HP UFT .NET 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。

SlickEdit®は，SlickEdit Inc.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

.NET Add-in Extensibility へようこそ

HP UFT .NET Add-in Extensibility は, Unified Functional Testing .NET 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 .NET Windows Forms 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 .NET Add-in Extensibility SDK について

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

- Unified Functional Testing .NET Add-in を拡張する API。これにより，.NET Windows Forms のカスタム・コントロールをサポートします。
- Microsoft Visual Studio 用の Custom Server C# および Visual Basic プロジェクト・テンプレート。

注：サポート対象の Microsoft Visual Studio 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。

カスタム・サーバ・テンプレートには，空コードのフレームワーク，サンプル・コード，UFT プロジェクト・リファレンスなど，カスタム・サーバのビルドに必要な項目が含まれています。

- ウィザード。プロジェクトの新規作成時にカスタム・サーバ・テンプレートを選択すると起動します。.NET Add-in Extensibility を使用してカスタム・サーバ .NET DLL を作成するための Microsoft Visual Studio プロジェクトを簡単にセットアップできます。詳細については，58 ページ「.NET DLL を使用してカスタム・コントロールのサポートを拡張」を参照してください。
- この .NET Add-in Windows Forms 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**］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。
  - .NET Add-in Extensibility Configuration Schema ヘルプ
  - .NET Add-in Extensibility Control Definition 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）。
- サンプルの .NET Add-in Extensibility サポート・セット。UFT GUI テストを拡張することによって SandBar ツールバー・カスタム・コントロールをサポートします。

このガイドについて

このガイドでは，UFT .NET 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 .NET Windows Forms 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，.NET Add-in Extensibility オンライン・ヘルプ（[**スタート**] > [**すべてのプログラム**] > [**HP Software**] > [**HP Unified Functional Testing**] > [**Extensibility**] > [**Documentation**] > [**.NET Add-in Windows Forms Extensibility Help**]）の次の項目も参照してください。

- UFT .NET Add-in Extensibility API Reference
- UFT .NET Add-in Extensibility Systems Forms Configuration Schema ヘルプ
- UFT .NET Add-in Extensibility Control Definition Schema ヘルプ
- HP Unified Functional Testing Test Object Schema Help

このドキュメントに併せて，次のドキュメント（UFT と一緒にインストールされます。UFT メイン・ウィンドウの [**ヘルプ**] > [**HP Unified Functional Testing ヘルプ**] からアクセスできます）も参照してください。

- 『HP Unified Functional Testing ユーザーズ・ガイド』
- 『HP Unified Functional Testing アドイン・ガイド』の .NET に関する説明
- 『HP Unified Functional Testing Object Model Reference』

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 .NET Windows Forms 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 .NET Add-in
- C# または Visual Basic での .NET 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 1 章

UFT .NET Add-in Extensibility の概要

UFT .NET Add-in Extensibility を使用することで，Unified Functional Testing .NET Add-in では標準ではサポートされないサードパーティ製またはカスタマイズされた .NET Windows Forms コントロールに対して高いレベルのサポートを提供することができます。

本章の内容

UFT .NET Add-in Extensibility について

Unified Functional Testing .NET Add-in は，よく使用される各種 .NET Windows Forms コントロールのサポートを提供します。UFT .NET Add-in Extensibility を使用すると，.NET Add-in では標準でサポートされないサードパーティ製およびユーザ定義の .NET Windows Forms コントロールをサポートできます。

UFT がアプリケーション内のオブジェクトを学習する場合，オブジェクトを特定のテスト・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。

Extensibility を使用しない場合, 標準でサポートされない .NET Windows Forms コントロールは，UFT GUI テストで汎用の **SwfObject** テスト・オブジェクトによって表されます。ただし，汎用テスト・オブジェクトにはテスト対象の .NET Windows Forms コントロールに固有の特性が含まれてい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するテスト・ステップを実行しようとするときに，使用可能なテスト・オブジェクト・メソッドでは十分ではない場合があります。さらに，サポートされていないコントロールに関するテストを記録すると，ウィンドウ・メッセージとして送信される低レベル・アクティビティが記録したステップに反映されてしまい，コントロールの動作が的確に記録されません。

UFT .NET Add-in Extensibility を使用することにより，UFT はカスタム .NET Windows Forms コントロールを的確に認識することができるようになります。カスタム・コントロールを既存の UFT テスト・オブジェクトにマッピングすると，UFT のステートメント自動補完機能やより有効なテスト・ステップを作成する機能をはじめとする UFT テスト・オブジェクトの全機能を活用できます。

注 : UFT が .NET コントロールを標準で認識し，コントロールを表すのに SwfObject ではなく .NET アドインのテスト・オブジェクトを使用する場合，このコントロールはこれ以外のテスト・オブジェクト・タイプにマッピングできません。

既存のテスト・オブジェクト・メソッドの動作では，カスタム・コントロールに対応できないことがあります。このような場合には，既存のテスト・オブジェクト・メソッドの動作を変更する方法や，コントロールで必要な動作を定義する新しいメソッドを使用して UFT テスト・オブジェクトを拡張する方法で対応できます。

カスタム・サーバを作成し，アプリケーション内のコントロールでメソッドを実行する .NET Add-in インタフェースを拡張します。カスタム・サーバでは，既存のメソッドのオーバーライドと新規メソッドの作成が可能です。

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

Unified Functional Testing .NET Add-in を使用することにより，ほとんどの .NET Windows Forms コントロールで必要なレベルのサポートを提供できます。カスタム .NET Windows Forms コントロールのサポートを拡張する場合には，UFT の機能面からサポートの範囲を分析し，変更が必要なサポート要素を特定します。

カスタム .NET Windows Forms コントロールの分析には，.NET Windows Forms 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できます。16 ページ「カスタマイズ可能な UFT サポート要素」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機能や動作が十分でない場合には，次の場合で示されるように，.NET Windows Forms コントロールで .NET Add-in Extensibility を使用することをお勧めします。

- UFT は汎用 SwfObject としてコントロールを認識しているが，これよりも適切な動作をするテスト・オブジェクト・クラスが別に存在するケース。この場合，.NET Add-in Extensibility を使用して，このテスト・オブジェクト・クラスにコントロールをマッピングします。
- UFT がユーザ・ニーズに合わないテスト・オブジェクトを使用してコントロールを認識しているケース。この場合，.NET Add-in Extensibility を使用してメソッドを変更すれば，テスト・オブジェクトの機能を変更できます。
- UFT が，カスタム・コントロール内のサブコントロールを個々に識別できるが，メイン・コントロールを正しく識別できないケース。たとえば，メインのカスタム・コントロールがデジタル時計で，時間と分のエディット・ボックスがあるとします。時間の変化を，エディット・ボックスの **Set** 操作としてではなく，クロック・コントロールの **SetTime** 操作として認識する必要がある場合，.NET Add-in Extensibility でメッセージ・フィルタを設定して子コントロールからのメッセージを処理し，子コントロールでのイベントに応じてメイン・コントロールでの操作を記録します。

カスタマイズ可能な UFT サポート要素

UFT GUI テスト サポートには次のような要素が含まれています。これらの要素のうちのいくつかのサポートを拡張することで，必要なサポートを開発して，有意義で保守が容易なテストを作成できます。

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，特定のテスト・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。テスト・オブジェクト・クラスによって，UFT で使用できる認識プロパティとテスト・オブジェクト・メソッドが決まります。また，コントロールを表すのに別のテスト・オブジェクト・クラスを使用する設定も可能です。

テスト・オブジェクト・メソッド

.NET Windows Forms コントロールを表すのに使用されるテスト・オブジェクト・クラスによって，テスト・オブジェクトで使用できるテスト・オブジェクト・メソッドが決まります。ただし，同じテスト・オブジェクト・クラスのテスト・オブジェクトで表される異なる .NET Windows Forms コントロールに対して，同じテスト・オブジェクト・メソッドの動作が異なる場合があります。UFT では，.NET Windows Forms コントロールのタイプによって，テスト・オブジェクト・メソッドの実行方法が異なるケースがあるからです。

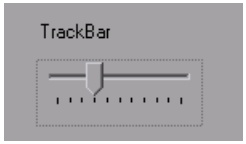
イベントの記録

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，対応するテスト・ステップを登録します。.NET Windows Forms コントロールで使用するテスト・オブジェクト・クラスとカスタム・サーバに基づいて，UFT が .NET Windows Forms コントロールでリッスンできるイベントと，発生するイベントで記録するテスト・ステップが決まります。

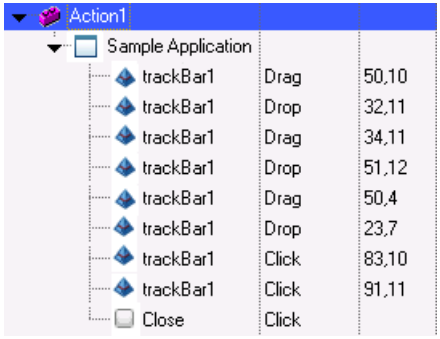
例：イベントの意味のある動作の記録をカスタマイズ

意味のあるコントロールの動作とは，テスト対象となる動作を指します。たとえば，アプリケーションのラジオ・ボタン・グループに含まれるボタンをクリックする場合，記録が必要になるのは，ユーザが選択した値であり，**Click** イベントやクリック位置の座標ではありません。したがって，ラジオ・ボタン・グループの場合，意味のある動作とは選択内容の変更ということになります。

コントロールのサポートを拡張しない状態でカスタム・コントロールに関するテストまたはビジネス・コンポーネントを記録すると，低レベルのコントロール動作が記録されてしまいます。次に示す例は .NET のサンプル・アプリケーションにある TrackBar コントロールですが，対応する UFT テスト・オブジェクトがありません。



TrackBar の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。

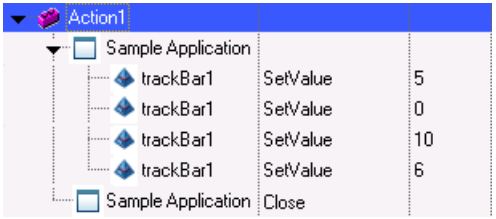


エディタでは，記録されたテストは次のように表示されます。

```
SwfWindow("Sample Application").SwfObject("trackBar1").Drag 50,10
SwfWindow("Sample Application").SwfObject("trackBar1").Drop 32,11
SwfWindow("Sample Application").SwfObject("trackBar1").Drag 34,11
SwfWindow("Sample Application").SwfObject("trackBar1").Drop 51,12
SwfWindow("Sample Application").SwfObject("trackBar1").Drag 50,4
SwfWindow("Sample Application").SwfObject("trackBar1").Drop 23,7
SwfWindow("Sample Application").SwfObject("trackBar1").Click 83,10
SwfWindow("Sample Application").SwfObject("trackBar1").Click 91,11
SwfWindow("Sample Application").SwfButton("Close").Click
```

Drag，**Drop**，**Click** の各メソッドは TrackBar の低レベル・アクションであり，コントロール内の特定の座標で記録されています。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。

これに対して，.NET Add-in Extensibility を使用して TrackBar コントロールをサポートすると，有効な結果を取得できます。下の図は，カスタマイズした **SetValue** メソッドを実装するカスタム・サーバを使用して，TrackBar でテストを記録したときのキーワード・ビューです。



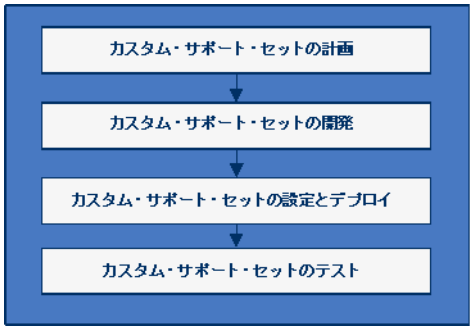
エディタでは，記録されたテストは次のように表示されます。

```
SwfWindow("Sample Application").SwfObject("trackBar1").SetValue 5
SwfWindow("Sample Application").SwfObject("trackBar1").SetValue 0
SwfWindow("Sample Application").SwfObject("trackBar1").SetValue 10
SwfWindow("Sample Application").SwfObject("trackBar1").SetValue 6
SwfWindow("Sample Application").Close
```

カスタム・テスト・オブジェクトを使用しない場合には **Drag**，**Drop**，**Click** など低レベル操作が記録されていましたが，このような操作は記録されなくなり，新しいスライダ位置を示す **SetValue** 操作が記録されています。この情報はわかりやすく，テストの変更作業も簡単になります。

.NET Add-in Extensibility の実装方法

.NET Add-in Extensibility サポート・セットを開発することによって，カスタム・コントロール・セットの .NET Add-in Extensibility サポートを実装します。.NET Add-in Extensibility サポート・セットは次の工程に従って開発します。次に，それぞれの手順を詳しく説明します。



.NET Add-in Extensibility サポート・セットの計画

.NET Add-in Extensibility サポート・セットの基本要素を適切に構築するには，UFT がカスタム・コントロールを認識する方法を詳細に計画する必要があります。一般的に，サポート・セットの計画では次の作業を行います。

- サポートのカスタマイズが必要な .NET Windows Forms コントロールを特定します。
- テストが必要なコントロールとビジネス・プロセスに基づいて，サポート対象となるテスト・オブジェクトと操作を特定することによって，テスト・オブジェクト・モデルの計画を立てます。
- サポートを実装するのに最適な方法を計画します。

詳細については，35 ページ「サポート・セットの計画」を参照してください。

.NET Add-in Extensibility サポート・セットの開発

.NET Add-in Extensibility サポート・セットの開発は，次の手順で行います。

➤ テスト・オブジェクト・モデルの定義

➤ カスタム・サーバの作成

➤ カスタム・コントロールからテスト・オブジェクト・クラスへのマッピング

次に，それぞれの作業を詳しく説明します。

テスト・オブジェクト・モデルの定義

アプリケーションとコントロールのテストで UFT が使用するテスト・オブジェクト・モデルを定義します。テスト・オブジェクト・モデルは，ユーザ環境でカスタム・コントロールを表すテスト・オブジェクト・クラスと，そのテスト・オブジェクト・メソッドを含むリストです。

テスト・オブジェクト・モデルは，テスト・オブジェクト設定 XML ファイルで定義します。詳細については，48 ページ「テスト・オブジェクト・モデルの定義」を参照してください。

カスタム・サーバの作成

各カスタム・コントロールを処理するカスタム・サーバ（DLL またはコントロール定義 XML ファイル）を作成します。カスタム・サーバでは，次の内容を変更できます。

➤ 記録セッション中に記録するステップ

➤ テスト・オブジェクト・メソッドの実装

➤ テーブル・チェックポイントおよび出力値のサポート

カスタム・サーバは，UFT と .NET アプリケーション間を連携します。記録セッション中，カスタム・サーバはイベントをリッスンし，ユーザ・アクティビティを有効なテスト・オブジェクト・メソッドにマッピングします。テスト実行中，カスタム・サーバは .NET Windows Forms コントロールでテスト・オブジェクト・メソッドを実行します。

カスタム・サーバのコーディング・オプション

カスタム・サーバは，次のコーディング・オプションのいずれかで実装可能です。

- .NET DLL
- XML。スキーマ・ベース（.NET DLL カスタム・サーバの作成で UFT が使用）

詳細については，次を参照してください。

- 58 ページ「.NET DLL を使用してカスタム・コントロールのサポートを拡張」
- 83 ページ「XML ファイルを使用してカスタム・コントロールのサポートを拡張」

カスタム・サーバの実行時コンテキスト

カスタム・サーバで提供されるクラスは，次のソフトウェア・プロセス（実行時コンテキスト）でインスタンス化される可能性があります。

- **Application under test** コンテキスト：テスト中のアプリケーションのコンテキストで作成されたオブジェクトは，.NET Windows Forms コントロールのイベント，メソッド，プロパティに直接アクセスできます。ただし，ウィンドウ・メッセージのリッスンはできません。
- **UFT** コンテキスト：UFT コンテキストで作成されたオブジェクトは，ウィンドウ・メッセージをリッスンできます。ただし，.NET Windows Forms コントロールのイベント，メソッド，プロパティには直接アクセスできません。

カスタム・サーバが .NET DLL として実装されている場合，UFT で作成されたオブジェクトは，テスト中のアプリケーションで稼働する補助クラスを作成できます。

実行時コンテキストの詳細については，39 ページ「テスト関数に応じてカスタム・サーバ実行時コンテキストを選択」を参照してください。

補助クラスの詳細については，58 ページ「.NET DLL を使用してカスタム・コントロールのサポートを拡張」と『UFT .NET Add-in Extensibility API Reference』を参照してください。

カスタム・コントロールをテスト・オブジェクトにマッピング

テスト・オブジェクトへのマッピングには，.NET Add-in Extensibility 設定ファイル（**SwfConfig.xml**）を使用します。このファイルは **<UFT インストール・パス>\dat** フォルダに格納されており，次のデータが保存されています。

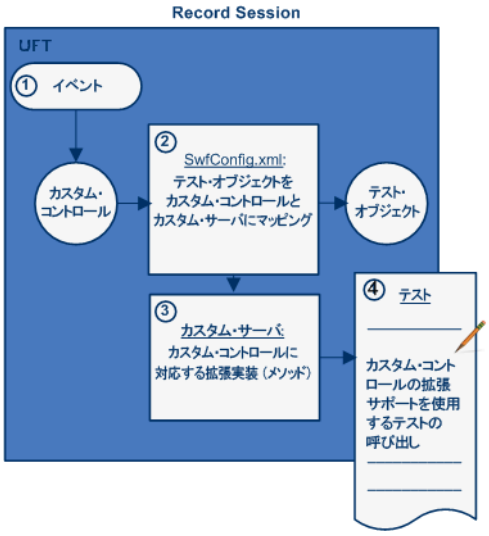
- カスタム・コントロールからテスト・オブジェクトへのマッピング。
- 対応するカスタム・サーバへのマッピング。このマッピングを行うことによって，UFT テスト・オブジェクトで全機能が利用可能になります。

詳細については，57 ページ「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」を参照してください。

次の図は，それぞれ記録セッションと実行セッションで，.NET Add-in Extensibility がカスタム・コントロールをテスト・オブジェクトとカスタム・サーバにマッピングする方法を示しています。

UFT がカスタム・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する方法（記録セッション）

次の図と表は，UFT が記録セッションで，カスタム・コントロールをテスト・オブジェクトにマッピングし，カスタム・コントロールに対応する拡張実装を特定し，必要なテスト・ステップを記録する方法を示しています。

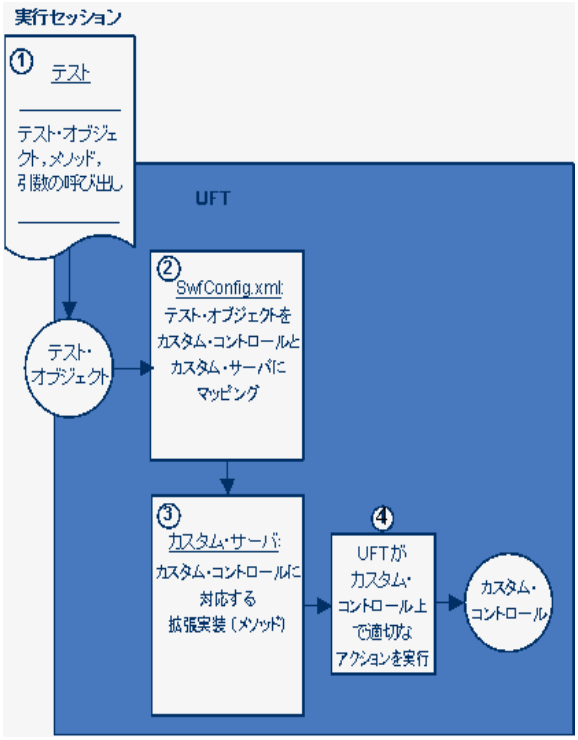


ステップ	説明
1	UFT が認識できないタイプのコントロール，または記録の実装がカスタマイズされているタイプのコントロールでイベントが発生します。
2	UFT は，**SwfConfig.xml** ファイルで **Control** 要素の **Type** 属性をチェックし，このタイプのカスタム・コントロールの情報を取得します。次に UFT は，**MappedTo** 属性をチェックし，このタイプのコントロールにマッピングされているテスト・オブジェクト・クラスを特定します。テスト・オブジェクト・クラスが指定されていない場合は，**SwfObject** が使用されます。

ステップ	説明
3	UFT は，**SwfConfig.xml** ファイルで **DLLName** 要素をチェックし，このタイプのカスタム・コントロールの実装を含むカスタム・サーバを特定してカスタム・サーバと通信します。
4	カスタム・サーバは，イベント発生時にテストに追加するステップを UFT に指示します。

UFT がカスタム・コントロールをカスタム・サーバにマッピングする方法（実行セッション）

次の図と表は，UFT が実行セッションで，カスタム・コントロールをテスト・オブジェクトにマッピングし，カスタム・コントロールに対応する拡張実装を特定して，カスタム・コントロール上で適切な操作を実行する方法を示しています。



ステップ	説明
1	テストを実行します。このテストには，実装をカスタマイズしたカスタム・コントロールを表すテスト・オブジェクトが含まれています。
2	UFT は，**SwfConfig.xml** ファイルの **Control** 要素で，このテスト・オブジェクトにマッピングされているカスタム・コントロールの情報を取得します。
3	UFT は，**SwfConfig.xml** ファイルで **DLLName** 要素をチェックし，このタイプのカスタム・コントロールの実装を含むカスタム・サーバを特定します。
4	UFT は，カスタム・コントロールの実装で定義された内容に従って，テスト・オブジェクト操作の適切な実装を使用してテストを実行します。

.NET Add-in Extensibility サポート・セットのデプロイ

.NET Add-in Extensibility サポート・セットをデプロイし，UFT がコントロールをサポートできるようにするには，作成したファイルを UFT インストール・フォルダ内の所定の場所にコピーする必要があります。

詳細については，89 ページ「サポート・セットの設定とデプロイ」を参照してください。

.NET Add-in Extensibility サポート・セットのテスト

コントロールで使用する .NET Add-in Extensibility サポートを作成したら，.NET Add-in Extensibility サポート・セットをテストします。

.NET Add-in Extensibility サポート・セットの具体的な開発方法は，第 6 章「簡単な .NET Windows Forms カスタム・コントロールの作成の実習」と第 7 章「複雑な .NET Windows Forms カスタム・コントロールの作成の実習」のレッスンを参考にしてください。

第 2 章

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

本章では，HP UFT .NET Add-in Extensibility SDK のインストール・プロセスについて説明します。

HP UFT .NET Add-in Extensibility SDK でインストールされる項目については，8 ページ「UFT .NET Add-in Extensibility SDK について」を参照してください。

本章の内容

インストールの前に

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

- ➤ Unified Functional Testing インストール DVD が使用可能であること。
- ➤ Microsoft Visual Studio のサポート対象バージョンがコンピュータにインストールされていること。

注 : サポート対象の Microsoft Visual Studio 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。

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

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

注 : UFT .NET Add-in Extensibility SDK を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 .NET Add-in Extensibility SDK をインストールするには, 次の手順で行います。

1. Microsoft Visual Studio のすべてのインスタンスを閉じます。
2. CD-ROM または 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 ［**UFT .NET Add-in Extensibility SDK Setup**］をクリックします。UFT .NET Add-in Extensibility SDK のセットアップ・ウィザードが起動します。

注: ウィザード画面に SDK インストールの修復または削除を選択するオプションが表示された場合，UFT .NET Add-in Extensibility SDK は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（34 ページ「HP UFT .NET Add-in Extensibility SDK のアンインストール」を参照してください）。

---

5 ウィザードの指示に従って，インストールを完了します。

6 セットアップ・ウィザードの最後の画面で［**Readme を表示する**］チェックボックスを選択して［**閉じる**］をクリックすると，UFT .NET 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**］＞［**.NET Add-in Extensibility Readme**］を選択します。

7 ［**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。

8 英語以外のバージョンの Visual Studio を使用している場合は，次の手順に従って，インストール済みの **UFT CustomServer** プロジェクト・テンプレートを Visual Studio エディションに適用します。

注 : 次の手順は，Visual Studio 2008 を 32 ビット版オペレーティング・システムにインストールした環境で行います。Visual Studio 2010 を使用している場合や 64 ビット版オペレーティング・システムを使用している場合には，フォルダ名とファイル名が若干異なります。

---

a **QuickTestCustomServerVB.zip** ファイルを，
C:\Program Files\Microsoft Visual Studio 9.0\Common7\IDE\ProjectTemplates\VisualBasic\Windows\1033（英語の設定フォルダ）から，使用する言語のフォルダ（フランス語の場合は 1036 など）にコピーします。

b **C:\Program Files\Microsoft Visual Studio 9.0\Common7\IDE** フォルダにある **PostCustomVizard.exe** プログラムを実行します。

c C# テンプレートについても上記の手順を実行します。
QuickTestCustomServer.zip ファイルを
C:\Program Files\Microsoft Visual Studio 9.0\Common7\IDE\ProjectTemplates\CSharp\Windows\1033 からコピーします。

インストールが成功したことを確認するために，次の手順を行います。

注：ここで掲載する Microsoft Visual Studio のダイアログ・ボックスの図と手順は，Microsoft Visual Studio 2008 環境の内容です。これ以外のバージョンの Microsoft Visual Studio では，ダイアログ・ボックスの表示や，ツリー構造内で **UFT CustomServer** テンプレートが表示されるノードの場所が若干異なる場合があります。

---

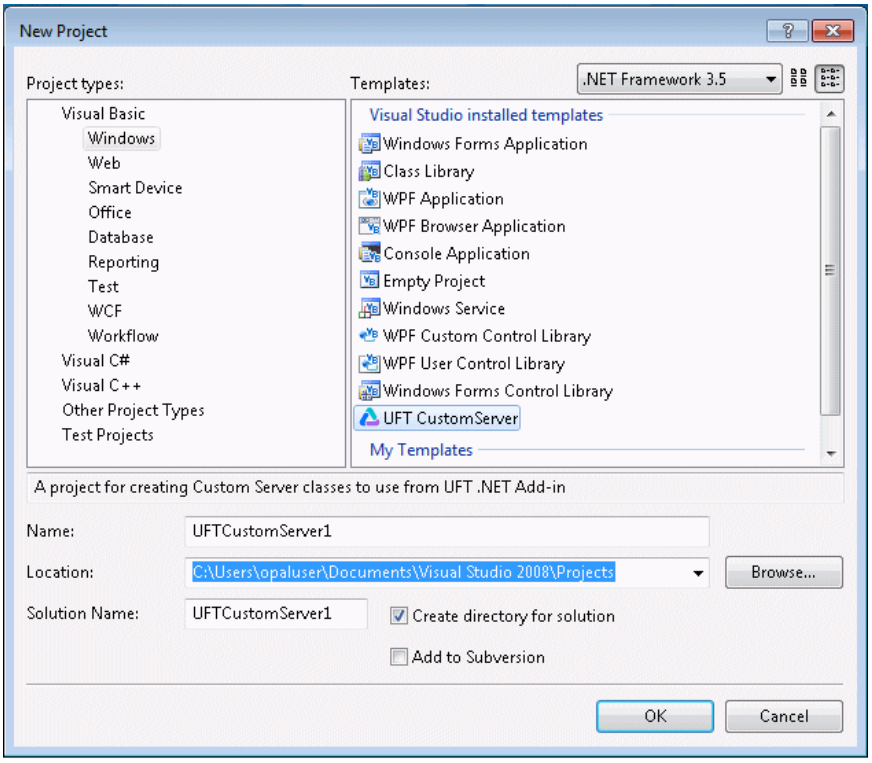
1 Microsoft Visual Studio のサポート対象バージョンを起動します。

サポート対象の Microsoft Visual Studio 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。

2 ［**ファイル**］＞［**新規作成**］＞［**プロジェクト**］を選択します。［プロジェクトの新規作成］ダイアログ・ボックスが開きます。

3 ［**プロジェクトのタイプ**］ツリーで［**Visual Basic**］＞［**Windows**］ノードを選択します。

4 **UFT CustomServer** テンプレート・アイコンが［**テンプレート**］表示枠に表示されていることを確認します。



5 ［**プロジェクトのタイプ**］ツリーで［**Visual C#**］＞［**Windows**］ノードを選択します。

6 **UFT CustomServer** テンプレート・アイコンが［**テンプレート**］表示枠に表示されていることを確認します。

注: 新しいバージョンの Microsoft Visual Studio にアップグレードする場合は, .NET Add-in Extensibility SDK をアンインストールしてから再インストールしないと **UFT CustomServer** テンプレートにアクセスできません。

HP UFT .NET Add-in Extensibility SDK の修復

既存の HP Unified Functional Testing .NET Add-in SDK インストール環境で欠落しているファイルや破損しているファイルがある場合，Unified Functional Testing のセットアップ・プログラムを使ってファイルを置換することによって修復します。

注：

- ➤ 最初のインストール時に使用したバージョンのセットアップ・プログラムを使用してください。
- ➤ 修復を行うには，管理者権限でログインする必要があります。
- ➤ オペレーティング・システムでユーザ・アカウント制御（UAC）が有効になっている場合，修復中は UAC を無効にしてください。

HP UFT .NET Add-in Extensibility SDK を修復するには，次の手順で行います。

1. CD-ROM または 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）。

2. ［**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。

3. ［**UFT .NET Add-in Extensibility SDK Setup**］をクリックします。.NET Add-in Extensibility SDK セットアップ・ウィザードが起動し，SDK の修復または削除を選択する画面が表示されます。

4. ［**修復**］を選択し，［**完了**］をクリックします。UFT .NET Add-in Extensibility SDK ファイルが置換され，［インストールの完了］画面が開きます。

5. ［**閉じる**］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。

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

HP Unified Functional Testing .NET Add-in 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。

注：

- ➤ 最初のインストール時に使用したバージョンのセットアップ・プログラムを使用してください。
- ➤ UFT .NET 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 .NET 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. CD-ROM または 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）。

2. ［**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。

3. ［**UFT .NET Add-in Extensibility SDK Setup**］をクリックします。.NET Add-in Extensibility SDK セットアップ・ウィザードが起動し，SDK の修復または削除を選択する画面が表示されます。

4. ［**削除**］を選択し，［**完了**］をクリックします。UFT .NET Add-in Extensibility SDK ファイルが削除され，［インストールの完了］画面が開きます。

5. ［**閉じる**］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。

第 3 章

サポート・セットの計画

カスタム・コントロールのサポー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。.NET Add-in Extensibility サポート・セットの基本要素を適切に構築するには，UFT でカスタム・コントロールを認識する方法を詳細に計画する必要があります。

本章の内容

注：本章は，第 1 章「UFT .NET Add-in Extensibility の概要」の概念を理解していることが前提になります。

.NET Add-in Extensibility コントロールで使用する UFT GUI テスト・サポートの計画について

カスタム .NET Windows Forms コントロールを認識する目的で Unified Functional Testing .NET Add-in のサポートを拡張するプロセスでは，綿密な計画が必要です。本章の項では，このプロセスをスムーズに進めるために，カスタム・コントロールのサポートを実装する上で考慮すべき点をチェックリストにまとめています。.NET Add-in Extensibility サポート・セットを作成する場合は，このチェックリストの回答に基づいて実装してください。

カスタム・コントロールの関連情報の確認

このサポート・セットがサポートするコントロールを決定します。

カスタム .NET Windows Forms コントロールのサポートを計画する作業では，コントロールに対するアクセス権限が割り当てられていることと，コントロールの動作を理解していることが前提となります。

コントロールの実際の動作を確認できるアプリケーションが必要です。

また，コントロールを実装するソースを参照できることも必要です。UFT でカスタム・コントロールをサポートするためにコントロールのソースを変更する必要はありませんが，ソースを理解しておく必要はあります。

特定のタイプのコントロールのカスタム・サポートを計画する際には，UFT でコントロールを認識する方法（UFT GUI テストでコントロールを表すテスト・オブジェクトのタイプ，使用するテスト・オブジェクト・メソッドなど）をよく検討する必要があります。このタイプのコントロールと，ユーザがこれらのコントロールで行う操作を使用してテストされる可能性のあるビジネス・プロセスに基づいて決定してください。

- コントロールがサポートするメソッド，コントロールのプロパティ，リッスン可能なイベントなどについて，内容を把握しておきます。
- 既存のテスト・オブジェクト・クラスの中で，カスタム .NET Windows Forms コントロールに類似した機能を持つものがないか確認します。
- コントロールをサポートする上で，記述または変更が必要なメソッドがあるか確認します。

カスタム・コントロールの分析

.NET Windows Forms Spy，キーワード・ビュー，記録オプションでは，カスタム・コントロールを含むアプリケーションを実行することにより，コントロールを UFT の観点から分析できます。これにより，カスタム・サポートがない状態で UFT がコントロールをどのように認識するかを調べて，何を変更すればよいかを決めることができます。

.NET Windows Forms スパイの使用

.NET Windows Forms スパイは，.NET Windows Forms コントロールの拡張を開発するのに役に立ちます。.NET Windows Forms スパイには次のような機能があります。

- 選択した .NET Windows Forms コントロールとその実行時オブジェクトのプロパティの詳細を表示します。
- アプリケーションの変更につながるイベントを特定し（記録と実行を行う拡張の実装を簡素化），この変更内容がコントロールの状態にどのような影響を与えるのかを確認します。

.NET Windows Forms スパイにアクセスするには，UFT のメイン・ウィンドウで［**ツール**］＞［**.NET Windows Forms Spy**］を選択します。

注：.NET Windows Forms アプリケーションを調査するには，アプリケーションを FullTrust で実行する必要があります。FullTrust で実行する設定が行われていないアプリケーションでは，.NET Windows Forms スパイで .NET Windows Forms コントロールを調査することはできません。.NET アプリケーションの信頼レベルの定義に関する詳細については，Microsoft のドキュメントを参照してください。

.NET Windows Forms スパイの詳細については，『HP Unified Functional Testing アドイン・ガイド』を参照してください。

特定のコントロールのサポートを計画する作業で検討すべき項目は，.NET Add-in Extensibility 計画チェックリストのリストにまとめられています。検討項目の内容について十分に理解している場合は，検討項目を簡単にまとめた印刷形式のチェックリスト（44 ページ）を使用してカスタム・サポート・クラスを設計することができます。

カスタム・サーバの実装で使用するコーディング・オプションの選択

カスタム .NET Windows Forms コントロールで使用するカスタム・サポートは，次の方法で実装できます。

- **.NET DLL**：.NET アセンブリを使用してコントロールのサポートを拡張します。
- **XML**：XML ファイルを使用して，スキーマに基づいてコントロールのサポートを拡張します。

.NET DLL：完全なプログラム開発環境

ほとんどのカスタム・サーバは，.NET DLL として実装されます。次のような理由から，このオプションを使用することをお勧めします。

- 構文チェック，デバッグ，Microsoft IntelliSense など，プログラム開発環境で提供されるすべてのサービスを利用できます。
- テーブル・チェックポイントおよび出力値のサポートが提供されるのは，カスタム・サーバを .NET DLL として実装した場合のみです。
- カスタム・サーバを .NET DLL として実装すると，UFT コンテキストで Test Record 関数の一部と，テスト対象アプリケーションのコンテキストで一部を実行できます。詳細については，58 ページ「.NET DLL を使用してカスタム・コントロールのサポートを拡張」と『UFT .NET Add-in Extensibility API Reference』（UFT .NET Add-in Extensibility オンライン・ヘルプで提供）を参照してください。

実行時コンテキストの詳細については，39 ページ「テスト関数に応じてカスタム・サーバ実行時コンテキストを選択」を参照してください。

XML の実装

次に示す場合には，XML コーディングを使用してカスタム・サーバを実装する方法が最適です。

- コントロールが比較的単純で，詳細なドキュメントが作成されている場合。
- コントロールが既存のオブジェクトにマッピングされている状態で，記録セッション（Test Record）中に実装を置換する必要がある場合や，実行セッション（Test Run）中に数個のテスト・オブジェクト・メソッドを置換または追加する必要がある場合。
- 完全なプログラミング環境を用意できない場合。XML カスタム・サーバを使用した実装で必要になるのはテキスト・エディタのみです。

ただし, XML でカスタム・コントロールを実装する方法には, 次のような欠点もあります。

- プログラム開発環境で提供されるサポートを利用できません。
- XML の実装には C# プログラミング・コマンドが含まれ，**Application under test** コンテキスト内でのみ実行可能です。

詳細については，83 ページ「XML ファイルを使用してカスタム・コントロールのサポートを拡張」を参照してください。

テスト関数に応じてカスタム・サーバ実行時コンテキストを選択

各カスタム・サーバは，次のテスト関数をコントロールごとに実装できます。

- Test Record
- Test Run
- Table Verification（チェックポイントおよび出力値をサポート）
- 上記のテスト関数の組み合わせ

次に，実行時コンテキストを説明します。

➤ **Application under test**：テスト中のアプリケーションのコンテキスト。

➤ **UFT**：UFT コンテキスト。

次の表では，各実行時コンテキストについて，実装可能なテスト関数を判断する際のガイドラインを示しています。

タスク	Test Record	Test Run	Table Verification	実行時コンテキスト	説明
キーワード駆動のテストを使用してタスクを作成（アプリケーションのステップを記録する方法を使用しない）	非対応	対応	.NET DLL カスタム・サーバのみ	**Application under test** または **UFT**	Test Record テスト関数は，テスト対象アプリケーションで実行されるアクションと，実行結果の動作を記録します。記録された内容がテストに変換されます。アプリケーションのステップを記録する方法ではなく，キーワード駆動のテストを使用して GUI テストを作成する場合，Test Record 関数を実装する必要はありません。
Application under test コンテキストでカスタム・サーバを実装	任意	任意（通常）	.NET DLL カスタム・サーバのみ	**Application under test**	Test Run 関数は, テストを実行して結果を追跡することにより，アプリケーションが必要に応じて適切に実行されるかどうかをテストします。Test Run は，ほとんどの場合 **Application under test** コンテキストで実装されます。

タスク	Test Record	Test Run	Table Verification	実行時コンテキスト	説明
ウィンドウ・メッセージのリッスン	対応	補助クラスを使用する場合のみ	.NET DLL カスタム・サーバのみ	**UFT**	.NET DLL カスタム・サーバで，ウィンドウ・メッセージをリッスンし，さらにコントロールのイベントおよびプロパティにもアクセスする場合，補助クラスを使用します。**UFT** コンテキストで稼働するカスタム・サーバは，**Application under test** コンテキストで実行する補助クラス・オブジェクトを使用することで，**Application under test** コンテキスト内のイベントをリッスンできます。このオブジェクトは，コントロール・プロパティへの直接アクセスにも使用できます。
テーブル・チェックポイントおよび出力値をカスタム・グリッド・コントロールに実装	任意	任意	.NET DLL カスタム・サーバのみ	**Application under test** または **UFT**	テーブル・チェックポイントと出力値のサポートは，.NET DLL を実行するコンテキストにかかわらず，カスタム・グリッド・コントロールに実装できます。
アプリケーションが使用するサービスは，カスタム・コントロールのサービスよりも UFT サービスの方が多い	対応。ただし効率が低下する可能性がある	効率化できる可能性がある	効率化できる可能性がある	**UFT** の方を推奨	Test Run セッションでウィンドウ・メッセージをリッスンする必要がないので，**UFT** コンテキストは必須ではありません。ただし，アプリケーションが使用するサービスはカスタム・コントロールよりも **UFT** のサービスの方が多いので，**UFT** コンテキストで Test Run を実装する方が効率化できることがあります。

カスタム・コントロールの分析とテスト・オブジェクトへのマッピング

.NET Add-in Extensibility の開発では，.NET Windows Forms コントロールを既存の UFT .NET Add-in テスト・オブジェクト・クラスにマッピングし，さらに開発するカスタム・サーバにマッピングします。

最初のマッピングでは，UFT でカスタム・コントロールを表すのに使用するテスト・オブジェクト・クラスが指定されます。2 番目のマッピングでは，使用する カスタム・サーバが指定されます。カスタム・サーバは，必要なコントロール機能を提供するために，コントロールで使用するテスト・オブジェクトの機能を拡張します。

UFT が .NET コントロールを標準で認識し，それを表すのに SwfObject ではなく .NET アドイン・テスト・オブジェクト（SwfEdit や SwfList など）を使用する場合，このコントロールはこれ以外のテスト・オブジェクト・タイプにマッピングできません。ただし，カスタム・サーバにマッピングしてテスト・オブジェクトの機能を拡張することは可能です。

カスタム・コントロールをテスト・オブジェクトにマッピング

カスタム・コントロールをテスト・オブジェクトにマッピングするには, UFT .NET Add-in Extensibility の System Windows Forms 設定ファイル（**SwfConfig.xml**）で **MappedTo** 属性を指定します。各カスタム・コントロールのマッピング先には，コントロールのサポートに必要な動作と類似した動作を含む UFT テスト・オブジェクト・クラスを選択します。

マッピングを指定しない場合，UFT はカスタム・コントロールを標準の汎用テスト・オブジェクトである **SwfObject** にマッピングします。**SwfConfig.xml** の詳細については，第 5 章「UFT Windows Forms Extensibility の設定方法」を参照してください。

マッピングを行うだけで，プログラミングが不要になることがあります。既存の UFT テスト・オブジェクトがコントロールを十分にカバーしている場合，コントロールを UFT テスト・オブジェクトにマッピングするだけで，十分です。

カスタム・コントロールをカスタム・サーバにマッピング

機能的に類似している UFT テスト・オブジェクトにコントロールをマッピングすると，カスタム・サーバでは，カスタム・コントロールを変更せずに適用するテスト・オブジェクト・メソッドをオーバーライドする必要はなくなります。たとえば，ほとんどのコントロールには **Click** メソッドが含まれています。既存のテスト・オブジェクトの **Click** メソッドがカスタム・コントロールの **Click** メソッドを適切に実装していれば，既存のオブジェクト・メソッドをオーバーライドする必要はありません。

既存のオブジェクトにはないカスタム・オブジェクトの Test Run 機能をカバーするには，カスタム・サーバに新しいメソッドを追加します。カバーする機能のメソッドの名前が同じで実装が異なる場合には，既存のオブジェクトのメソッドをオーバーライドしてください。

UFT テスト・オブジェクトが Test Record を適切にカバーしている状態で，Test Run をカスタマイズする必要がある場合，Test Record は実装しないでください。Test Record を実装すると，既存のオブジェクトの実装が置換されてしまい，必要な Test Record 機能をすべて実装する必要があります。

UFT では，カスタム・コントロールのサポートのためにカスタマイズしたテスト・オブジェクトでステップを編集すると，ステートメントの自動補完機能により，UFT の定義に加えて，テスト・オブジェクトで定義したカスタム・プロパティとメソッドが表示されます。UFT は，テスト・オブジェクト設定ファイルを使用して，カスタム・テスト・オブジェクトのメソッドとプロパティを一覧表示します。

.NET Add-in Extensibility 計画チェックリスト

特定のタイプのコントロールのサポートを計画する際には，いくつかの質問に対する答えを考える必要があります。ここでは，チェックリストについて説明します。簡略化された，印刷形式のチェックリスト（44 ページ）が利用できます。

1. UFT がインストールされた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使用できることを確認します。
2. カスタム・コントロールを表す .NET Windows Forms テスト・オブジェクト・クラスを選択します（標準設定は **SwfObject** です）。

3 選択したテスト・オブジェクト・クラスのカスタマイズが必要か？

a テスト・オブジェクト定義に追加するテスト・オブジェクト・メソッドがあれば，指定します。メソッドの引数タイプと名前，およびメソッドがリターン・コードに加えて値を返すかどうかを指定します。

この内容は，.NET Add-in Extensibility サポート・セットの設計時にテスト・オブジェクト設定ファイルで指定します。

b 動作の変更またはオーバーライドが必要なテスト・オブジェクト・メソッドがあれば，指定します。

.NET Add-in Extensibility カスタム・サーバの設計時に，新しいテスト・オブジェクト・メソッドや，動作のオーバーラードが必要なテスト・オブジェクトの実装が必要になります。

4 このクラスのテスト・オブジェクトを .NET Windows Forms スパイに表示するか？（標準設定では表示）。

5 記録のサポートを提供するか ？提供する場合，記録をトリガするイベントのリストを指定します。

6 テーブル・コントロールのサポートを作成する場合，このコントロールでのテーブル・チェックポイントと出力値のサポートを提供するかどうかを検討してください。

.NET Add-in Extensibility の計画チェックリスト

このチェックリストを使用して，カスタム・コントロールのサポートの計画を立てます。

☑	カスタム・コントロール・サポートの計画チェックリスト	テスト・オブジェクト設定ファイルで指定するか？	.NET Add-in Extensibility 設定ファイルで指定するか？	カスタム・サーバで指定するか？
❑	このカスタム・コントロールのソースの場所：			
❑	コントロールにマッピングする .NET テスト・オブジェクト・クラス（標準設定は **SwfObject**）			

☑	カスタム・コントロール・サポートの計画チェックリスト	テスト・オブジェクト設定ファイルで指定するか？	.NET Add-in Extensibility 設定ファイルで指定するか？	カスタム・サーバで指定するか？
❑	追加または変更が必要なテスト・オブジェクト（必要に応じて，引数と戻り値を指定）：			
❑	このクラスのテスト・オブジェクトを .NET Windows Forms スパイで表示するか？ **表示する**（標準設定）**/表示しない**			
❑	記録のサポートを提供するか？ **する/しない** その場合，記録をトリガするイベントを列挙：			
❑	テーブル・チェックポイントおよび出力値のサポートを提供するか？ **する/しない**			

その他の情報

カスタム・コントロール・サポートの計画作業が完了したら，.NET Add-in Extensibility サポート・セットを作成します。第 4 章「サポート・セットの開発」では，.NET Add-in Extensibility サポート・セットの開発方法について説明します。

第 4 章

サポート・セットの開発

本章では，カスタム .NET Windows Forms コントロールの拡張サポートを開発する方法について説明します。具体的には，.NET Add-in Extensibility サポート・セット用に作成する必要があるファイル，ファイルの構造と内容，ユーザ環境に合わせた各種 UFT 機能のサポートに必要なファイルの作成方法について説明します。

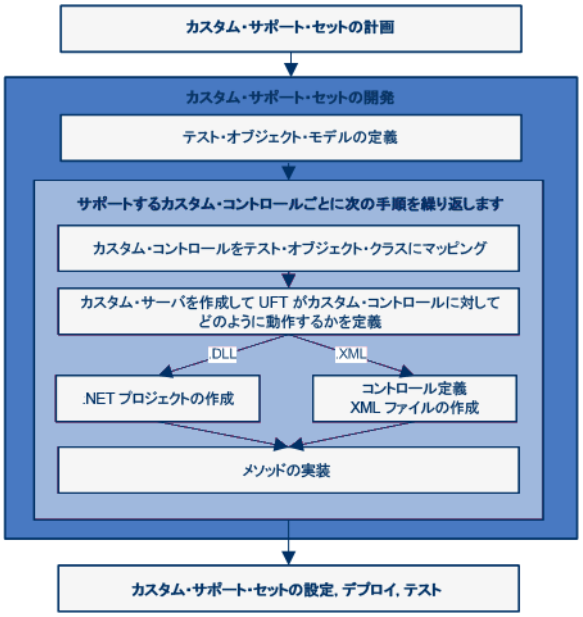
注 : サポート・セットの作成作業では，綿密な計画が必要です。詳細については，35 ページ「サポート・セットの計画」を参照してください。

設計したサポートをアクティブ化するには, 所定の場所に格納する必要があります。.NET Add-in Extensibility サポート・セット・ファイルの格納場所については，第 5 章「サポート・セットの設定とデプロイ」を参照してください。

本章の内容

開発ワークフロー

.NET Add-in Extensibility サポート・セットの実装は，次の工程に従って行われます。次の項では，サポート・セットの開発ワークフローについて説明します。



テスト・オブジェクト・モデルの定義

カスタム・コントロールのサポートを開発する最初の工程では，アプリケーションとコントロールのテストで UFT が使用するテスト・オブジェクト・モデルを導入します。テスト・オブジェクト・モデルとは，ユーザ環境でカスタム・コントロールを表すテスト・オブジェクト・クラスと，カスタム・コントロールをサポートするテスト・オブジェクト・メソッドの構文を含むリストです。

テスト・オブジェクト・モデルは，特定の XML スキーマに基づいてテスト・オブジェクト設定ファイルで定義します。テスト・オブジェクト設定ファイルの作成方法の詳細については，49 ページ「テスト・オブジェクト設定ファイルの作成」を参照してください。

テスト・オブジェクト・モデルを定義する利点

テスト・オブジェクト設定ファイルの実装は，必須ではありません。テスト・オブジェクト設定ファイルを実装しなくても .NET カスタム・サーバ DLL またはコントロール定義ファイルで定義されているテスト・オブジェクト・メソッドは想定どおりの動作をしますが，次に示す機能は提供されません。

テスト・オブジェクト設定ファイルでカスタム・テスト・オブジェクトを定義することにより，UFT で GUI テストを編集する作業において次の機能が利用可能になります。

- キーワード・ビューの［**操作**］カラムと，ステートメントの自動補完機能を有効にしたエディタで，使用可能なカスタム・テスト・オブジェクト・メソッドが一覧表示されます。
- このクラスのテスト・オブジェクトのステップを生成する際，キーワード・ビューとステップ・ジェネレータでテスト・オブジェクト・メソッドが 1 つ標準で選択されます。
- キーワード・ビューの［**注釈**］カラムに，カスタム・テスト・オブジェクト・メソッドの注釈が表示されます。
- アイコンとコンテキスト・ヘルプが表示されます（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・メソッドを追加した場合のみ）。

テスト・オブジェクト設定ファイルの作成

テスト・オブジェクト設定ファイルを作成する方法について説明します。

テスト・オブジェクト設定ファイルを作成するには，次の手順で行います。

1. **<UFT インストール・フォルダ>\dat\Extensibility\DotNet\DotNetCustomServerMethods.xml** ファイルをコピーして，同じフォルダに新しいテスト・オブジェクト設定ファイルを作成します（元のファイルは変更しないでください）。
2. 新しく作成したテスト・オブジェクト設定ファイルを編集し，動作を変更したいテスト・オブジェクト・クラスがあれば適宜変更します。変更が不要なテスト・オブジェクト・クラスは削除します。
3. テスト・オブジェクト設定ファイルを保存してから閉じます。

詳細については，次を参照してください。

➤ 50 ページ「テスト・オブジェクト設定ファイルの内容」

➤ 52 ページ「既存のテスト・オブジェクト・クラスの変更」

➤ 53 ページ「テスト・オブジェクト設定ファイルの定義と カスタム・サーバ の定義が一致することを確認」

➤ 53 ページ「複数のテスト・オブジェクト設定ファイルの実装」

➤ 56 ページ「テスト・オブジェクト設定ファイルの例」

➤ UFT Test Object Schema ヘルプ

テスト・オブジェクト設定ファイルの内容

テスト・オブジェクト設定ファイルでは，次の内容が記述されています。

➤ テスト・オブジェクト・クラスの名前と属性。

➤ テスト・オブジェクト・クラス定義に関連するカスタム・コントロールの名前。

➤ テスト・オブジェクト・クラスのメソッド。各メソッドには次の内容が含まれます。

  ➤ 引数（引数タイプと方向）。

  ➤ 引数が必須かどうか。必須ではない場合，標準設定値。

  ➤ 説明（キーワード・ビュー，エディタ，ステップ・ジェネレータでツールヒントとして表示）。

  ➤ 注釈の文字列（キーワード・ビューの［**注釈**］カラムと，ステップ・ジェネレータに表示）。

- キーワード・ビューやエディタでテスト・オブジェクト・メソッドを開いた状態で F1 キーを押したとき，またはステップ・ジェネレータでメソッドを開いた状態で［**操作ヘルプ**］ボタンをクリックしたときに開くコンテキスト・ヘルプ・トピック。ヘルプ・ファイルのパスとファイル内の該当するヘルプ ID を定義します（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・メソッドを追加した場合のみ）。
- 戻り値のタイプ。

- キーワード・ビューとステップ・ジェネレータでこのクラスのテスト・オブジェクトのステップを生成する際に，標準で選択されるテスト・オブジェクト・メソッド。

次の例は，テスト・オブジェクト設定ファイルで定義されている **SwfObject** テスト・オブジェクト・クラスの一部を示しています。**MyCustomButtonSet** メソッドを追加することにより，**SwfObject** が拡張されています。このメソッドには引数が 1 つあり（**Percent**，コントロールで設定されるパーセンテージ），キーワード・ビューで表示される注釈文字列もあります。

```
</TypeInformation>
...
   <ClassInfo BaseClassInfoName="SwfObject" Name="MyCompany.MyButton">
...
      <TypeInfo>
         <Operation Name="MyCustomButtonSet" PropertyType="Method"
               ExposureLevel="CommonUsed">
                <Description>Set the percentage in the task bar</Description>
               <Documentation><![CDATA[Set the %I %t to <Percent> percent.]]>
                </Documentation>
                 <Argument Name="Percent" IsMandatory="true" Direction="In">
                    <Type VariantType="Integer"/>
                    <Description>The percentage to set in the task bar.</Description>
                 </Argument>
         </Operation>
      </TypeInfo>
   </ClassInfo>
</TypeInformation>
```

テスト・オブジェクト設定ファイルの構造と構文については，UFT Test Object Schema ヘルプ（UFT .NET Add-in Extensibility オンライン・ヘルプで利用可能）を参照してください。

既存のテスト・オブジェクト・クラスの変更

コントロールを部分的にサポートできるテスト・オブジェクト・クラスが存在しても，テスト・オブジェクト・メソッドの追加や変更といった部分的な変更が必要になる場合があります。

このような場合には，追加のテスト・オブジェクト・メソッドを定義して実装することで，テスト・オブジェクトの機能を拡張できます。さらに，既存のテスト・オブジェクト・メソッドをオーバーライドする別の実装を用意することもできます。テスト・オブジェクト設定ファイルでメソッドを追加または変更し，カスタム・サーバ で実装を設計します。

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

テスト・オブジェクト設定ファイルでテスト・オブジェクト・クラスを定義する作業では，この定義に関連するカスタム・コントロールを指定します（**ClassInfo** 要素の，**BaseClassInfoName** 属性でテスト・オブジェクト・クラスを指定し，**Name** 属性でカスタム・クラスの名前を指定します)。

次に，このテスト・オブジェクト・クラスの定義にカスタム・テスト・オブジェクト・メソッドを追加します。これにより，このメソッドは UFT で表示され，指定したタイプのカスタム・コントロールを表すテスト・オブジェクトのみで使用可能になります。

たとえば，**MyCompany.MyButton** コントロールでの使用時に **Set** メソッドを SwfEditor テスト・オブジェクト・クラスに追加した場合，このコントロールを表すオブジェクトに対してのみ，このメソッドが UFT でテスト・オブジェクト・メソッドのステートメント自動補完リストに表示されます。ただし，SwfEditor テスト・オブジェクトをほかのタイプのコントロールに使用する場合には，このメソッドは使用できません。

テスト・オブジェクト設定ファイルの定義と カスタム・サーバ の定義が一致することを確認

テスト・オブジェクト設定ファイル内の定義が，.NET カスタム・サーバ DLL またはコントロール定義ファイルの定義内容と完全に一致することを確認します。たとえば，両方の定義で，テスト・オブジェクト・メソッドの名前が完全に一致していなければなりません。名前が異なると，メソッドは表示されますが（ステートメントの自動補完機能を使用する場合など），実際には正常に動作せず実行セッションは失敗します。さらに，テスト・オブジェクト設定ファイルで指定したカスタム・コントロールの名前は，.NET Add-in Extensibility 設定ファイルで指定した名前と一致していなければなりません。

複数のテスト・オブジェクト設定ファイルの実装

テスト・オブジェクト設定ファイルは，1 つまたは複数のファイルを実装できます（不要ならば実装しなくても問題ありません）。たとえば，1 つのテスト・オブジェクト設定ファイルで 1 つのテスト・オブジェクト・クラスのカスタム・メソッドを定義し，別のテスト・オブジェクト設定ファイルで別のテスト・オブジェクトのカスタム・メソッドを定義できます。また，1 つのテスト・オブジェクト設定ファイルでテスト・オブジェクト・クラスのカスタム・メソッドを複数定義し，これと同じテスト・オブジェクト・クラスについて，別のテスト・オブジェクト設定ファイルでカスタム・メソッドを複数定義することもできます。

UFT は，起動のたびにテスト・オブジェクト設定ファイルをすべて読み込み，各テスト・オブジェクト・クラス情報を 1 つのテスト・オブジェクト・クラス定義にマージします。このため，複数のカスタム・ツールキットのサポートに同じテスト・オブジェクト・クラス定義を使用できます。

UFT が設定ファイルをマージする方法

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)。

テスト・オブジェクト設定ファイルの例

次に，**ToolStrip** テスト・オブジェクトの定義を例として示します。

```
<ClassInfo Name="System.Windows.Forms.ToolStrip"
            BaseClassInfoName="SwfToolBar" FilterLevel="1">
  <TypeInfo>
    <Operation Name="Select" PropertyType="Method"
                ExposureLevel="CommonUsed" SortLevel="-1">
      <Description>Selects a menu item from a SwfToolBar dropdown menu.
      </Description>
      <Argument Name="Item" Direction="In" IsMandatory="true">
        <Type VariantType="VT_BSTR"/>
      </Argument>
    </Operation>
    <Operation Name="IsItemEnabled" PropertyType="Method"
                ExposureLevel="Expert" SortLevel="-1">
      <Description>Indicates whether the toolbar item is enabled.</Description>
      <Argument Name="Item" Direction="In" IsMandatory="true">
        <Type VariantType="VT_BSTR"/>
      </Argument>
      <ReturnValueType><Type VariantType="VT_BOOL"/></ReturnValueType>
    </Operation>
    <Operation Name="ItemExists" PropertyType="Method"
                ExposureLevel="Expert" SortLevel="-1">
      <Description>Indicates whether the specified toolbar item exists.</Description>
      <Argument Name="Item" Direction="In" IsMandatory="true">
        <Type VariantType="VT_BSTR"/>
      </Argument>
      <ReturnValueType> <Type VariantType="VT_BOOL"/></ReturnValueType>
    </Operation>
  </TypeInfo>
</ClassInfo>
```

この例では，**ToolStrip** テスト・オブジェクト・クラスによって **SwfToolBar** テスト・オブジェクト・クラスが拡張されています。**ToolStrip** テスト・オブジェクト・クラスで指定されている標準テスト・オブジェクト・メソッドは **Select** です（入力パラメータは **Item** であり，必須です）。

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

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する操作は，**<UFT インストール・パス>\dat** フォルダにある .NET Add-in Extensibility 設定ファイル（**SwfConfig.xml**）で定義します。この XML ファイルでは，各カスタム・コントロールを表すテスト・オブジェクト・クラス，各コントロールとの相互作用に必要な情報の格納場所が記述されています。マッピングの詳細については，90 ページ「カスタム・サーバの使用に必要な UFT の設定」を参照してください。

UFT がカスタム・コントロールと対話する方法を定義

UFT がカスタム・コントロールを認識できるようにしたら，テスト・オブジェクト・メソッドを実行するためのサポートを提供する必要があります。カスタム・テスト・オブジェクト・メソッドを実装しない状態で，このメソッドを実行するステップを含むテストを実行しようとすると，テストは失敗して実行時エラーが発生します。

カスタム・サーバ には，UFT とカスタム・コントロールが対話する方法を示す実装が含まれています。カスタム・サーバ には，.DLL ファイルまたは .XML ファイル（必要に応じて UFT が .DLL ファイルに変換します）を使用できます。各メソッドが適する用途については，35 ページ「サポート・セットの計画」を参照してください。

- ほとんどの実装では DLL ファイルが使用されます。詳細については，58 ページ「.NET DLL を使用してカスタム・コントロールのサポートを拡張」を参照してください。
- 実装方法としてより簡単なのは XML ファイルを使用する方法であり，カスタム・コントロールごとにコントロール定義ファイルを作成します。詳細については，83 ページ「XML ファイルを使用してカスタム・コントロールのサポートを拡張」を参照してください。

カスタム・サーバ を作成したら，カスタム・サーバを使用する設定を UFT で行います。詳細については，90 ページ「カスタム・サーバの使用に必要な UFT の設定」を参照してください。

.NET DLL を使用してカスタム・コントロールのサポートを拡張

.NET Windows Forms コントロールをサポートするには，.NET DLL として実装された カスタム・サーバ を作成します。次のインタフェースを組み合わせて実装する .NET DLL クラス・ライブラリとして，.NET プロジェクトを Microsoft Visual Studio でセットアップします。

- Test Record（IRecord インタフェース）
- Test Run（Replay インタフェース）
- Table Verification（チェックポイントおよび出力値をサポート）

注：IRecord インタフェースは UFT .NET Add-in Extensibility SDK で提供されています。UFT カスタム・サーバ 設定ウィザードで .NET DLL カスタム・サーバ を作成する手順では，最初に IRecord インタフェースを実装するコードが提供されます。

Replay と Table Verification の各インタフェースは SDK で提供されていないので，それぞれ実装する必要があります。

詳細については，67 ページ「IRecord インタフェースの実装」と『UFT .NET Add-in Extensibility API Reference』（UFT .NET Add-in Extensibility オンライン・ヘルプで提供）を参照してください。

.NET DLL カスタム・サーバ を作成する作業では，.NET アセンブリのプログラム方法に関する知識が必要です。この項で掲載する図と手順は，Microsoft Visual Studio 2008 を開発環境として使用し，カスタム・サーバ プロジェクト・テンプレートがインストールされていることを前提としています。詳細については，27 ページ「HP UFT .NET Add-in Extensibility SDK のインストール」を参照してください。

カスタム・サーバ DLL を使用する際の考慮事項

- 設計した カスタム・サーバ DLL は 32 ビット UFT アプリケーションと，テスト対象アプリケーションにロードされます。したがって，64 ビット・アプリケーションでサポートを使用可能にするには，**Platform target** オプションを **Any CPU** に設定し，Custom Server DLL をビルドする必要があります。
- .NET Framework version 1.1 で実行しているアプリケーションの場合, Visual Studio 2005 以降を使用して作成した DLL は使用できません。したがって，テスト対象アプリケーションを .NET Framework version 1.1 で実行する場合，Visual Studio 2005 以降を使用して .NET DLL として実装した カスタム・サーバ は使用できません。
- テストを開くと，UFT によって カスタム・サーバ がロードされます。したがって，カスタム・サーバ を .NET DLL として実装する場合，カスタム・サーバ のロード後に行った DLL の変更は，次回テストを開いたときに有効になります。

カスタム・サーバ DLL の設計

カスタム・サーバ を .NET DLL として実装する作業では，次のタスクを行います。

- .NET プロジェクトのセットアップ（59ページを参照）
- .NET DLL を使用してカスタム・コントロール用に Test Record を実装（66ページを参照）
- .NET DLL を使用してカスタム・コントロール用に Test Run を実装（72ページを参照）
- .NET DLL カスタム・サーバ でテーブル・チェックポイントおよび出力値のサポートを実装（73ページを参照）
- Application Under Test のコードを UFT コンテキストから実行（79ページを参照）

.NET プロジェクトのセットアップ

カスタム・サーバ C# プロジェクト・テンプレートまたは カスタム・サーバ Visual Basic プロジェクト・テンプレートを使用して，Microsoft Visual Studio で .NET プロジェクトをセットアップします（このテンプレートは，インストール時に自動的にインストールされます。詳細については，第 2 章「HP UFT .NET Add-in Extensibility SDK のインストール」を参照してください)。

.NET プロジェクトのセットアップ作業では，カスタム・サーバ プロジェクト・テンプレートは次のことを行います。

- .DLL ファイルのビルドに必要なプロジェクト・ファイルを作成します。
- 選択したテンプレートに応じて C# ファイルまたは Visual Basic ファイルをセットアップします。このファイルには，Test Record または Test Run の実装時にオーバーライドできるメソッドの定義を含むコードがコメントとして記述されています。
- Test Record と Test Run の実装テクニックのデモとなるサンプル・コードを提供します。
- カスタム・サーバ を定義する XML ファイルのセグメントを作成します。この内容は .NET Add-in Extensibility 設定ファイル（**SwfConfig.xml**）にコピーできます。

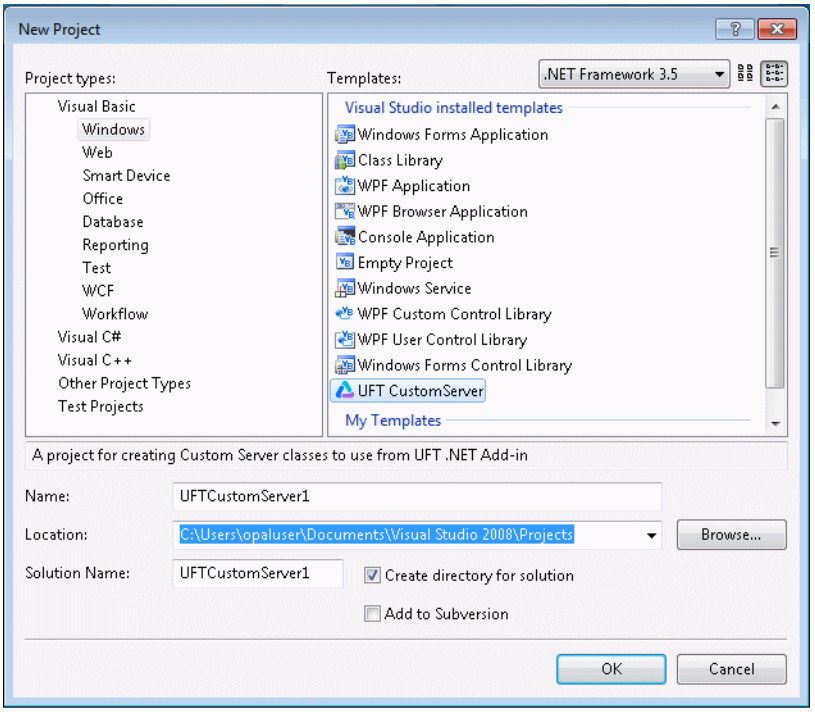
新しい .NET プロジェクトをセットアップするには，次の手順を行います。

注意 : カスタム・サーバ・プロジェクト・テンプレートをもとに .NET プロジェクトを作成するには，次のフォルダとすべてのサブフォルダに対して管理者権限または読み取りおよび書き込み権限が必要です。
<Microsoft Visual Studio インストール・フォルダ>\VC#\VC#Wizards

---

1. Microsoft Visual Studio を起動します。
2. ［**ファイル**］＞［**新規作成**］＞［**プロジェクト**］を選択するか，CTRL + SHIFT + N キーを押すと，［プロジェクトの新規作成］ダイアログ・ボックスが開きます。

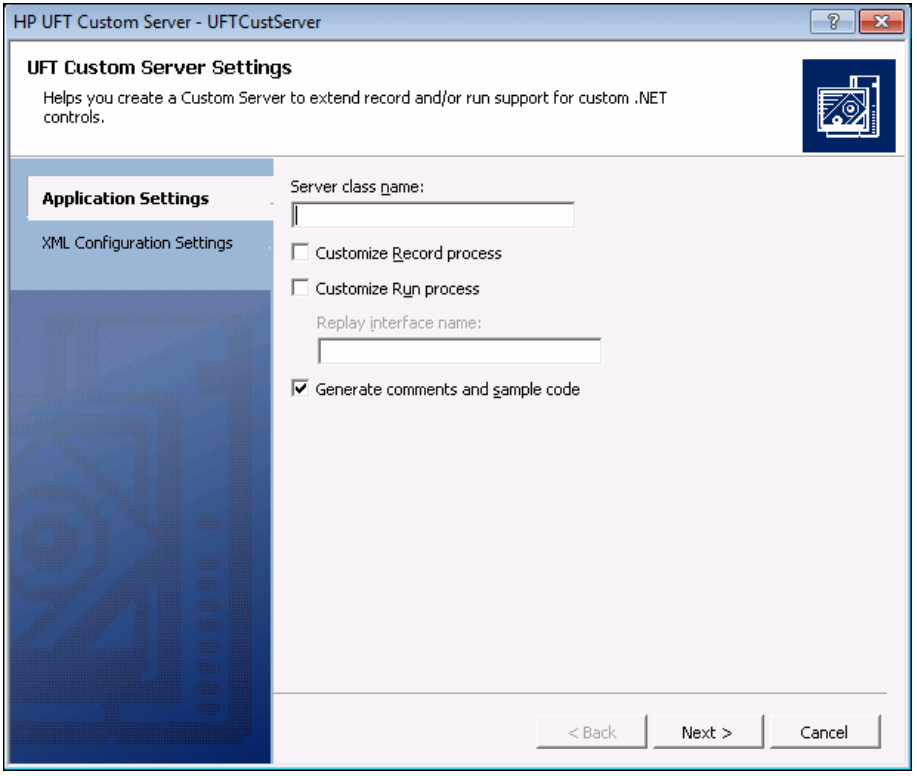
3 ［**プロジェクト タイプ**］ツリーで［**Visual C#**］＞［**Windows**］ノードまたは［**Visual Basic**］＞［**Windows**］ノードを選択します。



注：2008 以外のバージョンの Microsoft Visual Studio では，ダイアログ・ボックスの表示や，ツリー構造内で **UFT CustomServer** テンプレートが表示されるノードの場所が若干異なる場合があります。

---

4 ［テンプレート］表示枠で［**UFT CustomServer**］テンプレートを選択します。新しいプロジェクトの名前と，プロジェクトの保存先を入力します。［**OK**］をクリックします。UFT カスタム・サーバ 設定ウィザードが起動します。



5 カスタム・サーバ で拡張するサポート（Test Record サポートのみ，Test Run サポートのみ，または両方）を決定し，ウィザードの［Application Settings］ページで選択します。

➤［**Server class name**］ボックスには，カスタム・サーバ クラスの名前をわかりやすい形式で入力します。

➤ UFT で Test Record プロセスを実装する場合は，［**Customize Record process**］チェックボックスを選択します。

［**Customize Record process**］チェックボックスを選択すると，記録ステップの実装に使用するコードのフレームワークが作成されます。

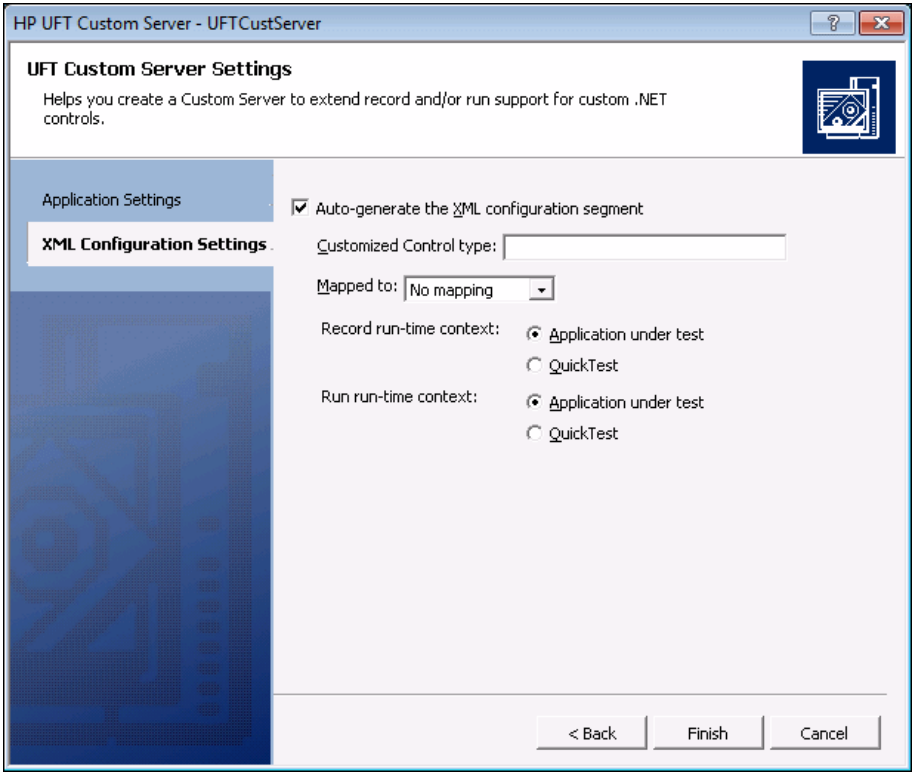
UFT でテストを手動で作成する場合，またはこのコントロールのマッピング先となる既存のテスト・オブジェクトの Test Record 関数を使用する場合には，このチェックボックスは選択しないでください。カスタム・コントロールのマッピング先となる既存のテスト・オブジェクトから，Test Record の実装が継承されることはなく，既存のオブジェクトの Test Record 実装を完全に置換します。したがって，既存のオブジェクトの機能の中で必要なものがある場合は，明示的なコーディングが必要になります。

➤ Test Run 関数をカスタム・コントロール用に実装する場合（つまり，既存のテスト・オブジェクトのメソッドをオーバーライドするか，新しいメソッドでテスト・オブジェクトを拡張する場合），［**Customize Run process** ］チェックボックスを選択します。作成する再生インタフェースの名前を，［**Replay interface name**］ボックスに入力します。

［**Customize Run process**］チェックボックスを選択すると，Test Run サポートの実装で使用するコードのフレームワークが作成されます。

➤ ウィザードで生成されるコードにコメントとサンプルを追加する場合は，［**Generate comments and sample code**］チェックボックスを選択します。

6 ［**Next**］をクリックします。ウィザードの［XML Configuration Settings］ページが開きます。



7 ウィザードの［XML Configuration Settings］ページでは，.NET Add-in Extensibility 設定ファイル（**SwfConfig.xml**）にコピーできる XML コードのセグメントを生成できます。このファイルはカスタム・コントロールをテスト・オブジェクトにマッピングし，テスト・オブジェクトの カスタム・サーバ の場所を UFT に提示します（XML 設定セグメントを自動生成しない場合は，後で .NET Add-in Extensibility 設定ファイルを手動で編集できます）。**SwfConfig.xml** ファイルにこのセグメントをコピーする手順については，93 ページ「UFT カスタム・サーバ設定ウィザードで生成された設定情報のコピー」を参照してください。

➤ ［**Auto-generate the XML configuration segment**］チェックボックスを選択すると，ウィザードによって設定セグメントが作成され，**Configuration.xml** ファイルに保存されます。

➤ ［**Customized Control type**］ボックスに，作成中の カスタム・サーバ で使用するコントロールの完全なタイプ名（ラッピングしているすべての名前空間を含む。たとえば，System.Windows.Forms.CustCheckBox など）を入力します。

注：複数のアセンブリに含まれるコントロール・タイプを指定するには，アセンブリの名前など，タイプを完全に示す情報を指定してください。たとえば，次のような値を入力できます。
- System.Windows.Forms.CustCheckBox, System2.Windows.Forms.v8.5
- System.Windows.Forms.CustCheckBox, System2.Windows.Forms.v8.5, Version=8.5.20072.1093, Culture=neutral, PublicKeyToken=8aa4d5436b5ad4cd

このような方法で指定しておくと，アプリケーション内にバージョンの異なるコントロールが存在する場合に便利です。

➤ ［**Mapped to**］ボックスでは，カスタム・サーバ のマッピング先となるテスト・オブジェクトを選択します。［**No mapping**］を選択すると，カスタム・サーバ は **SwfObject** テスト・オブジェクトに自動的にマッピングされます。

詳細については，22 ページ「カスタム・コントロールをテスト・オブジェクトにマッピング」を参照してください。

➤ Test Record または Test Run の実行時コンテキストを選択します。実行時コンテキストとは，テスト対象アプリケーション（**Application under test**）のコンテキスト，または UFT（**QuickTest**）のコンテキストです。

詳細については，20 ページ「カスタム・サーバの作成」を参照してください。

8 ［**Finish**］をクリックします。ウィザードが終了し，新しいプロジェクトが開いてコーディング可能な状態になります。

ウィザードで［**Finish**］をクリックすると，**Configuration.xml** セグメント・ファイルが作成されてプロジェクトに追加されます。設定セグメントのファイルを必要に応じて更新または変更します。セグメント・ファイルの使用方法の詳細については，93 ページ「UFT カスタム・サーバ設定ウィザードで生成された設定情報のコピー」を参照してください。

.NET DLL を使用してカスタム・コントロール用に Test Record を実装

コントロールに関するビジネス・コンポーネントまたはテストを記録する作業では，コントロールのアクティビティをリッスンし，アクティビティをテスト・オブジェクトのメソッド呼び出しに変換して，メソッド呼び出しをテストに書き込みます。コントロールのアクティビティをリッスンする処理には，コントロール・イベントのリッスン，ウィンドウ・メッセージのフック，またはその両方を併用します。

注 : アプリケーションのステップを記録する方法ではなく，キーワード駆動のテストを使用して GUI テストを作成する場合，Test Record を実装する必要はありません。

Test Record のコードを記述する作業には，**IRecord** インタフェース（UFT .NET Add-in Extensibility SDK で提供）に基づいて，ウィザードで生成されたコード・セグメントにメソッドを実装します。カスタム・コントロールのマッピング先となる既存のテスト・オブジェクトから，Test Record の実装が継承されることはなく，既存のオブジェクトの Test Record 実装を完全に置換します。したがって，既存のオブジェクトの機能の中で必要なものがある場合は，明示的なコーディングが必要になります。

この項の作業を進める前に，20 ページ「カスタム・サーバの作成」の内容をよく読んでおいてください。

本項の内容

➤ IRecord インタフェースの実装

➤ テストへのテスト・オブジェクト・メソッドの記述

この項で説明するインタフェース，クラス，列挙，メソッドの詳細については，『UFT .NET Add-in Extensibility API Reference』（UFT .NET Add-in Extensibility オンライン・ヘルプで提供）を参照してください。

IRecord インタフェースの実装

IRecord インタフェースを実装するには，次で説明するコールバック・メソッドをオーバーライドし，イベント・ハンドラまたはメッセージ・ハンドラに実装の詳細情報を追加します。

次に挙げる例では，コールバック・メソッドはそれぞれ C# で書かれています。

InitEventListener コールバック・メソッド

カスタム・サーバ のロード時に，**CustomServerBase.InitEventListener** が UFT によって呼び出されます。このメソッドを使用するイベント・ハンドラとメッセージ・ハンドラを追加します。

イベント・ハンドラとメッセージ・ハンドラを追加するには，次の手順で行います。

1 コントロールのイベントのハンドラを実装します。

一般的に，ハンドラはイベントをキャプチャして，メソッドをテストに書き込みます。次に，簡単なイベント・ハンドラの例を示します。

```
public void OnMouseDown(object sender, MouseEventArgs e)
{
    // If a button other than the left was clicked, do nothing.
    if(e.Button != System.Windows.Forms.MouseButtons.Left)
    return;
    /*
    For more complex events, here you would get any
    other information you need from the control.
    */
    // Write the test object method to the test
    RecordFunction("MouseDown",
    RecordingMode.RECORD_SEND_LINE,
    e.X,e.Y);
}
```

詳細については，71 ページ「テストへのテスト・オブジェクト・メソッドの記述」を参照してください。

2 イベント・ハンドラを **InitEventListener** に追加します。

```
public override void InitEventListener()
{
    .....
    // Adding OnMouseDown handler.
    Delegate  e = new MouseEventHandler(this.OnMouseDown);
    AddHandler("MouseDown", e);
    .....
}
```

Test Record 実装をテスト対象アプリケーションのコンテキストで実行する場合，次の構文を使用できます。

```
SourceControl.MouseDown += e;
```

この構文を使用する場合，**ReleaseEventListener** のハンドラをリリースする必要があります。

3 リモート・イベント・リスナを追加します。

カスタム・サーバ を UFT コンテキストで実行する場合，イベント処理にリモート・イベント・リスナを使用します。イベントを処理する **EventListenerBase** タイプのリモート・リスナを実装し，**InitEventListener** メソッドに **AddRemoteEventListener** の呼び出しを追加します。

```
public class EventsListenerAssist :EventsListenerBase
{
    // class implementation.
}
public override void InitEventListener()
{
    ...
    AddRemoteEventListener(typeof(EventsListenerAssist));
    ...
}
```

リモート・イベント・リスナの実装では，**EventListenerBase.InitEventListener** と **EventListenerBase.ReleaseEventListener** のオーバーライドと，**CustomServerBase** 内のこれらのコールバック関数のオーバーライドが必要です。この **EventListenerBase** コールバックの使用方法は，**CustomServerBase** コールバックと同じです。詳細については，『UFT .NET Add-in Extensibility API Reference』の **EventsListenerBase** クラスを参照してください。

UFT コンテキストからイベントを処理する場合，イベント引数をシリアル化する必要があります。詳細については，『UFT .NET Add-in Extensibility API Reference』の **CustomServerBase.AddHandler(String, Delegate, Type)** と **IEventArgsHelper** インタフェースを参照してください。

リモート・イベント・リスナをわかりやすい方法で実行するためにも，上記で示した **Application under test** コンテキストで実行してください。

OnMessage コールバック・メソッド

OnMessage は, UFT がフックしたウィンドウ・メッセージで呼び出されます。Test Record を UFT コンテキストで実行し，メッセージ処理が必要になる場合は，メッセージ処理をこのメソッドで実装します。

Test Record を **Application under test** コンテキストで実行する場合，この関数はオーバーライドしないでください。

詳細については,『UFT .NET Add-in Extensibility API Reference』の「**CustomServerBase.OnMessage**」を参照してください。

GetWndMessageFilter コールバック・メソッド

Test Record を UFT コンテキストで実行し，ウィンドウ・メッセージをリッスンする場合，このメソッドをオーバーライドすることにより，カスタム・サーバ が処理するメッセージを特定のカスタム・オブジェクトに限定するかどうか, 子オブジェクトからのメッセージも含めて処理するかどうかを UFT に通知します。

詳細については,『UFT .NET Add-in Extensibility API Reference』の「**IRecord.GetWndMessageFilter**」を参照してください。

その他の参照項目：87 ページ「トラブルシューティングと制限事項 - 設計したサポートの実行」

ReleaseEventListener コールバック・メソッド

UFT は，記録セッションの最後にこのメソッドを呼び出します。**ReleaseEventListener** で，カスタム・サーバ がリッスンするすべてのイベントから登録を解除します。たとえば，次の構文を使用して，**InitEventListener** の **OnClick** に登録する場合：

```
SourceControl.Click += new EventHandler(this.OnClick);
```

次のようにリリースが必要です。

```
public override void ReleaseEventListener()
{
    ....
    SourceControl.Click -= new EventHandler(this.OnClick);
    ....
}
```

ただし，**AddHandler** メソッドでイベントに登録している場合，UFT によって登録が自動解除されます。

テストへのテスト・オブジェクト・メソッドの記述

コントロールのアクティビティに関する情報は，イベント，ウィンドウ・メッセージ，この 2 つの組み合わせで提供されますが，この情報をアプリケーションごとに適切な方法で処理し，ステップをテスト・オブジェクトのメソッド呼び出しとして記述する必要があります。

テスト・ステップの記述では，**CustomServerBase** クラスまたは **EventsListenerBase** の **RecordFunction** メソッドを使用します。

ただし，次のアクティビティが発生するまで，アクティビティを適切に処理する方法がわからない場合もあります。したがって，ステップを保存しておき，**RecordFunction** を呼び出す過程でテストへの記述が必要かどうかを判断する仕組みが提供されています。詳細については，『UFT .NET Add-in Extensibility API Reference』の「**RecordingMode Enumeration**」を参照してください。

テスト・オブジェクトのメソッド呼び出しで指定する引数値を決定するためには，イベント引数またはウィンドウ・メッセージでは使用できないコントロールから情報を取得しなければならないことがあります。カスタム・サーバの Test Record 実装をテスト対象アプリケーションのコンテキストで実行する場合，**CustomServerBase** クラスの **SourceControl プロパティ**を使用して，コントロールのパブリック・メンバへの直接アクセスを取得します。コントロールがスレッド・セーフでない場合は，**ControlGetProperty** メソッドを使用してコントロールのステート情報を取得してください。

.NET DLL を使用してカスタム・コントロール用に Test Run を実装

Test Run で使用するテスト・オブジェクト・メソッドの定義では，メソッドをステップで実行する際にカスタム・コントロールで実行するアクションを指定します。一般的に，テスト・オブジェクト・メソッドの実装では次に示す操作を行います。

- カスタム・コントロールの属性値を設定する
- カスタム・コントロールのメソッドを呼び出す
- マウスとキーボードのシミュレーションを呼び出す
- ステップの結果を UFT に報告する
- エラーを UFT に報告する
- 別のライブラリを呼び出す（メッセージ・ボックスの表示，カスタム・ログの書き込みなど）

既存のテスト・オブジェクトの既存のメソッドをオーバーライドするか，新しいメソッドを追加することによって既存のテスト・オブジェクトを拡張する場合，カスタム Test Run メソッドを定義します。

記録されたテスト・オブジェクト・メソッドすべてが，既存のテスト・オブジェクトまたはこのカスタム・サーバによって，Test Run に実装されていることを確認します。

カスタム Test Run メソッドを定義するには, インタフェースを定義して **ReplayInterface** 属性を適用し，UFT で Test Run インタフェースとして識別されるように設定します。カスタム・サーバ で実装可能な Replay インタフェースは 1 つのみです。既存のオブジェクトの既存のメソッドと同じ名前でメソッドを定義する場合, このインタフェース・メソッドはテスト・オブジェクト実装をオーバーライドします。既存オブジェクトと異なる名前のメソッドは，新しいメソッドとして追加されます。

PrepareForReplay の呼び出しでテスト・オブジェクト・メソッドの実装を開始し，実行するアクティビティを指定して，**ReplayReportStep** または **ReplayThrowError** を呼び出して終了します。

詳細については,『UFT .NET Add-in Extensibility API Reference』（UFT .NET Add-in Extensibility オンライン・ヘルプで提供）を参照してください。

.NET DLL カスタム・サーバ でテーブル・チェックポイントおよび出力値のサポートを実装

テーブル・チェックポイントをテストに追加することにより，UFT ユーザはアプリケーションで表示されるテーブルの内容とプロパティをチェックできます。テーブル出力値ステップをテストに追加することにより，テーブルから値を取得し，保存してから，これを実行セッション内にある別の処理で入力値として使用することが可能になります。

.NET Add-in Extensibility を使用することにより，UFT でカスタム・テーブル（グリッド）コントロールのテーブル・チェックポイント値と出力値をサポートできます。

テーブル・チェックポイントおよび出力値のサポートを実装するには，検証クラスを **VerificationServerBase** クラスから継承して カスタム・サーバ に追加し，必要なメソッドをオーバーライドします（詳細は次を参照してください）。.NET Add-in Extensibility 設定ファイルで，各カスタム・テーブル・コントロールを **SwfTable** テスト・オブジェクトと，カスタム・サーバ の検証クラスにマッピングします。検証クラス・メソッドの構文については，『UFT .NET Add-in Extensibility API Reference』（.NET Add-in Extensibility SDK オンライン・ヘルプで利用可能）を参照してください。

注 : UFT カスタム・サーバ の設定ウィザードで カスタム・サーバ を作成する場合，ウィザードで生成されるソース・コードには，テーブル・チェックポイントおよび出力値のサポート用のコメント・コードは含まれません。したがって，実装を手動で追加してください。

テーブル・チェックポイントおよび出力値をカスタム・テーブル・オブジェクトで実装するには，次の手順で行います。

1 カスタム・テーブル・コントロールを **SwfTable** テスト・オブジェクト・クラスにマッピングします。これにより UFT は，**SwfTable** テスト・オブジェクトを使用して，GUI テストまたはコンポーネントのカスタム・テーブル・コントロールを表します。

.NET Add-in Extensibility 設定ファイル（**<UFT インストール・フォルダ>\dat\SwfConfig.xml**）で **Control** 要素を作成し，**Type**属性でカスタム・テーブル・コントロール名を指定し，**MappedTo** 属性を **SwfTable** に設定します。

SwfConfig.xml ファイルの詳細については，90 ページ「UFT Windows Forms Extensibility の設定方法」と .NET Add-in Extensibility Configuration Schema ヘルプ（.NET Add-in Extensibility SDK オンライン・ヘルプで利用可能）を参照してください。

2 このカスタム・テーブル・コントロールの カスタム・サーバ で使用するテーブル検証の設定情報を指定します。

上記の **SwfConfig.xml** ファイルで，**CustomVerify** 要素を定義します。この要素では次の内容を指定します。

➤ 実行時コンテキスト。この要素では，必ず **AUT** にします。

➤ このコントロールに適用するテーブル・チェックポイントおよび出力値サポートの実装を含むカスタム・サーバ（DLL）の名前。

➤ カスタム・サーバ（DLL）内の検証クラスのタイプ名。ラッピングしている名前空間を含みます。

CustomVerify 要素のサンプルを示します。

```
<Control Type="System.Windows.Forms.DataGridView"  MappedTo="SwfTable">
        <CustomRecord>
        ...
        ...
        </CustomRecord>
        <CustomReplay>
        ...
        ...
        </CustomReplay>
        <CustomVerify>
            <Context>AUT</Context>
            <DllName>C:\MyProducts\Bin\\VfySrv.dll</DllName>
            <TypeName>VfySrv.DataGridCPSrv</TypeName>
        </CustomVerify>
        <Settings>
</Control>
```

3 検証クラスでは, 次の protected メソッドをオーバーライドすることにより, UFT はテーブル・チェックポイントおよび出力値のサポートで必要になるデータを取得できるようになります。

➤ **GetTableData**

UFT はこのメソッドを呼び出すことにより，指定された範囲の行からテーブル・データを取得し，オブジェクト配列として返します。

テーブル・チェックポイントおよび出力値の操作では，UFT はこのメソッドの前に **GetTableRowRange** メソッドを呼び出します。これにより，**GetTableData** メソッドはテーブルのデータ範囲内にある最初と最後の行を認識します。

➤ **GetTableRowRange**

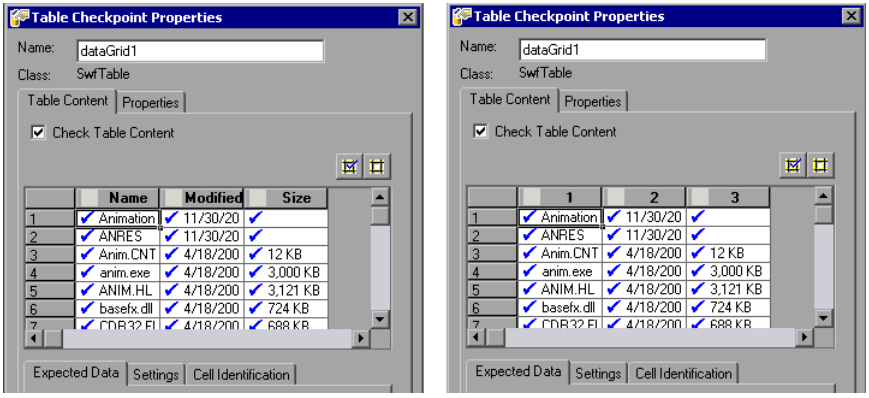
UFT は，このメソッドを呼び出すことにより，チェックポイントおよび出力値に含まれるテーブル行の数と範囲を取得します。

チェックポイントおよび出力値の操作では，UFT はこのメソッドを **GetTableData** メソッドの前に呼び出します。**GetTableRowRange** メソッドは，テーブルのデータ範囲内にある最初と最後の行の値を初期化します。**GetTableData** メソッドは，この値を入力値として使用します。

➤ **GetTableColNames**

UFT は，このメソッドを呼び出して，カラム名を文字列の配列として取得します。UFT では，この内容が［Table Checkpoint Properties］ダイアログ・ボックスと［Table Output Value Properties］ダイアログ・ボックスでカラム名として表示されます。このメソッドを実装しないと，このダイアログ・ボックスではカラム名ではなく数値が表示されます。

次の図は，GetTableColNames を実装しない場合の［Table Checkpoint Properties］ダイアログ・ボックスの例です。



次のサンプル・コード（C# で記述）は，GetTableData メソッド，GetTableColNames, メソッド，GetTableRowRange メソッドの実装例を示しています。

```
using System;
using System.Collections.Generic;
using System.Text;
using Mercury.QTP.CustomServer;
using System.Windows.Forms;
namespace VfySrv
{
  public class DataGridCPSrv :VerificationServerBase
   {
   /// GetTableData() is called by UFT to retrieve the data in a table.
   /// The following base class properties are used:
   ///    SourceControl - Reference to the grid (table) object
   ///    FirstRow - The (zero-based) row number of the start of the checkpoint
   ///                  or output value
   ///    LastRow - The (zero-based) row number of the end of the checkpoint
   ///                  or output value
   /// Returns a two-dimensional array of objects.
   protected override object[,] GetTableData()
    {
     DataGridView GridView = (DataGridView)(base.SourceControl);
     int TotalRows = GridView.Rows.Count;
     int TotalColumns = GridView.Columns.Count;
     int FirstRowN = base.FirstRow;
     int LastRowN = base.LastRow;
     TotalRows = LastRowN - FirstRown + 1;
     object[,] Data = new object[TotalRows, TotalColumns];
     DataGridViewRowCollection Rows = GridView.Rows;
      for (int i = FirstRowN; i <= LastRowN; i++)
       {
        DataGridViewRow Row = Rows[i];
        DataGridViewCellCollection Cells = Row.Cells;
        for (int k = 0; k < TotalColumns; k++)
         {
          Data[i - FirstRown, k] = Cells[k].Value;
         }
       }
      return Data;
    }
```

```
        /// GetTableColNames is called by UFT to
        /// retrieve the column names of the table.
        /// Returns an array of column names.
        protected override string[] GetTableColNames()
         {
           DataGridView GridView = (DataGridView)(this.SourceControl);
           int TotalColumns = GridView.Columns.Count;
           string[] ColNames = new string[TotalColumns];
           for (int i = 0; i < TotalColumns; i++)
            {
              ColNames[i] = GridView.Columns[i].HeaderText;
            }
           return ColNames;
         }

        /// GetTableRowRange is called by UFT to
        /// obtain the number of rows in the table.
        protected override void GetTableRowRange
                          (out int FirstVisible, out int LastVisible, out int Total)
         {
           DataGridView GridView = (DataGridView)(this.SourceControl);
           DataGridViewRowCollection Rows = GridView.Rows;
           FirstVisible = -1;
           LastVisible = Rows.Count -1;
           for (int i = 0; i < Rows.Count; i++)
            {
              if (Rows[i].Visible == false)
                continue;
              FirstVisible = i;
              break;
            }
           for (int i = FirstVisible + 1; i < Rows.Count; i++)
            {
              if (Rows[i].Visible)
                 continue;
              LastVisible = i;
              break;
            }
           FirstVisible++;
           LastVisible++;
           Total = GridView.Rows.Count;
         }
      }
    }
```

Application Under Test のコードを UFT コンテキストから実行

カスタム・サーバ を **UFT** コンテキストで実行する場合，コントロールは別の実行時プロセス内にあるので，コントロールには直接アクセスできません。コントロールに直接アクセスするには，コードの一部を **Application under test** コンテキストで実行する必要があり，補助クラスを使用します。

Application under test コンテキストで実行するコードを UFT コンテキストから呼び出すには，**CustomAssistantBase** から継承した補助クラスを実装します。補助クラスのインスタンスを生成するには，**CreateRemoteObject** を呼び出します。オブジェクトは，**SetTargetControl** でコントロールにアタッチしてから使用します。

SetTargetControl を呼び出したら，次のいずれかの方法で補助クラスのメソッドを呼び出すことができます。

- **Application under test** プロセスの任意のスレッドでメソッドを実行可能であれば，コントロールの値の読み出しと設定，コントロール・メソッドの呼び出しは簡単な **obj.Member** 構文で実行できます。

```
int i = oMyAssistant.Add(1,2);
```

- コントロールのスレッドでメソッドを実行する必要がある場合は，**InvokeAssistant** メソッドを使用します。

```
int i = (int)InvokeAssistant(oMyAssistant, "Add", 1, 2);
```

ヒント：EventListenerBase の使用が可能です。このクラスは，コントロール・イベントのリッスンをサポートする補助クラスです。

よく使用される API 呼び出し

この項では，よく使用される API 呼び出しについて簡単に説明します。メソッドを実装する前に，この項の内容をよく読んでください。

ここでは，特に記載がない場合，CustomServerBase で使用するメソッドについて説明します。

詳細については，『UFT .NET Add-in Extensibility API Reference』（UFT .NET Add-in Extensibility オンライン・ヘルプで提供）を参照してください。

Test Record メソッド

AddHandler	イベント・ハンドラを，イベントの最初のハンドラとして追加します。
RecordFunction	テスト内のステップを記録します。

Test Record コールバック・メソッド

GetWndMessageFilter	ウィンドウ・メッセージ・フィルタを設定するメソッドであり，UFT によって呼び出されます。
InitEventListener	イベント・ハンドラをロードしてイベントのリッスンを開始するメソッドであり，UFT によって呼び出されます。
OnMessage	UFT がウィンドウ・メッセージをフックした時点で呼び出されます。
ReleaseEventListener	イベントのリッスンを停止します。

Test Run メソッド

DragAndDrop，KeyDown，KeyUp，MouseClick，MouseDblClick，MouseDown，MouseMove，MouseUp，PressKey，PressNKeys，SendKeys，SendString	マウス・およびキーボードのシミュレーション・メソッドです。
PrepareForReplay	アクション実行で使用するコントロールを準備します。
ReplayReportStep	テスト・レポートにイベントを書き込みます。
ReplayReportStep	エラー・メッセージを生成し，報告されたステップ・ステータスを変更します。
ShowError	.NET 警告アイコンを表示します。
TestObjectInvokeMethod	テスト・オブジェクトの IDispatch インタフェースが公開しているメソッドの 1 つを呼び出します。

プロセス間のメソッド

AddRemoteEventListener	**Application under test** プロセスで **EventListener** インスタンスを生成します。
CreateRemoteObject	**Application under test** プロセスで補助オブジェクトのインスタンスを生成します。
GetEventArgs (IEventArgsHelper)	**EventArgs** オブジェクトを取得して逆シリアル化します。
Init (IEventArgsHelper)	**EventArguments** ヘルパ・クラスを **EventArgs** オブジェクトで初期化します。
InvokeAssistant	コントロールのスレッドで **CustomAssistantBase** クラスのメソッドを呼び出します。
InvokeCustomServer (EventsListenerBase)	**UFT** プロセスで実行中の カスタム・サーバ のメソッドを **Application under test** プロセスから呼び出します。
SetTargetControl (CustomAssistantBase)	コントロールのウィンドウ・ハンドルで，ソースのカスタム・コントロールにアタッチします。

一般的なメソッド

ControlGetProperty	スレッド・セーフでないコントロールのプロパティを取得します。
ControlInvokeMethod	スレッド・セーフでないコントロールのメソッドを呼び出します。
ControlSetProperty	スレッド・セーフでないコントロールのプロパティを設定します。
GetSettingsValue	このコントロールについて, 設定ファイルでの設定値からパラメータ値を取得します。
GetSettingsXML	このコントロールについて, 設定ファイルで入力されているこのコントロールの設定を返します。

テーブル・チェックポイントおよび出力値をサポートするメソッド

GetTableData (VerificationServerBase)	テーブルのデータを取得するメソッドであり，UFT によって呼び出されます。
GetTableRowRange (VerificationServerBase)	テーブルの最初と最後の行を取得するメソッドであり, UFT によって呼び出されます。
GetTableColNames (VerificationServerBase)	テーブル・カラムの名前を取得するメソッドであり, UFT によって呼び出されます。

XML ファイルを使用してカスタム・コントロールのサポートを拡張

カスタム・コントロールのサポートを実装する際，カスタム・コントロールで使用する Test Record および Test Run 命令をコントロール定義ファイルに入力することにより，.NET DLL のプログラミングが不要になります（コントロール定義ファイルは，カスタマイズしたいコントロールごとに作成します)。カスタム・コントロールの実装命令のロードを UFT に指示するには，.NET Add-in Extensibility 設定ファイル（**SwfConfig.xml**）で各コントロール定義ファイルを指定します。

注：XML ファイルを使用してサポートを拡張する場合，UFT は XML ファイルを元にアドホックな .NET DLL を生成します。このアドホック .NET DLL が，このコントロールのカスタム・サーバになります。

この方法で実装する場合，.NET 開発環境（オブジェクト・ブラウザ，デバッガ，テーブル・チェックポイントや出力値の作成機能など）はサポートされません。ただし，.NET 開発環境がなくてもカスタム・コントロール・サポートの実装が可能なので，現場で比較的短時間で実装できるという利点があります。

この機能が最も適しているのは，コントロールが比較的シンプルで，ドキュメントが整備されている場合，既存のオブジェクトへのマッピングが十分に行われているが Test Record 定義を置換する必要がある場合，数個のテスト・オブジェクトの Test Run メソッドを置換または追加する必要がある場合などです。

コントロール定義ファイルについて

コントロール定義ファイルでは，コントロールの記録をカスタマイズする **Record** 要素と，テスト・オブジェクト・メソッドをカスタマイズする **Replay** 要素を指定できます。

- **Record** 要素では，記録セッション中に UFT がテスト（またはコンポーネント）にステップを追加する条件となるコントロール・イベントを指定します。ステップとは，カスタム・コントロールのテスト・オブジェクトのテスト・オブジェクト・メソッドに対する呼び出しです。
- **Replay** 要素は，実行セッション中に各テスト・オブジェクト・メソッドに対してコントロールで UFT が実行する操作を指定します。

Record 要素と **Replay** 要素は，両方を指定する必要はありません。

- カスタム・テスト・オブジェクトで，既存のテスト・オブジェクトでの定義とは異なる Test Record の実装を使用する場合，カスタム・コントロールで使用するコントロール定義ファイル内に **Record** 要素を作成します。
- 同様に，カスタム・テスト・オブジェクトで，既存のテスト・オブジェクトでの定義とは異なる Test Run の実装を使用する場合，カスタム・コントロールで使用するコントロール定義ファイル内に **Replay** 要素を作成します。

Record 要素を作成すると，既存のテスト・オブジェクトの Test Record 実装全体が置換されます。**Replay** 要素を作成すると，既存のオブジェクトの Test Run 実装が継承され，それが拡張されます。テスト・オブジェクトのマッピング・オプションについては，22 ページ「カスタム・コントロールをテスト・オブジェクトにマッピング」を参照してください。

コントロール定義 XML ファイルの要素の詳細については，.NET Add-in Extensibility Control Definition Schema ヘルプ（.NET Add-in Extensibility SDK オンライン・ヘルプで利用可能）を参照してください。

コントロール定義ファイルの例

次に，コントロール定義ファイルの例を示します。各 **MouseUp** イベントで，値が変更されたオブジェクトを処理しています。値はオブジェクトの **Value** プロパティに格納されており，**MouseUp** イベント・ハンドラには **Button**，**Clicks**，**Delta**，**X**，**Y** というイベント引数があります。

Record 要素では，**MouseUp** イベントを **SetValue** コマンドに変換する方法を記述します。**Replay** 要素では，記録された Value にオブジェクトの値を設定し，デバッグ用にマウス・ポインタの位置を表示する **SetValue** コマンドを定義します。

```
<?xml version="1.0" encoding="UTF-8"?>
<Customization>
    <Record>
        <Events>
            <Event name="MouseUp" enabled="true">
                <RecordedCommand name="SetValue">
                    <Parameter>
                        Sender.Value
                    </Parameter>
                    <Parameter lang="C#">
                        String xy;
                        xy = EventArgs.X + ";" + EventArgs.Y;
                        Parameter = xy;
                    </Parameter>
                </RecordedCommand>
            </Event>
        </Events>
    </Record>
    <Replay>
        <Methods>
            <Method name="SetValue">
                <Parameters>
                    <Parameter type="int" name="Value"/>
                    <Parameter type="String" name="MousePosition"/>
                </Parameters>
                <MethodBody>
                    RtObject.Value = Value;
                    System.Windows.Forms.MessageBox.Show(MousePosition,
                        "Mouse Position at Record Time");
                </MethodBody>
            </Method>
        </Methods>
    </Replay>
</Customization>
```

サンプルの .NET Add-in Extensibility の使用方法

.NET Add-in Extensibility SDK には，.NET Add-in Extensibility の学習支援として，サンプルのサポート・セットが収録されています。ツールキット・サポート・セットのファイルは，**<UFT .NET Add-in Extensibility SDK インストール・フォルダ>\samples\WinFormsExtSample** フォルダにインストールされます。ツールキット・サポート・セットを開発する際の参考として，このファイルを活用してください。

サンプルのサポート・セットは，SandBar カスタム .NET Windows Forms コントロールの UFT サポートを拡張します。このサンプルのカスタム・サーバは，115 ページ「複雑な .NET Windows Forms カスタム・コントロールの作成の実習」で作成したものと類似しています。

WinFormsExtSample フォルダにある **SandbarSample.sln** ソリューション・ファイルには，設定ファイルと，SandBar コントロールをサポートする完全実装のカスタム・サーバが含まれています。SandBarCustomServer の実装は C# と Visual Basic で提供され，それぞれ別のプロジェクト（**SandbarCustomServer** と **VBSandbarCustomServer**）で提供されています。さらに，SandbarSample ソリューションには，SandBar ツールバー・コントロール（**SandbarTestApp**）を使用するサンプルの .NET Windows Forms アプリケーションも含まれています。

サンプルのツールキット・サポート・セットを UFT にデプロイする前と後に，サンプル・アプリケーションで UFT GUI テストを作成して実行すると，拡張機能によって UFT とカスタム・コントロールとの相互作用がどのように影響されるのかを確認することができます。

SandBar サポートのサンプルを使用する際の考慮事項

- SandbarSample ソリューションを開くには，Microsoft Visual Studio 2005 以降を使用します。
- SandbarSample ソリューションのビルドでは，次の内容が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。
  - UFT .NET Add-in Extensibility SDK
  - SandBar for .NET 2.0/3.x
    （http://www.divil.co.uk/net/download.aspx?product=2&license=5 からダウンロードできます）
- SandbarSample ソリューションをビルドしたら，C# または Visual Basic のカスタム・サーバをデプロイします（第 5 章「サポート・セットの設定とデプロイ」の手順で作成）。

- **Configuration.xml** の内容に基づいて **SwfConfig.xml** ファイルを更新する前に，次の点を確認してください。SandbarSample ソリューションの **Configuration.xml** ファイルは，C# プロジェクトによって生成される DLL（**<UFT .NET Add-in Extensibility インストール・フォルダ>\samples\WinFormsExtSample\Bin** に格納）を使用するように設定されています。
  - **VBCustomSandBarSrv.dll** を使用するには，SandbarCustomServer が現れるところをVBCustomSandBarSrv.dll で置き換える必要があります。
  - DLL ファイルが別の場所に保存されている場合，**DllName** 要素のパスを更新します。

トラブルシューティングと制限事項 - 設計したサポートの実行

この項では，サポート・セットの開発に関連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。

カスタム・サーバが一部のウィンドウ・メッセージを受信しない

記録セッション中，カスタム・コントロールにマッピングされているカスタム・サーバは，カスタム・コントロール上でなんらかの操作が実行されないと，作成されません。

GetWndMessageFilter メソッドを設計し，ほかのコントロールで発生したメッセージをカスタム・サーバで処理するように指定する場合，カスタム・サーバが作成されないとメッセージは処理できません。

したがって，カスタム・コントロールをクリックしないと，アプリケーション内にあるほかのコントロールで発生したメッセージをカスタム・サーバで取得および処理できない場合があります。

カスタム・コントロールでの記録のサポートを実装する方法によっては，サポート・セットを使用する UFT ユーザにこの問題について説明する必要があります。

UFT でのテスト実行中に一般実行エラーが発生

.NET Add-in Extensibility API を Microsoft .NET Framework 1.1 で使用する場合，テスト実行中に**一般実行エラー**が発生することがあります。この問題の原因は，テスト対象アプリケーション（AUT）で発生する**実行エンジン例外**エラーです。

回避策：Microsoft .NET Framework 1.1 の Service Pack 1 以降をインストールします。

UFT でのテスト実行中に実行エラーが発生

XML ベースのカスタム・サーバを使用する環境に Microsoft .NET Framework の複数バージョンをインストールすると，実行セッション中に実行時エラーが発生することがあります。ログ・ファイルには，設定ファイルにコンパイル・エラーが含まれていることを示すエラー・メッセージが記録されます。この問題は，Microsoft .NET Framework version 2.0 以降でコンパイルされたアセンブリは，これよりも古いバージョンの Microsoft .NET Framework では認識されないことが原因で発生します。

回避策：次の手順のいずれかを実行します。

- **方法 1：HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\.NETFramework レジストリ・キーに** DWORD 値 "OnlyUseLatestCLR"=dword:00000001 を追加します。
- **方法 2：**テスト中の .NET アプリケーションに設定ファイルがある場合，次の内容をファイルに追加します。

```
<configuration>
    <startup>
        <supportedRuntime version="v2.0.50727"/>
    </startup>
</configuration>
```

設定ファイルには **<実行可能ファイルの名前>.exe.Config** という形式で名前を付け，テスト対象の .NET アプリケーションの実行可能ファイルと同じフォルダに保存してください。

第 5 章

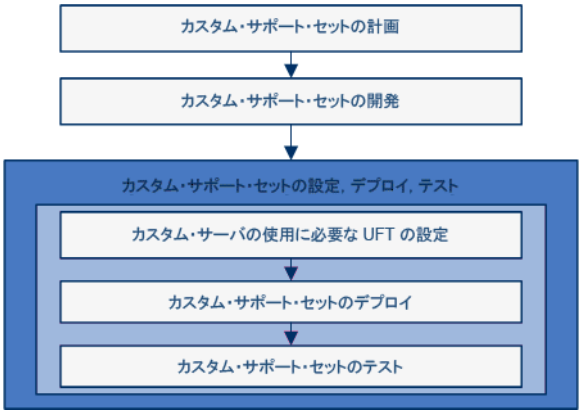
サポート・セットの設定とデプロイ

カスタム・サーバの実装のデプロイが完了したら，UFT .NET Add-in Extensibility サポート・セットの設定とデプロイを行います。

本章の内容

デプロイメント・ワークフローについて

.NET Add-in Extensibility サポート・セットのデプロイは，下の図で強調表示した工程に従って行います。次に，それぞれの工程を詳しく説明します。



カスタム・サーバの使用に必要な UFT の設定

.NET Add-in Extensibility 設定ファイル（**SwfConfig.xml**）は，カスタム・サーバのロードに必要な設定情報を UFT に提供するファイルです。

UFT Windows Forms Extensibility の設定方法

カスタム・サーバ を正しくロードする方法を UFT に指示するには，.NET Add-in Extensibility 設定ファイルに情報を入力します。**SwfConfig.xml** ファイルは **<UFT インストール・フォルダ>\dat** フォルダに格納されています。

SwfConfig.xml ファイルには，次の方法で設定情報を入力できます。

- テキスト・エディタを使用してファイルを手動で編集する方法。
- UFT カスタム・サーバの設定ウィザードで生成した **configuration.xml** ファイルの内容をコピーする方法。

ウィザードの詳細については，58 ページ「.NET DLL を使用してカスタム・コントロールのサポートを拡張」を参照してください。

configuration.xml ファイルから情報をコピーする方法については，93 ページ「UFT カスタム・サーバ設定ウィザードで生成された設定情報のコピー」を参照してください。

UFT Windows Forms Extensibility の設定では，カスタム・サーバの実装で選択したコーディング・オプションに従って要素を定義します。

➤ 91 ページ「.NET DLL カスタム・サーバを使用する場合」

➤ 92 ページ「XML カスタム・サーバ を使用する場合」

.NET DLL カスタム・サーバを使用する場合

SwfConfig.xml ファイルでは，.NET DLL カスタム・サーバを使用して実装するカスタム .NET コントロールごとに，次の内容を定義できます。

➤ **MappedTo** 属性。標準の汎用テスト・オブジェクト（**SwfObject**）以外のテスト・オブジェクトを使用する場合に定義します。

➤ **CustomRecord** 要素。コントロールでの記録をカスタマイズします。

➤ **CustomReplay** 要素。カスタム・コントロールでテスト・ステップを実行する方法をカスタマイズします。

➤ **CustomVerify** 要素。カスタム・テーブル・コントロールで使用するチェックポイントおよび出力値のサポートを追加する場合に指定します。

➤ **Settings** 要素。**Parameter** 要素を使用して，実行時にカスタム・サーバに値を渡すことができます。

XML カスタム・サーバ を使用する場合

SwfConfig.xml ファイルでは, XML カスタム・サーバを使用して実装するカスタム .NET コントロールごとに，次の内容を定義できます。

- **MappedTo** 属性。標準の汎用テスト・グリッド・オブジェクトである **SwfTable** 以外のテスト・オブジェクトで使用するカスタム・コントロールが必要な場合に指定します。
- **CustomRecord** 要素の **Context** 属性。コントロールでの記録操作をカスタマイズする場合に指定します。
- **CustomReplay** 要素の **Context** 属性。テスト・ステップをカスタム・コントロールで実行する方法をカスタマイズする場合に指定します。
- **Settings** 要素。**Parameter** 要素を使用して，実行時にカスタム・サーバに値を渡すことができます。

注 : テストを開くと，UFT によってカスタム・サーバがロードされます。したがって，カスタム・サーバを .NET DLL として実装する場合，カスタム・サーバのロード後に行った DLL の変更は，次回テストを開いたときに有効になります。

.NET Add-in Extensibility 設定ファイル（**SwfConfig.xml**）の要素の詳細については，.NET Add-in Extensibility Configuration Schema ヘルプ （.NET Add-in Extensibility SDK オンライン・ヘルプで提供）を参照してください。

UFT カスタム・サーバ設定ウィザードで生成された設定情報のコピー

UFT カスタム・サーバ設定ウィザードを実行してカスタム・サーバを作成する手順では，XML 設定セグメントが生成されます。ウィザードによって生成されるこのセグメントは，.NET Add-in Extensibility 設定ファイルに入力する設定情報として使用できます。

カスタム・サーバをデプロイする前に XML 設定セグメントの内容をコピーするには，次の手順で行います。

1. プロジェクト内にある **Configuration.xml** セグメント・ファイルを編集し，内容に誤りがないことを確認します。**DllName** 要素の値を，カスタム・サーバをデプロイする場所に設定します。**Test Record** または **Test Run** が同じ実行時コンテキストでロードされていない場合，**Context** 値を編集します。
2. **<Control> から </Control>** のノード全体をコピーします。ただし，要素を囲んでいる **<Controls>** タグは除外してください。
3. .NET Add-in Extensibility 設定ファイル（**<UFT インストール・フォルダ>\dat\SwfConfig.xml**）を開きます。**Configuration.xml** の最後にある **</Controls>** タグの前に，**Control** ノードを貼り付けます。
4. ファイルを保存します。UFT が開いている場合は，終了してから再度起動します。これにより，**SwfConfig.xml** の変更内容が有効になります。

注 : 作成した設定ファイルを，**<UFT インストール・フォルダ>\dat\SwfConfig.xsd** ファイルに基づいて検証します。

.NET Add-in Extensibility 設定ファイルの例

次に，UFT の設定を行うファイルの例を示します。この例では，次のコントロールを認識します。

➤ **MyCompany.WinControls.MyListView** コントロールのサポートは **CustomMyListView.CustListView** .NET DLL カスタム・サーバで実装されます。カスタム・サーバは GAC にインストールされないので，DLL 名はパスおよびファイル名として指定されます（GAC の標準的な構文に従ったタイプ名としては渡されません）。

MyListView は **SwfListView** テスト・オブジェクトにマッピングされ，テスト対象アプリケーションのコンテキストで実行されます。

➤ **mySmileyControls.SmileyControl2** コントロールのサポートは，XML ファイルで実装されます。したがって，実装を含むコントロール定義ファイルのパスおよびファイル名は，**Parameter** 要素を使用して実行時に UFT に渡されます。

SmileyControl2 コントロールは，**SwfConfig.xml** ファイルではテスト・オブジェクトに明示的にマッピングされていないので，UFT デフォルトの汎用テスト・オブジェクトである **SwfObject** にマッピングされます。

➤ **System.Windows.Forms.DataGridView** コントロールで使用するカスタマイズ・レコードと実行サポートは，.NET DLL カスタム・サーバ（**CustomMyTable.dll**）で実装されます。**System.Windows.Forms.DataGridView** コントロールで使用するテーブル・チェックポイントおよび出力値のサポートは，.NET DLL カスタム・サーバ（**VfySrv.dll**）で実装されます。

DataGridView は **SwfTable** テスト・オブジェクトにマッピングする必要があります（これは，スキーマの **TableElement** 複合タイプ要素の制約です）。さらに，カスタマイズしたサポートにはテーブル・チェックポイントおよび出力値があるので，テスト対象アプリケーションのコンテキストで実行する必要があります。

```
<?xml version="1.0" encoding="UTF-8"?>
<Controls>
    <Control Type="MyCompany.WinControls.MyListView "
            MappedTo="SwfListView">
        <CustomRecord>
            <Component>
                <Context>AUT</Context>
                <DllName>C:\MyProducts\Bin\CustomMyList View.dll</DllName>
                <TypeName>CustomMyListView.CustListView</TypeName>
            </Component>
        </CustomRecord>
        <CustomReplay>
            <Component>
                <Context>AUT</Context>
                <DllName>C:\MyProducts\Bin\CustomMyList View.dll</DllName>
                <TypeName>CustomMyListView.CustListView</TypeName>
            </Component>
        </CustomReplay>
        <Settings>
            <Parameter Name="sample name">sample value</Parameter>
            <Parameter Name="ConfigPath">C:\Program Files\HP\Unified
Functional Testing\dat\Extensibility\dotNET\MyContrSIM.xml</Parameter>
        </Settings>
    </Control>

    <Control Type="mySmileyControls.SmileyControl2">
        <Settings>
            <Parameter Name="ConfigPath">d:\UFT\bin\ConfigSmiley.xml
            </Parameter>
        </Settings>
        <CustomRecord>
            <Component>
                <Context>AUT-XML</Context>
            </Component>
        </CustomRecord>
        <CustomReplay>
            <Component>
                <Context>AUT-XML</Context>
            </Component>
        </CustomReplay>
    </Control>
```

```
        <Control Type="System.Windows.Forms.DataGridView"
                 MappedTo="SwfTable">
            <CustomRecord>
                <Component>
                    <Context>QTP</Context>
                    <DllName>C:\MyProducts\Bin\CustomMyTable.dll</DllName>
                    <TypeName>CustomMyTable.CustTableView</TypeName>
                </Component>
            </CustomRecord>
            <CustomReplay>
                <Component>
                    <Context>AUT-XML</Context>
                </Component>
            </CustomReplay>
            <CustomVerify>
                 <Context>AUT</Context>
                 <DllName>C:\MyProducts\Bin\VfySrv.dll</DllName>
                 <TypeName>VfySrv.DataGridCPSrv</TypeName>
             </CustomVerify>
            <Settings>
                <Parameter Name="sample name">sample value</Parameter>
            </Settings>
        </Control>

    </Controls>
```

カスタム・サポート・セットのデプロイ

カスタム・コントロールの UFT GUI テスト サポート拡張に必要な次の手順は，デプロイメントです。この手順では，作成したすべてのファイルを正しい場所に格納します。これにより，UFT はカスタム・サポートを使用できるようになります。

カスタム・サポートのデプロイが完了したら，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，そのアプリケーションで UFT 操作を実行することにより，作成したサポートがどのように動作するかを確認できます。

ファイルの格納場所

作成したサポート・セットをデプロイするには，次の表に従ってファイルを正しい場所に格納する必要があります。UFT を終了してから，ファイルを配置してください。

ファイル名	場所
SwfConfig.xml	**<UFT インストール・パス>\dat**
<テスト・オブジェクト設定ファイル名>.xml **注 :** テスト・オブジェクト設定ファイルは複数作成でき，任意の名前を付けることができます。	➤ **<UFT インストール・パス>\dat\Extensibility\DotNet** ➤ **<ALM 用 UFT アドインのインストール・パス>\dat\Extensibility\DotNet** （任意。ALM 用 UFT アドインがインストールされている場合のみ指定します）
<コントロール定義ファイル名>.xml **注 :** コントロール定義ファイルは，XML コーディング・オプションを使用してカスタム・サーバを作成する際に使用されます。コントロール定義ファイルは複数作成できます（カスタム・コントロールごとに 1 つ）。	➤ **<UFT インストール・パス>\dat\Extensibility\DotNet** ➤ **<ALM 用 UFT アドインのインストール・パス>\dat\Extensibility\DotNet** （任意。ALM 用 UFT アドインがインストールされている場合のみ指定します）
<カスタム・サーバ・ファイル名>.dll **注 :** このタイプのカスタム・サーバを使用するのは，.NET DLL コーディング・オプションを使用してカスタム・サーバを作成する場合です。カスタム・サーバは複数作成できます（カスタム・コントロールごとに 1 つ）。	**SwfConfig.xml** ファイルで，コンパイルした カスタム・サーバ（DLL）の場所を指定します。

デプロイしたサポートの変更

UFT にデプロイ済みのサポート・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。

- .NET Add-in Extensibility 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 を再度起動し，GUI テストを開きます。

デプロイしたサポートの削除

デプロイ済みのカスタム・コントロールのサポートを UFT から削除するには，**SwfConfig.xml ファイル**（**<UFT インストール・パス>\dat**）から該当セクションを削除し，テスト・オブジェクト設定ファイルを **<UFT インストール・パス>\dat\Extensibility\DotNet** から削除します。

テスト・オブジェクト設定ファイルに新しく追加したテスト・オブジェクト・メソッドのサポートを削除する場合は，メソッド定義（またはファイル全体）を削除する必要があります。これにより，このメソッドを呼び出すテスト・ステップが作成されなくなります。テスト・オブジェクト設定ファイル（**<UFT インストール・パス>\Dat\Extensibility\DotNet**（さらに，必要に応じて **<ALM 用 UFT アドインのインストール・パス>\Dat\Extensibility\DotNet**）を変更または削除してください。

カスタム・サポート・セットのテスト

カスタム・サポートでは増分テストを行うことをお勧めします。まず，サポート・セットの基本機能，次に実装をテストします。

- 99 ページ「サポート・セットの基本機能のテスト」
- 101 ページ「実装のテスト」

サポート・セットの基本機能のテスト

基本的な .NET Windows Forms 設定ファイルで，UFT が各コントロールで使用するテスト・オブジェクト・クラスを識別するための情報を定義した後，テスト・オブジェクト設定ファイルでテスト・オブジェクト・モデルを定義（オプション）すると，そのサポート・セットにある既存の機能のテストを実行できます。テストを行うには，サポート・セットをデプロイし，ユーザ環境での UFT とコントロールのインタラクションを確認してください。

テスト・オブジェクト・クラスの定義と，カスタム .NET Windows Forms コントロールへのマッピングが完了した後にサポート・セットをテストするには，次の手順で行います。

1 テスト・オブジェクト設定ファイルで，**TypeInformation\DevelopmentMode** 属性を **true** に設定します。これにより，UFT は起動時にすべてのテスト・オブジェクト・クラス情報を読み込みます。サポート・セットの開発が完了した時点で，この属性を **false** に設定してください。

2 UFT コンピュータにサポート・セットをデプロイします。サポート・セットのファイルを UFT インストール・フォルダ内の正しい場所にコピーします（97 ページ「ファイルの格納場所」を参照してください）。

3 UFT を開き，.NET Add-in をロードしてから GUI テストを開きます（UFT の起動時に［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』を参照してください）。

4 カスタム・コントロールでアプリケーションを開きます。

5 UFT は，作成したマッピングの定義に基づいてコントロールを識別します。

コントロールの情報を取得するには，［オブジェクト リポジトリ］ダイアログ・ボックスの［**ローカルへオブジェクトを追加**］ボタンを使用します。

6 テスト・オブジェクト設定ファイルを作成すると，定義した内容は UFT に反映されます。

a テスト・オブジェクト・メソッドをテスト・オブジェクト・クラスに追加した場合には，オブジェクト・スパイを使用します。

b 追加したテスト・オブジェクト・メソッドを使用するテスト・ステップを作成します（このテスト・オブジェクト・メソッドをサポートするカスタム・サーバを実装していない場合，テスト・ステップを実行すると実行時エラーが発生します）。

キーワード・ビュー：

変更したクラスのテスト・オブジェクトを使用してテスト・ステップを作成します。

- テスト・オブジェクト・メソッドをテスト・オブジェクト・クラスに追加した場合，[**操作**]カラムに使用可能な操作のリストとしてメソッドが表示されます。
- 操作を選択すると，選択した操作の引数の数に応じて［**値**］セルが分割されます。操作で想定される値が定義されている場合（**ListOfValues** 要素）には一覧表示されます。
- テスト・オブジェクト・メソッドで定義した説明はツールヒント，注釈の文字列は［**注釈**］カラムに表示されます。

エディタ：

変更したクラスのテスト・オブジェクトを使用してテスト・ステップを作成します。ステートメント自動補完機能では，テスト・オブジェクトに対して使用できるすべての操作と，これらの操作に対して使用できる入力値（該当する場合）が，テスト・オブジェクト設定ファイル内の定義に基づいて表示されます。

ステップ・ジェネレータ：

変更したクラスのテスト・オブジェクトを使用してテスト・ステップを作成します。テスト・オブジェクト設定ファイルで定義した操作が［**操作**］リストに表示され，各操作で定義した説明がツールヒントとして表示されます。

UFT で上記のオプションを使用する方法の詳細については，『HP Unified Functional Testing ユーザーズ・ガイド』を参照してください。

実装のテスト

ユーザ環境で使用するサポートの開発で必要な追加工程が完了したら，サポート・セットを再度デプロイし，UFT とコントロール間のインタラクションについてさらにテストを行います（GUI テストの実行と記録など）。

追加の UFT 機能で使用するサポートの開発が完了し，サポート・セットをテストするには，次の手順で行います。

1. サポート・セットの基本機能のテストについては，1 から 4 の手順（99 ページ）を行います。UFT を起動し，サポートをロードして，コントロールを使ってアプリケーションを実行します。
2. 開発したサポートで使用する UFT 機能に応じて，このサポートのテストに必要な UFT 操作を実行します。これには，アプリケーションでのテスト実行，アプリケーションでのテスト・ステップの記録などがあります。

第 6 章

簡単な .NET Windows Forms カスタム・コントロールの作成の実習

このチュートリアルでは，Microsoft TrackBar コントロール用のカスタム・サーバを作成することにより，このコントロールで UFT が **SetValue** 操作の記録と実行を実行できるようにします。カスタム・サーバの実装は C# で行いますが，Visual Basic でも同様の方法で実装できます。

このチュートリアルでは Microsoft Visual Studio 2008 環境での手順を説明していますが，Visual Studio のサポート対象バージョンであれば本書の手順でカスタム・サーバを作成できます。

注：本章では，Microsoft Visual Studio 2008 のダイアログ・ボックスと手順が掲載されています。これ以外のバージョンでは，ダイアログ・ボックスの表示内容や，ツリー構造内で **UFT CustomServer** テンプレートが表示されるノードの場所が若干異なる場合があります。

本章の内容

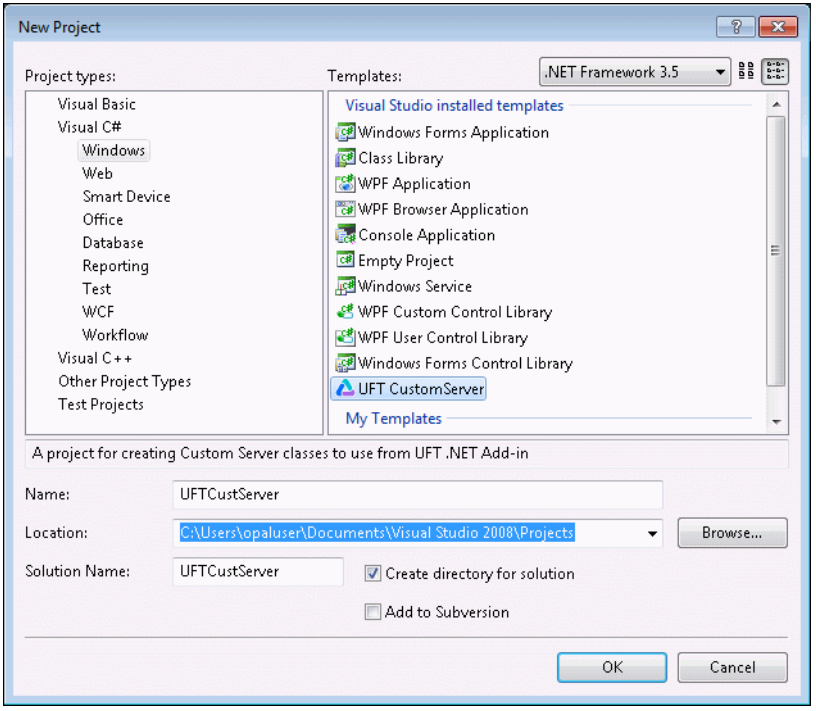
新しいサポート・セットの作成

カスタム・コントロールのサポートを作成するには，まず新しいカスタム・サーバ・プロジェクトを作成します。このプロジェクトによって，TrackBar コントロールのサポートが作成されます。

新しいカスタム・サーバ・プロジェクトを作成するには，次の手順で行います。

1 Microsoft Visual Studio を起動します。

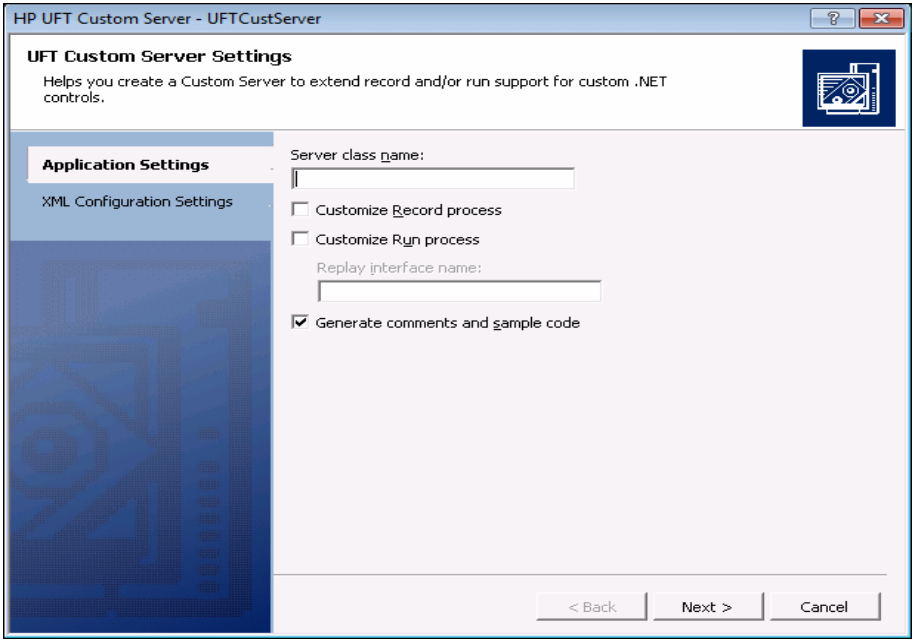
2 ［**ファイル**］＞［**新規作成**］＞［**プロジェクト**］を選択します。［プロジェクトの新規作成］ダイアログ・ボックスが開きます。



3 次の設定を指定します。

- ［**プロジェクトのタイプ**］ツリーで［**Visual C#**］＞［**Windows**］ノードを選択します。(Microsoft Visual Studio 2008 以外のバージョンを使用している場合，ツリー内で **UFT CustomServer** テンプレートが若干異なるノードに表示されることがあります)。
- ［**テンプレート**］表示枠で［**UFT CustomServer**］を選択します。
- ［**名前**］ボックスで，プロジェクト名として「UFTCustServer」と指定します。
- これ以外は，標準設定をそのまま使用します。

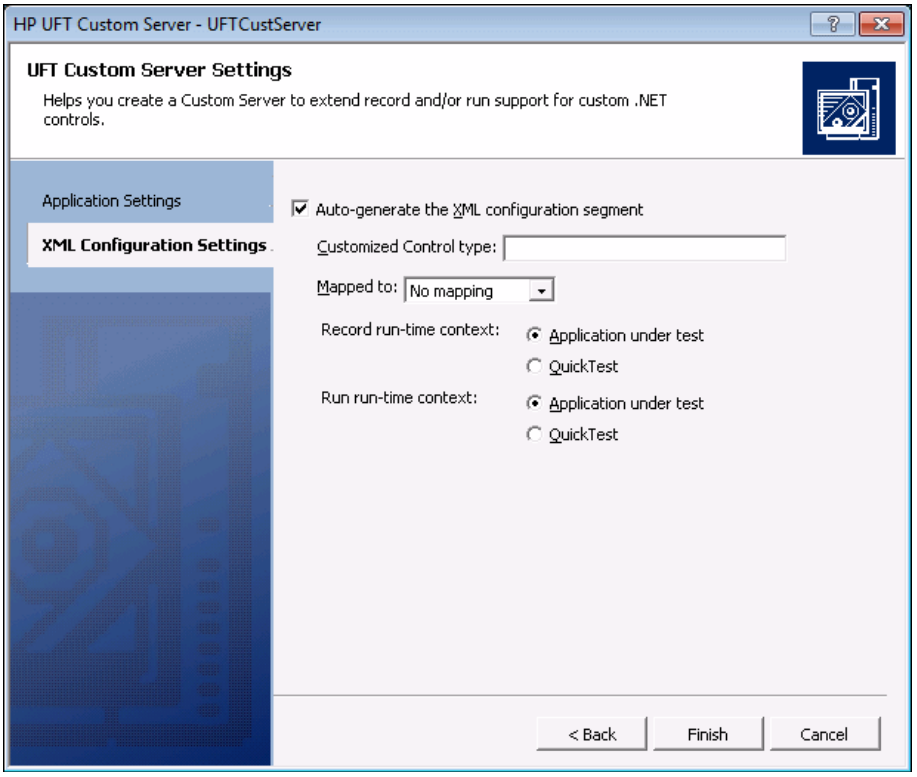
4 ［**OK**］をクリックします。UFT カスタム・サーバ設定ウィザードが起動します。



5 ［Application Settings］ページで，次の内容を設定します。

- ［**Server class name**］ボックスに，「TrackBarSrv」と入力します。
- ［**Customize Record process**］チェック・ボックスを選択します。
- ［**Customize Run process**］チェック・ボックスを選択します。
- これ以外は，標準設定をそのまま使用します。

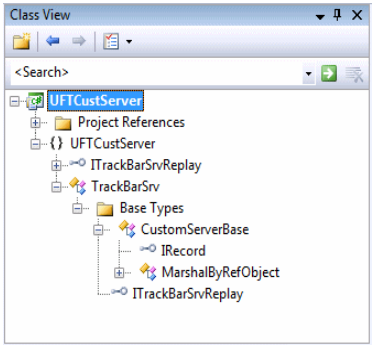
6 ［**Next**］をクリックします。［XML Configuration Settings］ページが開きます。



7 ［XML Configuration Settings］ページでは，次の内容を指定します。

- ［**Auto-generate the XML configuration segment**］チェック・ボックスを選択します。
- ［**Customized Control type**］ボックスに「System.Windows.Forms.TrackBar」と入力します。
- これ以外は，標準設定をそのまま使用します。

8 ［**Finish**］をクリックします。［Class View］ウィンドウには，**CustomServerBase** クラスから生成された **TrackBarSrv** クラスと **ITrackBarSrvReplay** インタフェースが表示されます。



Test Record ロジックの実装

ValueChanged イベント発生時に **SetValue(X)** コマンドを記録するロジックを，イベント・ハンドラ関数を使用して実装します。

Test Record ロジックを実装するには，次の手順で行います。

1 **TrackBarSrv** クラス内で，新しいメソッド **OnValueChanged** を追加する位置を特定します。たとえば，**IRecord override Methods** 領域内にあるほかのイベント・ハンドラ・メソッド（**OnMessage** など）の後に追加します。

2 次の署名で新しいメソッドを **TrackBarSrv** クラスに追加します。

```
public void OnValueChanged(object sender, EventArgs e) { }
```

注：新しいメソッドの追加は，手動で行う方法と Visual Studio のウィザードを使用する方法があります。ウィザードでは，メソッドと関数がクラスに追加されます。

---

3 追加した関数に，次の実装を追加します。

```
public void OnValueChanged(object sender, EventArgs e)
{
 System.Windows.Forms.TrackBar trackBar =
(System.Windows.Forms.TrackBar)sender;
 // get the new value
 int newValue = trackBar.Value;
 // Record SetValue command to the test
 RecordFunction("SetValue", RecordingMode.RECORD_SEND_LINE, newValue);
}
```

4 次のコードを InitEventListener メソッドに追加します。これにより，OnValueChanged イベント・ハンドラが ValueChanged イベントで登録されます。

```
public override void InitEventListener()
{
  Delegate e = new System.EventHandler(this.OnValueChanged);
  AddHandler("ValueChanged", e);
}
```

Test Run ロジックの実装

SetValue メソッドをテストの Test Run 用に実装します。

Test Run ロジックを実装するには，次の手順で行います。

1 次のメソッドの定義を **ITrackBarSrvReplay** インタフェースに追加します。

```
[ReplayInterface]
public interface ITrackBarSrvReplay
{
   void SetValue(int newValue);
}
```

2 次のメソッドの実装を，**Replay interface implementation** 領域内の **TrackBarSrv** クラスに追加します。

```
public void SetValue(int newValue)
{
 System.Windows.Forms.TrackBar trackBar =
(System.Windows.Forms.TrackBar)SourceControl;
 trackBar.Value = newValue;
}
```

3 プロジェクトを作成します。

TrackBarSrv.cs ファイルの確認

次に，TrackBarSrv クラスのソース・コード全体を示します。作成された **TrackBarSrv.cs** ファイルと比較し，同様の内容であることを確認してください。

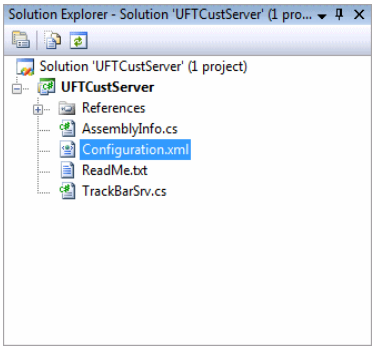
```
using System;
using Mercury.QTP.CustomServer;
namespace UFTCustServer
{
    [ReplayInterface]
    public interface ITrackBarSrvReplay
    {
        void SetValue(int newValue);
    }
    public class TrackBarSrv:
        CustomServerBase,
        ITrackBarSrvReplay
    {
        public TrackBarSrv()
        {
        }
        public override void InitEventListener()
        {
            Delegate  e = new System.EventHandler(this.OnValueChanged);
            AddHandler("ValueChanged", e);
        }
        public override void ReleaseEventListener()
        {
        }
        public void OnValueChanged(object sender, EventArgs e)
        {
            System.Windows.Forms.TrackBar trackBar =
                                (System.Windows.Forms.TrackBar)sender;
            int newValue = trackBar.Value;
            RecordFunction("SetValue",
                                RecordingMode.RECORD_SEND_LINE,
                                newValue);
        }
        public void SetValue(int newValue)
        {
            System.Windows.Forms.TrackBar trackBar =
                                (System.Windows.Forms.TrackBar)SourceControl;
            trackBar.Value = newValue;
        }
    }
}
```

サポート・セットの設定とデプロイ

UFT カスタム・サーバ の作成が完了したので，次に，**TrackBar** コントロールで GUI テストを記録および実行する際に，作成したカスタム・サーバを使用できるように UFT を設定します。

カスタム・サーバ の使用に必要な UFT の設定は，次の手順で行います。

1 ［ソリューション エクスプローラ］ウィンドウで **Configuration.XML** ファイルをダブルクリックします。



次のような内容が表示されます。

```
<!-- Merge this XML content into file "<UFT installation folder>\dat\
SwfConfig.xml".-->
<Control Type="System.Windows.Forms.TrackBar">
        <CustomRecord>
            <Component>
                <Context>AUT</Context>
                <DllName>D:\Projects\UFTCustServer\Bin\UFTCustServer.dll
                    </DllName>
                <TypeName>UFTCustServer.TrackBarSrv</TypeName>
            </Component>
        </CustomRecord>
        <CustomReplay>
            <Component>
                <Context>AUT</Context>
                <DllName>D:\Projects\UFTCustServer\Bin\UFTCustServer.dll
                    </DllName>
                <TypeName>UFTCustServer.TrackBarSrv</TypeName>
            </Component>
        </CustomReplay>
        <!--<Settings>
             <Parameter Name="sample name">sample value</Parameter>
        </Settings> -->
</Control>
```

2 **<Control>から</Control>** セグメントを選択し，［**Edit**］＞［**Copy**］を選択します。

3 **<UFT installation folder>\dat** にある **SwfConfig.xml** ファイルを開きます。

4 **Configuration.xml** からコピーした **<Control>から</Control>** セグメントを，**SwfConfig.xml** の **<Controls>** タグに貼り付けます。貼り付け後の **SwfConfig.xml** ファイルは次のようになります。

```
<?xml version="1.0" encoding="UTF-8"?>
<Controls>
        <Control Type="System.Windows.Forms.TrackBar">
            <CustomRecord>
                <Component>
                    <Context>AUT</Context>
                    <DllName>D:\Projects\UFTCustServer\Bin\UFTCustServer.dll
                        </DllName>
                    <TypeName>UFTCustServer.TrackBarSrv</TypeName>
                </Component>
            </CustomRecord>
            <CustomReplay>
                <Component>
                    <Context>AUT</Context>
                    <DllName>D:\Projects\UFTCustServer\Bin\UFTCustServer.dll
                        </DllName>
                    <TypeName>UFTCustServer.TrackBarSrv</TypeName>
                </Component>
            </CustomReplay>
        </Control>
</Controls>
```

5 **<DllName>** 要素で，カスタム・サーバ DLL の正しいパスが指定されていることを確認してください。

6 **SwfConfig.xml** ファイルを保存します。

サポート・セットのテスト

これで，カスタム・サーバをテストする準備が整いました。カスタマイズした TrackBar コントロール上で UFT が GUI テストを記録し，想定どおりに動作することを確認できます。

カスタム・サーバ をテストするには，次の手順で行います。

1. UFT を開いて .NET Add-in をロードし，GUI テストを開きます。
2. **System.Windows.Forms.TrackBar** コントロールを使用して .NET アプリケーションでの記録を開始します。
3. **TrackBar** コントロールをクリックします。UFT では，次のようなコマンドが記録されます。

```
SwfWindow("Form1").SwfObject("trackBar1").SetValue 2
```

4. テストを実行します。TrackBar コントロールが正しい値を取得していることを確認します。

第 7 章

複雑な .NET Windows Forms カスタム・コントロールの作成の実習

このチュートリアルでは，複雑な実装ソリューションを必要とするコントロール用にカスタム・サーバを作成し，コントロールで UFT が操作の記録と実行を実行できるようにします。カスタム・サーバの実装は C# で行いますが，Visual Basic でも同様の方法で実装できます。

本章の内容は，.NET Add-in Extensibility の概念に関する知識があり，カスタム・サーバの実装方法を理解している方を対象とします。

本章の内容

SandBar ツールバーの例

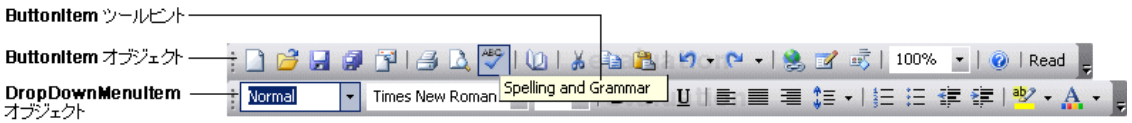
ここでは，Divelements Limited の **TD.SandBar.Toolbar** コントロール用に .NET Add-in Extensibility を実装する方法を実習します。

ToolBarSrv.cs クラス実装の最終的なソース・コードは，122 ページ「ToolBarSrv.cs ファイルについて」で参照できます。

SandBar コントロールの最終的なサポート・セットは C# と Visual Basic で実装され，**<UFT .NET Add-in Extensibility SDK インストール・フォルダ>\samples\WinFormsExtSample** に収録されています。このチュートリアルでの実習では，このファイルを参考にしてください。詳細については，86 ページ「サンプルの .NET Add-in Extensibility の使用方法」を参照してください。

ヒント：TD.SandBar.Toolbar コントロールの確認用コピーは，http://www.divil.co.uk/net/download.aspx?product=2&license=5 からダウンロードできます。

次の図は，**Toolbar** コントロールを示します。



Toolbar コントロールは，次のような各種オブジェクトで構成されます。

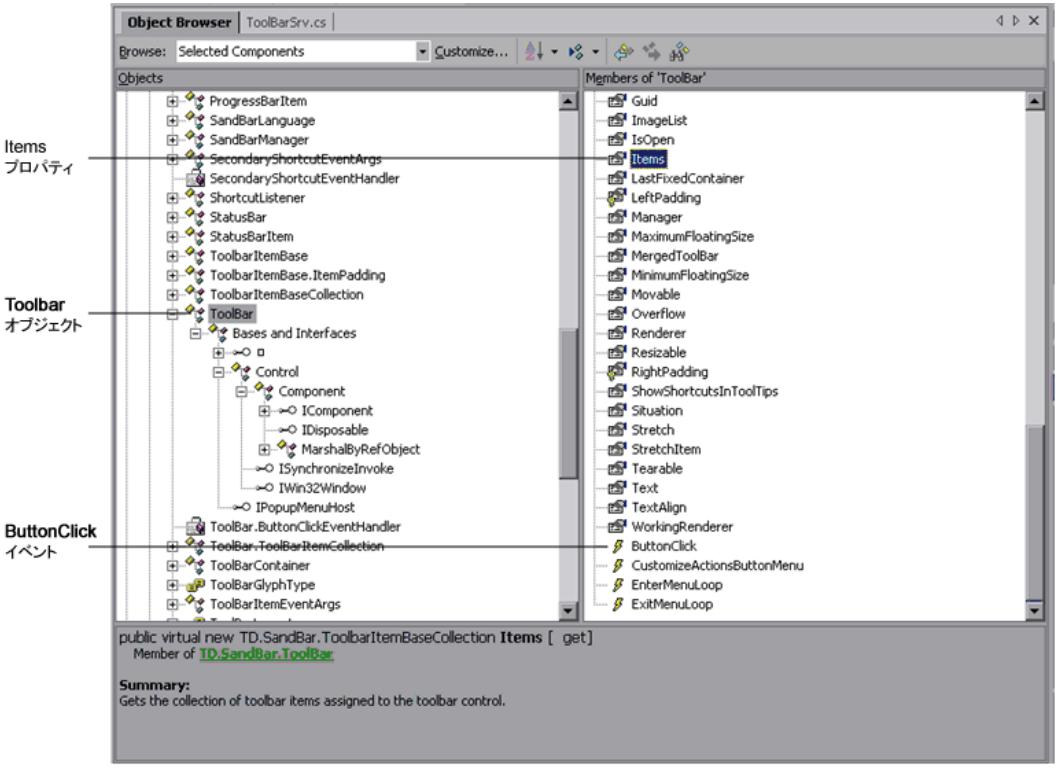
- **ButtonItem** オブジェクト。ツールバーのボタンです。**ButtonItem** オブジェクト。画像だけでテキストはありません。**ButtonItem** オブジェクトにはそれぞれツールヒントが表示されます。
- **DropDownMenuItem** オブジェクト。ツールバーのドロップダウン・メニューです。

ButtonItem オブジェクトと **DropDownMenuItem** オブジェクトはいずれも **ToolbarItemBase** オブジェクトから派生しています。

カスタム・コントロール用にカスタム・サーバを実装することにより，UFT はカスタム・コントロールでユーザ・アクションの記録と実行を行います。テストの記録では，カスタム・サーバはコントロールのイベントをリッスンし，イベントを処理して所定のアクションを実行して，ステップを UFT GUI テストに追加します。テスト実行では，そのコントロールでユーザが実行したとおりにアクションをシミュレーション（再生）します。

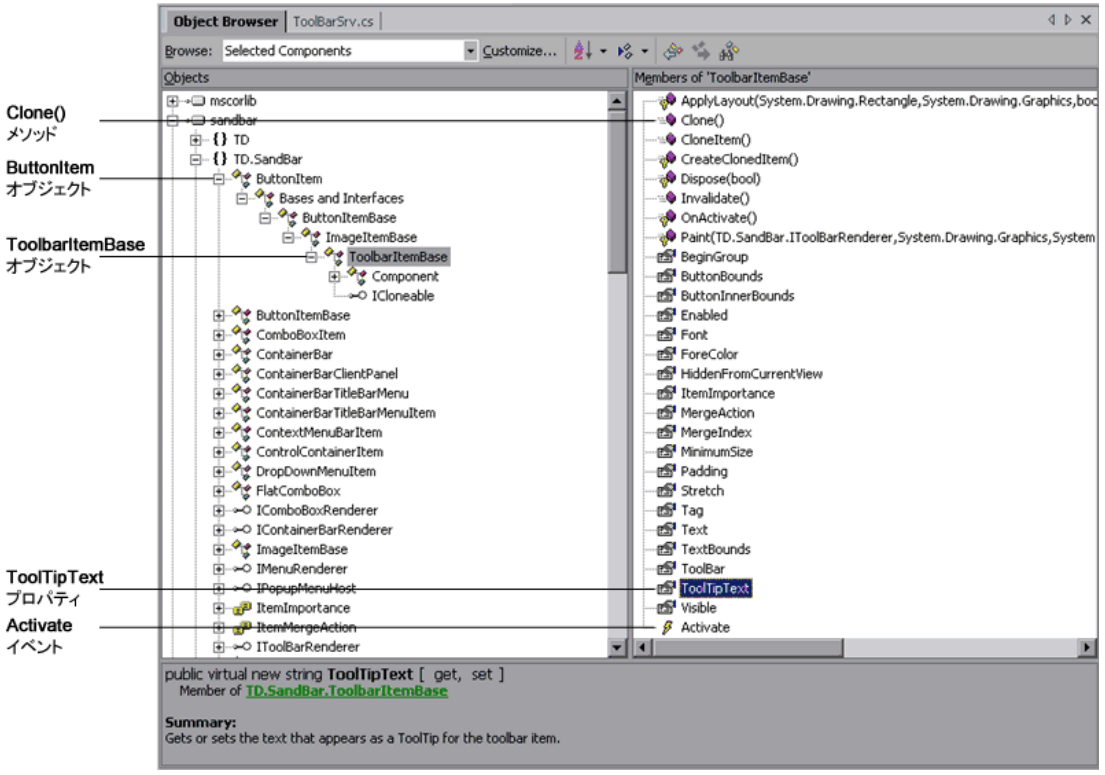
この例では，カスタム・ツールバーのボタンを押すという操作を行うユーザを実装します。作業を始める前に，ツールバー・コントロール，プロパティ，メソッド，それぞれのカスタム・サーバの実装での使用方法についてよく理解しておいてください。

次の図は，Visual Studio のオブジェクト・ブラウザの画面です。SandBar **ToolBar** オブジェクトのプロパティとイベントが表示されています（画像ではメソッドは表示されてません）。



上の図では，**ToolBar** オブジェクトに **Items** という名前のプロパティがあることがわかります。このプロパティは，**ToolBar** コントロールに割り当てられている **ToolbarItemBase** オブジェクトを取得します。また，**ToolBar** コントロールには **ButtonClick** という名前のイベントもあります。カスタム・サーバは **ButtonClick** イベントをリッスンすることにより，ツールバー・ボタンのクリックを認識します。ただしこのイベントでは，特定のツールバー・ボタンのクリックを識別しません。

ButtonItem オブジェクトを展開し，プロパティ，メソッド，イベントを詳しくみてみましょう。

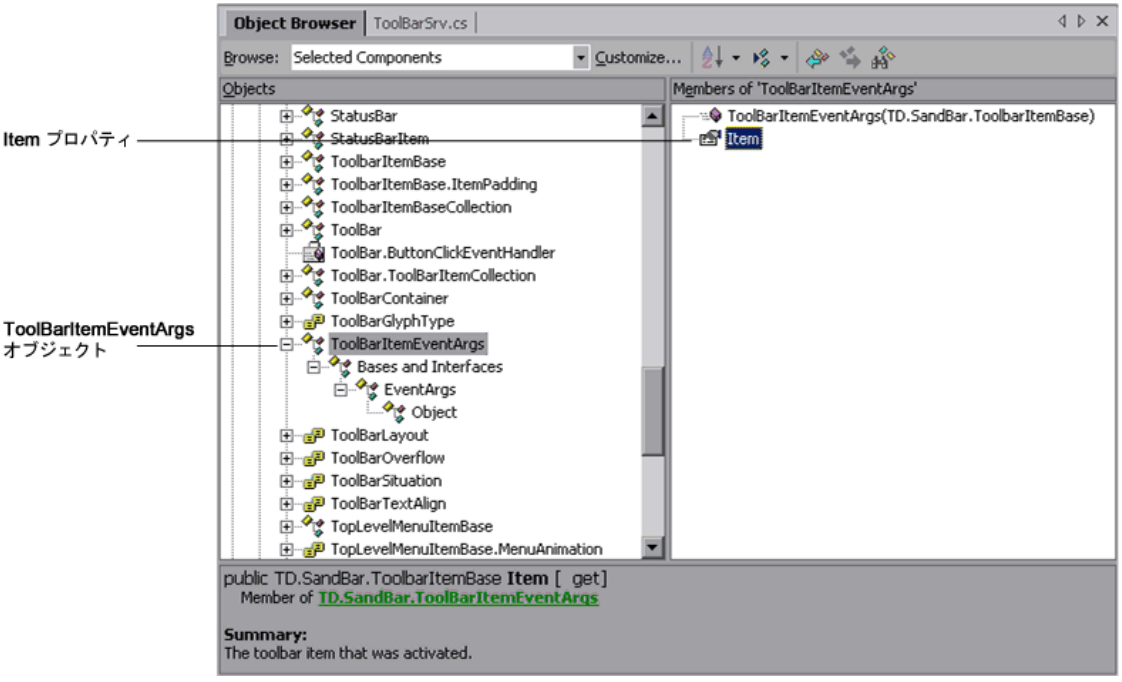


上の図から，**ButtonItem** オブジェクトは **ToolbarItemBase** オブジェクトから派生していることがわかります。**ToolbarItemBase** オブジェクトには **ToolTipText** プロパティがありますが，**Click** イベントやメソッドはありません。

このカスタム・ツールバー・オブジェクトでは，実装で次のような問題が発生する可能性があります。

1 記録中に ButtonClick イベントを処理する場合，ツールバー・ボタンのクリックをどのように認識すればよいか？

解決策 : ToolBar オブジェクトのイベントはすべて **ToolBarItemEventArgs** イベントであり，これは **EventArgs** オブジェクトから派生しています。



Item プロパティは，どのツールバー項目（ボタン）がイベントの発生元なのかを示します。各ツールバー項目の **ToolTipText** プロパティを使用することにより，ボタンがクリックされたことを認識し，これを UFT GUI テストに追加できます。

手順としては，次のコードを **ToolBarSrv.cs** ファイルの **Record events handlers** セクションに追加します。

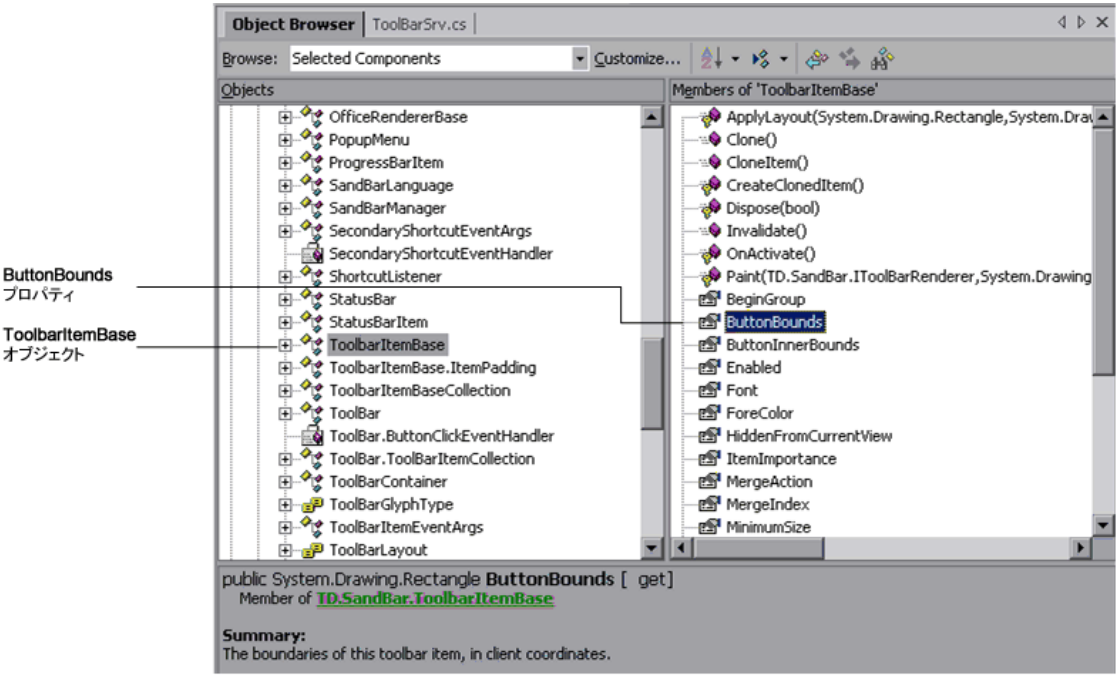
```
#region Record events handlers
private void oControl_ButtonClick(object sender, TD.SandBar.ToolBarItemEventArgs e)
{
    TD.SandBar.ToolBar  oControl = (TD.SandBar.ToolBar)SourceControl;

    //Add a step in the test for the test object with the ClickButton method and the
        tooltip text as an argument
    base.RecordFunction("ClickButton",
        RecordingMode.RECORD_SEND_LINE, e.Item.ToolTipText);
}
#endregion
```

ツールバー・ボタンのクリックを記録するたびに，**ClickButton** メソッドとボタンのツールヒント・テキスト（引数）を使用して，ツールバー・テスト・オブジェクトのテストにステップが追加されます。次に例を示します。

```
SwfToolbar("MySandBar").ClickButton "Spelling and Grammar"
```

2 テスト実行時中，ButtonItem オブジェクトに Click メソッドまたはイベントがなく，ButtonItem オブジェクトのツールヒント・テキストのみがわかっている場合，ClickButton メソッドをどのような方法で実行すればよいか？

解決策：ToolbarItemBase オブジェクトには **ButtonBounds** という名前のプロパティがあります。



すべての **ToolbarItemBase** オブジェクトをループして，**ButtonItem** オブジェクトとツールヒント・テキストが同じ **ToolbarItemBase** オブジェクトを検出します。その **ToolbarItemBase** オブジェクトの四角形の境界線を検出し，境界線の中心を計算してそのポイントをクリックします。

手順としては，次のコードを **ToolBarSrv.cs** ファイルの **Replay interface implementation** セクションに追加します。

```
#region Replay interface implementation
public void  ClickButton(string  text)
{
TD.SandBar.ToolBar  oControl = (TD.SandBar.ToolBar)SourceControl;

    //Find the correct item in the toolbar according to its tooltip text.
    for(int i=0; i<oControl.Items.Count; i++)
    {
        //Found the correct ButtonItem
        if(oControl.Items[i].ToolTipText == text)
        {
            //Retrieve the rectangle of the button's boundaries and locate its center
            System.Drawing.Rectangle oRect = oControl.Items[i].ButtonBounds;
            int x = oRect.X + oRect.Width/2;
            int y = oRect.Y + oRect.Height/2;

            //Click the middle of the button item
            base.MouseClick(x, y, MOUSE_BUTTON.LEFT_MOUSE_BUTTON);
            break;
        }
    }

    //Add the step to the report
    base.ReplayReportStep("ClickButton", EventStatus.EVENTSTATUS_GENERAL, text);
}
#endregion
```

ToolBarSrv.cs ファイルについて

次に，**ToolBarSrv.cs** クラスのソース・コード全体を示します。これを使用して UFT レコードを実装し，**TD.SandBar.Toolbar** コントロールのサポートを実行します。

```
using System;
using Mercury.QTP.CustomServer;
//using TD.SandBar;

namespace ToolBar
{
    [ReplayInterface]
```

```
public interface IToolBarSrvReplay
{
    void  ClickButton(string  text);
}
/// <summary>
/// Summary description for ToolBarSrv.
/// </summary>
public class ToolBarSrv:
    CustomServerBase,
    IToolBarSrvReplay
{
    // You shouldn't call Base class methods/properties at the constructor
    // since its services are not initialized yet.
    public ToolBarSrv()
    {
        //
        // TODO:Add constructor logic here
        //
    }
    #region IRecord override Methods
    #region Wizard generated sample code (commented)
    /// <summary>
    /// To change Window messages filter, implement this method.
    /// The default implementation is to get only the control's
    /// Window messages.
    /// </summary>
    public override WND_MsgFilter GetWndMessageFilter()
    {
        return(WND_MsgFilter.WND_MSGS);
    }

    /*
    /// <summary>
    /// To catch Window messages, you should implement this method.
    /// Note that this method is called only if the CustomServer is running
    /// under UFT process.
    /// </summary>
    public override RecordStatus OnMessage(ref Message tMsg)
    {
        // TODO:Add OnMessage implementation.
        return RecordStatus.RECORD_HANDLED;
    }
    */
    #endregion

    /// <summary>
```

```
        /// If you are extending the Record process, you should add your event
        /// handlers to listen to the control's events.
        /// </summary>
        public override void InitEventListener()
        {
            TD.SandBar.ToolBar  oControl =
(TD.SandBar.ToolBar)SourceControl;
            oControl.ButtonClick += new

  TD.SandBar.ToolBar.ButtonClickEventHandler(oControl_ButtonClick);
            //AddHandler("ButtonClick", new
            //
TD.SandBar.ToolBar.ButtonClickEventHandler(oControl_ButtonClick));
        }

        /// <summary>
        /// At the end of the Record process, this method is called by UFT to
        /// release all the handlers the user added in the InitEventListener method.
        /// Note that handlers added via UFT methods are released by
        /// the UFT infrastructure.
        /// </summary>
        public override void ReleaseEventListener()
        {
            TD.SandBar.ToolBar  oControl = (TD.SandBar.ToolBar)SourceControl;
            oControl.ButtonClick -= new
            TD.SandBar.ToolBar.ButtonClickEventHandler(oControl_ButtonClick);
        }
        #endregion

        #region Record events handlers
        private void oControl_ButtonClick(object sender,
                                TD.SandBar.ToolBarItemEventArgs e)
        {
            TD.SandBar.ToolBar  oControl = (TD.SandBar.ToolBar)SourceControl;
            // Add a step in the test for the test object with the ClickButton method
            // and the tooltip text as an argument
            base.RecordFunction("ClickButton",
                    RecordingMode.RECORD_SEND_LINE, e.Item.ToolTipText);
        }
        #endregion
        #region Replay interface implementation
        public void  ClickButton(string text)
        {
        TD.SandBar.ToolBar  oControl = (TD.SandBar.ToolBar)SourceControl;
        //Find the correct item in the toolbar according to its tooltip text.
        for(int i=0; i<oControl.Items.Count; i++)
```

```
        {
            //Found the correct ButtonItem
            if(oControl.Items[i].ToolTipText == text)
            {
            // Retrieve the rectangle of the button's boundaries and
            // locate its center
            System.Drawing.Rectangle oRect=oControl.Items[i].ButtonBounds;
            int x = oRect.X + oRect.Width/2;
            int y = oRect.Y + oRect.Height/2;
            //Click the middle of the button item
            base.MouseClick(x, y, MOUSE_BUTTON.LEFT_MOUSE_BUTTON);
            break;
            }
        }
        //Add the step to the report
        base.ReplayReportStep("ClickButton",
                                    EventStatus.EVENTSTATUS_GENERAL, text);
        }
        #endregion
    }
}
```