinnerEvent);
        }

        private function updateText() : void {
            if(text != null) {
                text.text = _value.toString();
            }
        }

        public function get currentValue() : int {
            return _value;
        }

        public function set currentValue(v : int) : void {
            _value = v;
            if(v < lowerBound) {
                _value = lowerBound;
            }
            else if(v > upperBound) {
                _value = upperBound;
            }
            updateText();

        }

        public function get stepSize() : int {
```

```
            return _stepSize;
        }

        public function set stepSize(v : int) : void {
            _stepSize = v;
        }

        public function get lowerBound() : int {
            return _lowerBound;
        }

        public function set lowerBound(v : int) : void {
            _lowerBound = v;
            if(currentValue < lowerBound) {
                currentValue = lowerBound;
            }
        }

        public function get upperBound() : int {
            return _upperBound;
        }

        public function set upperBound(v : int) : void {
            _upperBound = v;
            if(currentValue > upperBound) {
                currentValue = upperBound;
            }
        }
    }
}
```

3. コントロールが使用するイベントを定義します。
たとえば、Spinner コントロール タイプでは、以下のように記述します。

```
package customcontrols
{
    import flash.events.Event;

    public class SpinnerEvent extends Event
    {
        public static const INCREMENT : String = "increment";
        public static const DECREMENT : String = "decrement";

        private var _steps : int;

        public function SpinnerEvent(eventName : String) {
            super(eventName);
        }

        public function set steps(value:int) : void {
            _steps = value;
        }

        public function get steps() : int {
            return _steps;
        }

    }
}
```

次のステップでは、テスト アプリケーションのオートメーション サポートを実装します。

動的呼び出しを使用した Flex カスタム コントロールのテスト

Silk4J では、動的呼び出しを使用したカスタム コントロールの記録と再生のサポートが提供されており、これにより再生中にカスタム コントロールを操作できます。 作業量が少なく済むこのアプローチは、テスト アプリケーションにおいて、Silk4J が公開しないカスタム コントロールのプロパティおよびメソッドにアクセスする場合に使用します。 カスタム コントロールの開発者は、コントロールのテストを容易にすることのみを目的としたメソッドおよびプロパティをカスタム コントロールに追加することもできます。

1. 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。
2. 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。
3. 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。
4. 動的プロパティーを取得する場合には getProperty メソッドを、動的プロパティーを設定する場合には、 setProperty メソッドを使用します。

例

この例では、以下の図に示すように、2 つのボタンと 1 つのテキスト フィールドを含む Spinner カスタム コントロールをテストします。

ユーザーは、**Down** をクリックしてテキスト フィールドに表示されている値を 1 減分させ、**Up** をクリックしてテキスト フィールドの値を 1 増分させることができます。

カスタム コントロールには、設定および取得が可能なパブリックの CurrentValue プロパティが用意されています。

Spinner の値を 4 に設定するには、以下のように入力します。

```
FlexBox spinner = _desktop.<FlexBox>find("//
FlexBox[@className=customcontrols.Spinner]");
spinner.setProperty("CurrentValue", 4);
```

オートメーション サポートを使用したカスタム コントロールのテスト

カスタム コントロールに対して、特定のオートメーション サポートを作成できます。 この追加のオートメーション サポートによって、記録のサポートおよびより強力な再生のサポートが提供されます。 オートメーション サポートを作成するには、テスト アプリケーションを変更し、Open Agent を拡張する必要があります。

カスタム コントロールをテストする前に、以下のステップを実行します。

- テスト アプリケーションでのカスタム コントロールの定義
- オートメーション サポートの実装

テスト アプリケーションを変更してオートメーション サポートを組み込んだあと、以下のステップを実行します。

1. テスト内のカスタム コントロールをテストするために、カスタム コントロールの Java クラスを作成します。
   たとえば、Spinner コントロール クラスは、以下の内容を含んでいる必要があります。

```
package customcontrols;

import com.borland.silktest.jtf.Desktop;
```

```
import com.borland.silktest.jtf.common.JtfObjectHandle;
import com.borland.silktest.jtf.flex.FlexBox;

/**
 * Implementation of the FlexSpinner Custom Control.
 */
public class FlexSpinner extends FlexBox {

  protected FlexSpinner(JtfObjectHandle handle, Desktop desktop) {
    super(handle, desktop);
  }

  @Override
  protected String getCustomTypeName() {
    return "FlexSpinner";
  }

  public Integer getLowerBound() {
    return (Integer) getProperty("lowerBound");
  }

  public Integer getUpperBound() {
    return (Integer) getProperty("upperBound");
  }

  public Integer getCurrentValue() {
    return (Integer) getProperty("currentValue");
  }

  public void setCurrentValue(Integer currentValue) {
    setProperty("currentValue", currentValue);
  }

  public Integer getStepSize() {
    return (Integer) getProperty("stepSize");
  }

  public void increment(Integer steps) {
    invoke("Increment", steps);
  }

  public void decrement(Integer steps) {
    invoke("Decrement", steps);
  }

}
```

2. この Java クラスをテストが含まれている Silk4J テスト プロジェクトに追加します。

   ヒント: 同じカスタム コントロールを複数の Silk4J プロジェクトで使用する場合は、カスタム コントロールを含む別個のプロジェクトを作成し、Silk4J テスト プロジェクトからそれを参照することをお勧めします。

3. <SilkTest のインストール ディレクトリ>¥ng¥agent¥plugins¥com.borland.silktest.jtf.agent.customcontrols_<バージョン>¥config¥classMapping.properties ファイルに以下の行を追加します。

   ```
   FlexSpinner=customcontrols.FlexSpinner
   ```

   等号記号の左側のコードは、XML ファイルに定義されているカスタム コントロール名である必要があります。 等号記号の右側のコードは、カスタム コントロールの Java クラスの完全修飾名である必要があります。

これで、Silk4J でカスタム コントロールを使用する場合に、記録および再生が完全にサポートされるようになりました。

使用例

以下の例は、オートメーションの委譲に実装されている「Increment」メソッドを使用して、Spinner の値を 3 つ増分する方法を示しています。

```
desktop.<FlexSpinner>find("//FlexSpinner[@caption='index:
1']").increment(3);
```

以下の例は、Spinner の値を 3 に設定する方法を示しています。

```
desktop.<FlexSpinner>find("//FlexSpinner[@caption='index:
1']").setCurrentValue(3);
```

カスタム コントロールのオートメーション サポートの実装

カスタム コントロールをテストする前に、カスタム コントロールの ActionScript でオートメーション サポート（オートメーションの委譲）を実装し、テスト アプリケーションにコンパイルします。

以下の手順では、Flex のカスタム Spinner コントロールを使用して、カスタム コントロールのオートメーション サポートの実装方法を示します。 Spinner カスタム コントロールは、以下のグラフィックに示すように、2 つのボタンと 1 つのテキスト フィールドを含んでいます。

ユーザーは、**Down** をクリックしてテキスト フィールドに表示されている値を 1 減分させ、**Up** をクリックしてテキスト フィールドの値を 1 増分させることができます。

カスタム コントロールには、設定および取得が可能なパブリックの CurrentValue プロパティが用意されています。

1. カスタム コントロールの ActionScript でオートメーション サポート（オートメーションの委譲）を実装します。
   オートメーションの委譲の実装の詳細については、Adobe Live ドキュメント（*http://livedocs.adobe.com/flex/3/html/help.html?content=functest_components2_14.html*）を参照してください。
   この例では、オートメーションの委譲によって、「increment」および「decrement」メソッドに対してサポートが追加されます。 オートメーションの委譲のコード例は以下のとおりです。

```
package  customcontrols
{
      import  flash.display.DisplayObject;
      import  mx.automation.Automation;
      import  customcontrols.SpinnerEvent;
      import  mx.automation.delegates.containers.BoxAutomationImpl;
      import  flash.events.Event;
      import  mx.automation.IAutomationObjectHelper;
      import  mx.events.FlexEvent;
      import  flash.events.IEventDispatcher;
      import  mx.preloaders.DownloadProgressBar;
      import  flash.events.MouseEvent;
      import  mx.core.EventPriority;

      [Mixin]
      public  class  SpinnerAutomationDelegate  extends  BoxAutomationImpl
      {

            public  static  function  init(root:DisplayObject)  :  void  {
```

```
            // register delegate for the automation
            Automation.registerDelegateClass(Spinner, SpinnerAutomationDelegate);
        }

        public function SpinnerAutomationDelegate(obj:Spinner) {
            super(obj);
            // listen to the events of interest (for recording)
            obj.addEventListener(SpinnerEvent.DECREMENT, decrementHandler);
            obj.addEventListener(SpinnerEvent.INCREMENT, incrementHandler);
        }

        protected function decrementHandler(event : SpinnerEvent) : void {
            recordAutomatableEvent(event);
        }

        protected function incrementHandler(event : SpinnerEvent) : void {
            recordAutomatableEvent(event);
        }

        protected function get spinner() : Spinner {
            return uiComponent as Spinner;
        }

        //---------------------------------
        //   override functions
        //---------------------------------

        override public function get automationValue():Array {
            return [ spinner.currentValue.toString() ];
        }

        private function replayClicks(button : IEventDispatcher, steps : int) : Boolean
{
            var helper : IAutomationObjectHelper =
Automation.automationObjectHelper;
            var result : Boolean;
            for(var i:int; i < steps; i++) {
                helper.replayClick(button);
            }
            return result;
        }

        override public function replayAutomatableEvent(event:Event):Boolean {

            if(event is SpinnerEvent) {
                var spinnerEvent : SpinnerEvent = event as SpinnerEvent;
                if(event.type == SpinnerEvent.INCREMENT) {
                    return replayClicks(spinner.upButton, spinnerEvent.steps);
                }
                else if(event.type == SpinnerEvent.DECREMENT) {
                    return replayClicks(spinner.downButton, spinnerEvent.steps);
                }
                else {
                    return false;
                }

            }
            else {
                return super.replayAutomatableEvent(event);
```

```
            }
        }

        // do not expose the child controls (i.e the buttons and the textfield) as individual controls
        override public function get numAutomationChildren():int {
            return 0;
        }

    }
}
```

2. Open Agent にオートメーションの委譲を導入するために、カスタム コントロールを記述する XML ファイルを作成します。
クラス定義ファイルには、インストルメント化されたすべての Flex コンポーネントについての情報が含まれています。 このファイルでは、記録中にイベントを送信でき、再生中にイベントを受け取ることができるコンポーネントについての情報が提供されます。 クラス定義ファイルには、サポートされているプロパティの定義も含まれています。
Spinner カスタム コントロールの XML ファイルは以下のようになります。

```
<?xml version="1.0" encoding="UTF-8"?>
<TypeInformation>
    <ClassInfo Name="FlexSpinner" Extends="FlexBox">
        <Implementation
            Class="customcontrols.Spinner" />
        <Events>
            <Event Name="Decrement">
                <Implementation
                    Class="customcontrols.SpinnerEvent"
                    Type="decrement" />
                <Property Name="steps">
                    <PropertyType Type="integer" />
                </Property>
            </Event>
            <Event Name="Increment">
                <Implementation
                    Class="customcontrols.SpinnerEvent"
                    Type="increment" />
                <Property Name="steps">
                    <PropertyType Type="integer" />
                </Property>
            </Event>
        </Events>
        <Properties>
            <Property Name="lowerBound" accessType="read">
                <PropertyType Type="integer" />
            </Property>
            <Property Name="upperBound" accessType="read">
                <PropertyType Type="integer" />
            </Property>
            <!-- expose read and write access for the currentValue property -->
            <Property Name="currentValue" accessType="both">
                <PropertyType Type="integer" />
            </Property>
            <Property Name="stepSize" accessType="read">
                <PropertyType Type="integer" />
            </Property>
        </Properties>
    </ClassInfo>
</TypeInformation>
```

3. サポートされている Flex コントロールのすべてのクラス、およびそのメソッドとプロパティを記述するすべての XML ファイルが格納されるフォルダに、カスタム コントロールの XML ファイルを配置します。

   SilkTest には、サポートされている Flex コントロールのすべてのクラス、およびそのメソッドとプロパティを記述するいくつかの XML ファイルが含まれています。 これらの XML ファイルは、<<SilkTest_install_directory>¥ng¥agent¥plugins¥com.borland.fastxd.techdomain.flex.agent_<バージョン>¥config¥automationEnvironment フォルダにあります。

   独自の XML ファイルを提供する場合は、XML ファイルをこのフォルダにコピーする必要があります。 Open Agent が起動して、Adobe Flex のサポートを初期化する場合、このディレクトリの内容が読み込まれます。

   Flex の Spinner サンプル コントロールをテストするには、CustomControls.xml ファイルをこのフォルダにコピーする必要があります。 Open Agent が現在実行されている場合は、ファイルをフォルダにコピーしたあと、Open Agent を再起動します。

Flex クラス定義ファイル

クラス定義ファイルには、インストルメント化されたすべての Flex コンポーネントについての情報が含まれています。 このファイルでは、記録中にイベントを送信でき、再生中にイベントを受け取ることができるコンポーネントについての情報が提供されます。 クラス定義ファイルには、サポートされているプロパティの定義も含まれています。

SilkTest には、Flex の共通コントロールおよび特殊化されたコントロールのすべてのクラス、イベント、およびプロパティを記述するいくつかの XML ファイルが含まれています。 これらの XML ファイルは、<SilkTest_install_directory>¥ng¥agent¥plugins¥com.borland.fastxd.techdomain.flex.agent_<バージョン>¥config¥automationEnvironment フォルダにあります。

独自の XML ファイルを提供する場合は、XML ファイルをこのフォルダにコピーする必要があります。 SilkTest Agent が起動して Adobe Flex のサポートを初期化するとき、このディレクトリの内容が読み込まれます。

XML ファイルの基本的な構造は以下のとおりです。

```
<TypeInformation>

<ClassInfo>

<Implementation />

<Events>

<Event />

...

</Events>

<Properties>

<Property />

...

</Properties>

</ClassInfo>

</TypeInformation>
```

Flex メソッドの動的呼び出し

動的呼び出し機能を使用して Silk4J が対象としないコントロールのメソッドを呼び出したり、プロパティを取得/設定することができます。 この機能は、カスタム コントロールを使用したり、カスタマイズせずに Silk4J がサポートするコントロールを使用する場合に有効です。

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。

動的プロパティーを取得する場合には getProperty メソッドを、動的プロパティーを設定する場合には、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。

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。

サポートされているメソッドおよびプロパティ

次のメソッドとプロパティを呼び出すことができます。

- Silk4J がサポートするコントロールのメソッドとプロパティ
- Flex API で定義されているすべてのパブリック メソッド
- コントロールが標準コントロールから派生したカスタム コントロールの場合、標準コントロールが呼び出すことのできるすべてのメソッドとプロパティ

サポートされているパラメータ型

次のパラメータ型がサポートされます。

- すべての組み込み Silk4J 型

  Silk4J 型には、プリミティブ型 (boolean、int、string など)、リスト、およびその他の型 (Point など) が含まれます。

戻り値

プロパティや戻り値を持つメソッドの場合は、次の値が返されます。

- すべての組み込み Silk4J 型の場合は正しい値。 これらの型は、「サポートされているパラメータ型」のセクションに記載されています。
- 戻り値を持たないすべてのメソッドの場合は、null が返されます。

Java AWT/Swing アプリケーションおよびアプレット

Silk4J は、Java AWT/Swing コントロールを使用するアプリケーションおよびアプレットのテストを組み込みでサポートしています。

注: Java AWT/Swing アプリケーションまたはアプレットに埋め込まれた Java SWT コントロールや、Java SWT アプリケーションに埋め込まれた Java AWT/Swing コントロールもテストできます。

サンプル アプリケーション、プロジェクト、スクリプト

Silk4J には、サンプルのアプリケーション、プロジェクト、およびスクリプトがあります。 *http://techpubs.borland.com/silk_gauntlet/SilkTest/* から、サンプル アプリケーションをダウンロードします。 ダウンロードしたサンプル アプリケーションには、サンプル Silk4J プロジェクトとスクリプトも含まれています。

サンプル アプリケーションをインストールしたあと、**スタート** > **すべてのプログラム** > **Silk** > **SilkTest** > **Sample Applications** > **Java Swing** を選択し、サンプル アプリケーションを選択してください。

注: Windows Vista および Windows 7.0 の場合、ファイルの場所は、Documents and Settings¥All Users¥Shared Documents¥SilkTest¥samples¥ ではなく、Users¥Public¥Documents¥SilkTest¥samples¥Silk4J¥ となります。

注: サンプル スクリプトは、テスト対象アプリケーションの場所へのパスを基本状態の中に保持しています。 SilkTest がデフォルトの場所にインストールされていない場合には、基本状態を編集して、正しい場所を参照するようパスを調整してください。

サポートするバージョン、既知の問題、および回避策についての最新の情報は、リリース ノートを確認してください。

サポートするコントロール

Java AWT/Swing のテストで利用可能なコントロールの完全な一覧については、「*API* リファレンス」の以下のパッケージ内でサポートされる Swing クラスの一覧を参照してください。

- com.borland.silktest.jtf.swing : Java Swing 固有のクラスを含みます。
- com.borland.silktest.jtf.common.types : データ型を含みます。

サポートされている属性の一覧については、「サポートする属性の種類」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 メソッドの動的な呼び出し

動的呼び出し機能を使用して Silk4J が対象としないコントロールのメソッドを呼び出したり、プロパティを取得/設定することができます。 この機能は、カスタム コントロールを使用したり、カスタマイズせずに Silk4J がサポートするコントロールを使用する場合に有効です。

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。

動的プロパティーを取得する場合には getProperty メソッドを、動的プロパティーを設定する場合には、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。

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。

サポートされているメソッドおよびプロパティ

次のメソッドとプロパティを呼び出すことができます。

- Silk4J がサポートするコントロールのメソッドとプロパティ
- SWT、AWT、または Swing ウィジェットのすべてのパブリック メソッド
- コントロールが標準コントロールから派生したカスタム コントロールの場合、標準コントロールが呼び出すことのできるすべてのメソッドとプロパティ

サポートされているパラメータ型

次のパラメータ型がサポートされます。

- プリミティブ型 (boolean、integer、long、double、string)

  プリミティブ型 (int など) とオブジェクト タイプ (java.lang.Integer など) の両方がサポートされます。 プリミティブ型は必要に応じて拡大変換されます。たとえば、long が必要な場所で int を渡すことができます。
- 列挙型

  列挙パラメータは文字列として渡す必要があります。 文字列は、列挙値の名前と一致しなければなりません。 たとえば、メソッドが列挙型 java.sql.ClientInfoStatus のパラメータを必要とする場合、REASON_UNKNOWN、REASON_UNKNOWN_PROPERTY、REASON_VALUE_INVALID、または REASON_VALUE_TRUNCATED の文字列値を使用できます。
- リスト

  リスト、配列、または可変長引数のパラメータを持つメソッドを呼び出すことができます。 リストの要素がターゲットの配列型に代入可能の場合、配列型への変換は自動的に行われます。
- その他のコントロール

  コントロール パラメータは、TestObject として渡したり、返したりできます。

戻り値

プロパティや戻り値を持つメソッドの場合は、次の値が返されます。

- すべての組み込み Silk4J 型の場合は正しい値。 これらの型は、「サポートされているパラメータ型」のセクションに記載されています。
- 戻り値を持たないすべてのメソッドの場合は、null が返されます。

Silk4J を構成して、Java Network Launching Protocol（JNLP）を使用するアプリケーションを起動する

Java Network Launching Protocol（JNLP）を使用して起動するアプリケーションでは、Silk4J に追加の構成が必要です。 これらのアプリケーションは Web から起動されるため、実際のアプリケーションと「Web Start」を起動するように、アプリケーション構成を手動で構成する必要があります。 このようにしない場合、アプリケーションがすでに実行されていないかぎり、テストを再生すると、失敗します。

1. テスト アプリケーションのプロジェクトとテスト クラスを作成します。
2. SilkTest ツール バー アイコン 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。
3. 基本状態を編集して、再生中に Web Start が起動されるようにします。

a) **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。
b) **実行可能ファイル** テキスト ボックスに、javaws.exe への絶対パスを入力します。
たとえば、以下のように入力します。

```
%ProgramFiles%¥Java¥jre6¥bin¥javaws.exe
```

c) **コマンド ライン引数** テキスト ボックスに、Web Start への URL を含むコマンド ライン パターンを入力します。

```
"<url-to-jnlp-file>"
```

たとえば、SwingSet3 アプリケーションの場合、以下のように入力します。

```
"http://download.java.net/javadesktop/swingset3/SwingSet3.jnlp"
```

d) **OK** をクリックします。

4. **OK** をクリックします。
テストは、基本状態を使用し、Web 起動アプリケーションとアプリケーション構成の実行可能パターンを開始して javaw.exe に接続し、テストを実行します。

テストを実行すると、アプリケーション構成の EXE ファイルが基本状態の EXE ファイルと一致しないという警告が表示されます。 テストは予想したとおりに実行されているため、このメッセージは無視できます。

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

Silk4J は、SWT（Standard Widget Toolkit）コントロールのウィジェットを使用したアプリケーションのテストをサポートします。

Silk4J は、以下をサポートします。

- Java AWT/Swing アプリケーションに埋め込まれた Java SWT コントロール、および Java SWT アプリケーションに埋め込まれた Java AWT/Swing コントロールのテスト。
- IBM JDK または Sun JDK を使用する Java SWT アプリケーションのテスト。
- レンダリングに SWT ウィジェットを使用する Eclipse ベースのアプリケーション。 Silk4J は、Eclipse IDE ベース、および RCP ベースの両方のアプリケーションをサポートします。

サンプル アプリケーション、プロジェクト、スクリプト

Silk4J には、サンプルのアプリケーション、プロジェクト、およびスクリプトがあります。 *http://techpubs.borland.com/silk_gauntlet/SilkTest/* から、サンプル アプリケーションをダウンロードしま

す。 ダウンロードしたサンプル アプリケーションには、サンプル Silk4J プロジェクトとスクリプトも含まれています。

サンプル アプリケーションをインストールしたあと、**スタート** > **すべてのプログラム** > **Silk** > **SilkTest** > **Sample Applications** > **Java SWT** を選択し、使用するサンプル アプリケーションを選択してください。

Documents and Settings¥All Users¥Shared Documents¥SilkTest¥samples¥Silk4J¥SWT から、Java SWT サンプル プロジェクトをインポートします。 製品をより理解するために、サンプル スクリプトを実行してみてください。

注: Windows Vista および Windows 7.0 の場合、ファイルの場所は、Documents and Settings¥All Users¥Shared Documents¥SilkTest¥samples¥ ではなく、Users¥Public¥Documents¥SilkTest¥samples¥Silk4J¥ となります。

注: サンプル スクリプトは、テスト対象アプリケーションの場所へのパスを基本状態の中に保持しています。 SilkTest がデフォルトの場所にインストールされていない場合には、基本状態を編集して、正しい場所を参照するようパスを調整してください。

サポートするバージョン、既知の問題、および回避策についての最新の情報は、リリース ノートを確認してください。

サポートするコントロール

Java SWT のテストで利用可能なコントロールの完全な一覧については、「*API* リファレンス」の以下のパッケージ内でサポートされる Java SWT クラスの一覧を参照してください。

- com.borland.silktest.jtf.swt：Java SWT 固有のクラスを含みます。
- com.borland.silktest.jtf： SilkTest Classic4Test の winclass.inc のような、共通のクラスを含みます。

サポートされている属性の一覧については、「サポートする属性の種類」を参照してください。

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。

Java メソッドの動的な呼び出し

動的呼び出し機能を使用して Silk4J が対象としないコントロールのメソッドを呼び出したり、プロパティを取得/設定することができます。 この機能は、カスタム コントロールを使用したり、カスタマイズせずに Silk4J がサポートするコントロールを使用する場合に有効です。

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。

動的プロパティーを取得する場合には getProperty メソッドを、動的プロパティーを設定する場合には、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。

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。

サポートされているメソッドおよびプロパティ

次のメソッドとプロパティを呼び出すことができます。

- Silk4J がサポートするコントロールのメソッドとプロパティ
- SWT、AWT、または Swing ウィジェットのすべてのパブリック メソッド
- コントロールが標準コントロールから派生したカスタム コントロールの場合、標準コントロールが呼び出すことのできるすべてのメソッドとプロパティ

サポートされているパラメータ型

次のパラメータ型がサポートされます。

- プリミティブ型 (boolean、integer、long、double、string)

  プリミティブ型 (int など) とオブジェクト タイプ (java.lang.Integer など) の両方がサポートされます。 プリミティブ型は必要に応じて拡大変換されます。たとえば、long が必要な場所で int を渡すことができます。
- 列挙型

  列挙パラメータは文字列として渡す必要があります。 文字列は、列挙値の名前と一致しなければなりません。 たとえば、メソッドが列挙型 java.sql.ClientInfoStatus のパラメータを必要とする場合、REASON_UNKNOWN、REASON_UNKNOWN_PROPERTY、REASON_VALUE_INVALID、または REASON_VALUE_TRUNCATED の文字列値を使用できます。
- リスト

  リスト、配列、または可変長引数のパラメータを持つメソッドを呼び出すことができます。 リストの要素がターゲットの配列型に代入可能の場合、配列型への変換は自動的に行われます。
- その他のコントロール

  コントロール パラメータは、TestObject として渡したり、返したりできます。

戻り値

プロパティや戻り値を持つメソッドの場合は、次の値が返されます。

- すべての組み込み Silk4J 型の場合は正しい値。 これらの型は、「サポートされているパラメータ型」のセクションに記載されています。
- 戻り値を持たないすべてのメソッドの場合は、null が返されます。

Windows API ベースのクライアント/サーバー アプリケーション

Silk4J は、Microsoft Windows API ベースのアプリケーションのテストを組み込みでサポートしています。

サンプル アプリケーション、プロジェクト、スクリプト

Silk4J には、サンプルのアプリケーション、プロジェクト、およびスクリプトがあります。 *http://techpubs.borland.com/silk_gauntlet/SilkTest/* から、サンプル アプリケーションをダウンロードします。 ダウンロードしたサンプル アプリケーションには、サンプル Silk4J プロジェクトとスクリプトも含まれています。

サンプル アプリケーションをインストールしたあと、**スタート** > **すべてのプログラム** > **Silk** > **SilkTest** > **Sample Applications** > **Win32** を選択し、使用するサンプル アプリケーションを選択してください。

注: Windows Vista および Windows 7.0 の場合、ファイルの場所は、Documents and Settings¥All Users¥Shared Documents¥SilkTest¥samples¥ ではなく、Users¥Public¥Documents¥SilkTest¥samples¥Silk4J¥ となります。

注: サンプル スクリプトは、テスト対象アプリケーションの場所へのパスを基本状態の中に保持しています。 SilkTest がデフォルトの場所にインストールされていない場合には、基本状態を編集して、正しい場所を参照するようパスを調整してください。

サポートするバージョン、既知の問題、および回避策についての最新の情報は、リリース ノートを確認してください。

サポートするコントロール

Windows API ベースのテストで利用可能なコントロールの完全な一覧については、「*API* リファレンス」の以下のパッケージ内でサポートされる Windows クラスの一覧を参照してください。

- com.borland.silktest.jtf.win32 : Windows API 固有のクラスを含みます。
- com.borland.silktest.jtf : SilkTest Classic 4Test の winclass.inc のような、共通のクラスを含みます。

サポートされている属性の一覧については、「サポートする属性の種類」を参照してください。

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。

Rumba のサポート

Rumba は、世界トップクラスの Windows デスクトップ端末エミュレーション ソリューションです。 SilkTest は、Rumba の記録および再生を組み込みでサポートしています。

Rumba でのテスト時には、以下の点を考慮してください。

- Rumba のバージョンは、Silk4J のバージョンと互換性がある必要があります。 バージョン 8.1 以前の Rumba はサポートされていません。
- Rumba のグリーン スクリーンの周囲にあるコントロールはすべて WPF の基本機能 (または Win32) を使用しています。
- サポートされている Rumba デスクトップ タイプは、以下のとおりです。
    - メインフレーム ディスプレイ
    - AS400 ディスプレイ

Rumba テストで使用できる記録および再生のコントロールの完全な一覧については、「*Rumba* クラス リファレンス」を参照してください。

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

Silk4J は、SAP コントロールを使用したアプリケーションのテストをサポートしています。

注: Internet Explorer や Firefox 上から SAP NetWeaver を使用する場合、SilkTest Workbench は、xBrowser テクノロジ ドメインを使用してアプリケーションをテストします。

サポートするバージョン、既知の問題、および回避策についての最新の情報は、リリース ノートを確認してください。

注: SAP アプリケーションでは、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 を設定する必要があります。 デフォルト設定を変更するには、**スクリプト オプション** ダイアログ ボックスで、**OPT_ALTERNATE_RECORD_BREAK** チェック ボックスをオンにします。

サポートするコントロール

SAP のテストで利用可能なコントロールの完全な一覧については、「*API* リファレンス」の com.borland.silktest.jtf.sap パッケージ内でサポートされる SAP クラスの一覧を参照してください。

サポートされている属性の一覧については、「サポートする属性の種類」を参照してください。

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。

SAP メソッドの動的な呼び出し

動的呼び出し機能を使用して Silk4J が対象としないコントロールのメソッドを呼び出したり、プロパティを取得/設定することができます。 この機能は、カスタム コントロールを使用したり、カスタマイズせずに Silk4J がサポートするコントロールを使用する場合に有効です。

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。

動的プロパティーを取得する場合には getProperty メソッドを、動的プロパティーを設定する場合には、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。

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。

サポートされているメソッドおよびプロパティ

次のメソッドとプロパティを呼び出すことができます。

- Silk4J がサポートするコントロールのメソッドとプロパティ
- SAP オートメーション インターフェイスによって定義されているすべての public メソッド
- コントロールが標準コントロールから派生したカスタム コントロールの場合、標準コントロールが呼び出すことのできるすべてのメソッドとプロパティ

サポートされているパラメータ型

次のパラメータ型がサポートされます。

- すべての組み込み Silk4J 型

Silk4J 型には、プリミティブ型 (boolean、int、string など)、リスト、およびその他の型 (Point や Rect など) が含まれます。
- UI コントロール

  UI コントロールは、TestObject として渡したり、返したりできます。

戻り値

プロパティや戻り値を持つメソッドの場合は、次の値が返されます。

- すべての組み込み Silk4J 型の場合は正しい値。 これらの型は、「サポートされているパラメータ型」のセクションに記載されています。
- 戻り値を持たないすべてのメソッドの場合は、null が返されます。

Silverlight アプリケーションのサポート

Microsoft Silverlight (Silverlight) は、Adobe Flash と同等の機能と目的を持つリッチ インターネット アプリケーションを記述/実行するためのアプリケーション フレームワークです。 Silverlight のランタイム環境は、プラグイン形式で提供され、多くの Web ブラウザで利用可能です。

Silk4J は、 Silverlight アプリケーションのテストを組み込みでサポートします。 Silk4J は、ブラウザ上およびブラウザ外で実行する Silverlight アプリケーションをサポートしており、 Silverlight 3（Silverlight Runtime 4）または Silverlight 4（Silverlight Runtime 4）のコントロールを記録/再生できます。

Silverlight をベースにした次のアプリケーションがサポートされます。

- Windows Internet Explorer で実行する Silverlight アプリケーション。
- Mozilla Firefox 4.0 以降で実行する Silverlight アプリケーション。
- ブラウザ外実行の Silverlight アプリケーション。

サポートするバージョン、既知の問題、および回避策についての最新の情報は、 リリース ノートを参照してください。

サポートするコントロール

Silk4J は、 Silverlight コントロールの記録と再生をサポートしています。

Silverlight テストで使用可能なコントロールの完全な一覧については、「 *Silverlight* クラス リファレンス」を参照してください。

前提条件

Silverlight アプリケーションのテストを Microsoft Windows XP 上でサポートするには、サービス パック 3 をインストールし、Windows 7 で提供される MSUIA (Microsoft User Interface Automation) の Windows XP 用の更新プログラムを適用する必要があります。 更新プログラムは、 *http://www.microsoft.com/download/en/details.aspx?id=13821* からダウンロードできます。

注: Silverlight サポートには、MSUIA のインストールが必須です。Windows OS 上で Silverlight サポートが機能しない場合は、利用中のオペレーティング システムに一致した MSUIA の更新プログラムを *http://support.microsoft.com/kb/971513* からダウンロードしてインストールしてください。

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"]

Silverlight メソッドの動的呼び出し

動的呼び出し機能を使用して Silk4J が対象としないコントロールのメソッドを呼び出したり、プロパティを取得/設定することができます。 この機能は、カスタム コントロールを使用したり、カスタマイズせずに Silk4J がサポートするコントロールを使用する場合に有効です。

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。

動的プロパティーを取得する場合には getProperty メソッドを、動的プロパティーを設定する場合には、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。

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。

サポートされているパラメータ型

次のパラメータ型がサポートされます。

- すべての組み込み Silk4J 型。

  Silk4J 型には、プリミティブ型 (boolean、int、string など)、リスト、およびその他の型 (Point、Rect など) が含まれます。
- 列挙型。

  列挙パラメータは文字列として渡す必要があります。 文字列は、列挙値の名前と一致しなければなりません。 たとえば、メソッドが .NET 列挙型 System.Windows.Visiblity のパラメーターを必要とする場合には、Visible、Hidden、Collapsed の文字列値を使用できます。
- .NET 構造体とオブジェクト。

  .NET 構造体とオブジェクト パラメータはリストとして渡します。 リスト内の要素は、テスト アプリケーションの .NET オブジェクトで定義されているコンストラクターの 1 つと一致しなければなりません。 たとえば、メソッドが .NET 型 System.Windows.Vector のパラメーターを必要とする場合、2 つの整数値を持つリストを渡すことができます。 これが機能するのは、 System.Windows.Vector 型が 2 つの整数値を引数に取るコンストラクターを持つためです。
- その他のコントロール。

  コントロール パラメータは TestObject として渡すことができます。

サポートされているメソッドおよびプロパティ

次のメソッドとプロパティを呼び出すことができます。

- MSDN が定義する AutomationElement クラスのすべてのパブリック メソッドとプロパティ。詳細については、 *http://msdn.microsoft.com/ja-jp/library/system.windows.automation.automationelement.aspx* を参照してください。
- MSUIA が公開するすべてのメソッドとプロパティ。利用可能なメソッドとプロパティは「パターン」で分類されます。パターンとは、MSUIA 固有の用語です。すべてのコントロールは、いくつかのパターンを実装します。一般的なパターンについての概要およびすべての利用可能なパターンについては、*http://msdn.microsoft.com/ja-jp/library/ms752362.aspx* を参照してください。カスタム コントロールの開発者は、MSUIA パターン セットを実装することによって、カスタム コントロールのテスト サポートを提供できます。

戻り値

プロパティや戻り値を持つメソッドの場合は、次の値が返されます。

- すべての組み込み Silk4J 型の場合は正しい値。
- 戻り値を持たないすべてのメソッドの場合は、NULL が返されます。
- すべてのその他の型の場合は文字列。

  この文字列表現を取得するには、テスト対象アプリケーションの返された .NET オブジェクトに対して ToString メソッドを呼び出します。

例

Silverlight の TabItem。これは TabControl の項目です。

```
tabItem.Invoke("SelectionItemPattern.Select")
mySilverligtObject.GetProperty("IsPassword")
```

Silverlight アプリケーションのテスト時のトラブルシューティング

Silk4J で Silverlight アプリケーションの内部を確認できず、記録時に緑色の四角形が描画されない。

次の理由により、Silk4J は Silverlight アプリケーションの内部を確認できなくなっています。

原因	解決策
使用している Mozilla Firefox のバージョンが 4.0 以前です。	Mozilla Firefox 4.0 以降を使用してください。
使用している Silverlight のバージョンが 3 以前です。	Silverlight 3（Silverlight Runtime 4）または Silverlight 4（Silverlight Runtime 4）を使用してください。
使用している Silverlight アプリケーションがウィンドウレス モードで実行されています。	Silk4J は、ウィンドウレス モードで実行される Silverlight アプリケーションをサポートしません。 このようなアプリケーションをテストするには、Silverlight アプリケーションが実行されている Web サイトを変更する必要があります。 したがって、Silverlight アプリケーションがホストされている HTML または ASPX ファイルのオブジェクト タグの windowless パラメータを false に設定する必要があります。 以下のコードは、windowless パラメータを false に設定する例を示します。 `<object ...> <param name="windowless" value="false"/> ... </object>`

Windows Forms アプリケーション

Silk4J は、スタンドアロンの Windows Forms アプリケーションのテストを組み込みでサポートしています。 Silk4J は、.NET Framework 2.0 以降に組み込まれたコントロールを再生することができます。

サンプル アプリケーション、プロジェクト、スクリプト

Silk4J には、サンプルのアプリケーション、プロジェクト、およびスクリプトがあります。 *http://techpubs.borland.com/silk_gauntlet/SilkTest/* から、サンプル アプリケーションをダウンロードします。 ダウンロードしたサンプル アプリケーションには、サンプル Silk4J プロジェクトとスクリプトも含まれています。

サンプル アプリケーションをインストールしたあと、**スタート** > **すべてのプログラム** > **Silk** > **SilkTest** > **Sample Applications** > **Microsoft .NET** > **Windows Forms Sample Application** を選択してください。

Documents and Settings¥All Users¥Shared Documents¥SilkTest¥samples¥Silk4J¥Windows Forms から、Windows Forms サンプル プロジェクトをインポートします。 製品をより理解するために、サンプル スクリプトを実行してみてください。

注: Windows Vista および Windows 7.0 の場合、ファイルの場所は、Documents and Settings¥All Users¥Shared Documents¥SilkTest¥samples¥ ではなく、Users¥Public¥Documents¥SilkTest¥samples¥Silk4J¥ となります。

注: サンプル スクリプトは、テスト対象アプリケーションの場所へのパスを基本状態の中に保持しています。 SilkTest がデフォルトの場所にインストールされていない場合には、基本状態を編集して、正しい場所を参照するようパスを調整してください。

サポートするバージョン、既知の問題、および回避策についての最新の情報は、リリース ノートを確認してください。

サポートするコントロール

Windows Forms のテストで利用可能なコントロールの完全な一覧については、「*API* リファレンス」の以下のパッケージ内でサポートされる Windows Forms クラスの一覧を参照してください。

- com.borland.silktest.jtf.windowsforms：Windows Forms 固有のクラスを含みます。
- com.borland.silktest.jtf.win32：Windows Forms テクノロジ ドメインが基にしている Windows API 固有のクラスを含みます。
- com.borland.silktest.jtf：SilkTest Classic 4Test の winclass.inc のような、共通のクラスを含みます。

サポートされている属性の一覧については、「サポートする属性の種類」を参照してください。

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 Forms メソッドの動的な呼び出し

動的呼び出し機能を使用して Silk4J が対象としないコントロールのメソッドを呼び出したり、プロパティを取得/設定することができます。 この機能は、カスタム コントロールを使用したり、カスタマイズせずに Silk4J がサポートするコントロールを使用する場合に有効です。

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。

動的プロパティーを取得する場合には getProperty メソッドを、動的プロパティーを設定する場合には、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。

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。

サポートされているメソッドおよびプロパティ

次のメソッドとプロパティを呼び出すことができます。

- Silk4J がサポートするコントロールのメソッドとプロパティ
- MSDN が定義するコントロールのパブリック メソッドとプロパティ
- コントロールが標準コントロールから派生したカスタム コントロールの場合、標準コントロールが呼び出すことのできるすべてのメソッドとプロパティ

サポートされているパラメータ型

次のパラメータ型がサポートされます。

- すべての組み込み Silk4J 型

  Silk4J 型には、プリミティブ型 (boolean、int、string など)、リスト、およびその他の型 (Point や Rect など) が含まれます。
- 列挙型

  列挙パラメータは文字列として渡す必要があります。 文字列は、列挙値の名前と一致しなければなりません。 たとえば、メソッドが .NET 列挙型 System.Windows.Visiblity のパラメータを必要とする場合は、値が Visible、Hidden、または Collapsed の文字列を使用できます。
- .NET 構造体とオブジェクト

  .NET 構造体とオブジェクト パラメータはリストとして渡す必要があります。 リスト内の要素は、テスト アプリケーションの .NET オブジェクトで定義されているコンストラクターの 1 つと一致しなければなりません。 たとえば、メソッドが .NET 型 System.Windows.Vector のパラメータを必要とする場合、2 つの整数値を持つリストを渡すことができます。 これが機能するのは、System.Windows.Vector 型が 2 つの整数値を引数に取るコンストラクターを持つためです。
- その他のコントロール

  コントロール パラメータは、TestObject として渡したり、返したりできます。

戻り値

プロパティや戻り値を持つメソッドの場合は、次の値が返されます。

- すべての組み込み Silk4J 型の場合は正しい値。 これらの型は、「サポートされているパラメータ型」のセクションに記載されています。
- 戻り値を持たないすべてのメソッドの場合は、null が返されます。

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

Silk4J は、Windows Presentation Foundation (WPF) アプリケーションのテストをサポートしています。 Silk4J はスタンドアロンおよびブラウザホストのアプリケーションをサポートしており、.NET バージョン 3.5 以上に埋め込まれたコントロールを再生できます。

サポートするバージョン、既知の問題、および回避策についての最新の情報は、リリース ノートを確認してください。

サンプル アプリケーション、プロジェクト、スクリプト

Silk4J には、サンプルのアプリケーション、プロジェクト、およびスクリプトがあります。 *http://techpubs.borland.com/silk_gauntlet/SilkTest/* から、サンプル アプリケーションをダウンロードします。 ダウンロードしたサンプル アプリケーションには、サンプル Silk4J プロジェクトとスクリプトも含まれています。

サンプル アプリケーションをインストールしたあと、**スタート** > **すべてのプログラム** > **Silk** > **SilkTest** > **Sample Applications** > **Microsoft .NET** を選択し、使用するサンプル アプリケーションを選択してください。

Documents and Settings¥All Users¥Shared Documents¥SilkTest¥samples¥Silk4J¥WPF から、WPF サンプル プロジェクトをインポートします。 製品をより理解するために、サンプル スクリプトを実行してみてください。

注: Windows Vista および Windows 7.0 の場合、ファイルの場所は、Documents and Settings¥All Users¥Shared Documents¥SilkTest¥samples¥ ではなく、Users¥Public¥Documents¥SilkTest¥samples¥Silk4J¥ となります。

注: サンプル スクリプトは、テスト対象アプリケーションの場所へのパスを基本状態の中に保持しています。 SilkTest がデフォルトの場所にインストールされていない場合には、基本状態を編集して、正しい場所を参照するようパスを調整してください。

サポートするコントロール

Silk4J は、Windows Presentation Foundation（WPF）コントロールの記録と再生をサポートしています。 Silk4J 2009 では、WPF の再生がサポートされていました。 ただし、Silk4J 2010 のリリースでは、MSUIA で始まる以前の WPF コントロールは廃止されたため、新しい WPF テクノロジ ドメインを使用してください。 新しいテスト クラスを記録するときは、Silk4J は、新しい WPF テクノロジ ドメインを自動的に使用します。

MSUIA テクノロジ ドメインを使用して Silk4J 2009 で記録したテストは、引き続き機能します。 ただし、以前の MSUIA クラスを使用するテストで定数 TechDomain.WPF（Desktop.attach や Desktop.executeBaseState メソッドのために）を手動で追加した場合は、テストを成功させるために値を TechDomain.MSUIA に変更する必要があります。

MSUIA で使用するすべての Silk4J クラスは deprecated（廃止）とマークされています。 Eclipse では、廃止 API が使用されている場合、取り消し線表示されます。

Silk4J は、WPF アプリケーションを自動化するために、MSUIA (Microsoft UI Automation) を使用します。 WPF のテストで利用可能なコントロールの完全な一覧については、「*API* リファレンス」の com.borland.silktest.jtf.wpf パッケージ内でサポートされる WPF クラスの一覧を参照してください。 廃止されたコントロールの完全な一覧については、「*API* リファレンス」の com.borland.silktest.jtf.msuia パッケージ内で廃止された WPF クラスの一覧を参照してください。

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']"

属性の種類	説明	例
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>
```

WPFItemsControl クラスから派生したクラス

Silk4J は、2 つの方法を使用して WPFItemsControl から派生したクラス (WPFListBox、WPFTreeView、WPFMenu など) を操作することができます。

- コントロールでの作業

  ほとんどのコントロールには、標準的なユースケースのためのメソッドやプロパティがあります。 項目は、テキストや索引によって識別されます。
- WPFListBoxItem、WPFTreeViewItem、WPFMenuItem などの個々の項目での作業

  高度なユースケースの場合、個々の項目を使用します。 たとえば、リスト ボックスの特定の項目のコンテキスト メニューを開いたり、項目に相対的な場所をクリックしたりする場合に個々の項目を使用します。

カスタム WPF コントロール

一般的に、Silk4J では、すべての標準 WPF コントロールの記録と再生がサポートされています。

Silk4J は、カスタム コントロールが実装された方法を基にしてカスタム コントロールを処理します。 次の方法を使用してカスタム コントロールを実装することができます。

- UserControl から派生したクラスを定義する

  複合コントロールを作成する典型的な方法です。 Silk4J は、これらのユーザー コントロールを WPFUserControl として認識し、含まれるコントロールを完全にサポートしています。
- ListBox などの標準 WPF コントロールから派生したクラスを定義する

  Silk4J は、これらのコントロールを派生元の標準 WPF コントロールのインスタンスとして扱います。 ユーザー コントロールの振る舞いがその基底クラスの実装と大きく異なる場合には、子の記録、再生、解決は機能しない可能性があります。
- テンプレートを使用して視覚デザインを変更した標準コントロールを使用する

  低レベルの再生が機能しない可能性があります。 その場合には、「高レベル」再生モードに切り替えます。 再生モードを変更するには、**スクリプト オプション** ダイアログ ボックスを使用して、**OPT_REPLAY_MODE** オプションを変更します。

Silk4J は、一般的に機能テストに無関係なコントロールは除外します。 たとえば、レイアウトを目的として使用されるコントロールは含まれません。 しかし、カスタム コントロールが除外されたクラスから派生している場合、除外されたコントロールを記録/再生の対象とするためには、関連する WPF クラスの名前を指定します。

WPF メソッドの動的な呼び出し

動的呼び出し機能を使用して Silk4J が対象としないコントロールのメソッドを呼び出したり、プロパティを取得/設定することができます。 この機能は、カスタム コントロールを使用したり、カスタマイズせずに Silk4J がサポートするコントロールを使用する場合に有効です。

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。

動的プロパティーを取得する場合には getProperty メソッドを、動的プロパティーを設定する場合には、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。

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。

サポートされているメソッドおよびプロパティ

次のメソッドとプロパティを呼び出すことができます。

- Silk4J がサポートするコントロールのメソッドとプロパティ
- MSDN が定義するコントロールのパブリック メソッドとプロパティ
- コントロールが標準コントロールから派生したカスタム コントロールの場合、標準コントロールが呼び出すことのできるすべてのメソッドとプロパティ

サポートされているパラメータ型

次のパラメータ型がサポートされます。

- すべての組み込み Silk4J 型

  Silk4J 型には、プリミティブ型 (boolean、int、string など)、リスト、およびその他の型 (Point や Rect など) が含まれます。
- 列挙型

  列挙パラメータは文字列として渡す必要があります。 文字列は、列挙値の名前と一致しなければなりません。 たとえば、メソッドが .NET 列挙型 System.Windows.Visiblity のパラメータを必要とする場合は、値が Visible、Hidden、または Collapsed の文字列を使用できます。
- .NET 構造体とオブジェクト

  .NET 構造体とオブジェクト パラメータはリストとして渡す必要があります。 リスト内の要素は、テスト アプリケーションの .NET オブジェクトで定義されているコンストラクターの 1 つと一致しなければなりません。 たとえば、メソッドが .NET 型 System.Windows.Vector のパラメータを必要とする場合、2 つの整数値を持つリストを渡すことができます。 これが機能するのは、System.Windows.Vector 型が 2 つの整数値を引数に取るコンストラクターを持つためです。
- WPF コントロール

  WPF コントロール パラメータは TestObject として渡すことができます。

戻り値

プロパティや戻り値を持つメソッドの場合は、次の値が返されます。

- すべての組み込み Silk4J 型の場合は正しい値。 これらの型は、「サポートされているパラメータ型」のセクションに記載されています。
- 戻り値を持たないすべてのメソッドの場合は、null が返されます。

- すべてのその他の型の場合は文字列

返された .NET オブジェクトに対して ToString を呼び出せば、文字列表現を取得できます。
たとえば、アプリケーション開発者が次のメソッドとプロパティを持つ Calculator カスタム コントロールを作成したとします。

```
public void Reset()
public int Add(int number1, int number2)
public System.Windows.Vector StrechVector(System.Windows.Vector vector, double
factor)
public String Description { get;}
```

テスト担当者は、テスト内からメソッドを直接呼び出すことができます。 例：

```
customControl.invoke("Reset");
int sum = customControl.invoke("Add", 1, 2);
// the vector can be passed as list of integer
List<Integer> vector = new ArrayList<Integer>();
vector.add(3);
vector.add(4);
// returns "6;8" because this is the string representation of the .NET object
String strechedVector = customControl.invoke("StrechVector", vector, 2.0);
String description = customControl.getProperty("Description");
```

廃止された WPF テクノロジ ドメインを使用するために定数 TechDomain.WPF を更新する

MSUIA テクノロジ ドメインを使用して Silk4J 2009 で記録したテストは、引き続き機能します。 ただし、以前の MSUIA クラスを使用するテストで定数 TechDomain.WPF（Desktop.attach や Desktop.executeBaseState メソッドのために）を手動で追加した場合は、テストを成功させるために値を TechDomain.MSUIA に変更する必要があります。

1. Desktop.attach または Desktop.executeBaseState メソッドの定数 TechDomain.WPF に移動します。
2. 廃止された WPF コントロールを使用し続けるには、この定数を TechDomain.MSUIA に変更します。たとえば、次のようになります。変更前:

   ```
   desktop.attach("C:/myWpfApplication.exe",
   TechDomain.WPF);
   ```

   変更後

   ```
   desktop.attach("C:/myWpfApplication.exe",
   TechDomain.MSUIA);
   ```

   変更前:

   ```
   desktop.executeBaseState(new
   BaseState("myWpfApplication.exe", "//MsuiaWindow[@caption='my main
   window']", TechDomain.WPF));
   ```

   変更後

   ```
   desktop.executeBaseState(new
   BaseState("myWpfApplication.exe", "//MsuiaWindow[@caption='my main
   window']", TechDomain.MSUIA));
   ```

Web アプリケーション

Silk4J は、Web アプリケーションのテストをサポートしています。 ユーザーは、Internet Explorer、Firefox、または組み込みブラウザのコントロールを使用したアプリケーションをテストすることができま

す。 たとえば、Internet Explorer や、Java SWT アプリケーション内に組み込まれているブラウザ内で実行するアプリケーションをテストすることができます。

サンプル アプリケーション、プロジェクト、スクリプト

Silk4J には、サンプルのアプリケーション、プロジェクト、およびスクリプトがあります。 *http://techpubs.borland.com/silk_gauntlet/SilkTest/* から、サンプル アプリケーションをダウンロードします。 ダウンロードしたサンプル アプリケーションには、サンプル Silk4J プロジェクトとスクリプトも含まれています。

サンプル Web アプリケーションには、次の URL を使ってアクセスできます：

- *http://demo.borland.com/InsuranceWebExtJS/*
- *http://demo.borland.com/gmopost/*

Silk4J には、組み込みのブラウザを含んだサンプル Java SWT テスト アプリケーションがあります。 サンプル アプリケーションは、**スタート** > **すべてのプログラム** > **Silk** > **SilkTest** > **Sample Applications** > **Java SWT** > **SWT Test Application <バージョン>** にあります。 環境に適したサンプル アプリケーションのバージョンを選択してください。 **Control** > **Standard Ctrl Sample** を選択し、**Browser** タブをクリックします。

xBrowser サンプル プロジェクトを Documents and Settings¥All Users¥Shared Documents ¥SilkTest¥samples¥Silk4J¥xBrowser からインポートします。 製品をより理解するために、サンプル スクリプトを実行してみてください。 サンプル スクリプトを見れば、Silk4J ネイティブ API をカプセル化するさまざまな方法がわかり、テスト スクリプトの保守性や可読性を高めるために役立つでしょう。

注: Windows Vista および Windows 7.0 の場合、ファイルの場所は、Documents and Settings ¥All Users¥Shared Documents¥SilkTest¥samples¥ ではなく、Users¥Public¥Documents ¥SilkTest¥samples¥Silk4J¥ となります。

注: サンプル スクリプトは、テスト対象アプリケーションの場所へのパスを基本状態の中に保持しています。 SilkTest がデフォルトの場所にインストールされていない場合には、基本状態を編集して、正しい場所を参照するようパスを調整してください。

サポートするバージョン、既知の問題、および回避策についての最新の情報は、リリース ノートを確認してください。

クイック スタート チュートリアル

このクイック スタート チュートリアルでは、Silk4J を使用し、Web アプリケーションのテストが行えるよう、導入手順をステップ by ステップで提供します。

サポートするコントロール

Web アプリケーションのテストで利用可能なコントロールの完全な一覧については、「*API* リファレンス」の com.borland.silktest.jtf.xbrowser パッケージ内でサポートされる xBrowser クラスの一覧を参照してください。

サポートされている属性の一覧については、「サポートする属性の種類」を参照してください。

xBrowser のページ同期

同期は、すべてのメソッド呼び出しの前後に実行されます。 メソッド呼び出しは、同期条件が満たされるまで開始せず、終了もしません。

注: プロパティのアクセスは同期されません。

同期モード

Silk4J には、HTML および AJAX 用の同期モードがあります。

HTML モードを使用すると、すべての HTML ドキュメントが対話的な状態になることが保証されます。 このモードでは、単純な Web ページをテストすることができます。 Java Script が含まれるより複雑なシナリオが使用される場合は、以下の同期関数を使用して、手動でスクリプトを記述することが必要になることがあります。

- WaitForObject
- WaitForProperty
- WaitForDisappearance
- WaitForChildDisappearance

AJAX モードでは、ブラウザがアイドル状態に類似した状態になるまで待機します。このことは、AJAX アプリケーションまたは AJAX コンポーネントを含むページに対して特に効果的です。 AJAX モードを使用すると、同期関数を手動で記述する必要がなくなるため、スクリプト（オブジェクトの表示または非表示を待機したり、特定のプロパティ値を待機するなど）の作成処理が大幅に簡略化されます。 また、この自動同期は、スクリプトを手動で適用しないで記録と再生を正常に行うための基礎となります。

トラブルシューティング

AJAX の非同期の特性のため、ブラウザが完全にアイドル状態になることはありません。 このため、Silk4J でメソッド呼び出しの終了が認識されず、特定のタイムアウト時間が経過したあとで、タイムアウト エラーが発生することがまれにあります。 この場合は、少なくとも、問題が発生する呼び出しに対して、同期モードを HTML に設定する必要があります。

使用するページ同期メソッドにかかわらず、Flash オブジェクトがサーバーからデータを取得し、計算を実行してデータをレンダリングするテストでは、手動でテストに同期メソッドを追加する必要があります。 メソッドを追加しないと、Silk4J は、Flash オブジェクトが計算を完了するまで待機しません。 たとえば、Thread.sleep(millisecs) を使用します。

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。

監視ツールを使用して、同期の問題により再生エラーが発生するかどうかを判断します。 たとえば、FindBugs（*http://findbugs.sourceforge.net/*）を使用して、AJAX 呼び出しが再生に影響を及ぼしているかどうかを判断できます。 次に、問題が発生するサービスを **同期除外リスト** に追加します。

注: URL を除外すると、指定した URL を対象とする各呼び出しに対して同期が無効になります。 その URL に対して必要な同期は、手動で呼び出す必要があります。 たとえば、WaitForObject をテストに手動で追加する必要がある場合があります。 手動で数多くの呼び出しを追加することを避けるために、可能なかぎり、最上位の URL ではなく、具体的に対象を絞って URL を除外します。

ページ同期設定の構成

スクリプト オプション ダイアログ ボックスでは、各テストのページ同期設定を個別に構成したり、すべてのテストに適用するグローバル オプションを設定したりできます。

URL を除外フィルタに追加するには、**スクリプト オプション** ダイアログ ボックスの **同期除外リスト** で URL を指定します。

ビジュアル テストの個別の設定を構成するには、テストを記録し、次に、グローバル再生値を上書きするステップを挿入します。 たとえば、タイム サービスを除外するには、以下のように入力します。

```
desktop.setOption(CommonOptions.OPT_XBROWSER_SYNC_EXCLUDE_URLS,
    Arrays.asList("timeService"));
```

xBrowser における API 再生とネイティブ再生の比較

Silk4J では、Web アプリケーション用に API 再生とネイティブ再生がサポートされています。 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、API 再生を使用することをお勧めします。

ネイティブ再生には以下のような利点があります。

- ネイティブ再生では、マウス ポインタを要素上に移動し、対応する要素を押すことによって、エージェントはユーザー入力をエミュレートします。 この結果、再生はほとんどのアプリケーションで変更なしで動作します。
- ネイティブ再生では、Flash や Java アプレットなどのプラグイン、および AJAX を使用するアプリケーションをサポートしていますが、高レベルの API 記録はサポートしていません。

API 再生には以下のような利点があります。

- API 再生では、Web ページが onmouseover や onclick などの DOM イベントによって直接実行されます。
- API 再生を使用するスクリプトでは、ブラウザをフォアグラウンドで実行する必要はありません。
- API 再生を使用するスクリプトでは、要素をクリックする前に、要素が表示されるようにスクロールする必要はありません。
- 一般的に、高レベルのユーザー入力は再生中にポップアップ ウィンドウやユーザー対話の影響を受けないため、API スクリプトの信頼性は高くなります。
- API 再生は、ネイティブ再生よりも高速です。

スクリプト オプション ダイアログ ボックスを使用して、記録する関数の種類を構成したり、ユーザーの入力そのものを使用するかどうかを指定したりできます。

API 再生とネイティブ再生の関数の違い

DomElement クラスには、API 再生とネイティブ再生に対して異なる関数が備えられています。

以下の表に、API 再生とネイティブ再生で使用する関数を示します。

	API 再生	ネイティブ再生
マウス操作	DomClick	Click
	DomDoubleClick	DoubleClick
	DomMouseMove	MoveMouse
		PressMouse
		ReleaseMouse
キーボード操作	使用不可	TypeKeys
特殊な関数	Select	使用不可
	SetText	
	など	

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']
```

ActiveX/Visual Basic アプリケーション

Silk4J は、ActiveX/Visual Basic アプリケーションのテストをサポートしています。

サポートするバージョン、既知の問題、および回避策についての最新の情報は、リリース ノートを確認してください。

ActiveX/Visual Basic メソッドの動的な呼び出し

動的呼び出し機能を使用して Silk4J が対象としないコントロールのメソッドを呼び出したり、プロパティを取得/設定することができます。 この機能は、カスタム コントロールを使用したり、カスタマイズせずに Silk4J がサポートするコントロールを使用する場合に有効です。

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。

動的プロパティーを取得する場合には getProperty メソッドを、動的プロパティーを設定する場合には、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。

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。

サポートされているメソッドおよびプロパティ

次のメソッドとプロパティを呼び出すことができます。

- Silk4J がサポートするコントロールのメソッドとプロパティ
- コントロールが標準コントロールから派生したカスタム コントロールの場合、標準コントロールが呼び出すことのできるすべてのメソッドとプロパティ

サポートされているパラメータ型

次のパラメータ型がサポートされます。

- すべての組み込み Silk4J 型

  Silk4J 型には、プリミティブ型 (boolean、int、string など)、リスト、およびその他の型 (Point など) が含まれます。

戻り値

プロパティや戻り値を持つメソッドの場合は、次の値が返されます。

- すべての組み込み Silk4J 型の場合は正しい値。 これらの型は、「サポートされているパラメータ型」のセクションに記載されています。
- 戻り値を持たないすべてのメソッドの場合は、null が返されます。

サンプル プロジェクトとスクリプト

Silk4J が提供するサンプル プロジェクトやスクリプトを使用して、典型的なスクリプトの構成やテスト ケースの実行を見てみましょう。

http://techpubs.borland.com/silk_gauntlet/SilkTest/ から、サンプル アプリケーションをダウンロードします。 ダウンロードしたサンプル アプリケーションには、サンプル Silk4J プロジェクトとスクリプトも含まれています。 サンプルでの学習を終えたら、自身のテスト ケースを作成し始めましょう。 テスト サイクルの間、必要に応じて修正することができます。

環境	ファイルの場所
Java SWT	Documents and Settings¥All Users¥Shared Documents ¥SilkTest¥samples¥Silk4J¥SWT
Windows Forms	Documents and Settings¥All Users¥Shared Documents ¥SilkTest¥samples¥Silk4J¥Windows Forms
Windows Presentation Foundation (WPF)	Documents and Settings¥All Users¥Shared Documents ¥SilkTest¥samples¥Silk4J¥WPF
xBrowser	Documents and Settings¥All Users¥Shared Documents ¥SilkTest¥samples¥Silk4J¥xBrowser

注: Windows Vista および Windows 7.0 の場合、ファイルの場所は、Documents and Settings ¥All Users¥Shared Documents¥SilkTest¥samples¥ ではなく、Users¥Public¥Documents ¥SilkTest¥samples¥Silk4J¥ となります。

注: サンプル スクリプトは、テスト対象アプリケーションの場所へのパスを基本状態の中に保持しています。 SilkTest がデフォルトの場所にインストールされていない場合には、基本状態を編集して、正しい場所を参照するようパスを調整してください。

64 ビット アプリケーションのサポート

Silk4J では、以下のテクノロジについて、64 ビット アプリケーションのテストがサポートされています。

- Windows Forms
- Windows Presentation Foundation (WPF)
- Microsoft Windows API ベース
- Java AWT/Swing
- Java SWT

サポートするバージョン、既知の問題、および回避策についての最新の情報は、リリース ノートを確認してください。

Silk4J を使用したベスト プラクティス

ベスト プラクティスとして、テスト内でよく利用するコントロールを検索するためのメソッドを分離することをお勧めします。 例：

```
public Dialog getSaveAsDialog(Desktop desktop) {
 return desktop.find("//Dialog[@caption = 'Save As']");
}
```

Find および FindAll メソッドはそれぞれ一致したオブジェクトのハンドルを返し、そのハンドルは、アプリケーション内でオブジェクトが存在する間だけ有効です。 たとえば、ダイアログへのハンドルは、ダイアログが一旦閉じられると無効になります。 ダイアログを閉じたあとに、このハンドルに対してメソッドを実行すると、InvalidObjectHandleException がスローされます。 同様に、Web ページ上の DOM オブジェクトのハンドルも、Web ページが再読み込みされると無効になります。 テスト メソッド間の実行や、その順番の独立性を保ってデザインすることは共通のプラクティスであるため、それぞれのテスト メソッドでオブジェクトの新しいハンドルを取得するようにします。 XPath クエリの重複を避けるため、getSaveAsDialog のようなヘルパ メソッドを作成します。 例：

```
@Test
public void testSaveAsDialog() {
 // ... some code to open the 'Save As' dialog (e.g by clicking a menu item) ...
 Dialog saveAsDialog = getSaveAsDialog(desktop);
 saveAsDialog.close();
 // ... some code to open the 'Save As' dialog again
 getSaveAsDialog(desktop).click(); // works as expected
 saveAsDialog.click(); //  fails because an InvalidObjectHandleException is thrown
}
```

このコードの最後の行は、存在しないオブジェクトのハンドルを使用しているため、失敗します。

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

Silk4J 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

Silk4J 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、Silk4J 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、Silk4J 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。

Silk4J 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。 関係する XPath ロケーター文字列や属性を、スクリプト内の Find や FindAll メソッドの属性にコピーできます。 Locator Spy を使用すると、XPath クエリ文字列が有効であることが保証されます。

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、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。

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	//WPFButton[@automationId='Ok']"

属性の種類	説明	例
	性を使用することを推奨します。	
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>
```

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、SilkTest 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。

テクニカル サポート

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、テクニカル サポートを受けることができます。 弊社の熟練したテクニカル サポートのスタッフが、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。

テクニカル サポートを利用するには

Borland は、戦略的パートナーである Micro Focus 社によって本製品のサポートが行われるよう契約を結びました。 サポートを受ける場合は、カスタマー ケアにアクセスしてください。

索引

記号

数字

A

D

F

I

J

L

M

O

P

R

S

T

V

W

X

い

か