HP UFT Java Add-in Extensibility

ソフトウェア・バージョン：11.50

開発者ガイド

ドキュメント・リリース日：2012 年 12 月（英語版）
ソフトウェア・リリース日：2012 年 12 月（英語版）

ご注意

保証

HP製品、またはサービスの保証は、当該製品、およびサービスに付随する明示的な保証文によってのみ規定されるものとします。ここでの記載で追加保証を意図するものは一切ありません。ここに含まれる技術的、編集上の誤り、または欠如について、HPはいかなる責任も負いません。

ここに記載する情報は、予告なしに変更されることがあります。

権利の制限

機密性のあるコンピューターソフトウェアです。これらを所有、使用、または複製するには、HPからの有効な使用許諾が必要です。商用コンピューターソフトウェア、コンピューターソフトウェアに関する文書類、および商用アイテムの技術データは、FAR12.211および12.212の規定に従い、ベンダーの標準商用ライセンスに基づいて米国政府に使用許諾が付与されます。

著作権について

商標について

Adobe®およびAcrobat®は、Adobe Systems Incorporated (アドビシステムズ社) の登録商標です。

Intel®，Pentium®およびIntel® Xeon™ は、米国およびその他の国におけるIntel Corporationまたはその子会社の商標または登録商標です。

Javaは、Oracle Corporationおよびその関連会社の登録商標です。

Microsoft®，Windows®，Windows NT®および Windows® XPは、米国におけるMicrosoft Corporationの登録商標です。

Oracle®は、カリフォルニア州レッドウッド市のOracle Corporationの米国登録商標です。

Unix®は、The Open Groupの登録商標です。

ドキュメントの更新情報

このマニュアルの表紙には、以下の識別情報が記載されています。

- ソフトウェアバージョンの番号は、ソフトウェアのバージョンを示します。
- ドキュメントリリース日は、ドキュメントが更新されるたびに変更されます。
- ソフトウェアリリース日は、このバージョンのソフトウェアのリリース期日を表します。

更新状況、およびご使用のドキュメントが最新版かどうかは、次のサイトで確認できます。

http://support.openview.hp.com/selfsolve/manuals

このサイトを利用するには、HP Passportへの登録とサインインが必要です。HP Passport IDの登録は、次のWebサイトから行なうことができます。

http://h20229.www2.hp.com/passport-registration.html (英語サイト)

または、HP Passportのサインインページの [**New users - please register**] をクリックします。

適切な製品サポートサービスをお申し込みいただいたお客様は、更新版または最新版をご入手いただけます。詳細は、HPの営業担当にお問い合わせください。

サポート

次のHPソフトウェアサポートのWebサイトを参照してください。

http://support.openview.hp.com

このサイトでは、HPのお客様窓口のほか、HPソフトウェアが提供する製品、サービス、およびサポートに関する詳細情報をご覧いただけます。

HPソフトウェアオンラインではセルフソルブ機能を提供しています。お客様のビジネスを管理するのに必要な対話型の技術サポートツールに、素早く効率的にアクセスできます。HPソフトウェアサポートのWebサイトでは、次のようなことができます。

- 関心のあるナレッジドキュメントの検索
- サポートケースの登録とエンハンスメント要求のトラッキング
- ソフトウェアパッチのダウンロード
- サポート契約の管理
- HP サポート窓口の検索
- 利用可能なサービスに関する情報の閲覧
- 他のソフトウェアカスタマーとの意見交換
- ソフトウェアトレーニングの検索と登録

一部のサポートを除き、サポートのご利用には、HP Passportユーザーとしてご登録の上、サインインしていただく必要があります。また、多くのサポートのご利用には、サポート契約が必要です。HP Passport IDを登録するには、次のWebサイトにアクセスしてください。

http://h20229.www2.hp.com/passport-registration.html (英語サイト)

アクセスレベルの詳細については、次のWebサイトをご覧ください。

http://support.openview.hp.com/access_level.jsp

目次

目次

Java Add-in Extensibility へようこそ

HP UFT Java Add-in Extensibility は，Unified Functional Testing Java Add-in の標準でサポートされていないサードパーティ製やカスタマイズされた Java コントロールを使用するテスト・アプリケーションをサポートする SDK（Software Development Kit）パッケージです。

本章の内容

UFT Java Add-in Extensibility SDK について

UFT Java Add-in Extensibility SDK では次の内容が提供されます。

- Unified Functional Testing Java Add-in を拡張する API。これにより，Java のカスタム・コントロールをサポートします。
- Eclipse Java 開発環境用のプラグイン。カスタム・ツールキット・サポート・セットの作成と編集を支援するウィザードとコマンドが提供されています。
- このヘルプ（[**スタート**] > [**すべてのプログラム**] > [**HP Software**] > [**HP Unified Functional Testing**] > [**Extensibility**] > [**Documentation**] でアクセス）。次の内容が含まれます。
  - 開発者ガイド。詳しい手順を説明するチュートリアル形式で，サンプルのカスタム・コントロールのサポートを開発します。
  - API Reference。
  - ツールキット Configuration Schema ヘルプ
  - Unified Functional Testing Test Object Schema ヘルプ
- 印刷用（PDF）バージョンの開発者ガイド（[**スタート**] > [**すべてのプログラム**] > [**HP Software**] > [**HP Unified Functional Testing**] > [**Extensibility**] > [**Documentation**] からアクセス，または **<Unified Functional Testing インストール・ディレクトリ>\help\Extensibility** フォルダに収録）。
- サンプル・アプリケーションと，アプリケーションのサポートを拡張する Java Add-in Extensibility プロジェクト。

このガイドについて

このガイドでは，UFT Java Add-in Extensibility をセットアップし，UFT GUI テスト を拡張することによってサードパーティが提供またはカスタマイズされた Java コントロールをサポートする方法について説明します。

このガイドを使用するには UFT 機能に関する知識が必要です。また，Java Add-in Extensibility ヘルプ（[**スタート**] > [**すべてのプログラム**] > [**HP Software**] > [**HP Unified Functional Testing**] > [**Extensibility**] > [**Documentation**] > [**Java Add-in Extensibility Help**]）で提供されている『API Reference』，『Toolkit Configuration Schema のヘルプ』，『UFT Test Object Schema のヘルプ』も併せて参照してください。

このドキュメントに併せて，『HP Unified Functional Testing ユーザーズ・ガイド』，『HP Unified Functional Testing アドイン・ガイド』の Java の説明，『HP Unified Functional Testing Object Model Reference』（UFT と一緒にインストールされます。UFT メイン・ウィンドウから［**ヘルプ**］＞［**HP Unified Functional Testing ヘルプ**］でアクセスできます）も参照してください。

注：本書の情報，例，画面キャプチャは，特に UFT GUI テストで作業するものに的を絞っています。ただし，このガイドの内容のほとんどはコンポーネントにも適用できます。

ビジネス・コンポーネントおよびスクリプト・コンポーネントは，HP Business Process Testing の一部で，アプリケーションのテストにキーワード駆動型の方法論が使用されます。詳細については，『HP Unified Functional Testing ユーザーズ・ガイド』を参照してください。

対象読者

このガイドは，プログラマ，QA エンジニア，システム・アナリスト，システム・デザイナ，テクニカル・マネージャを対象に，UFT GUI テストを拡張することによって Java のカスタム・コントロールをサポートする方法について説明します。

このガイドを使用するには，次の内容に関する知識が必要になります。

- UFT の主要機能
- UFT オブジェクト・モデル
- Unified Functional Testing Java Add-in
- Java プログラミング
- XML（基本的な知識）

Unified Functional Testing ヘルプ

Unified Functional Testing ヘルプから，UFT に関するドキュメントにアクセスできます。

Unified Functional Testing ヘルプには，次の方法でアクセスできます。

- UFT で［**ヘルプ**］＞［**HP Unified Functional Testing**］を選択します。
- UFT の［スタート］メニューから，［**すべてのプログラム**］＞［**HP Software**］＞［**HP Unified Functional Testing**］＞［**Documentation**］＞［**HP Unified Functional Testing ヘルプ**］を選択します。
- 選択した UFT ウィンドウおよびダイアログ・ボックスをクリックするか，F1 キーを押します。
- UFT テスト・オブジェクト，メソッド，またはプロパティの上にカーソルを置いて F1 キーを押すと，説明，構文，例が表示されます。

その他のオンライン・リソース

トラブルシューティング&ナレッジベース : 問題の自己解決が可能な技術情報を検索できる、HPソフトウェアサポートWebサイトのトラブル シューティングのページにアクセスできます。［**ヘルプ**］＞［**トラブルシューティング&ナレッジ ベース**］を選択します。このWebサイトのURLは、http://support.openview.hp.com/troubleshooting.jsp です。

HP ソフトウェアサポート : HP ソフトウェアオンラインではセルフソルブ機能を提供しています。また、ユーザディスカッションフォーラムへの書き込みや検索、サポート要求の送信、パッチや更新されたドキュメントのダウンロードなどを行なうこともできます。［**ヘルプ**］＞［**HPソフトウェア サポート**］を選択します。このWebサイトのURLは http://support.openview.hp.com/ です。

一部のサポートを除き、サポートのご利用には、HP Passportユーザーとしてご登録の上、サインインしていただく必要があります。また、多くのサポートのご利用には、サポート契約が必要です。

アクセスレベルの詳細については、次のWebサイトをご覧ください。

http://support.openview.hp.com/access_level.jsp

HP Passport IDを登録するには、次のWebサイトにアクセスしてください。

http://h20229.www2.hp.com/passport-registration.html (英語サイト)

HPソフトウェアWebサイト：HPソフトウェアWebサイトにアクセスします。このサイトでは、HPソフトウェア製品に関する最新の情報をご覧いただけます。新しいソフトウェアのリリース、セミナー、展示会、カスタマーサポートなどの情報も含まれています。［**ヘルプ**］＞［**HP ソフトウェア Web サイト**］を選択します。このWebサイトのURLは、http://support.openview.hp.com です。

HP Software は、新しい情報を提供する目的で、製品の文書を継続的に更新しています。

更新状況、およびご使用のドキュメントが最新版かどうかは、HP Software 製品マニュアル (http://support.openview.hp.com/selfsolve/manuals)で確認できます。

第 I 部

Java Add-in Extensibility を使った作業

第 1 章

UFT Java Add-in Extensibility の概要

UFT Java Add-in Extensibility へようこそ。

UFT Java Add-in Extensibility を使用すると，購入時の Unified Functional Testing Java Add-in ではサポートされていないサードパーティ製およびカスタムの Java コントロールに対する高いレベルのサポートを提供できるようになります。

本章の内容

UFT Java Add-in Extensibility について

Unified Functional Testing Java Add-in には，よく使用される各種 Java オブジェクトの組み込みサポートが用意されています。UFT Java Add-in Extensibility を使用すると，このサポートを拡張し，UFT に追加の Java コントロールを認識させることができます。

UFT は，アプリケーション内のオブジェクトを学習する際に，コントロールを特定のテスト・オブジェクト・クラスに属するものとして認識します。これによって，UFT 内でアプリケーションのオブジェクトを表すテスト・オブジェクトの認識プロパティとテスト・オブジェクト・メソッドが決定されます。

UFT では，Extensibility を使用せずに Java Add-in で標準でサポートされない Java コントロールを学習できます。ただし，UFT はサポートされていない Java コントロールを学習する際に，コントロールを汎用の Java テスト・オブジェクトとして認識します。このタイプの Java テスト・オブジェクトには，Java コントロールに固有の特性がない可能性があります。このため，このテスト・オブジェクトを使用してテスト・ステップを作成する場合，使用可能な認識プロパティとテスト・オブジェクト・メソッドでは十分でない可能性があります。

たとえば，特殊なボタンであるカスタム・コントロールが，UFT で通常の JavaObject として認識されたとします。JavaObject テスト・オブジェクトは，単純な **Click** 操作をサポートしていません。**JavaObject.Click** メソッドでは，クリックの座標が引数として要求されます。このカスタム・コントロールをクリックするテスト・ステップを作成する場合には，ボタンの場所を計算してクリックの座標を提供する必要があります。

Java Add-in Extensibility を使用して Java コントロールのサポートを作成することにより，UFT にコントロールを特定のテスト・オブジェクト・クラスに属するものとして認識させ，テスト・オブジェクトの振る舞いを指定できるようになります。また，UFT が認識する使用可能なテスト・オブジェクト・クラスのリストを拡張することもできます。これにより，カスタム Java コントロール固有の動作を完全にサポートするテストを作成することが可能です。

Java Add-in Extensibility の構成要素

次の各項では，UFT オブジェクト・サポートを構成する主な要素について説明します。これらの要素は，Java Add-in Extensibility の構成要素です。これらの要素で提供されている既存のサポートを拡張することにより，有意義で管理のしやすいテストの作成に必要なサポートを開発できます。

テスト・オブジェクト・クラス

UFT では，アプリケーションの各オブジェクトは，それぞれのテスト・オブジェクト・クラスのテスト・オブジェクトで表されます。Java Add-in は，サポートされている各クラスを特定のテスト・オブジェクト・クラスにマッピングします。UFT は，このマッピングに基いて，使用するテスト・オブジェクト・クラスを決定します。

UFT では，まだサポートされていない Java クラス（カスタム・クラス）のコントロールを学習する際に，クラスの継承階層に基づいてコントロールを表すテスト・オブジェクト・クラスを選択します。UFT はサポートされているクラスの中から最も近い先祖を検索し，このクラスにマッピングされているテスト・オブジェクト・クラスを使用します。たとえば，カスタム・クラスが **java.awt.Applet** を拡張する場合，UFT はコントロールを **JavaApplet** テスト・オブジェクトとして認識します。カスタム・クラスが **java.awt.Canvas** を拡張する場合，UFT はコントロールを **JavaObject** テスト・オブジェクトとして認識します。

UFT のキーワード・ビューやオブジェクト・リポジトリなどで，このタイプのオブジェクトを表すのに使用されるアイコンも，テスト・オブジェクト・クラスによって決定されます。

テスト・オブジェクト名

UFT は，オブジェクトを学習する際に，オブジェクトのデータを使用してテスト・オブジェクトの名前を生成します。わかりやすいテスト・オブジェクト名を使用すれば，同じクラスのテスト・オブジェクトの間の区別が可能になり，オブジェクト・リポジトリやテストでの識別が容易になります。

UFT が，サポートされていない Java クラスのコントロールを学習し，その先祖のいずれかにマッピングされているテスト・オブジェクト・クラスを使用する場合，テスト・オブジェクト名はそのテスト・オブジェクト・クラス用に定義されたルールに基づきます。多くの場合，これは必ずしもカスタム・コントロールに最適な名前ではありません。

テスト・オブジェクトの認識プロパティ

テスト・オブジェクトの認識プロパティは，Java クラスにマッピングされているテスト・オブジェクト・クラスによって決まります。また，オブジェクトを一意に認識するのに使用される認識プロパティ，チェックポイントに使用できる認識プロパティ（[チェックポイントのプロパティ] ダイアログ・ボックス），チェックポイントに対して標準設定で選択される認識プロパティも，これによって決まります。ただし，認識プロパティの実際の値は，カスタム・クラスの定義から得られます。そのため，同じテスト・オブジェクトにマッピングされている複数のカスタム・クラスが，同じ認識プロパティに対して異なる定義を持つ場合もあります。

テスト・オブジェクト・メソッド

テスト・オブジェクトのテスト・オブジェクト・メソッドは，Java クラスにマッピングされているテスト・オブジェクト・クラスによって決まります。ただし，テスト・オブジェクト・メソッドの実際の動作は，特定のカスタム・サポート・クラスの定義に左右されます。これは，同じテスト・オブジェクト・クラスにマッピングされているカスタム・クラスの間でも，同じテスト・オブジェクト・メソッドの動作が異なる可能性があることを意味します。

イベントの記録

UFT GUI テストの作成方法の 1 つに，アプリケーションの操作を記録する方法があります。記録セッションを開始すると，UFT はアプリケーション内のオブジェクトで発生するイベントをリッスンし，該当するテスト・ステップを登録します。それぞれの Java オブジェクト・クラスでは，UFT でリッスンできるイベントを定義します。発生するイベントごとに記録するテスト・ステップは，Java Add-in によって決まります。

どのような場合に Java Add-in Extensibility を使用するか

Unified Functional Testing Java Add-in を使用することにより，すべての Java コントロールに対して一定レベルのサポートを提供できます。カスタム Java コントロールのサポートを拡張する場合には，UFT の見方でサポートの範囲を分析し，変更が必要なサポート要素を決定します。

カスタム・コントロールを分析するには，オブジェクト・スパイ，キーワード・ビュー，エディタ，記録オプションを使用します。18 ページ「Java Add-in Extensibility の構成要素」を参考に，要素のチェックを行ってください。

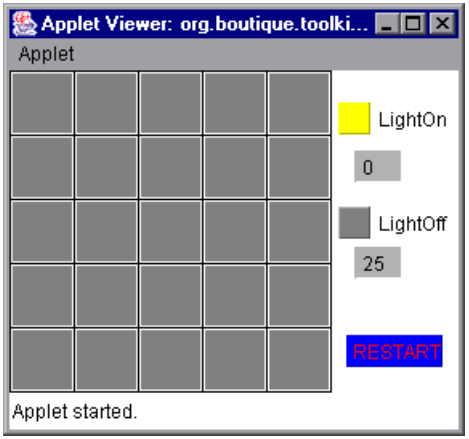
既存のオブジェクトの認識や動作が不十分な場合は，以下に示す例のように，コントロールは Java Add-in Extensibility の候補となります。

- UFT がニーズに一致しないテスト・オブジェクト・クラスを使用してコントロールを認識する場合。Java Add-in Extensibility を使用すると，カスタム・クラスを既存の別のテスト・オブジェクト・クラスまたは作成した新しいテスト・オブジェクト・クラスにマッピングすることができます。
- コントロールに対してマッピングされたテスト・オブジェクト・クラスはニーズを満たしているが，一部のテスト・オブジェクト・メソッドや認識プロパティの動作をカスタマイズする必要がある場合。Java Add-in Extensibility を使用すると，これらのプロパティやメソッドの標準実装をカスタム実装でオーバーライドできます。
- 特定の Java クラスのすべてのコントロールに対して UFT が生成するテスト・オブジェクト名が同一（一意のカウンタを除く）である場合や，コントロールに使用される名前が対応するオブジェクトを明確に示さない場合。Java Add-in Extensibility を使用すると，該当する Java クラスに対するテスト・オブジェクト名の命名方法を変更できます。
- UFT がカスタム・コントロール内の個々のサブコントロールを認識するが，メイン・コントロールを適切に認識しない場合。たとえば，メイン・カスタム・コントロールが時と分の数字を表示したエディット・ボックスを持つデジタル時計である場合，時刻の変更をエディット・ボックスに対する **Set** 操作でなく時計コントロールに対する **SetTime** 操作として認識させるようにします。Java Add-in Extensibility を使用すると，カスタム・コントロールをそれに含まれるコントロールの**ラッパー**・オブジェクトとして扱うことができます。UFT はラッパー・オブジェクトに含まれる個別のコントロールを学習しません。
- 記録セッション中に，コントロールに対して操作を実行するかイベントをトリガしたときに，UFT が全くステップを記録しないか，コントロールの動作に固有でないステップを記録する場合。あるいは，1 つの操作とみなすべきイベントに対して UFT が複数のステップを記録したり，ステップを記録すべきでないときに記録したりする場合もあります。Java Add-in Extensibility を使用すると，リッスンするイベントや特定のイベントに対して記録するテスト・ステップを変更できます。

サンプル・カスタム・コントロールに対する標準設定の UFT サポートと Extensibility オプションの分析

次の例は，Java Add-in Extensibility を使用して UFT のカスタム・コントロールのサポートを改善する方法を示します。

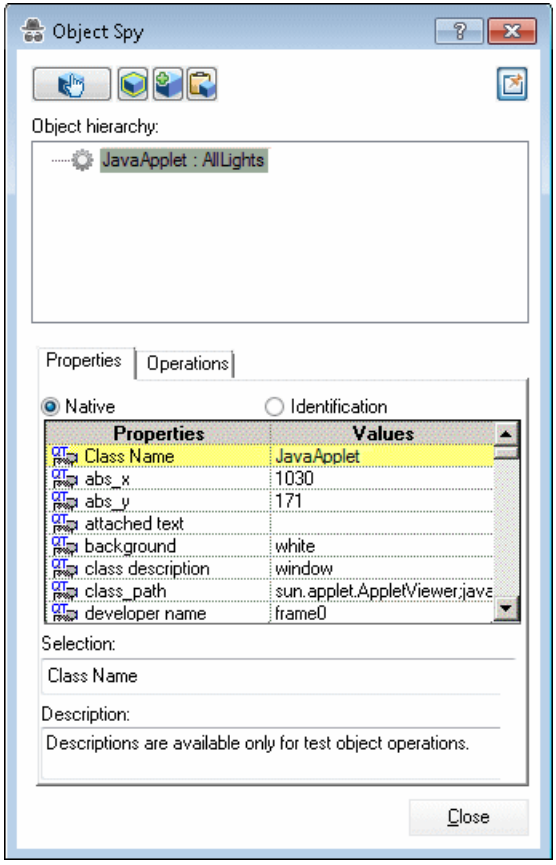
次に示す AllLights コントロールは，UFT に固有のサポートがないゲーム・アプリケーションです。



このアプリケーションは，次のように動作します。

- グリッド領域内をクリックすると，内部のルール・セットに基づいて，さまざまなライトがオン（またはオフ）になり，**LightOn** カウンタと **LightOff** カウンタが更新されます。
- ［**RESTART**］ボタンをクリックすると，すべてのライトがオフになります。**LightOn** カウンタと **LightOff** カウンタはそれに合わせて更新されます。
- その他の領域をクリックしても何も起こりません。
- このゲームの目的はすべてのライトをオンにすることです。すべてのライトがオンになると目的達成のメッセージが表示されます。

オブジェクト・スパイを使用してこのコントロールをポイントすると，UFT はこれを AllLights（カスタム・クラスの名前）という名前の汎用 JavaApplet と認識します。表示されるアイコンは標準の JavaApplet クラスのアイコンになります。



AllLights コントロールに対するサポートを実装せずに記録を実行すると，キーワード・ビューは次のようになります。

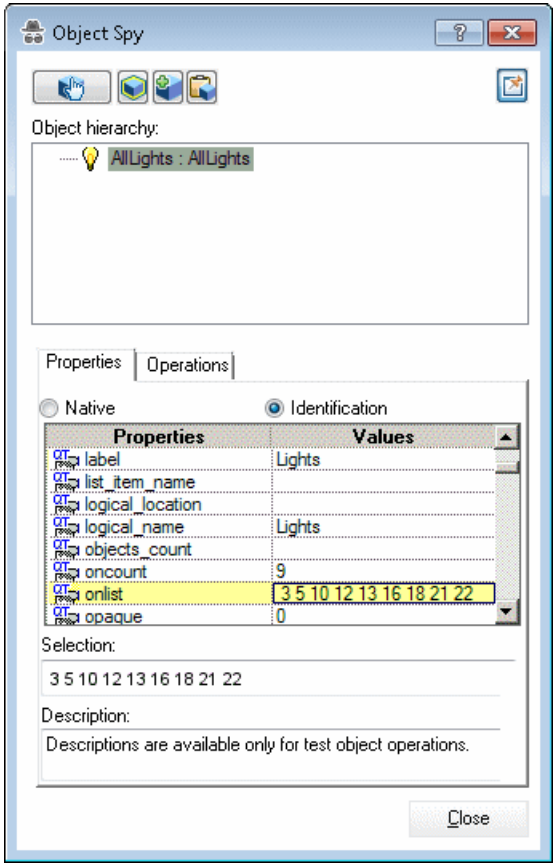
Item	Operation	Value	Documentation
Action1			
AllLights			
AllLights	Click	142,144,"LEFT"	Click the "AllLights" applet with the "LEFT" mouse button.
AllLights	Click	16,188,"LEFT"	Click the "AllLights" applet with the "LEFT" mouse button.
AllLights	Click	211,35,"LEFT"	Click the "AllLights" applet with the "LEFT" mouse button.

エディタでは，記録されたテストは次のように表示されます。

```
JavaApplet("AllLights").Click 59,60,"LEFT"
JavaApplet("AllLights").Click 76,31,"LEFT"
JavaApplet("AllLights").Click 147,20,"LEFT"
```

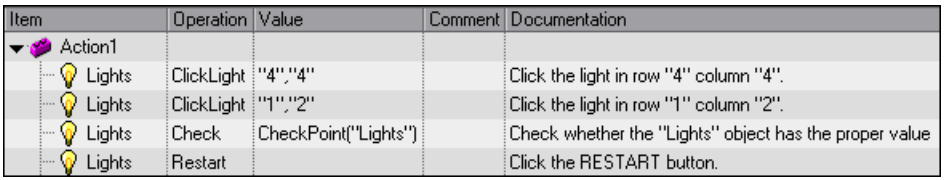
汎用の **Click** ステップのみが，低レベル記録の詳細を表す引数（x 座標と y 座標およびクリックを実行したマウス・ボタン）とともに記録されます。このステップはわかりにくく，変更作業も難しくなります。

Java Add-in Extensibility を使用して AllLights コントロールをサポートすると，よりわかりやすい結果が得られます。UFT は，コントロールを **Lights** という名前の AllLights テスト・オブジェクトとして認識し，カスタマイズされたアイコンを使用します。認識プロパティには，**oncount** や **onlist** といった関連情報が含まれます。これらは，ある時点でオンになっているライトの総数と，グリッド内の順序位置を示します。



コントロールに関するテストを作成する準備ができると，**ClickLight** メソッドと **Restart** メソッドがサポートされます。これらのメソッドは，記録することも，キーワード・ビューの［**操作**］カラムで手動で選択することもできます。また，チェックポイントを作成して，**gameover** （すべてのライトがオンになっているかどうか，つまりゲームに勝ったかどうかを示す）などの認識プロパティの値をチェックすることもできます。

キーワード・ビューでは，テストは次のように表示されます。



Item	Operation	Value	Comment	Documentation
Action1				
Lights	ClickLight	"4","4"		Click the light in row "4" column "4".
Lights	ClickLight	"1","2"		Click the light in row "1" column "2".
Lights	Check	CheckPoint("Lights")		Check whether the "Lights" object has the proper value
Lights	Restart			Click the RESTART button.

エディタでは，テストは次のように表示されます。

```
AllLights("Lights").ClickLight "4","4"
AllLights("Lights").ClickLight "1","2"
AllLights("Lights").Check CheckPoint("Lights")
AllLights("Lights").Restart
```

このテストはわかりやすく，テストの変更作業も容易になります。

第 2 章

HP UFT Java Add-in Extensibility Software Development Kit のインストール

本章では，インストールの前提条件と HP UFT Java Add-in Extensibility SDK のインストール方法について説明します。

本章の内容

HP UFT Java Add-in Extensibility SDK のインストールについて

HP UFT Java Add-in Extensibility SDK では，カスタム Java コントロールの UFT サポートを設計できます。この SDK のインストールには，次のものが含まれています。

- カスタム Java コントロールのサポートの作成に使用する API
- 次の機能を備えた Java 開発用の Eclipse IDE（統合開発環境）向けプラグイン
  - カスタム・ツールキット・サポート・セットを作成するためのウィザード

    Eclipse の Java Add-in Extensibility ウィザードでは，必要なファイル，クラス，メソッドがすべて作成されます。また，ウィザードで作成されるメソッド・スタブを利用して，メソッドを実装することもできます。

- ➤ 作成後にファイルを編集するためのコマンド

- ➤ サンプル・アプリケーションと，アプリケーションのサポートを拡張する，完成した Java Add-in Extensibility プロジェクトのセット（サンプル・アプリケーションとそれぞれのサポート・セットは，**<Java Add-in Extensibility SDK インストール・フォルダ>\samples** フォルダにインストールされます）。

UFT Java Add-in Extensibility サンプルの使用

Java Add-in Extensibility SDK の一部として提供されているサンプルを使用すると，Java Add-in Extensibility サポート・セットの設計について詳細に学習できます。

サンプルでは，SDK が 32 ビット版オペレーティング・システムの **C:\Program Files\HP\Unified Functional Testing** フォルダにインストールされていることを前提としています。

これ以外の場合，UFT でサンプルを使用する前に，サンプルのツールキット・サポート・セットで次の調整を行う必要があります。

SDK が **C:\Program Files\HP\Unified Functional Testing** にインストールされていない場合：

1. 各サンプルについて，**Configuration** フォルダ（**<Java Add-in Extensibility SDK インストール・フォルダ>\samples\<サンプル名>Support** フォルダ内）にあるツールキット設定 XML ファイルで，現在の UFT インストール・パスを使用して **SupportClassPath** プロパティを更新します。
2. 各サンプルについて，**.classpath** ファイル（**<Java Add-in Extensibility SDK インストール・フォルダ>\samples\<サンプル名>Support** フォルダ内）で，現在の UFT インストール・パスに従って関連するすべてのファイル・パスを更新します。

SDK が 64 ビット版オペレーティング・システムにインストールされている場合は，各サンプルについてさらに次の変更を行います。**.classpath** ファイルで，**mic.jar** ファイルへのパスを **C:/Program Files/HP/Unified Functional Testing/bin/java/classes/mic.jar** から **<Java Add-in Extensibility SDK インストール・フォルダ>/bin/java/classes64/mic.jar** に変更します。

インストールの前提条件

UFT Java Add-in Extensibility SDK のインストールを開始する前に, 次の条件を満たしていることを確認してください。

➤ Java Add-in Extensibility Eclipse プラグインを使用する予定がある場合は，使用するコンピュータに Java 開発用の Eclipse IDE がインストールされていることを確認します。Eclipse IDE は，http://www.eclipse.org/downloads から無料でダウンロードできます。サポート対象の Eclipse バージョンについては,『HP Unified Functional Testing 使用可能製品マトリクス』を参照してください。これは，Unified Functional Testing ヘルプにアクセスするか, Unified Functional Testing DVD のルート・フォルダに収録されています。

Eclipse IDE をインストールするときに，コンピュータ上のインストール先をメモしておいてください。Java Add-in Extensibility SDK をインストールする際に，この情報を入力する必要があります。

注：Java Add-in Extensibility Eclipse プラグインは，第 II 部「チュートリアル：Java カスタム・ツールキット・サポートの作成方法の学習」のチュートリアルを実行する場合に必要になります。また，少なくともツールキット・サポートの基本構造の設計には，このプラグインを使用することをお勧めします。

➤（オプション）UFT と Java Add-in が同じコンピュータにインストールされていることを確認します。これにより，Java Add-in Extensibility Eclipse プラグインと UFT を連携させて，カスタム・ツールキット・サポートのデバッグやテストを効率的に行うことができます。たとえば，UFT のコンピュータ上で Java Add-in Extensibility Eclipse プラグインを使用すると，ボタンをクリックするだけで簡単にツールキット・サポートを UFT にデプロイしてデバッグを行うことができます。

注: UFT Java Add-in Extensibility SDK をインストールする**前に** UFT と Java Add-in をインストールしておかないと，UFT との連携が必要な Java Add-in Extensibility Eclipse の機能は使用できなくなります。

HP UFT Java Add-in Extensibility SDK のインストール

HP UFT Java Add-in Extensibility SDK のインストールには，Unified Functional Testing のセットアップ・プログラムを使用します。

UFT Java Add-in Extensibility SDK をインストールするには，次の手順を実行します。

1 Eclipse と UFT のすべてのインスタンスを閉じます。

2 CD-ROM/DVD ドライブに Unified Functional Testing DVD を挿入します。［Unified Functional Testing のセットアップ］ウィンドウが開きます（ウィンドウが開かない場合は，DVD のルート・フォルダにある **setup.exe** をダブルクリックします）。

3 ［**アドインによる機能拡張と Web ツールキット**］をクリックします。［Unified Functional Testing Add-in Extensibility と Web 2.0 Toolkit のサポート］画面が開きます。

4 ［**HP UFT Java Add-in Extensibility SDK Setup**］をクリックします。

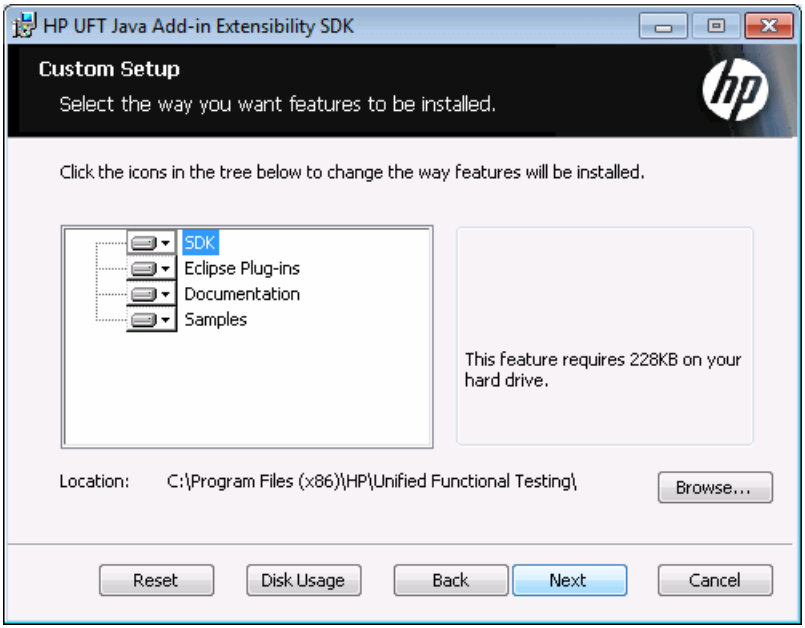
HP UFT Java Add-in Extensibility SDK Setup ウィザードの Welcome 画面が表示されます。

5 ［**Next**］をクリックします。［End-User License Agreement］画面が開きます。

注：［Modify, Repair, or Remove Installation］画面が表示される場合，SDK はコンピュータ上にすでにインストールされています。この場合，新しいバージョンをインストールする前に，既存のバージョンをアンインストールする必要があります（33 ページ「HP UFT Java Add-in Extensibility SDK のアンインストール」を参照してください）。

License Agreement（使用許諾契約）の内容を確認して，［**I accept the terms in the License Agreement**］を選択します。

6 ［**Next**］をクリックします。［Custom Setup］画面が開きます。



➤ セットアップを行うと，［Custom Setup］画面に表示されるすべての機能が自動的にインストールされます。

➤ この画面には，UFT Java Add-in Extensibility SDK のインストール先が表示されます。

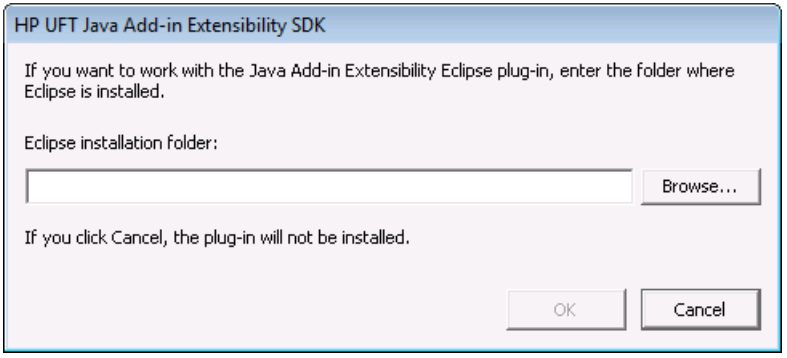
UFT がインストールされているコンピュータに UFT Java Add-in Extensibility SDK をインストールする場合，UFT のインストール・フォルダが標準で選択されます。

別のインストール先を選択するには，［**Browse**］をクリックしてフォルダを選択し，続いて［**OK**］をクリックします。

➤ ［**Disk Usage**］をクリックすると，コンピュータ上の空きディスク容量とインストールに必要な容量を示すウィンドウが表示されます。インストールに必要な容量には，UFT Java Add-in Extensibility SDK のファイルとフォルダに必要な容量（このインストール用に選択したディスクの容量）と，インストール時にのみ使用するシステム・ディスク（オペレーティング・システムがインストールされているディスク）上の容量が含まれます。

7 ［**Next**］をクリックします。［Ready to Install］画面が表示されます。

8 ［**Install**］をクリックします。UFT Java Add-in Extensibility SDK がインストールされ，コンピュータ上の Eclipse のインストール・フォルダを指定するダイアログ・ボックスが表示されます。



指定した Eclipse のインストール・フォルダに基いて，Eclipse に Java Add-in Extensibility Eclipse プラグインがインストールされます。

注：UFT Java Add-in Extensibility SDK のインストールが完了した後で，別の Eclipse インストールに Java Add-in Extensibility Eclipse プラグインをインストールすることができます。これを行うには，インストール対象の **<UFT Java Add-in Extensibility SDK インストール・フォルダ>\eclipse** フォルダを参照して，**deploysdkplugins.exe** を実行します。表示されるダイアログ・ボックスに Eclipse インストール・フォルダを入力して，［**OK**］をクリックします。

このプラグインを使用する予定がない場合は，［**Cancel**］をクリックして手順 9 に進みます。それ以外の場合は，［**Browse**］をクリックして Eclipse インストール・フォルダに移動し，ルートの **eclipse** フォルダを選択して，［**OK**］をクリックします。続いて，［**OK**］をクリックして，Eclipse インストール・フォルダを確定します。

9 最後の画面で［**Show Readme**］チェックボックスを選択して［**Finish**］をクリックすると，UFT Java Add-in Extensibility の Readme ファイルが開きます。Readme ファイルには，技術情報とトラブルシューティングに関する最新情報が記載されています。後で Readme ファイルを開くには，［**スタート**］＞［**すべてのプログラム**］＞［**HP Software**］＞［**HP Unified Functional Testing**］＞［**Extensibility**］＞［**Documentation**］＞［**Java Add-in Extensibility Readme**］を選択します。

［**Finish**］をクリックすると，セットアップ・ウィザードが終了します。

ヒント: インストール後, Eclipse に UFT のメニューやツールバーが表示されない場合は，コンピュータ上でコマンド・ライン <Eclipse インストール・フォルダ>\eclipse -clean を実行して Eclipse プラグインの設定をリフレッシュしてから，Eclipse を再度開いてください。

HP UFT Java Add-in Extensibility SDK のアンインストール

HP UFT Java Add-in Extensibility SDK のアンインストールは, ほかのプログラムと同様に，［**プログラムの追加と削除**］で行います。または，Unified Functional Testing のセットアップ・プログラムを使用することもできます。

SDK のアンインストールに関する注意事項

➤ HP UFT Java Add-in Extensibility SDK のアンインストールを行うと，すべての Eclipse インストールから Java Add-in Extensibility Eclipse プラグインが削除されます。

アンインストール後, Eclipse に UFT のメニューやツールバーが引き続き表示される場合は，コンピュータ上でコマンド・ライン <Eclipse インストール・フォルダ>\eclipse -clean を実行して Eclipse プラグインの設定をリフレッシュしてから, Eclipse を再度開いてください。

➤ セットアップ・プログラムを使用して SDK をアンインストールする場合は，最初のインストール時に使用したのと同じバージョンのセットアップ・プログラムを使用する必要があります。

➤ UFT Java Add-in Extensibility SDK をアンインストールするには，管理者権限でログインする必要があります。

HP UFT Java Add-in Extensibility SDK をアンインストールするには，次の手順で行います。

1 Eclipse と UFT のすべてのインスタンスを閉じます。

2 CD-ROM/DVD ドライブに Unified Functional Testing DVD を挿入します。［Unified Functional Testing のセットアップ］ウィンドウが開きます（ウィンドウが開かない場合は，DVD のルート・フォルダにある **setup.exe** をダブルクリックします）。

3 ［**アドインによる機能拡張と Web ツールキット**］をクリックします。［Unified Functional Testing Add-in Extensibility と Web 2.0 Toolkit のサポート］画面が開きます。

4 ［**HP UFT Java Add-in Extensibility SDK Setup**］をクリックします。HP UFT Java Add-in Extensibility SDK Setup ウィザードの Welcome 画面が表示されます。

注：以前のバージョンの SDK がインストールされている場合は，HP QuickTest Professional Java Add-in Extensibility SDK のセットアップ・ウィザードが起動します。このウィザードを使用して，次の指示に従って古いバージョンの SDK をアンインストールします。

---

5 ［**Next**］をクリックします。［Modify, Repair, or Remove Installation］画面が表示されます。

6 ウィザードの指示に従って，HP UFT Java Add-in Extensibility SDK を削除します。

トラブルシューティングと制限事項 - Java Add-in Extensibility Eclipse プラグイン

本項では，UFT Java Add-in Extensibility の使用に関するトラブルシューティングと制限事項について説明します。

➤ Eclipse 3.3 に Java Add-in Extensibility プラグインをインストールした場合, Eclipse ヘルプでソフトウェア更新オプションが利用できない可能性があります。

回避策：eclipse\features\com.mercury.qtjext.PluginFeature_1.0.0\feature.xml ファイルを ANSI 形式ではなく UTF-8 形式で保存します。

第 3 章

カスタム・ツールキットのサポートの実装

Java Add-in Extensibility を実装するには，サポートする各 Java ツールキットに対して，**カスタム・ツールキット・サポート・セット**を作成します。カスタム・ツールキット・サポート・セットは，Java クラスと XML 設定ファイルで構成されます。ユーザが作成する Java クラスは，メソッドのオーバーライドや新規メソッドの定義を行うことで，既存の Java Add-in クラスとそれらが提供するサポートを拡張します。

本章では，カスタム・ツールキットに対するサポートの作成方法を説明します。具体的には，カスタム・ツールキット・サポート・セット用に作成するファイル，これらのファイルの構造と内容，これらのファイルの格納場所について説明します。

本章の内容

カスタム・ツールキットのサポートについて

カスタム・ツールキットの UFT サポートを拡張する場合は，既存の UFT Java Add-in をベースとする API を作成して追加します。この API（カスタム・ツールキット・サポート・セット）は，Java クラスと XML 設定ファイルから構成されます。API は UFT とテスト対象の Java アプリケーションとの間のインタフェースを提供することにより，UFT でアプリケーション内の Java コントロールを認識し，これらのコントロールに対して正しく操作を実行できるようにします。

本章では，カスタム・ツールキット・サポート・セットに必要なさまざまなファイル，クラス，メソッド，定義について説明します。詳細については，『UFT Java Add-in Extensibility API Reference』(『Java Add-in Extensibility SDK ヘルプ』で利用可能)を参照してください。

カスタム・ツールキット・サポート・セットを作成する際には，事前に綿密な計画を作成する必要があります。詳細については，91 ページ「カスタム・ツールキット・サポートの計画」を参照してください。

UFT Java Add-in Extensibility SDK には，Eclipse Java 開発環境用のプラグインが用意されています。このプラグインでは，カスタム・ツールキット・サポート・セットを作成するためのウィザードが利用できます。また，このプラグインには，作成後のファイルを編集するためのコマンドも用意されています。

Java Add-in Extensibility ウィザードを使用してカスタム・ツールキット・サポートを作成すると，必要なファイル，クラス，基本的なメソッドがすべて作成されます。また，ウィザードで作成されるメソッド・スタブを利用して，追加のメソッドを実装することもできます。

実際に設計を始める前にカスタム・ツールキット・サポート・セットの設計についての理解を深めるには，第 II 部「チュートリアル : Java カスタム・ツールキット・サポートの作成方法の学習」のレッスンを実施してください。これらのレッスンでは，Eclipse で Java Add-in Extensibility ウィザードを使用して，サンプル・カスタム・コントロールのカスタム・サポートを作成します。

Java ソフトウェアの開発に Eclipse を使用していない場合でも，チュートリアルを実行する際には，Java Add-in Extensibility のために Eclipse を使用することをお勧めします。一般に，Java Add-in Extensibility ウィザードを使用してカスタム・ツールキット・サポートの基本構造を作成する方が，手動で作成するよりも簡単です。ウィザードを使用してカスタム・ツールキット・サポートの基本構造を作成してから，それぞれが選択した開発環境でツールキット・サポートの設計を続けることができます。

Eclipse と UFT Java Add-in Extensibility Eclipse プラグインの設定，およびプラグインの使用については，27 ページ「HP UFT Java Add-in Extensibility Software Development Kit のインストール」を参照してください。

Eclipse で Java Add-in Extensibility ウィザードを使用しない場合でも，本章の情報を使用して，手動でカスタム・ツールキットに対する完全サポートを拡張することができます。

Java Add-in Extensibility の用語の概要

本書では，次に示す UFT Java Add-in Extensibility 固有の用語を使用しています。

- **基本ユーザ・インタフェース・コンポーネント**：
  - AWT ツールキットの場合：**java.awt.Component**
  - SWT ツールキットの場合：**org.eclipse.swt.widgets.Widget**
- **カスタム・クラス**：UFT サポートを作成する **java.awt.Component** または **org.eclipse.swt.widgets.Widget** を拡張する Java クラス。
- **カスタム・ツールキット**：同じネイティブ・ツールキットの基本ユーザ・インタフェース・コンポーネントを拡張する一連のクラス。
- **カスタム・ツールキット・サポート**：UFT の機能を拡張して，カスタム・ツールキット内のコントロールをテスト・オブジェクトとして認識し，そのコントロールのプロパティの表示と確認を行い，コントロールに関するテストを実行できるようにします（本書では，カスタム・ツールキット・サポートを**カスタム・サポート**または**ツールキット・サポート**とも表記しています）。
- **ネイティブ・ツールキット**：ネイティブ API による描画を実装するツールキット。
  - Abstract Windows Toolkit（AWT）と Standard Widgets Toolkit（SWT）はネイティブ・ツールキットです。
  - Java Foundation Classes（JFC）は，AWT を拡張したもので，ネイティブ・ツールキットではありません。

カスタム・ツールキットのサポートの作成の準備

java.awt.Component または **org.eclipse.swt.widgets.Widget** を拡張するクラスを含むツールキットに対して UFT サポートを拡張することができます。

カスタム・ツールキットごとにカスタム・ツールキット・サポート・セットを作成する場合は，最初に，使用するカスタム・ツールキットを含むクラスのセットを決定します。Extensibility の場合，**カスタム・ツールキット**は，同じネイティブ・ツールキットの基本ユーザ・インタフェース・コンポーネントを拡張するクラスのセットです。

ただし，**java.awt.Component** を拡張するクラスと同時に，**org.eclipse.swt.widgets.Widget** を拡張するクラスを含むツールキットのサポートを作成できないわけではありません。このようなツールキットは 2 つの異なるカスタム・ツールキットとして表示されるため，それぞれのクラスのセットごとにサポートを作成する必要があります。

同様に，同じネイティブ・ツールキットの基本ユーザ・インタフェース・コンポーネントを拡張し，別々の Java アーカイブまたはクラス・フォルダとしてパッケージ化されるユーザ・インタフェース・コントロール・クラスがある場合は，これらを 1 つのカスタム・ツールキットとして扱うことができます。これは，これらのすべてのクラスに対応する 1 つのカスタム・ツールキット・サポート・セットを作成できることを意味します。

カスタム・ツールキット内では，コントロール（または，類似のコントロールのグループ）ごとに，UFT サポートを拡張します。そのためには，ツールキット内の異なる複数のカスタム・コントロール・クラスに対して**カスタム・サポート・クラス**を作成します（本書では，カスタム・サポート・クラスを**サポート・クラス**とも表記しています）。

カスタム・コントロールに対する UFT サポートを拡張する前に，コントロールに対するフルアクセスを確保し，コントロールの動作を理解しておく必要があります。アプリケーションでコントロールの実際の動作を確認でき，コントロールを実装するクラスにアクセスできることが必要です。

UFT でカスタム・コントロールをサポートするのに，カスタム・コントロールのソースを変更する必要はありませんが，ソースの内容を確認する作業は必要になります。外部からアクセスできるメンバ（フィールドおよびメソッド）やリッスンできるイベントなどを把握しておく必要があります。この情報は，サポート・クラスを設計するときに使用します。UFT とカスタム・クラスとの間のインタフェースを実装するために，サポート・クラスはカスタム・クラスのメンバを使用します。サポート・クラスは，publicと定義されているカスタム・クラスのメンバにのみアクセスできます。

また，Java アーカイブまたはクラス・フォルダのコンパイル済みのクラスへのアクセスも必要です。これは，サポート・クラスのコンパイル時にコンパイル済みのクラスをクラスパスに追加するためです。

カスタム・ツールキット・サポート・セットの作成

UFT サポートを拡張するカスタム・クラスのセットの決定が済んだら，カスタム・ツールキット・サポート・セットを作成します。

Java Add-in Extensibility のカスタム・ツールキット・サポート・セットは，次の Java クラスと XML 設定ファイルで構成されます。

- 1 つのツールキット・サポート・クラス（43 ページを参照）
- 1 つのツールキット設定ファイル（44 ページを参照）
- 1 つ以上のテスト・オブジェクト設定ファイル（このサポート・セットで新しいテスト・オブジェクト・クラスが導入されるか，既存のテスト・オブジェクト・クラスが拡張される場合）（45 ページを参照）
- （カスタム・クラスにマッピングされる）カスタム・サポート・クラス（55 ページを参照）

カスタム・ツールキット・サポート・セットの Java クラスは，**com.mercury.ftjadin.qtsupport.<カスタム・ツールキット名>** という名前のツールキットのルート・パッケージでパッケージ化されます。このパッケージ内で，カスタム・サポート・クラスは，**com.mercury.ftjadin.qtsupport.<カスタム・ツールキット名>.cs** という名前のサブパッケージに格納されます。設定ファイルは UFT のインストール・フォルダの下に格納されて，Java パッケージを参照します。詳細については，78 ページ「カスタム・ツールキット・サポートのデプロイと実行」を参照してください。

カスタム・ツールキット・サポート・セットを作成するには，次の手順を実行します。

1 カスタム・ツールキットを表す一意の名前を選択します。

カスタム・ツールキット名は，ツールキット・サポート・クラスの名前の作成とパッケージ化に使用されます。名前の先頭は英字で，英数字とアンダースコアだけが使用できます。

サポートを作成して UFT にデプロイすると，アドインやサポートされている環境のリストを表示する UFT のすべてのダイアログ・ボックスにカスタム・ツールキット名が表示されます。たとえば，UFT が開いたときには，［アドイン マネージャ］ダイアログ・ボックスに Java Add-in の子としてカスタム・ツールキット名が表示され，UFT ユーザはそのツールキットのサポートをロードするかどうかを指定できます。

一意のツールキット名を指定することで，1 つの UFT で多数のカスタム・ツールキット・サポート・セットを同時にサポートすることが可能になります。そのため，MyToolkit といった名前は推奨されません。

2 次のツールキットのルート・パッケージを作成します。
com.mercury.ftjadin.qtsupport.<カスタム・ツールキット名>

3 ツールキットのルート・パッケージ内にツールキット・サポート・クラスを作成します。作成したクラスに **<カスタム・ツールキット名 >Support.java** という名前を付けます。
このクラスの内容については，43 ページ「ツールキット・サポート・クラスについて」を参照してください。

4 ツールキット設定ファイルを作成します。作成したファイルに**<カスタム・ツールキット名>.xml** という名前を付けます。
このファイルの内容については，44 ページ「ツールキット設定ファイルについて」を参照してください。

注 : UFT に表示されるカスタム・ツールキット名（アドイン・マネージャやその他のダイアログ・ボックス）は，このファイル名から来ています。

---

5 カスタム・コントロールの動作（フィールドおよびメソッド）を考慮し，カスタム・コントロールを UFT のテスト・オブジェクト・クラスにマッピングします。詳細については，61 ページ「カスタム・コントロールのテスト・オブジェクト・クラスへのマッピング」を参照してください。

カスタム・ツールキットのコントロールにマッピングする UFT の新しいテスト・オブジェクト・クラスが必要な場合は，テスト・オブジェクト設定ファイルを作成します。作成したファイルに **<カスタム・ツールキット名>TestObjects.xml** という名前を付けます。

このファイルの内容およびこのファイルを格納する場所については，45 ページ「テスト・オブジェクト設定ファイルについて」を参照してください。

注 : 多くの場合，カスタム・ツールキット・サポート・セットには，**<カスタム・ツールキット名>TestObjects.xml** という名前のテスト・オブジェクト設定ファイルが 1 つだけあります。ただし，異なるテスト・オブジェクト・クラスの定義を別のテスト・オブジェクト設定ファイルに格納することは可能です。テスト・オブジェクト設定ファイルはすべて，『UFT Test Object Schema ヘルプ』（『Java Add-in Extensibility SDK ヘルプ』で利用可能）に従って作成します。テスト・オブジェクト設定ファイルはすべて，78 ページ「カスタム・ツールキット・サポートのデプロイと実行」で指定された，同じフォルダに配置する必要があります。

UFT の起動時に，UFT ユーザは（［アドイン マネージャ］ダイアログ・ボックスで）サポートをロードする環境またはカスタム・ツールキットを選択できます。選択すると，UFT は，サポートをロードするすべてのカスタム Java ツールキットに対して，（テスト・オブジェクト設定ファイルから）テスト・オブジェクト・クラスの定義をロードします。これにより，サポートする Java ツールキットが複数存在する場合でも同じテスト・オブジェクト・クラスの定義を使用できます。

---

6 サポート・クラスのサブパッケージ **com.mercury.ftjadin.qtsupport.<カスタム・ツールキット名>.cs** を作成します。

7 サポート・クラスのサブパッケージで，サポートが必要なクラスに対するカスタム・サポート・クラスを作成します。

ほとんどの場合，カスタム・サポート・クラスには **<カスタム・クラス名>CS** という名前を付けます。カスタム・ツールキットに異なる複数のパッケージのクラスが含まれている場合，同じ名前のカスタム・クラスが存在する可能性があります。この場合は，1 つのパッケージに格納できるようにカスタム・サポート・クラスに異なる名前を指定する必要があります。サポート・クラスの内容については，55 ページ「カスタム・サポート・クラスについて」を参照してください。

次の例は，**javaboutique** という名前のカスタム・ツールキットに対するカスタム・ツールキット・サポート・セット内の Java クラスの構造を示しています。このツールキット内では，**AllLights** と **AwtCalc** の 2 つのカスタム・クラスがサポートされています。

8 Java Add-in Extensibility ウィザードを使用してカスタム・サポートを作成する場合，ウィザードによって必要な環境変数が定義されます。このウィザードを使用しない場合は，次の項目をビルド・パス（コンパイラが使用するクラスパス）に追加する必要があります。

- **<Java Add-in Extensibility SDK インストール・フォルダ>\bin\Java\sdk\eclipse\plugins\com.mercury.java.ext.lib_1.0.0\mic.jar**
- **<Java Add-in Extensibility SDK インストール・フォルダ>\bin\Java\sdk\eclipse\plugins\com.mercury.java.ext.lib_1.0.0\jacob.jar**
- コンパイル済みカスタム・クラスの格納場所（クラス・フォルダまたは Java アーカイブの場所）。

注：

- ビルド・パスには，カスタム・クラスのすべての親クラスの場所も含める必要があります。SWT，AWT，または JFC（Swing）から直接派生したカスタム・クラスがなく，親クラスがカスタム・クラスと同じ場所に存在しない場合は，ビルド・パスにこれらの場所を手動で追加します。
- カスタム・コントロールが変更されて，サポートに影響が及ぶ可能性がある場合は，必要に応じてサポート・クラスを再コンパイルして調整する必要があります。

ツールキット・サポート・クラスについて

あるカスタム・ツールキット内のすべてのクラスが別のツールキットの基本ユーザ・インタフェース・クラス（たとえば，**java.awt.Component**）を拡張する場合，カスタム・ツールキットがツールキット（この例では，**AWT**）を拡張する，といいます。すべてのカスタム・ツールキット・サポート・セットには，カスタム・ツールキットが拡張する**ネイティブ・ツールキット**を示すツールキット・サポート・クラスが 1 つ存在します。

カスタム・ツールキット・サポート・クラスを適切なネイティブ・ツールキット・サポート・セットから拡張することで，基本的な機能（イベント処理やディスパッチなど）に必要なすべてのユーティリティ・メソッドが確実にツールキットに継承されます。

Unified Functional Testing Java Add-in では，AWT，SWT，JFC（Swing）に対応したカスタム・ツールキット・サポート・クラスが利用できます。新しい Java Add-in Extensibility カスタム・ツールキット・サポート・クラスを作成する際には，これらのいずれかのカスタム・ツールキット・サポート・クラス，または Extensibility のほかの既存のカスタム・ツールキット・サポート・セットのカスタム・ツールキット・サポート・クラスを拡張します。

ツールキット・サポート・クラスの継承階層には，カスタム・ツールキットの階層が反映されます。たとえば，JFCSupport クラスは AWTSupport クラスを拡張します。JFC を拡張するツールキットのツールキット・サポート・クラスは JFCSupport を拡張するため，AWTSupport の機能を継承します。このクラスでは，追加の実装は必要ありません。

たとえば, これは, 次のように **AWT** ネイティブ・ツールキットを拡張する, **Javaboutique** カスタム・ツールキットに対するツールキット・サポート・クラスです。

```
package com.mercury.ftjadin.qtsupport.javaboutique;
import com.mercury.ftjadin.support.awt.AwtSupport;
public class JavaboutiqueSupport extends AwtSupport {}
```

次の表に，AWT，SWT，または JFC 用のツールキット・サポート・クラスを拡張する場合に，拡張対象となるツールキット・サポート・クラスを示します。

次のツールキットのツールキット・サポート・クラスを拡張する場合	次のツールキット・サポート・クラスを拡張
AWT	com.mercury.ftjadin.support.awt.AwtSupport
JFC11（Swing）	com.mercury.ftjadin.support.jfc.JFCSupport
SWT	com.mercury.ftjadin.support.swt.SwtSupport

ツールキット設定ファイルについて

カスタム・ツールキット・サポート・セットには，**<カスタム・ツールキット名>.xml** という名前のツールキット設定ファイルが 1 つずつ存在します。この設定ファイルは，UFT のインストール・フォルダの下に格納されます。このファイルでは，UFT でカスタム・ツールキット・サポート・セットのクラスを見つけるのに必要な情報が提供されます。

ツールキット設定ファイルでは，次の内容を指定します。

- ツールキット・サポート・クラスの場所
- コンパイル済みのサポート・クラスの場所（クラス・フォルダまたは Java アーカイブの場所）

  UFT では，アプリケーションの実行時にこの場所を Java アプリケーションのクラスパスに追加して，アプリケーションが必要なサポート・クラスを認識できるようにします。

- サポート・ツールキットの説明

  UFT のアドイン・マネージャで，使用可能なアドインのリストからカスタム・ツールキットの名前を選択すると，この説明が表示されます。このツールキット・サポート・セットを配布用に開発する場合は，関連する人物または会社を指定する **Provided by** 句をこの説明に含めます。

- 各カスタム・クラスからそれぞれのカスタム・サポート・クラスへのマッピング

  1 つのカスタム・サポート・クラスを複数のカスタム・クラスにマッピングすることは可能ですが，カスタム・クラスはそれぞれ 1 つのカスタム・サポート・クラスにしかマッピングできません。

次の例は，**javaboutique** ツールキット・サポートの設定ファイルを示しています。1 つのカスタム・クラス（**AwtCalc**）がサポートされています。

```
<?xml version="1.0" encoding="UTF-8"?>
<Controls
   class="com.mercury.ftjadin.qtsupport.javaboutique.javaboutiqueSupport"
   SupportClasspath="C:\JE\workspace\javaboutiqueSupport\bin"
   description="Javaboutique toolkit support.">
   <Control Type="org.boutique.toolkit.AwtCalc">
      <CustomRecordReplay>
         <ImplementationClass>
               com.mercury.ftjadin.qtsupport.javaboutique.cs.AwtCalcCS
         </ImplementationClass>
      </CustomRecordReplay>
   </Control>
</Controls>
```

ツールキット設定ファイルは，次のファイルで検証できます。
<UFT インストール・フォルダ>\bin\java\sdk\eclipse\plugins\com.mercury.qtjext.plugin.QTJavaExt_1.0.0\ToolkitSchema.xsd

ツールキット設定ファイルの構造および構文については，『UFT Java Add-in Extensibility Toolkit Configuration Schema ヘルプ』（『Java Add-in Extensibility SDK ヘルプ』で利用可能）を参照してください。

ツールキット設定ファイルの格納場所については，78 ページ「カスタム・ツールキット・サポートのデプロイと実行」を参照してください。

テスト・オブジェクト設定ファイルについて

カスタム・コントロールを新しい（または変更された）テスト・オブジェクト・クラスにマッピングする場合は，カスタム・ツールキット・サポート・セット内に 1 つ以上のテスト・オブジェクト設定ファイルを作成する必要があります。詳細については，61 ページ「カスタム・コントロールのテスト・オブジェクト・クラスへのマッピング」を参照してください。

テスト・オブジェクトの設定 XML では，テスト・オブジェクト・クラス（サポートされるテスト・オブジェクト・メソッド，認識プロパティなど）を定義します。

既存のテスト・オブジェクト・クラスの定義をテスト・オブジェクト設定 XML で作成することが可能です。作成した定義はテスト・オブジェクト・クラスの既存の定義に追加され，このクラスのテスト・オブジェクトすべてに適用されます。したがって，この方法で既存のテスト・オブジェクト・クラスを変更することはお勧めしません。次に例を示します。

- 追加したテスト・オブジェクト・メソッドは，UFT でテスト・オブジェクト・メソッドのリストに表示されますが，このテスト・オブジェクト・メソッドをテストで使用していてもオブジェクトで実装していない場合には実行時エラーが発生します。

  テスト・オブジェクト・メソッドを既存のテスト・オブジェクト・クラスに追加する場合は，メソッド名にプレフィックスを追加できます。このプレフィックスは，メソッドの追加先となるツールキット・サポートを示します（**CustomButtonClick** や **CustomEditSet** など）。テスト設計者はプレフィックスからカスタム・メソッドを簡単に特定し，各オブジェクトでサポートされるカスタム・メソッドのみをテスト・ステップで使用できるようになるので便利です。

- 追加した認識プロパティは，このクラスのすべてのテスト・オブジェクトで使用されるプロパティのリストに表示されますが，サポート対象オブジェクトで実装されていない場合，値はありません。

テスト・オブジェクト設定 XML ファイルでは，定義するテスト・オブジェクト・クラスごとに **ClassInfo** 要素を作成します。さらに，テスト・オブジェクト・クラスを使用する環境やカスタム・ツールキットの名前（**TypeInformation** 要素の **PackageName** 属性）と，このテスト・オブジェクト・クラスが拡張する対象となる UFT アドイン（**TypeInformation** 要素の **AddinName** 属性）を定義します。

UFT の起動時にアドインがロードされないと，UFT はこの XML 内の情報をロードしません。同様に，環境またはカスタム・ツールキットの名前が［アドイン マネージャ］ダイアログ・ボックスに表示されていて，対応するチェック・ボックスが選択されていない場合，この XML の情報はロードされません。

詳細については，52 ページ「UFT でテスト・オブジェクト設定 XML をロードする方法」を参照してください。

次の項では，テスト・オブジェクト・クラスで定義できる内容について説明します。

クラス名と基本クラス

テスト・オブジェクトの名前と属性です。これには，基本クラスも含まれます。基本クラスとは，新しいテスト・オブジェクト・クラスが拡張するスト・オブジェクト・クラスです（新しいテスト・オブジェクト・クラスを定義する場合のみ）新しいテスト・オブジェクト・クラスを定義することにより，既存の Java UFT テスト・オブジェクト・クラスを直接的または間接的に拡張できます。基本クラスには UFT で提供されているクラスのほかに，Java Add-in Extensibility を使用して定義したクラスも含まれます。

標準設定では，基本クラスは JavaObject です。

テスト・オブジェクト・クラスの名前は，同時にロードする可能性のあるサポートについて，すべての環境内で一意になるように指定する必要があります。たとえば，新しいテスト・オブジェクト・クラスを定義する場合，既存の UFT アドインのテスト・オブジェクト・クラスの名前（JavaButton，JavaEdit，など）は使用しないでください。

注：

- テスト・オブジェクト・クラスは，基本クラスのテスト・オブジェクト操作（メソッドとプロパティ），汎用タイプ，標準設定操作，アイコンを継承します。ただし認識プロパティは，継承されません。
- ほかのツールキット・サポート・セットで定義されたテスト・オブジェクト・クラスを拡張するテスト・オブジェクト・クラスを作成した場合，2 つのツールキット・サポート・セットの間に依存関係が生じます。拡張元となるツールキット・サポート・セットを UFT アドイン・マネージャにロードする場合には，拡張対象となるツールキット・サポート・セットもロードする必要があります。

汎用タイプ

新しいテスト・オブジェクト・クラスを定義するとき，新しいテスト・オブジェクト・クラスの汎用タイプとして基本クラスとは異なるタイプを選択したい場合に，テスト・オブジェクト・クラスに指定する汎用タイプです（たとえば，新しいテスト・オブジェクト・クラスが JavaObject（汎用タイプは **object**）を拡張する場合に，UFT ではこのテスト・オブジェクト・クラスを **edit** として分類したいケースなど)。

汎用タイプは，オブジェクトのフィルタ処理（ステップ・ジェネレータの［ステップでオブジェクトを選択］ダイアログ・ボックスでの操作や，複数のテスト・オブジェクトをオブジェクト・リポジトリに追加する操作など）で使用します。また，キーワード・ビューの［注釈］カラムで使用する文字列を作成する際にも使用します（テスト・オブジェクト設定ファイルで別途定義されていない場合)。

テスト・オブジェクト操作

テスト・オブジェクト・クラスに対する操作のリスト。各操作に関する次の情報を含みます。

- 引数。引数のタイプ（String，Integer など），方向（In または Out），引数の指定が必須かどうか，必須でない場合は標準設定値。
- 操作の説明（オブジェクト・スパイと，キーワード・ビューおよびステップ・ジェネレータのツールヒントで表示)。
- 注釈の文字列(キーワード・ビューとステップ・ジェネレータの［**注釈**］カラムで表示)。
- 戻り値のタイプ。
- キーワード・ビューまたはエディタで開いているテスト・オブジェクト操作で F1 キーを押すか，ステップ・ジェネレータで開いている操作で［**操作ヘルプ**］ボタンをクリックすると，コンテキスト・ヘルプ・トピックが開きます。設定内容には，ヘルプ・ファイルのパスとヘルプ ID が含まれます。

標準設定操作

このクラスのオブジェクトに対してステップが生成されたときに，キーワード・ビューとステップ・ジェネレータで標準設定で選択されるテスト・オブジェクト操作。

認識プロパティ

テスト・オブジェクト・クラスの認識プロパティのリストです。また，次の情報も定義できます。

- オブジェクト記述で使用する認識プロパティ。
- **スマート認識**で使用する認識プロパティ（この情報が使用されるのは，テスト・オブジェクト・クラスでスマート認識が有効になっている場合のみです。スマート認識を有効にする操作は，UFT の［オブジェクトの認識］ダイアログ・ボックスで行います）。
- チェックポイントと出力値で使用できる認識プロパティ。
- チェックポイントに標準で選択される認識プロパティ（UFT の［チェックポイントのプロパティ］ダイアログ・ボックス）。

アイコン・ファイル

このテスト・オブジェクト・クラスに使用するアイコン・ファイルのパス（オプション。定義しない場合は，基本クラスのアイコンが使用されます）。指定可能なファイルは，**.dll**，**.exe**，**.ico** のいずれかです。

ヘルプ・ファイル

キーワード・ビューまたはエディタでテスト・オブジェクトに対して F1 キーが押されたときに表示されるコンテキスト・ヘルプ・トピック。設定ファイルでは，ヘルプ・ファイルである **.chm** のパスとヘルプ ID を定義します。

注：テスト・オブジェクト設定ファイルに対する変更は，UFT の再起動後に有効になります。

新しいテスト・オブジェクト・クラスにマッピングされるカスタム・コントロールのサポートの作成は，チュートリアルの 241 ページ「複雑なコントロールのサポート方法の学習」で練習することができます。

テスト・オブジェクト設定ファイルは，次のファイルで検証できます。
<UFT インストール・フォルダ>\bin\java\sdk\eclipse\plugins\com.mercury.qtjext.plugin.QTJavaExt_1.0.0\ClassesDefintions.xsd

テスト・オブジェクト設定ファイルの構造および構文の詳細については,『UFT Test Object Schema ヘルプ』(『Java Add-in Extensibility SDK ヘルプ』で利用可能) を参照してください。

テスト・オブジェクト設定ファイルの格納場所については，78 ページ「カスタム・ツールキット・サポートのデプロイと実行」を参照してください。

テスト・オブジェクト設定ファイルの例

次の例は，**javaboutique** カスタム・ツールキット用に **Calculator** テスト・オブジェクト・クラスの定義を指定するテスト・オブジェクト設定ファイルの一部を示しています。

```
<TypeInformation Load="true" PackageName="javaboutique"
                 AddinName="Java">
  <ClassInfo BaseClassInfoName="JavaApplet"
    DefaultOperationName="Calculate" Name="Calculator">
    <IconInfo
    IconFile="C:\Program Files\HP\Unified Functional Testing\samples\
              Javaboutique\Calculator_3D.ico"/>
    <TypeInfo>
      <Operation ExposureLevel="CommonUsed" Name="Calculate"
        PropertyType="Method">
        <Description>Builds the whole calculation process</Description>
        <Documentation><![CDATA[Perform %a1 operation with %a2 and
          %a3 numbers]]></Documentation>
        <Argument Direction="In" IsMandatory="true" Name="operator">
          <Type VariantType="Variant"/>
        </Argument>
        <Argument Direction="In" IsMandatory="true" Name="num1">
          <Type VariantType="Variant"/>
        </Argument>
        <Argument Direction="In" IsMandatory="true" Name="num2">
          <Type VariantType="Variant"/>
        </Argument>
      </Operation>
      ...
    </TypeInfo>
    <IdentificationProperties>
      <IdentificationProperty ForVerification="true"
               ForDefaultVerification="true "Name="value"/>
      <IdentificationProperty ForVerification="true" Name="objects count"/>
      <IdentificationProperty Name="width"/>
      <IdentificationProperty ForDescription="true" Name="toolkit class"/>
      ...
    </IdentificationProperties>
  </ClassInfo>
</TypeInformation>
```

この例で，**Calculator** テスト・オブジェクト・クラスは **JavaApplet** テスト・オブジェクト・クラスを拡張します。アイコン・ファイルには **Calculator_3D.ico** を使用します。また，標準設定のテスト・オブジェクト・メソッドは **Calculate** です（このメソッドには，**Variant** 型の 3 つの必須の入力パラメータ **operator**，**num1**，**num2** があります)。

TypeInformation 要素内の **PackageName** 属性は，**Calculator** テスト・オブジェクト・クラスを **javaboutique** ツールキット・サポート用に作成することを示しています。

Calculator テスト・オブジェクト・クラスに対して，次の認識プロパティが定義されています。

- **value :** チェックポイントで利用可能で，UFT の［チェックポイントのプロパティ］ダイアログ・ボックスで標準で選択されます。
- **objects count**：チェックポイントで利用可能ですが，標準では選択されません。
- **toolkit class**：テスト・オブジェクト記述で使用されますが，チェックポイントでは利用できません。

UFT でテスト・オブジェクト設定 XML をロードする方法

UFT を実行すると，すべてのテスト・オブジェクト設定ファイルが読み込まれ，さまざまなファイルのテスト・オブジェクト・クラスの情報が，1 つのテスト・オブジェクト・クラスの定義に結合されます。詳細については，53 ページ「UFT でのテスト・オブジェクト設定ファイルの結合方法について」を参照してください。

テスト・オブジェクト設定ファイル中の **IdentificationProperty** 要素には，UFT で（［オブジェクトの認識］ダイアログ・ボックスを使用して）変更される可能性がある情報を指定する属性があります。その属性とは，**AssistivePropertyValue**，**ForAssistive**，**ForBaseSmartID**，**ForDescription**，**ForOptionalSmartID**，**OptionalSmartIDPropertyValue** です。この属性に基づいて，UFT で各種用途に使用する認識プロパティのリストが決まります。

したがって標準設定では，UFT は属性値を XML ファイルから 1 度だけ読み込みます。これにより，［オブジェクトの認識］ダイアログ・ボックスでユーザが行った変更内容が上書きされることはなくなり，UFT では，このようにして，ユーザ定義のプロパティ・リストが変更されないようにしています。詳細については，85 ページ「テスト・オブジェクト設定ファイルで指定された認識プロパティ属性の変更」を参照してください。

UFT でのテスト・オブジェクト設定ファイルの結合方法について

UFT は起動のたびに，**<UFT インストール・フォルダ>\dat\Extensibility\<UFT アドイン名>** フォルダにあるテスト・オブジェクト設定ファイルをすべて読み込みます。次に，テスト・オブジェクト設定ファイルのそれぞれの優先度に応じて，各ファイルのテスト・オブジェクト・クラスの情報を 1 つのテスト・オブジェクト・クラス定義にマージします。

次に該当する場合，テスト・オブジェクト設定ファイル内の定義は無視されます。

- **TypeInformation** 要素の **Load** 属性が false に設定されている場合。
- テスト・オブジェクト設定ファイルに関連する環境は［アドイン マネージャ］ダイアログ・ボックスに表示されているが，UFT ユーザが環境をロードしない選択を行っている場合。

テスト・オブジェクト設定ファイルの優先度は，**TypeInformation** 要素の **Priority** 属性で定義します。

テスト・オブジェクト設定ファイルの優先度が既存のクラス定義よりも高い場合，既存のテスト・オブジェクト・クラスの定義（組み込みの UFT 情報を含む）はオーバーライドされます。したがって，テスト・オブジェクト設定ファイルの優先度を変更する場合には，組み込み関数がオーバーライドされる可能性がある点に注意してください。

テスト・オブジェクト・クラスの定義が複数存在する場合，競合する定義は UFT によって処理されます。次の項では，**ClassInfo**，**ListOfValues**，**Operation** の各要素がテスト・オブジェクト設定ファイル内で複数定義されている場合の処理について説明します。各テスト・オブジェクト・クラスの **IdentificationProperty** 要素はすべて 1 つのテスト・オブジェクト設定ファイルで定義する必要があります。

ClassInfo 要素

- **ClassInfo** 要素がテスト・オブジェクト設定ファイルで定義されていて，既存の定義よりも優先度が高い場合，この情報は既存の定義に追加されます。複数のファイルで定義されている **ClassInfo** 間で競合が発生した場合，優先度の高いファイルの定義で低いファイルの定義がオーバーライド（置換）されます。
- **ClassInfo** 要素がテスト・オブジェクト設定ファイルで定義されていて，優先度が既存の定義と同じかまたは低い場合，差分が既存の定義に追加されます。複数のファイルで定義されている **ClassInfo** 間で競合が発生した場合，優先度の低い定義は無視されます。

ListOfValues 要素

- 複数のファイルで定義されている **ListOfValues** 間で競合が発生した場合, 優先度の高いファイルの定義で低いファイルの定義がオーバーライド（置換）されます（マージされません）。
- **ListOfValues** の定義で既存のリストがオーバーライドされる場合, 新しいリストには, このテスト・オブジェクト設定ファイルに含まれるクラスの操作で定義されている **Enumeration** タイプの引数すべてが反映されます。
- **ListOfValues** が設定ファイルで定義されていて，優先度が既存の定義より低い場合, 優先度の低い定義は無視されます。

Operation 要素

- **Operation** 要素は，テスト・オブジェクト設定ファイルの優先度に基づいて追加，無視，オーバーライドされます。
- **Operation** 要素がテスト・オブジェクト設定ファイルで定義されていて，既存の定義よりも優先度が高い場合, クラスの既存の定義にこの操作が追加されます。複数のファイルで定義されている **Operation** 間で競合が発生した場合，優先度の高いファイルの定義で低いファイルの定義がオーバーライド（置換）されます（マージされません）。

詳細については,『UFT Test Object Schema ヘルプ』(『Java Add-in Extensibility SDK ヘルプ』で使用可能）を参照してください。

カスタム・サポート・クラスについて

カスタム・ツールキット・サポート・セットでは，サポートされるカスタム・クラスごとにカスタム・サポート・クラスが存在します。カスタム・サポート・クラスは，カスタム・クラスのメソッドと UFT の機能との間の実際のインタフェースを提供します。このようにして，UFT Java Add-in Extensibility が提供されます。

1 つのカスタム・サポート・クラスを使用して，複数のカスタム・クラスに対してサポートを提供することができます。サポート・クラスは，ツールキット設定ファイル（44 ページを参照）で，複数のカスタム・クラスにマッピングできます。その場合，このサポート・クラスは，それにマッピングされているカスタム・クラスと，それらの子孫に対してサポートを提供します。

サポート・クラスの作成では，最初に，クラスの継承階層を決定します。これには，カスタム・ツールキット内のクラスに対するサポートを作成する順序の決定と，新しいサポート・クラスが拡張する既存のサポート・クラスの決定も含まれます。詳細については，56 ページ「サポート・クラスの継承階層の決定」を参照してください。

次に，カスタム・コントロールにマッピングするテスト・オブジェクト・クラスを決定します。詳細については，61 ページ「カスタム・コントロールのテスト・オブジェクト・クラスへのマッピング」を参照してください。

上記の階層とテスト・オブジェクト・クラスが決定したら，UFT Java Add-in Extensibility のメイン・パートであるカスタム・サポート・クラスを作成できます。

各カスタム・サポート・クラスでは，それぞれがサポートするカスタム・コントロールにマッピングされるテスト・オブジェクト・クラスと，認識プロパティおよびテスト・オブジェクト・メソッドの実装方法を決定します。

カスタム・サポート・クラスは，それぞれのスーパークラスのメソッドを継承します。必要に応じて，スーパークラスの実装を使用したり，メソッドをオーバーライドしたり，新しいメソッドを追加したりすることができます。サポート・クラスでは，次の種類のメソッドを使用します。

- **認識プロパティ・サポート・メソッド**：認識プロパティをサポートするのに使用します。詳細については，62 ページ「認識プロパティのサポート」を参照してください。
- **再生メソッド**：テスト・オブジェクト・メソッドをサポートするのに使用します。詳細については，65 ページ「テスト・オブジェクト・メソッドのサポート」を参照してください。

➤ **イベント・ハンドラ・メソッド**：カスタム・コントロールでの記録をサポートするのに使用します。Extensibility のこの部分はオプションです。記録用のサポートを実装しない場合でも，カスタム・コントロールでの UFT の基本機能（オブジェクトの学習，オブジェクトに関するテストの実行，プロパティと値のチェックなど）は完全にサポートされます。
カスタム・クラスが SWT を拡張する場合，UFT の記録機能のサポートを作成することはできません。詳細については，68 ページ「記録オプションのサポート」を参照してください。

➤ **ユーティリティ・メソッド**：Extensibility を制御するのに使用します。これらのメソッドは，UFT とカスタム・アプリケーションとの間のインタフェースを制御するもので，カスタム・クラスの固有の機能はサポートしていません。用途に応じて異なるユーティリティ・メソッドを使用します。

使用可能なユーティリティ・メソッドは，サポート・クラスのサマリ（77 ページ）で確認できます。これらのメソッドについては，68 ページ「記録オプションのサポート」，71 ページ「トップレベル・オブジェクトのサポート」，71 ページ「ラッパー・コントロールのサポート」の各項で詳しく説明します。

これらのメソッドをカスタム・サポート・クラスで実装する際には，**MicAPI** で提供される各種メソッドを使用できます。詳細については，78 ページ「MicAPI のメソッドの使用」および『UFT Java Add-in Extensibility API Reference』（『Java Add-in Extensibility SDK ヘルプ』で利用可能)を参照してください。

カスタム・クラスに含まれるメソッドの種類の概要については，77 ページ「サポート・クラスのサマリ」を参照してください。

サポート・クラスの継承階層の決定

UFT サポートを作成するカスタム・ツールキットでは，次の内容を決定する必要があります。

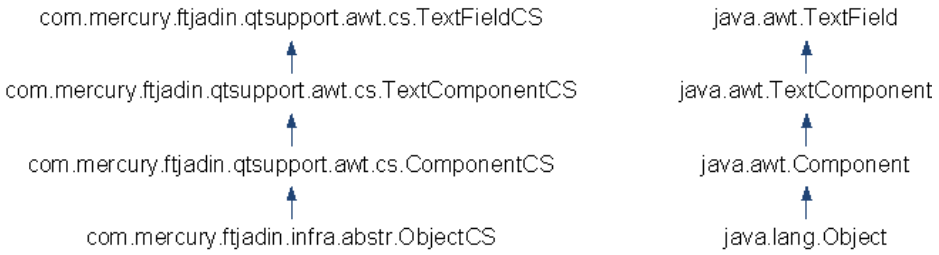
➤ 一致するサポート・クラスが必要なカスタム・クラスと，それぞれのスーパークラスのサポート・クラスでサポートできるカスタム・クラス。

➤ 新しいサポート・クラスが拡張する既存のサポート・クラス。
（これにより，サポート・クラスの作成順序も決まります。）

サポート・クラスの階層について

サポート・クラスの階層は，カスタム・クラスの階層を反映する必要があります。

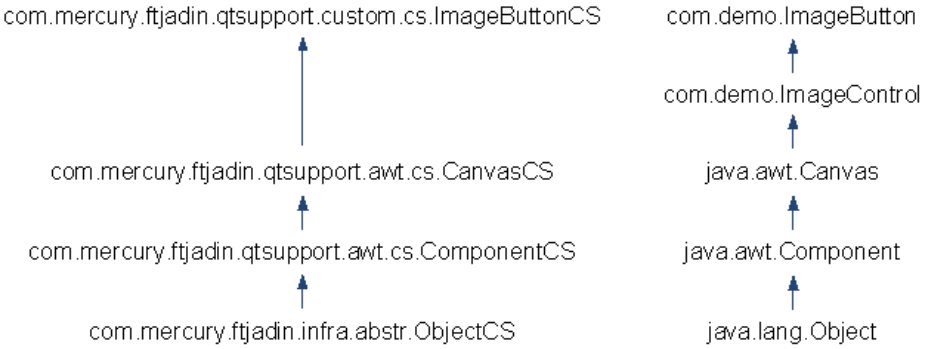
次の例は，**TextField** クラスのサポートの階層を示しています。左側の列は，TextField のサポート・クラス（**TextFieldCS**）の階層を表しています。右側の列は，AWT ツールキットでの **TextField** クラスの階層を表しています。



この例では，カスタム・クラスごとにサポート・クラスが存在していますが，これは必須でありません。

UFT では，オブジェクトを学習する際に，オブジェクトのクラス名を常に識別できます。UFT は，クラスに基いてこのクラスの継承階層を決定します。続いて，UFT は，ツールキット設定ファイルで，そのクラスにマッピングされているサポート・クラスを検索します。サポート・クラスが見つからない場合，UFT はそのサポート・クラスの 1 つ上のスーパークラスにマッピングされているサポート・クラスを検索します。それでも見つからない場合は，さらに 1 つ上のスーパークラスにマッピングされているサポート・クラスを検索し，これを一致するサポート・クラスが見つかるまで行います。サポート・クラスは，HP やその他のベンダから提供することもできます。ほかのサポート・クラスが見つからない場合，AWT オブジェクトは **ComponentCS** クラスによってサポートされ，SWT オブジェクトは **WidgetCS** クラスによってサポートされます。

次の例は，**ImageButton** クラスのサポートの階層を表しています左側の列は，ImageButton のサポート・クラス（**ImageButtonCS**）の階層を表しています。右側の列は，AWT ツールキットでの **ImageButton** クラスの階層を表しています。



ImageButton のスーパークラスである **ImageControl** には，サポート・クラスはマッピングされていません。そのため，**ImageButton** のサポート・クラスは，上位クラスである **CanvasCS** にマッピングされたサポート・クラスを拡張します。

作成するサポート・クラスの決定

サポート・クラスを必要とするカスタム・クラスを決定する際には，カスタム・クラスの機能と階層を考慮する必要があります。

カスタム・クラスのスーパークラスに対して提供されるサポートで十分にこのカスタム・クラスをサポートできる場合（つまり，カスタム・クラスにサポートが必要な固有の動作がない場合），カスタム・クラスに対するサポート・クラスを作成する必要はありません。

それ以外の場合には，スーパークラスのサポート・クラスを拡張する新しいサポート・クラスを作成し，ツールキット設定ファイル（44 ページを参照）で，カスタム・クラスにマッピングする必要があります。新しく作成するサポート・クラスでは，スーパークラスのサポート・クラスのサポートでは不十分なサポートの要素のみを実装する必要があります。

複数のカスタム・クラスが同じスーパークラスを拡張し，それらに同じサポートを必要とする認識プロパティやテスト・オブジェクト・メソッドがある場合は，このサポートを各クラスのサポート・クラスで別々に提供せずに，スーパークラスのサポート・クラスで提供します。

新しいサポート・クラスが拡張するクラスの決定

新しいサポート・クラスで拡張する既存のサポート・クラスを決定するには，カスタム・クラスの階層を調べて，そのスーパークラスにマッピングされているサポート・クラスを確認します。

Java Add-in Extensibility ウィザードを使用してカスタム・ツールキット・サポートを作成する場合は，New Custom Support Class ウィザードで，作成するサポート・クラスごとに拡張するクラスが決定されます。このウィザードでは，カスタム・クラスの階層が表示され，新しいサポート・クラスのベース（スーパークラス）となる既存のサポート・クラスが通知されます。詳細については，117 ページ「[Custom Class Selection] 画面」を参照してください。

Java Add-in Extensibility ウィザードを使用せずにサポート・クラスの継承を決定するには，次の手順を実行します。

1 カスタム・クラスの継承階層を決定します。

2 ツールキット設定ファイルでカスタム・クラスのスーパークラスにすでにマッピングされているサポート・クラスを検索します。

Unified Functional Testing Java Add-in に含まれるツールキット設定ファイルと，Extensibility のカスタム・ツールキット・サポートに含まれるツールキット設定ファイルを検索する必要があります。これらのファイルはそれぞれ，**<UFT インストール・フォルダ> bin\java\classes\builtin** と **<UFT インストール・フォルダ> bin\java\classes\extension** にあります。

3 最も近いスーパークラスにマッピングされているサポート・クラスを拡張して，カスタム・クラスのサポート・クラスを作成します。

注：最も近いサポート・クラスが Unified Functional Testing Java Add-in に含まれている場合，このサポート・クラスは **com.mercury.ftjadin.support** パッケージ内にあります。この場合，このサポート・クラスを直接拡張する代わりに，**com.mercury.ftjadin.qtsupport** パッケージで提供される同じ名前のクラスを拡張する必要があります。

次の例では，**ImageButton** カスタム・コントロールを使用して，サポート・クラスの階層を決定するプロセスについて説明します。

ImageButton クラスの階層は，次のとおりです。

```
java.lang.Object
 └java.awt.Component
    └java.awt.Canvas
       └com.demo.ImageControl
          └com.demo.ImageButton
```

ImageButton の最も近いスーパークラスである **com.demo.ImageControl** は，サポート・クラスにマッピングされていません。次のスーパークラスである **java.awt.Canvas** は，**com.mercury.ftjadin.support.awt.cs.CanvasCS** にマッピングされています。これは Unified Functional Testing Java Add-in に含まれます。そのため，**ImageButtonCS** は **qtsupport** パッケージ内の **CanvasCS** クラス（**com.mercury.ftjadin.qtsupport.awt.cs.CanvasCS**）を拡張します。**ImageButtonCS** の定義は，次のようになります。

```
package com.mercury.ftjadin.qtsupport.imagecontrols.cs;
import com.mercury.ftjadin.qtsupport.awt.cs.CanvasCS;
...
public class ImageButtonCS extends CanvasCS {};
```

注：ほかのツールキット・サポート・セットのサポート・クラスを拡張するサポート・クラスを設計した場合，2 つのツールキット・サポート・セットの間に依存関係が生じます。拡張している方のツールキット・サポート・セットを UFT のアドイン・マネージャにロードする場合は，拡張される方のツールキット・サポート・セットもロードするように選択する必要があります。

カスタム・コントロールのテスト・オブジェクト・クラスへのマッピング

UFT がカスタム・コントロールで使用する認識プロパティとテスト・オブジェクト・メソッドは，カスタム・コントロールにマッピングされるテスト・オブジェクト・クラスによって決まります。これらのプロパティとメソッドの値と動作は，カスタム・コントロールのサポート・クラスで実装されるサポート・メソッドによって決まります。

カスタム・コントロールは，カスタム・コントロールに関連するすべての認識プロパティとテスト・オブジェクト・メソッドを持つ既存のテスト・オブジェクト・クラスにマッピングできます。また，新しいテスト・オブジェクト・クラスの定義を（テスト・オブジェクト設定ファイルで）作成して，カスタム・コントロールを新しいテスト・オブジェクト・クラスにマッピングすることもできます。

新しいテスト・オブジェクト・クラスは，既存のテスト・オブジェクト・クラスをベースに，認識プロパティとテスト・オブジェクト・メソッドのセットを拡張します。すべてのテスト・オブジェクト・クラスは，JavaObject クラスを拡張します。既存のテスト・オブジェクト・クラスの定義に，必要な認識プロパティとテスト・オブジェクト・メソッドの一部だけが含まれる場合は，既存のテスト・オブジェクト・クラスを拡張する新しいテスト・オブジェクト・クラスを作成します（既存のテスト・オブジェクト・クラスに認識プロパティとテスト・オブジェクト・メソッドを追加する方法は，このクラスのすべてのテスト・オブジェクトに影響するため，お勧めできません）。

関連するテスト・オブジェクト・クラスの名前を返すように，サポート・クラスで **to_class_attr** メソッドを実装して，テスト・オブジェクト・クラスにカスタム・コントロールをマッピングします。継承された **to_class_attr** メソッドによって返されるテスト・オブジェクト・クラスがカスタム・コントロールに適している場合は，新しいサポート・クラスで **to_class_attr** メソッドをオーバーライドする必要はありません。

to_class_attr メソッドは，**Class Name** 認識プロパティに値を提供します。UFT では，オブジェクトを学習する際に，このオブジェクトで使用するサポート・クラスを検出します （57 ページ「サポート・クラスの階層について」を参照）。次に UFT は，**Class Name** 認識プロパティを使用して，このコントロールにマッピングされているテスト・オブジェクト・クラスを決定します。UFT は続いて，このテスト・オブジェクト・クラス名を使用してテスト・オブジェクトの定義を見つけます。テスト・オブジェクトの定義は，UFT の既存のテスト・オブジェクト，または新しく作成するテスト・オブジェクト設定ファイルから取得できます。

詳細については，45 ページ「テスト・オブジェクト設定ファイルについて」を参照してください。

認識プロパティのサポート

カスタム・コントロールの認識プロパティは，テスト・オブジェクト・クラスで定義されます。これは，UFT の既存のテスト・オブジェクト・クラスであるか，またはテスト・オブジェクト設定ファイルで定義するテスト・オブジェクト・クラスです。

認識プロパティのサポートをサポート・クラスで提供するには，認識プロパティごとに次の署名を使用してメソッドを実装します。

```
public String <認識プロパティ名>_attr(Object obj)
```

メソッド名には小文字のみ使用できます（テスト・オブジェクト設定ファイルのプロパティ名に大文字が使用されている場合も同様です）。**obj** 引数は，カスタム・コントロールを表すオブジェクトです。

メソッド内では，カスタム・クラスの public メンバを使用して必要なプロパティの値を返します（サポート・クラスは，publicと定義されているカスタム・クラスのメンバにのみアクセスできます）。

たとえば，次の **width_attr** メソッドは **width** 認識プロパティのサポートを実装します。

```
public String width_attr(Object obj) {
   return Integer.toString(((Component) obj).getBounds().width);
}
```

サポート・クラスが機能の似たコントロールのサポート・クラスを拡張する場合，カスタム・コントロールに変更なしで適用される認識プロパティのサポートを実装する必要はありません。たとえば，多くのコントロールには **label** プロパティがあります。**label** プロパティの実装済みのサポートで十分にカスタム・コントロールをサポートできる場合，親のメソッドをオーバーライドする必要ありません。

テスト・オブジェクト・クラスの定義に含まれない認識プロパティのサポート・メソッドを継承（または作成）する場合があります。これらの認識プロパティは，UFT の［オブジェクト スパイ］ダイアログ・ボックスや［チェックポイントのプロパティ］ダイアログ・ボックスには表示されません。これらの認識プロパティにアクセスするには，**GetROProperty** メソッドを使用します。**GetROProperty** メソッドの詳細については，『HP Unified Functional Testing Object Model Reference』を参照してください。

親のサポート・クラスでサポートされないカスタム・コントロールの認識プロパティをサポートする場合は，使用するサポート・クラスに新しいメソッドを追加します。サポートされているものと名前が同じで実装の異なる認識プロパティをサポートする場合は，親のメソッドをオーバーライドします。

予約済みの認識プロパティ

UFT では，内部で複数の認識プロパティを使用しています。これらの認識プロパティは次のように実装されている必要があります。

- UFT はすべてのテスト・オブジェクト・クラスに対して，**index**（または **class_index**），**class**（または **class_name**），**to_class**，**toolkit_class** の認識プロパティをサポートしており，これらのプロパティを使用してオブジェクトに固有の情報を取得します。ツールキット・サポート・セットでは，これらの認識プロパティの実装をオーバーライドしないでください。

- JavaTree テスト・オブジェクトと JavaList テスト・オブジェクトには，それぞれ **tree_content** および **list_content** という名前の認識プロパティが存在します。これらは，チェックポイントで使用します。UFT では，**count** 認識プロパティと **GetItem** テスト・オブジェクト・メソッドに基づいてこれらのプロパティを計算します。UFT は，**count** 認識プロパティを取得し，ツリーまたはリストの項目ごとに（0 から count-1 まで），**GetItem** テスト・オブジェクト・メソッドを呼び出します。

  count_attr または **GetItem_replayMethod** の実装をオーバーライドする場合は，UFT で期待されるタイプの情報を返す必要があります。たとえば，**count_attr** は数値を返し，**GetItem_replayMethod** は 0 から count-1 の各インデックスの項目を返す必要があります。

  カスタム・コントロールを JavaTree または JavaList テスト・オブジェクト・クラスにマッピングし，カスタム・サポート・クラスが **count_attr** メソッドと **GetItem_replayMethod** メソッドを継承しない場合は，UFT で期待される情報を返すようにこれらを実装する必要があります。

一般的な認識プロパティ・サポート・メソッド

サポート・クラスの作成時には，次の基本的な認識プロパティ・サポート・メソッドが一般的に使用されます。第 II 部「チュートリアル : Java カスタム・ツールキット・サポートの作成方法の学習」では，これらのメソッドの使用について練習することができます。

- **to_class_attr** メソッド（61 ページ「カスタム・コントロールのテスト・オブジェクト・クラスへのマッピング」を参照）は，**Class Name** 認識プロパティをサポートします。このメソッドは，関連するテスト・オブジェクト・クラスの名前を返すことによって，カスタム・コントロールのテスト・オブジェクト・クラスへのマッピングを実現します。UFT は，このプロパティを使用して，カスタム・コントロールにマッピングされているテスト・オブジェクト・クラスを決定します。

- テスト・オブジェクトの名前は，オブジェクトの **tag** プロパティで決まります。AWT サポート・クラスはすべて ObjectCS を拡張します。ObjectCS は，実装している **tag_attr** メソッドを使用して指定された順序で一連のプロパティをチェックし，見つかった最初の有効値を返します。有効値は空でない値で，スペースを含みません。

  ObjectCS クラスの **tag_attr** メソッドでは，（記載された順序で）次のプロパティがチェックされます。

  - label
  - attached_text（詳細は，下記を参照）
  - 非修飾カスタム・クラス（パッケージ名を含まないクラスの名前）

  カスタム・コントロールのテスト・オブジェクトの名前を変更するには，サポート・クラスで **tag_attr** メソッドをオーバーライドせず，既存の実装を使用して，**label_attr** メソッドをオーバーライドします。

- また，ObjectCS は **attached_text_attr** メソッドを実装します（AWT サポート・クラスはすべて ObjectCS を拡張します）。このメソッドは，カスタム・コントロールに隣接する静的テキスト・オブジェクトを検索して，そのテキストを返します。このメカニズムは，独自の説明テキストを持たないラベル付きのエディット・ボックスやリスト・ボックスのようなコントロールの場合に役に立ちます。

  カスタム静的テキスト・コントロールのサポートを作成すると，UFT でその **label** プロパティを隣接するコントロールの **attached text** として使用することができます。詳細については，148 ページ「New UFT Custom Static-Text Support Class ウィザード」を参照してください。

- **class_attr** メソッドは，テスト・オブジェクトの汎用タイプの名前（**object**，**button**，**edit**，**menu**，**static_text** など）を返します。これは，オブジェクトにマッピングされた特定のテスト・オブジェクト・クラスでなく，テスト・オブジェクト・クラスの一般的なタイプです。静的テキスト・コントロールのサポート・クラスを作成する場合は，文字列 static_text を返すように **class_attr** メソッドを実装する必要があります。それ以外の場合にはオーバーライドしないでください。
- **value_attr** メソッドは必須でありませんが，コントロールの現在の状態を表すのによく使用される，**value** 認識プロパティを実装します。たとえば，**value_attr** メソッドでは，タブ・コントロールで現在選択されているタブの名前，ツリーで現在選択されている項目のパス，メニューで現在表示されている項目などを返すことができます。新しいテスト・オブジェクト・クラスを作成していて，**現在の状態**が関連する場合は，**value** 認識プロパティのサポートを実装します。使用するサポート・クラスが **value_attr** メソッドを継承する場合は，その実装がサポートされるコントロールに適していることを確認します。

テスト・オブジェクト・メソッドのサポート

カスタム・コントロールのテスト・オブジェクト・メソッドは，テスト・オブジェクト・クラスで定義されます。これは，UFT の既存のテスト・オブジェクト・クラスであるか，またはテスト・オブジェクト設定ファイルで定義するテスト・オブジェクト・クラスです。

テスト・オブジェクト・メソッドのサポートをサポート・クラスで提供するには，テスト・オブジェクト・メソッドごとに次の署名を使用して **replay** メソッドを実装します。

```
public Retval <テスト・オブジェクト・メソッド名>_replayMethod(Object obj, <... list of String arguments>)
```

obj 引数は，カスタム・コントロールを表すオブジェクトです。

replay メソッドは文字列の引数のみ受け入れるため，UFT はすべての引数を文字列形式で replay メソッドに渡します。ブール値または数値の引数を使用する場合は，**MicAPI.string2int** を使用します。

replay メソッド内では，カスタム・クラスの public メソッドを使用するか，MicAPI メソッドを使用して低レベル・イベントをディスパッチすることにより，カスタム・コントロール上で必要な操作を実行します（サポート・クラスは，public と定義されているカスタム・クラス・メソッドにのみアクセスできます）。詳細については，『UFT Java Add-in Extensibility API Reference』(『Java Add-in Extensibility SDK ヘルプ』で利用可能)を参照してください。

たとえば，次のように，（ImageButtonCS クラスの）**Click_replayMethod** は，ImageButton カスタム・コントロール上で **Click** テスト・オブジェクト・メソッドをサポートします。

```
public Retval Click_replayMethod(Object obj) {
   ImageButton button = (ImageButton) obj;
   MicAPI.mouseClick((Object) button, button.getWidth() / 2,
                                              button.getHeight() / 2);
   Return Retval.OK;
}
```

replay メソッドはすべて **MicAPI.Retval** 値を返す必要があります。**Retval** 値には，必ずリターン・コードが含まれており，文字列の戻り値を含めることもできます。UFT は，リターン・コードでテスト・オブジェクト・メソッドの成功または失敗に関する情報を把握します。戻り値を取得して UFT GUI テストの後続のステップで使用することができます。

たとえば，（SearchBox カスタム・コントロールをサポートする）SearchBoxCS クラスの GetItem_replayMethod は，リターン・コード **OK** に加えて，指定された項目の名前を返します。

```
public Retval GetItem_replayMethod(Object obj, String Index) {
   SearchBox sb = (SearchBox) obj;
   int indexint;
   String item;
   indexint = MicAPI.string2int(Index);
   if (indexint == MicAPI.BAD_STRING) {
      return Retval.ILLEGAL_PARAMETER;
   }
   if (indexint < 0 || indexint > sb.getItemCount() - 1) {
      return Retval.OUT_OF_RANGE;}
   item = sb.getItem(indexint);
   return new Retval(RError.E_OK, item);
}
```

UFT で認識される **MicAPI.Retval** の値の詳細については，『UFT Java Add-in Extensibility API Reference』（『Java Add-in Extensibility SDK ヘルプ』で利用可能）を参照してください。

サポート・クラスが機能の似たコントロールのサポート・クラスを拡張する場合，カスタム・コントロールに変更なしで適用されるテスト・オブジェクト・メソッドのサポートを実装する必要はありません。たとえば，多くのコントロールには **Click** テスト・オブジェクト・メソッドが含まれています。**Click** テスト・オブジェクト・メソッドの実装済みのサポートで十分にカスタム・コントロールをサポートできる場合，親のメソッドをオーバーライドする必要ありません。

親のサポート・クラスでサポートされないカスタム・コントロールのテスト・オブジェクト・メソッドをサポートするには，使用するサポート・クラスに新しいメソッドを追加します。サポートされているものと名前が同じで実装の異なるテスト・オブジェクト・メソッドをサポートする場合は，親のメソッドをオーバーライドします。

CheckProperty，**FireEvent**，**GetRoProperty**，**GetTOProperty**，**SetTOProperty**，**WaitProperty** などの，UFT の基本的なメソッドの実装はオーバーライドしないでください。

注 : JavaTree テスト・オブジェクトと JavaList テスト・オブジェクトをサポートする場合は, **count_attr** メソッドと **GetItem_replayMethod** メソッドが UFT で期待されるタイプの情報を返すようにする必要があります。詳細については，63 ページ「予約済みの認識プロパティ」を参照してください。

記録オプションのサポート

記録オプションの UFT サポートは, AWT を拡張するコントロールに対してのみ拡張することができます。

記録用のサポートを実装しなくても，カスタム・コントロールに対する UFT のその他のすべての機能（オブジェクトの学習，オブジェクトに関するテストの実行，プロパティと値のチェックなど）は完全にサポートされます。

カスタム・コントロールで記録をサポートするには，カスタム・サポート・クラスで次を実行する必要があります。

- 記録をトリガするイベントのリスナを実装する必要があります。
- 作成時にカスタム・コントロールにリスナを登録する必要があります。
- 関連するイベントが発生したときに UFT に記録イベントを送信する必要があります。
- より複雑な操作を記録する場合は，低レベル記録をオーバーライドする必要があります。たとえば，**JavaEdit.Set** 操作を記録する場合は，個別のキーボード入力の記録をオーバーライドする必要があります。メニューのオプション選択を記録する場合は，マウス・クリックの記録をオーバーライドする必要があります。

第 II 部「チュートリアル : Java カスタム・ツールキット・サポートの作成方法の学習」では，カスタム・コントロール上の記録のサポートの作成を練習することができます。

カスタム・サポート・クラスに記録のサポートを追加するには，次の手順を実行します。

1 サポート・クラスの署名にリスナを追加します。次の例では，ImageButton のサポート・クラスである ImageButtonCS がアクション・イベントをリッスンします。

```
public class ImageButtonCS extends CanvasCS implements ActionListener {}
```

2 サポート・クラスのコンストラクタを使用して，カスタム・コントロールに登録するすべてのリスナと，これらのリスナの追加と削除に使用するメソッドを含むリストを生成します。

このためには，各リスナに対してユーティリティ・メソッド **addSimpleListener** を呼び出します。このメソッドは，文字列型の 3 つの引数（リスナの名前，登録メソッドの名前，リスナの削除に使用するメソッドの名前）を受け取ります。

次の例では, アクション・リスナを ImageButton カスタム・コントロールに登録します。

```
public ImageButtonCS() {
  addSimpleListener("ActionListener", "addActionListener", "removeActionListener");
}
```

UFT はカスタム・コントロールを初めて認識したときに，このカスタム・コントロールのサポート・クラスのインスタンスを作成します。このサポート・クラスのインスタンスは，このカスタム・クラスの以降のすべてのコントロールをサポートするのに使用されます。カスタム・クラスのインスタンスが作成されると，サポート・クラスは指定した登録メソッドを使用してオブジェクトに必要なリスナを登録します。

3 低レベル記録をオーバーライドします（オプション）。

低レベル・マウス・イベントの記録をオーバーライドする場合：

```
protected Object mouseRecordTarget(MouseEvent e) {
   return null;
}
```

低レベル・キーボード・イベントの記録をオーバーライドする場合：

```
protected Object keyboardRecordTarget(KeyEvent e) {
   return null;
}
```

4 **MicAPI.record** メソッドを使用して，記録メッセージを UFT に送信するために，リスナ・インタフェースから関連するイベント・ハンドラ・メソッドを実装します。

MicAPI.record の使用方法については,『UFT Java Add-in Extensibility API Reference』(『Java Add-in Extensibility SDK ヘルプ』で利用可能)を参照してください。

たとえば，ImageButtonCS（ImageButton のサポート・クラス）内では，次のイベント・ハンドラ・メソッドが実装されます。

```
public void actionPerformed(ActionEvent e) {
   try {
         if (!isInRecord())
           return;
         MicAPI.record(e.getSource(), "Click");
      } catch(Throwable tr)
         { tr.printStackTrace();
         }
}
```

ImageButton でアクション・イベントが発生すると，UFT は ImageButton 上の **Click** 操作を記録します。

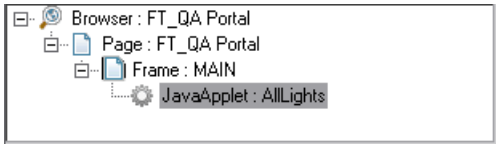
try ... catch ブロックは，UFT がアイドル状態のときに Java アプリケーションを実行してこのコードに到達した場合に，必要のない動作が実行されないようにします。スタック・トレースが Java Add-in Extensibility のほかのログ・メッセージと同じログ・ファイルに出力されるため，このメソッドが誤って呼び出された場合にそのタイミングを確認することができます。詳細については，88 ページ「カスタム・サポート・クラスのログとデバッグ」を参照してください。

ラッパー・コントロールでの記録については，71 ページ「ラッパー・コントロールのサポート」を参照してください。

注 : UFT のアクティブな記録セッションが存在しない状態で **MicAPI.record** を呼び出しても，何も起こりません。**MicAPI.record** を呼び出す前に計算や代入を追加で行う場合は，最初に **isInRecord** を呼び出して記録セッションがアクティブかどうかを確認する必要があります。記録セッションがアクティブでない場合は，特定の操作を回避するようにしてください。

トップレベル・オブジェクトのサポート

UFT でカスタム・コントロールをテスト・オブジェクト階層の最上位の Java オブジェクトとして認識する必要がある場合は，この Java コントロールがトップレベル・オブジェクトであることを UFT に指示する必要があります。このためには，サポート・クラスでユーティリティ・メソッド **isWindow(Object obj)** をオーバーライドして **true** を返すようにします。次の例で，JavaApplet **AllLights** はトップレベルの Java オブジェクトです。

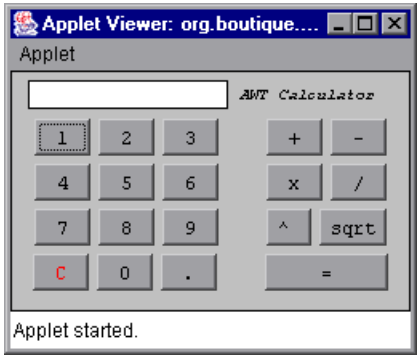


トップレベル・オブジェクトにすることができるのは，コンテナ・オブジェクトのみです。コンテナ・オブジェクトは，**java.awt.container**（AWT ベースの場合）または **org.eclipse.swt.widgets.Composite**（SWT ベースの場合）を拡張するオブジェクトです。

ある状況でのみコントロールがトップレベル・オブジェクトになる場合は，その状況で **true** を返し，それ以外の状況では **false** を返すように **isWindow** メソッドを実装することができます。たとえば，アプレットはスタンドアロン・アプリケーションにすることも，Web ブラウザ内のオブジェクトにすることもできます。

ラッパー・コントロールのサポート

ラッパー・コントロールは，ラッパー・コントロール内のコントロールをグループ化して，1 つのコントロールとして表すコンテナ・コントロールです。たとえば，AwtCalc 計算機コントロールは，ラッパー・コントロールです。



UFT は，ラッパー・コントロールを学習する際に，ラッパー・コントロール内のコントロールを子孫として個別に学習しません。ラッパー・コントロールでテストを記録する場合，ラッパー・コントロール内のコントロールで発生するイベントはラッパー・コントロール上の操作として記録されます。

注 : UFT では，AWT ベースのコントロールのみをラッパー・コントロールとしてサポートできます。SWT ベースのカスタム・コントロールの場合は，必ずそのすべての子孫とともに学習が行われます。

たとえば，AwtCalc 計算機コントロールには，数字と演算子のシンプルなボタンが含まれています。このコントロールの記録セッションでは，単純な **Click** 操作を意味のある計算機操作として解釈することが必要になる場合があります。Java Add-in Extensibility を使用すると，UFT で数字ボタンのクリックを **Calculator.SetValue** ステップとして記録し，演算子ボタンのクリックを **Calculator.SetOperator** ステップとして記録するように指定することができます。

UFT でのラッパー・コントロールの扱い方について

ラッパー・コントロールでは，ラップする各種コントロールにラッパー・コントロールそのものをラッパーとして登録する必要があります。

UFT では，コントロールを子孫として学習する前に，このタイプのコントロールに登録されているラッパーが存在するかどうかを確認します。登録されているラッパーが存在する場合，UFT はこのコントロールが含まれるラッパーを検索します。UFT は，この検索を行うために，登録されている各ラッパーの **checkWrappedObject** メソッドを呼び出します。該当するラッパーが見つかった場合，UFT は子孫のコントロールを学習しません。該当するラッパーが見つからない場合，UFT は子孫のコントロールを学習します。

（特定のコントロールをクリックして）コントロールを個別に学習する場合，UFT はラッパーの確認を行いません。

同様に，UFT では，コントロール上の操作を記録する前に，このタイプのコントロールに登録されているラッパーが存在するかどうかを確認します。登録されているラッパーが存在する場合，UFT はこのコントロールが含まれるラッパーを検索します。該当するラッパーが見つかった場合，UFT はテストにステップを追加する前にラッパー・コントロールに記録メッセージを渡します。該当するラッパーが見つからない場合は，操作がそのまま記録されます。

ラッパーが記録メッセージ（ラッパーに含まれるいずれかのオブジェクトで実行された操作によってトリガされたメッセージ）を受け取ったときには，次のいずれかを実行できます。

- メッセージを破棄して，操作の記録が行われないようにします。
- メッセージを変更して，異なる操作を記録します。
- メッセージをそのままにして，介入なしに操作を記録します。

次の項では，AwtCalc ラッパー・コントロールの例を使用して，このメカニズムの実装方法を説明します。AwtCalc コントロールのサポートを実装した後では，コントロールで記録されたテストは次のようになります。

Item	Operation	Value	Documentation
Action1			
AwtCalculator	Reset		Reset the calculator value
AwtCalculator	SetValue	"2"	Set "2" value into "AwtCalculator" object
AwtCalculator	SetOperator	"+"	Set "+" operation to calculate
AwtCalculator	SetValue	"2"	Set "2" value into "AwtCalculator" object
ETextField	Click	82,5,"LEFT"	Click the "ETextField" object with the "LEFT" mouse button.
AWT Calculator(st)	Click	40,8,"LEFT"	Click the "AWT Calculator(st)" text label with the "LEFT" mouse button.
AwtCalculator	Enter		Calculate the incoming data

ラッパー・コントロールのサポートの実装

ラッパー・コントロールをサポートする場合は，MicAPI で **com.mercury.ftjadin.infra.abstr.RecordWrapper** インタフェースを実装する必要があります。このインタフェースには，次のメソッドが含まれています。

- **public void registerWrapperInspector()**
- **public Object checkWrappedObject(Object obj)**
- **public RecordMessage wrapperRecordMessage(RecordMessage message, Object wrapper)**
- **public boolean blockWrappedObjectRecord()**

次の各項では，これらのメソッドについて詳しく説明します。

public void registerWrapperInspector()

registerWrapperInspector メソッドは，関連するタイプのコントロールにラッパーとして登録する場合に使用します。

次の例では，AwtCalcCS サポート・クラスがそれ自身を **Button** のラッパーとして登録します。

```
public void registerWrapperInspector() {
   MicAPI.registerWrapperInspector(Button.class, this);
}
```

AwtCalcCS は **Button** コントロールでのみラッパーとして登録されます。そのため，**AWT Calculator** ラベルやエディット・ボックス上での操作はラッパーの介在なしに記録されます。また，AwtCalc コントロールの学習時には，ラベルとエディット・ボックスがコントロールの子孫として学習されます。

public Object checkWrappedObject(Object obj)

UFT は，**checkWrappedObject** メソッドを呼び出して，特定のオブジェクトがカスタム・コントロールに含まれるかどうかを確認します。サポート・クラスでは，**obj** がカスタム・コントロールでラップされている場合に特定のラッパー・インスタンスを返すように，このメソッドを実装します。それ以外の場合は，null が返されます。

たとえば，AwtCalcCS の **checkWrappedObject** メソッドは，次のように実装されます。

```
public Object checkWrappedObject(Object obj) {
Component comp = (Component)obj;
if (comp.getParent().getClass().getName().equals("org.boutique.toolkit.AwtCalc"))
   return comp.getParent();
return null;
}
```

public RecordMessage wrapperRecordMessage(RecordMessage message, Object wrapper)

記録セッション時にラップされたオブジェクトが記録メッセージを送信すると，UFT は **wrapperRecordMessage** メソッドを呼び出します。UFT はテストにステップを追加する前に，この記録メッセージをラッパー・コントロールに渡します。

このメソッドは，次のいずれかを返します。

- **null**（このメッセージを無視して，ステップを記録しないことを示す）
- 元のメッセージの代わりに送信する変更された記録メッセージ
- 元の記録メッセージ

たとえば，AwtCalcCS の **wrapperRecordMessage** メソッドでは，ボタン上で記録対象の操作が行われた場合に，このメソッドで記録対象の適切な操作（Reset，Enter，SetOperator，または SetValue（該当するパラメータを含む））に置き換えます。ラベル・フィールドやテキスト・フィールドで記録メッセージの操作が行われた場合，AwtCalc は記録に干渉しません。

```
public RecordMessage wrapperRecordMessage(RecordMessage message,
Object wrapper) {
    Object subject = message.getSubject();
    if (subject instanceof Button) {
        // Get the label of the button
        String value = ((Button) subject).getLabel().trim();
        String operation;
        // Select what method will be recorded and with what parameters
        if (value.equals("=")) {
            return RecordMessage.getRecordMessageInstance(wrapper,"Enter");
        }
```

```
            if (value.equals("C")) {
                return RecordMessage.getRecordMessageInstance(wrapper,"Reset");
            } else {
                if (value.equals("+") || value.equals("-") || value.equals("x")
                        || value.equals("/") || value.equals("^")
                        || value.equals("sqrt"))
                    operation = "SetOperator";
                else
                    operation = "SetValue";
             }
            String params[] = new String[1];
            params[0] = value;
            RecordMessage res =
                RecordMessage.getRecordMessageInstance(wrapper, operation,
                    params, AgentRecordMode.NORMAL_RECORD);
            return res;
        }
        // AwtCalc does not interfere if the message is not from a button
        return message;
    }
```

boolean blockWrappedObjectRecord()

blockWrappedObjectRecord メソッドが **false** を返す場合，ラッパーに含まれるコントロールは，独立したコントロールの場合と同様に，イベントに応答して記録メッセージを生成します。その後，UFT は **wrapperRecordMessage** を呼び出して，ラップされたコントロールから受け取る記録メッセージをラッパーに渡します。その後，ラッパーでは，メッセージを破棄するか，メッセージを変更するか，または操作をそのまま記録するかを判断することができます。

blockWrappedObjectRecord メソッドが **true** を返す場合，ラッパーに含まれるすべてのコントロールですべてのイベントが無視されます。ラップされたコントロールが記録メッセージを UFT に送信することはなく，**wrapperRecordMessage** が呼び出されることもありません。

blockWrappedObjectRecord が null を返す場合で，ラッパーに含まれるオブジェクトで発生するイベントを記録する必要がある場合は，ラッパー自身がラップされたオブジェクトに新しいイベント・リスナを登録する必要があります。その後，記録セッション中に（**MicAPI.record** を使用して）適切なテスト・ステップを生成するようにイベントを処理する必要があります。

サポート・クラスのサマリ

次の表は，カスタム・サポート・クラスで使用するメソッドの種類をまとめたものです。詳細については，『UFT Java Add-in Extensibility API Reference』(『Java Add-in Extensibility SDK ヘルプ』で利用可能)を参照してください。

メソッドの種類	構文	一般的なメソッド
認識プロパティ・メソッド	public String <認識プロパティ名> _attr(Object obj)	to_class_attr tag_attr label_attr attached_text_attr class_attr value_attr
テスト・オブジェクト・メソッド	public Retval <テスト・オブジェクト・メソッド名>_replayMethod(Object obj, <... list of String arguments>)	
イベント・ハンドラ・メソッド	実装するリスナによって異なります。	イベント・ハンドラ・メソッドから MicAPI.record を呼び出します。
使用するユーティリティ・メソッド	protected void addSimpleListener(String listenerName, String addMethodName, String removeMethodName) public static final boolean isInRecord()	
オーバーライドするユーティリティ・メソッド	public boolean isWindow(Object obj) protected Object mouseRecordTarget(MouseEvent e) protected Object keyboardRecordTarget(KeyEvent e) public boolean blockWrappedObjectRecord() public void registerWrapperInspector() public Object checkWrappedObject(Object obj) public RecordMessage wrapperRecordMessage(RecordMessage message, Object wrapper)	

MicAPI のメソッドの使用

MicAPI には，次の種類の機能を提供する場合にカスタム・サポート・クラスで使用できる複数のメソッド・セットが含まれています。

- 低レベル・イベントのディスパッチ。これらのメソッドには，**MouseClick**，**KeyType**，**postEvent** などがあります。これらのメソッドは一般的に再生メソッドで使用されます。
- UFT でのカスタム・コントロールの操作の記録。これらのメソッドは一般的にイベント・ハンドラ・メソッドで使用されます。
- サポート・クラスからのメッセージやエラーのログ記録。これらのメソッドは，ログ・メッセージやエラー・メッセージを出力する場合に，カスタム・サポート・クラス内で使用されます。詳細については，88 ページ「カスタム・サポート・クラスのログとデバッグ」を参照してください。

MicAPI で提供されるメソッドを使用するには，import com.mercury.ftjadin.custom.MicAPI; ステートメントをコードに追加します。これらのメソッドの詳細については，『UFT Java Add-in Extensibility API Reference』(『Java Add-in Extensibility SDK ヘルプ』で利用可能)を参照してください。

カスタム・ツールキット・サポートのデプロイと実行

カスタム・ツールキットに対する UFT サポートの拡張では，最後にデプロイを行います。この手順では，作成したすべてのファイルを適切な場所に配置して，カスタム・ツールキット・サポートが UFT で利用できるようにします。

開発段階でツールキット・サポートをデプロイして，UFT への影響のテストや，作成しているカスタム・ツールキット・サポート・セットのデバッグを行うこともできます。

カスタム・ツールキット・サポートのデプロイについて

UFT がインストールされているコンピュータにツールキット・サポート・セットをデプロイすると，UFT ユーザはツールキット・サポート・セットを UFT アドインとして使用できるようになります。

UFT が起動すると，アドイン・マネージャに Java Add-in ノードの子ノードとしてカスタム・ツールキット名が表示されます。カスタム・ツールキットのチェック・ボックスを選択すると，開発したツールキット・サポート・セットを使用して，そのツールキットのサポートがロードされます。

注 : カスタム・ツールキットのサポートをロードまたはアンロードした後で開いたアプリケーションだけに影響します。

カスタム・ツールキットのサポートをロードしない場合，ツールキット・サポート・セットで設計したコードは実行されません。

カスタム・ツールキットのサポートをロードすると，次の処理が実行されます。

- UFT でカスタム・ツールキットのコントロールが認識され，そのコントロールに関するテスト・ステップを実行することができます。
- アドインまたはサポートされている環境をリスト表示するすべてのダイアログ・ボックスで，［**環境**］リストにカスタム・ツールキットの名前が表示されます。
- 各アドインまたは環境で使用可能なテスト・オブジェクト・クラスをリスト表示するダイアログ・ボックス（たとえば，［新規テスト オブジェクトの定義］ダイアログ・ボックスや［オブジェクトの認識］ダイアログ・ボックスなど）で，ツールキット・サポート・セットで定義したテスト・オブジェクト・クラスのリストが表示されます。

注: バージョン 10.00 より前の Java Add-in Extensibility SDK を使用して開発されたツールキット・サポート・セットで定義されたテスト・オブジェクト・クラスは，Java テスト・オブジェクト・クラスとして UFT のダイアログ・ボックスに表示されます。UFT でこれらのテスト・オブジェクト・クラスを正しい環境名の下に表示するには，アドイン・マネージャで表示されるとおりに，テスト・オブジェクト設定ファイルの **PackageName** 属性をカスタム・ツールキットの名前に変更します。また，ツールキット・サポート・セットのテスト・オブジェクト・クラスで **index** 認識プロパティが実装されている場合は，この実装を削除して［オブジェクトの認識］ダイアログ・ボックスで［**スクリプトの生成**］ボタンを使用できるようにします。

カスタム・ツールキット・サポートのデプロイ

ツールキット・サポート・ファイルを配置する場所を次の表にまとめます。

ファイル名	場所
<カスタム・ツールキット名>.xml	**<UFT インストール・フォルダ>\bin\java\classes\extension**
<カスタム・ツールキット名>TestObjects.xml オプション。カスタム・クラスを新しいテスト・オブジェクト・クラスにマッピングする場合のみ指定します。 **注 :** このファイル名は，Java Add-in Extensibility ウィザードで使用されるものです。テスト・オブジェクト設定ファイルを複数作成して，それぞれに任意の名前を付けることができます。	➤ **<UFT インストール・フォルダ>\Dat\Extensibility\Java** ➤ **<Unified Functional Testing Add-in for ALM インストール・フォルダ>\Dat\Extensibility\Java** （オプション。ALM 用 Unified Functional Testing アドインがインストールされている場合のみ必要です）
<カスタム・ツールキット名>Support.class **<CustomClass>CS.class**	コンパイル済みのすべての Java サポート・クラスを, UFT がインストールされたコンピュータ（またはネットワーク上のアクセス可能な場所）にあるクラス・フォルダや Java アーカイブにパッケージ化できます。 ファイルの場所は，**<カスタム・ツールキット名>.xml** で指定します。
新しいテスト・オブジェクト・クラスのアイコン・ファイル（オプション）	**.dll** または **.ico** ファイルを, UFT がインストールされているコンピュータまたはネットワーク上のアクセス可能な場所に格納します。 ファイルの場所は，**<カスタム・ツールキット名>TestObjects.xml** で指定します。

開発段階でのカスタム・サポートのデプロイ

カスタム・ツールキット・サポートの設計段階では，サポート・クラス・ファイルをワークスペース内に残しておくことができます。カスタム・ツールキット・サポートをデプロイするには，ツールキット設定ファイル（テスト・オブジェクト設定ファイルを含む）を適切な場所に配置し，ツールキット設定ファイル（XML ファイル）でコンパイル済みのサポート・クラスの場所を指定します。さらに，新しいテスト・オブジェクト・クラスで固有のアイコンを使用する場合には，テスト・オブジェクト設定ファイルでそれぞれのアイコンの場所を指定します。

注：デプロイする前にサポート・クラスをコンパイルし，コンパイル・エラーを確認して，実行時エラーが起きないようにしてください。

テスト・オブジェクト設定ファイルの **Identification Property** 要素の属性を変更する場合，カスタム・ツールキット・サポートの設計段階では，**TypeInformation** 要素の **DevelopmentMode** 属性の値を true にしておいてください。詳細については，85 ページ「テスト・オブジェクト設定ファイルで指定された認識プロパティ属性の変更」を参照してください。

Eclipse で UFT Java Add-in Extensibility プラグインを使用してカスタム・ツールキット・サポートを開発し，UFT が使用するコンピュータにインストールされている場合，ツールキット・サポートをデプロイするには，Eclipse のツールバーの［**Deploy Toolkit Support**］ボタンをクリックするか，［**UFT**］＞［**Deploy Toolkit Support**］を選択します。XML 設定ファイルが UFT の適切な場所にコピーされます。Java クラスのファイルは Eclipse ワークスペース内にそのまま残ります（ツールキット・サポート・クラスとカスタム・サポート・クラスの実際の場所は，ツールキット設定ファイル内に記載されます）。Eclipse プラグインを使用したサポートのデプロイの詳細については，153 ページ「Deploy Toolkit Support」を参照してください。

Eclipse で UFT Java Add-in Extensibility プラグインを使用しない場合，または UFT が別のコンピュータにインストールされている場合は，表（78 ページ）に記載された情報に基いて，手動でデプロイを行う必要があります。

開発段階でカスタム・サポートを手動でデプロイするには，次の手順を実行します。

1 コンパイル済みサポート・クラス（ツールキット・サポート・クラスとカスタム・サポート・クラス）が UFT でアクセス可能な場所にあることを確認します。

2 コンパイル済みサポート・クラスとアイコン・ファイル（該当する場合）の場所を反映して設定ファイルを更新します。

3 表（78 ページ）の説明に従って，設定ファイルを適切なフォルダにコピーします。

設計完了後のカスタム・サポートのデプロイ

カスタム・ツールキット・サポートの設計が完了したら，UFT がインストールされている任意のコンピュータにデプロイすることができます。

カスタム・ツールキット・サポートの開発時に，テスト・オブジェクト設定ファイルの **TypeInformation** 要素の **DevelopmentMode** 属性を true に設定した場合は，実際の使用のためにカスタム・サポートをデプロイする前に，この属性を削除（または false に設定）してください。詳細については，85 ページ「テスト・オブジェクト設定ファイルで指定された認識プロパティ属性の変更」を参照してください。

設計完了後にカスタム・サポートをデプロイするには，次の手順を実行します。

1 コンパイル済みサポート・クラス（ツールキット・サポート・クラスとカスタム・サポート・クラス）をそれぞれの恒久的な場所に配置します。これらのクラスは，UFT からアクセス可能な場所にある，クラス・フォルダまたは Java アーカイブに配置できます。

また，固有のアイコンを使用する新しいテスト・オブジェクト・クラスがある場合は，アイコン・ファイルを UFT からアクセス可能な場所に配置します。

2 コンパイル済みサポート・クラスの場所を反映して，ツールキット設定ファイルを更新します。

必要に応じて，アイコン・ファイルの場所を反映して，テスト・オブジェクト設定ファイルを更新します。

3 表（78 ページ）の説明に従って，設定ファイルを適切なフォルダにコピーします。

サポートされているカスタム・コントロールを含むアプリケーションの実行

カスタム・ツールキット・サポートのデプロイが済んだら，サポートされているカスタム・コントロールを含むアプリケーションに対して UFT の操作を実行して，サポートの影響をテストすることができます。

アプリケーションは，どのような方法で実行しても構いません。

バージョン 3.3 より前の Eclipse から SWT アプリケーションを実行した場合, Java ライブラリ・パスがオーバーライドされて SWT の dll が追加されます。そのため，(Java Add-in で必要となる) **jvmhook.dll** のパスをライブラリ・パスに手動で追加する必要があります。

jvmhook.dll のパスをライブラリ・パスに追加するには，次の手順を実行します（バージョン 3.3 より前の Eclipse を使用する場合）。

1. Eclipse Package Explorer でアプリケーション・ファイルを右クリックします。[**Run As**] ＞［**SWT Application**］を選択します。
2. Eclipse のツールバーで，［**Run**］＞［**Run**］を選択します。［Run］ダイアログ・ボックスが開きます。
3. ［**Configurations**］リストで SWT アプリケーションを選択します。
4. ［**Arguments**］タブをクリックします。
5. ［VM Arguments］領域に，次のように入力します。

   ```
   -Djava.library.path=<システム・フォルダ>\system32
   ```

   （例：-Djava.library.path=c:\WINNT\system32）
6. アプリケーションを閉じ，手順 1 を繰り返してアプリケーションを再度実行します。

デプロイ済みサポートの変更

UFT にデプロイ済みのツールキット・サポート・セットを変更する場合，どのような変更を行うかによって実行するべきアクションが異なります。

- ツールキット設定ファイルまたはテスト・オブジェクト設定ファイルを変更する場合には，サポートをデプロイする必要があります。
- テスト・オブジェクト設定ファイルを変更する場合には，サポートのデプロイ後に UFT を再度起動する必要があります。
- 設定ファイルを変更する場合でも，Java　サポート・クラスのみを変更する場合でも，変更内容を反映するには，Java アプリケーションを再起動する必要があります。

テスト・オブジェクト設定ファイルで指定された認識プロパティ属性の変更

テスト・オブジェクト設定ファイルに含まれる**識別子プロパティ**要素について，**AssistivePropertyValue**，**ForAssistive**，**ForBaseSmartID**，**ForDescription**，**ForOptionalSmartID**，**OptionalSmartIDPropertyValue** の各属性を UFT で変更できます（[オブジェクトの認識］ダイアログ・ボックスを使用）。この属性に基づいて，UFT で各種用途に使用する認識プロパティのリストが決まります。詳細については，UFT Test Object Schema ヘルプ（UFT Java Add-in Extensibility ヘルプからアクセス）を参照してください。

したがって標準設定では，UFT は属性値を XML ファイルから 1 度だけ読み込みます。これにより，［オブジェクトの認識］ダイアログ・ボックスでユーザが行った変更内容が上書きされることはなくなり，ユーザ定義のプロパティ・リストが保持されます。

［オブジェクトの認識］ダイアログ・ボックスの［**テスト オブジェクトを元に戻す**］ボタンをクリックすると，属性値が XML から再ロードされます。

前回のロード以降 XML が変更されている場合（システムのファイル変更日時から判断），属性値が XML から読み込まれます。さらにこの属性値に基づいて，認識プロパティが関連のリストに追加されます（必要に応じてリスト内の順序も調整されます）。ただし，リストから既存の認識プロパティが削除されることはありません。

UFT を起動するたびに XML 内で定義されている属性値に基づいて認識プロパティすべてを更新するには，テスト・オブジェクト設定ファイルの **TypeInformation** 要素の **DevelopmentMode** 属性を true に設定します。

認識プロパティの属性の変更に関する考慮事項

- テスト・オブジェクト設定ファイルの**Identification Property** 要素を変更する場合，カスタム・ツールキット・サポートの設計中は，**TypeInformation** 要素の **DevelopmentMode** 属性の値を true にしてください。これにより，ファイルへの変更内容すべてが UFT で有効になります。
- 通常のツールキット・サポート・セットをデプロイする前に，**TypeInformation** 要素の **DevelopmentMode** 属性を削除（または false に設定）してください。削除または無効にしないと，UFT が起動するたびに，テスト・オブジェクト設定ファイルでの定義に基づいてプロパティ・リストが更新されます。UFT ユーザが［オブジェクトの認識］ダイアログ・ボックスでプロパティ・リストを変更した後，UFT を再起動すると，変更内容は破棄されます。
- テスト・オブジェクト設定ファイルの読み込み時，既存のプロパティがプロパティ・リストから削除されることはありません（ただし，**DevelopmentMode** 属性が true に設定されている場合を除く）。プロパティはリストに追加され，ファイル内の定義に基づいて順序が調整されます。UFT ユーザが［オブジェクトの認識］ダイアログ・ボックスでリストからプロパティを削除した場合や，順序を変更した場合には，変更後のファイルをロードした時点で変更内容は破棄されます。

  カスタム・ツールキット・サポート・セットを第三者に提供し，変更済みのテスト・オブジェクト設定ファイルを含むアップグレードを行う場合には，認識プロパティのリストが変更される可能性があることを UFT ユーザに通知してください。

デプロイ済みサポートの削除

UFT では，特定のツールキット向けに提供されているサポートを起動時にロードするかどうかをアドイン・マネージャで指定できます。カスタム・ツールキットのサポートをロードしない場合，ツールキット・サポート・セットで設計したコードは実行されません。また，テスト・オブジェクト設定ファイルで定義したテスト・オブジェクト・クラスを UFT で使用することもできません。

- デプロイの完了後，カスタム・ツールキットのサポートを UFT から削除するには，次の場所からツールキット設定ファイルを削除する必要があります。
  <UFT インストール・フォルダ>\bin\java\classes\extension
- テスト・オブジェクト設定ファイル内に，カスタム・コントロールを表すテスト・オブジェクト・クラス定義が存在しない場合（つまりファイルが不要），このファイルを次の場所から削除できます。
  <UFT インストール・フォルダ>\Dat\Extensibility\Java（および **<Unified Functional Testing Add-in for ALM インストール・フォルダ>\Dat\Extensibility\Java**（該当する場合））
- 作成したカスタム・ツールキット・サポートの一部のみを削除する場合は，次の点に注意してください。
  - 特定のカスタム・クラスのサポートを削除する場合は，そのカスタム・サポート・クラスを削除し，ツールキット設定ファイルからこのサポート・クラスへの参照を削除します。

    カスタム・サポート・クラスを削除する際には，そのカスタム・サポート・クラスを拡張するほかのカスタム・サポート・クラスが存在しないことを確認してください。
  - 定義した新しいテスト・オブジェクト・クラスを削除する場合は，テスト・オブジェクト設定ファイルからその定義を削除します。

    テスト・オブジェクト・クラスの定義を削除する際には，このテスト・オブジェクト・クラスにマッピングされているカスタム・クラスが存在しないこと，およびそのテスト・オブジェクト・クラスを拡張するほかのテスト・オブジェクト・クラスが存在しないことを確認してください。
  - 追加したテスト・オブジェクト・メソッドまたは認識プロパティのサポートを削除する場合は，カスタム・サポート・クラスから関連するサポート・メソッドを削除します。

    サポート・クラスからテスト・オブジェクト・メソッドまたは認識プロパティのサポートを削除しても，これらがテスト・オブジェクト・クラスの定義から削除されることはありません。これらは，テストを編集する際に UFT で使用することはできますが，このカスタム・クラスではサポートされません。

- 実装をオーバーライドしたテスト・オブジェクト・メソッドまたは認識プロパティのカスタム・サポートを削除する場合は，カスタム・サポート・クラスから関連するサポート・メソッドを削除します。
- テスト・オブジェクト・クラスの定義からテスト・オブジェクト・メソッドまたは認識プロパティを削除する場合は，これらをテスト・オブジェクト設定ファイルから削除します。

カスタム・サポート・クラスのログとデバッグ

サポート・クラスを設計する際には，問題が発生した場合のデバッグに役立つように，ログ・ファイルへのメッセージの書き込みを追加することをお勧めします。

ログ・ファイルにメッセージを送信するには，**MicAPI.logLine** メソッドを使用します。詳細については，『UFT Java Add-in Extensibility API Reference』（『Java Add-in Extensibility SDK ヘルプ』で利用可能）を参照してください。

ログ・メッセージの出力を制御する（常時すべてのメッセージが出力されないようにする）には，各サポート・クラスにデバッグ・フラグを作成します。**MicAPI.logLine** を呼び出す際に，該当するデバッグ・フラグを最初の引数として指定します。**MicAPI.logLine** は，指定したデバッグ・フラグがオンの場合にのみ，ログ・メッセージを出力します。

次の例は，サポート・クラスでのログ・メッセージの出力方法を示しています。

```
private static final String DEBUG_ALLLIGHTSCS = "DEBUG_ALLLIGHTSCS";

public String light_on_positions_attr(Object obj) {
        AllLights lights = (AllLights) obj;
...
        for (int i = 0; i < 5; i++) {
           for (int j = 0; j < 5; j++) {
             if(lights.isSet(j, i)) {
                 MicAPI.logLine(DEBUG_ALLLIGHTSCS, "Light "+i+":"+j+" is set");
...
             }
        }
    }
}
```

UFT でログを制御するには，次の 2 つの行を含むテストを作成して実行します。テスト内では，有効にすべきフラグとメッセージの書き込み先のファイルが列挙します。

```
javautil.SetAUTVar "sections_to_debug", "DEBUG_ALLLIGHTSCS"
javautil.SetAUTVar "debug_file_name", "C:\JavaExtensibility\Javalog.txt"
```

同時に複数のフラグを有効にする場合は，次の例のように，すべてのフラグ文字列を 2 番目の引数に続けて入力します（スペースで区切ります）。

```
javautil.SetAUTVar "sections_to_debug", "DEBUG_ALLLIGHTSCS
DEBUG_AWTCALC"
```

MicAPI.logLine によって出力されるメッセージは，設定したフラグに応じて，サポート・クラスの実行時に指定されたファイルに出力されます。ログの出力を制御するフラグの変更や，出力先のファイルの変更を行う場合は，適切な引数を指定して UFT GUI テストを再度実行します。

カスタム・ツールキット・サポートのデバッグ

Java サポート・クラスは，テスト中のアプリケーションのコンテキストで実行されます。そのため，サポート・クラスをデバッグする場合は，アプリケーション自身をデバッグするのと同じ方法でデバッグを行うことができます。

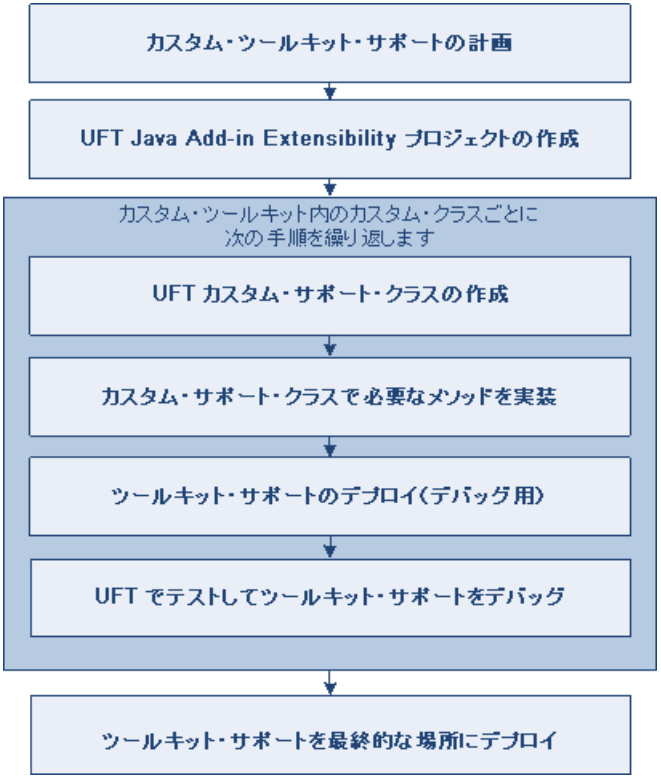
デバッグを開始するには，サポート・クラス内にブレークポイントを配置し，アプリケーションをデバッグする場合と同じようにアプリケーションを実行し，サポート・クラスのさまざまな部分に到達するように，アプリケーションに対して UFT の各種操作を実行します。

Eclipse でアプリケーション・コードを保存している場合は，Eclipse からデバッグ・モードでアプリケーションを実行できます（アプリケーション・ファイルを右クリックして，［**Debug As**］＞［**Java Applet**］（または［**Application**］）または［**Debug As**］＞［**SWT Application**］を選択）。

アプリケーション・コードを Eclipse で保存していない場合は，アプリケーションに対するリモート・デバッグを使用してサポート・クラスをデバッグします。リモート・デバッグの詳細については，『Eclipse ヘルプ』を参照してください。

Java Add-in Extensibility の実装のワークフロー

次のワークフローは，カスタム・ツールキットに対する UFT Java Add-in Extensibility サポートを作成する場合に実行する手順をまとめたものです。サポートするカスタム・ツールキットごとに，これらの手順を実行します。



* UFT Java Add-in Extensibility Eclipse プラグインのウィザードを使用すると，カスタム・ツールキット・サポート・プロジェクト，カスタム・クラス，必要なすべてのファイルを作成することができます。ウィザードを使用しない場合は，39 ページ「カスタム・ツールキット・サポート・セットの作成」の説明に従って，必要なパッケージとファイルを手動で作成する必要があります。また，その後カスタム・クラスを新しいテスト・オブジェクト・クラスにマッピングする場合は，テスト・オブジェクト設定ファイルで新しいテスト・オブジェクト・クラスを定義する必要があります。

第 4 章

カスタム・ツールキット・サポートの計画

カスタム・ツールキットのサポートの作成を始める前に，サポートを注意深く計画する必要があります。UFT にカスタム・コントロールをどのように認識させるかを詳細に計画し，カスタム・ツールキット・サポートの基本要素を正しく構築します。事前に詳細をすべて計画しておくことが重要です。後から変更を加えようとすると，複雑な手動での変更が必要になる可能性があります。また，カスタム・サポートの再作成が必要になることもあります。

注：本章は，第 3 章「カスタム・ツールキットのサポートの実装」に記載されている概念について理解していることが前提となっています。

本章の内容

カスタム・ツールキット・サポートの計画について

カスタム・ツールキット・サポートの作成では，詳細な計画を作成する必要があります。この作業に役立つように，この章の各項には，カスタム・ツールキットおよびそのコントロールのサポートの実装に関連するいくつかの確認事項が記されています。カスタム・ツールキット・サポートを作成する際には，これらの確認事項に基づいてサポートを実装します。

最初に，カスタム・ツールキットに関する一般的な情報を決定し，その後に，サポートする各カスタム・クラスに関する具体的な情報を定義します。

カスタム・ツールキット関連情報の決定

カスタム・ツールキットに関する詳細な計画を作成するために，次の質問に答えます。

➤ カスタム・ツールキットの名前は何か？

カスタム・ツールキットの一意の名前を指定します。サポートを作成して UFT にデプロイすると，アドインやサポートされている環境のリストを表示する UFT のすべてのダイアログ・ボックスにカスタム・ツールキット名が表示されます。たとえば，UFT が開いたときには，［アドイン マネージャ］ダイアログ・ボックスに Java Add-in の子としてカスタム・ツールキット名が表示され，UFT ユーザはそのツールキットのサポートをロードするかどうかを指定できます。

➤ カスタム・ツールキットに含まれるカスタム・クラスは何か？

カスタム・クラスの場所を列挙します。カスタム・クラスの場所には，Eclipse プロジェクト，Java アーカイブ・ファイル，またはクラス・フォルダを指定できます。

カスタム・クラスをサポートするツールキットにグループ化するルールについては，38 ページ「カスタム・ツールキットのサポートの作成の準備」を参照してください。

➤ カスタム・ツールキットが拡張するネイティブ・ツールキット（または，サポートされている既存のツールキット）は何か？

注：あるカスタム・ツールキット内のすべてのクラスが別のツールキットの基本ユーザ・インタフェース・クラス（たとえば，**java.awt.Component**）を拡張する場合，カスタム・ツールキットがツールキット（この例では，**AWT**）を拡張する，といいます。

➤ どのような順序でツールキット内の各種クラスのサポートを作成するか？

この確認事項については，56 ページ「サポート・クラスの継承階層の決定」を参照してください。

各カスタム・クラスに対するサポート情報の決定

カスタム・クラスのサポートの計画を始める前に，コントロールに対するフルアクセスを確保し，コントロールの動作を十分に理解しておく必要があります。アプリケーションでコントロールの実際の動作を確認でき，コントロールを実装するカスタム・クラスにアクセスできることが必要です。

UFT でカスタム・コントロールをサポートするのに，カスタム・コントロールのソースを変更する必要はありませんが，ソースの内容を確認する作業は必要になります。外部からアクセスできるメンバ（フィールドおよびメソッド）やリッスンできるイベントなどを把握しておく必要があります。

特定のクラスのカスタム・サポートを計画する際には，UFT でこのクラスのコントロールをどのように認識させるか（UFT GUI テストでコントロールを表すのに必要なテスト・オブジェクトのタイプ，使用する認識プロパティとテスト・オブジェクト・メソッドなど）を十分に検討する必要があります。このためには，カスタム・コントロールを含むアプリケーションを実行し，オブジェクト・スパイ，キーワード・ビュー，記録オプションを使用して UFT の観点からコントロールを分析するのが最適です。カスタム・サポートがない状態で UFT がコントロールを認識する方法を把握できるので，どのような変更が必要なのかを判断できます。

UFT を使用してカスタム・コントロールを分析する例については，22 ページ「サンプル・カスタム・コントロールに対する標準設定の UFT サポートと Extensibility オプションの分析」を参照してください。

カスタム・クラス・サポート計画のチェックリストについて

特定のクラスのカスタム・サポートを計画する際には，一連の確認事項について検討する必要があります。ここでは，これらの確認事項について説明します。また，確認事項を簡単にまとめた印刷形式のチェックリスト（98 ページ）も利用できます。

注 : 確認事項 1，4，5 は，カスタム・ツールキット・サポートの設計の基礎になります。これらの確認事項をサポートの作成後に変更すると，複雑な手動での変更が必要になる可能性があります。また，カスタム・サポートの再作成が必要になることもあります。

---

1 サポートする適切なカスタム・クラスを選択していることを確認します。

- **a** カスタム・クラスに UFT カスタム・サポートが提供されていないスーパークラスが存在するか？
- **b** カスタム・コントロールに，同じスーパークラスを拡張するほかのコントロールと同じ UFT サポートを必要とする認識プロパティまたはテスト・オブジェクト・メソッドが存在するか？

存在する場合には，最初にスーパークラスのサポートを作成することを検討します。

2 UFT がインストールされているコンピュータで，カスタム・クラスのソースとカスタム・コントロールを実行するアプリケーションにアクセスできることを確認します。

3 プログラミングを行うコンピュータ上でコンパイル済みカスタム・クラスにアクセスできることを確認します。これらのクラスは，クラス・フォルダ，Java アーカイブ，または Eclipse プロジェクトに存在します。

4 カスタム・コントロールを適切に表す既存の Java テスト・オブジェクト・クラスがあるか ?ある場合，それはどれか？

5 存在しない場合は，新しいテスト・オブジェクト・クラスを作成する必要があります。

a 拡張することでカスタム・コントロールを表すことができる既存の Java テスト・オブジェクトがあるか?ある場合，それはどれか?存在しない場合，新しいテスト・オブジェクト・クラスは JavaObject クラスを拡張する必要があります。

注 : ほかのツールキット・サポート・セットで定義されたテスト・オブジェクト・クラスを拡張するテスト・オブジェクト・クラスを作成した場合，2 つのツールキット・サポート・セットの間に依存関係が生じます。拡張している方のツールキット・サポート・セットを UFT のアドイン・マネージャにロードする場合は，拡張される方のツールキット・サポート・セットもロードするように選択する必要があります。

---

b 新しいテスト・オブジェクトに対して UFT で異なるアイコンを使用するか？
使用する場合，アイコン・ファイルは非圧縮の **.ico** 形式である必要があります。

c (コントロールのテスト・オブジェクト・クラスと完全修飾 Java クラス名に加えて) コントロールを一意に認識するのに使用できる 1 つまたは複数の認識プロパティを指定します。

d このクラスのオブジェクトに対するステップが生成されたときに，キーワード・ビューとステップ・ジェネレータに表示される標準のテスト・オブジェクト・メソッドを指定します。

6 UFT でカスタム・コントロールをトップレベルの Java テスト・オブジェクトとして認識させるか？

7 カスタム・コントロールに，このコントロールのコンテキストでのみ意味を持つオブジェクトが含まれているか？つまり，このカスタム・コントロールはラッパーか？(たとえば，Calculator オブジェクトは計算機ボタン・オブジェクトのラッパーです)。

8 コントロールを表すテスト・オブジェクトの名前を付けるための基礎を指定します。

9 サポートする認識プロパティを列挙します。

新しいテスト・オブジェクト・クラスを作成する場合は，UFT の［チェックポイントのプロパティ］ダイアログ・ボックスで標準で選択するプロパティを併せて決定します。

10 サポートするテスト・オブジェクト・メソッドを列挙します。メソッドの引数タイプと名前を指定し，メソッドがリターン・コードに加えて戻り値を返すかどうかを指定します。

11 カスタム・コントロールが AWT ベースである場合，記録オプションを使用して UFT GUI テストの作成のサポートを提供するか？

提供する場合，UFT 記録セッション中にカスタム・コントロールに対して記録するイベントのリストを指定します。

カスタム・クラス・サポート計画のチェックリスト

このチェックリストは，カスタム・クラス・ツールキット・サポートの計画で使用します。

☑	カスタム・クラス・サポート計画のチェックリスト
❑	カスタム・クラスに UFT カスタム・サポートが提供されていないスーパークラスが存在するか？ **はい/いいえ**
❑	存在する場合，階層内の上位のコントロールに対するサポートを最初に拡張すべきか？ **はい/いいえ**
❑	UFT がインストールされているコンピュータ上でカスタム・コントロールを実行するアプリケーションが存在するか？ **はい/いいえ**
❑	このカスタム・コントロール・クラスのソースが存在する場所：
❑	カスタム・コントロールに適合する既存の Java テスト・オブジェクトはどれか？
❑	存在しない場合に作成する新しい Java テスト・オブジェクト・クラス： ➤ 新しいテスト・オブジェクト・クラスによって拡張されるもの：（標準 JavaObject） ➤ アイコン・ファイルの場所（オプション）： ➤ 説明用の認識プロパティ： ➤ 標準設定のテスト・オブジェクト・メソッド：
❑	UFT でカスタム・コントロールをトップレベルの Java テスト・オブジェクトとして認識させるか？ **はい/いいえ**
❑	カスタム・コントロールはラッパーか？ **はい/いいえ**
❑	テスト・オブジェクトの名前付けの基礎を指定：
❑	サポートする認識プロパティを列挙し，標準設定のチェックポイントのプロパティをマークする：
❑	サポートするテスト・オブジェクト・メソッドを列挙する（必要に応じて，引数と戻り値も記入する）：
❑	記録をサポートするか？（AWT ベースのみ） **はい/いいえ**
❑	その場合，記録をトリガするイベントを列挙：

その他の情報

カスタム・ツールキット・サポートの計画が終了したら，計画に基づいて，カスタム・ツールキットをサポートするカスタム・ツールキット・サポート・セットを作成します。Eclipse の UFT Java Add-in Extensibility ウィザードを使用すると，必要なファイル，クラス，基本的なメソッドをすべて作成できます。また，ウィザードで作成されるメソッド・スタブを利用して，追加のメソッドを実装することもできます。詳細については，101 ページ「UFT Java Add-in Extensibility Eclipse プラグインの使用」を参照してください。

Eclipse で Java Add-in Extensibility ウィザードを使用しない場合は，第 3 章「カスタム・ツールキットのサポートの実装」の情報を使用して，手動でカスタム・ツールキットに対する完全なサポートを拡張することができます。

第 5 章

UFT Java Add-in Extensibility Eclipse プラグインの使用

UFT Java Add-in Extensibility SDK には，Eclipse の Java 開発環境用のプラグインが含まれています。このプラグインでは，カスタム・ツールキット・サポート・セットの作成に使用するウィザードと，作成後にファイルを編集するためのコマンドが利用できます。

Java Add-in Extensibility ウィザードを使用しない場合は，この章を省略することができます。この場合は，35 ページ「カスタム・ツールキットのサポートの実装」の説明に従って，カスタム・ツールキットのすべてのサポートを手動で拡張することができます。

本章の内容

UFT Java Add-in Extensibility Eclipse プラグインについて

UFT Java Add-in Extensibility SDK をインストールすると，UFT Java Add-in Extensibility プラグインが Eclipse に追加されます。このプラグインでは，カスタム・ツールキット・サポート・セットの作成に使用するウィザードと，作成後にファイルを編集するためのコマンドが利用できます。Java Add-in Extensibility SDK のインストールおよびアンインストールについては，27 ページ「HP UFT Java Add-in Extensibility Software Development Kit のインストール」を参照してください。

Eclipse で UFT Java Add-in Extensibility プラグインのウィザードを使用すると，カスタム・ツールキット・サポートを作成してデプロイすることができます。このウィザードでは，カスタム・クラスと必要なサポートについて指定する詳細情報に基づいて，必要なファイル，クラス，メソッドがすべて作成されます。また，ウィザードで作成されるメソッド・スタブを利用して，追加のメソッドを実装することもできます。

本章を利用するには，カスタム・ツールキット・サポートを構成する要素およびこのサポートを作成するワークフローについて説明した，本書の「カスタム・ツールキットのサポートの実装」の章（35 ページ）を読んでおく必要があります。

カスタム・ツールキットのサポートを作成する場合，最初に New UFT Java Add-in Extensibility Project ウィザードを使用して，カスタム・ツールキット・サポート用のパッケージとファイルを含む Eclipse プロジェクトを作成します。

続いて，New UFT Custom Support Class ウィザード（116 ページを参照）を使用して，関連するカスタム・クラスに対するサポート・クラスを作成します。カスタム静的テキスト・クラスに対するサポート・クラスを作成する場合は，New UFT Custom Static-Text Support Class ウィザード（148 ページを参照）を使用します。

ウィザードで指定内容に基づいたサポート・クラスが作成されたら，カスタム・サポートの設計を完成させる必要があります。これを行うには，カスタム・コントロールのニーズに合わせて，ウィザードで作成されたメソッド・スタブを実装します。

UFT Java Add-in Extensibility Eclipse プラグインには，設計中のサポートの編集や，デバッグ時の UFT へのデプロイに使用できるコマンドも用意されています。これらのコマンドについては，153 ページ「Eclipse での UFT コマンドの使用」を参照してください。

注意：ウィザードを使用する際には，ウィザードによって作成されるファイル名を変更したりファイルを削除したりしないでください。ウィザードは指定されたコマンドを実行する際に，ウィザードで作成された名前に基いてファイルを検索します。カスタム・ツールキット・サポート・セットが完成して最後にデプロイを行う際に，設定ファイルの名前を変更することができます。最後のデプロイ段階では，テスト・オブジェクト設定ファイルを複数のファイルに分割することもできます。カスタム・ツールキット・サポート・セットのファイルは，83 ページ「設計完了後のカスタム・サポートのデプロイ」に従って，適切なフォルダに配置してください。

New UFT Java Add-in Extensibility Project ウィザード

New UFT Java Add-in Extensibility Project ウィザードを使用して，特定のカスタム・ツールキット用のサポート・セットを構成するファイルを含むプロジェクトを Eclipse で新規に作成します。カスタム・ツールキットの詳細を指定すると，ウィザードによって必要なツールキット・サポート・ファイルが作成されます。

New UFT Java Add-in Extensibility Project の作成が済んだら，各カスタム・ツールキット・クラスのサポートを作成することができます。これを行うには，New UFT Custom Support Class ウィザード（116 ページを参照）（または，New UFT Custom Static-Text Support Class ウィザード（148 ページを参照））を使用します。

Eclipse で New UFT Java Add-in Extensibility Project ウィザードを起動するには，次の手順を実行します。

1 ［**File**］ ＞ ［**New**］ ＞ ［**Project**］を選択します。［New Project］ダイアログ・ボックスが開きます。

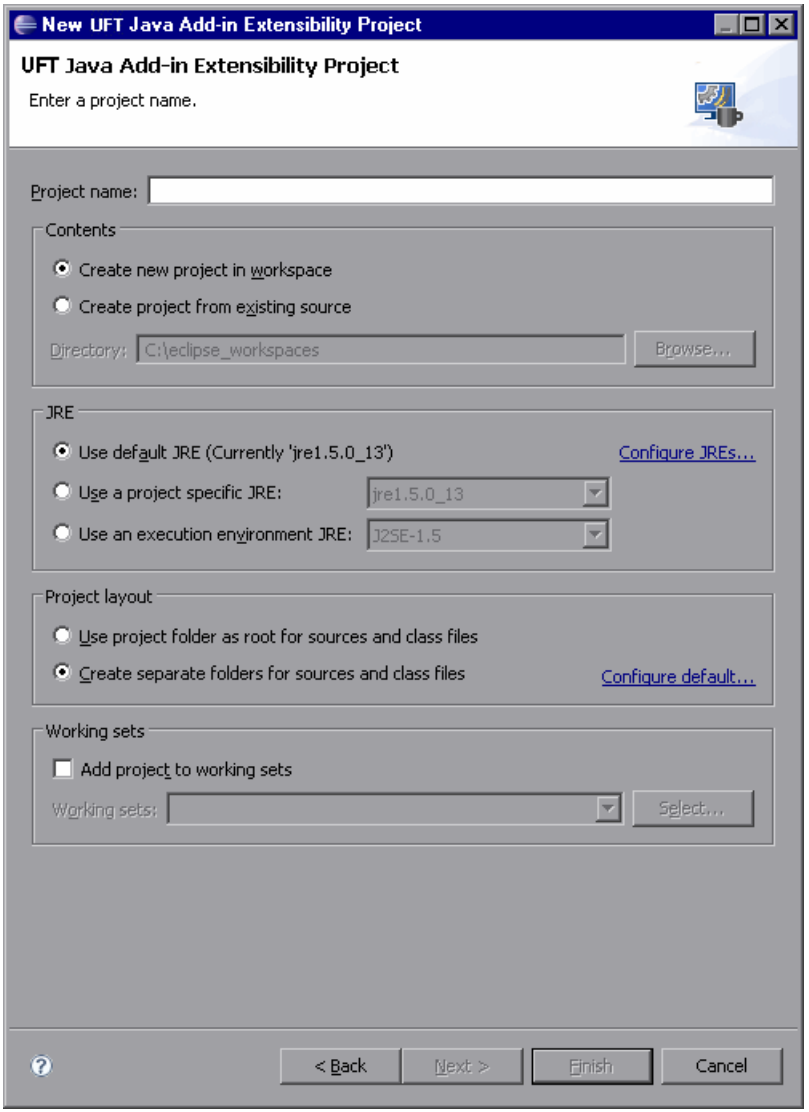
2 **Unified Functional Testing** フォルダを展開し，**UFT Java Add-in Extensibility Project** を選択します。

3 ［**Next**］をクリックします。New ［UFT Java Add-in Extensibility Project］画面（105ページ を参照）が開きます。

ヒント：Eclipse をカスタマイズして **UFT Java Add-in Extensibility Project** を［**New**］メニューのオプションにすると，この手順を短縮することができます。これを行うには，［**Window**］ ＞ ［**Customize Perspective**］を選択します。表示されるダイアログ・ボックスの［Shortcuts］タブで，［**Unified Functional Testing**］と［**UFT Java Add-in Extensibility Project**］のチェック・ボックスを選択します。［**OK**］をクリックします。

［UFT Java Add-in Extensibility Project］画面

［UFT Java Add-in Extensibility Project］画面で，UFT Java Add-in Extensibility プロジェクトを作成し，プロジェクトのレイアウトを定義します。この画面の詳細は，使用する Eclipse のバージョンによって異なることがあります。



次の手順を実行します。

1 ［**Project name**］ボックスに，プロジェクトの名前を入力します。

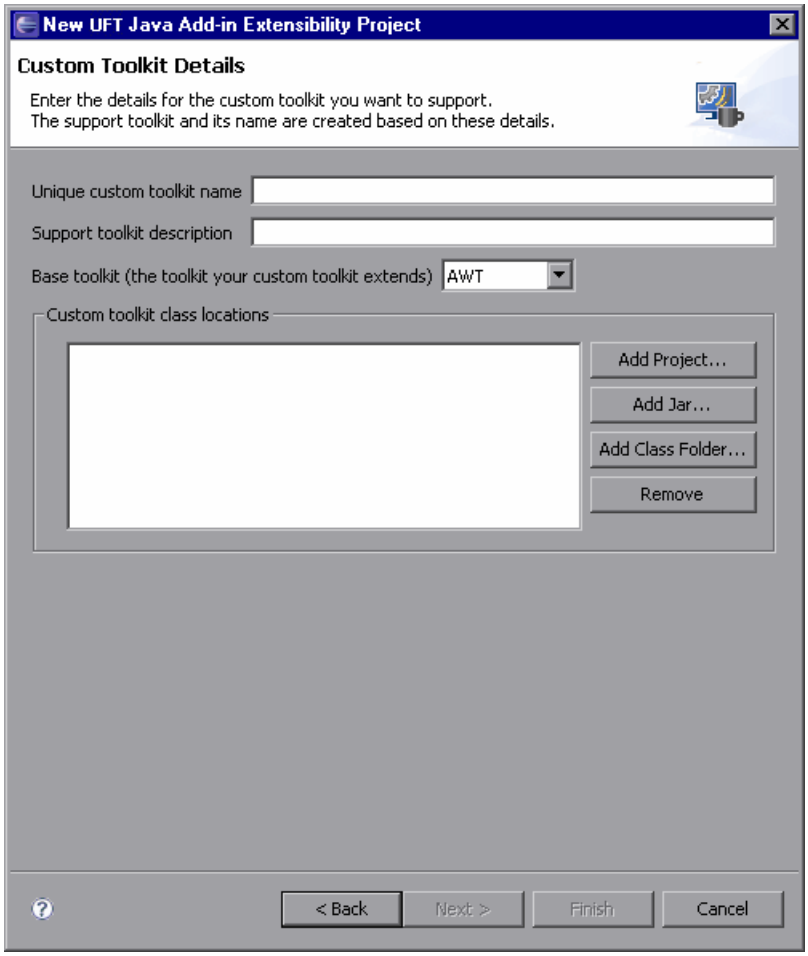
2 ［**Project Layout**］領域で，［**Create separate folders for sources and class files**］を選択します（以前のバージョンの Eclipse では，このオプションの表記は［**Create separate source and output folders**］となっています）。

3 ［**Next**］をクリックして［Custom Toolkit Details］画面（107ページ ページ）に進みます。

この Eclipse のウィザード画面で利用できるオプションについては，『Eclipse ヘルプ』を参照してください。

［Custom Toolkit Details］画面

［Custom Toolkit Details］画面では，対応するカスタム・ツールキット・サポート・セットをウィザードで生成できるように，カスタム・ツールキットの詳細を指定します。［**Finish**］をクリックすると，［Project Summary］画面（113ページ ページ）が開きます。



このウィザード画面では，次の詳細情報を指定します。

➤ **Unique custom toolkit name :** サポートの作成対象のカスタム・ツールキットを一意に表す名前。UFT のアドインやサポートされている環境を一覧表示するすべてのダイアログ・ボックスでこの名前が表示されます。一意のツールキット名を指定することで，1 つの UFT で多数のカスタム・ツールキット・サポート・セットを同時にサポートすることが可能になります。

名前には英数字とアンダースコアのみを使用し，最初の文字は必ず英文字にする必要があります。

ウィザードでは，新しいツールキット・サポート・セットを作成する際に，この名前を使用します。次に例を示します。

- ➤ ツールキット・サポート・クラスには，<custom toolkit name>Support という名前が付けられます。
- ➤ ツールキット設定ファイルには，<custom toolkit name>.xml という名前が付けられます（UFT のアドイン・マネージャやその他のダイアログ・ボックスに表示されるカスタム・ツールキット名は，このファイル名から来ています）。
- ➤ ウィザードでテスト・オブジェクト設定ファイルが作成される場合，**TypeInformation** 要素の **PackageName** 属性にカスタム・ツールキット名が入力されます。これにより，UFT で新しいテスト・オブジェクト・クラスを適切なカスタム・ツールキットに関連付けることができます。

サポートがすでに UFT にデプロイされているカスタム・ツールキットの名前を指定することはできません。ウィザードを使用して新しいプロジェクトを作成し，このプロジェクトを使用して既存のカスタム・ツールキット・サポートを置き換える場合は，最初に既存のサポートを手動で削除する必要があります。これを行うには，**<UFT インストール・フォルダ> bin\java\classes\extension** を参照し，ツールキット設定ファイルを削除してから，**Reload Support Configuration** コマンド（155 ページを参照）を使用します。

➤ **Support toolkit description :** サポート・ツールキットに関する説明文。この説明文はツールキット設定ファイルに保存されます。

➤ **Base toolkit :** カスタム・ツールキットによって拡張されるツールキット。カスタム・ツールキットのすべてのカスタム・コントロールがベース・ツールキットのコントロールを拡張する場合，ツールキットをカスタム・ツールキットのベース・ツールキットとみなすことができます。

［**Base toolkit**］リストには，UFT サポートがすでに存在しているツールキットのリストが表示されます。独自のツールキット用のサポートを作成してデプロイすると，そのツールキットもこのリストに表示されます。

ウィザードで新しいカスタム・ツールキット・サポート・セットを作成する際には，新しいツールキット・サポート・クラスが作成されます。この新しいツールキット・サポート・クラスは，選択したベース・ツールキットのツールキット・サポート・クラスを拡張します。そのため，新しいカスタム・ツールキット・サポートは，基本的な機能（たとえば，イベントの処理やディスパッチなど）に必要なユーティリティ・メソッドをベース・ツールキット・サポートからすべて継承します。

➤ **Custom toolkit class locations** : このプロジェクトでサポートするカスタム・クラスの場所のリスト。Eclipse プロジェクト，**.jar** ファイル，Java クラスのフォルダ（コンパイル済みの Java クラスを格納するファイル・システムのフォルダ）を指定することができます。

新しい Java Add-in Extensibility プロジェクトがビルドされると，これらの場所がプロジェクトのビルド・パスに追加されます。

ビルド・パスには，カスタム・クラスのすべての親クラスの場所も含める必要があります。SWT，AWT，または JFC（Swing）から直接派生したカスタム・クラスがなく，親クラスがカスタム・クラスと同じ場所に存在しない場合は，プロジェクトのビルド・パスにこれらの場所を手動で追加します。

注 :

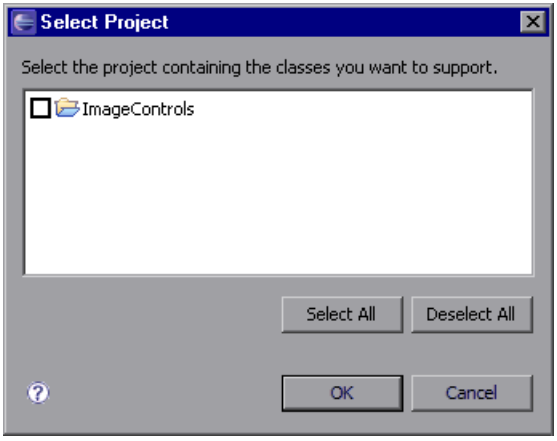
➤ New UFT Custom Support Class ウィザードの［Custom Class Selection］画面（117 ページを参照）には，このボックスに列挙した場所のカスタム・クラスが表示されます。このため，カスタム・サポート・クラスの作成時に必要なカスタム・クラスを選択することができます（新しい Java Add-in Extensibility プロジェクトをビルドした後でカスタム・サポート・クラスを作成します）。

➤ Java Add-in Extensibility プロジェクトの作成後に，プロジェクトのカスタム・クラスの場所を追加または削除するには，UFT Java Add-in Extensibility プロジェクトの［Properties］ダイアログ・ボックスを使用します（115 ページを参照）。

カスタム・ツールキット・クラスの場所をリストに追加するには，次の手順を実行します。

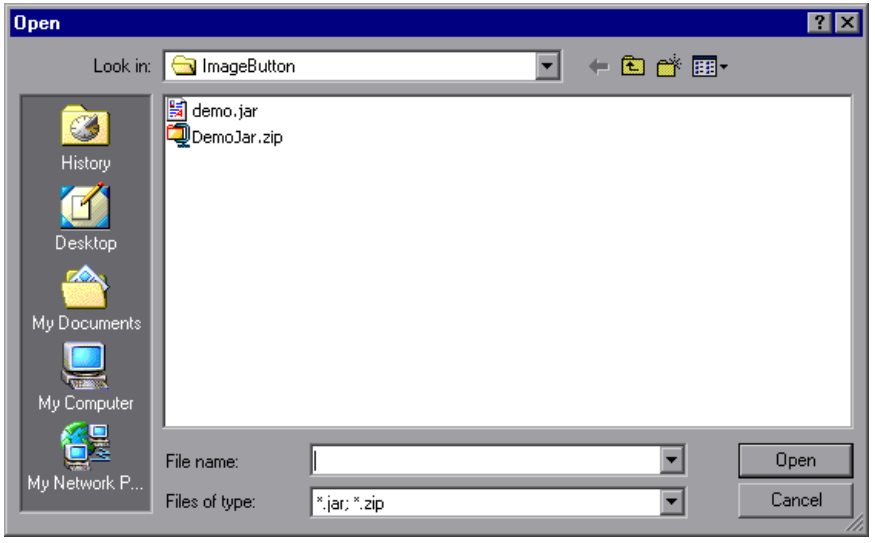
以下のオプションを 1 つ以上使用して，カスタム・ツールキット・クラスの場所を追加します。

➤ [**Add project**] をクリックして, Eclipse プロジェクトを選択します。[Select Project] ダイアログ・ボックスが開き，現在の Eclipse ワークスペースのプロジェクトが表示されます。



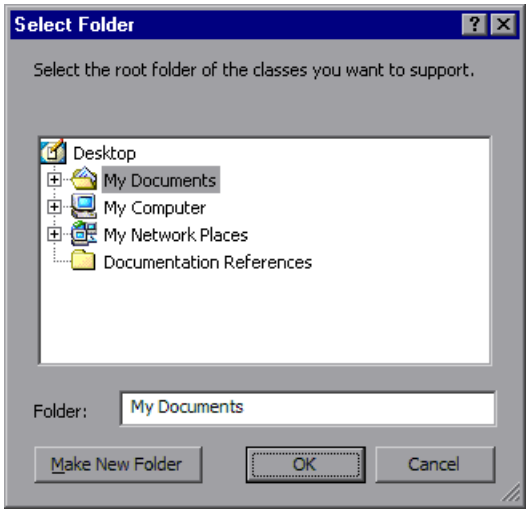
関連するプロジェクトのチェック・ボックスを選択し，[**OK**] をクリックして [**Custom toolkit class locations**] ボックスに追加します。

➤ [**Add Jar**] をクリックして，Java アーカイブ（**.jar**）ファイルを追加します。[Open] ダイアログ・ボックスが開きます。



関連する Java アーカイブ・ファイルを参照して選択し，[**OK**] をクリックして [**Custom toolkit class locations**] ボックスに追加します。

➤ [**Add Class Folder**] をクリックして，クラス・フォルダを追加します。[Select Folder] ダイアログ・ボックスが開きます。



関連するフォルダを参照して選択し，[**OK**]をクリックして[**Custom toolkit class locations**]ボックスに追加します。

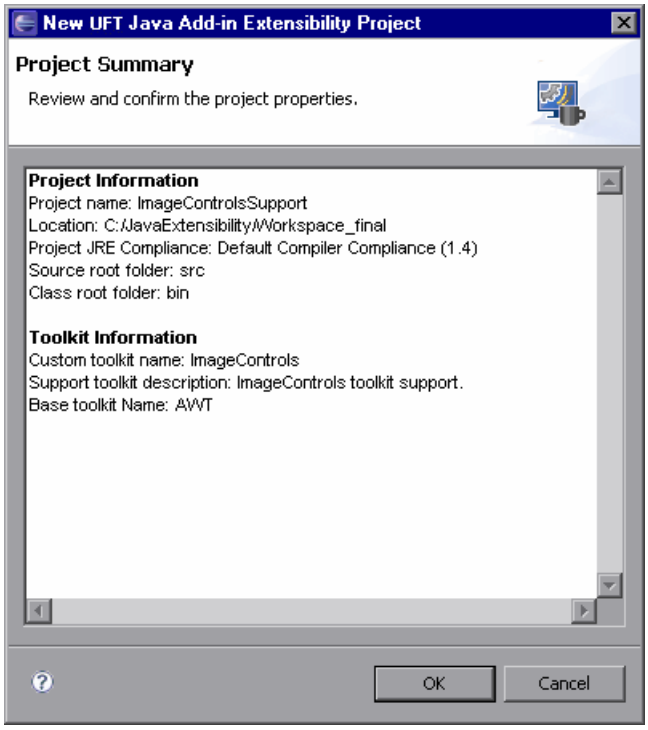
注：コンパイル済みのクラス・パッケージを含むルート・フォルダを選択します。たとえば，ファイル **ImageButton.java** では，クラス **com.demo.ImageButton** が定義されています。このクラスをコンパイルして結果を **bin** フォルダに保存した場合，クラス・ファイル **ImageButton.class** の場所は **bin\com\demo\ ImageButton.class** になります。[**Custom toolkit class locations**]でこのクラスの場所を選択する場合は，**bin** フォルダを選択します。

カスタム・ツールキット・クラスの場所をリストから削除するには，次の手順を実行します。

[**Custom toolkit class locations**]ボックスで削除する場所を選択し，[**Remove**]をクリックします。

［Project Summary］画面

カスタム・ツールキット・サポート・ファイルが作成される前に，ウィザードの［Project Summary］画面に Java Add-in Extensibility の新規プロジェクト用に指定した内容のサマリが表示されます。



表示された内容を確認します。変更が必要なデータがある場合は，［**Cancel**］をクリックして ［Custom Toolkit Details］画面（107 ページを参照）に戻ります。［**Back**］ボタンと［**Next**］ボタンを使用して，関連する画面を開き，必要な変更を行います。

完了したら，[**OK**] をクリックします。次の項目を含む UFT Java Add-in Extensibility の新規プロジェクトが作成されます。

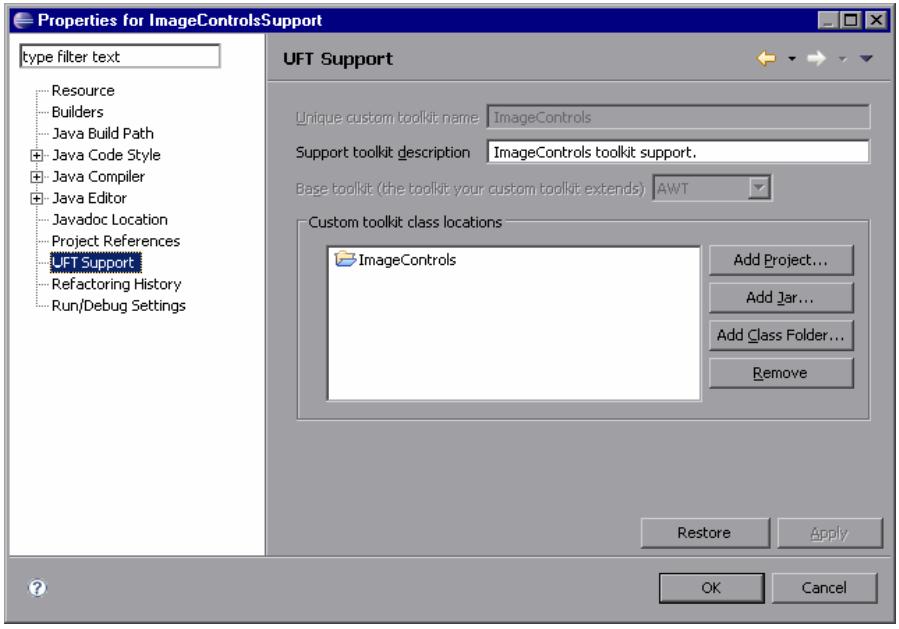
- ツールキットのルート・パッケージ : **com.mercury.ftjadin.qtsupport.<カスタム・ツールキット名>**。これには，次の内容が含まれます。
  - ツールキットのルート・パッケージのツールキット・サポート・クラス : **<カスタム・ツールキット名>Support.java**

    このクラスの内容については，43 ページ「ツールキット・サポート・クラスについて」を参照してください。
  - サポート・クラスのサブパッケージ : **com.mercury.ftjadin.qtsupport.<カスタム・ツールキット名>.cs**
- **Configuration** という名前の設定ファイル用のフォルダ。これには，次の内容が含まれます。
  - ツールキット設定ファイル **<カスタム・ツールキット名>.xml**。このファイルの内容については，44 ページ「ツールキット設定ファイルについて」を参照してください。
  - テスト・オブジェクト設定ファイル用の **TestObjects** フォルダ。

注 : お使いのコンピュータに複数の Java Runtime Environment（JRE）がインストールされていて，指定した 1 つ以上のカスタム・ツールキット・クラスの場所が Eclipse プロジェクトである場合には，カスタム・ツールキット・プロジェクトと新しい Java Add-in Extensibility プロジェクトが同じ JRE を使用していることを確認する必要があります。同じ JRE を使用していない場合は，すべてのプロジェクトが同じ JRE を使用するように，プロジェクトの JRE を変更してください。

UFT Java Add-in Extensibility プロジェクトのプロパティの変更

Eclipse のメニュー・バーで，［**Project**］＞［**Properties**］を選択します。［Properties］ダイアログ・ボックスが開きます。左側の表示枠で，プロパティ・タイプのリストから **UFT Support** を選択します（このリストの項目は，使用する Eclipse のバージョンによって異なる場合ことがあります)。右の表示枠に **UFT Support** のプロパティが表示されます。



このダイアログ・ボックスのオプションについては，107 ページ「［Custom Toolkit Details］画面」を参照してください。

Java Add-in Extensibility プロジェクトの作成後に，［**Unique custom toolkit name**］または［**Base toolkit**］を変更することはできません。

［**Support toolkit description**］を変更することは可能です。また，［**Custom toolkit class locations**］リストで場所の追加や削除を行うこともできます。このリストを変更する際には，それに合わせてプロジェクトのビルド・パスを変更する必要があります。

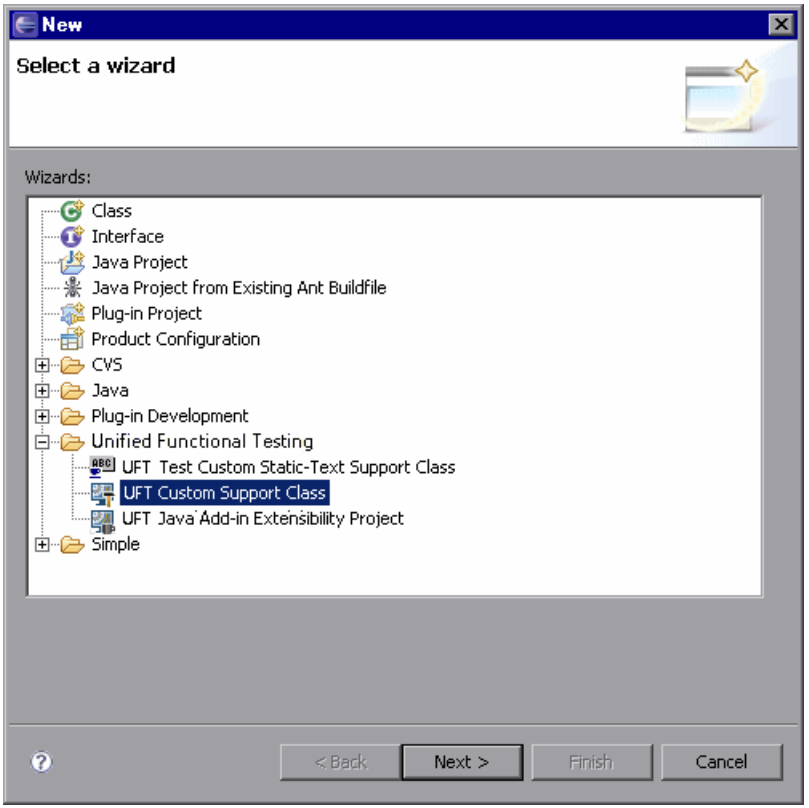
［**Restore**］ボタンをクリックすると，このダイアログ・ボックスの設定を最後に保存した値に戻すことができます。

New UFT Custom Support Class ウィザード

Java Add-in Extensibility プロジェクト内でサポート・クラスを作成する場合は，New UFT Custom Support Class ウィザードを使用します。ウィザードでカスタム・クラスと必要な UFT サポートの詳細を指定すると，それに合わせてサポート・クラスと必要なすべてのメソッドが作成されます。また，ウィザードで作成されるメソッド・スタブを利用して，追加のメソッドを実装することもできます。

Eclipse で New UFT Custom Support Class ウィザードを起動するには，次の手順を実行します。

1 ［Eclipse Package Explorer］タブで，UFT Java Add-in Extensibility プロジェクトを選択します。［**File**］＞［**New**］＞［**Other**］を選択します。［New］ダイアログ・ボックスが開きます。



2 **Unified Functional Testing** フォルダを展開し，**UFT Custom Support Class** を選択します。

3 ［**Next**］をクリックします。［Custom Class Selection］画面が開きます。

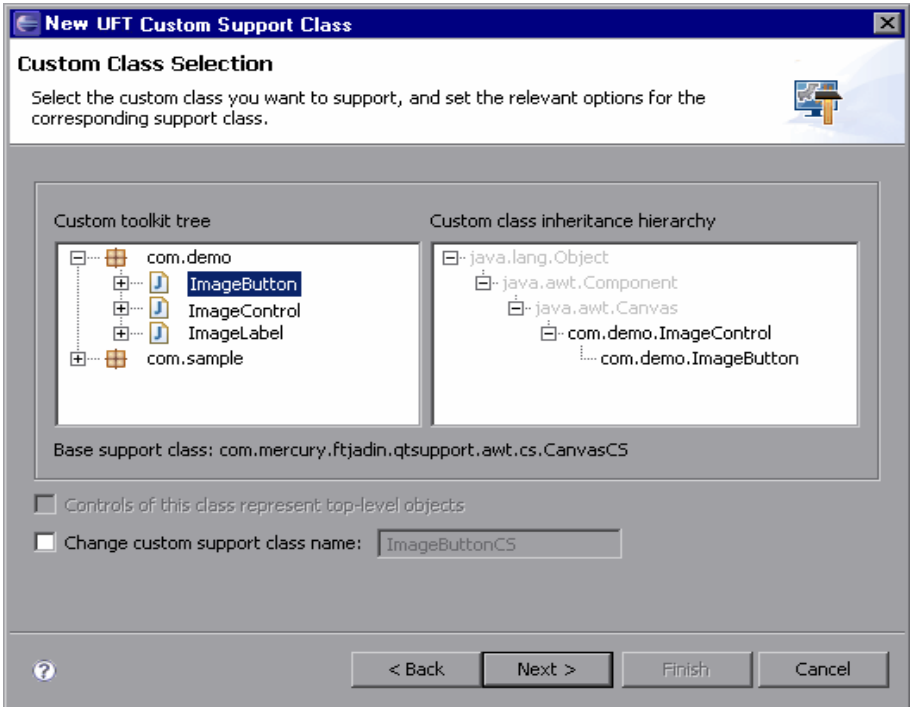
ヒント : Eclipse をカスタマイズして **UFT Custom Support Class** を［**New**］メニューのオプションにすると，この手順を短縮することができます。これを行うには，［**Window**］＞［**Customize Perspective**］を選択します。表示されるダイアログ・ボックスの［Shortcuts］タブで，［**Unified Functional Testing**］と［**UFT Custom Support Class**］のチェック・ボックスを選択します。［**OK**］をクリックします。

［Custom Class Selection］画面

［Custom Class Selection］画面は，New UFT Custom Support Class ウィザードの最初の画面です。この画面では，サポートするカスタム・クラスを選択し，関連するオプションを設定します。ウィザードは，カスタム・クラスの継承階層に基づいて，新しいサポート・クラスで拡張する必要のある既存のサポート・クラスを自動的に判別します。

［**Next**］をクリックすると，［Test Object Class Selection］画面（122ページ を参照）が開きます。

注 : サポートするクラスの選択は，カスタム・サポート・クラスの作成のベースとなります。後続の画面で変更を行った後に，この画面に戻って別のクラスを選択した場合，後続の画面で行った変更は破棄されます。



この画面のメイン領域には，次のオプションがあります。

➤ **Custom toolkit tree**：サポートの候補となるカスタム・ツールキット内のすべてのクラスが表示されます（New UFT Java Add-in Extensibility Project ウィザードで列挙したカスタム・ツールキット・クラスの場所から取得されます）。展開記号（+）と折りたたみ記号（-）を使用すると，ツリーの展開と折りたたみを行い，パッケージとクラスを表示することができます。

次の条件を満たすクラスのみが表示されます。

➤ **java.awt.Component** または **org.eclipse.swt.widgets.Widget** を拡張するクラス。

➤ UFT のサポートがまだ拡張されていないクラス。カスタム・クラスのサポートがすでに UFT にデプロイされている場合，または現在の Eclipse プロジェクトでカスタム・クラスのサポートを作成している場合，カスタム・クラスはこのツリーに表示されません。

注：上記の要件のすべて満たしているにも関わらず，ツリーに表示されないクラスがある場合は，**Reload Support Configuration** コマンド（155 ページを参照）を使用して環境を更新してください。

たとえば，Eclipse の Java Add-in Extensibility プロジェクトでカスタム・サポートを削除して，同じカスタム・コントロール用のサポートを新規に作成する場合は，サポートの設定を再ロードする必要があります。これにより，カスタム・クラスを **Custom toolkit tree** に表示することができます。

➤ **Custom class inheritance hierarchy** : **Custom toolkit tree** で選択したクラスの継承階層が表示されます。グレー表示のノードは，このツールキットに含まれないクラスを示しています。黒く表示されるノードは，カスタム・ツールキットに含まれるクラスを示しています。

［**Custom toolkit tree**］または［**Custom class inheritance hierarchy**］では，拡張するカスタム・クラスを選択できます（［**Custom class inheritance hierarchy**］では，黒く表示されるノードと，UFT サポートを持たないクラスのみを選択できます）。

➤ **Base support class :** 新しいサポート・クラスによって必ず拡張されるサポート・クラス。この情報を変更することはできません。ウィザードでは，UFT サポートを含む階層内の最も近い先祖のサポート・クラスが選択されます（カスタム・クラスのサポートがすでに UFT にデプロイされている場合，または現在の Eclipse プロジェクトでカスタム・クラスのサポートを作成している場合，ウィザードはこのカスタム・クラスを UFT サポートを含むものと認識します）。

UFT が特定のサポート・クラスにマッピングされていない Java オブジェクトを認識した場合，このオブジェクトの最も近い先祖にマッピングされているサポート・クラスが使用されます。そのため，ベース・サポート・クラスは，特定のサポート・クラスにマッピングされていない場合にカスタム・コントロールにサポートを提供するクラスになります。新しいカスタム・サポート・クラスでは，ベース・サポート・クラスで十分に提供されないサポートのみを実装（またはオーバーライド）する必要があります。

［**Custom class inheritance hierarchy**］および［**Base support class**］に表示される情報を使用すると，最初に別の（階層の上位の）カスタム・クラスのサポートを拡張すべきかどうかを判断することができます。判断する際には，次の点に注意してください。

➤ UFT サポートを持たない上位階層のカスタム・クラスが存在するかどうか。

➤ 存在する場合，そのカスタム・クラスに複数の子孫に対して同じ方法でサポートする必要のある要素が存在するかどうか。

上記の答えが「存在する」である場合には，上位クラスのサポートを最初に作成することを検討してください。この場合，そのサポート・クラスを［**Base support class**］として使用することが可能になります。クラスが階層内で黒く表示される場合には，この画面で選択し，このウィザード・セッションで対応するサポートを作成することができます。クラスがグレー表示される場合には，このツールキットに含まれないため，現在の UFT Java Add-in Extensibility プロジェクト内でサポートを作成することはできません。

上位クラスが現在のサポート・プロジェクトのベース・ツールキットを拡張する場合，これをカスタム・ツールキットに追加することによって，このプロジェクトのスコープに追加することができます。ベース・ツールキットについては，107 ページ「［Custom Toolkit Details］画面」を参照してください。既存のサポート・プロジェクトに対するカスタム・クラスの追加については，115 ページ「UFT Java Add-in Extensibility プロジェクトのプロパティの変更」を参照してください。

それ以外の場合（最初に上位クラスのサポートを作成してから，そのサポート・クラスをベース・サポート・クラスとして使用する場合）は，121ページに記載されている手順を実行する必要があります。

［Custom Class Selection］画面の下の部分には，次のオプションがあります。

➤ **Controls of this class represent top-level objects**：UFT でコントロールがテスト・オブジェクト階層の最上位の Java オブジェクトとして認識されると期待できるように指定することができます。詳細については，71 ページ「トップレベル・オブジェクトのサポート」を参照してください。

このチェック・ボックスを選択すると，ウィザードは新しいカスタム・サポート・クラスに **isWindow** メソッドを実装します。このメソッドは **true** を返します。

このオプションは，サポートするように選択したクラスがコンテナ・クラスである（つまり，**java.awt.container** または **org.eclipse.swt.widgets.Composite** を拡張する）場合にのみ使用できます。新しいサポート・クラスが **ShellCS**（SWT），**WindowCS**（AWT），**AppletCS**（AWT）のいずれかのサポート・クラスを拡張する場合，このチェック・ボックスは標準で選択されます。

➤ **Change custom support class name :** 必要に応じて，サポート・クラスのウィザードによる標準設定の名前を変更することができます。

標準設定で，サポート・クラスの名前は **<カスタム・クラス名>CS** になります。ほとんどの場合，標準設定の名前を変更する必要はありません。ただし，カスタム・ツールキットに異なる複数のパッケージのクラスが含まれている場合，同じ名前のカスタム・クラスが複数存在する可能性があります。この場合は，1 つのパッケージに格納できるようにカスタム・サポート・クラスに異なる名前を指定する必要があります。

カスタム・サポート・クラス名を変更するには，［**Change custom support class name**］チェック・ボックスを選択してから，新しい名前を入力します。

注: ［Custom Class Selection］画面のオプションは, New UFT Custom Static-Text Support Class ウィザードの［Custom Static-Text Class Selection］画面で利用できるオプションと同じです（148ページを参照）。

このカスタム・ツールキットに含まれない上位クラスに対するサポートを作成し，このサポートをベース・サポート・クラスとして使用するには，次の手順を実行します。

1 別の UFT Java Add-in Extensibility プロジェクトで上位クラスに対するサポートを作成します。

2 作成したサポートを UFT にデプロイします。

3 元の UFT Java Add-in Extensibility プロジェクトを再度開きます。［**UFT**］＞［**Reload Support Configuration**］を選択するか，［**Reload Support Configuration**］ボタンをクリックします。

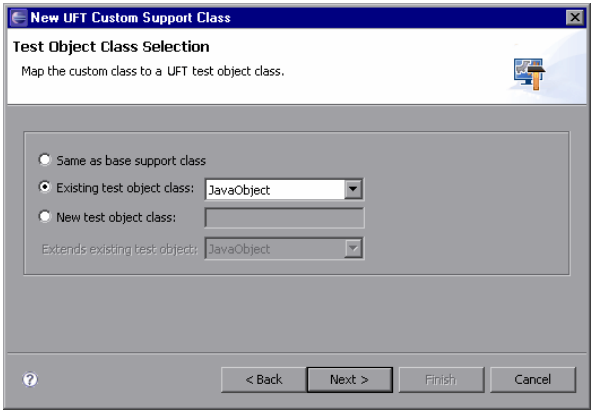
4 New UFT Custom Support Class ウィザード（116ページを参照）を開きます。これで，ウィザードで新しく作成したサポート・クラスが**ベース・サポート・クラス**として選択されるようになります。

［Test Object Class Selection］画面

［Test Object Class Selection］画面では，カスタム・クラスをテスト・オブジェクト・クラスにマッピングします。UFT GUI のテストで，カスタム・クラス・コントロールは，選択したテスト・オブジェクト・クラスのテスト・オブジェクトによって表されます。カスタム・サポート・クラスでは，この画面で選択するテスト・オブジェクト・クラスを返すように実装された **to_class_attr** プロパティ・メソッドが追加されます。これにより，サポート・クラスで，カスタム・クラスにマッピングされているテスト・オブジェクト・クラスを UFT に通知することが可能になります。

［**Next**］をクリックすると，［Custom Support Test Object Identification Properties］画面（125 ページ を参照）が開きます。

注：カスタム・クラスにマッピングするテスト・オブジェクト・クラスの選択は，カスタム・サポート・クラスの作成のベースとなります。後続の画面で変更を行った後に，この画面に戻って別のテスト・オブジェクト・クラスを選択した場合，後続の画面で行った変更は破棄されます。



このウィザード画面では，次のいずれかのオプションを選択します。

➤ **Same as base support class**：ベース・サポート・クラスの **to_class_attr** プロパティ・メソッドによって返されるテスト・オブジェクト・クラスにカスタム・クラスをマッピングします（このオプションを選択した場合，作成される新しいサポート・クラスに **to_class_attr** メソッドは追加されません。新しいサポート・クラスはベース・サポート・クラスのメソッドを継承します）。

［Custom Class Selection］画面（117 ページを参照）では，新しいサポート・クラスによって張されるサポート・クラスであるベース・サポート・クラスを決めました。ベース・サポート・クラスによってサポートされるカスタム・クラスは，特定のテスト・オブジェクト・クラスにマッピングされます。このテスト・オブジェクト・クラスがカスタム・クラスに対する論理テスト・オブジェクトでもある場合には，［**Same as base support class**］オプションを選択します。

どのような場合に［**Same as base support class**］オプションを選択するかについては，次の例を参照してください。

- ベース・サポート・クラスでサポートされるものに類似したカスタム・コントロールをサポートする場合。コントロールが同じ認識プロパティとテスト・オブジェクト・メソッドを持ち，これらのプロパティとメソッドが異なる形で実装されている場合，これらのコントロールは類似であるとみなされます。この場合，サポート対象のカスタム・コントロールには，ベース・サポート・クラスの **to_class_attr** プロパティ・メソッドによって返されるテスト・オブジェクト・クラスが適しています。
- 実際のコントロールをサポートするためではなく，ほかのサポート・クラスで拡張するサポート・クラスを作成する場合。この場合，どのテスト・オブジェクト・クラスをカスタム・クラスにマッピングするかは重要でないため，このオプションを選択することができます。このような例については，229 ページ「ImageControl カスタム・クラスのサポートの作成」を参照してください。

- **Existing test object class**：UFT ですでにサポートされている既存のテスト・オブジェクト・クラスにカスタム・クラスをマッピングすることができます。このリストには，UFT でサポートされているすべての Java オブジェクト・タイプが含まれています。カスタム・サポートに対する新しいテスト・オブジェクト・クラスを定義すると，このテスト・オブジェクト・クラスもリストに追加されます。

  Eclipse の現在のワークスペースで新しいテスト・オブジェクト・クラスを定義した場合，このリストに直ちに表示されます。それ以外の場合，新しいテスト・オブジェクト・クラスは，UFT にデプロイして設定を再ロードした後にリストに表示されます（詳細については，155 ページ「Reload Support Configuration」を参照）。

  プロジェクト内で定義されていないテスト・オブジェクト・クラスを選択する場合は，サポートが適切に機能するように，そのテスト・オブジェクト・クラスの定義も UFT にデプロイする必要があります。

この［**Existing test object class**］オプションを選択する際には，適切な既存のテスト・オブジェクト・クラスをリストから選択する必要があります。

ヒント：このオプションは，このテスト・オブジェクト・クラスにカスタム・コントロールの認識プロパティとテスト・オブジェクト・メソッドがすべて含まれている場合にのみ選択してください。プロパティやメソッドの追加が必要な場合は，［**New test object class**］を選択します。

➤ **New test object class：**作成する新しいテスト・オブジェクト・クラスにカスタム・コントロールをマッピングすることができます。このオプションは，カスタム・コントロールのすべての認識プロパティとテスト・オブジェクト・メソッドを含む既存のテスト・オブジェクト・クラスが存在しない場合に選択してください。このオプションを選択した場合は，新しいテスト・オブジェクト・クラスの名前を入力します。テスト・オブジェクト・クラス名には英数字とアンダースコアのみを使用し，最初の文字は必ず英文字にする必要があります。

このオプションを選択すると，［**Extends existing test object**］オプションが有効になります。

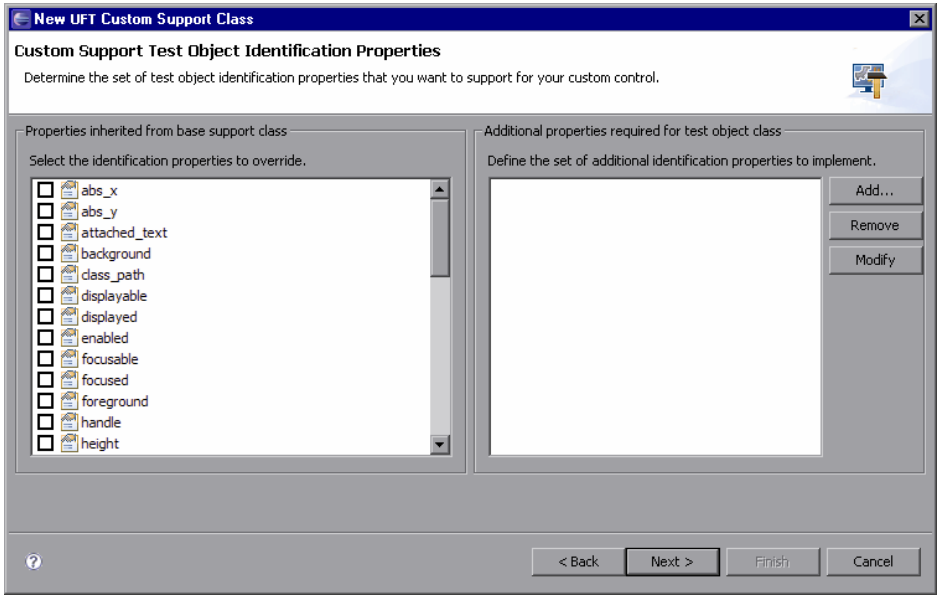
Extends existing test object：新しいテスト・オブジェクト・クラスは，既存のテスト・オブジェクト・クラスをベースに，認識プロパティとテスト・オブジェクト・メソッドのセットを拡張します。すべてのテスト・オブジェクト・クラスは，JavaObject クラスを拡張します。ここでは，特定の既存のテスト・オブジェクト・クラスをリストから選択することができます。このリストは，［**Existing test object class**］オプションで選択できる既存のテスト・オブジェクト・クラスのリストと同じです。

［**New test object class**］オプションを選択する場合は，［New Test Object Class Details］画面（144 ページを参照）で新しいテスト・オブジェクト・クラスに関する詳細を追加で定義します。新しいテスト・オブジェクト・クラスの定義はテスト・オブジェクト設定ファイルに追加されます。このファイルの構造と内容については，『UFT Test Object Schema ヘルプ』（『Java Add-in Extensibility SDK ヘルプ』で利用可能）を参照してください。

［Custom Support Test Object Identification Properties］画面

［Custom Support Test Object Identification Properties］画面には，拡張しようとしているベース・サポート・クラスでサポートされている認識プロパティと，選択したテスト・オブジェクト・クラスで定義されていて，まだサポートされていない追加プロパティが表示されます。ここでは，サポートを実装または新しい機能でオーバーライドするプロパティを選択することができます。また，新しいプロパティを追加することもできます。

［**Next**］をクリックすると，［Custom Support Test Object Methods］画面（129 ページ を参照）が開きます。



ベース・サポート・クラスから継承されるプロパティ

左側の表示枠には，ベース・サポート・クラスによって実装されるすべての認識プロパティが表示されます。これらの認識プロパティは，作成中のサポート・クラスによって継承されます。任意の認識プロパティを選択し，それぞれのサポートを異なる実装でオーバーライドすることができます。

注：これらの認識プロパティの中には，テスト・オブジェクト・クラスの定義に含まれないものがあります。そのため，これらは，UFT の［オブジェクト スパイ］ダイアログ・ボックスや［チェックポイントのプロパティ］ダイアログ・ボックスには表示されません。これらの認識プロパティにアクセスするには，**GetROProperty** メソッドを使用します。**GetROProperty** メソッドの詳細については，『HP Unified Functional Testing Object Model Reference』を参照してください。

このウィザードでは，サポート・クラス・ファイルの作成時に，選択する認識プロパティごとに **<認識プロパティ名>_attr** という名前のサポート・メソッド・スタブが追加されます。サポート・メソッド・スタブは，ベース・サポート・クラスのサポート・メソッドと同じ値を返します。カスタム・コントロールのニーズに合わせて，新しいサポート・メソッドを実装することができます。

テスト・オブジェクト・クラスに必要な追加のプロパティ

右側の表示枠には，選択したテスト・オブジェクト・クラスで定義されているが，ベース・サポート・クラスではサポートされていない認識プロパティが表示されます。このリストは，［**Add**］，［**Remove**］，［**Modify**］ボタンを使用して変更することができます。

この表示枠の認識プロパティごとに，ウィザードで作成されるサポート・クラスにサポート・メソッド・スタブが追加されます。カスタム・コントロールのニーズに合わせて実装するまでの間，これらのサポート・メソッド・スタブは **null** を返します。

このリストに認識プロパティを追加すると，テスト・オブジェクト設定ファイルのテスト・オブジェクト・クラスの定義にこれらの認識プロパティが追加されます。このファイルの構造と内容については,『UFT Test Object Schema ヘルプ』(『Java Add-in Extensibility SDK ヘルプ』で利用可能）を参照してください。

注 :［Test Object Class Selection］画面（122 ページを参照）で［**Same as base support class**］オプションを選択した場合，ウィザードはどのテスト・オブジェクト・クラスがカスタム・コントロールにマッピングされているかを認識しません。その結果，認識プロパティは右側の表示枠に表示されません。認識プロパティを追加すると，ウィザードで作成されるサポート・クラスに適切なサポート・メソッド・スタブが追加されます。ただし，この認識プロパティは，いずれのテスト・オブジェクト・クラスの定義にも追加されません。

プロパティの追加または削除を行う際の考慮事項

➤ このリストに追加した認識プロパティは，テスト・オブジェクト・クラスの定義に追加されます。これは，このクラスのすべてのテスト・オブジェクトに対し，UFT の認識プロパティのリストに新しいプロパティが追加されることを意味します。

そのため，プロパティを追加しようと考えている場合は，既存のテスト・オブジェクト・クラスを使用する代わりに，既存のテスト・オブジェクト・クラスをベースに新しいテスト・オブジェクト・クラスを作成することをお勧めします。

➤ リストから認識プロパティを削除した場合, 削除した認識プロパティはこのカスタム・クラスに対してサポートされなくなります。ただし，テスト・オブジェクト・クラスの定義には含まれたままになります。そのため，このプロパティは UFT オブジェクト・スパイに表示される認識プロパティのリストには引き続き表示されますが，値はありません。

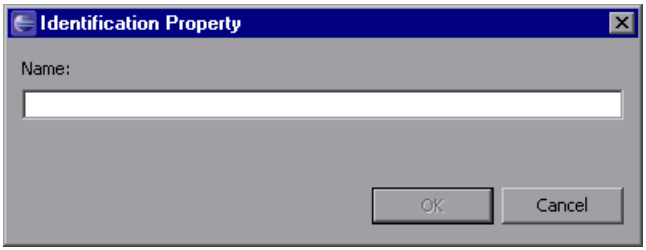
➤ 認識プロパティを変更することは，認識プロパティを削除して新しい認識プロパティを追加するのと同じです。

認識プロパティのリストの管理

下記の手順では，テスト・オブジェクト・クラスに必要な追加プロパティのリストで，認識プロパティの追加，削除，変更を行う方法について説明します。

認識プロパティを追加するには，次の手順を実行します。

1 ［**Additional properties required for test object class**］表示枠で，［**Add**］をクリックします。［Identification Property］ダイアログ・ボックスが開きます。



2 新しい認識プロパティの名前を入力し，［**OK**］をクリックします。(認識プロパティ名には英数字とアンダースコアのみを使用し，最初の文字は必ず英文字にする必要があります)。

認識プロパティを削除するには，次の手順を実行します。

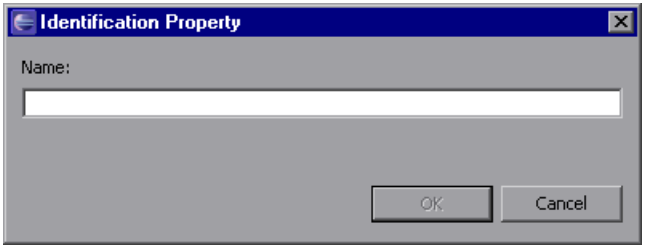
1 ［**Additional properties required for test object class**］表示枠で，削除するプロパティを選択します。

2 ［**Remove**］をクリックし，［**Yes**］をクリックして確定します。

認識プロパティを変更するには，次の手順を実行します。

1 ［**Additional properties required for test object class**］表示枠で，名前を変更するプロパティを選択します。

2 ［**Modify**］をクリックします。［Identification Property］ダイアログ・ボックスが開きます。



3 認識プロパティの名前を変更し，［**OK**］をクリックします。

ヒント：サポート・クラスが作成された後で認識プロパティを追加する場合は，［**Add Identification Property**］ボタンを使用するか，Eclipse で［**UFT**］＞［**Add Identification Property**］を選択します。

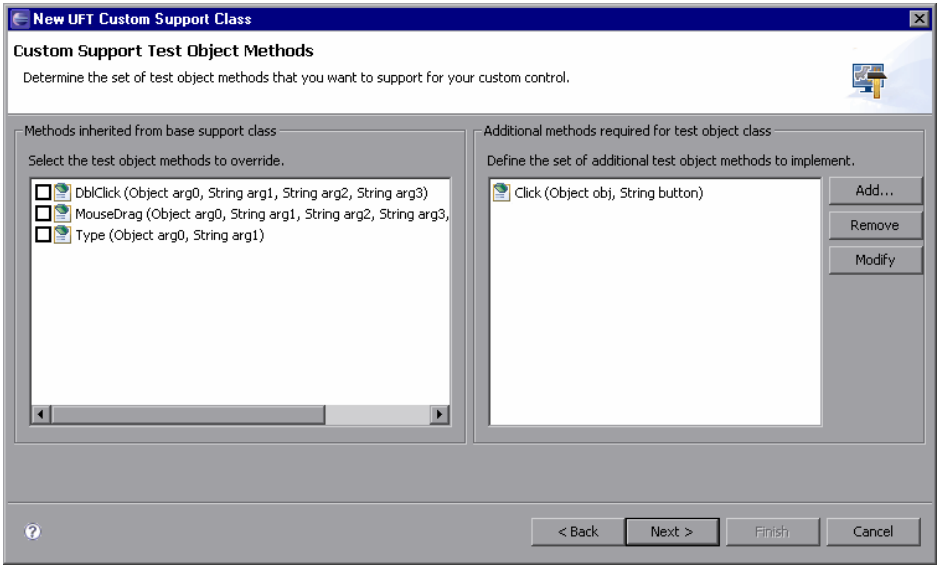
［Custom Support Test Object Methods］画面

［Custom Support Test Object Methods］画面には，カスタム・コントロールにマッピングしたテスト・オブジェクト・クラスに対して定義したテスト・オブジェクト・メソッドが表示されます。この画面では，サポートを実装または新しい機能でオーバーライドするテスト・オブジェクト・メソッドの選択，および新しいテスト・オブジェクト・メソッドの追加を行います。

［**Next**］をクリックすると，次のいずれかの画面が開きます。

- AWT ベースのカスタム・コントロールのサポートを作成している場合は，［Custom Control Recording Support］画面（138ページ を参照）が開きます。
- SWT ベースのカスタム・コントロールのサポートを作成していて，カスタム・コントロールに新しいテスト・オブジェクト・クラスをマッピングした場合は，［New Test Object Class Details］画面（144ページ を参照）が開きます。

➤ 前述のいずれの条件にも当てはまらない場合は，［Custom Control Support Class Summary］画面（146ページ を参照）が開きます。



ベース・サポート・クラスから継承されるメソッド

左側の表示枠には，選択したテスト・オブジェクト・クラスで定義され，ベース・サポート・クラスによって実装されるすべてのテスト・オブジェクト・メソッドが表示されます。これらのテスト・オブジェクト・メソッドは，作成中のサポート・クラスによって継承されます。任意のテスト・オブジェクト・メソッドを選択し，それぞれのサポートを異なる実装でオーバーライドすることができます。

このウィザードでは，サポート・クラス・ファイルの作成時に，選択したテスト・オブジェクト・メソッドごとに **<テスト・オブジェクト・メソッド名>_replayMethod** という名前のサポート・メソッド・スタブが追加されます。サポート・メソッド・スタブは，ベース・サポート・クラスのサポート・メソッドと同じ値を返します。カスタム・コントロールのニーズに合わせて，新しいサポート・メソッドを実装することができます。

注：［Test Object Class Selection］画面（122 ページを参照）で［**Same as base support class**］オプションを選択した場合，ウィザードはどのテスト・オブジェクト・クラスがカスタム・コントロールにマッピングされているかを認識しません。その結果，左側の表示枠にテスト・オブジェクト・メソッドは表示されません。新しいサポート・クラスが作成された後で，ベース・サポート・クラスから継承する再生メソッドをオーバーライドするには，これらをクラスに手動で追加します。

テスト・オブジェクト・クラスに必要な追加のメソッド

右側の表示枠には，選択したテスト・オブジェクト・クラスで定義されていて，ベース・サポート・クラスではサポートされていないテスト・オブジェクト・メソッドが表示されます。

この表示枠のリストは，［**Add**］，［**Remove**］，［**Modify**］ボタンを使用して変更することができます。

メソッドの名前を変更することは，メソッドを削除して新しいメソッドを追加するのと同じであることに注意してください。詳細については，133 ページ「［Test Object Method］ダイアログ・ボックスについて」を参照してください。

ヒント：サポート・クラスが作成された後でテスト・オブジェクト・メソッドを追加する場合は，［**Add Test Object Method**］ボタンを使用するか，Eclipse で［**UFT**］＞［**Add Test Object Method**］を選択します。

この表示枠のテスト・オブジェクト・メソッドごとに，ウィザードで作成されるサポート・クラスにサポート・メソッド・スタブが追加されます。カスタム・コントロールのニーズに合わせて実装するまでの間，これらのサポート・メソッド・スタブはエラー値 **Retval.NOT_IMPLEMENTED** を返します。

このリストにテスト・オブジェクト・メソッドを追加すると，ウィザードはテスト・オブジェクト設定ファイルのテスト・オブジェクト・クラスの定義にこれらのテスト・オブジェクト・メソッドを追加します。このファイルの構造と内容については，『UFT Test Object Schema ヘルプ』（『Java Add-in Extensibility SDK ヘルプ』で利用可能）を参照してください。

注：［Test Object Class Selection］画面（122 ページを参照）で［**Same as base support class**］オプションを選択した場合，ウィザードはどのテスト・オブジェクト・クラスがカスタム・コントロールにマッピングされているかを認識しません。その結果，右側の表示枠にテスト・オブジェクト・メソッドは表示されません。テスト・オブジェクト・メソッドを追加すると，作成されるサポート・クラスに適切な再生メソッド・スタブが追加されます。ただし，このテスト・オブジェクト・メソッドは，いずれのテスト・オブジェクト・クラスの定義にも追加されません。

テスト・オブジェクト・メソッドの追加または削除を行う際の考慮事項

- このリストに追加したテスト・オブジェクト・メソッドは，既存のテスト・オブジェクト・クラスに追加されます。これは，このクラスのすべてのテスト・オブジェクトに対し，これらのオブジェクトでサポートされるかどうかに関係なく，UFT に新しいメソッドが追加されることを意味します。UFT GUI のテストでは，オブジェクトに対するテスト・オブジェクト・メソッドを呼び出して，そのメソッドがサポートされていない場合，実行時エラーが発生します。

  そのため，カスタム・コントロールをサポートするためにテスト・オブジェクト・メソッドを追加しようと考えている場合は，既存のテスト・オブジェクト・クラスを使用する代わりに，既存のテスト・オブジェクト・クラスをベースに新しいテスト・オブジェクト・クラスを作成することをお勧めします。

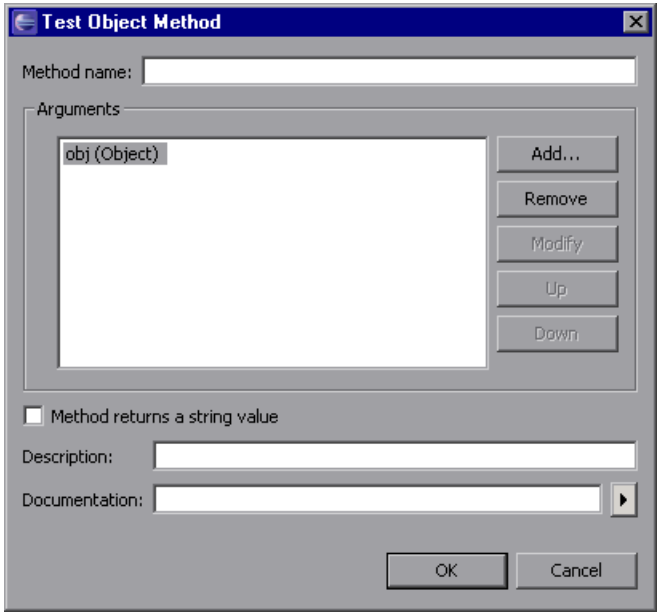
- このリストからテスト・オブジェクト・メソッドを削除した場合，削除したテスト・オブジェクト・メソッドはこのカスタム・クラスに対してサポートされなくなります。ただし，テスト・オブジェクト・クラスの定義には含まれたままになります。そのため，UFT のテスト・オブジェクト・メソッドのリストには引き続き表示されます。

  UFT GUI のテストでカスタム・コントロール上でこのテスト・オブジェクト・メソッドを使用した場合，実行時エラーが発生します。たとえば，**drop-down-list** コントロールは **List** テスト・オブジェクトとしてサポートされますが，**drop-down-list** コントロールに対して **select_range** テスト・オブジェクト・メソッドを選択し，このテスト・オブジェクト・メソッドがサポートされていない場合，実行時エラーが発生します。

［Test Object Method］ダイアログ・ボックスについて

［Custom Support Test Object Methods］画面（129 ページを参照）で［**Add**］または［**Modify**］をクリックすると，［Test Object Method］ダイアログ・ボックスが開きます。

［Test Object Method］ダイアログ・ボックスでは，［Custom Support Test Object Methods］画面の［**Additional methods required for test object class**］表示枠に表示されるテスト・オブジェクト・メソッドの詳細を指定することができます。



［Test Object Method］ダイアログ・ボックスには，次の項目があります。

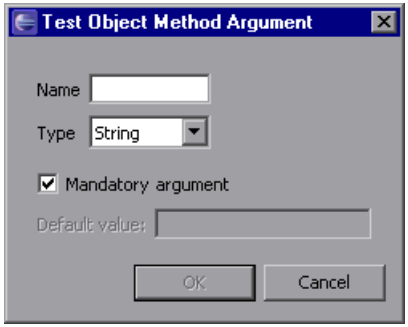
オプション	**説明**
Method name	UFT GUI のテストで表示されるテスト・オブジェクト・メソッドの名前。テスト・オブジェクト・メソッドの名前は，ユーザがステップ・ジェネレータやキーワード・ビューで選択できるように，テスト・オブジェクト・メソッドの機能がわかるような名前にしてください。メソッドの名前に英字以外の文字を含めることはできません。メソッド名の最初の文字は必ず英文字にする必要があります。また，スペースや次の記号を含めることはできません： ! @ # $ % ^ & * ( ) + = [ ] \ { } | ; ': " , / < > ? **注：** ➤ サポート・クラス内にすでに存在しているテスト・オブジェクト・メソッドの名前は使用しないでください（［Custom Support Test Object Methods］画面には，サポート・クラス内にすでに存在しているテスト・オブジェクト・メソッドが表示されます）。既存のテスト・オブジェクト・メソッドの実装をオーバーライドする場合は，（同じ名前の新しいテスト・オブジェクト・メソッドを作成するのではなく）［Custom Support Test Object Methods］画面の左側の表示枠で既存のテスト・オブジェクト・メソッドを選択してください。 ➤ メソッドの名前を変更することは，メソッドを削除して新しいメソッドを追加するのと同じです。

オプション	説明
Arguments	テスト・オブジェクト・メソッドの引数とそのタイプのリストです。 リストを変更する場合は，次のボタンを使用します。 ➤ **Add：**テスト・オブジェクト・メソッドに引数を追加することができます。詳細については，下記の"テスト・オブジェクト・メソッドの引数の追加または変更,"を参照してください。 ➤ **Remove：**選択した引数をリストから削除します。 ➤ **Modify：**テスト・オブジェクト・メソッドの引数を変更することができます。詳細については，下記の"テスト・オブジェクト・メソッドの引数の追加または変更,"を参照してください。 ➤ **Up：**選択した引数を上方向に移動します。 ➤ **Down：**選択した引数を下方向に移動します。 **注：** ➤ どのテスト・オブジェクト・メソッドでも，最初の引数は **obj (Object)** になります。この引数を削除，変更，または移動することはできません。 ➤［Test Object Class Selection］画面（122 ページを参照）で選択した既存のテスト・オブジェクト・クラスに属するテスト・オブジェクト・メソッドの署名を変更することはできません（つまり，既存のテスト・オブジェクト・メソッドで，引数の追加や削除，または引数のタイプの変更を行うことはできません）。

オプション	説明
Method returns a string value	このテスト・オブジェクト・メソッドは，リターン・コードに加えて，文字列値を返します（戻り値を取得して UFT GUI テストの後続のステップで使用することができます。） このチェック・ボックスを選択した場合，次のようになります。 ➤ テスト・オブジェクト設定ファイルに作成されるテスト・オブジェクト・メソッドの定義に，**ReturnValueType** 要素が追加されます。 ➤ 新しいサポート・クラス内に作成されるメソッド・スタブは，リターン・コード **OK** と空の文字列を含むオブジェクト **Retval("")** を返します。 このテスト・オブジェクト・メソッドに対して再生メソッドを実装する際には，さまざまな種類の **Retval** を使用することができます。メソッドが成功すると，**OK** と該当する文字列値が返されます。それ以外の場合は，関連するエラー・コードのみが返されます。詳細については，『UFT Java Add-in Extensibility API Reference』（『Java Add-in Extensibility SDK ヘルプ』で利用可能)を参照してください。
Description	ステップ・ジェネレータやキーワード・ビューでテスト・オブジェクト・メソッド上にカーソルを置いたとき，およびエディタでステートメント補完機能を使用しているときに表示されるツールヒントです。
Documentation	テスト・オブジェクト・メソッドを含むステップが実際に行う内容に関する説明文です。この説明文は，ステップ・ジェネレータの［**ステップについてのコメント**］ボックスおよびキーワード・ビューの［**注釈**］カラムに表示されます。 ▸ をクリックして関連する引数を選択すると，［**Documentation**］テキストに引数を挿入することができます。その後，引数は関連する値に動的に置き換えられます。

テスト・オブジェクト・メソッドの引数の追加または変更

［Test Object Method］ダイアログ・ボックスで［**Add**］または［**Modify**］をクリックすると，［Test Object Method Argument］ダイアログ・ボックスが開きます。［Test Object Method Argument］ダイアログ・ボックスでは，［Test Object Method］ダイアログ・ボックスに表示される各引数の詳細を指定することができます。



［Test Object Method Argument］ダイアログ・ボックスには，次の項目があります。

オプション	説明
Name	UFT GUI のテストで表示される引数の名前。引数の名前には，どのような値を入力する必要があるかが明確にわかる名前を付けてください。引数の名前には英数字のみ使用できます。引数名の最初の文字は必ず英文字にする必要があります。また，スペースや次の記号を含めることはできません： ! @ # $ % ^ & * ( ) + = [ ] \ { } | ; ' : " , / < > ?

オプション	説明
Type	UFT で次のいずれかを行うように設定します。 ➤ このテスト・オブジェクト・メソッドを含むテスト・ステップで，この引数に対して **String** 型の値を要求します。 ➤ **Variant** 型の値を許可します。 ［**Type**］を **Variant** に定義した場合でも，すべての引数が再生メソッドに文字列として渡されるわけではありません。また，テスト・ステップを記録する場合，引数は常に文字列として登録されます。 **注 :** 引数に対する使用可能な値のリストを定義する場合は，手動で設定を行う必要があります。テスト・オブジェクト設定ファイルで，値のリストを定義し，引数のタイプを **ListOfValues** に変更します。 詳細については，『UFT Test Object Schema ヘルプ』（『Java Add-in Extensibility SDK ヘルプ』で使用可能）を参照してください。
Mandatory argument	UFT でテストの記述者に引数の値の指定を要求するかどうかを設定します。 引数のリストで，必須引数の後にオプション引数を続けることはできません。
Default value	引数がオプションの場合，ほかの値が定義されていない場合に UFT で使用する標準設定値を指定することができます。 このオプションは，必須引数では使用できません。

［Custom Control Recording Support］画面

注 : SWT ベースのカスタム・クラスに対するサポート・クラスを作成する場合，［Custom Control Recording Support］画面は開きません。

カスタム・コントロール上で記録をサポートするには，サポート・クラスで記録をトリガするイベントのリスナを実装する必要があります。

［Custom Control Recording Support］画面には，拡張用に選択したサポート・クラスによって実装されるイベント・ハンドラ・メソッドが表示されます。

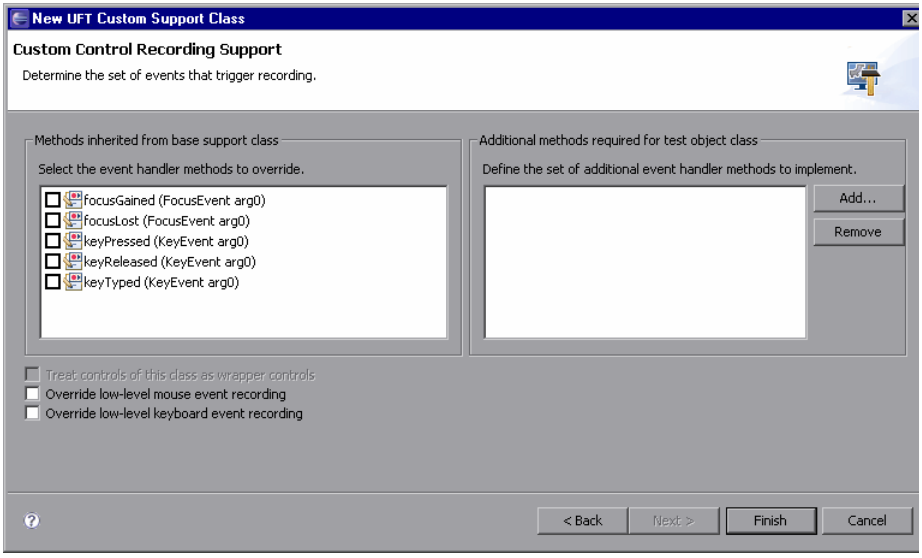
［Custom Control Recording Support］画面では，次の操作を行うことができます。

- 新しい機能で実装をオーバーライドするメソッドの選択
- 実装する新しいイベント・リスナの追加
- 記録に関連するオプションの設定

この画面で指定する詳細がどのように実装されるかについては，142 ページ「ウィザードによってサポート・クラスに追加される内容について」を参照してください。

記録に関連するサポート情報の設定が完了したら，次の操作を行います。

- 新しいテスト・オブジェクト・クラスをカスタム・コントロールにマッピングした場合は，［**Next**］をクリックして［New Test Object Class Details］画面（144 ページ を参照）に進みます。
- それ以外の場合は，［**Finish**］をクリックして［Custom Control Support Class Summary］画面（146ページ を参照）に進みます。



ベース・サポート・クラスから継承されるメソッド

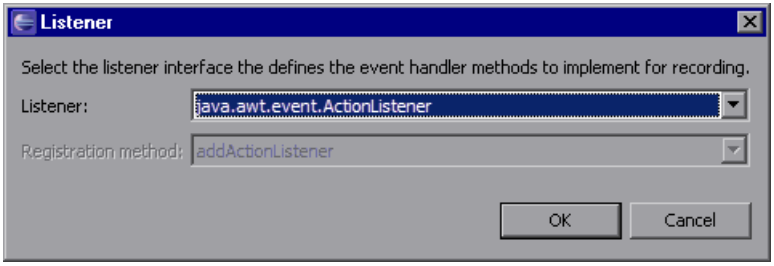
左側の表示枠には，ベース・サポート・クラスによって実装されるイベント・ハンドラ・メソッドが表示されます。オーバーライドするメソッドを選択することができます。

テスト・オブジェクト・クラスに必要な追加のメソッド

右側の表示枠では，新しいサポート・クラスに追加するリスナを指定します。選択した各リスナは，実装できるイベント・ハンドラ・メソッドのセットを含んでいます。

リストにリスナを追加するには，次の手順を実行します。

1 ［**Add**］をクリックし，表示される［Listener］ダイアログ・ボックスから適切なリスナを選択します。



このリストには，カスタム・コントロール上で登録できるリスナが表示されます。ウィザードは，カスタム・クラスとそのスーパークラスのリスナ登録メソッドを識別することによって，このリストを作成します。ウィザードで登録メソッドとして識別されるのは，最初の引数が **java.util.EventListener** を拡張する **add<XXX>Listener** という名前のメソッドのみです。

カスタム・クラスがこの定義と異なる登録メソッドを使用する場合，ウィザードを使用して対応するリスナを追加することはできません。ウィザードで新しいカスタム・サポート・クラスが作成された後で，必要なサポートを手動で実装することができます。

2 選択したリスナに複数の登録メソッドがある場合には，［**Registration method**］リストからメソッドを 1 つ選択します。

3 ［**OK**］をクリックします。選択したリスナと，それに含まれるすべてのイベント・ハンドラ・メソッドがリストに追加されます。

リストからリスナを削除するには，次の手順を実行します。

リスナまたはいずれかのイベント・ハンドラ・メソッドを選択して，［**Remove**］をクリックします。

ヒント : サポート・クラスが作成された後でイベント・ハンドラを追加する場合は，［**Add Event Handler**］ボタンを使用するか，Eclipse で［**UFT**］＞［**Add Event Handler**］を選択します。

［Custom Control Recording Support］画面のオプション

［Custom Control Recording Support］画面には，次のオプションがあります。

オプション	説明
Treat controls of this class as wrapper controls	新しいサポート・クラスで **com.mercury.ftjadin.infra.abstr.RecordWrapper** インタフェースを実装するように設定します。 カスタム・コントロールが **java.awt.container** を拡張する場合，このチェック・ボックスは標準で選択されています。それ以外の場合，このチェック・ボックスは使用できません。 詳細については，143 ページ「サポート・クラスでのラッパーの実装」を参照してください。
Override low-level mouse event recording	新しいサポート・クラスで **null** を返すように **mouseRecordTarget** メソッドを実装します。 このオプションを選択した場合，UFT で低レベル・マウス・イベント（座標ベースの操作）は記録されないため，メニューのオプション選択などの，より複雑な操作を記録することができます。

オプション	説明
Override low-level keyboard event recording	新しいサポート・クラスで **null** を返すように **keyboardRecordTarget** メソッドを実装します。 このオプションを選択した場合，UFT で低レベル・キーボード・イベントは記録されないため，エディット・ボックスでの値の設定などの，より複雑なイベントを記録することができます。

上記の表のオプションは，ウィザードでのみ使用できます（作成後にサポート・クラスを編集するための Eclipse の UFT コマンドでは使用できません）。サポート・クラスを作成する際にこれらのオプションを選択せず，後からこれらを実装する必要がある場合には，手動で設定を行う必要があります。

ウィザードによってサポート・クラスに追加される内容について

次の各項では，［Custom Control Recording Support］画面での指定内容に基いて，サポート・クラスに追加されるメソッドについて説明します。

サポート・クラスでのリスナの実装

ウィザードで作成されるサポート・クラス・ファイルでは，指定したリスナとオプションが，次のように実装されます。

- 実装されたリスナのインタフェースがサポート・クラスの署名に追加されます。
- コンストラクタがサポート・クラスに追加され，カスタム・コントロール上で登録が必要なすべてのリスナが列挙されます。また，リスナの追加と削除に使用するメソッドも列挙されます。このために，各リスナに対して **addSimpleListener** が呼び出されます。
- メソッド・スタブが左側の表示枠で選択した各イベント・ハンドラ・メソッドのサポート・クラスに追加されます。このメソッド・スタブは，ベース・サポート・クラスの対応するイベント・ハンドラ・メソッドを呼び出します。カスタム・コントロールのニーズに合わせて，新しいイベント・ハンドラ・メソッドを実装することができます。

一部のイベント・ハンドラ・メソッドは，既存のサポート・クラス内で final メソッドとして実装されます。final メソッドはオーバーライドすることができません。左側の表示枠でこれらのメソッドのいずれかを選択すると，作成されるメソッド・スタブのメソッド名の先頭にアンダースコアが追加されます。たとえば，**focusGained**，**focusLost**，**keyTyped**，**keyPressed**，または **keyReleased** を選択すると，それぞれ，**_focusGained**，**_focusLost**，**_keyTyped**，**_keyPressed**，または **_keyReleased** が作成されます。個々の final メソッドは，それぞれの基本的な機能の実行後に **_<メソッド名>** を呼び出すように実装されます。そのため，**_<メソッド名>** メソッドをオーバーライドすると，これらの final メソッドに機能を追加することができます。

- メソッド・スタブが右側の表示枠に表示される各イベント・ハンドラのサポート・クラスに追加されます。**MicAPI.record** を呼び出すようにイベント・ハンドラ・メソッドを実装する必要があります（各メソッド・スタブには，実装が必要であることを示すコメントと，推奨されるメソッドの基本構造が含まれています）。詳細については，68 ページ「記録オプションのサポート」を参照してください。

サポート・クラスでのラッパーの実装

コンテナ・コントロール内のコントロールをグループ化して単一のコントロールとして表すコンテナ・コントロールのサポートを作成する場合は，[**Treat controls of this class as wrapper controls**] チェック・ボックスを選択します。このチェック・ボックスを選択すると，サポート・クラスに次のメソッドが追加されます。

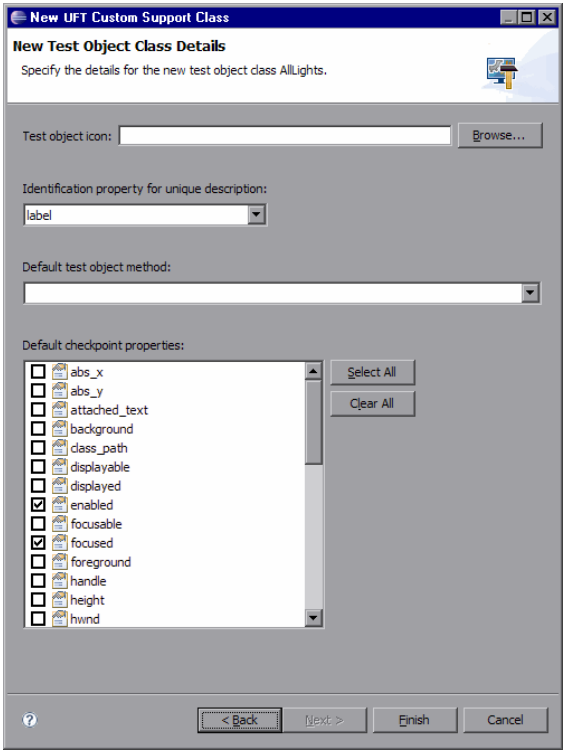
- **blockWrappedObjectRecord**（**False** を返します。）
- **registerWrapperInspector**（このクラスを特定のコントロール・タイプのラッパーとして登録するには，このメソッドを実装する必要があることを示すコメントが追加されます。）
- **checkWrappedObject**（**null** を返します。）
- **wrapperRecordMessage**（介入を行うことなく，ラップされたコントロールによって送信された記録メッセージを返します。）

これらのメソッドを実装すると，必要なラッピング機能を実現することができます。詳細については，71 ページ「ラッパー・コントロールのサポート」を参照してください。

［New Test Object Class Details］画面

新しいテスト・オブジェクト・クラスをカスタム・コントロールにマッピングした場合は，［New Test Object Class Details］画面で新しいテスト・オブジェクト・クラスに関する詳細を定義します。

［**Finish**］をクリックすると，［Custom Control Support Class Summary］画面（146 ページを参照）が開きます。



［New Test Object Class Details］画面には，次のオプションがあります。

オプション	説明
Test object icon	このテスト・オブジェクト・クラスのキーワード・ビューで使用するアイコン・ファイルのパスです。アイコン・ファイルは非圧縮の **.ico** 形式でなければなりません。 これは省略可能です。アイコン・ファイルを定義しない場合は，JavaObject のアイコンが使用されます。
Identification property for unique description	（**toolkit_class** プロパティと **index** プロパティに加えて）UFT でコントロールを一意に認識するための認識プロパティを指定します。 リストから認識プロパティを選択するか，標準設定で選択されたプロパティをそのまま使用します。
Default test object method	このクラスのオブジェクトに対するステップが生成されたときに，キーワード・ビューとステップ・ジェネレータに表示される標準のテスト・オブジェクト・メソッドを指定します。 リストからテスト・オブジェクト・メソッドを選択します。
Default checkpoint properties	このクラスのオブジェクトに対するチェックポイントを作成する際に，標準で選択される認識プロパティを指定します。 該当するプロパティのチェック・ボックスを選択します。すべてのチェック・ボックスを選択または選択解除する場合は，［**Select All**］または［**Clear All**］をクリックします。

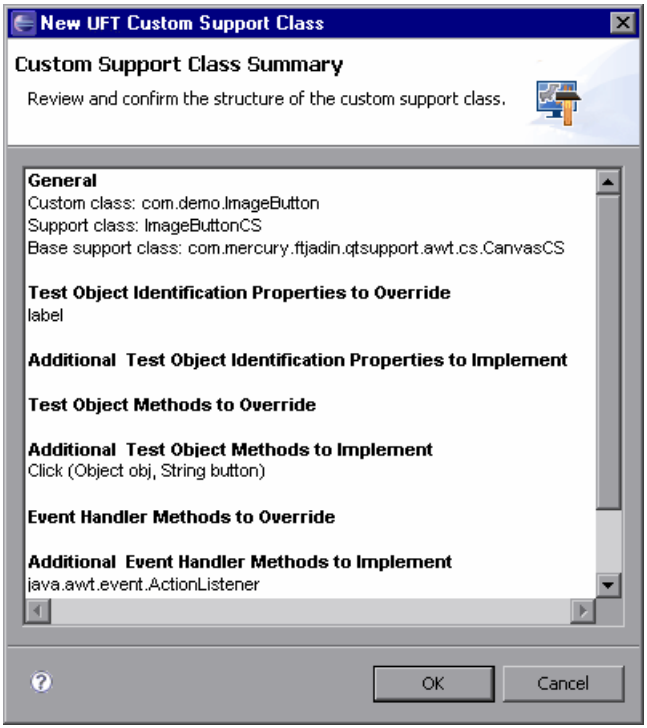
ウィザードでは，新しいサポート・クラスを作成する際に，新しいテスト・オブジェクト・タイプがテスト・オブジェクト設定ファイルに追加されます。［New Test Object Class Details］画面で指定するオプションは，このファイル内に記録されます。このファイルの構造については，『UFT Test Object Schema ヘルプ』（『Java Add-in Extensibility SDK ヘルプ』で利用可能）を参照してください。

UFT でテスト・オブジェクト記述に認識プロパティを追加する場合は，テスト・オブジェクト設定ファイルでこれを手動で指定する必要があります。テスト・オブジェクト・クラスの定義はテスト・オブジェクト設定ファイルに追加されます。テスト・オブジェクト記述に追加するプロパティごとに，そのプロパティに関して記述したファイル内の行を特定します。Property と Name の間に，ForDescription="true" を追加します。

テスト・オブジェクト記述内の認識プロパティのリストは，［オブジェクトの認識］ダイアログ・ボックスを使用して UFT で変更することができます。そのため，標準設定では，UFT でユーザが行う変更内容を上書きしないように，UFT はテスト・オブジェクト設定ファイルから一度だけこの情報を読み取ります。UFT で ForDescription 属性に対して行う変更を確実に読み取る方法については，85 ページ「テスト・オブジェクト設定ファイルで指定された認識プロパティ属性の変更」を参照してください。

［Custom Control Support Class Summary］画面

カスタム・サポート・クラス・ファイルが作成される前に，［Custom Support Class Summary］画面に新しいサポート・クラス用に指定した内容のサマリが表示されます。



変更が必要なデータがある場合は，［**Cancel**］をクリックしてウィザードの前の画面に戻ります。［**Back**］ボタンと［**Next**］ボタンを使用して，関連する画面を開き，必要な変更を行います。

完了したら，［**OK**］をクリックします。指定内容に従って，必要なすべてのメソッドを含む新しいサポート・クラスが作成されます。

また，次のいずれかの条件を満たす場合には，テスト・オブジェクト・クラスの定義がテスト・オブジェクト設定ファイルに追加されます。

- 新しいテスト･オブジェクト･クラスをカスタム･コントロールにマッピングした場合。
- 既存のテスト・オブジェクト・クラスに認識プロパティやテスト・オブジェクト・メソッドを追加した場合。

注：テスト・オブジェクト設定ファイルが存在しない場合は，この時点でテスト・オブジェクト設定ファイルが作成されます。テスト・オブジェクト設定ファイルの構造については，『UFT Test Object Schema ヘルプ』（『Java Add-in Extensibility SDK ヘルプ』で利用可能）を参照してください。

カスタム・クラス・サポートの完成

（New UFT Custom Support Class ウィザードを使用して）カスタム・サポート・クラスの作成が完了したら，次の追加手順を実行する必要があります。

- クラスを保存します。

  Eclipse では，右側の表示枠のタブに新しいクラス・ファイルが開いて表示されます。クラスを保存していないと，サポート・クラス・ファイル名の横のタブにアスタリスク（*****）が表示されます。ウィザードで行われる変更は相互に依存するため，必ず変更を保存して不一致が生じないようにする必要があります。

- ウィザードで新しいカスタム・サポート・クラス内に作成されたメソッド・スタブを実装します。詳細については，43 ページ「ツールキット・サポート・クラスについて」を参照してください。

  テスト・オブジェクト・クラスに新しいテスト・オブジェクト・メソッドや認識プロパティを追加した場合，テスト・オブジェクト設定ファイル内のテスト・オブジェクト・クラスの定義にこれらが追加されます。

ウィザードによってサポート・クラス内に作成されたサポート・メソッドを削除すると（実装しないと），テスト・オブジェクト・メソッドや認識プロパティがテスト・オブジェクト・クラスの定義に含まれたままになります。これらは，テストを編集する際に UFT で使用することはできますが，このカスタム・クラスではサポートされません。

➤ サポートが使用可能になるように，ツールキット・サポートを UFT にデプロイします。詳細については，78 ページ「カスタム・ツールキット・サポートのデプロイと実行」を参照してください。

New UFT Custom Static-Text Support Class ウィザード

Java Add-in Extensibility プロジェクト内でカスタム静的テキスト・クラスのサポート・クラスを作成する場合は，New UFT Custom Static-Text Support Class ウィザードを使用します。静的テキスト・クラスをサポートすると，UFT でその **label** プロパティを隣接するコントロールの**添付テキスト**として使用することができます。

このウィザードで指定する必要があるのは，静的テキスト・クラスとしてサポートするカスタム・クラスだけです（該当する場合は，［**Controls of this class represent top-level objects**］を選択する必要があります）。このウィザードでは，静的テキスト・オブジェクトのサポートに必要なメソッドを使用して，新しいサポート・クラスが作成されます。これらのメソッドについては，151 ページ「［Custom Static-Text Support Class Summary］画面」を参照してください。

新しいサポート・クラスが作成されたら，152 ページ「カスタム静的テキスト・クラスのサポートの完成」の手順に従って実装を完成します。

ほとんどの場合，静的テキスト・コントロールに対して追加の認識プロパティやテスト・オブジェクト・メソッドをサポートする必要はありません。ただし，ウィザードで新しいサポート・クラスを作成した後に，クラスにメソッドを追加して，追加の認識プロパティやテスト・オブジェクト・メソッド，または記録用のサポートを提供することは可能です。これらのメソッドは，手動で追加するか，または 153 ページ「Eclipse での UFT コマンドの使用」に記載されているコマンドを使用して追加することができます。

Eclipse で New UFT Custom Static-Text Support Class ウィザードを起動するには, 次の手順を実行します。

1 ［Eclipse Package Explorer］タブで，UFT Java Add-in Extensibility プロジェクトを選択します。［**File**］＞［**New**］＞［**Other**］を選択します。［New］ダイアログ・ボックスが開きます。

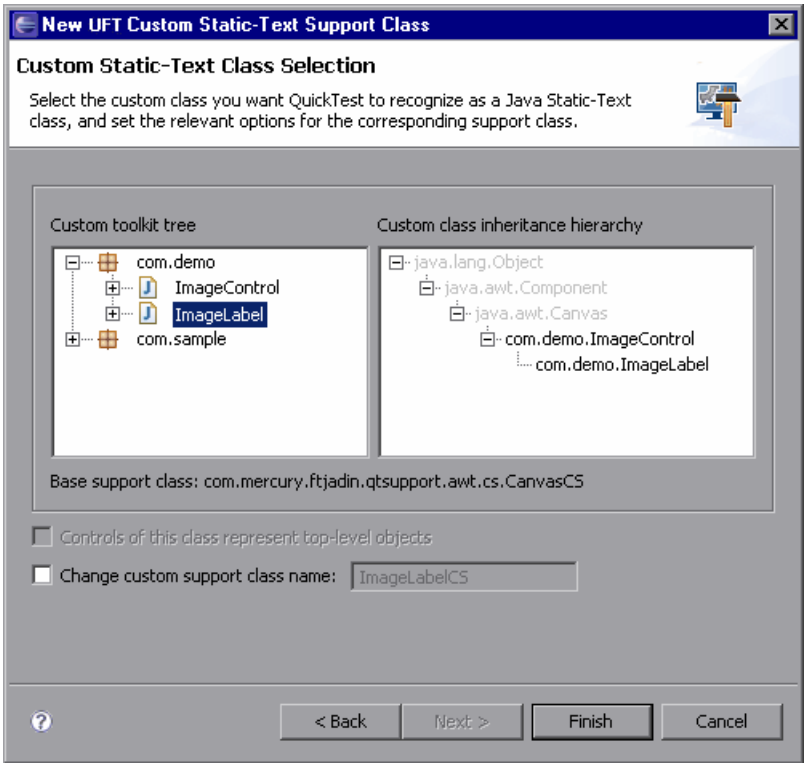
2 **Unified Functional Testing** フォルダを展開し，**UFT Custom Static-Text Support Class** を選択します。

3 ［**Next**］をクリックします。［Custom Static-Text Class Selection］画面が開きます。

ヒント：Eclipse をカスタマイズして **UFT Custom Static-Text Support Class** を［**New**］メニューのオプションにすると，この手順を短縮することができます。これを行うには，［**Window**］＞［**Customize Perspective**］を選択します。表示されるダイアログ・ボックスの［Shortcuts］タブで，［**Unified Functional Testing**］と［**UFT Custom Static-Text Support Class**］のチェック・ボックスを選択します。［**OK**］をクリックします。

［Custom Static-Text Class Selection］画面

［Custom Static-Text Class Selection］画面のオプションは，［Custom Class Selection］画面（117 ページを参照）のオプションと同じです。［**Finish**］をクリックすると，［Custom Static-Text Support Class Summary］画面（151ページを参照）が開きます。

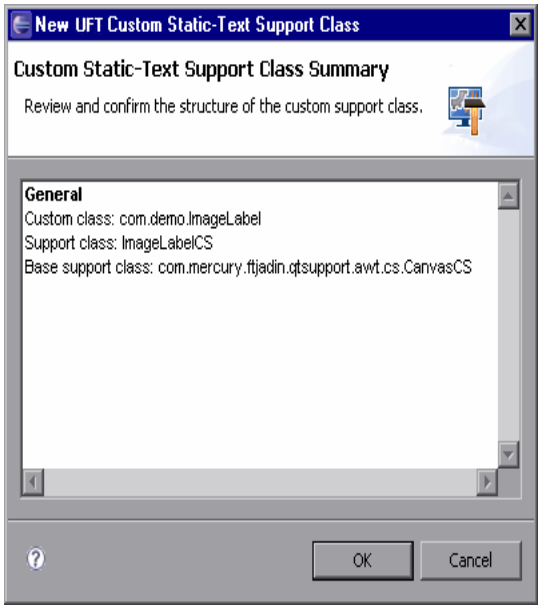


UFT で静的テキストと認識させるカスタム・クラスを選択して，関連するオプションを設定します。

静的テキスト・コントロールには，通常，UFT GUI のテストに関連する認識プロパティやテスト・オブジェクト・メソッドはありません。そのため，このウィザードで追加の指定を行う必要はありません。

［Custom Static-Text Support Class Summary］画面

カスタム・サポート・クラス・ファイルが作成される前に，［Custom Static-Text Support Class Summary］画面に新しいサポート・クラスに対して指定した内容のサマリが表示されます。



変更が必要なデータがある場合は，［**Cancel**］をクリックして上記の［Custom Static-Text Class Selection］画面に戻ります。

完了したら，［**OK**］をクリックします。このウィザードでは，静的テキスト・オブジェクトのサポートに必要な次のメソッドを使用して，新しいサポート・クラスが作成されます。

- **class_attr :** 文字列 static_text を返します。UFT で，このクラスのオブジェクトを静的テキスト・コントロールとして認識することが可能になります。
- **label_attr :** スーパークラスの **label** プロパティを返します。

  Java コントロールの **label** プロパティが空の場合，UFT は隣接する静的テキスト・コントロールを探して，その **label** プロパティをテスト・オブジェクト名に使用します。そのため，適切な名前（静的テキスト・コントロールによって表示される文字列など）を返すように **label_attr** メソッドを実装することができます。

➤ **tag_attr**：（**label** プロパティ値を返す）スーパークラスの **tag** プロパティにサフィックス (st) を付けて返します。**label_attr** メソッドは隣接するコントロールで使用する名前を提供しますが，このメソッドは静的テキスト・コントロールそのもののテスト・オブジェクト名を提供します。

たとえば，**label_attr** メソッドを実装して MyButton を返す場合，**tag_attr** メソッドは MyButton(st) を返します。

詳細については，64 ページ「一般的な認識プロパティ・サポート・メソッド」を参照してください。

➤ **value_attr**：**label** プロパティを返します。

value プロパティは，コントロールのテスト・オブジェクトの状態を表します。静的テキスト・コントロールの場合は，**label** プロパティがこの状態を表します。

カスタム静的テキスト・コントロールのサポートの作成は，チュートリアルの 211 ページ「カスタム静的テキスト・コントロールのサポート方法の学習」で練習することができます。

カスタム静的テキスト・クラスのサポートの完成

（New UFT Custom Static-Text Support Class ウィザードを使用して）カスタム静的テキスト・クラスに対するカスタム・サポート・クラスの作成が完了したら，次の追加手順を実行する必要があります。

➤ クラスを保存します。

Eclipse では，右側の表示枠のタブに新しいクラス・ファイルが開いて表示されます。クラスを保存していないと，サポート・クラス・ファイル名の横のタブにアスタリスク（*****）が表示されます。ウィザードで行われる変更は相互に依存するため，必ず変更を保存して不一致が生じないようにする必要があります。

➤ 必要に応じて，**label_attr** メソッドを実装します。

➤ サポートが使用可能になるように，ツールキット・サポートを UFT にデプロイします。詳細については，78 ページ「カスタム・ツールキット・サポートのデプロイと実行」を参照してください。

Eclipse での UFT コマンドの使用

Java Add-in Extensibility Eclipse プラグインを含む UFT Java Add-in Extensibility SDK をインストールすると，次のボタンを備えたツールバーが Eclipse に追加されます。

ボタン	定義	ボタン	定義
	ツールキット・サポートのデプロイ		認識プロパティの追加
	サポート設定の再ロード		テスト・オブジェクト・メソッドの追加
	カスタム・サポートの削除		イベント・ハンドラの追加

また，これらと同じコマンドを含む新しい **UFT** メニューが Eclipse に追加されます。これらのコマンドについては，次の各項で詳しく説明します。

Deploy Toolkit Support

Deploy Toolkit Support コマンドは，［Eclipse Package Explorer］タブで UFT Java Add-in Extensibility プロジェクト（または，その中の要素）を選択したときに使用できます。

注 : UFT と Java Add-in をインストールする前に UFT Java Add-in Extensibility SDK をインストールした場合，**Deploy Toolkit Support** コマンドは使用できません。この問題を解決するには，UFT Java Add-in Extensibility SDK をアンインストールしてから再度インストールします。詳細については，27 ページ「HP UFT Java Add-in Extensibility Software Development Kit のインストール」を参照してください。

Deploy Toolkit Support コマンドは，開発段階でツールキット・サポートをデプロイする場合に使用します。このコマンドを使用するには，UFT と UFT Java Add-in Extensibility Eclipse プラグインが同じコンピュータ上にインストールされている必要があります。

このコマンドは, ツールキット設定ファイルとテスト・オブジェクト設定ファイルを UFT の適切なフォルダにコピーします。このツールキット設定ファイルで，コンパイルしたサポート・クラスに対して指定される場所は Eclipse ワークスペースです。次回 Java アプリケーションを実行したときに, 作成したサポートを UFT 上で使用することができます。詳細については，78 ページ「カスタム・ツールキット・サポートのデプロイと実行」を参照してください。

注：このデプロイ・コマンドは，デプロイする前に Java クラスをコンパイルしますが，コンパイル結果の検証は行いません。デプロイする前にサポート・クラスを保存し，コンパイル・エラーを確認して，実行時エラーが起きないようにしてください。

Deploy Toolkit Support コマンドは，**<カスタム・ツールキット名>TestObjects.xml** という名前のテスト・オブジェクト設定ファイルのみをコピーします。テスト・オブジェクト設定ファイルを追加で作成する場合は，78 ページ「カスタム・ツールキット・サポートのデプロイと実行」で指定された，適切なフォルダにコピーする必要があります。

Deploy Toolkit Support コマンドでは，対応する XSD ファイルに対する設定ファイルの検証が行われ，形式が要件に適合している場合にのみ設定ファイルがデプロイされます（要件に適合しないと表示された場合でも，デプロイするように指定した場合はデプロイされます）。設定ファイルの構造については，『UFT Java Add-in Extensibility Toolkit Configuration Schema ヘルプ』および『UFT Test Object Schema ヘルプ』（いずれも『Java Add-in Extensibility SDK ヘルプ』で利用可能）を参照してください。

ツールキット設定ファイルは，次のファイルに対して検証されます。
<UFT インストール・フォルダ>\bin\java\sdk\eclipse\plugins\com.mercury.qtjext.plugin.QTJavaExt_1.0.0\ToolkitSchema.xsd

テスト・オブジェクト設定ファイルは，次のファイルに対して検証されます。
<UFT インストール・フォルダ>\bin\java\sdk\eclipse\plugins\com.mercury.qtjext.plugin.QTJavaExt_1.0.0\ClassesDefintions.xsd

Reload Support Configuration

Reload Support Configuration コマンドは，［Eclipse Package Explorer］タブで UFT Java Add-in Extensibility プロジェクト（または, その中の要素）を選択したときに使用できます。

注 : UFT と Java Add-in をインストールする前に UFT Java Add-in Extensibility SDK をインストールした場合，**Reload Support Configuration** コマンドは使用できません。この問題を解決するには, UFT Java Add-in Extensibility SDK をアンインストールしてから再度インストールします。詳細については，27 ページ「HP UFT Java Add-in Extensibility Software Development Kit のインストール」を参照してください。

Reload Support Configuration コマンドでは, 次の内容を再ロードすることにより, UFT Java Add-in Extensibility Eclipse プラグインのサポートされる Java クラスとテスト・オブジェクト・クラスのリストを更新します。

- 選択したプロジェクトの設定ファイルとサポート・クラス
- UFT のインストール・フォルダのすべてのツールキット設定ファイルとテスト・オブジェクト設定ファイル

Reload Support Configuration コマンドは，New UFT Custom Support Class ウィザードの次の項目に影響を及ぼします。

- ［Custom Class Selection］画面（117 ページを参照）のカスタム・ツールキット・ツリーに表示されるカスタム・クラスのリスト。
- ［Custom Class Selection］画面（117 ページを参照）のウィザードによるベース・サポート・クラスの選択内容。
- ［Test Object Class Selection］画面（122 ページを参照）に表示される既存のテスト・オブジェクト・クラスのリスト。

サポート設定の再ロードが必要な場合については，次の例を参照してください。

- UFT Java Add-in Extensibility プロジェクトでテスト・オブジェクト設定ファイルを変更し，テスト・オブジェクト・クラスを追加または削除した場合。この場合は，ウィザードの既存のテスト・オブジェクト・メソッドのリストにこれらの変更を反映する必要があります。
- カスタム・ツールキットのサポートを UFT に手動でデプロイし，クラスがサポートされていることをウィザードで認識する必要がある場合。
- UFT から一部のクラスのサポートを手動で削除し，Eclipse プラグインのサポートされているクラスのリストから，これらのクラスを削除する必要がある場合。
- Eclipse プロジェクトでカスタム・ツールキット・サポート・セット（Support Set A）を作成してデプロイした後で，別の Eclipse プロジェクトで別のカスタム・ツールキットに対してカスタム・ツールキット・サポート・セット（Support Set B）を作成する際に，Support Set A のサポート・クラスを Support Set B のサポート・クラスのベース・サポート・クラスとして使用する必要がある場合。

Delete Custom Support

Delete Custom Support コマンドは，［Eclipse Package Explorer］タブで UFT Java Add-in Extensibility カスタム・サポート・クラスを選択したときに使用できます。

注：このコマンドは，この Eclipse ワークスペースでこのクラスが UFT Java Add-in Extensibility のカスタム・サポート・クラスとして作成されている場合にのみ使用できます。

このコマンドを使用すると，特定のカスタム・クラスに対するサポートが削除されます。サポート・クラスが削除されるだけでなく，ツールキット設定ファイル内のリストも削除されます。このサポート・クラスに対する新しいテスト・オブジェクト・クラスを作成している場合は，ほかのサポート・クラスで使用できるため，テスト・オブジェクト設定ファイルからは削除されません。

Delete Custom Support コマンドの代わりに，Eclipse の削除コマンドを使用してサポート・クラスを削除する場合は，ツールキット設定ファイルでこのサポート・クラスに対するカスタム・コントロールのマッピングを手動で削除する必要があります。

ヒント：サポート・クラスを作成してすぐにそのサポート・クラスを削除する必要がある場合は，最初にそのサポート・クラスを保存してください。

サポート・クラスを削除した後で，このカスタム・クラスのサポートをすでに UFT にデプロイしている場合は，ツールキット・サポートを再度デプロイする必要があります。こうすることで，ツールキット設定ファイルが最新のものに置き換わり，同時に UFT からこのカスタム・クラスのサポートが削除されます。

ほかのサポート・クラスのベース・サポート・クラスの役割を果たしているサポート・クラスを削除する場合は，このほかのクラスの継承を手動で変更する必要があります。たとえば，InheritedCS は ToDeleteCS を拡張し，ToDeleteCS は BuiltInCS を拡張します。ToDeleteCS を削除する場合は，BuiltInCS を拡張するように InheritedCS を手動で変更する必要があります。

UFT Java Add-in Extensibility Eclipse プラグインのコマンドを使用して，完成したツールキットのサポートを削除することはできません。これを行うには，UFT のフォルダのそれぞれの場所から，ツールキット設定ファイルを手動で削除する必要があります。詳細については，78 ページ「カスタム・ツールキット・サポートのデプロイと実行」を参照してください。

Add Identification Property

Add Identification Property コマンドは，［Eclipse Package Explorer］タブで UFT Java Add-in Extensibility カスタム・サポート・クラスを選択したときに使用できます。

注：このコマンドは，この Eclipse ワークスペースでこのクラスが UFT Java Add-in Extensibility のカスタム・サポート・クラスとして作成されている場合にのみ使用できます。

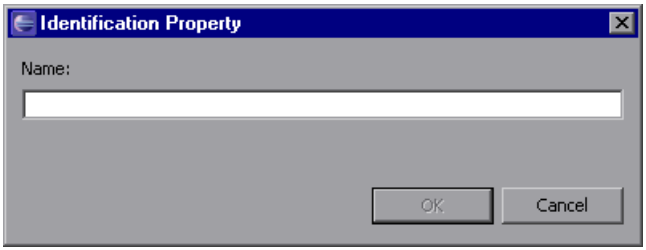
このコマンドは，サポート・クラスの作成後に認識プロパティを追加する場合に使用します。

UFT 上で変更を反映させるには，ツールキット・サポートをデプロイする必要があります。

テスト・オブジェクト・クラスの定義に追加した認識プロパティは，このクラスのすべてのテスト・オブジェクトに対して，UFT の認識プロパティのリストに表示されます。このため，プロパティを追加しようと考えている場合には，既存のテスト・オブジェクト・クラスを変更する代わりに，既存のテスト・オブジェクト・クラスをベースに新しいテスト・オブジェクト・クラスを作成します。

認識プロパティを追加するには，次の手順を実行します。

1 Eclipse の UFT ツールバーで［**Add Identification Property**］ボタンをクリックします。［Identification Property］ダイアログ・ボックスが開きます。



2 新しい認識プロパティの名前を入力し，［**OK**］をクリックします。

3 新しい認識プロパティはサポート・クラスに追加されるだけでなく，サポートされているコントロールにマッピングされたテスト・オブジェクト・クラスの定義にも追加されることを示す確認ボックスが表示されます。この認識プロパティは，このクラスのすべてのテスト・オブジェクトに対し，UFT の認識プロパティのリストに表示されます。

続行する場合は，［**Yes**］をクリックします（［**No**］をクリックすると，新しい認識プロパティは破棄されます)。

<認識プロパティ名>_attr という名前の，定義した認識プロパティに対するサポート・メソッド・スタブがサポート・クラスに追加されます。カスタム・コントロールのニーズに合わせて実装するまでの間，このメソッド・スタブは **null** を返します。

4 新しい認識プロパティをチェックポイントで標準で選択するかどうかを選択するためのメッセージ・ボックスが表示されます。

メッセージ・ボックスで選択を行うと，新しい認識プロパティがテスト・オブジェクト設定ファイルのテスト・オブジェクト・クラスの定義に追加されます。

［**Yes**］をクリックした場合，**ForDefaultVerification** 属性が認識プロパティの定義に追加されて **true** に設定されます。それ以外の場合，**ForDefaultVerification** は追加されません（どちらの場合も，**ForVerification** 属性が追加されて **true** に設定されます。そのため，新しい認識プロパティは必ずチェックポイントで使用することができます)。

認識プロパティを追加して一意のテスト・オブジェクト記述の一部にする場合は，テスト・オブジェクト設定ファイルでこれを手動で定義する必要があります。この認識プロパティの行で，Property と Name の間に，ForDescription="true" を追加します。これにより，**ForDescription** 属性が **Property** 要素に追加されて **true** に設定されます。

詳細については,『UFT Test Object Schema ヘルプ』(『Java Add-in Extensibility SDK ヘルプ』で利用可能）を参照してください。

Add Test Object Method

Add Test Object Method コマンドは，［Eclipse Package Explorer］タブで UFT Java Add-in Extensibility カスタム・サポート・クラスを選択したときに使用できます。

注：このコマンドは，この Eclipse ワークスペースでこのクラスが UFT Java Add-in Extensibility のカスタム･サポート･クラスとして作成されている場合にのみ使用できます。

このコマンドは，サポート・クラスの作成後にテスト・オブジェクト・メソッドを追加する場合に使用します。

UFT 上で変更を反映させるには, ツールキット･サポートをデプロイする必要があります。

既存のテスト・オブジェクト・クラスにテスト・オブジェクト・メソッドを追加すると，このクラスのすべてのテスト・オブジェクトに対し，これらのオブジェクトでサポートされるかどうかに関係なく，UFT に新しいメソッドが追加されます。UFT GUI のテストでは，オブジェクトに対するテスト・オブジェクト・メソッドを呼び出して，そのメソッドがサポートされていない場合，実行時エラーが発生します。

そのため，カスタム・コントロールをサポートするためにテスト・オブジェクト・メソッドを追加しようと考えている場合は，既存のテスト・オブジェクト・クラスを変更する代わりに，既存のテスト・オブジェクト・クラスをベースに新しいテスト・オブジェクト・クラスを作成することをお勧めします。

テスト・オブジェクト・メソッドを追加するには，次の手順を実行します。

1 Eclipse の UFT ツールバーで［**Add Test Object Method**］ボタンをクリックします。［Test Object Method］ダイアログ・ボックスが表示されます。

2 追加するテスト・オブジェクト・メソッドの詳細を定義して，［**OK**］をクリックします。詳細については，133 ページ「［Test Object Method］ダイアログ・ボックスについて」を参照してください。

3 新しいテスト・オブジェクト・メソッドはサポート・クラスに追加されるだけでなく，サポートされているコントロールにマッピングされたテスト・オブジェクト・クラスの定義にも追加されることを示す確認ボックスが表示されます。このテスト・オブジェクト・メソッドは，このクラスのすべてのテスト・オブジェクトに対して UFT に表示されます。

続行する場合は，［**Yes**］をクリックします（［**No**］をクリックすると，新しいテスト・オブジェクト・メソッドは破棄されます）。

<テスト・オブジェクト・メソッド名>_replayMethod という名前の，定義したテスト・オブジェクト・メソッドに対するサポート・メソッド・スタブがサポート・クラスに追加されます。カスタム・コントロールのニーズに合わせて実装するまでの間，このメソッド・スタブはエラー値 **Retval.NOT_IMPLEMENTED** を返します。

また，このテスト・オブジェクト・メソッドは，テスト・オブジェクト設定ファイルのテスト・オブジェクト・クラスの定義にも追加されます。このファイルの構造と内容については，『UFT Test Object Schema ヘルプ』（『Java Add-in Extensibility SDK ヘルプ』で利用可能）を参照してください。

Add Event Handler

Add Event Handler コマンドは，［Eclipse Package Explorer］タブで AWT ベースの UFT Java Add-in Extensibility カスタム・サポート・クラスを選択したときに使用できます。

注：このコマンドは，この Eclipse ワークスペースでこのクラスが UFT Java Add-in Extensibility のカスタム・サポート・クラスとして作成されている場合にのみ使用できます。

このコマンドは，サポート・クラスの作成後にイベント・ハンドラを追加する場合に使用します。

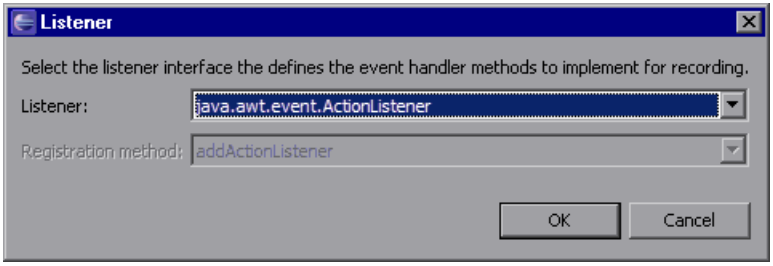
［Custom Control Recording Support］ウィザード画面では，新しいサポート・クラスの作成時に次のオプションが利用できます。

- **Treat controls of this class as wrapper controls**
- **Override low-level mouse event recording**
- **Override low-level keyboard event recording**

サポート・クラスの作成時にこれらを選択しなかった場合，これらを実装するには，手動で設定を行う必要があります。これを行う手順については，68 ページ「記録オプションのサポート」を参照してください。

イベント・ハンドラ・メソッドを追加するには，次の手順を実行します。

1 Eclipse の UFT ツールバーで［**Add Event Handler**］ボタンをクリックします。［Listener］ダイアログ・ボックスが開きます。



2 リストからリスナを選択します。

選択したリスナに複数の登録メソッドがある場合には，［**Registration method**］リストからメソッドを 1 つ選択します。

3 ［**OK**］をクリックします。

選択したリスナがサポート・クラスの署名に追加されます。

選択したリスナに対して **addSimpleListener** を呼び出すため，コンストラクタがサポート・クラスに追加されます（すでに存在する場合は変更されます）。これにより，カスタム・コントロール上で登録が必要なリスナー覧にリスナが追加され，登録および削除に使用するメソッドが指定されます。

選択したリスナを含むすべてのイベント・ハンドラ・メソッドに対するメソッド・スタブがサポート・クラスに追加されます。メソッド・スタブには，イベント・ハンドラを実装し，**MicAPI.record** を呼び出して記録メッセージを UFT に送信する必要があることを示すコメントが追加されます。詳細については，68 ページ「記録オプションのサポート」を参照してください。

第 II 部

チュートリアル：Java カスタム・ツールキット・サポートの作成方法の学習

第 6 章

UFT Java Add-in Extensibility チュートリアルの使用

UFT Java Add-in Extensibility チュートリアルでは，カスタム・ツールキット・サポートの作成，テスト，デプロイの手順をレッスンします。このチュートリアルの内容をすべて学習すると，カスタム・ツールキットやカスタム・コントロールに対する UFT サポートを作成できるようになります。

本章の内容

チュートリアルの構成について

このチュートリアルは，複数のレッスンで構成されています。それぞれのレッスンは，前のレッスンをすでに終了しているか，同等レベルの経験があることが前提となっています。これらのレッスンでは，UFT Java Add-in Extensibility で利用できる機能や手法について詳細に学習します。順を追ってレッスンを進めることをお勧めします。

このチュートリアルの各レッスンでは, UFT Java Add-in Extensibility Eclipse プラグインを使用して，各種カスタム・コントロールに対して UFT サポートを拡張します。カスタム・コントロールは，**<Java Add-in Extensibility SDK インストール・フォルダ>\samples** フォルダにあるサンプル・カスタム・ツールキットに用意されています。

このフォルダには，これらのカスタム・コントロールをサポートするのに必要なカスタム・ツールキット・サポート・セット，およびカスタム・ツールキットとそのサポートの追加サンプルも含まれています。サンプルのカスタム・ツールキット・サポート・セットを手動でデプロイする場合は，前もって Java クラスをコンパイルする必要があります。

このチュートリアルの各レッスンでは，それぞれのレッスンで使用する必要のある Java Add-in Extensibility ウィザードのオプションについて説明しています。これらのウィザードの詳細については，101 ページ「UFT Java Add-in Extensibility Eclipse プラグインの使用」を参照してください。

シンプルなコントロールのサポート方法の学習について

169 ページ「シンプルなコントロールのサポート方法の学習」のレッスンでは，ImageButton という名前の基本的なカスタム Java コントロールを使用して，カスタム・サポートの基本的要素について学習します。このレッスンでは，カスタム・サポート・クラスを 1 つ含むカスタム・ツールキット・サポート・プロジェクトの作成に必要な手順について学習し，カスタム・サポートを構成するファイルやメソッドについて理解します。

このレッスンでは，UFT Java Add-in Extensibility Eclipse プラグインで提供される 2 つのウィザード（New UFT Java Add-in Extensibility Project ウィザードと New UFT Custom Support Class ウィザード）を使用します。

カスタム静的テキスト・コントロールのサポート方法の学習について

211 ページ「カスタム静的テキスト・コントロールのサポート方法の学習」のレッスンでは，ImageLabel コントロールを使用して，静的テキスト・コントロールをサポートする方法を学習します。このレッスンでは，既存のカスタム・ツールキット・サポート・プロジェクトにおける静的テキスト・コントロールのサポート・クラスの作成に必要な手順について学習します。（ImageLabel コントロールは，前のレッスンで使用した ImageButton コントロールと同じカスタム・ツールキットに属しています。）このレッスンを通して，静的テキスト・コントロールのサポート・クラスで使用する基本的なメソッドについて学習することができます。

このレッスンでは，UFT Java Add-in Extensibility Eclipse プラグインで提供される New UFT Custom Static-Text Support Class ウィザードを使用します。

複雑なコントロールのサポート方法の学習について

241 ページ「複雑なコントロールのサポート方法の学習」のレッスンでは，カスタム Java コントロール AllLights を使用して，カスタム・サポートについて詳しく学習します。AllLights は固有の動作を持つ複雑なコントロールで，テスト・オブジェクト・クラスを新規に定義する必要があります。このレッスンでは，親のサポート・クラスには存在しなかった新しい認識プロパティとテスト・オブジェクト・メソッドを含むカスタム・サポート・クラスを作成する手順について学習します。また，テスト・オブジェクト設定ファイルについても理解することができます。

チュートリアルの前提条件の確認

このチュートリアルのレッスンを始める前に，ここに記載した要件を満たしていることを確認してください。

システム要件

このチュートリアルのカスタム・ツールキットおよびカスタム・コントロール用のサポートが実装されていないコンピュータに，以下をインストールしておく必要があります。サポートがすでに実装されている場合は，78 ページ「カスタム・ツールキット・サポートのデプロイと実行」を参照してサポートを削除してください。

Eclipse

サポート対象の Eclipse バージョンについては，『HP Unified Functional Testing 使用可能製品マトリクス』を参照してください。これは，Unified Functional Testing ヘルプにアクセスするか，Unified Functional Testing DVD のルート・フォルダに収録されています。

Java Add-in Extensibility SDK

Eclipse または Java Add-in Extensibility SDK のインストールについては，27 ページ「HP UFT Java Add-in Extensibility Software Development Kit のインストール」を参照してください。

Unified Functional Testing（Java Add-in を含む）

Unified Functional Testing（UFT）のインストールについては,『HP Unified Functional Testing インストール・ガイド』を参照してください。

UFT が Eclipse と同じコンピュータにインストールされていない場合でも，このチュートリアルのレッスンを行うことができます。ただし，UFT にツールキット・サポートをデプロイするように指示された場合には，78 ページ「カスタム・ツールキット・サポートのデプロイと実行」を参照して，カスタム・サポート・クラス・ファイルや設定ファイルを UFT のコンピュータの適切なフォルダに手動で転送する必要があります。

必要な知識

このチュートリアルのレッスンでは，以下の知識があることが前提となっています。

UFT の主要な特徴や機能に関する理解

テスト・オブジェクト，オブジェクト・リポジトリ，オブジェクト・スパイ，キーワード・ビュー，エディタについて十分に理解している必要があります。また，テストの記録，編集，実行に関する経験も必要です。詳細については，『HP Unified Functional Testing ユーザーズ・ガイド』を参照してください。

Java プログラミングの経験

Java プログラミングに関連する概念（クラス，パッケージ，インタフェース，継承など）についての知識があり，Java クラスの記述およびコンパイル方法について理解している必要があります。

XML に関する知識

要素と属性の概念を理解し，スキーマの使用や XML ファイルの編集に慣れている必要があります。

「Implementing Custom Toolkit Support」の章に記載されている基本的な概念について理解している必要があります。

このチュートリアルは，「カスタム・ツールキットのサポートの実装」（35 ページ～）に記載されている概念について理解していることが前提となっています。

第 7 章

シンプルなコントロールのサポート方法の学習

このレッスンでは，ImageControls ツールキット内に ImageButton コントロールのサポートを作成します。ImageButton コントロールのサポートの追加では，最小限のカスタマイズのみを行うため，カスタム・ツールキット・サポート・セットの作成に関する基本事項を習得することができます。

このレッスンを行う前に，本書の「カスタム・ツールキットのサポートの実装」と「カスタム・ツールキット・サポートの計画」の章をよく読み，165 ページ「UFT Java Add-in Extensibility チュートリアルの使用」に記載されているチュートリアルの前提条件を確認しておく必要があります。

このレッスンは，次の内容で構成されています。

このレッスンの準備

カスタム・コントロールに対する UFT サポートを拡張する前に，以下の点を確認しておく必要があります。

- コントロールに対するフルアクセスが可能であることを確認します。
- コントロールの動作とテストが必要な機能を確認します。
- コントロールを表示して操作するためのアプリケーションを用意します。
- コントロールを実装するクラスにアクセスできることを確認します(カスタム・サポートの作成時にカスタム・コントロール・クラスを変更することはありませんが，コンパイルしたクラスを参照して，ソース・ファイルから取得できる情報を使用します)。

次の手順を実行して，ImageControls カスタム・ツールキット・クラスを含む Eclipse プロジェクトと，カスタム・コントロールを含むサンプル・アプリケーションを作成します。

注：このサンプル・アプリケーションは，標準の **<UFT Java Add-in Extensibility SDK インストール>\samples** フォルダから実行するように設計されています。SDK を別の場所にインストールする場合は，このレッスンを始める前にサンプル・アプリケーションを若干変更する必要があります。詳細については，172 ページ「別の場所から実行するようにサンプル・アプリケーションを変更する」を参照してください。

Eclipse で ImageControls サンプルを使用して新しい Java プロジェクトを作成するには，次の手順を実行します。

1. Eclipse を実行し，［**File**］＞［**New**］＞［**Project**］を選択します。［New Project］ダイアログ・ボックスが開きます。
2. ［**Java Project**］を選択し，［**Next**］をクリックします。［New Java Project］ダイアログ・ボックスが開きます。
3. ［**Project name**］ボックスに ImageControls と入力します。
4. ［**Create project from existing source**］オプションを選択します。
5. ［**Browse**］ボタンをクリックして，**<UFT Java Add-in Extensibility SDK インストール・フォルダ>\samples\ImageControls\src** フォルダを参照します。［**OK**］をクリックして，［New Java Project］ダイアログ・ボックスに戻ります。

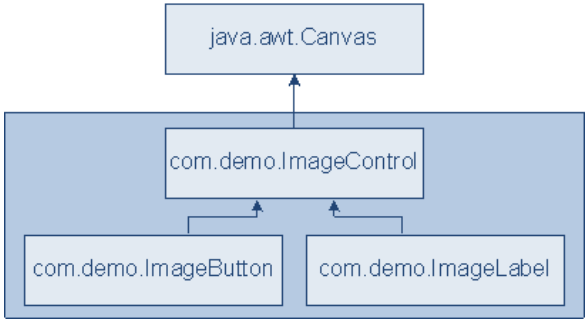
6 ［**Finish**］をクリックします。ImageControls のサンプル・ソース・ファイルを使用して，新規の Java プロジェクトが作成されます。［Package Explorer］タブに，**ImageControls** という名前の新規プロジェクトが表示されます。

注：Eclipse で新しいプロジェクトを作成する手順は，使用する Eclipse のバージョンによって異なることがあります。

ImageControls プロジェクトを展開して内容を表示します。**ImageControls\src** パッケージ・フォルダには，次の 2 つのパッケージが含まれています。

- **com.sample** パッケージには，サンプル・アプリケーション（**SampleApp**）が含まれています。
- **com.demo** パッケージには，3 つのカスタム・コントロール（**ImageButton**，**ImageControl**，**ImageLabel**）が含まれています。

次の図は，**com.demo** パッケージ内のクラスの継承階層を表しています。



このパッケージ内のクラスで提供される機能は，次のとおりです。

- **ImageControl** : このクラスは **Canvas** クラスを拡張し，コントロール上に画像を表示します。
- **ImageLabel** : このクラスは **ImageControl** クラスを拡張し，コントロールに表示される画像の上にテキストを表示します。
- **ImageButton** : このクラスは **ImageControl** クラスを拡張し，コントロールの周りにボタン状の長方形を描画します。このクラスはコントロール上の低レベル・イベントをリッスンして，ボタンがクリックされたときに **Action** イベントをトリガします。

別の場所から実行するようにサンプル・アプリケーションを変更する

UFT Java Add-in Extensibility SDK が **C:/Program Files/HP/Unified Functional Testing** 以外のフォルダにインストールされている場合は，このレッスンを始める前にサンプル・アプリケーションを変更する必要があります。

サンプル・アプリケーションを変更するには，次の手順を実行します。

1. ImageControls ソース・ファイルを Eclipse にコピーした後で，Eclipse でパッケージ **ImageControls\src\com.sample** を参照して, **SampleApp.java** ファイルを開きます。
2. 次の画像ファイルのパスを含むコードを探します。

```
C:/Program Files/HP/Unified Functional Testing/samples/ImageControls/images/
mercury.gif
C:/Program Files/HP/Unified Functional Testing/samples/ImageControls/images/
JavaExt1.gif
```

3. これらのパスの C:/Program Files/HP/Unified Functional Testing を実際のインストール・フォルダに置き換えて，サンプル・アプリケーションが正しく機能するようにします。

ImageButton コントロールのサポートの計画

ここでは，ImageButton コントロールの現在の UFT サポートを分析し，95 ページ「カスタム・クラス・サポート計画のチェックリストについて」の質問に答えて，177 ページ「カスタム・クラス・サポート計画のチェックリスト」に記入します。

このためには，カスタム・コントロールを含むアプリケーションを実行し，オブジェクト・スパイ，キーワード・ビュー，記録オプションを使用して UFT の観点から分析を行うのが最適です。

1 SampleApp アプリケーションを実行して，UFT を開きます。

a ［Eclipse Package Explorer］タブで，**SampleApp** を右クリックします。［**Run As**］＞［**Java Application**］を選択します。SampleApp アプリケーションが開きます。

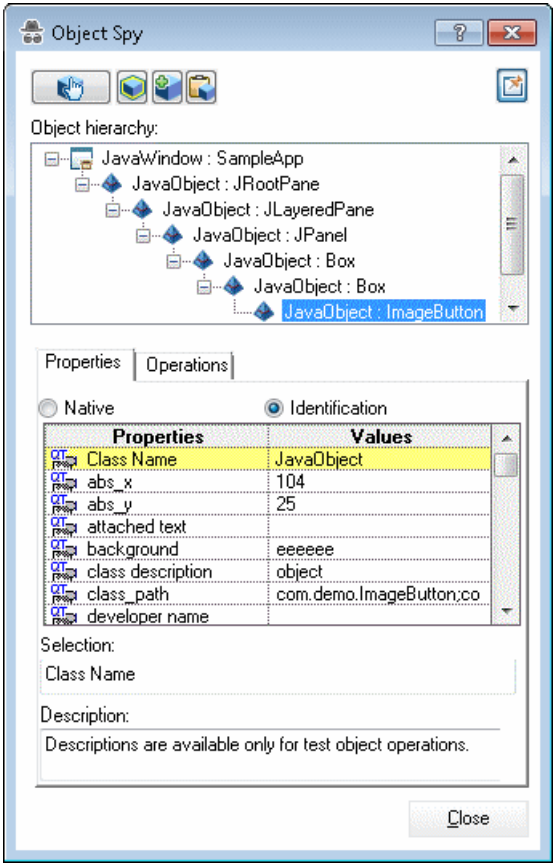
b UFT を開いて，Java Add-in をロードします。

2 オブジェクト・スパイを使用して ImageButton のプロパティを表示します。

a UFT で，GUI テストを開き，［**ツール**］＞［**オブジェクト スパイ**］を選択するか，［**オブジェクト スパイ**］ツールバー・ボタンをクリックして，［オブジェクト スパイ］ダイアログ・ボックスを開きます。［**プロパティ**］タブをクリックします。

b ［オブジェクト スパイ］ダイアログ・ボックスで，指差しアイコンをクリックしてから，SampleApp アプリケーション内のボタンをクリックします。

ImageButton コントロールは，UFT で認識されないカスタム・クラスに基づいています。ボタンは **ImageButton** という名前の汎用の **JavaObject** として認識されるため，表示されるアイコンは標準の JavaObject クラスのアイコンになります。



c オブジェクト・スパイを閉じます。

3 ImageButton コントロールで操作を記録します。

a UFT で，［**記録**］＞［**記録と実行環境設定**］を選択して，［記録と実行環境設定］ダイアログ・ボックスを開きます。［Java］タブで，［**開いているすべての Java アプリケーションでテストを記録して実行する**］を選択します。Web アドインもロードされる場合は，［**Web**］タブをクリックして［**開いているすべてのブラウザでテストを記録して実行する**］を選択します。［**OK**］をクリックします。

b ［**記録**］ボタンをクリックするか，［**記録**］＞［**記録**］を選択します。SampleApp アプリケーションのボタンをクリックします。エディット・ボックスのカウンタ値が 1 ずつ増えます。

新しいステップがテストに追加されます。

Item	Operation	Value	Documentation
Action1			
SampleApp			
ImageButton	Click	30,26,"LEFT"	Click the "ImageButton" object with the "LEFT" m

c ［**停止**］ボタンをクリックするか，［**記録**］＞［**停止**］を選択して記録セッションを終了します。

ImageButton という JavaObject で記録される **Click** 操作は汎用クリックで，低レベル記録の詳細を表す引数を持っています（x 座標と y 座標およびクリックを実行したマウス・ボタン）。

4 ImageButton コントロールが属するカスタム・ツールキットを決定します。

コントロールに対する UFT サポートを拡張する場合は，必ずツールキットのコンテキストで行います。このチュートリアルでは，同じ先祖（**java.awt.Canvas**）を持つ 3 つのクラスをまとめて，ImageControls というカスタム・ツールキットを形成します（ImageButton，ImageLabel，これらのスーパークラスの ImageControl）。

このレッスンでは，ImageControls ツールキットのサポートを作成しますが，最初は ImageButton クラスのみをサポートします。

5 カスタム・クラス・サポート計画のチェックリストを完成させます。

ここでは，UFT で ImageButton を特殊なボタンとして扱い，実行される操作をサポートする必要があります。そのため，このコントロールに対する Extensibility サポートを作成します。

カスタム・クラス ImageButton は別のカスタム・クラスである ImageControl を拡張します。これに対しても，UFT はサポートを提供していません。この時点では，ImageButton が ImageControl を拡張するほかのクラスと共有する特別な UFT サポートを必要とする機能が存在するようには見えません。そのため，ImageButton クラスに直接サポートを拡張すれば十分です。

ImageButton クラスが完全にサポートされると，UFT は ImageButton コントロールを JavaButton テスト・オブジェクトとして認識します。このタイプのコントロールを表す JavaButton テスト・オブジェクトには，コントロールで表示される画像ファイルの名前に準じた名前を付けるものとします。

このカスタム・サポートには，シンプルなボタン・クリック操作のサポートも含める必要があります（UFT では，シンプルな JavaButton の **Click** 操作は，クリックを実行したマウス・ボタンを示すオプションの引数を取ります）。ImageButton カスタム・クラスは低レベル・マウス・イベントをリッスンして，ボタン動作と関連性の高いイベント（この場合は，**Action** イベント）に置き換えます。そのため，マウス・クリックを記録するには, サポート・クラスで **Action** イベントをリッスンする必要があります。

次のページに上記の情報に基いて作成したチェックリストを示します。

カスタム・クラス・サポート計画のチェックリスト

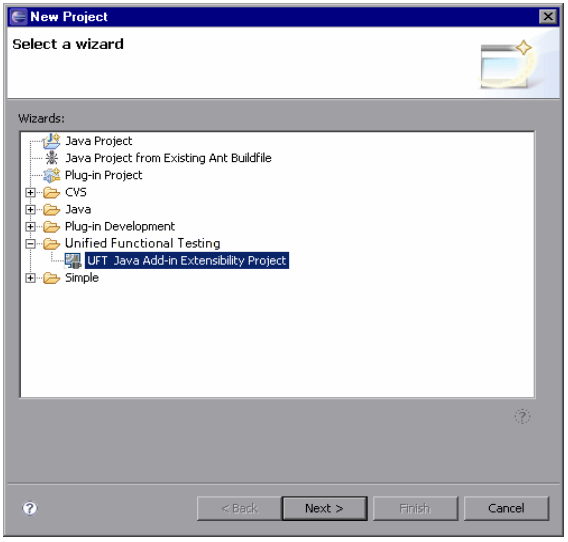
☑	カスタム・クラス・サポート計画のチェックリスト
☑	カスタム・クラスに UFT カスタム・サポートが提供されていないスーパークラスが存在するか？ **いいえ**
☑	存在する場合，階層内の上位のコントロールに対するサポートを最初に拡張すべきか？ **該当なし**
☑	UFT がインストールされているコンピュータ上でカスタム・コントロールを実行するアプリケーションが存在するか？ **はい**
☑	このカスタム・コントロール・クラスのソースが存在する場所： ImageControls という Eclipse プロジェクト
☑	カスタム・コントロールに適合する既存の Java テスト・オブジェクトはどれか？ JavaButton
☑	存在しない場合に作成する新しい Java テスト・オブジェクト・クラス：**該当なし** ➤ 新しいテスト・オブジェクト・クラスによって拡張されるもの：（標準 JavaObject） ➤ アイコン・ファイルの場所（オプション）： ➤ 説明用の認識プロパティ： ➤ 標準設定のテスト・オブジェクト・メソッド：
☑	カスタム・コントロールはトップレベル・オブジェクトか？ **いいえ**
☑	カスタム・コントロールはラッパーか？ **いいえ**
☑	テスト・オブジェクトの名前の基準を指定： **テスト・オブジェクトの画像ファイル名**
☑	サポートする認識プロパティを列挙し，標準設定のチェックポイントのプロパティをマークする： **特になし**
☑	サポートするテスト・オブジェクト・メソッドを列挙する（必要に応じて，引数と戻り値も記入する）： Click（**ボタン**）
☑	記録をサポートするか？（AWT ベースのみ） **はい**
☑	その場合，記録をトリガするイベントを列挙： ActionEvents

UFT Java Add-in Extensibility プロジェクトの新規作成

ここでは，ImageControls ツールキット・サポートのためのプロジェクトを新規に作成します。これを行うには，Eclipse で UFT Java Add-in Extensibility プラグインによって提供されるいずれかのウィザードを使用します。

1 New UFT Java Add-in Extensibility Project ウィザードを開きます。

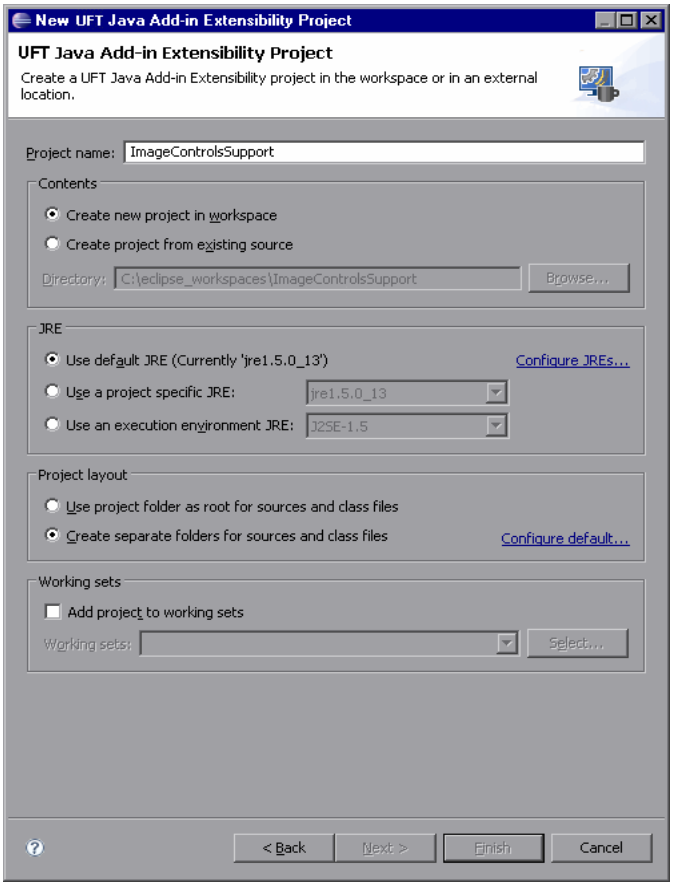
a Eclipse で，［**File**］＞［**New**］＞［**Project**］を選択します。［New Project］ダイアログ・ボックスが開きます。**Unified Functional Testing** フォルダを展開し，**UFT Java Add-in Extensibility Project** を選択します。



b ［**Next**］をクリックします。［UFT Java Add-in Extensibility Project］画面が開きます。この画面の詳細は，使用する Eclipse のバージョンによって異なることがあります。

2 UFT Java Add-in Extensibility プロジェクトの詳細を入力します。

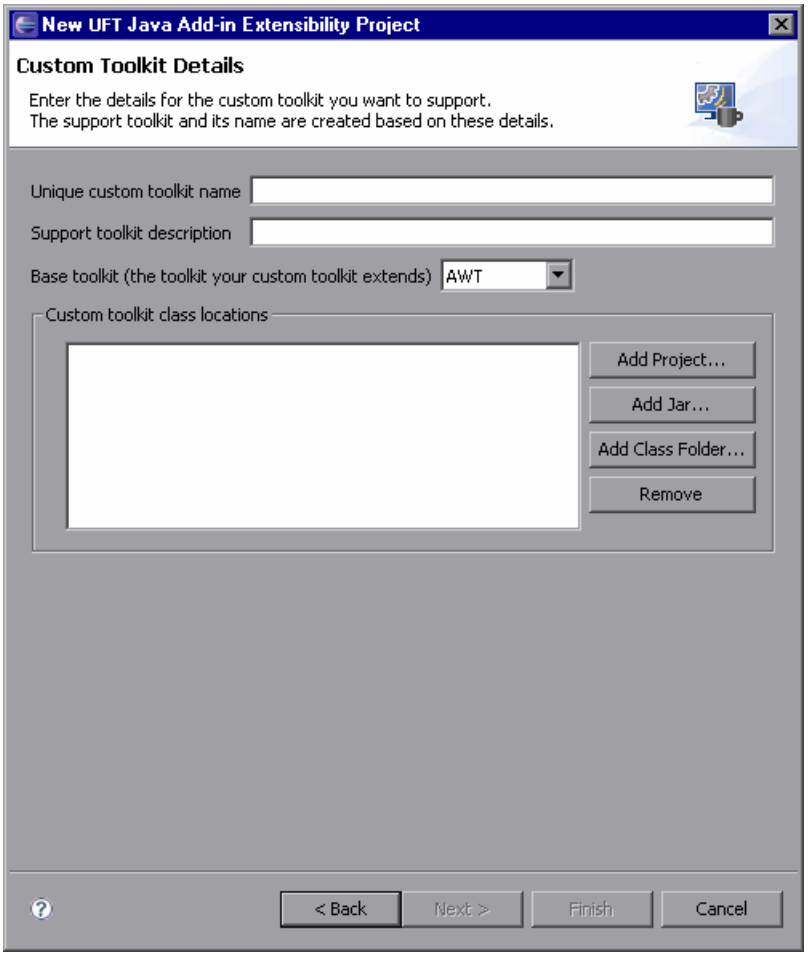
a ［**Project name**］ボックスに, ImageControlsSupport と入力します。［**Create separate folders for sources and class files**］を選択します（以前のバージョンの Eclipse では，このオプションの表記は［**Create separate source and output folders**］となっています)。このダイアログ・ボックスの詳細については，Eclipse のヘルプを参照してください。



b ［**Next**］をクリックします。［Custom Toolkit Details］画面が表示されます。

3 カスタム・ツールキットの詳細を入力します。

この画面では，対応するカスタム・ツールキット・サポート・セットをウィザードで生成できるように，ImageControls ツールキットの詳細を指定します。



a 次の内容を定義します。

➤［**Unique custom toolkit name**］ボックスに，サポートの作成対象のカスタム・ツールキットを一意に表す名前を入力します。新しいツールキット・サポート・クラスには，この名前の末尾に Support という語を追加した名前が付与されます。一意のツールキット名を指定することで，1 つの UFT で多数のカスタム・ツールキット・サポート・セットを同時にサポートすることが可能になります。

サポートを作成して UFT にデプロイすると，アドインやサポートされている環境のリストを表示する UFT のすべてのダイアログ・ボックスにカスタム・ツールキット名が表示されます。

ImageControls という名前を入力します。

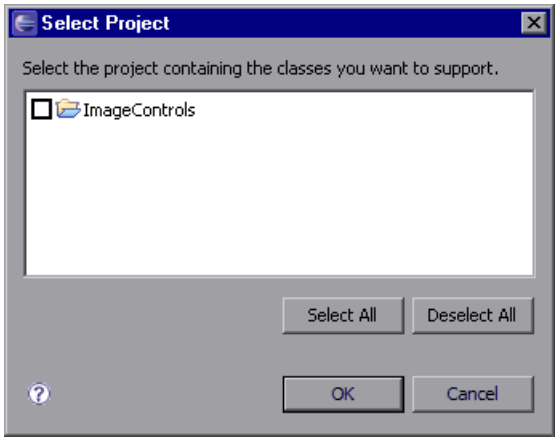
➤［**Support toolkit description**］ボックスに，ImageControls toolkit support と入力します。

➤［**Base toolkit**］リストには，UFT サポートがすでに存在しているツールキットのリストが表示されます。独自のツールキット用のサポートを作成すると，このリストにそのツールキットも表示されます。

ImageButton カスタム・クラスは **AWT** コンポーネントを拡張するため，［**Base toolkit**］は標準設定の **AWT** のままにしておきます。

➤ このツールキットでサポートするカスタム・クラスの場所を指定する必要があります。新しい Java Add-in Extensibility プロジェクトがビルドされると，これらのクラスがプロジェクトのビルド・パスに追加されます。クラスの場所には，**.jar** ファイルまたはファイル・システムのフォルダを指定することができます。

［**Custom toolkit class locations**］領域で［**Add project**］をクリックして，ImageControls ツールキット用のカスタム・クラスを含む Eclipse の Java プロジェクトを選択します。［Select Project］ダイアログ・ボックスが開き，現在の Eclipse ワークスペースのプロジェクトが表示されます。

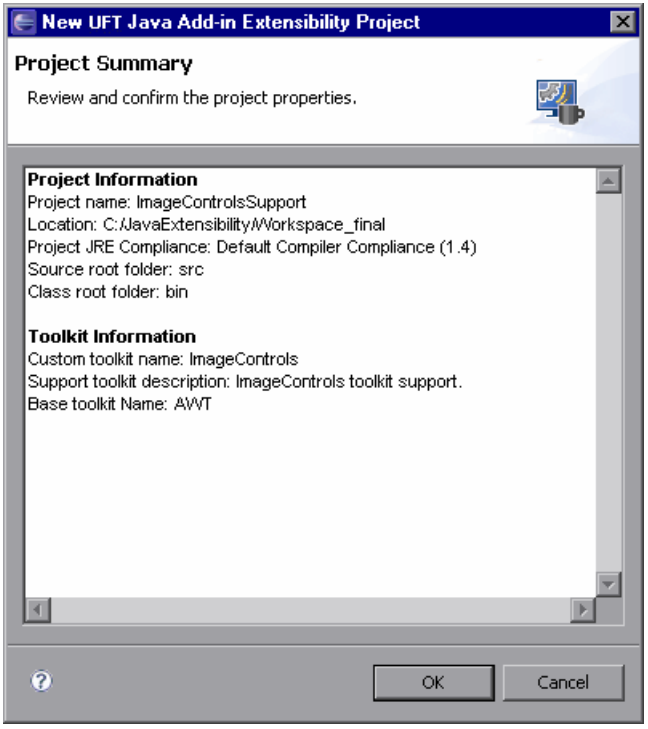


b ［**ImageControls**］チェック・ボックスを選択します。［**OK**］をクリックします。ImageControls プロジェクトが［**Custom toolkit class locations**］ボックスに追加されます。

c ［**Finish**］をクリックします。［Project Summary］画面が開きます。

4 ［Project Summary］画面を表示します。

プロジェクトの詳細を確認して［**OK**］をクリックします。



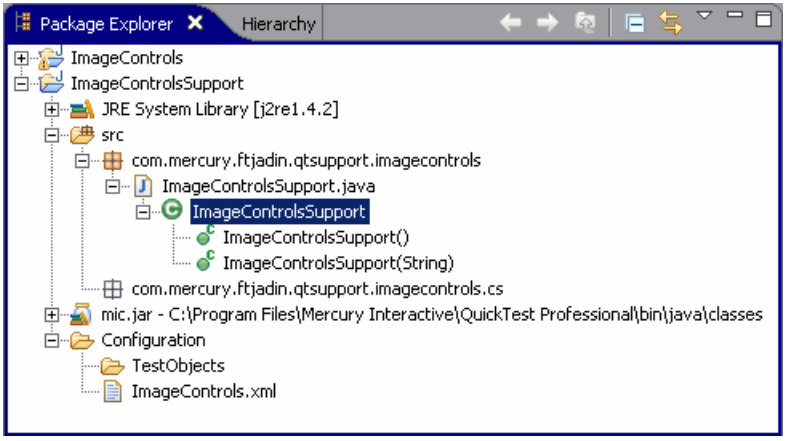
ImageControlsSupport という名前の新しい Java Add-in Extensibility プロジェクトが作成されます。これには，カスタム・ツールキット・サポートに必要な基本的なファイルが含まれます。

新しいカスタム・ツールキット・サポート・セットについて

新しい Java Add-in Extensibility プロジェクトは，［Package Explorer］タブに表示されます。

注：コンピュータ上に複数の JRE がインストールされている場合は，ImageControls プロジェクトと ImageControlsSupport プロジェクトで同じ JRE バージョンを使用するようにしてください。同じ JRE バージョンを使用していない場合は，どちらかのプロジェクトの JRE を変更して，同じバージョンを使用するようにしてください。

ImageControlsSupport プロジェクトを展開して内容を表示します。



src フォルダには，以下のパッケージが含まれています。

➤ **com.mercury.ftjadin.qtsupport.imagecontrols**

このパッケージには，新しいツールキット・サポート・クラス「**ImageControlsSupport**」を定義する，新しいツールキット・サポート・クラス・ファイル「**ImageControlsSupport.java**」が含まれています。

```
public class ImageControlsSupport extends AwtSupport {
}
```

サポートの作成対象の ImageControls ツールキットは AWT を拡張します。そのため，ImageControls ツールキット・サポート・クラスは，組み込みの UFT **AwtSupport** を拡張します。このクラスには，追加の実装は必要ありません。

➤ **com.mercury.ftjadin.qtsupport.imagecontrols.cs**

このパッケージは現在空です。個別のカスタム・コントロール・サポート・クラスを作成すると，このパッケージに保存されます。

Configuration フォルダには，以下の内容が含まれています。

➤ **TestObjects** フォルダ：

このフォルダは現在空です。ツールキットでカスタム・コントロールを表す新しいテスト・オブジェクト・クラスを作成すると，テスト・オブジェクト設定ファイルがこのフォルダに作成されます。これについては，このレッスンでは扱いません。

➤ ツールキット設定ファイル：**ImageControls.xml**

ファイルを開いて内容を表示します。

```
<Controls
   class="com.mercury.ftjadin.qtsupport.imagecontrols.ImageControlsSupport"
   SupportClasspath="C:\JavaExtensibility\Workspace_final\ImageControlsSupport\
                     bin"
   description="ImageControls toolkit support.">
</Controls>
```

この時点で，XML ファイルには，**class**，**SupportClasspath**，**description** 属性の値を指定してツールキット・サポート・クラスを宣言する **Controls** 要素が 1 つ含まれます。

個別のカスタム・コントロール・サポート・クラスを作成すると，そのサポート・クラスに対する各カスタム・コントロールのマッピングがこの設定ファイルに追加されます。

現在，サポート・クラスの場所は Eclipse ワークスペース内にあります。カスタム・サポートの作成段階では，この場所を使用するのが適しています。サポートの実装とテストがすべて済んだら，サポート・クラスを UFT コンピュータ上の恒久的な場所に保存して，ツールキット設定ファイルの値を更新してください。詳細については，78 ページ「カスタム・ツールキット・サポートのデプロイと実行」を参照してください。

このファイルの構造の詳細については，『UFT Java Add-in Extensibility Toolkit Configuration Schema ヘルプ』（『Java Add-in Extensibility SDK ヘルプ』で利用可能）を参照してください。

UFT カスタム・サポート・クラスの新規作成

ここでは，ImageControls ツールキットに含まれる ImageButton コントロールのカスタム・サポート・クラスを作成します。これを行うには，Eclipse で UFT Java Add-in Extensibility プラグインによって提供されるいずれかのウィザードを使用します。各ウィザード画面では，カスタム・サポート計画の内容に合わせて詳細を指定します。続く項では，ウィザードによってこのクラス内に作成されるメソッドの実装を行います。

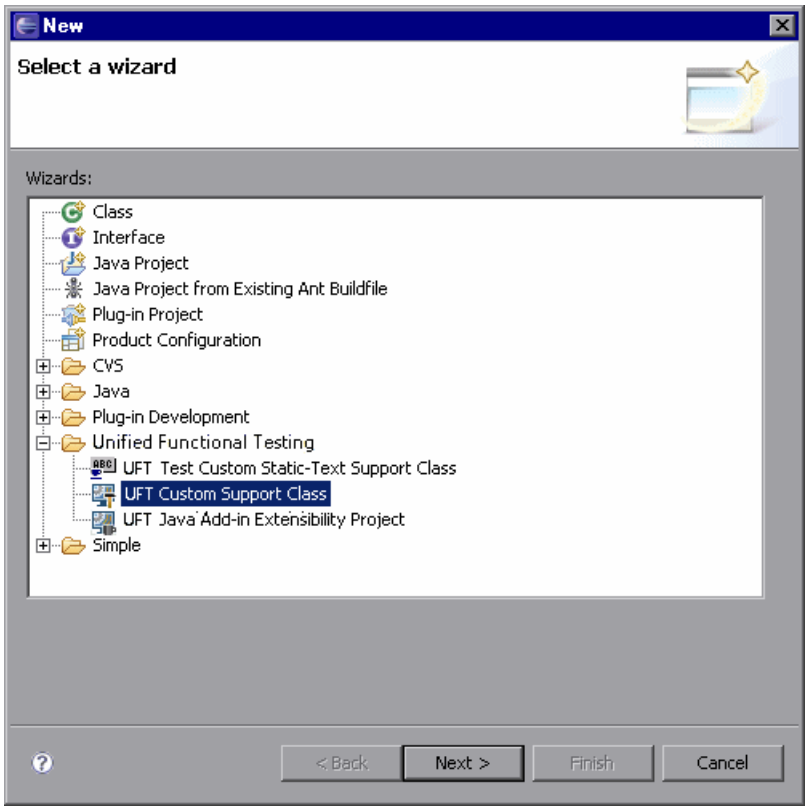
注：次の項では，このレッスンに関連するウィザード画面のオプションについてのみ説明します。ウィザード画面で指定できるすべてのオプションの説明は，第 5 章「UFT Java Add-in Extensibility Eclipse プラグインの使用」を参照してください。

---

1 New UFT Custom Support Class ウィザードを開きます。

a ［Eclipse Package Explorer］タブで，新しい UFT Java Add-in Extensibility プロジェクト **ImageControlsSupport** を選択します。［**File**］＞［**New**］＞［**Other**］を選択します。［New］ダイアログ・ボックスが開きます。

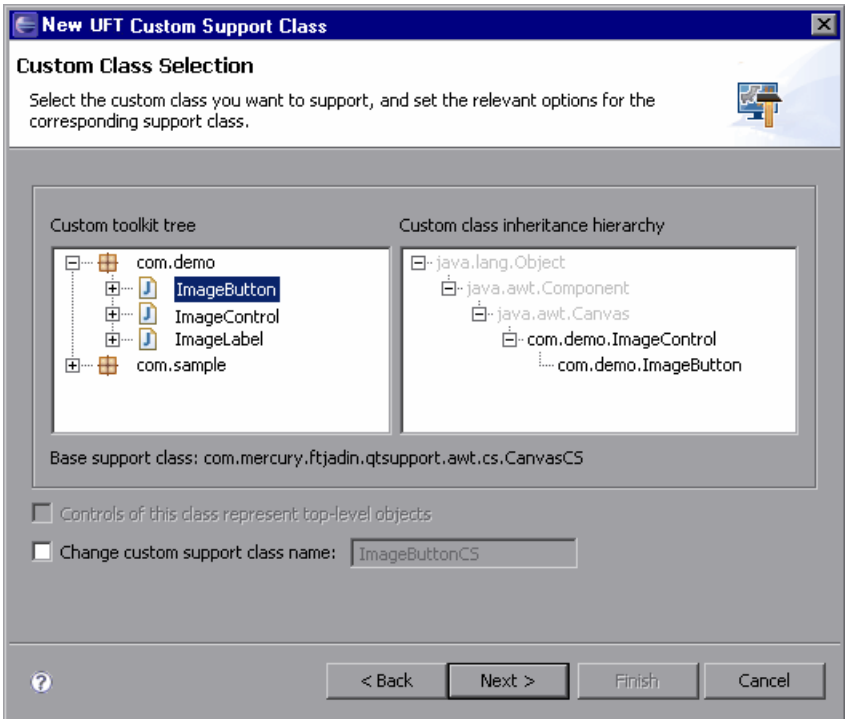
b **Unified Functional Testing** フォルダを展開し，**UFT Custom Support Class** を選択します。



c ［**Next**］をクリックします。［Custom Class Selection］画面が表示されます。

2 サポートするカスタム・クラスを選択し，サポート・クラスのオプションを設定します。

a **com.demo** パッケージを展開して，**ImageButton** クラスを選択します。



［**Custom toolkit tree**］表示枠では，「UFT Java Add-in Extensibility プロジェクトの新規作成」のステップ 3（180 ページ）で，［**Custom toolkit class locations**］用に選択した ImageControls プロジェクトの構造を表示することができます。**com.demo** パッケージには，カスタム・クラスを含む ImageControls カスタム・ツールキットが格納されます（170 ページ「このレッスンの準備」を参照）。

注：サンプル・アプリケーションを簡単に実行できるように，ImageControls サンプル・プロジェクトには，**com.sample** パッケージが含まれています。ImageControls プロジェクトのメイン・コンテンツは，**com.demo** パッケージに含まれている ImageControls カスタム・ツールキットです。

［**Custom class inheritance hierarchy**］表示枠では，選択した **ImageButton** クラスの階層を表示することができます。このクラスは同じツールキットに含まれる **ImageControl** クラスを拡張するため，黒で表示されます。

ImageControl カスタム・クラスはサポートされていませんが，**Canvas** クラスには一致するサポート・クラスがあります（**com.mercury.ftjadin.support.awt.cs** パッケージで提供されます）。そのため，作成中の **ImageButton** サポート・クラスの **Base support class** は **CanvasCS** になります。これが，新しいサポート・クラスによって拡張されるクラスです。

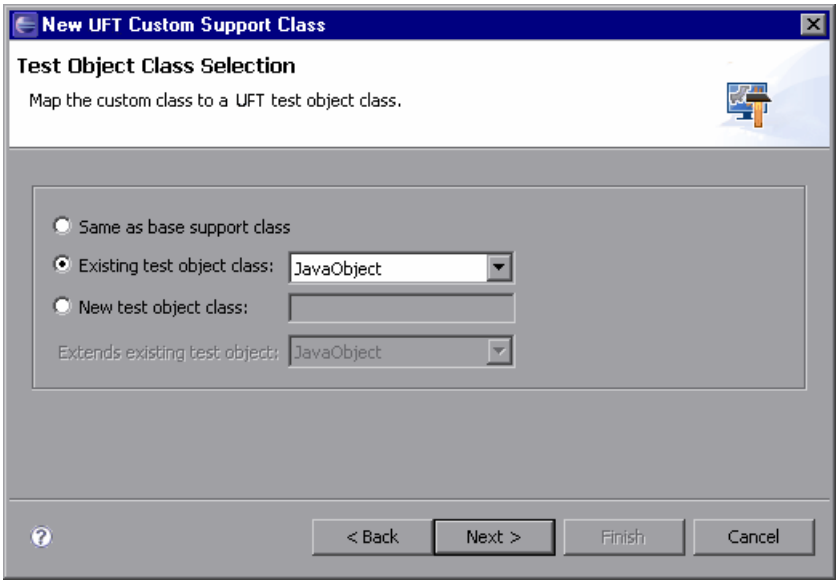
ImageButton クラスはコンテナ・クラスではないため，［**Controls of this class represent top-level objects**］オプションは無効になっています。

ImageButton サポート・クラスの名前は，標準設定で ImageButtonCS になります。標準設定の名前をそのまま使用することをお勧めします。

b ［**Next**］をクリックします。［Test Object Class Selection］画面が表示されます。

3 カスタム・コントロールを表すテスト・オブジェクト・クラスを選択します。

この画面では，カスタム・コントロールをテスト・オブジェクト・クラスにマッピングします。UFT のテストでは，カスタム・クラス・コントロールは，このテスト・オブジェクト・クラスのテスト・オブジェクトによって表されます。これは，カスタム・サポート・クラスを作成するときに行う最初の最も重要な判断です。



前の画面では，新しいサポート・クラスによって拡張されるサポート・クラスを決定しました。サポートを拡張するクラスにマッピングされたテスト・オブジェクトがカスタム・クラス用の論理テスト・オブジェクトである場合には，［**Same as base support class**］を選択します。ImageButtonCS クラスは CanvasCS を拡張しますが，CanvasCS のテスト・オブジェクト・クラスは ImageButton コントロールを適切に表すわけではありません。

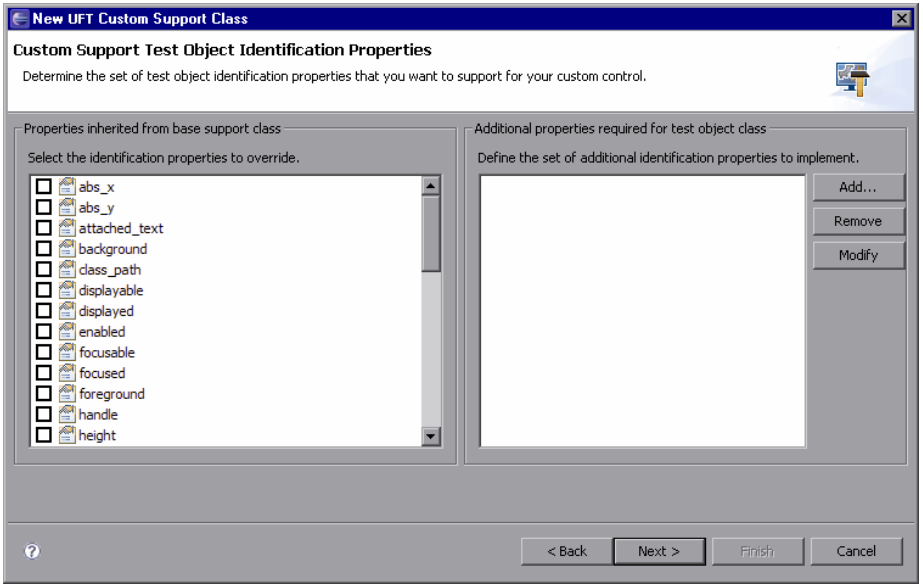
既存の JavaButton テスト・オブジェクトは，カスタム・サポートのニーズに対応しています。

a ［**Existing test object class**］オプションを選択し，リストから **JavaButton** を選択します。この既存のテスト・オブジェクトのリストには，UFT がサポートしているすべての Java オブジェクトが表示されます。カスタム・サポートに対する新しいテスト・オブジェクトを定義すると，このテスト・オブジェクトもリストに追加されます。

b ［**Next**］をクリックします。［Custom Support Test Object Identification Properties］画面が表示されます。

4 ImageButtonCS で実装するテスト・オブジェクト認識プロパティのセットを決定します。

この画面には，拡張しようとしているベース・サポート・クラスでサポートされている認識プロパティと，選択したテスト・オブジェクト・クラスで定義されていて，まだサポートされていない追加プロパティが表示されます。この画面では，新しい機能でサポートを実装またはオーバーライドするプロパティの選択，および新しいプロパティの追加を行うことができます。



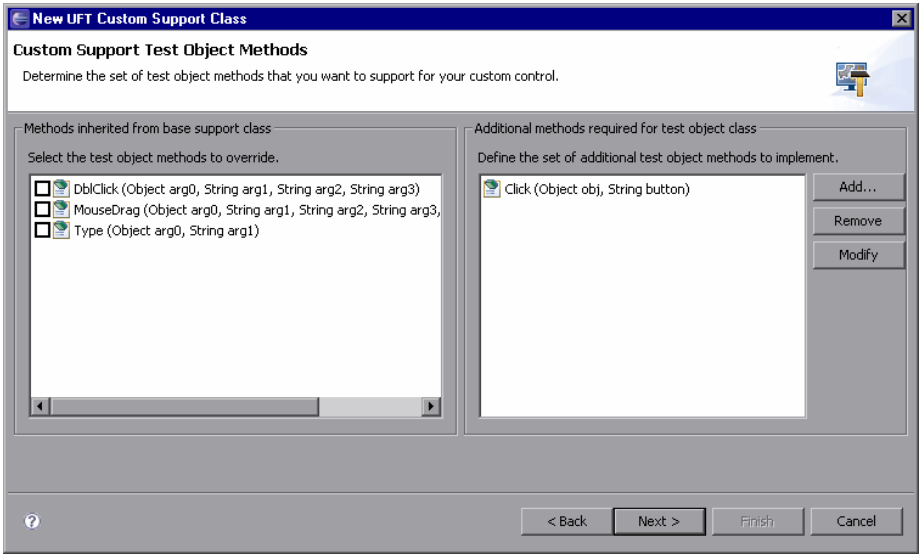
左側の表示枠には，CanvasCS でサポートが実装されている認識プロパティがすべて表示されます。これらの認識プロパティのサポートは，新しい ImageButtonCS サポート・クラスによって継承されます。このリストのほとんどのプロパティは，標準設定の実装で十分です。

a チェック・ボックスをクリックして **label** プロパティを選択します。ウィザードによるサポート・ファイルの生成が完了したら，このプロパティの既存のサポートをカスタム・コントロールのニーズに合わせたカスタム実装でオーバーライドします。

b ［**Next**］をクリックします。［Custom Support Test Object Methods］画面が表示されます。

5 ImageButtonCS で実装するテスト・オブジェクト・メソッドのセットを決定します。

この画面には，選択したオブジェクト・クラスで定義されているテスト・オブジェクト・メソッドが表示されます。この画面では，新しい機能でサポートを実装またはオーバーライドするメソッドの選択，および新しいメソッドの追加を行うことができます。



左側の表示枠には，CanvasCS でサポートが実装されている（選択したテスト・オブジェクト・クラスで定義された）テスト・オブジェクト・メソッドがすべて表示されます。これらのテスト・オブジェクト・メソッドのサポートは，ImageButtonCS によって継承されます。ImageButton では，この既存の実装で十分で，オーバーライドするメソッドを選択する必要はありません。

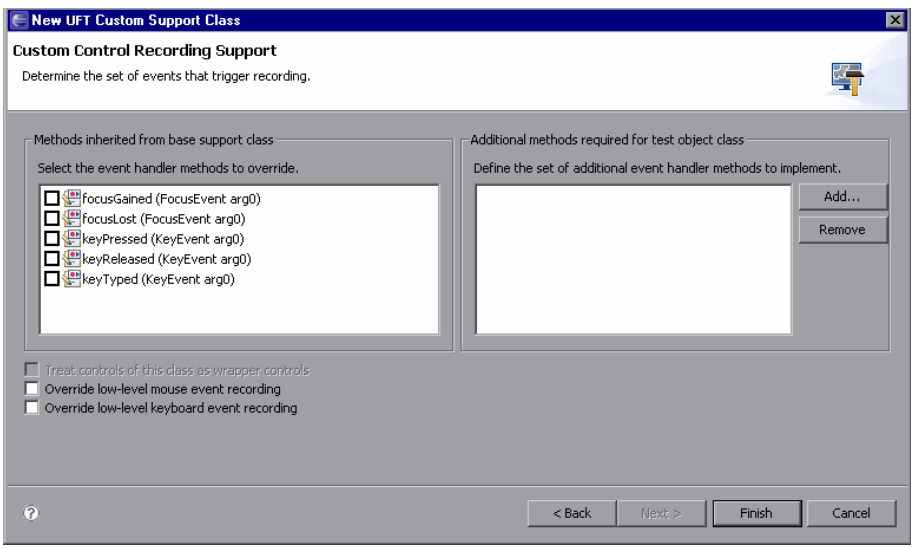
右側の表示枠では，JavaButton テスト・オブジェクト・クラス用に定義され，CanvasCS でサポートされていないテスト・オブジェクト・メソッドを確認することができます。

a 該当するメソッドが 1 つだけ存在しています（**Click(Object obj, String button)**）。ウィザードによるサポート・ファイルの生成が完了したら，このメソッドに対する ImageButton のサポートを実装します。

b ［**Next**］をクリックします。［Custom Control Recording Support］ウィザードの画面が表示されます。

6 ImageButton コントロールでの記録をサポートするため，リッスンするイベントのセットを決定します。

この画面には，拡張対象のサポート・クラスで実装されているイベント・リスナが表示されます。この画面では，新しい機能で実装をオーバーライドするのイベント・ハンドラ・メソッドの選択，および実装する新しいイベント・リスナの追加を行うことができます。



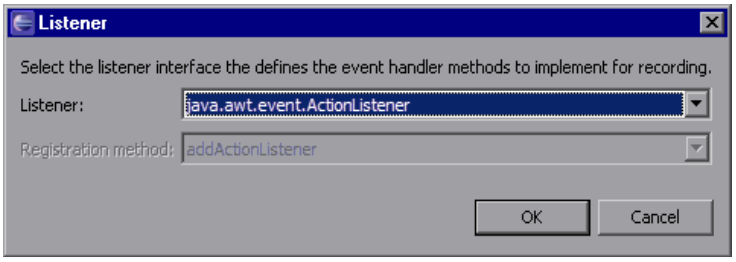
左側の表示枠では, CanvasCS で実装されているリスナを確認できます。ImageButtonCS カスタム・サポート・クラスでは，これらをオーバーライドする必要はありません。

右側の表示枠では，ImageButtonCS に追加するリスナを指定します。選択した各リスナは，サポート・クラスに追加されるイベント・ハンドラ・メソッドのセットを含んでいます。

次の手順を実行します。

a ［**Add**］をクリックして **ActionListener** を追加します。

［Listener］ダイアログ・ボックスが開きます。



b まだ java.awt.event.ActionListener が選択されていない場合は，［**Listener**］リストから **java.awt.event.ActionListener** を選択します。選択したリスナに複数の登録メソッドがある場合には，［**Registration method**］リストからメソッドを 1 つ選択します。

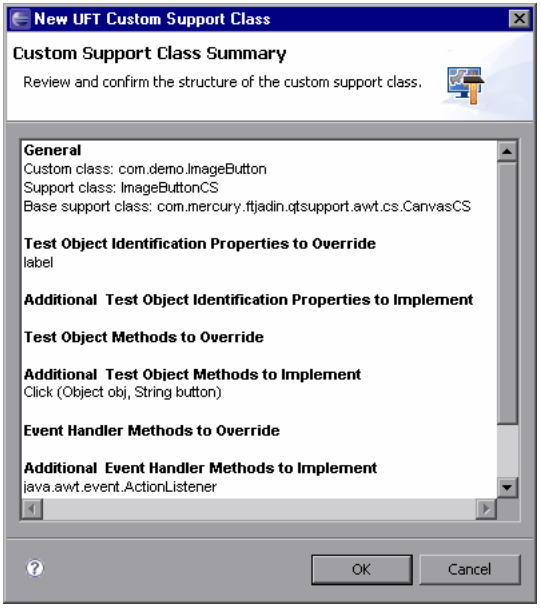
c ［**OK**］をクリックします。［Listener］ダイアログ・ボックスが閉じて, **ActionListener** とこれに含まれるすべてのイベント・ハンドラ・メソッドがウィザード画面の右側の表示枠のリストに追加されます。

d ［Custom Control Recording Support］画面で，［**Override low-level mouse event recording**］チェック・ボックスを選択して，記録したいイベントの代わりに低レベル・イベント（座標ベースの操作）が記録されないようにします。このオプションの詳細については，198 ページ「イベント記録のサポートについて」を参照してください。

e ［**Finish**］をクリックします。［Custom Control Support Class Summary］画面が表示されます。

7 カスタム・コントロール・サポート・クラスのサマリを表示します。

カスタム・サポート・クラスの計画内容を確認して［**OK**］をクリックします。



ImageControlsSupport プロジェクトでは，次の変更が行われています。

- 新しい UFT カスタム・サポート・クラス（ImageButtonCS）が **com.mercury.ftjadin.qtsupport.imagecontrols.cs** パッケージに作成されています。このファイルは，右側の表示枠のタブに表示されます。
- 新しい **ImageControlsTestObjects.xml** ファイルが **Configuration\TestObjects** フォルダに作成されています。
- **ImageControls.xml** ファイルが変更されています。

これらの変更の詳細については，196 ページ「新しいカスタム・サポートについて」を参照してください。

ImageButtonCS のファイル名（ImageButtonCS タブ）の横のアスタリスク（*****）は，ファイルの保存が完了していないことを示しています。ウィザードで行われる変更は相互に依存するため，不一致が生じないように必ず変更を保存する必要があります。［**File**］＞［**Save**］を選択するか，［**Save**］ボタンをクリックします。

新しいカスタム・サポートについて

新しい UFT Java Add-in Extensibility カスタム・ツールキット・サポート・セットは，以下で構成されています。

- 1 つのツールキット・サポート・クラス（**ImageControlsSupport**）：これは，**ImageControlsSupport** プロジェクトの作成時にウィザードによって作成され，変更されません。
- 1 つのツールキット設定ファイル（**ImageControls.xml**）：このファイルは，**ImageControlsSupport** プロジェクトの作成時にウィザードによって作成されます。このツールキットにサポート・クラスを追加するたびに更新されます。

  ImageControls.xml ファイルは，com.demo.ImageButton カスタム・コントロールをそのサポート・クラスである com.mercury.ftjadin.qtsupport.imagecontrols.cs.ImageButtonCS にマッピングするように更新されています。
- 1 つのテスト・オブジェクト設定ファイル（**ImageControlsTestObjects.xml**）: JavaButton テスト・オブジェクト・クラスに認識プロパティやテスト・オブジェクト・メソッドを追加しなかったため，このファイルに重要な情報は含まれていません。

  このファイルの構造の詳細については,『UFT Test Object Schema ヘルプ』(『Java Add-in Extensibility SDK ヘルプ』)を参照してください。
- カスタム・サポート・クラス（カスタム・クラスごとに 1 つ）。この例では，1 つのカスタム・サポート・クラス（**ImageButtonCS**）を作成しました。

  以降の項では，ウィザードによって **ImageButtonCS** クラス内に作成される要素について説明します。

ImageButtonCS クラスの基本事項について

UFT Java Add-in Extensibility ウィザードは，入力された仕様に基いてカスタム・サポート・クラスを作成し，それをツールキット・サポート設定ファイル内に登録します。

サポート・クラスの最も基本的な特性は，次の 2 つです。

- サポート・クラスによって拡張されるサポート・クラス
- カスタム・コントロールにマッピングされたテスト・オブジェクト・クラス

ImageButtonCS.java を開いて，ImageButton に対して作成されたサポート・クラスを確認します。

最初の宣言では，ウィザードで選択した内容を反映して，CanvasCS クラスが拡張されます。

```
public class ImageButtonCS extends CanvasCS implements ActionListener {
    private static final String DEBUG_IMAGEBUTTONCS =
"DEBUG_IMAGEBUTTONCS";
...
}
```

注：DEBUG_IMAGEBUTTONCS は，ログ・メッセージの出力を制御するために定義されます。詳細については，88 ページ「カスタム・サポート・クラスのログとデバッグ」を参照してください。

---

to_class_attr メソッドで実装された **to_class** プロパティでは，このカスタム・コントロールを表すために選択されたテスト・オブジェクト・クラスが定義されます。UFT は，このマッピングに基いてテスト・オブジェクトに対する認識プロパティとテスト・オブジェクト・メソッドのセットを決定します。

```
    public String to_class_attr(Object obj) {
        return "JavaButton";
    }
```

UFT でカスタム・コントロールの初期の認識を提供するのには，この実装で十分です。

認識プロパティとテスト・オブジェクト・メソッドのサポートについて

特定のカスタム・コントロールに対して学習できる各認識プロパティは，サポート・クラスの **<プロパティ名>_attr** というメソッドで表されます。このコントロールに対してサポートできる各テスト・オブジェクト・メソッドは，**<テスト・オブジェクト・メソッド名>_replayMethod** というメソッドで表されます。

ウィザードは，サポート・クラスを作成する際に，実装するように選択した認識プロパティとテスト・オブジェクト・メソッドに従って，必要なメソッドに対応するスタブを挿入します。

「UFT カスタム・サポート・クラスの新規作成」の手順 4（191 ページ）では，CanvasCS から継承されるラベル認識プロパティをオーバーライドするように選択したため，次のメソッド・スタブが追加されました。

```
public String label_attr(Object arg0) {
    return super.label_attr(arg0);
}
```

「UFT カスタム・サポート・クラスの新規作成」の手順 5（192 ページ）では，**Click(Object obj)** テスト・オブジェクト・メソッドを実装するように選択したため，次のメソッド・スタブが追加されました。

```
public Retval Click_replayMethod(Object obj, String button) {
    return Retval.NOT_IMPLEMENTED;
}
```

イベント記録のサポートについて

ImageButtonCS クラスでは，次の要素がイベント記録のベースとなります。

➤ 低レベル記録のオーバーライド（より高いレベルのイベントの記録が可能になる）：

```
protected Object mouseRecordTarget(MouseEvent e) {
    return null;
}
```

このメソッドが追加されるのは，「UFT カスタム・サポート・クラスの新規作成」の手順 6（193 ページ）で［**Override low-level mouse event recording**］チェック・ボックスを選択したためです。

➤ ImageButton コントロールでの登録用に ActionListener を列挙する

```
public ImageButtonCS() {
        addSimpleListener("ActionListener", "addActionListener",
                          "removeActionListener");
    }
```

このコンストラクタ・メソッドが追加されるのは，「UFT カスタム・サポート・クラスの新規作成」の手順 6（193 ページ）で，実装するリスナのリストに **ActionListener** を追加したためです。

このコンストラクタは **addSimpleListener** メソッドを呼び出して，**ActionListener** をカスタム・コントロールでの登録が必要なリスナのリストに追加します。

➤ アクション・イベント・ハンドラの実装：

```
public void actionPerformed(ActionEvent arg0) {
        try {
            if (!isInRecord())
                return;
            // TODO:Uncomment and edit the call to MicAPI.record
            // MicAPI.record(arg0.getSource(), <Operation>, new
            // String[]{<Parameters>});
            } catch (Throwable th) {
        }
    }
```

ウィザードでは，このメソッド・スタブが作成され，実際の実装は行われません。実装は 206 ページ「記録をサポートするためのイベント・ハンドラ・メソッドの実装」の手順に進んだ時点で行います。このメソッド・スタブには，**try ... catch** ブロックと **isInRecord** のチェックが含まれています。これは，このメソッドの推奨構造です。詳細については，68 ページ「記録オプションのサポート」を参照してください。

新しいカスタム・ツールキット・サポートのデプロイとテスト

ここでは，Eclipse で UFT の **Deploy Toolkit Support** コマンドを使用して，ImageControls ツールキット・サポートを UFT にデプロイします。現在，このツールキット内の唯一のコントロールである ImageButton コントロールがサポートされています。ツールキット・サポートはまだ完成していませんが，この時点までに作成したサポートをテストすることはできます。

1 ImageControls ツールキット・サポートを UFT にデプロイします。

［Eclipse Package Explorer］タブで，**ImageControlsSupport** プロジェクトを選択します。

［**Deploy Toolkit Support**］ボタンをクリックするか，［**UFT**］＞［**Deploy Toolkit Support**］を選択します。確認メッセージが表示されたら，［**Yes**］をクリックしてから［**OK**］をクリックします。

ツールキット設定ファイルとテスト・オブジェクト設定ファイルが，UFT のインストール・フォルダ内の関連するフォルダにコピーされます。このカスタム・サポートは，次回 UFT を起動してカスタム・アプリケーションを開始したときに利用できるようになります。

カスタム・ツールキット・サポートのデプロイの詳細については，78 ページ「カスタム・ツールキット・サポートのデプロイと実行」を参照してください。

2 UFT を起動し，Java Add-in とカスタム・ツールキット・サポートをロードします。

UFT を開きます。［アドイン マネージャ］ダイアログ・ボックスで，使用可能なアドインのリストに **Java** アドインの子として **ImageControls** が表示されます（［アドイン マネージャ］ダイアログ・ボックスが開かない場合は，『HP Unified Functional Testing アドイン・ガイド』の手順を参照してください）。

ImageControls のチェック・ボックスを選択し，［**OK**］をクリックします。UFT が開き，設計したサポートをロードします。

3 SampleApp アプリケーションを実行します。

UFT は，アプリケーションが開いたときにアプリケーションとの接続を確立します。そのため，SampleApp アプリケーションが開いている場合は，アプリケーションを終了してから再度実行する必要があります。

［Eclipse Package Explorer］タブで，SampleApp を右クリックします。［**Run As**］＞［**Java Application**］を選択します。SampleApp アプリケーションが開きます。

4 新しいカスタム・サポートをテストします。

173 ページ「ImageButton コントロールのサポートの計画」の手順 2 および 3 を繰り返し，UFT オブジェクト・スパイを使用して ImageButton コントロールを表示し，このコントロール上での **Click** 操作を記録します。

- UFT では，この ImageButton は ImageButton という名前の JavaButton として認識されます。
- 新しいサポート・クラス（ImageButtonCS）は，ベース・サポート・クラス（CanvasCS）から，JavaButton テスト・オブジェクト・クラスの定義に含まれない一部の認識プロパティを継承しました。これらのプロパティは，［Custom Support Test Object Identification Properties］画面（191 ページ）に表示されますが，UFT の［オブジェクト スパイ］ダイアログ・ボックスや［チェックポイントのプロパティ］ダイアログ・ボックスには表示されません。これらの認識プロパティにアクセスするには，**GetROProperty** メソッドを使用します。**GetROProperty** メソッドの詳細については，『HP Unified Functional Testing Object Model Reference』を参照してください。
- 低レベル記録のオーバーライドは完了していますが，**actionPerformed(ActionEvent arg0)** イベント・ハンドラ・メソッドの実装が未完了であるため，ボタンをクリックしても UFT では何も記録されません。

テスト・オブジェクトの名前の変更

ここでは，作成した計画（173 ページ「ImageButton コントロールのサポートの計画」）に基いて ImageButton コントロールの名前を認識するように，ImageButton コントロールの UFT サポートを拡張します。これを行うため，ObjectCS で実装される特別なプロパティ・メソッド（**tag_attr** および **attached_text_attr**）について学習します。

テスト・オブジェクトの名前は，オブジェクトの **tag** プロパティで決まります。AWT サポート・クラスはすべて ObjectCS を拡張します。ObjectCS は，実装している **tag_attr** メソッドを使用して指定された順序で一連のプロパティをチェックし，見つかった最初の有効値を返します。有効値は空でない値で，スペースを含みません。

ObjectCS クラスの **tag_attr** メソッドでは，記載された順序で次のプロパティがチェックされます。

- label
- attached_text（詳細は，下記を参照）
- 非修飾カスタム・クラス

label プロパティは，**label_attr** メソッドを使用してカスタム・サポート・クラスに実装されます。ImageButtonCS で，現在このメソッドはスーパークラスである CanvasCS と同様に Null を返します。

attached_text_attr メソッドも ObjectCS によって実装されます。このメソッドは，オブジェクトに隣接する静的テキスト・オブジェクトを検索して，そのテキストを返します。このメカニズムは，独自の説明テキストを持たないラベル付きのエディット・ボックスやリスト・ボックスのようなコントロールの場合に役に立ちます。

注 : UFT Custom Static-Text Support Class ウィザードを使用してカスタム静的テキスト・オブジェクトを認識するように UFT を学習させることができます。このウィザードには，Eclipse の［New］ダイアログ・ボックスからアクセスします。詳細については，211 ページ「カスタム静的テキスト・コントロールのサポート方法の学習」を参照してください。

ImageButton では，**attached_text** プロパティが空です。そのため，UFT で代替メカニズムを使用する必要があります。パッケージ名を含まないクラスの名前である**非修飾カスタム・クラス**が使用されます。この場合，カスタム・クラス **com.demo.ImageButton** は，テスト・オブジェクトの名前が ImageButton になります。

カスタム・コントロールのテスト・オブジェクトの名前を変更するには，サポート・クラスで **tag_attr** メソッドをオーバーライドせず，既存の実装を使用して，**label_attr** メソッドをオーバーライドします。

1 ImageButtonCS クラスで label_attr メソッドをオーバーライドします。

a Eclipse で, **ImageButtonCS.java** ファイルの **label_attr** メソッド・スタブで, return super.label_attr(arg0); を次のコードに置き換えます。これにより，ImageButton で使用される画像ファイルの名前（完全パスなし）が返されます。

```
ImageButton ib = (ImageButton)arg0;
String res = ib.getImageString();
if(res == null || res.length() == 0)
        return null;
int last = res.lastIndexOf('/');
if(last == -1)
        return res;
return res.substring(last+1);
```

b ［**Save**］ボタンをクリックするか，［**File**］＞［**Save**］を選択して変更内容を保存します。

注 : 変更したのは Java のクラス・ファイルのみで，設定ファイルは変更していないため，ツールキット・サポートを UFT に再度デプロイする必要はありません。

---

2 新しいカスタム・サポートをテストします。

173 ページ「ImageButton コントロールのサポートの計画」の手順 1 および 2 を繰り返してアプリケーションを実行し，UFT のオブジェクト・スパイで ImageButton コントロールを表示します。

注 : 開いている UFT セッション（ImageControls カスタム・ツールキット・サポートをロードした状態で実行中）を使用することは可能ですが，カスタム・サポートに対する変更を反映するには，SampleApp アプリケーションを終了して再度実行する必要があります。

UFT で，この ImageButton は **JavaExt1.gif** という名前の JavaButton として認識されます。

テスト・オブジェクト・メソッドに対するサポートの実装

ここでは，ImageButton の UFT サポートを拡張して，ボタン・クリックのテスト・オブジェクト・メソッドをサポートします。これを行うには，カスタム・サポート・クラスで **Click_replayMethod** を実装して，該当する MicAPI 関数を呼び出す必要があります。

1 ImageButton で Click メソッドの現在の機能をテストします。

a UFT で新規の GUI テストを作成し，**JavaExt1.gif** ボタンをオブジェクト・リポジトリに追加して，このオブジェクトを含むステップを追加します。これを行う手順については，『HP Unified Functional Testing ユーザーズ・ガイド』を参照してください。

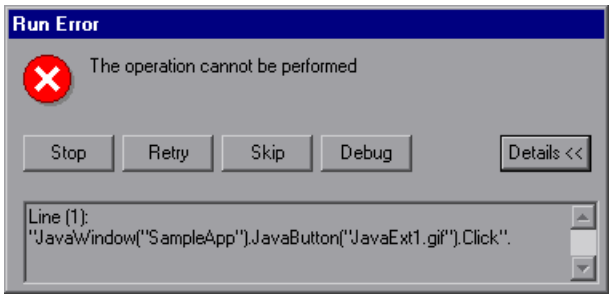
ImageButton は **JavaExt1.gif** という名前の JavaButton 項目（使用されるアイコンに注意）として認識されます。**Click** 操作は，この項目の標準操作です。この操作はすべての JavaButton 項目の標準操作です。

b ［**実行**］ボタンをクリックするか，［**実行**］＞［**実行**］を選択します。［実行］ダイアログ・ボックスが開きます。

c ［**新規実行結果フォルダ**］を選択します。結果フォルダ名をそのまま受け入れます。

d ［**OK**］をクリックして［実行］ダイアログ・ボックスを閉じます。

UFT でテストが実行され，エラー・メッセージが表示されます。メッセージ・ボックスの［**Details**］をクリックします。次の内容が表示されます。



このエラーが発生するのは，**Click** 操作を実行するために，UFT が Click_replayMethod を呼び出すためです。Click_replayMethod は，エラー・コード NOT_IMPLEMENTED を返すように ImageButtonCS で実装されています。

e ［**Stop**］をクリックして，テストの実行を停止します。

2 ImageButtonCS で Click_replayMethod メソッドを実装します。

a **Click_replayMethod** メソッド・スタブを，次のコードに置き換えます。

```
public Retval Click_replayMethod(Object obj, String button) {
        ImageButton ib = (ImageButton) obj;
        MicAPI.mouseClick((Object) ib, ib.getWidth() / 2,
                ib.getHeight() / 2);
        return Retval.OK;
    }
```

注：ウィザードで **ImageButtonCS.java** ファイルの作成時に，このコードのサポートに必要な import com.mercury.ftjadin.custom.MicAPI が自動的に追加されています。

---

b ［**Save**］ボタンをクリックするか，［**File**］＞［**Save**］を選択します。

注 : この実装は button 引数を無視します。この引数を考慮する実装をするために，別の **MicAPI.mouseClick** メソッドを呼び出すことが可能です。詳細については，『UFT Java Add-in Extensibility API Reference』(『Java Add-in Extensibility SDK ヘルプ』で利用可能)を参照してください。

---

3 新しいカスタム・サポートをテストします。

注 : 変更したのは Java のクラス・ファイルのみで，設定ファイルは変更していないため，ツールキット・サポートを UFT に再度デプロイする必要はありません。

---

a SampleApp アプリケーションを終了してから再度実行します。

b UFT で，上記の手順 1 で作成したテストを実行します。テストの実行が正常に終了します。テストで **Click** 操作を実行したときに，エディット・ボックスのクリック・カウンタが増加することを確認できます。

記録をサポートするためのイベント・ハンドラ・メソッドの実装

ImageButton コントロール上で記録をサポートするように計画しているため，このオブジェクト上での低レベル記録を抑止し，このコントロール上のアクション・イベントをリッスンするように登録しました。

ここでは，**MicAPI.record** を呼び出し，ImageButton オブジェクトで **Click** 操作を記録するために，**actionPerformed** リスナ・メソッドを実装します。

1 Click 操作を記録するため，actionPerformed リスナ・メソッドを実装します。

a Eclipse で，**ImageButtonCS.java** ファイルの **actionPerformed** リスナ・メソッド・スタブのコードを，次のように変更します。

```
public void actionPerformed(ActionEvent arg0)
{
   try {
         if (!isInRecord())
         return;
         MicAPI.record(arg0.getSource(), "Click");
       } catch (Throwable th)
        {
          MicAPI.logStackTrace(th);
        }
}
```

MicAPI.logStackTrace メソッドは，スタック・トレースを Java Add-in Extensibility のすべてのログ・メッセージを含むログ・ファイルに出力します。このメソッドにより，**actionPerformed** メソッドが誤って呼び出されたタイミングを決定することができます。詳細については，88 ページ「カスタム・サポート・クラスのログとデバッグ」を参照してください。

b ［**Save**］ボタンをクリックするか，［**File**］＞［**Save**］を選択します。

注 : 変更したのは Java のクラス・ファイルのみで，設定ファイルは変更していないため，ツールキット・サポートを UFT に再度デプロイする必要はありません。

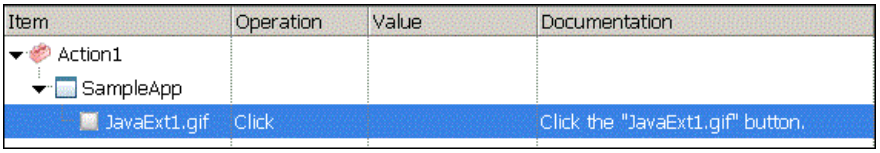
---

2 新しいカスタム・サポートをテストします。

a SampleApp アプリケーションを終了してから再度実行します。

b 新しい GUI テストを開き，［**記録**］ボタンをクリックするか，［**記録**］＞［**記録**］を選択します。［記録と実行環境設定］ダイアログ・ボックスが開いたら，［**開いているすべての Java アプリケーションでテストを記録して実行する**］オプションが選択されていることを確認し，［**OK**］をクリックします。SampleApp アプリケーションのボタンをクリックします。

JavaExt1.gif JavaButton でシンプルな **Click** 操作が記録されます。



これで，カスタム・サポートの計画時に決めた仕様に従って，ImageButton カスタム・コントロールが完全にサポートされました。

レッスンのまとめ

このレッスンでは，ImageButton コントロールのサポートを作成して，ImageButton コントロールを UFT で JavaButton テスト・オブジェクトとして認識されるようにしました。また，オブジェクト名を変更し，**Click** 操作をサポートしました。

- カスタム・サポート・クラスを 1 つ含むツールキット・サポート・プロジェクトの作成方法を学習しました。
- ツールキット・サポートを構成するファイルやメソッドを認識し理解できるようになりました。
- 次の認識プロパティ・サポート・メソッドの使用について学習しました。

  to_class_attr

  tag_attr

  label_attr

  attached_text_attr

  次の関数を使用しました。

  addSimpleListener

  mouseRecordTarget

  MicAPI.mouseClick

  MicApi.record

その他の情報

カスタム・ツールキット・サポート・セットの構造と内容の詳細は，35 ページ「カスタム・ツールキットのサポートの実装」を参照してください。

ツールキット設定ファイルの詳細については，『UFT Java Add-in Extensibility Toolkit Configuration Schema ヘルプ』（『Java Add-in Extensibility SDK ヘルプ』で利用可能）を参照してください。

MicAPI メソッドの詳細については，『UFT Java Add-in Extensibility API Reference』（『Java Add-in Extensibility SDK ヘルプ』で利用可能）を参照してください。

次のレッスンでは，静的テキスト・カスタム・コントロールのサポートの作成方法について学習します。通常，静的テキストのコントロールでは，特定の操作をサポートする必要はなく，隣接するコントロールのラベルを提供するだけです。静的テキスト・コントロールのサポート・クラスでは，特定のメソッドを実装するだけで，必要なサポートが提供されます。New UFT Custom Static-Text Support Class ウィザードは，静的テキスト・カスタム・コントロールに対するカスタム・サポートの作成専用です。

第 8 章

カスタム静的テキスト・コントロールのサポート方法の学習

このレッスンでは，ImageControls ツールキット内に ImageLabel コントロールのサポートを作成します。ImageLabel コントロールには，サポートする必要のある認識プロパティやテスト・オブジェクト・メソッドはありません。ImageLabel コントロールの主要な目的は，ラベルとして使用することです。そのため，ImageLabel のサポートを静的テキスト・オブジェクトとして作成します。

このレッスンは，ImageControls カスタム・ツールキットに対するカスタム・ツールキット・サポート・セットを作成する，169 ページ「シンプルなコントロールのサポート方法の学習」のレッスンをすでに行っていることが前提になっています。このレッスンでは，同じカスタム・ツールキット・サポート・セットに別のサポート・クラスを作成します。

このレッスンは，次の内容で構成されています。

このレッスンの準備

169 ページ「シンプルなコントロールのサポート方法の学習」のレッスンの準備で Eclipse で作成した ImageControls Java プロジェクトには，ImageLabel クラスが含まれています。そのレッスンで実行したサンプル・アプリケーションでは，ImageLabel コントロールが（ImageButton の左側に）表示されます。このアプリケーションにおける ImageLabel コントロールの目的は，独自のラベル認識プロパティを持たない，その下のテキスト・ボックスにラベルを提供することです。

Eclipse を開いて **ImageControls** Java プロジェクトを探します。

ImageLabel コントロールのサポートの計画

ここでは，ImageLabel コントロールと隣接するテキスト・ボックスの現在の UFT サポートを分析し，UFT でコントロールを認識する方法を決定し，それに基づいて 216 ページ「カスタム・クラス・サポート計画のチェックリスト」に記入します。

1 UFT を起動し，Java Add-in とカスタム・ツールキット・サポートをロードします。

a UFT を開きます。［アドイン マネージャ］ダイアログ・ボックスで，使用可能なアドインのリストに（前のレッスンでサポートを作成した）**ImageControls** が **Java** アドインの子として表示されます（［アドイン マネージャ］ダイアログ・ボックスが開かない場合は，『HP Unified Functional Testing アドイン・ガイド』の手順を参照してください）。

b **Java** と **ImageControls** の両方のチェック・ボックスが選択されていることを確認し，［**OK**］をクリックします。

2 SampleApp アプリケーションを実行します。

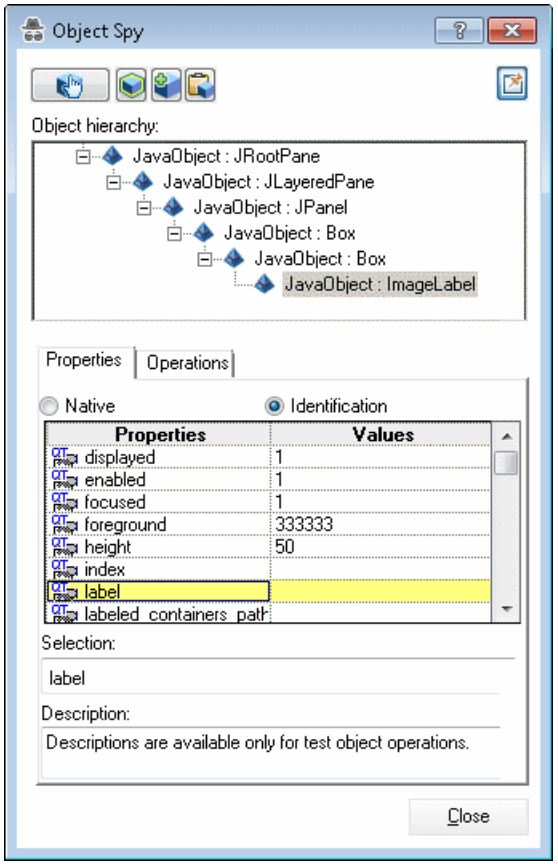
［Eclipse Package Explorer］タブで，**SampleApp** を右クリックします。［**Run As**］＞［**Java Application**］を選択します。SampleApp アプリケーションが開きます。

3 オブジェクト・スパイを使用して ImageLabel のプロパティを表示します。

a UFT で，GUI テストを開き，［**ツール**］＞［**オブジェクト スパイ**］を選択するか，［**オブジェクト スパイ**］ツールバー・ボタンをクリックして，［オブジェクト スパイ］ダイアログ・ボックスを開きます。［**プロパティ**］タブをクリックします。

b ［オブジェクト スパイ］ダイアログ・ボックスで，指差しアイコンをクリックしてから，SampleApp アプリケーションの左側にある画像をクリックします。

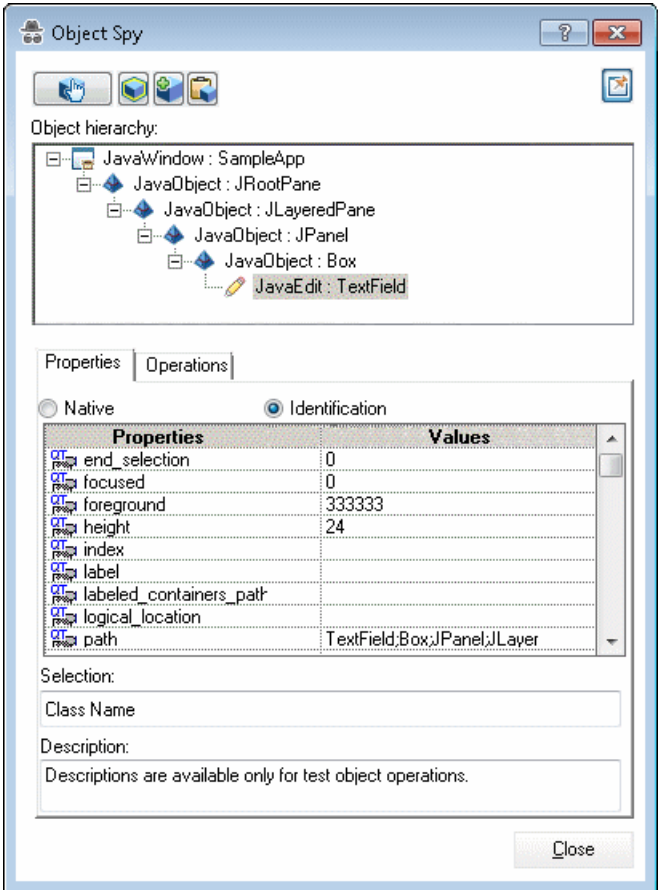
ImageLabel コントロールは，UFT で認識されないカスタム・クラスに基づいています。ボタンは **ImageButton** という名前の汎用の **JavaObject** オブジェクトとして認識されるため，表示されるアイコンは標準の JavaObject クラスのアイコンになります。**label** 認識プロパティは空です。



4 オブジェクト・スパイを使用してテキスト・ボックスのプロパティを表示します。

a ［オブジェクト スパイ］ダイアログ・ボックスで，指差しアイコンをクリックしてから，SampleApp アプリケーション内のテキスト・ボックスをクリックします。

このテキスト・ボックスは標準の TextField クラスに基づいているため，UFT はこれを **JavaEdit** テスト・オブジェクトとして認識します。ただし，**label** 認識プロパティが空であるため，UFT は隣接するコントロールを静的テキスト・コントロールとして認識しません。そのため，JavaEdit テスト・オブジェクトには，クラス名 **TextField** に基づいた名前が付けられます。



b オブジェクト・スパイを閉じます。

5 カスタム・クラス・サポート計画のチェックリストを完成させます。

ImageLabel コントロールは，静的テキスト・コントロールです。これを UFT で認識し，ImageLabel の **label** プロパティを，独自の **label** プロパティを持たない隣接するコントロールの **attached text** として使用する必要があります。

ImageLabel は画像ファイルを表示し，オプションでテキストを付加することができます。ImageLabel コントロールがテキストを表示しない場合，ImageLabel コントロールを表すテスト・オブジェクトの名前は, ImageLabel コントロールで表示される画像ファイルの名前をベースにすることができます。

ImageLabel そのものには，UFT GUI テストで認識する必要のあるその他の認識プロパティやテスト・オブジェクト・メソッドはありません。また，ImageLabel コントロールでは操作を記録する必要もありません。

次のページに上記の情報に基いて作成したチェックリストを示します。

カスタム・クラス・サポート計画のチェックリスト

☑	カスタム・クラス・サポート計画のチェックリスト
☑	カスタム・クラスに UFT カスタム・サポートが提供されていないスーパークラスが存在するか？ **いいえ**
☑	存在する場合，階層内の上位のコントロールに対するサポートを最初に拡張すべきか？ **該当なし**
☑	UFT がインストールされているコンピュータ上でカスタム・コントロールを実行するアプリケーションが存在するか？ **はい**
☑	このカスタム・コントロール・クラスのソースが存在する場所： ImageControls という Eclipse プロジェクト
☑	カスタム・コントロールに適合する既存の Java テスト・オブジェクトはどれか？ JavaStaticText
☑	存在しない場合に作成する新しい Java テスト・オブジェクト・クラス :**該当なし** ➤ 新しいテスト・オブジェクト・クラスによって拡張されるもの：(標準 JavaObject) ➤ アイコン・ファイルの場所（オプション）： ➤ 説明用の認識プロパティ： ➤ 標準設定のテスト・オブジェクト・メソッド：
☑	カスタム・コントロールはトップレベル・オブジェクトか？ **いいえ**
☑	カスタム・コントロールはラッパーか？ **いいえ**
☑	テスト･オブジェクトの名前の基準を指定: **テキストまたは(テキストが存在しない場合は)画像ファイル名**
☑	サポートする認識プロパティを列挙し，標準設定のチェックポイントのプロパティをマークする： **特になし**
☑	サポートするテスト・オブジェクト・メソッドを列挙する（必要に応じて，引数と戻り値も記入する）： **特になし**
☑	記録をサポートするか？(AWT ベースのみ) **いいえ**
☑	その場合，記録をトリガするイベントを列挙： **該当なし**

UFT のカスタム静的テキスト・サポート・クラスの作成

「シンプルなコントロールのサポート方法の学習」のレッスンでは，ImageControlsSupport UFT Java Add-in Extensibility プロジェクト（178 ページを参照）を作成しました。このプロジェクトでは, ImageButton コントロールのカスタム・サポート・クラスを作成しました。

ここでは，ImageLabel コントロールをサポートするために，同じプロジェクトに別のカスタム・サポート・クラスを作成します。

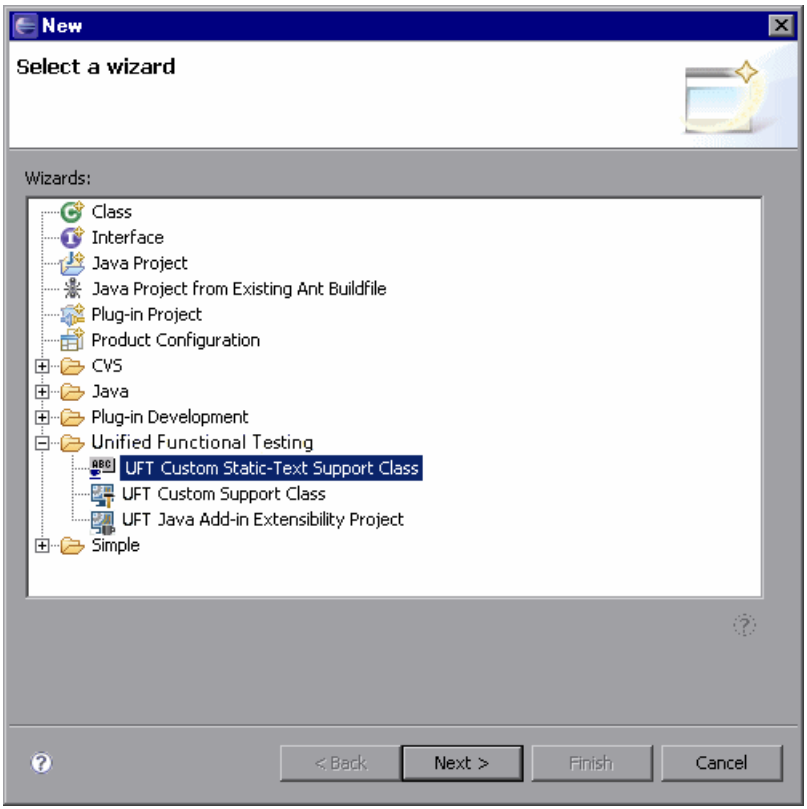
ほとんどの場合，静的テキスト・コントロールには，UFT のテストで認識する必要のある認識プロパティやテスト・オブジェクト・メソッドはありません。また，通常，静的テキスト・コントロールでは操作を記録する必要もありません。そのため，UFT Java Add-in Extensibility Eclipse プラグインには，静的テキスト・コントロール用のサポート・クラスを作成する特別なウィザードが用意されています。

このウィザードでは，静的テキスト・コントロールとしてサポートする ImageLabel クラスを選択するだけで，必要なすべてのメソッドを含む新しいサポート・クラスが作成されます。新しいサポート・クラスが作成された後で，作成されたメソッドを修正してサポートを完成させます。

1 New UFT Custom Static-Text Support Class ウィザードを起動します。

a ［Eclipse Package Explorer］タブで，UFT Java Add-in Extensibility プロジェクト **ImageControlsSupport** を選択します。［**File**］＞［**New**］＞［**Other**］を選択します。［New］ダイアログ・ボックスが開きます。

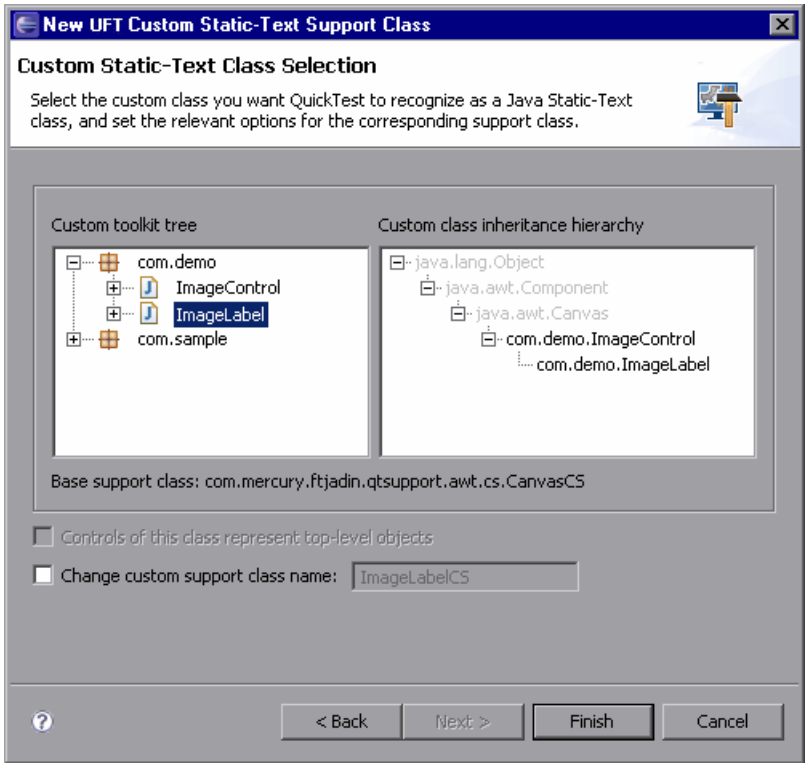
b **Unified Functional Testing** フォルダを展開し，**UFT Custom Static-Text Support Class** を選択します。



c ［**Next**］をクリックします。［Custom Class Selection］画面が表示されます。

2 サポートするカスタム・クラスを選択し，サポート・クラスのオプションを設定します。

a **com.demo** パッケージを展開して，**ImageLabel** クラスを選択します。



ImageControls カスタム・ツールキットでクラスのサポートを作成しているため，上記の［**Custom toolkit tree**］表示枠は，「シンプルなコントロールのサポート方法の学習」のレッスンのものと似ています（「UFT カスタム・サポート・クラスの新規作成」の手順 2（188 ページ）を参照）。この［**Custom toolkit tree**］には，サポートするように選択できるクラスが一覧表示されます。すでにサポートを作成しているため，ImageButton クラスはこのリストに表示されません。

［**Custom class inheritance hierarchy**］表示枠では，すでに選択している **ImageLabel** クラスの階層を表示することができます。このクラスは同じツールキットに含まれる **ImageControl** クラスを拡張するため，黒で表示されます。

ImageControl カスタム・クラスはサポートされていませんが，**Canvas** クラスには一致するサポート・クラスがあります（**com.mercury.ftjadin.support.awt.cs** パッケージで提供されます）。そのため，作成中の **ImageLabel** サポート・クラスの **Base support class** は **CanvasCS** になります。これが，新しいサポート・クラスによって拡張されるクラスです。

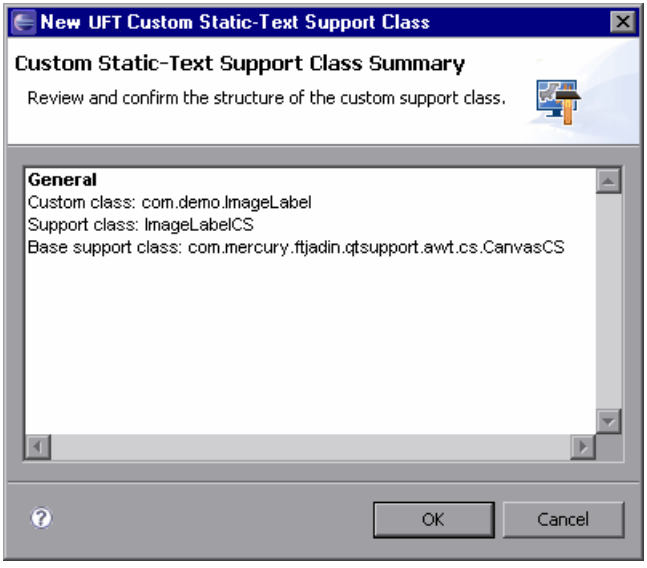
ImageLabel クラスはコンテナ・クラスではないため，［**Controls of this class represent top-level objects**］オプションは無効になっています。

ImageLabel サポート・クラスの名前は，標準設定で ImageLabelCS になります。標準設定の名前をそのまま使用することをお勧めします。

b ［**Finish**］をクリックします。［Custom Static-Text Support Class Summary］画面が表示されます。

3 カスタム静的テキスト・コントロール・サポート・クラスのサマリを表示します。

カスタム・サポート・クラスの計画内容を確認して［**OK**］をクリックします。



ImageControlsSupport プロジェクトでは，次の変更が行われています。

- ImageLabel カスタム・クラスをそのサポート・クラスである ImageLabelCS にマッピングするように，**ImageControls.xml** ファイルが変更されています。
- 新しい UFT カスタム・サポート・クラス（ImageButtonCS）が **com.mercury.ftjadin.qtsupport.imagecontrols.cs** パッケージの **ImageLabelCS.java** ファイルに作成されています。このファイルは，右側の表示枠のタブに表示されます。

  ImageLabelCS クラスの内容の詳細については，221 ページ「新しいカスタム静的テキスト・サポート・クラスについて」を参照してください。

ImageLabelCS のファイル名（ImageLabelCS タブ）の横のアスタリスク（*****）は，ファイルの保存が完了していないことを示します。ウィザードで行われる変更は相互に依存するため，不一致が生じないように必ず変更を保存する必要があります。［**File**］＞［**Save**］を選択するか，［**Save**］ボタンをクリックします。

新しいカスタム静的テキスト・サポート・クラスについて

新しい **ImageLabelCS.java** ファイルの内容を検証します。ImageLabelCS カスタム静的テキスト・サポート・クラスは CanvasCS を拡張します。

この新しいサポート・クラスでは，ウィザードによって次のメソッドに対応するスタブが作成されています。

- **class_attr**：文字列 static_text を返します。

  これにより，UFT は ImageLabel コントロールが JavaStaticText オブジェクトであることを認識します。このため，添付テキストを検索する UFT のメカニズムで ImageLabel の **label** プロパティを隣接するコントロールの **attached text** として使用することができます。

- **label_attr**：スーパークラス（この例では，CanvasCS）のラベル・プロパティを返します。

  このメソッドは，ImageLabel の **label** 認識プロパティを定義します。この認識プロパティのテキストは，隣接するコントロールの **attached text** で使用されます。このメソッド・スタブには，適切なテキストを返すように実装する必要があることを示すコメントが記載されています。

➤ **tag_attr** : このメソッドは，静的テキスト・テスト・オブジェクトの名前を表す **tag** プロパティをサポートします。

「シンプルなコントロールのサポート方法の学習」のレッスンの 202 ページ「テスト・オブジェクトの名前の変更」で，**tag** プロパティの実装方法を学習しました。ウィザードによって作成されるサポート・クラスの **tag_attr** メソッドは，super.tag_attr(obj) の末尾にサフィックス (st) を追加したものを返します。つまり，静的テキスト・テスト・オブジェクトの名前は，通常のテスト・オブジェクト（ラベル，添付テキスト，または非修飾クラス名）の場合と同じロジックを使用し，末尾に (st) を付加することによって得られます。

➤ **value_attr** : **label** プロパティを返します。

value プロパティは，コントロールのテスト・オブジェクトの状態を表します。静的テキスト・コントロールの場合，**label** プロパティがこの状態を表します。

これらの特別な認識プロパティの詳細については，64 ページ「一般的な認識プロパティ・サポート・メソッド」を参照してください。

新しいカスタム静的テキスト・サポート・クラスのデプロイとテスト

ここでは，Eclipse で UFT の **Deploy Toolkit Support** コマンドを使用して，ImageControls ツールキット・サポートを UFT にデプロイします。これにより，すでにサポートをデプロイ済みの ImageButton コントロールに加えて，ImageLabel のサポートが UFT に追加されます。ImageLabel のサポートはまだ完成していませんが，この時点までに作成したサポートをテストすることはできます。

1 ImageControls ツールキット・サポートを UFT にデプロイします。

［Eclipse Package Explorer］タブで，**ImageControlsSupport** プロジェクトを選択します。

［**Deploy Toolkit Support**］ボタンをクリックするか，［**UFT**］>［**Deploy Toolkit Support**］を選択します。確認メッセージが表示されたら，［**Yes**］をクリックしてから［**OK**］をクリックします。

ツールキット設定ファイルとテスト・オブジェクト設定ファイルが，UFT のインストール・フォルダ内の関連するフォルダにコピーされます。このカスタム・サポートは，次にカスタム・アプリケーションを開始したときに利用できるようになります（テスト・オブジェクト設定ファイルは変更されていないため，UFT を再度開く必要はありません）。

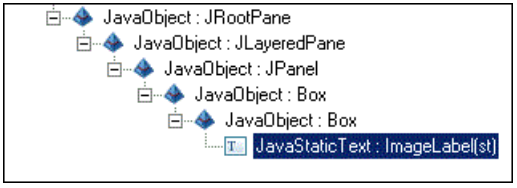
カスタム・ツールキット・サポートのデプロイの詳細については，78 ページ「カスタム・ツールキット・サポートのデプロイと実行」を参照してください。

2 新しいカスタム・サポートをテストします。

212 ページ「ImageLabel コントロールのサポートの計画」の手順 2 および 3 を繰り返してアプリケーションを実行し，UFT のオブジェクト・スパイで ImageLabel コントロールとテキスト・ボックスを表示します。

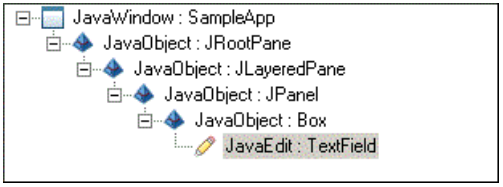
注： UFT は，アプリケーションが開いたときにアプリケーションとの接続を確立します。そのため，開いている UFT セッション（ImageControls ツールキット・サポートをロードした状態で実行中）を使用して変更をテストすることは可能ですが，SampleApp アプリケーションを終了して再度実行する必要があります。

UFT は ImageLabel を **ImageLabel(st)** という名前の **JavaStaticText** オブジェクトとして認識します。



ImageLabelCS によって拡張される CanvasCS は，**label** 認識プロパティに対するサポートを提供しません。そのため，ImageLabel の **label** プロパティは空になります（**attached text** プロパティと同様）。その結果，スーパークラスの **tag** プロパティは ImageLabel のクラス名を返し，ImageLabel の **tag** プロパティは **ImageLabel(st)** になります。

UFT は, 引き続きテキスト・ボックスを **TextField** という名前（クラス名）の **JavaEdit** テスト・オブジェクトとして識別します。これは，隣接する静的テキスト・オブジェクト **ImageLabel** の **label** プロパティが空のままであるためです。



静的テキスト・コントロールのサポートの完成

ここでは，ImageLabel で使用する画像ファイルの名前を返すため，ImageLabelCS クラスで **label_attr** メソッドを実装します。これにより, UFT で ImageLabel の **label** プロパティを隣接するコントロールの **attached text** として使用することが可能になります。さらに，ImageLabel の **label** プロパティを実装して，ImageLabel テスト・オブジェクトに具体的な名前を提供します。

1 ImageLabelCS クラスで label_attr メソッドを実装します。

a Eclipse で，**ImageLabelCS.java** ファイルの **label_attr** メソッド・スタブで，return super.label_attr(obj); を次のコードに置き換えます。

```
ImageLabel il = (ImageLabel)obj;
String res = il.getText();
if(res != null && res.length() > 0)
        return res;
res = il.getImageString();
if(res == null || res.length() == 0)
        return null;
int last = res.lastIndexOf('/');
if(last == -1)
        return res;
return res.substring(last+1);
```

ラベル認識プロパティは，ラベル上のテキスト（存在する場合）または ImageLabel で使用する画像ファイルの名前（完全パスなし）を返します。

b ［**Save**］ボタンをクリックするか，［**File**］＞［**Save**］を選択して変更内容を保存します。

注 : 変更したのは Java のクラス・ファイルのみで，設定ファイルは変更していないため，ツールキット・サポートを UFT に再度デプロイする必要はありません。

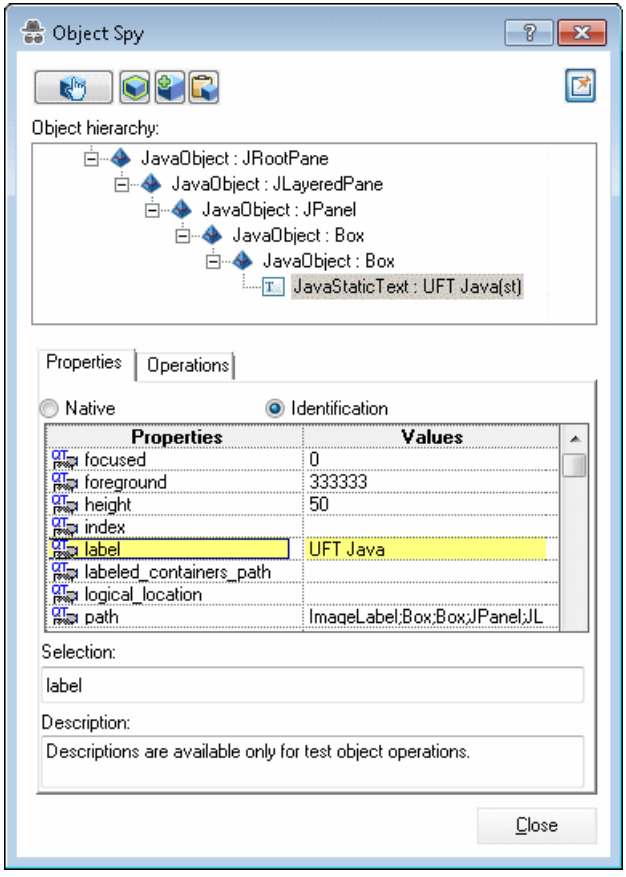
---

2 新しいカスタム・サポートをテストします。

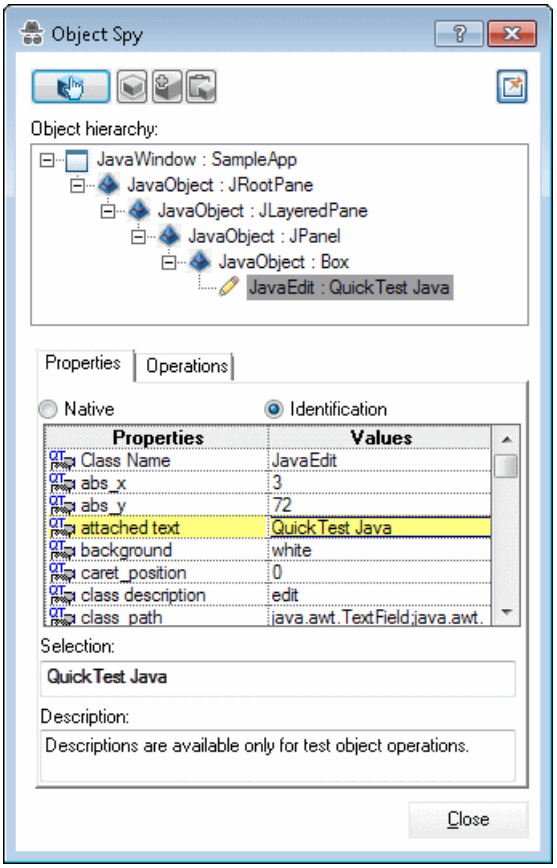
212 ページ「ImageLabel コントロールのサポートの計画」の手順 2 および 3 を繰り返してアプリケーションを実行し，UFT のオブジェクト・スパイで ImageLabel コントロールとテキスト・ボックスを表示します。

注 : テスト・オブジェクト設定ファイルは変更していません。そのため，開いている UFT セッション（ImageControls カスタム・ツールキット・サポートをロードした状態で実行中）を使用することはできますが，カスタム・サポートに対する変更を反映するには，SampleApp アプリケーションを終了して再度実行する必要があります。

UFT で，ImageLabel は **UFT Java(st)** という名前の **JavaStaticText** テスト・オブジェクトとして認識され，**label** プロパティは **UFT Java** になります。



UFT で，テキスト・ボックスは **UFT Java** という名前の **JavaEdit** テスト・オブジェクトとして認識されます。**JavaEdit** テスト・オブジェクトの **label** プロパティは空です。ImageLabel の **label** プロパティは，JavaEdit の **attached text** プロパティのテキストを提供します。これは，テスト・オブジェクト名として使用されます。



注：SampleApp アプリケーションを修正して imageLb.setText("QuickTest Java"); の行を削除した場合，ImageLabel はテキストを表示しません。その場合，UFT は ImageLabel を **mercury.gif(st)** という名前の **JavaStaticText** テスト・オブジェクトとして認識し，**label** プロパティは **mercury.gif** になります。また，UFT は，テキスト・ボックスを **mercury.gif** という名前の **JavaEdit** テスト・オブジェクトとして認識します。

これで，カスタム・サポートの計画時に決めた仕様に従って，ImageLabel 静的テキスト・カスタム・コントロールが完全にサポートされました。これで，ImageControls ツールキットのサポートが完成しました。このサポートの既製のサンプルは，**<UFT Java Add-in Extensibility SDK インストール・フォルダ>\samples\ ImageControlsSupport** フォルダにあります （このサンプルを手動でデプロイする場合は，前もって Java クラスをコンパイルする必要があります)。

ImageControls ツールキット・サポートの最適化

ImageLabel クラスの **label** 認識プロパティで使用した実装は, ImageButton クラスの **label** 認識プロパティの実装とよく似ています。これらのクラスはどちらも ImageControl クラスを拡張するため，ImageControl（ImageControlCS）のサポート・クラスで **label** 認識プロパティのサポートを実装する方が適切であった可能性があります。

これは, ImageButton コントロールと ImageLabel コントロールのサポートを計画する際の 216 ページ「カスタム・クラス・サポート計画のチェックリスト」の 2 番目の質問に対する回答が「**はい**」だった（最初に階層の上位のコントロールのサポートを拡張すべきだった）ことを意味します。その場合は，ImageButtonCS と ImageLabelCS は ImageControlCS を拡張し, ImageLabelCS で継承された **label_attr** メソッドをオーバーライドして **label** プロパティを調整することになります。

次の各項では，**label_attr** メソッドの実装が重複しないように，ImageControls ツールキット・サポート・セットを変更します。この変更によって，サポートの機能が影響を受けることはありません。ImageControlCS サポート・クラスを作成し，ImageControlCS を拡張するように ImageButtonCS と ImageLabelCS を変更します。

ImageControl カスタム・クラスのサポートの作成

ここでは，ImageControlsSupport プロジェクトで ImageControl クラス用のカスタム・サポート・クラスを作成します。

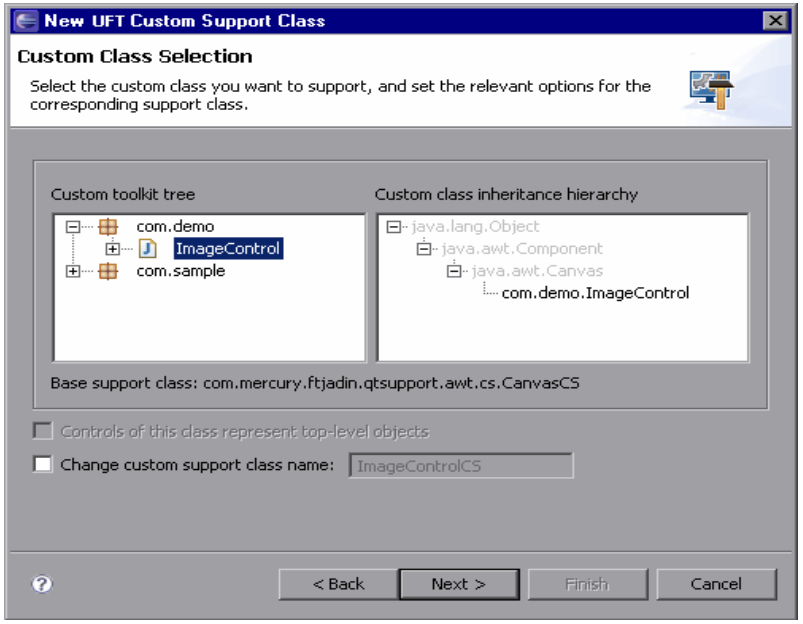
1 New UFT Custom Support Class ウィザードを開きます。

a ［Eclipse Package Explorer］タブで，新しい UFT Java Add-in Extensibility プロジェクト **ImageControlsSupport** を選択します。［**File**］＞［**New**］＞［**Other**］を選択します。［New］ダイアログ・ボックスが開きます。

b **Unified Functional Testing** フォルダを展開し，**UFT Custom Support Class** を選択して，［**Next**］をクリックします。［Custom Class Selection］画面が表示されます。

2 サポートするカスタム・クラスを選択し，サポート・クラスのオプションを設定します。

a **com.demo** パッケージを展開して，**ImageControl** クラスを選択します。



［**Custom toolkit tree**］表示枠で，ImageControl クラスが **com.demo** パッケージ内のサポートされていない唯一のクラスであることがわかります。

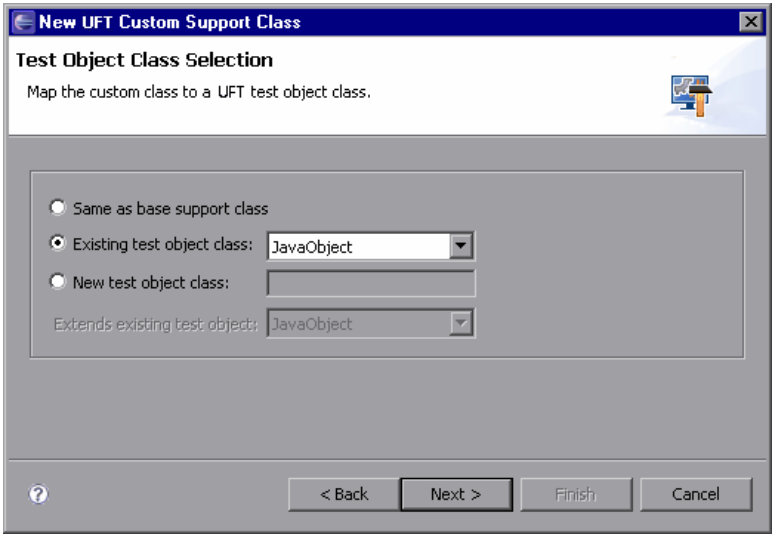
［**Custom class inheritance hierarchy**］表示枠では，すでに選択している **ImageControl** クラスの階層を表示することができます。**ImageControl** クラスは **java.awt.Canvas** を拡張します。そのため，作成中の ImageControl サポート・クラスの **Base support class** は **CanvasCS** になります。

ImageControl サポート・クラスでは，標準設定の名前 **ImageControlCS** をそのまま使用します。

b ［**Next**］をクリックします。［Test Object Class Selection］画面が表示されます。

3 カスタム・コントロールを表すテスト・オブジェクト・クラスを選択します。

ImageControlCS サポート・クラスを作成する目的は，実際のコントロールをサポートするためではなく，ほかのサポート・クラスのベース・サポート・クラスとして使用するためです。そのため，ImageControl カスタム・クラスをどのテスト・オブジェクト・クラスにマッピングするかは重要ではありません。



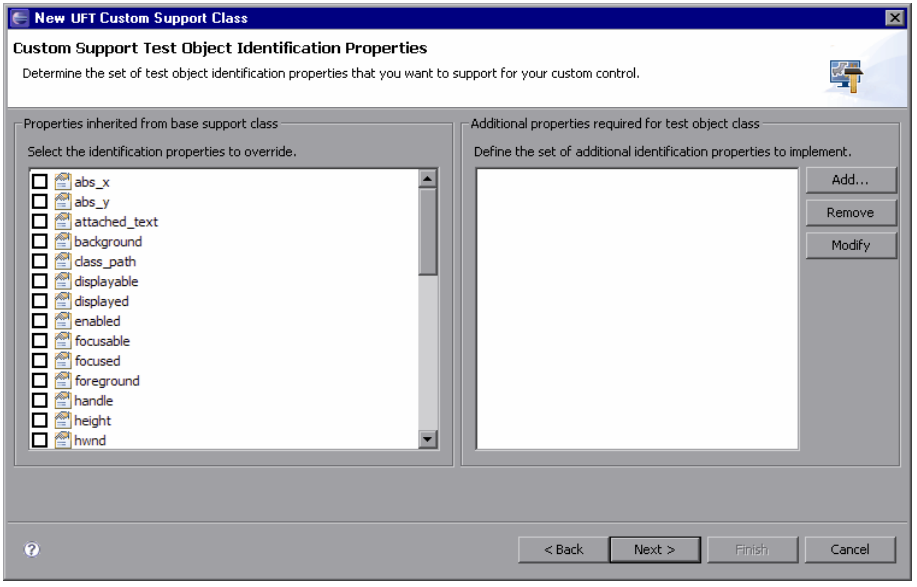
次の手順を実行します。

a ［**Same as base support class**］を選択します。これを選択すると，java.awt.Canvas にマッピングされるテスト・オブジェクト・クラスに ImageControl カスタム・クラスがマッピングされます。直接的なマッピングは行われません。新しいサポート・クラスで **to_class_attr** メソッドは実装されず，ベース・サポート・クラスから継承されます。

b ［**Next**］をクリックします。［Custom Support Test Object Identification Properties］画面が表示されます。

4 ImageControlCS で実装するテスト・オブジェクト認識プロパティのセットを決定します。

この画面には，拡張しようとしているベース・サポート・クラスでサポートされている認識プロパティと，選択したテスト・オブジェクト・クラスで定義されていて，まだサポートされていない追加プロパティが表示されます。



左側の表示枠には，CanvasCS でサポートが実装されている認識プロパティがすべて表示されます。これらの認識プロパティのサポートは，新しい ImageControlCS サポート・クラスによって継承されます。ここでは，新しい機能でサポートをオーバーライドするプロパティを選択することができます。

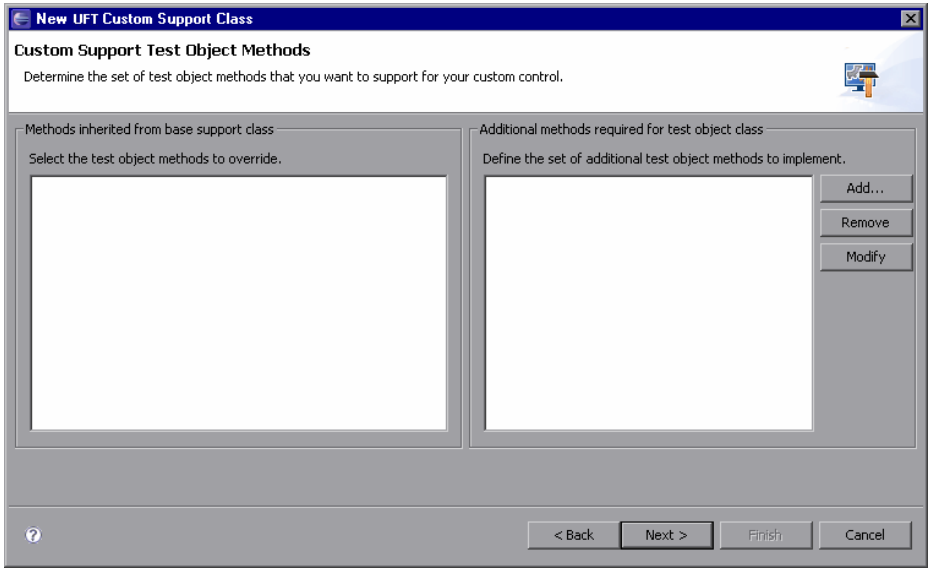
［Test Object Class Selection］画面（230 ページ）では，特定のテスト・オブジェクト・クラスを選択しませんでした。そのため，ウィザードはどのテスト・オブジェクト・クラスが ImageControl カスタム・コントロールにマッピングされているかを認識していません。その結果，認識プロパティは右側の表示枠に表示されません。

a チェック・ボックスをクリックして **label** プロパティを選択します。ウィザードによるサポート・ファイルの生成が完了したら，このプロパティの既存のサポートをカスタム・コントロールのニーズに合わせたカスタム実装でオーバーライドします。

b ［**Next**］をクリックします。［Custom Support Test Object Methods］画面が表示されます。

5 ImageControlCS で実装するテスト・オブジェクト・メソッドのセットを決定します。

この画面には，選択したオブジェクト・クラスで定義されているテスト・オブジェクト・メソッドが表示されます。



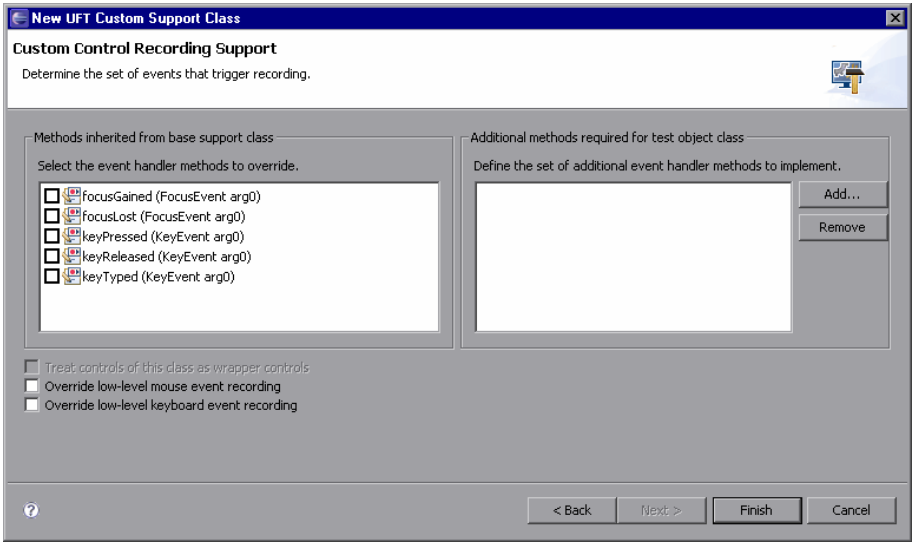
［Test Object Class Selection］画面（230 ページ）では，特定のテスト・オブジェクト・クラスを選択しませんでした。そのため，ウィザードはどのテスト・オブジェクト・クラスが ImageControl カスタム・コントロールにマッピングされているかを認識していません。その結果，テスト・オブジェクト・メソッドはこの画面に表示されません。

a ImageControl カスタム・コントロールにサポートが必要なテスト・オブジェクト・メソッドが存在しないとみなします。

b ［**Next**］をクリックします。［Custom Control Recording Support］ウィザードの画面が表示されます。

6 ImageControl コントロールでの記録をサポートするため，リッスンするイベントのセットを決定します。

この画面には，拡張対象のサポート・クラスで実装されているイベント・リスナが表示されます。この画面では，新しい機能で実装をオーバーライドするのイベント・ハンドラ・メソッドの選択，および実装する新しいイベント・リスナの追加を行うことができます。



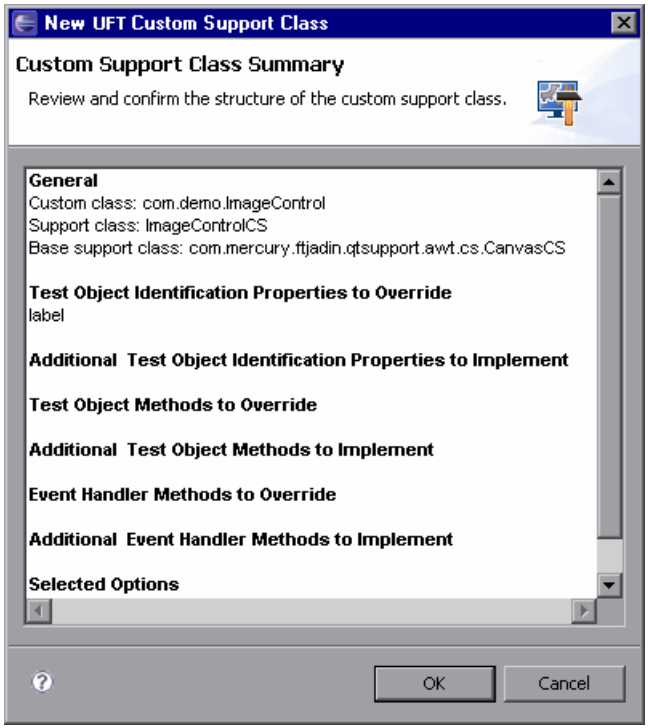
左側の表示枠では，CanvasCS で実装されているリスナを確認できます。ImageControlCS カスタム・サポート・クラスでは，これらをオーバーライドする必要はありません。

a ImageControlCS サポート・クラスを作成する目的は，実際のコントロールをサポートするためではなく，ほかのサポート・クラスのベース・サポート・クラスとして使用するためであることを考慮します。そのため，ImageControl コントロール上で記録をサポートする必要はありません。

b ［**Finish**］をクリックします。［Custom Control Support Class Summary］画面が表示されます。

7 カスタム・コントロール・サポート・クラスのサマリを表示します。

カスタム・サポート・クラスの計画内容を確認して［**OK**］をクリックします。



ImageControlsSupport プロジェクトでは，次の変更が行われています。

- ImageControl カスタム・クラスをそのサポート・クラスである ImageControlCS にマッピングするように，**ImageControls.xml** ファイルが変更されています。
- 新しい UFT カスタム・サポート・クラス（ImageControlCS）が **com.mercury.ftjadin.qtsupport.imagecontrols.cs** パッケージの **ImageControlCS.java** ファイルに作成されています。このファイルは，右側の表示枠のタブに表示されます。

  ImageControlCS クラスは CanvasCS を拡張し，メソッド・スタブを 1 つだけ含んでいます（**label_attr**）。

ImageControlCS のファイル名（ImageControlCS タブ）の横のアスタリスク（*****）は，ファイルの保存が完了していないことを示しています。ウィザードで行われる変更は相互に依存するため，不一致が生じないように必ず変更を保存する必要があります。［**File**］＞［**Save**］を選択するか，［**Save**］ボタンをクリックします。

8 ImageControlCS クラスで label_attr メソッドを実装します。

a Eclipse で，**ImageControlCS.java** ファイルの **label_attr** メソッド・スタブで，return super.label_attr(obj); を次のコードに置き換えます。これにより，ImageControl で使用される画像ファイルの名前（完全パスなし）が返されます。

```
ImageControl ic = (ImageControl)arg0;
String res = ic.getImageString();
if(res == null || res.length() == 0)
        return null;
int last = res.lastIndexOf('/');
if(last == -1)
        return res;
return res.substring(last+1);
```

b ［**Save**］ボタンをクリックするか，［**File**］＞［**Save**］を選択して変更内容を保存します。

ImageControls ツールキット・サポート階層の変更

サポート・クラスの階層は，カスタム・クラスの階層と一致する必要があります。ImageControl クラスがサポート・クラス ImageControlCS にマッピングされているので，ImageControl の子孫のサポート・クラスが ImageControlCS を拡張する必要があります。

ImageButtonCS と ImageLabelCS はどちらも **label_attr** メソッドを継承します。ImageLabelCS では，このメソッドをオーバーライドして **label** プロパティのサポートを調整する必要があります。

1 ImageControlCS を拡張するように ImageButtonCS クラスを変更します。

a Eclipse で ImageControlsSupport プロジェクトの **ImageButtonCS.java** ファイルを開き，**ImageButtonCS** クラスの署名を見つけます。

```
public class ImageButtonCS extends CanvasCS implements ActionListener
```

b ImageButtonCS が ImageControlCS を拡張するように署名を変更します。

```
public class ImageButtonCS extends ImageControlCS implements
ActionListener
```

c ImageButtonCS クラスから **label_attr** メソッドを削除します。

d **ImageButtonCS.java** ファイルを保存します。

2 ImageControlCS を拡張するように ImageLabelCS クラスを変更します。

a **ImageLabelCS.java** ファイルで，public class ImageLabelCS extends CanvasCS を public class ImageLabelCS extends ImageControlCS に置き換えます。

b ImageLabelCS クラスの **label_attr** メソッドで，次の行を

```
ImageLabel il = (ImageLabel)obj;
res = il.getImageString();
if(res == null || res.length() == 0)
        return null;
int last = res.lastIndexOf('/');
if(last == -1)
        return res;
return res.substring(last+1);
```

次のように置き換えます。

```
return super.label_attr(obj);
```

c 変更内容を保存します。

新しい ImageControls ツールキット・サポートのデプロイとテスト

新しい ImageControlCS サポート・クラスの作成時に，ウィザードでは **ImageControls.xml** ファイルを変更して ImageControl クラスを ImageControlCS サポート・クラスにマッピングしています。そのため，変更内容を反映するために，ImageControls ツールキット・サポートを再度デプロイする必要があります。

1 ImageControls ツールキット・サポートを UFT にデプロイします。

［Eclipse Package Explorer］タブで，**ImageControlsSupport** プロジェクトを選択します。

［**Deploy Toolkit Support**］ボタンをクリックするか，［**UFT**］>［**Deploy Toolkit Support**］を選択します。確認メッセージが表示されたら，［**Yes**］をクリックしてから［**OK**］をクリックします。

2 変更したカスタム・サポートをテストします。

173 ページ「ImageButton コントロールのサポートの計画」と 212 ページ「ImageLabel コントロールのサポートの計画」の手順を繰り返して，SampleApp アプリケーションを再度実行し，ImageButton と ImageLabel に対するサポートが正しく機能することを確認します。

注 : テスト・オブジェクト設定ファイルは変更されていないため，開いている UFT のセッション（ImageControls カスタム・ツールキット・サポートをロードした状態で実行中）を使用することができます。

カスタム・ツールキット・サポート・セットに変更を加えたことで, UFT での ImageLabel コントロールや ImageButton コントロールの認識方法やテスト方法が影響を受けることはありません。ただし，どちらのコントロールでも，**label** 認識プロパティのサポートが ImageControlCS クラスから継承されるようになります。ImageControl を拡張する追加のカスタム・クラスを作成した場合，それぞれの **label** プロパティも同様に UFT 上でサポートされ，追加の作業を行う必要はありません。

ImageControls ツールキットに対する改善されたサポートの既製のサンプルは，**<UFT Java Add-in Extensibility SDK インストール・フォルダ>\samples\ ImageControlsSupportAdvanced** フォルダにあります （このサンプルを手動でデプロイする場合は，前もって Java クラスをコンパイルする必要があります)。

レッスンのまとめ

このレッスンでは，ImageLabel コントロールのサポートを作成し，これを UFT で静的テキスト・オブジェクトとして認識して，**label** プロパティを隣接するコントロールの **attached text** として使用できるようにしました。

続いて，ImageControl クラスのサポートを作成してツールキット・サポートの柔軟性を高め, それに合わせて ImageControls ツールキット・サポート・セットの階層を変更しました。

➤ 次の認識プロパティ・サポート・メソッドを使用して，カスタム静的テキスト・コントロールのサポート・クラスを作成する方法について学習しました。

class_attr

tag_attr

label_attr

value_attr

➤［Test Object Class Selection］画面で［**Same as base support class**］オプションを使用し，その影響について学習しました。

その他の情報

このレッスンで使用した認識プロパティの詳細については，64 ページ「一般的な認識プロパティ・サポート・メソッド」を参照してください。

次のレッスンでは，新しいテスト・オブジェクト・クラスへのマッピングが必要なカスタム・コントロールに対するサポートの作成方法について学習します。新しいテスト・オブジェクト・クラスに対する特別な認識プロパティとテスト・オブジェクト・メソッドを定義して，対応するサポートを実装します。

第 9 章

複雑なコントロールのサポート方法の学習

このレッスンでは, Javaboutique ツールキット内に AllLights コントロールのサポートを作成します。AllLights は，新規のテスト・オブジェクト・クラスの定義が必要な，固有の動作を持つ複雑なコントロールです。

レッスン169 ページ「シンプルなコントロールのサポート方法の学習」では，シンプルなカスタム・コントロールのサポートの作成方法について学習しました。Java Add-in Extensibility の基本事項については習得済みであるため，このレッスンでは，より詳細な内容についてのみ説明します。

このレッスンは，次の内容で構成されています。

このレッスンの準備

カスタム・コントロールに対して UFT サポートを拡張する前に，そのクラスとそれを実行するアプリケーションへのアクセスを確保しておく必要があります。

ここでは，Javaboutique カスタム・ツールキット・クラスを含む Eclipse プロジェクトを作成します。AllLights クラスはアプレットとして実行できるため，カスタム・コントロールを含む追加のアプリケーションは必要ありません。

Eclipse で Javaboutique サンプルを使用して新しい Java プロジェクトを作成するには，次の手順を実行します。

1. Eclipse を実行し，［**File**］＞［**New**］＞［**Project**］を選択します。［New Project］ダイアログ・ボックスが開きます。
2. ［**Java Project**］を選択し，［**Next**］をクリックします。［New Java Project］ダイアログ・ボックスが開きます。
3. ［**Project name**］ボックスに Javaboutique と入力します。
4. ［**Create project from existing source**］オプションを選択します。
5. ［**Browse**］ボタンをクリックして，**<UFT Java Add-in Extensibility SDK インストール・フォルダ>\samples\Javaboutique\src** フォルダを参照します。［**OK**］をクリックして，［New Java Project］ダイアログ・ボックスに戻ります。
6. ［**Finish**］をクリックします。Javaboutique のサンプル・ソース・ファイルを使用して，新規の Java プロジェクトが作成されます。［Package Explorer］タブに，Javaboutique という名前の新規プロジェクトが表示されます。

注：Eclipse で新しいプロジェクトを作成する手順は，使用する Eclipse のバージョンによって異なることがあります。

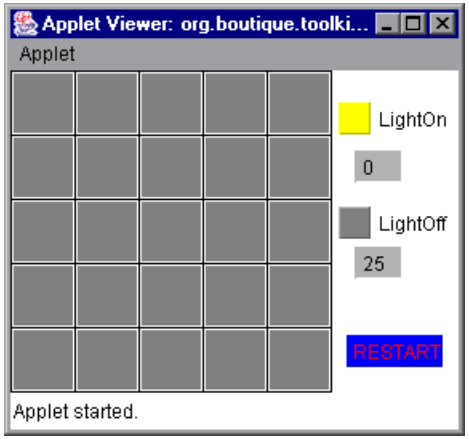
Javaboutique プロジェクトが作成されたら，プロジェクトを展開して内容を表示します。**Javaboutique\src** パッケージ・フォルダには，**org.boutique.toolkit** パッケージが含まれています。このパッケージには，**AllLights**，**AwtCalc**，**ETextField** という 3 つのカスタム・コントロールが含まれています。

このレッスンでは，Javaboutique カスタム・ツールキットに対する UFT Java Add-in Extensibility プロジェクトと AllLights のサポート・クラスを作成します。

AllLights および AwtCalc のサポートの既製のサンプルは, **<UFT Java Add-in Extensibility SDK インストール・フォルダ>\samples\JavaboutiqueSupport** フォルダにあります (このサンプルを手動でデプロイする場合は，前もって Java クラスをコンパイルする必要があります)。

次の手順で AllLights アプリケーションを実行して，AllLights コントロールの動作を理解します。

［Eclipse Package Explorer］タブで，**org.boutique.toolkit** パッケージの **Allights.java** クラスを右クリックし，［**Run As**］＞［**Java Applet**］を選択します。AllLights アプリケーションが開きます。



フレーム内のさまざまな場所をクリックします。

- グリッド領域のさまざまな場所をクリックすると，内部のルール・セットに基づいて，さまざまなライトがオン（またはオフ）になり，**LightOn** カウンタと **LightOff** カウンタが更新されます。
- ［**RESTART**］ボタンをクリックすると，すべてのライトがオフになります。**LightOn** カウンタと **LightOff** カウンタはそれに合わせて更新されます。
- その他の領域をクリックしても何も起こりません。
- このゲームの目的はすべてのライトをオンにすることです。すべてのライトがオンになると「Congratulation」のメッセージが表示されます。

AllLights コントロールのサポートの計画

ここでは，AllLights コントロールの現在の UFT サポートを分析し，95 ページ「カスタム・クラス・サポート計画のチェックリストについて」の質問に答えて，249 ページ「カスタム・クラス・サポート計画のチェックリスト」に記入します。

このためには，カスタム・コントロールを含むアプリケーションを実行し，オブジェクト・スパイ，キーワード・ビュー，記録オプションを使用して UFT の観点から分析を行うのが最適です。

1 AllLights アプリケーションを実行して，UFT を開きます。

a AllLights アプリケーションがすでに実行されている場合には，アプリケーションが上記の画像のようになるように，アプリケーション・ツールバーから［**Applet**］＞［**Restart**］を選択します。AllLights アプリケーションがまだ実行されていない場合は，［Eclipse Package Explorer］タブで **AllLights.Java** を右クリックし，［**Run As**］＞［**Java Applet**］を選択して実行します。

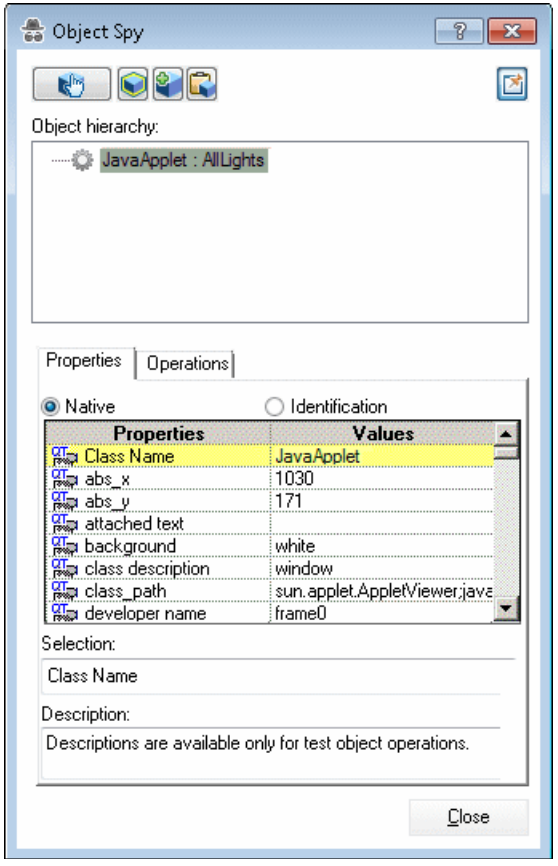
b UFT を開いて，Java Add-in をロードします。

2 オブジェクト・スパイを使用して AllLights のプロパティとメソッドを表示します。

a UFT で，GUI テストを開き，［**ツール**］＞［**オブジェクト スパイ**］を選択するか，［**オブジェクト スパイ**］ツールバー・ボタンをクリックして，［オブジェクト スパイ］ダイアログ・ボックスを開きます。［**プロパティ**］タブをクリックします。

b ［オブジェクト スパイ］ダイアログ・ボックスで，指差しアイコンをクリックしてから，AllLights アプリケーションをクリックします。

AllLights コントロールは，UFT サポートが組み込まれた JavaApplet を拡張します。そのため，アプリケーションを **AllLights** という名前の **JavaApplet** として認識します。表示されるアイコンは標準の JavaApplet クラスのアイコンになります。



c オブジェクト・スパイを閉じます。

3 AllLights コントロール上の操作を記録します。

a UFT で，［**記録**］＞［**記録と実行環境設定**］を選択して，［記録と実行環境設定］ダイアログ・ボックスを開きます。［Java］タブで，［**開いているすべての Java アプリケーションでテストを記録して実行する**］を選択します。Web アドインもロードされる場合は，［**Web**］タブをクリックして［**開いているすべてのブラウザでテストを記録して実行する**］を選択します。［**OK**］をクリックします。

b ［**記録**］ボタンをクリックするか，［**記録**］＞［**記録**］を選択します。グリッド，［**RESTART**］ボタン，カウンタなど，AllLights アプリケーション内のさまざまな場所をクリックします。

クリックするたびに，新しいステップがテストに追加されます。

Item	Operation	Value	Documentation
Action1			
AllLights			
AllLights	Click	142,144,"LEFT"	Click the "AllLights" applet with the "LEFT" mouse button.
AllLights	Click	16,188,"LEFT"	Click the "AllLights" applet with the "LEFT" mouse button.
AllLights	Click	211,35,"LEFT"	Click the "AllLights" applet with the "LEFT" mouse button.

c ［**停止**］ボタンをクリックするか，［**記録**］＞［**停止**］を選択して記録セッションを終了します。

AllLights という JavaApplet での **Click** 操作は，低レベル記録の詳細を表す引数（x 座標と y 座標およびクリックを実行したマウス・ボタン）を持つ汎用クリックです。

4 AllLights コントロールが属するカスタム・ツールキットを決定します。

コントロールに対する UFT サポートを拡張する場合は，必ずツールキットのコンテキストで行います。このチュートリアルのために，AWT を拡張する 3 つのクラス（AllLights，AwtCalc，ETextField）をまとめて，Javaboutique というカスタム・ツールキットを形成しています。

このレッスンでは，Javaboutique ツールキットのサポートを作成しますが，最初は AllLights クラスのみをサポートします。

5 カスタム・クラス・サポート計画チェックリストを完成します。

この手順では, AllLights コントロールに必要なサポートを計画し, サポート計画チェックリストにまとめます。

a サポートするカスタム・クラスを決定します。

AllLights カスタム・クラスは，AppletCS によって UFT 上でサポートされる Applet クラスを拡張します。

ここでは，UFT で AllLights を特別なアプレットとして扱い，実行される特別な操作をサポートし，そのプロパティを認識する必要があります。そのため，このコントロールに対する Extensibility サポートを作成します。

b テスト・オブジェクト・クラスをカスタム・コントロールにマッピングします。

JavaApplet テスト・オブジェクト・クラスは AllLights コントロール用の基本サポートを提供しますが，必要な認識プロパティとテスト・オブジェクト・メソッドをすべてサポートするわけではありません。そのため，AllLights という JavaApplet を拡張する新しいテスト・オブジェクト・クラスを作成して，AllLights カスタム・コントロールにマッピングします。

c 新しいテスト・オブジェクト・クラスの詳細を決定します。

- 新しいテスト・オブジェクト・クラスは, 次のアイコン・ファイルで表されます。**<UFT Java Add-in Extensibility SDK インストール・フォルダ>\samples\Javaboutique\AllLights_icon.ico**
- テスト・オブジェクトを一意に定義するのに標準で使用される認識プロパティ（ラベル，クラス，インデックス）で十分です。
- 標準のテスト・オブジェクト・メソッドは ClickLight です。
- 新しくサポートする認識プロパティは，OnCount，OnList，GameOver です。これらはすべて，［UFT Checkpoint Properties］ダイアログ・ボックスで標準で選択されています。

d コントロールがトップレベル・オブジェクトかラッパーかを決定し，テスト・オブジェクトに名前を付ける方法を決定します。

- AllLights コントロールはトップレベル・オブジェクトで，ラッパーではありません。
- テスト・オブジェクトそのものの名前は **Lights** とします。

e サポートが必要な認識プロパティを決定します。

- ➤ **OnCount :** ある時点でオンになっているライトの数を示します。
- ➤ **OnList :** ある時点でオンになっているライトの位置を示します。ライトには，左上から行ごとに 0 ～ 24 の番号が付けられます。このリストには，オンになっているライトの番号が格納されます。各番号の前にスペースが 1 つ配置されます。
- ➤ **GameOver : Yes** または **No** の文字列で，すべてのライトがオンになっているかどうかを示します。

f サポートが必要なテスト・オブジェクト・メソッドを決定します。

- ➤ **ClickLight :** グリッド内の特定のライトのクリックをシミュレートします。このメソッドでは，クリックするライトの位置を示す **Row** と **Column** の 2 つの引数が必要です。
- ➤ **Restart :**［RESTART］ボタンのクリックをシミュレートします。

g 記録のサポート方法を決定します。

- ➤ 低レベル・マウス・イベントの記録をオーバーライドします。
- ➤ マウス・イベントをリッスンします。クリックの位置に基づいて，記録メッセージを送信して ClickLight または Restart 操作を記録します。

次のページに上記の情報に基いて作成したチェックリストを示します。

カスタム・クラス・サポート計画のチェックリスト

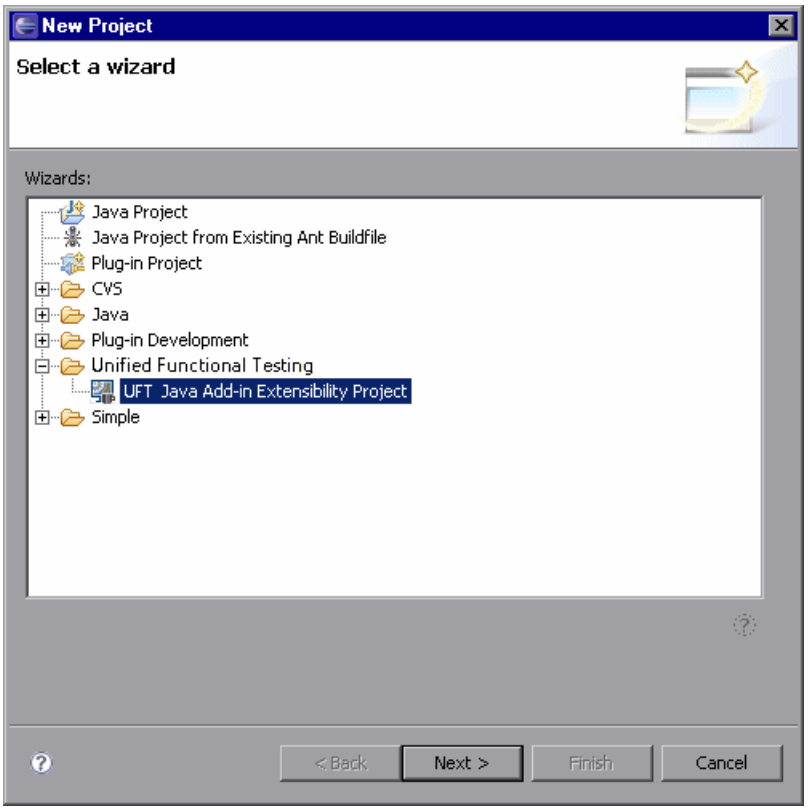
☑	カスタム・クラス・サポート計画のチェックリスト
☑	カスタム・クラスに UFT カスタム・サポートが提供されていないスーパークラスが存在するか？　**いいえ**
☑	存在する場合，階層内の上位のコントロールに対するサポートを最初に拡張すべきか？　**該当なし**
☑	UFT がインストールされているコンピュータ上でカスタム・コントロールを実行するアプリケーションが存在するか？　**はい**
☑	このカスタム・コントロール・クラスのソースが存在する場所： **Javaboutique という Eclipse プロジェクト**
☑	カスタム・コントロールに適合する既存の Java テスト・オブジェクトはどれか？　**なし**
☑	存在しない場合に作成する新しい Java テスト・オブジェクト・クラス：**AllLights** ➤ 新しいテスト・オブジェクト・クラスによって拡張されるもの：（標準 JavaObject）　**JavaApplet** ➤ アイコン・ファイルの場所（オプション）：**<UFT Java Add-in Extensibility SDK インストール・フォルダ>\samples\Javaboutique\AllLights_icon.ico** ➤ 説明用の認識プロパティ：　**label** ➤ 標準のテスト・オブジェクト・メソッド：　**ClickLight**
☑	カスタム・コントロールはトップレベル・オブジェクトか？　**はい**
☑	カスタム・コントロールはラッパーか？　**いいえ**
☑	テスト・オブジェクトの名前の基準を指定：　**"Lights" という名前を使用**
☑	サポートする認識プロパティを列挙し，標準設定のチェックポイントのプロパティをマークする： **OnCount，OnList，GameOver　（チェックポイントで標準ですべて選択される）**
☑	サポートするテスト・オブジェクト・メソッドを列挙する（必要に応じて，引数と戻り値も記入する）： **ClickLight（Variant Row，Variant Column）** **Restart（引数なし）**
☑	記録をサポートするか？（AWT ベースのみ）　**はい**
☑	その場合，記録をトリガするイベントを列挙： **MouseEvents**

UFT Java Add-in Extensibility プロジェクトの新規作成

ここでは，Javaboutique ツールキット・サポート用のプロジェクトを新規に作成します。これを行うには，Eclipse で UFT Java Add-in Extensibility プラグインによって提供されるいずれかのウィザードを使用します。

1 New UFT Java Add-in Extensibility Project ウィザードを開きます。

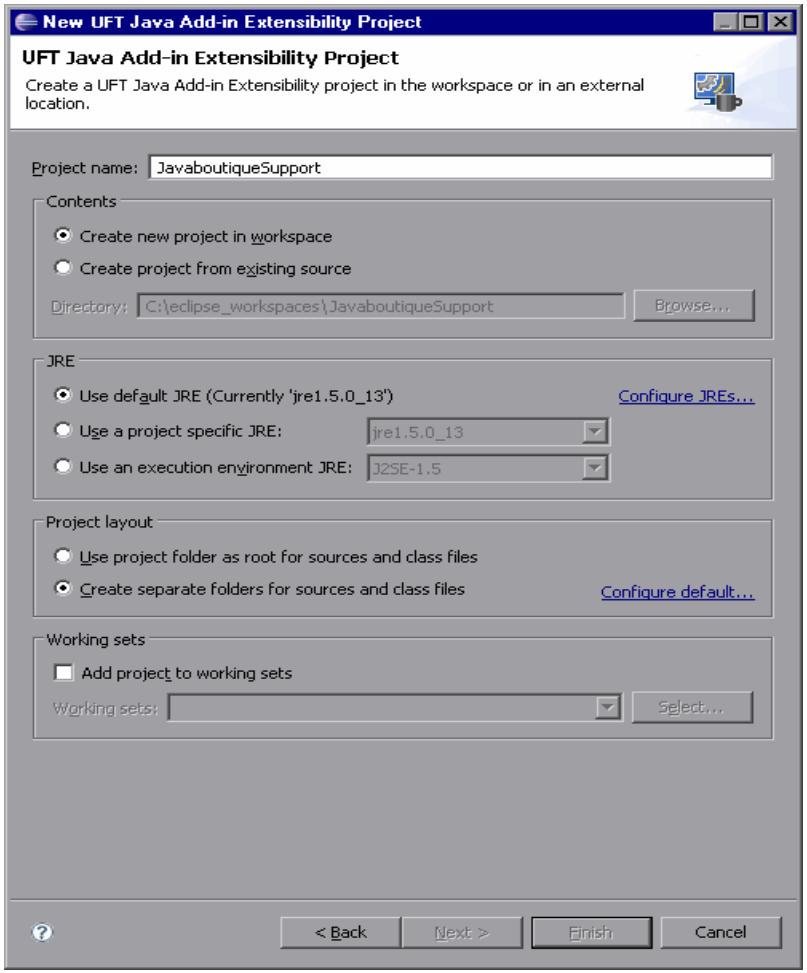
a Eclipse で，［**File**］＞［**New**］＞［**Project**］を選択します。［New Project］ダイアログ・ボックスが開きます。**Unified Functional Testing** フォルダを展開し，**UFT Java Add-in Extensibility Project** を選択します。



b ［**Next**］をクリックします。［UFT Java Add-in Extensibility Project］画面が開きます。この画面の詳細は，使用する Eclipse のバージョンによって異なることがあります。

2 UFT Java Add-in Extensibility プロジェクトの詳細を入力します。

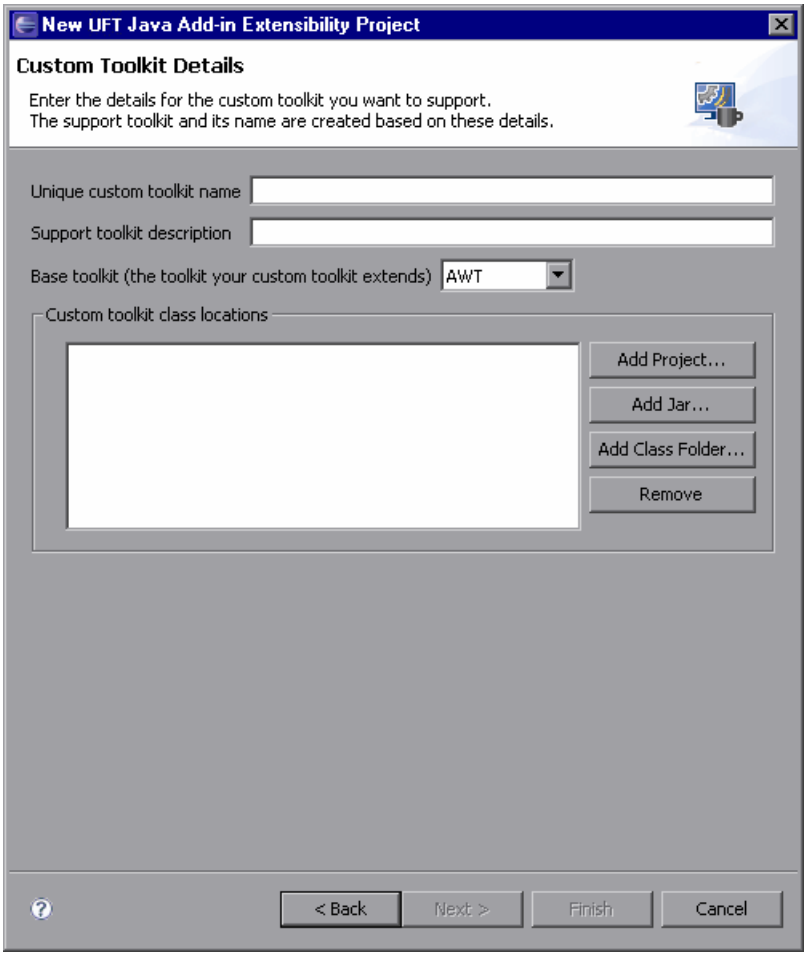
a ［**Project name**］ボックスに，JavaboutiqueSupport と入力します。［**Create separate folders for sources and class files**］を選択します（以前のバージョンの Eclipse では，このオプションの表記は［**Create separate source and output folders**］となっています)。このダイアログ・ボックスの詳細については，Eclipse のヘルプを参照してください。



b ［**Next**］をクリックします。［Custom Toolkit Details］画面が表示されます。

3 カスタム・ツールキットの詳細を入力します。

この画面では，対応するカスタム・ツールキット・サポート・セットをウィザードで生成できるように，Javaboutique ツールキットの詳細を指定します。



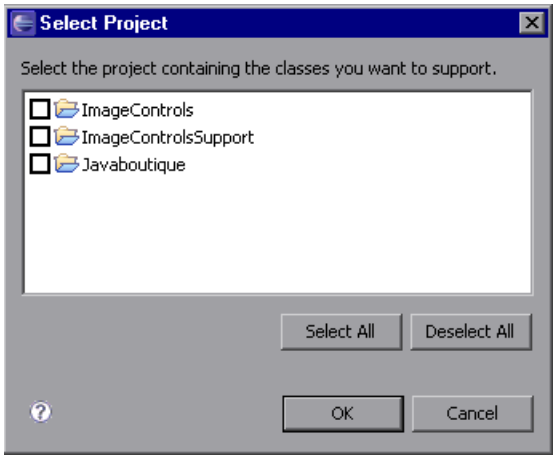
次の詳細情報を入力します。

a ［**Unique custom toolkit name**］に Javaboutique と入力します。

b ［**Support toolkit description**］ボックスに，Javaboutique toolkit support と入力します。

c AllLights カスタム・クラスは AWT コンポーネントを拡張するため，［**Base toolkit**］は標準設定の AWT のままにしておきます。

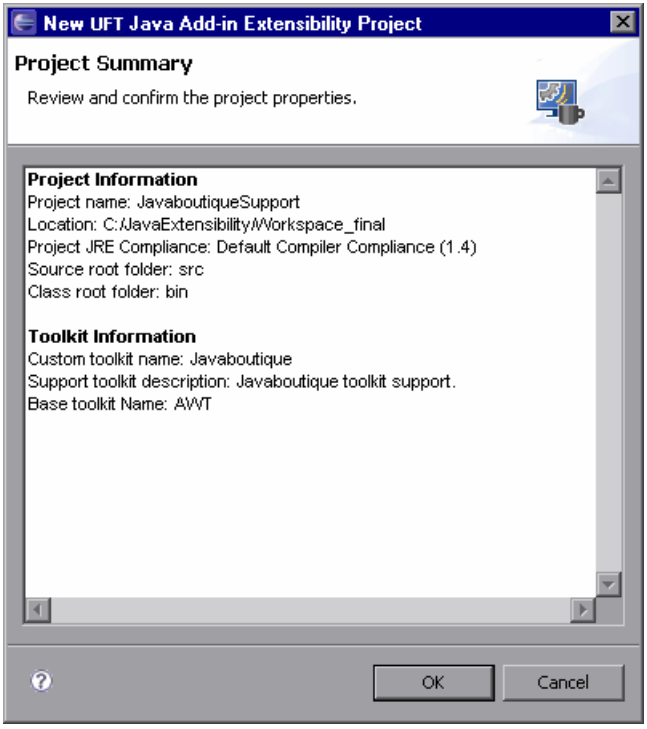
d ［**Custom toolkit class locations**］領域で［**Add project**］をクリックして，Javaboutique ツールキット用のカスタム・クラスを含む Eclipse の Java プロジェクトを選択します。［Select Project］ダイアログ・ボックスが開き，現在の Eclipse ワークスペースのプロジェクトが表示されます。



e ［**Javaboutique**］チェック・ボックスを選択します。［**OK**］をクリックします。Javaboutique プロジェクトが［**Custom toolkit class locations**］ボックスに追加されます。［**Finish**］をクリックします。［Project Summary］画面が開きます。

4 ［Project Summary］画面を表示します。

プロジェクトの詳細を確認して［**OK**］をクリックします。



次のカスタム・ツールキット・サポート・セットの基本的なパッケージとファイルを使用して，新しい UFT Java Add-in Extensibility プロジェクト JavaboutiqueSupport が作成されます。

- パッケージ **com.mercury.ftjadin.qtsupport.javaboutique**（新しいツールキット・サポート・クラス・ファイル **JavaboutiqueSupport.java** を含む）
- パッケージ **com.mercury.ftjadin.qtsupport.javaboutique.cs**
- **Configuration** フォルダ（**TestObjects** フォルダと新しいツールキット設定ファイル **Javaboutique.xml** を含む）

注：コンピュータ上に複数の JRE がインストールされている場合は，Javaboutique プロジェクトと JavaboutiqueSupport プロジェクトで同じ JRE バージョンを使用するようにしてください。同じ JRE バージョンを使用していない場合は，どちらかのプロジェクトの JRE を変更して，同じバージョンを使用するようにしてください。

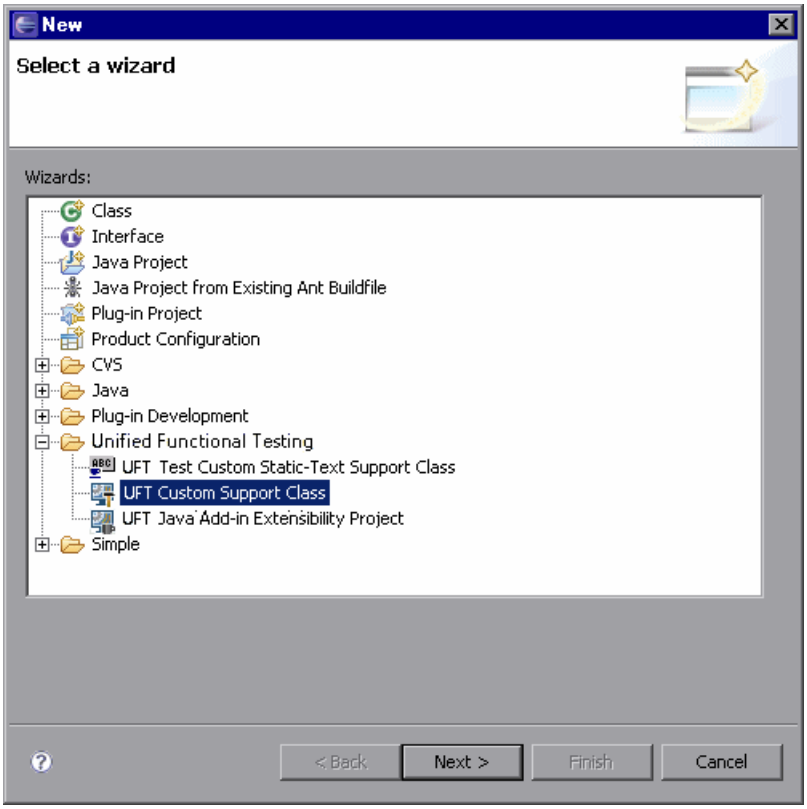
UFT カスタム・サポート・クラスの新規作成

ここでは，Javaboutique ツールキットに含まれる AllLights コントロールのカスタム・サポート・クラスを作成します。これを行うには，Eclipse で UFT Java Add-in Extensibility プラグインによって提供されるいずれかのウィザードを使用します。各ウィザード画面では，カスタム・サポート計画の内容に合わせて詳細を指定します。続く項では，ウィザードによってこのクラス内に作成されるメソッドの実装を行います。

注：次の項では，このレッスンに関連するウィザード画面のオプションについてのみ説明します。ウィザード画面で指定できるすべてのオプションの説明は，第 5 章「UFT Java Add-in Extensibility Eclipse プラグインの使用」を参照してください。

1 New UFT Custom Support Class ウィザードを開きます。

a ［Eclipse Package Explorer］タブで，新しい UFT Java Add-in Extensibility プロジェクト **JavaboutiqueSupport** を選択します。［**File**］＞［**New**］＞［**Other**］を選択します。［New］ダイアログ・ボックスが開きます。

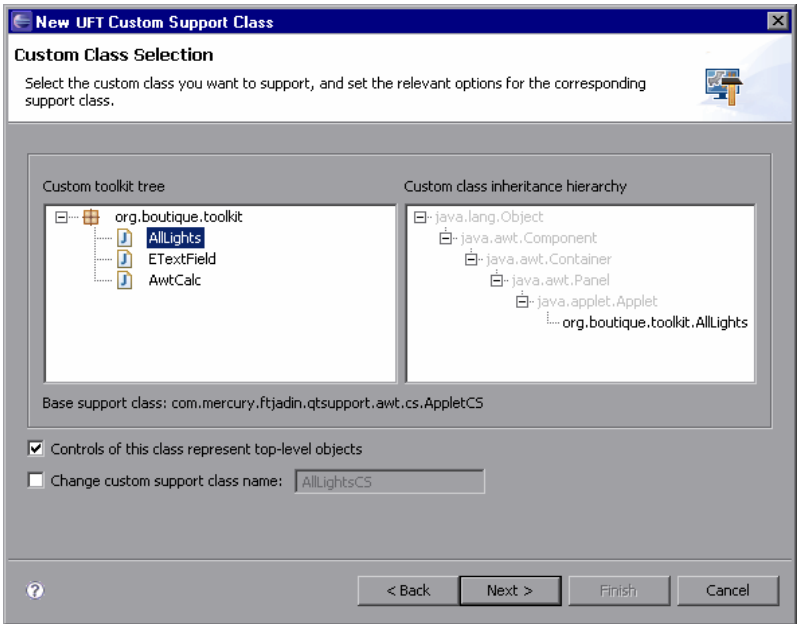


b **Unified Functional Testing** フォルダを展開し，**UFT Custom Support Class** を選択します。

c ［**Next**］をクリックします。［Custom Class Selection］画面が表示されます。

2 サポートするカスタム・クラスを選択し，サポート・クラスのオプションを設定します。

a **org.boutique.toolkit** パッケージの **AllLights** クラスを選択します。



AllLights カスタム・クラスは，UFT 上でサポートされる java.applet.Applet を拡張します。そのため，AllLights サポート・クラスは，**Base support class**: **com.mercury.ftjadin.qtsupport.awt.cs.AppletCS** を拡張します。その結果，［**Controls of this class represent top-level objects**］チェック・ボックスが標準で選択されます。

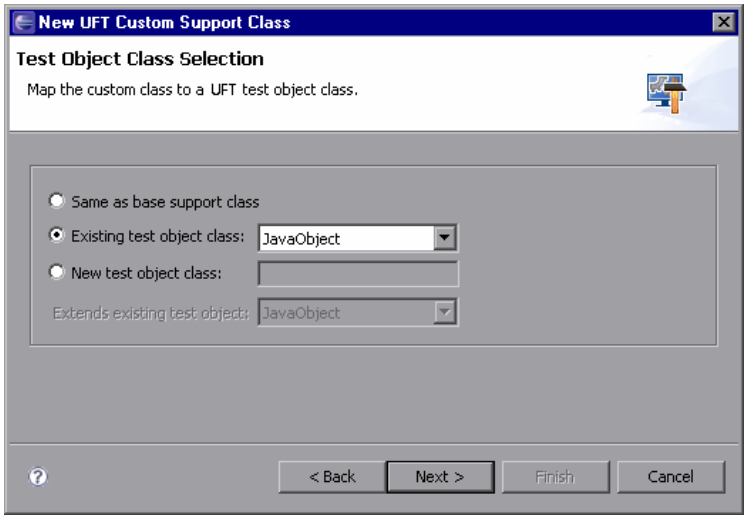
b このチェック・ボックスは選択したままにします。これは，UFT で AllLights コントロールをテスト・オブジェクト階層の最上位の Java テスト・オブジェクトとして認識する必要があるためです。

c 標準設定のカスタム・サポート・クラス名（AllLightsCS）をそのまま使用します。

d ［**Next**］をクリックします。［Test Object Selection］画面が表示されます。

3 カスタム・コントロールを表すテスト・オブジェクト・クラスを選択します。

この画面では，カスタム・コントロールをテスト・オブジェクト・クラスにマッピングします。UFT GUI のテストでは，カスタム・クラス・コントロールは，このテスト・オブジェクト・クラスのテスト・オブジェクトによって表されます。



「AllLights コントロールのサポートの計画」の手順 5 (247ページ を参照) では, AllLights カスタム・コントロールを JavaApplet を拡張する新しいテスト・オブジェクト・クラス（AllLights）にマッピングすることを決めました。

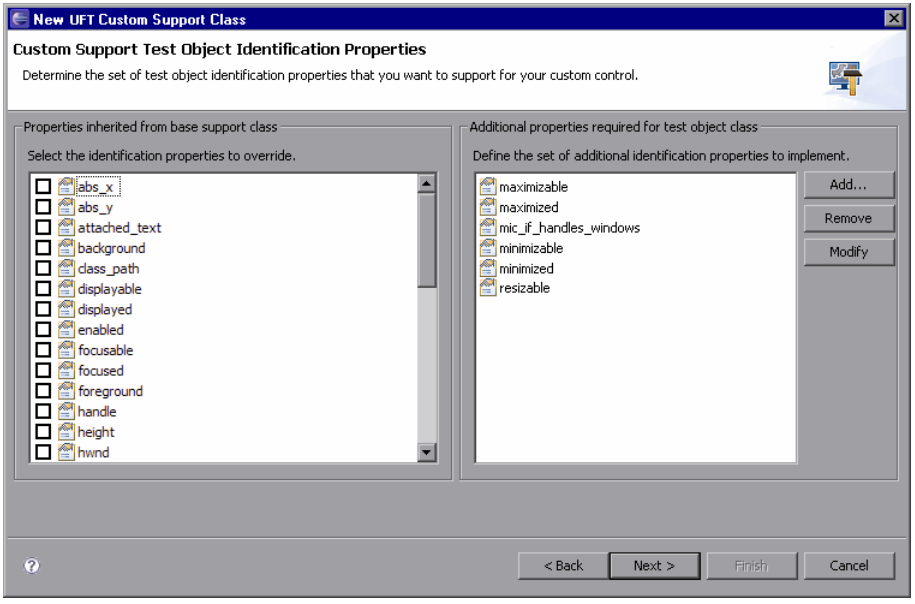
a ［**New test object class**］オプションを選択し，新しいテスト・オブジェクト・クラスの名前として AllLights と入力します。

b ［**Extends existing test object**］リストで，**JavaApplet** を選択します。このリストには，UFT で現在サポートされているすべての Java オブジェクトが含まれています。カスタム・サポートに対する新しいテスト・オブジェクトを定義すると，このテスト・オブジェクトもリストに追加されます。

c ［**Next**］をクリックします。［Custom Support Test Object Identification Properties］画面が表示されます。

4 AllLightsCS で実装するテスト・オブジェクト認識プロパティのセットを決定します。

この画面には，拡張しようとしているベース・サポート・クラスでサポートされている認識プロパティと，選択したテスト・オブジェクト・クラスで定義されていて，まだサポートされていない追加プロパティが表示されます。この画面では，サポートを実装または新しい機能でオーバーライドするプロパティの選択，および新しいプロパティの追加を行うことができます。



a 左側の表示枠には，AppletCS でサポートが実装されている認識プロパティがすべて表示されます。これらの認識プロパティのサポートは，新しい AllLightsCS サポート・クラスによって継承されます。このリストのほとんどのプロパティは，標準設定の実装で十分です。

［**label**］チェック・ボックスを選択します。ウィザードによるサポート・ファイルの生成が完了したら，このプロパティの既存のサポートをカスタム・コントロールのニーズに合わせたカスタム実装でオーバーライドします。

b 右側の表示枠に表示される認識プロパティは，AppletCS でサポートされていない JavaApplet のプロパティです。これらのプロパティは，AllLights のサポートに必要ありません。これらを選択して［**Remove**］をクリックし，［**Yes**］をクリックして確定します。

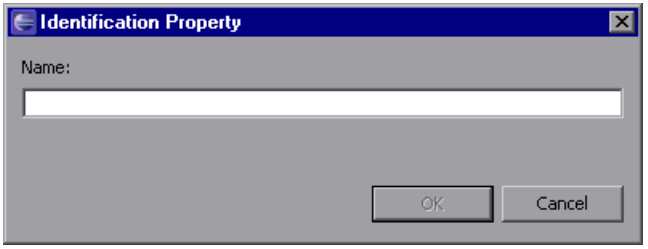
これらの認識プロパティは，JavaApplet テスト・オブジェクト・クラスに基づいて作成された AllLights テスト・オブジェクト・クラスの定義に含まれます。このリストから削除したプロパティは，AllLights コントロールでサポートされません。これらのプロパティは UFT オブジェクト・スパイに表示される認識プロパティのリストには引き続き表示されますが，値はありません。

c 「AllLights コントロールのサポートの計画」の手順 5（247 ページを参照）では，AllLights テスト・オブジェクト上で新しい認識プロパティをサポートすることを決めました。次の手順では，これらのプロパティをテスト・オブジェクト・クラスに必要な追加プロパティのリストに追加します。ウィザードによるサポート・ファイルの生成が完了したら，これらのプロパティのサポートを実装します。

これらの認識プロパティは，テスト・オブジェクト・クラスの定義に追加されます。これは，このクラスのすべてのテスト・オブジェクトに対し，UFT の認識プロパティのリストに新しいプロパティが追加されることを意味します。新しい AllLights テスト・オブジェクト・クラスを作成するのはこのためです。

5 AllLightsCS で実装する新しいテスト・オブジェクト認識プロパティを追加します。

a ［**Additional properties required for test object class**］表示枠の［**Add**］をクリックします。［Identification Property］ダイアログ・ボックスが開きます。



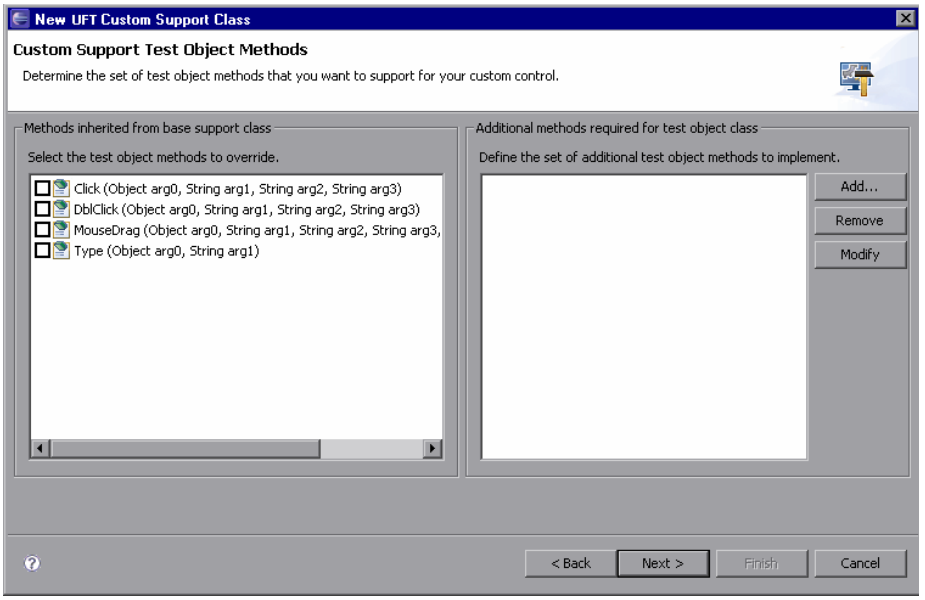
b ［**Name**］ボックスに，OnCount と入力します。［**OK**］をクリックして，新しい認識プロパティをリストに追加します。

c この手順を繰り返して，プロパティ OnList と GameOver を追加します。

d ［**Next**］をクリックします。［Custom Support Test Object Methods］画面が表示されます。

6 AllLightsCS で実装するテスト・オブジェクト・メソッドのセットを決定します。

この画面には，選択したオブジェクト・クラスで定義されているテスト・オブジェクト・メソッドが表示されます。この画面では，新しい機能でサポートを実装またはオーバーライドするメソッドの選択，および新しいメソッドの追加を行うことができます。



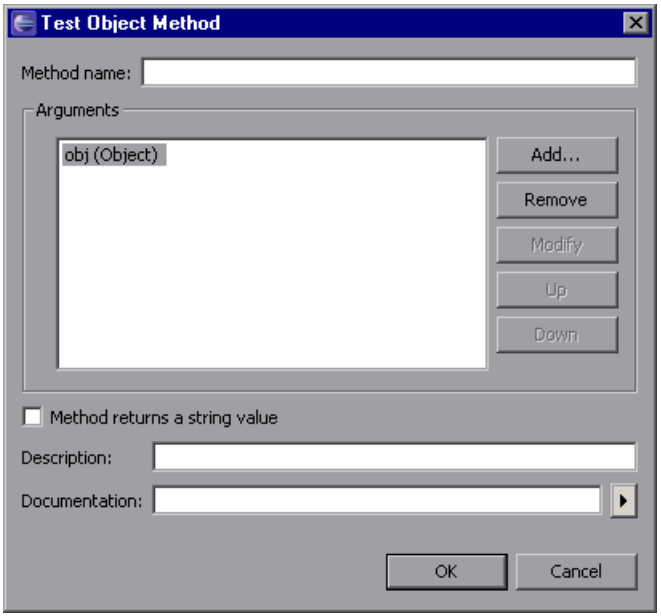
左側の表示枠には，AppletCS でサポートが実装されている（選択したテスト・オブジェクト・クラスで定義された）テスト・オブジェクト・メソッドがすべて表示されます。これらのテスト・オブジェクト・メソッドのサポートは，AllLightsCS によって継承されます。オーバーライドするメソッドを選択する必要はありません。

右側の表示枠には，AllLights テスト・オブジェクト・クラス用に定義され，AppletCS でサポートされていないテスト・オブジェクト・メソッドが表示されます。現在，該当するメソッドは定義されていません。

「AllLights コントロールのサポートの計画」の手順 5 (247 ページを参照) では，AllLights テスト・オブジェクト上で新しいテスト・オブジェクト・メソッドをサポートすることを決めました。ここでは，これらのメソッドをテスト・オブジェクト・クラスに必要な追加テスト・オブジェクト・メソッドのリストに追加する必要があります。ウィザードによるサポート・ファイルの生成が完了したら，追加するメソッドのサポートを実装します。

テスト・オブジェクト・メソッドは，既存のテスト・オブジェクト・クラスに追加されます。これは，このクラスのすべてのテスト・オブジェクトに対し，これらのオブジェクトでサポートされるかどうかに関係なく，UFT に新しいメソッドが追加されることを意味します。UFT GUI のテストでは，オブジェクトに対するテスト・オブジェクト・メソッドを呼び出して，そのメソッドがサポートされていない場合，実行時エラーが発生します。新しい AllLights テスト・オブジェクト・クラスを作成するのはこのためです。

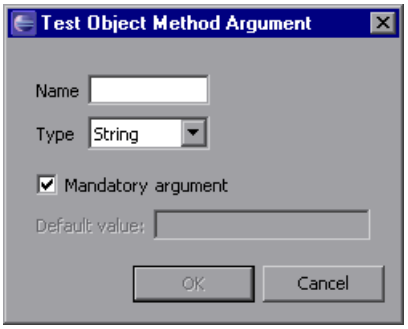
a ［**Additional test object methods required for test object class**］表示枠の［**Add**］をクリックします。［Test Object Method］ダイアログ・ボックスが表示されます。



- ［**Method Name**］ボックスに，Restart と入力します。Restart テスト・オブジェクト・メソッドには，カスタム・コントロールを表す必須の **obj (Object)** 以外の引数は必要ありません。
- ［**Method returns a string value**］チェック・ボックスは選択解除された状態のままにします。このメソッドは，リターン・コードのみを返します。
- ［**Description**］ボックスに，次のように入力します：［RESTART］**ボタンをクリックします。**
- ［**Documentation**］ボックスに，次のように入力します：［RESTART］**ボタンをクリックします。**
- ［**OK**］をクリックして［Test Object Method］ダイアログ・ボックスを閉じて，Restart メソッドをリストに追加します。

b ［**Add**］をもう一度クリックして，別のテスト・オブジェクト・メソッドを追加します。[Test Object Method]ダイアログ・ボックスが開いたら，次の操作を実行します。

- ［**Method Name**］ボックスに，ClickLight と入力します。
- ClickLight メソッドに引数として Row と Column を追加します。
- ［**Arguments**］領域で，［**Add**］をクリックします。［Test Object Method Argument］ダイアログ・ボックスが表示されます。



［**Name**］ボックスに，Row と入力します。

［**Type**］ボックスで **Variant** を選択します（**String** を選択した場合，ClickLight メソッドを使用して UFT GUI のテストにステップを追加する際に，行番号の引数を引用符で囲む必要があります)。

［**Mandatory argument**］チェック・ボックスは選択された状態のままにします。

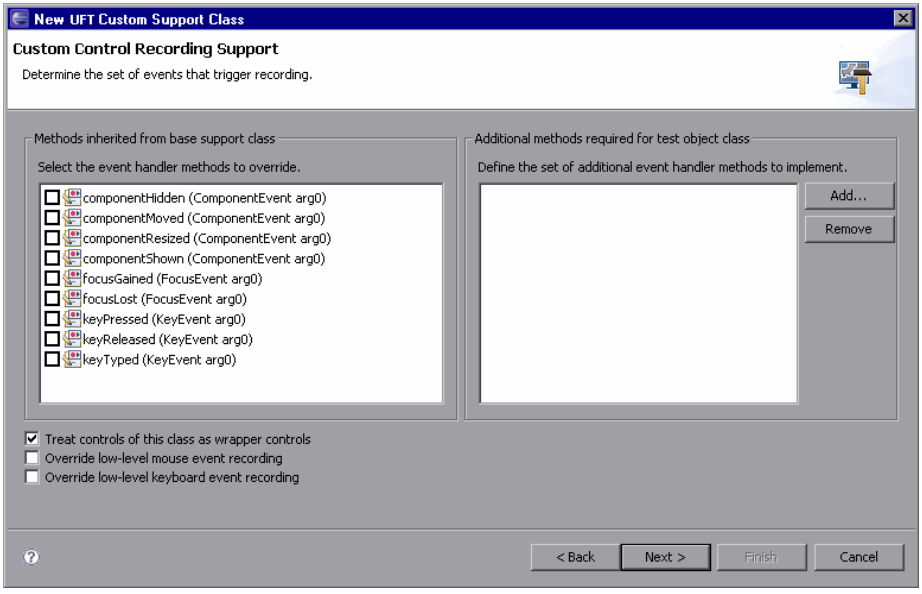
［**OK**］をクリックして［Test Object Method Argument］ダイアログ・ボックスを閉じ，ClickLight テスト・オブジェクト・メソッドの引数のリストに Row 引数を追加します。

- この手順を繰り返して，Column 引数をリストに追加します。
- ［**Method returns a string value**］チェック・ボックスは選択解除された状態のままにします。
- ［**Description**］ボックスに，次のように入力します : **指定されたライトをクリックします。**
- ［**Documentation**］ボックスに，次のように入力します : **行 <Row> 列 <Column> のライトをクリックします。** をクリックして関連する引数を選択し，入力した文内の <Row> 引数と <Column> 引数を入力します。［**Documentation**］ボックスのテキストは最終的に次のようになります : **行 %a1 列 %a2 のライトをクリックします。**
- ［**OK**］をクリックして［Test Object Method］ダイアログ・ボックスを閉じ，ClickLight メソッドをリストに追加します。

c ［**Next**］をクリックします。［Custom Control Recording Support］ウィザードの画面が表示されます。

7 AllLights コントロールでの記録をサポートするため，リッスンするイベントのセットを決定します。

この画面には，拡張用に選択したサポート・クラスでサポートされるイベント・リスナが表示されます。この画面では，新しい機能で実装をオーバーライドするリスナの選択，および実装する新しいイベント・リスナの追加を行うことができます。

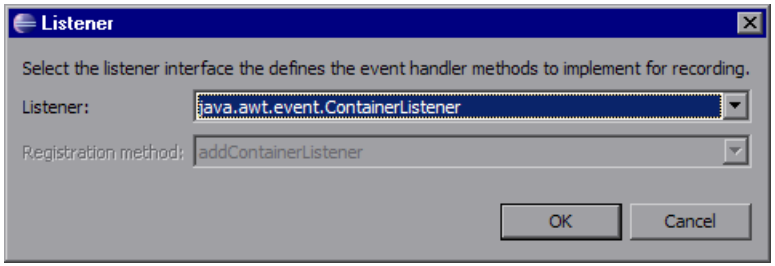


左側の表示枠では，AppletCS で実装されているリスナを確認できます。AllLightsCS カスタム・サポート・クラスでは，これらをオーバーライドする必要はありません。

右側の表示枠では, AllLightsCS に追加するリスナを指定します。選択した各リスナは, カスタム・サポート・クラスに追加するイベント・ハンドラ・メソッドのセットを含んでいます。

a ［**Add**］をクリックして **MouseListener** を追加します。

［Listener］ダイアログ・ボックスが開きます。



［**Listener**］リストから **java.awt.event.MouseListener** を選択し，［**OK**］をクリックします。［Listener］ダイアログ・ボックスが閉じて，**MouseListener** とこれに含まれるすべてのイベント・ハンドラ・メソッドがウィザード画面の右側の表示枠のリストに追加されます。

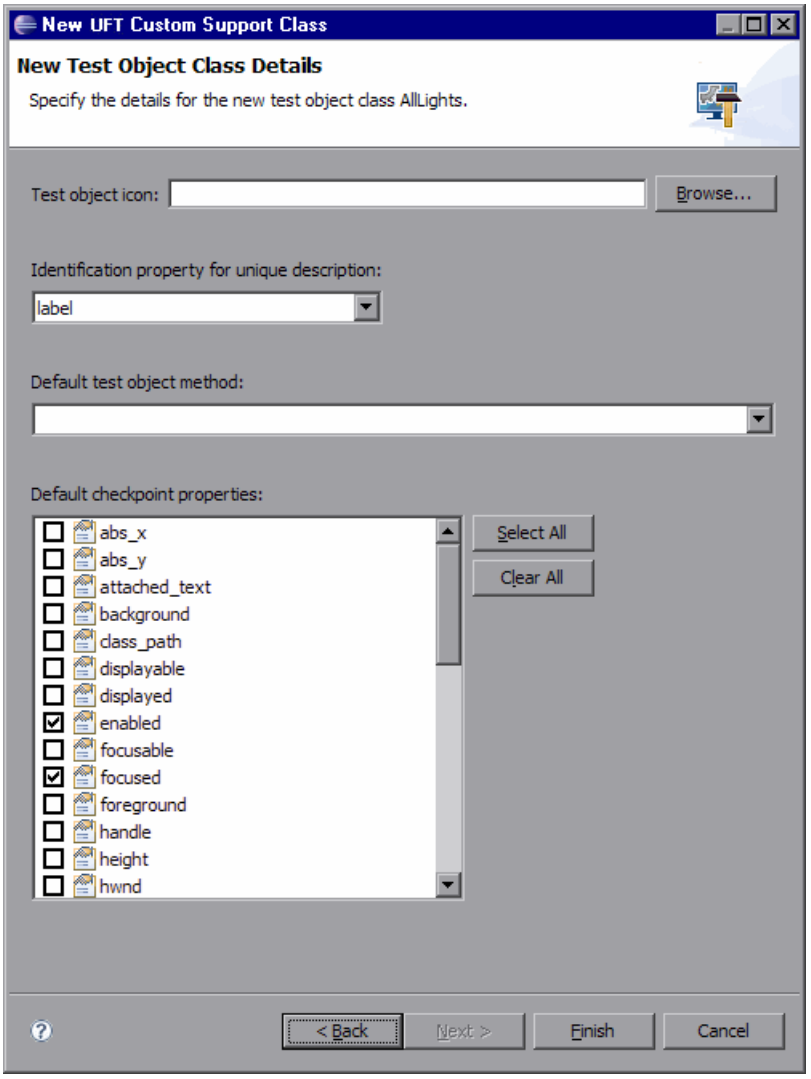
b ［Custom Control Recording Support］画面で，次の操作を実行します。

- ［**Treat controls of this class as wrapper controls**］チェック・ボックスを選択解除します。このチェック・ボックスは標準で選択された状態になっています。これは，AllLights クラスが **java.awt.container** を拡張するためです。
- ［**Override low-level mouse event recording**］チェック・ボックスを選択して，記録したいイベントの代わりに低レベル・イベント（座標ベースの操作）が記録されないようにします。

c ［**Next**］をクリックします。［New Test Object Class Details］画面が表示されます。

8 新しいテスト・オブジェクト・クラス AllLights の詳細を定義します。

この画面では，カスタム・コントロールにマッピングするために作成している新しいテスト・オブジェクト・クラスの詳細を定義します。



次の手順を実行します。

a ［**Test object icon**］で［**Browse**］をクリックし，**<UFT Java Add-in Extensibility SDK インストール・フォルダ>\samples\Javaboutique** フォルダを探して **AllLights_icon.ico** ファイルを選択します。

b ［**Identification property for unique description**］ボックスで，選択された **label** プロパティをそのままにします。

注 : テスト・オブジェクト記述に認識プロパティを追加する場合は，テスト・オブジェクト設定ファイルでこれを手動で指定する必要があります。詳細については，144 ページ「［New Test Object Class Details］画面」を参照してください。

---

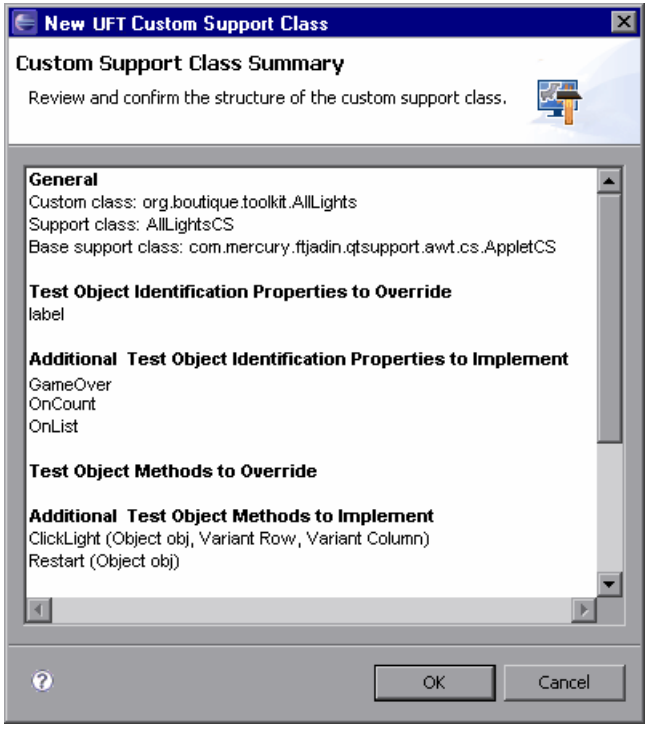
c ［**Default test object method**］リストで，**ClickLight** を選択します。

d ［**Default checkpoint properties**］ボックスで，選択済みのプロパティをそのままにして，**GameOver**，**OnCount**，**OnList** チェック・ボックスを併せて選択します。

e ［**Finish**］をクリックします。［Custom Control Support Class Summary］画面が表示されます。

9 カスタム・コントロール・サポート・クラスのサマリを表示します。

カスタム・サポート・クラスの計画内容を確認して［**OK**］をクリックします。



JavaboutiqueSupport プロジェクトでは，次の変更が行われています。

- 新しい UFT カスタム・サポート・クラス（AllLightsCS）が **com.mercury.ftjadin.qtsupport.Javaboutique.cs** パッケージに作成されています。このファイルは，右側の表示枠のタブに表示されます。
- 新しい **JavaboutiqueTestObjects.xml** ファイルが **Configuration\TestObjects** フォルダに作成されています。
- **Javaboutique.xml** ファイルが変更されています。ファイルに要素が追加され，**AllLights** カスタム・クラスが **AllLightCS** サポート・クラスにマッピングされます。このファイルの構造については，『UFT Java Add-in Extensibility Toolkit Configuration Schema ヘルプ』（『Java Add-in Extensibility SDK ヘルプ』で利用可能）を参照してください。

AllLightsCS クラスおよび **JavaboutiqueTestObjects.xml** ファイルの詳細については，269 ページ「新しいカスタム・サポート・ファイルについて」を参照してください。

AllLightsCS のファイル名（AllLightsCS タブ）の横のアスタリスク（*****）は，ファイルの保存が完了していないことを示します。ウィザードで行われる変更は相互に依存するため，不一致が生じないように必ず変更を保存する必要があります。［**File**］＞［**Save**］を選択するか，［**Save**］ボタンをクリックします。

新しいカスタム・サポート・ファイルについて

New UFT Custom Support Class ウィザードは終了時に，新しく作成したクラスをツールキット設定ファイルに登録し，次のファイルを作成します。

- **AllLightsCS.java :** このファイルには，新しい AllLightsCS サポート・クラスが含まれます。
- **JavaboutiqueTestObject.xml :** このファイルには，Javaboutique ツールキット・サポート用に定義された新しいテスト・オブジェクト・クラスが含まれます。この時点で，該当するテスト・オブジェクト・クラスは AllLights の 1 つだけです。

以降の項では，ウィザードによってこれらのファイル内に作成される内容について説明します。

AllLightsCS カスタム・サポート・クラスについて

169 ページ「シンプルなコントロールのサポート方法の学習」のレッスンを通じて，ウィザードが新しいカスタム・サポート・クラスに作成する基本的な要素についてはすでに理解できています。新しい **AllLightsCS.java** ファイルの内容を検証し，次のメソッドと宣言を確認します。

- **AllLightsCS** サポート・クラスの宣言：**AppletCS** サポート・クラスを拡張し，**MouseListener** インタフェースを実装します。
- **DEBUG_ALLLIGHTSCS** フラグの宣言：ログ・メッセージの出力を制御するのに使用できます。

- **AllLightsCS** コンストラクタ・メソッド : **addSimpleListener** を呼び出して, **MouseListener** を **AllLights** コントロールでの登録が必要なリスナのリストに追加します。
- **to_class_attr** メソッド:新しいテスト・オブジェクト・クラス名 **AllLights** を返します。
- **super.label_attr** を返す **label_attr** のメソッド・スタブ : 具体的なラベルに置き換えることができます。
- **oncount_attr**, **onlist_attr**, **gameover_attr** メソッドのメソッド・スタブ : 追加した認識プロパティをサポートするために実装する必要があります。実装するまでの間, これらのメソッドは Null を返します。これは, これらが追加された新しいメソッドで, AllLightsCS で拡張されるスーパークラスで実装されていないためです。

注 : ウィザード画面で指定する認識プロパティ名には, 大文字を使用することができます。指定した名前はテスト・オブジェクト設定ファイルにそのまま記述されます。ただし, これらの認識プロパティのサポート・メソッドの名前は, ウィザードによって大文字が小文字に置き換えられます。UFT では, 認識プロパティ名は小文字のみで表示されます。

- **Restart_replayMethod** および **ClickLight_replayMethod** メソッドのメソッド・スタブ : 追加したテスト・オブジェクト・メソッドをサポートするために実装する必要があります。実装するまでの間, これらのメソッドはエラー・コード NOT_IMPLEMENTED を返します。
- **mouseRecordTarget** メソッド : 低レベル・マウス・イベントの記録をオーバーライドするため **null** を返します。

➤ **MouseListener** インタフェースで定義されるイベント・ハンドラ・メソッド（**mouseClicked**, **mouseEntered**, **mouseExited**, **mousePressed**, **mouseReleased**）用のメソッド・スタブ。これらのメソッド・スタブには，必要に応じて実装する（**MicAPI.record** を呼び出して記録メッセージを UFT に送信する）必要があることを示すコメントが含まれています。

➤ **is_window** メソッド（**true** を返す）が **AllLightsCS** サポート・クラスに追加されました。これは，［Custom Class Selection］画面で［**Controls of this class represent top-level objects**］チェック・ボックスを選択したためです。テスト・オブジェクトの学習時に，UFT は **is_window** メソッドを呼び出して，親オブジェクトを探すか，このオブジェクトを階層内の最上位の Java オブジェクトとして表示するかを決定します。

Javaboutique テスト・オブジェクト設定ファイルについて

ウィザードでは，指定した詳細に基づいてテスト・オブジェクト設定ファイルにテスト・オブジェクト・クラスの定義が作成されます。

新しい **JavaboutiqueTestObject.xml** ファイルを開いて，その内容を検証します。このファイルの構造については，『UFT Test Object Schema ヘルプ』（『Java Add-in Extensibility SDK ヘルプ』で利用可能）を参照してください。

テスト・オブジェクト設定ファイルで，次の要素を確認します。

➤ このファイル内のテスト・オブジェクト・クラスが属するカスタム・ツールキットとアドインの名前（**TypeInformation** 要素内）：

```
PackageName="Javaboutique" AddinName="Java"
```

➤ 新しいテスト・オブジェクト・クラスによって拡張されるテスト・オブジェクト・クラス（**ClassInfo** 要素内）：

```
BaseClassInfoName="JavaApplet"
```

➤ 新しいテスト・オブジェクト・クラスの名前とその標準のテスト・オブジェクト・メソッド（**ClassInfo** 要素内）：

```
DefaultOperationName="ClickLight" Name="AllLights"
```

➤ アイコン・ファイルの場所（**IconInfo** 要素内）：

```
IconFile="<UFT Java Add-in Extensibility SDK インストール・
フォルダ>\samples\Javaboutique\AllLights_icon.ico"
```

➤ 追加した新しいテスト・オブジェクト・メソッドの定義，その説明，ドキュメント，引数（<TypeInfo> 要素内）。

➤ このテスト・オブジェクト・クラスの認識プロパティの定義（<IdentificationProperties> 要素内）。認識プロパティには，ForVerification，ForDefaultVerification，ForDescription というマークが付きます。

新しいカスタム・ツールキット・サポートのデプロイとテスト

ここでは，Eclipse で UFT の **Deploy Toolkit Support** コマンドを使用して，Javaboutique ツールキット・サポートを UFT にデプロイします。現在，このツールキット内の唯一のコントロールである AllLights コントロールがサポートされています。ツールキット・サポートはまだ完成していませんが，この時点までに作成したサポートをテストすることはできます。

1 Javaboutique ツールキット・サポートを UFT にデプロイします。

a ［Eclipse Package Explorer］タブで，**JavaboutiqueSupport** プロジェクトを選択します。

b ［**Deploy Toolkit Support**］ボタンをクリックするか，［**UFT**］＞［**Deploy Toolkit Support**］を選択します。確認メッセージが表示されたら，［**Yes**］をクリックしてから［**OK**］をクリックします。

ツールキット設定ファイルとテスト・オブジェクト設定ファイルが，UFT のインストール・フォルダ内の関連するフォルダにコピーされます。このカスタム・サポートは，次回 UFT を起動してカスタム・アプリケーションを開始したときに利用できるようになります。

カスタム・ツールキット・サポートのデプロイの詳細については，78 ページ「カスタム・ツールキット・サポートのデプロイと実行」を参照してください。

2 UFT を起動し，Java Add-in とカスタム・ツールキット・サポートをロードします。

a UFT を開きます。［アドイン マネージャ］ダイアログ・ボックスで，使用可能なアドインのリストに **Java** アドインの子として **Javaboutique** が表示されます（［アドイン マネージャ］ダイアログ・ボックスが開かない場合は，『HP Unified Functional Testing アドイン・ガイド』の手順を参照してください）。

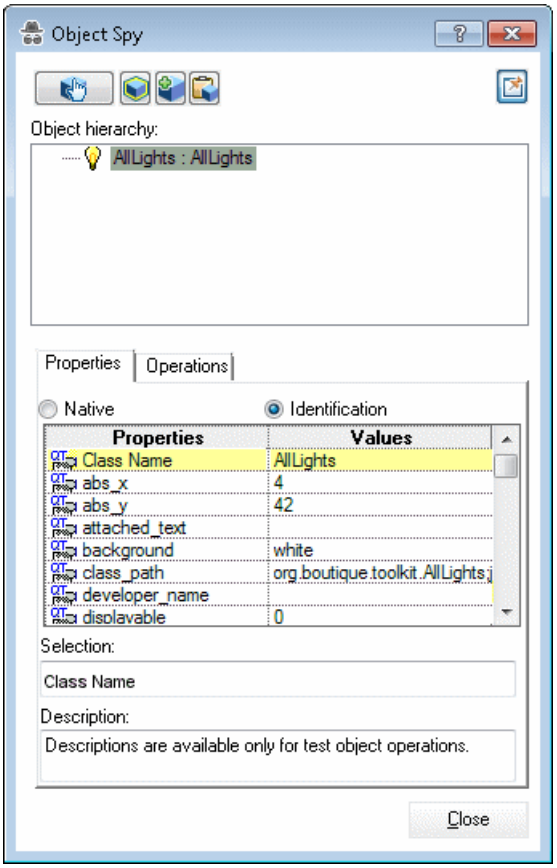
b **Javaboutique** のチェック・ボックスを選択し，［**OK**］をクリックします。UFT が開き，設計したサポートをロードします。

3 新しいカスタム・サポートをテストします。

244 ページ「AllLights コントロールのサポートの計画」の手順 2 および 3 を繰り返して，次の操作を実行します。

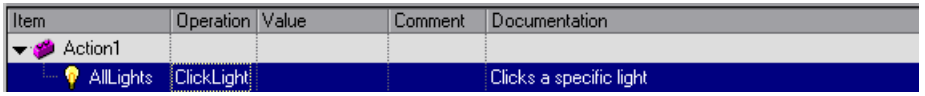
- アプリケーションを実行します（UFT はアプリケーションの起動時にアプリケーションとの接続を確立するため, SampleApp アプリケーションを終了してから再度実行する必要があります）。
- UFT オブジェクト・スパイを使用して，AllLights コントロールを表示します。
- AllLights コントロール上の **Click** 操作を記録します。

UFT は，（**to_class_attr** メソッドに従って）AllLights コントロールを AllLights という名前（カスタム・クラスの名前）の AllLights テスト・オブジェクトとして認識します。オブジェクト・スパイには，このテスト・オブジェクト・クラスに対してウィザードで指定したアイコンが表示されます。



低レベル記録のオーバーライドは完了していますが，**mouseClicked (MouseEvent arg0)** イベント・ハンドラ・メソッドの実装が未完了であるため，アプリケーションのフレームをクリックしても UFT では何も記録されません。

UFT で，AllLights オブジェクトをオブジェクト・リポジトリに追加し，キーワード・ビューでこのオブジェクトを含むテスト・ステップを作成します。



標準設定では，このステップの操作（Operation）として，ClickLight テスト・オブジェクト・メソッドが選択されます。このメソッドに必要な引数を提供し，このステップを含むテストを実行した場合，実行時エラーが発生します。これは，**ClickLight_replayMethod** メソッドが .NOT_IMPLEMENTED を返すためです。

AllLights コントロールのサポートの実装

ここでは，作成した計画（244 ページ「AllLights コントロールのサポートの計画」）に基いて，AllLightsCS クラスを変更して AllLights コントロールの UFT サポートを拡張します。

AllLightsCS.java ファイルを開きます。**label_attr** メソッドで，コード return super.label_attr(obj); をコード return "Lights"; で置き換えて，テスト・オブジェクトの名前を変更します。その後，次の手順を実行します。

➤ 新しい認識プロパティのサポートの実装（275 ページを参照）

➤ 新しいテスト・オブジェクト・メソッドに対するサポートの実装（277 ページを参照）

➤ 記録のサポートの実装（278 ページを参照）

➤ 完成したサポートのテスト（280 ページを参照）

新しい認識プロパティのサポートの実装

ここでは，AllLights テスト・オブジェクト・クラスに定義した新しい認識プロパティをサポートするメソッドを実装します。

AllLights カスタム・クラスを分析して，サポートされているプロパティを確認します。関連する認識プロパティを UFT に提供するために，新しいサポート・クラスからアクセスできるプロパティを決定します。

パブリック・メソッド GetcounterOn（ある時点でオンになっているライトの数を確認できる）と isSet（特定のライトのステータスを示す）に注目してください。

1 oncount_attr メソッドを実装します。

oncount_attr メソッドで，コード return null; を
return String.valueOf(((AllLights)obj).GetcounterOn()); で置き換えます。

この実装によって，AllLights カスタム・クラスからカウンタが取得されて UFT に返されます。

2 onlist_attr メソッドを実装します。

onlist_attr メソッドで，コード return null; を削除し，次のように，すべてのライトをスキャンして，オンになっているすべてのライトのリストを作成するメソッドを実装します。

```
public String onlist_attr (Object obj) {
    AllLights lights = (AllLights) obj;
    StringBuffer buffer = new StringBuffer();
    for (int i=0; i<5; i++)
        for (int j=0;j<5;j++)
            if (lights.isSet(j,i)) {
                buffer.append (" ");
                buffer.append (i*5+j+1);
                }
    return buffer.toString();
}
```

3 gameover_attr メソッドを実装します。

gameover_attr メソッドで，コード return null; を削除し，次のように，すべてのライトがオンになっているかどうかに応じて Yes または No を返すメソッドを実装します。

```
public String gameover_attr(Object obj) {
    if (((AllLights) obj).GetcounterOn() == 25)
        return "Yes";
    return "No";
}
```

［**File**］＞［**Save**］を選択するか，［**Save**］ボタンをクリックして **AllLightsCS.java** ファイルを保存します。

新しいテスト・オブジェクト・メソッドに対するサポートの実装

ここでは，AllLights テスト・オブジェクト・クラスに対して定義した新しいテスト・オブジェクト・メソッドをサポートするメソッドを実装します。

AllLights カスタム・クラスのメソッドを分析し，ユーザが［**RESTART**］ボタンまたはグリッド内のライトをクリックしたときに，クラスによって実行されるアクションを決定します。UFT でテスト・オブジェクト・メソッドを実行する際に，これらのアクションをシミュレートする必要があります。

1 Restart_replayMethod メソッドを実装します。

ユーザが［**RESTART**］ボタンの境界の内側をクリックすると，AllLights カスタム・クラスは **init** と **update(lights.getGraphics())** を呼び出し，アプリケーションを初期化して再描画します。**Restart_replayMethod** メソッドでは，同じメソッドを呼び出して，この動作をシミュレートする必要があります。

これを行うには，コード return Retval.NOT_IMPLEMENTED; を削除して，次のようにメソッドを実装します。

```
public Retval Restart_replayMethod (Object obj){
    AllLights lights = (AllLights) obj;
    lights.init();
    lights.update(lights.getGraphics());
    return Retval.OK;
}
```

2 ClickLight_replayMethod メソッドを実装します。

AllLights カスタム・クラスは，mouseUp イベントを受け取ったときに，クリックに応じてライトをオンまたはオフにするアルゴリズムを実行します。そのため，UFT で ClickLight_replayMethod を実行し，特定のライトでのクリックをシミュレートする場合は，AllLights オブジェクトに適切な座標を含む mouseUp イベントを送信するだけで済みます。

メソッド **ClickLight_replayMethod** で，コード return Retval.NOT_IMPLEMENTED; を削除し，次のようにメソッドを実装します。

```
public Retval ClickLight_replayMethod(Object obj, String Row, String Column) {
    AllLights lights = (AllLights) obj;
    int col_num = Integer.valueOf(Column).intValue();
    int row_num = Integer.valueOf(Row).intValue();
    /* Row and column are 40 pixels wide*/
    Event event = new Event (lights, System.currentTimeMillis(),
        Event.MOUSE_UP, col_num *40, row_num *40, 0, 0);
    lights.mouseUp(event, col_num *40, row_num *40);
    return Retval.OK;
}
```

注 : このコードをサポートするには，**AllLightsCS.java** で **java.awt.Event** をインポートします。

［**File**］＞［**Save**］を選択するか，［**Save**］ボタンをクリックして **AllLightsCS.java** ファイルを保存します。

記録のサポートの実装

AllLights コントロールでの記録をサポートするように計画しているため，このオブジェクト上での低レベル記録を抑止し，このコントロール上でマウス・イベントをリッスンするように登録しました。

AllLights コントロール上で記録をトリガする必要のあるマウス・イベントは，マウス・クリックだけです。そのため，ここでは，**mouseClicked (MouseEvent arg0)** イベント・ハンドラ・メソッドのみを実装し，その他のイベント・ハンドラ・メソッドは空のままにします。

mouseClicked メソッドを次のように実装し，**AllLightsCS.java** ファイルを保存します。

```
public void mouseClicked(MouseEvent arg0) {
    AllLights lights = (AllLights) arg0.getSource();
    int x = arg0.getX();
    int y = arg0.getY();

    try {
        if (!isInRecord())
            return;
        /* If click is within the Restart button borders*/
        if ((x > 210) && (x < 270) && (y > 165) && (y < 185)) {
            MicAPI.logLine(DEBUG_ALLLIGHTSCS, "Record Restart operation");
            MicAPI.record(lights, "Restart");
        }

        /* If click is within the borders of the grid - record a ClickLights*/
        if ((x >= 0) && (x < 200) && (y >= 0) && (y < 200)) {
            MicAPI.logLine(DEBUG_ALLLIGHTSCS, "Record ClickLight operation");
            MicAPI.record(lights, "ClickLight", new String[]
                {String.valueOf(y/40), String.valueOf(x/40)});
        }
    } catch (Throwable th) { MicAPI.logStackTrace(th);}
}
```

注：AllLightsCS.java ファイルの作成時に，このコードのサポートに必要な import com.mercury.ftjadin.custom.MicAPI がウィザードによって自動的に追加されています。

このイベント・ハンドラ・メソッドでは，**MicAPI.record** をさまざまな方法で呼び出します。Restart 操作を記録する場合は，オブジェクトと操作名を指定するだけです。ClickLight 操作を記録する場合は，クリックしたライトの座標を示す引数を併せて指定します。

isInRecord メソッドを呼び出すのは，UFT が記録を行っていない場合に，必要のない操作を実行しないようにするためです。

MicAPI.logLine メソッドは，DEBUG_ALLLIGHTSCS フラグがオンである場合にのみ，メッセージをログ・ファイルに出力します。詳細については，88 ページ「カスタム・サポート・クラスのログとデバッグ」を参照してください。

try ... catch ブロックは，UFT がアイドル状態のときに Java アプリケーションを実行してこのコードに到達した場合に，必要のない動作が実行されないようにします。**MicAPI.logStackTrace** メソッドは，スタック・トレースを Java Add-in Extensibility のほかのログ・メッセージと同じログ・ファイルに出力します。これにより，**mouseClicked** メソッドが誤って呼び出されたタイミングを決定することができます。

完成したサポートのテスト

ここでは，完成した Javaboutique ツールキット・サポートをテストします。そのためには，UFT での AllLights コントロールの表示がどのような影響を受けるかを分析します。

変更したのは Java のクラス・ファイルのみで，設定ファイルは変更していないため，ツールキット・サポートを UFT に再度デプロイしてテストする必要はありません。開いている UFT セッション（Javaboutique ツールキット・サポートをロードした状態で実行中）を使用することは可能ですが，カスタム・サポートに対する変更を反映するには，AllLights アプリケーションを終了して再度実行する必要があります。

1 オブジェクト・スパイで新しいカスタム・サポートをテストします。

a AllLights アプリケーションを終了し，再度実行します。

b UFT を起動し, Java Add-in と Javaboutique ツールキット・サポートをロードします。

c GUI テストを開き，オブジェクト・スパイを使用して AllLights のプロパティとメソッドを表示します。AllLights テスト・オブジェクトの名前が **Lights** になります。

d オブジェクト・スパイを閉じます。

2 AllLights コントロール上の UFT テストを作成して実行します。

a AllLights コントロールをテスト・オブジェクト・リポジトリに追加します。

b グリッド内の 2 つの場所をクリックし，ゲームが終了していないことを確認して，［RESTART］をクリックするテストを作成します。

作成するテストは，次のようになります。

Item	Operation	Value	Comment	Documentation
Action1				
Lights	ClickLight	"4","4"		Click the light in row "4" column "4".
Lights	ClickLight	"1","2"		Click the light in row "1" column "2".
Lights	Check	CheckPoint("Lights")		Check whether the "Lights" object has the proper value
Lights	Restart			Click the RESTART button.

注：ClickLight_replayMethod は，引数の値をチェックして値が有効であるかどうかを確認しません。範囲外の引数を指定した場合（列または行が 4 より大きい場合），実行時エラーが発生します。

---

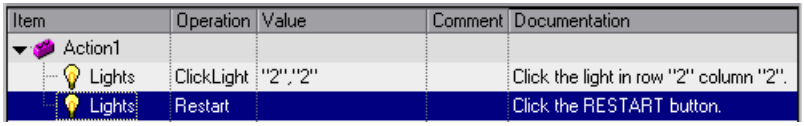
c テストを実行して，正しく動作することを確認します（ゲームが終了していないことだけを確認するようにチェックポイントを定義した場合，テストは成功します）。

3 AllLights コントロール上の操作を記録します。

a UFT で，新しい UFT テストを作成し，［**記録**］＞［**記録と実行環境設定**］を選択して，［記録と実行環境設定］ダイアログ・ボックスを開きます。［Java］タブで，［**開いているすべての Java アプリケーションでテストを記録して実行する**］を選択します。Web アドインもロードされる場合は，［**Web**］タブをクリックして［**開いているすべてのブラウザでテストを記録して実行する**］を選択します。［**OK**］をクリックします。

b ［**記録**］ボタンをクリックするか，［**記録**］＞［**記録**］を選択します。グリッド，［**RESTART**］ボタン，カウンタなど，AllLights アプリケーション内のさまざまな場所をクリックします。

グリッド内をクリックすると，テストに ClickLight ステップと関連する引数が追加されます。［RESTART］ボタンをクリックすると，Restart ステップが追加されます。ほかの場所をクリックしても，操作は記録されません（低レベル・マウス・イベント記録を無効にしたため）。記録されたテストは，次のようになります。



c ［**停止**］ボタンをクリックするか，［**記録**］＞［**停止**］を選択して記録セッションを終了します。

これで，カスタム・サポートの計画時に決めた仕様に従って，AllLights カスタム・コントロールが完全にサポートされました。

レッスンのまとめ

このレッスンでは，認識プロパティとテスト・オブジェクト・メソッドを定義して，新しいテスト・オブジェクト・クラス（AllLights）を作成しました。また，AllLights コントロールのサポートを作成し，UFT で AllLights テスト・オブジェクトとして認識されるようにしました。

- テスト・オブジェクト設定ファイルについて学習しました。
- カスタム・サポート・クラスでの新しい認識プロパティとテスト・オブジェクト・メソッドのサポート方法について学習しました。
- **is_window** ユーティリティ・メソッドを使用し，追加パラメータを使用して **MicAPI.record** メソッドを呼び出しました。

その他の情報

これで，このチュートリアルのレッスンはすべて終了しました。このレッスンで学習した Java Add-in Extensibility の考え方やスキルを各自のカスタム・ツールキット・サポートの作成に活用してください。

カスタム・ツールキット・サポート・セットの構造と内容の詳細は，35 ページ「カスタム・ツールキットのサポートの実装」を参照してください。

テスト・オブジェクト設定ファイルの構造と内容の詳細は，『UFT Test Object Schema ヘルプ』（『Java Add-in Extensibility SDK ヘルプ』で利用可能）を参照してください。