# HP UFT WPF および Silverlight 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。

UnixR®は、The Open Groupの登録商標です。

# ドキュメントの更新情報

このマニュアルの表紙には、以下の識別情報が記載されています。

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

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

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

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

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

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

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

# サポート

次のHPソフトウェアサポートのWebサイトを参照してください。

**http://support.openview.hp.com**

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

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

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

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

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

アクセスレベルの詳細については、次のWebサイトをご覧ください。

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

# 目次

# WPF および Silverlight Add-in Extensibility へようこそ

HP UFT WPF および Silverlight Add-in Extensibility は，Unified Functional Testing WPF Add-in および Silverlight 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 WPF または Silverlight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。

WPF と Silverlight のコントロールのサポートはそれぞれ別に開発し，異なる API を使用する必要があります。ただし，Silverlight サポートは WPF サポートと類似した方法で作成できるので，このガイドでは両方の手順を併せて説明しています。

**本章の内容**

# UFT WPF および Silverlight Add-in Extensibility SDK について

UFT WPF および Silverlight Add-in Extensibility SDK では次の内容が提供されます。

- Unified Functional Testing WPF Add-in または Silverlight Add-in を拡張する API。これにより，WPF または Silverlight のカスタム・コントロールをサポートします。
- Microsoft Visual Studio 用のWPF および Silverlight Custom Server C# 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。

- Visual Studio のカスタム・サーバの設定ダイアログ・ボックスでは，カスタム・サーバ・テンプレートを使用して作成するプロジェクトをカスタマイズできます。
- このヘルプ（[**スタート**] > [**すべてのプログラム**] > [**HP Software**] > [**HP Unified Functional Testing**] > [**Extensibility**] > [**Documentation**] でアクセス）。次の内容が含まれます。
  - 開発者ガイド。詳しい手順を説明するチュートリアル形式で，サンプルのカスタム・コントロールのサポートを開発します。
  - API Reference。
  - ツールキット Configuration Schema ヘルプ
  - Unified Functional Testing Test Object Schema ヘルプ
- 印刷用（PDF）バージョンの開発者ガイド（[**スタート**] > [**すべてのプログラム**] > [**HP Software**] > [**HP Unified Functional Testing**] > [**Extensibility**] > [**Documentation**] からアクセス，または **<Unified Functional Testing インストール・ディレクトリ>\help\Extensibility** フォルダに収録）。

➤ サンプルの WPF 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 **Microsoft.Windows.Controls.Calendar** 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，WPF または Silverlight 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 WPF または Silverlight 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，WPF および Silverlight Add-in Extensibility ヘルプ（[**スタート**] > [**すべてのプログラム**] > [**HP Software**] > [**HP Unified Functional Testing**] > [**Extensibility**] > [**Documentation**] > [**WPF および Silverlight Add-in Extensibility Help**]）で提供されている『API Reference』,『Toolkit Configuration Schema のヘルプ』,『UFT Test Object Schema のヘルプ』も併せて参照してください。

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

---

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

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

---

## 対象読者

このガイドは，プログラマ，QA エンジニア，システム・アナリスト，システム・デザイナ，テクニカル・マネージャを対象に，UFT GUI テストを拡張することによって WPF または Silverlight 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 WPF または Silverlight Add-in
- C# での WPF または Silverlight 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

# WPF または Silverlight のカスタム・ツールキットに対する UFT サポートの開発

WPF（Windows Presentation Foundation）および Silverlight のカスタマ・アプリケーションの多くは，その UI に Microsoft 以外のコントロールを使用しています。UFT では，これらのコントロールをそれぞれ汎用の WpfObject または SlvObject テスト・オブジェクトで表します。

複雑なコントロールは，機能的な意味を持つ上位レベルのコントロールではなく，下位レベルのコントロールの組み合わせとして認識されます。たとえば，WPF または Silverlight のカスタム・カレンダ・コントロールは，関連性のないいくつかのボタンとテキスト・ボックスとして認識される可能性があります。

このような場合，テスト担当者は，カスタム・コントロール・タイプに固有のロジックを含むメソッドを実行できません。また，そのコントロール・タイプに固有の記録ロジックを適用することもできません。

このようなカスタム・コントロールを表すための新しいテスト・オブジェクト・クラスは，WPF または Silverlight Add-in Extensibility を使用し，ツールキット・サポート・セットを作成して定義します。このサポート・セットにより，WPF または Silverlight のカスタム・コントロールを QA エンジニアが実行し，記録し，学習し，調査できるようになります。ツールキット・サポート・セットは，WPF と Silverlight のコントロール用に別々に作成し，異なる API を使用する必要があります。

**本章の内容**

➤ WPF または Silverlight のカスタム・ツールキットのサポートを作成する方法（24ページ）

➤ WPF または Silverlight のカスタム・コントロールのサポートを追加する方法（29ページ）

➤ カスタム・サーバの開発方法（31ページ）

➤［WPF/Silverlight Custom Server Setup］ダイアログ・ボックス（Microsoft Visual Studio）（39ページ）

➤ トラブルシューティングと制限事項 - サポートの開発（45ページ）

## WPF または Silverlight Add-in Extensibility ツールキット・サポート・セットの開発について

WPF および Silverlight Add-in Extensibility は，サポートされているバージョンの Microsoft Visual Studio を使用して C# で実装します（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，または Unified Functional Testing DVD のルート・フォルダに収録されています）。

➤ Visual Studio は，サポート・セットの開発にのみ必要です。使用する際は必要ありません。

➤ Silverlight Add-in Extensibility を開発するには，Microsoft Silverlight Tools for Visual Studio がインストールされている必要があります。

➤ UFT は，サポート・セットを実行しテストするときにのみ必要です。開発時には必要ありません。

ツールキット（環境）は, 1 つのパッケージでサポートを提供する一連のコントロールです。

ツールキット・サポート・セットは，次の要素で構成されます。

➤ **テスト・オブジェクト設定 XML ファイル :** このファイルで，新しいテスト・オブジェクト・タイプが定義されます。詳細については，15 ページ「テスト・オブジェクト設定 XML ファイル」を参照してください。

➤ **ツールキット設定ファイル :** このファイルで，WPF または Silverlight のコントロール・タイプが，テスト・オブジェクト・タイプ（クラス）にマッピングされ，記録および実行のロジックを実装するカスタム・サーバにマッピングされます。このファイルの構造と構文の詳細については，「ツールキット設定スキーマ」（WPF および Silverlight Add-in Extensibility ヘルプで利用可能）を参照してください。

➤ **カスタム・サーバの実装を含む .NET DLL**：詳細については，21 ページ「カスタム・サーバ」を参照してください。

➤ **アイコンおよびヘルプ・ファイル（オプション）**：

アイコン・ファイルには，テスト・オブジェクト・クラスを表すために UFT で使用されるアイコンが保存されています（サポートされるファイル・タイプ：**.ico**, **.exe**, **.dll**）。

ヘルプ・ファイルは，テスト・オブジェクト・クラスおよびそのメソッドとプロパティに関するコンテキスト・ヘルプに使用されます（サポートされるファイル・タイプ：**.chm**）。

## テスト・オブジェクト設定 XML ファイル

カスタム・ツールキットのサポートを開発する第 1 段階では，UFT がアプリケーション・コントロールを表す際に使用するテスト・オブジェクト・クラスを定義します。テスト・オブジェクト・クラスは，テスト・オブジェクト設定 XML ファイルで定義します。UFT サポートを拡張または変更するカスタム・コントロールのすべてのタイプに対して，テスト・オブジェクト・クラスを作成する必要があります。

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

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

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

テスト・オブジェクト設定ファイルの構造が適切かどうかは，**ClassesDefintions.xsd** ファイルを使ってチェックできます。このファイルは，UFT のインストール時に **<UFT インストール・フォルダ>\dat** フォルダに格納され，（下位互換性を維持するために，UFT がサポートする XML 構造には，この XSD によるチェック条件を満たしていないものも含まれています）。

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

### クラス名と基本クラス

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

標準設定では，基本クラスは WpfObject または SlvObject です。

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

---

**注：**

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

---

### 汎用タイプ

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

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

### テスト・オブジェクト操作

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

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

### 標準設定操作

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

### 認識プロパティ

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

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

### アイコン・ファイル

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

### ヘルプ・ファイル

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

テスト・オブジェクト設定ファイルの構造と構文の詳細については，Unified Functional Testing Test Object Schema ヘルプ（WPF および Silverlight Add-in Extensibility ヘルプで利用可能）を参照してください。

### テスト・オブジェクト設定ファイルの例

WPF Add-in Extensibility テスト・オブジェクト設定ファイルの例を下記に示します。Silverlight Add-in Extensibility テスト・オブジェクト設定ファイルでは，**TypeInformation** 要素の **AddinName** 属性を Silverlight に設定する必要があります。

また，**PackageName** 属性で指定するツールキット名は，それが WPF および Silverlight のどちらのツールキットかがわかるようにしてください。そうしておくのは，UFT のアドイン・マネージャには，WPF と Silverlight Add-in Extensibility でサポートされる環境が，両方とも WPF アドイン・ノード下の子ノードとして表示されるためです。

```
<TypeInformation Load="true" AddinName="WPF" PackageName="MyWpfToolkit">
    <ClassInfo Name="MyWpfButton" BaseClassInfoName="WpfButton"
            ROTypeInfo="false"
            GenericTypeID="button"
            DefaultOperationName="Click"
            FilterLevel="0">
        <IconInfo
            IconFile="INSTALLDIR\dat\Extensibility\WPF\MyWpfButton_icon.ico"/>
        <TypeInfo>
            <Operation Name="Click">
                <Argument Name="X" IsMandatory="false"
                        DefaultValue="-9999" Direction="In">
                    <Type VariantType="Integer"/>
                </Argument>
                <Argument Name="Y" IsMandatory="false"
                        DefaultValue="-9999" Direction="In">
                    <Type VariantType="Integer"/>
                </Argument>
                <Argument Name="MouseButton" IsMandatory="false"
                        DefaultValue="0" Direction="In">
                    <Type VariantType="Enumeration"
                        ListOfValuesName="E_ButtonType"/>
                </Argument>
            </Operation>
        </TypeInfo>
        <IdentificationProperties>
            <IdentificationProperty Name="devname" ForDescription="true"/>
            <IdentificationProperty Name="enabled" ForDescription="false"
                                    ForVerification="true"/>
        </IdentificationProperties>
    </ClassInfo>
</TypeInformation>
```

# カスタム・サーバ

サポートするカスタム・コントロールごとに，カスタム・サーバ・クラスを **CustomServerBase** クラスから派生して作成します。得られたカスタム・サーバ DLL は，アプリケーションのコンテキストで実行され，UFT とカスタム・コントロールとの間のインタフェースになります。この DLL は，UFT の要求に応じて，コントロールからの認識プロパティ値の取得，コントロールに対する操作の実行，コントロール上のユーザ・アクティビティに対して記録するステップの決定などを処理できます。複数のカスタム・サーバをコンパイルして，1 つの DLL にすることができます。

これらの機能は，それぞれ関連するインタフェースをカスタム・サーバ・クラスに実装することで実装します。インタフェースのメソッドとその構文の詳細については，『Custom Server API Reference』（WPF および Silverlight Add-in Extensibility ヘルプで利用可能）を参照してください。

- テスト・オブジェクト操作の実行をサポートするために，実行インタフェースを開発し，そのインタフェースにテスト・オブジェクト設定ファイルで定義した操作を実行するメソッドを含めます。このインタフェースには，**RunInterfaceAttribute** 属性を付ける必要があります。

  Silverlight コントロールのサポートを開発する場合は，テスト・オブジェクト操作の実行を実装するために作成する各メソッドに Microsoft Silverlight **ScriptableMember** 属性を付ける必要があります。

- コントロールからの認識プロパティ値の取得をサポートするために，プロパティ・インタフェースを開発し，そのインタフェースにテスト・オブジェクト設定ファイルで定義した認識プロパティの値を取得するプロパティを含めます。このインタフェースには，**CustomPropInterfaceAttribute** 属性を付ける必要があります。

---

**注 :** テスト・オブジェクト設定ファイルには，テスト・オブジェクト・クラスに関係するすべての認識プロパティを定義する必要があります。サポートされるすべてのプロパティに対して，プロパティ値を取得する実装は，基本クラスから継承されます。

---

- 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ために，**ITableVerify** インタフェースにメソッドを実装します。
- 記録をサポートするために，**IRecord** インタフェースを実装します。
- コントロールの子を無視するよう UFT に指示するため，**IComponentDetector** インタフェースの **IsKnownPartOf** メソッドを，子コントロールに対して true を返すように設計します（コントロールの子は，機能的にはコントロールの一部であり，子自体は独立したコントロールではありません）。

タスクの詳細については，31 ページ「カスタム・サーバの開発方法」を参照してください。

カスタム・サーバを作成するときは，WPF または Silverlight Add-in Extensibility API で提供されるユーティリティ・メソッドおよびプロパティを使用できます。

## ユーティリティ・メソッドおよびプロパティ

カスタム・サーバによって拡張される **CustomServerBase** クラスには，**UtilityObject** プロパティが含まれており，ユーティリティ・メソッドおよびプロパティを提供するオブジェクトを返すことができます。カスタム・サーバの実装では，次のメソッドとプロパティを呼び出すことができます。

- **AddHandler。**コントロールでイベントが発生したときに使用するイベント・ハンドラを登録します。ハンドラは，このイベントのイベント・ハンドラ呼び出しリストの先頭に追加されます。
- マウスおよびキーボード操作のシミュレーション・メソッド。これらのメソッドは，コントロールに関するステップを実行するメソッドで使用します。
- **GetSettingsValue**，**GetSettingsXML。**ツールキット設定ファイルでこのカスタム・サーバに対して定義されている設定を取得します。
- **Record。**テストにステップを追加し，テスト・オブジェクトがオブジェクト・リポジトリにない場合はそこに追加します。このメソッドは，コントロールでイベントが発生した後でテストのステップを記録するイベント・ハンドラで使用します。
- **ReportStepResult。**ステップの結果に関する情報を実行結果に追加します。このメソッドは，コントロールに関するステップを実行するメソッドで使用します。
- **ThrowRunError。**指定されたエラーに基づいて例外を発生させ，ステップのステータスを **EventStatus.EVENTSTATUS_FAIL** に設定します。

➤ **ApplicationObject。**このプロパティは，カスタム・サーバが関連付けられているコントロール・オブジェクトを返します。たとえば，チェックボックスに関連付けられた **CheckBoxCustomServer** の場合，このプロパティは，そのチェックボックスへの参照を返します。この参照を使用して，情報（**IsChecked** プロパティの値など）を取得したり，コントロールに対するアクティビティ（**IsChecked** プロパティを true に設定など）を実行したりすることができます。

詳細については,『Custom Server API Reference』(WPF および Silverlight Add-in Extensibility ヘルプで利用可能）の **Mercury.P.WPF.CustomServer** または **Mercury.QTP.Slv.CustomServer** 名前空間の項の **IUtilityObject** インタフェースを参照してください。

## WPF Add-in Extensibility のサンプル

WPF Add-in Extensibility SDK をインストールすると，WPF のカスタム・カレンダ・コントロールと，そのコントロールのサポートを拡張するサンプル・ツールキット・サポート・セットが，**<WPF Add-in Extensibility SDK インストール・フォルダ>\samples\WPFExtCalendarSample** フォルダにインストールされます。このサンプルについて詳しく調べることで，WPF および Silverlight Add-in Extensibility の実装方法について理解を深めることができます。サンプルのツールキット・サポート・セットをデプロイする前後にコントロールを UFT でテストして，このサンプルを試すこともできます。

サンプルのツールキット・サポート・セットをデプロイするには，提供された XML および DLL ファイルを，80 ページ「カスタム・ツールキット・サポートのデプロイ」の説明に従って，UFT コンピュータ上の正しい場所に配置します。

デプロイするファイルは，次のとおりです。

➤ **<WPF Add-in Extensibility SDK インストール・フォルダ>\samples\WPFExtCalendarSample\Support\QtCalendarSrv\MyWpfToolkit.cfg**

➤ **<WPF Add-in Extensibility SDK インストール・フォルダ>\samples\WPFExtCalendarSample\Support\QtCalendarSrv\MyWpfToolkitTestObjects.xml**

➤ **<WPF Add-in Extensibility SDK インストール・フォルダ>\samples\WPFExtCalendarSample\Support\QtCalendarSrv\bin\Release\QtCalendarSrv.dll**

# WPF または Silverlight のカスタム・ツールキットのサポートを作成する方法

このタスクでは，ツールキット・サポート・セットの作成方法，デプロイ方法，テスト方法について説明します。このツールキット・サポート・セットにより，WPF または Silverlight の一連のカスタム・コントロールに対する UFT のサポートが拡張されます。

サポートは，WPF コントロールと Silverlight コントロールで別々に作成する必要があります。ただし，サポートの作成方法は Silverlight コントロールと WPF コントロールで類似しているため，このタスクでは両方の手順をまとめて説明しています。

---

**ヒント：**

- まず，機能的な変更点を最小限にとどめて，テスト・オブジェクト・クラスが 1 つの基本的なツールキット・サポート・セットを作成し，そのセットが UFT で正しく認識されることをテストします。次に，複雑なサポートとテスト・オブジェクト・クラスを徐々に追加し，追加するたびにその内容をテストします。
- WPF または Silverlight Add-in Extensibility ファイルを作成するには，**UFT WPF CustomServer** または **UFT Silverlight CustomServer** プロジェクト・テンプレートを使用します（このテンプレートは，WPF および Silverlight Add-in Extensibility SDK によって Visual Studio にインストールされます）。

  このテンプレートを使用することで，開発の必要な XML ファイルとカスタム・サーバ・クラスがツールキット・サポート・セット内に設定され，以下で説明する作業の最初の 3 ステップが単純になります。詳細については，39 ページ「［WPF/Silverlight Custom Server Setup］ダイアログ・ボックス（Microsoft Visual Studio）」を参照してください。

---

このタスクには，次の手順が含まれています。


- 25 ページ「UFT がカスタム・コントロールに対して使用するテスト・オブジェクト・クラスの定義」
- 25 ページ「テスト・オブジェクト・クラスおよびカスタム・サーバへのカスタム・コントロールのマッピング」
- 27 ページ「サポートの実装を含むカスタム・サーバの設計」
- 27 ページ「UFT へのツールキット・サポート・セットのデプロイ」

➤ 27 ページ「開発したサポートの機能テスト」

➤ 28 ページ「サポートのデバッグ - オプション」

## 1 UFT がカスタム・コントロールに対して使用するテスト・オブジェクト・クラスの定義

**a** **<カスタム・ツールキット名>TestObjects.xml** というテスト・オブジェクト設定 XML ファイルを作成します。

この名前は，サポートのデプロイメントが完了した後で，アドイン・マネージャに WPF アドインの下の子アドインとして表示されます。したがって，これが WPF と Silverlight のどちらのツールキットかわかる名前にしてください。

**b** テスト・オブジェクト設定 XML ファイルで，新しいテスト・オブジェクト・クラスを定義します。

➤ WPF コントロールのサポートを作成する場合は，既存の UFT WPF テスト・オブジェクト・クラス（またはほかの WPF Add-in Extensibility テスト・オブジェクト・クラス）を拡張するテスト・オブジェクト・クラスを作成します。

➤ Silverlight コントロールのサポートを作成する場合は，既存の UFT Silverlight テスト・オブジェクト・クラス（またはほかの Silverlight Add-in Extensibility テスト・オブジェクト・クラス）を拡張するテスト・オブジェクト・クラスを作成します。

テスト・オブジェクト設定ファイルの構造と構文の詳細については，15 ページ「テスト・オブジェクト設定 XML ファイル」を参照してください。

## 2 テスト・オブジェクト・クラスおよびカスタム・サーバへのカスタム・コントロールのマッピング

ツールキット設定 XML ファイルで，カスタム・コントロール・タイプを，そのコントロール・タイプを UFT で表すテスト・オブジェクト・クラスにマッピングし，またサポートの実装を含めるカスタム・サーバにもマッピングします。サポートするコントロールのタイプごとに **Control** 要素を作成します。

ファイル名は **<カスタム・ツールキット名>.cfg** にします。

ツールキット設定ファイルの構造と構文の詳細については，『Toolkit Configuration Schema』（WPF および Silverlight Add-in Extensibility ヘルプで利用可能）を参照してください。

**WPF ツールキット設定ファイルの例：**

```
<?xml version="1.0" encoding="UTF-8"?>
<Controls>
    <Control Type="MyCompany.MyDataGrid" MappedTo="MyWpfTable">
        <CustomServer>
            <Component>
                <DllName>WpfCustomServers.dll</DllName>
                <TypeName>MyCompany.MyWpfDataGridCustServer</TypeName>
            </Component>
        </CustomServer>
    </Control>
</Controls>
```

**Silverlight ツールキット設定ファイルの例：**

```
<?xml version="1.0" encoding="UTF-8"?>
<Controls>
    <Control Type="MyCompany.MyDataGrid" MappedTo="MySlvTable">
        <CustomServer>
            <Component>
                <DllName>SlvCustomServers.dll</DllName>
                <TypeName>MyCompany.MySlvDataGridCustServer,
                            SlvCustomServers, Version=1.0.0.0, Culture=neutral,
                            PublicKeyToken=null </TypeName>
            </Component>
        </CustomServer>
    </Control>
</Controls>
```

### 3 サポートの実装を含むカスタム・サーバの設計

サポートするコントロールのタイプごとにカスタム・サーバを 1 つずつ設計します。

詳細については，31 ページ「カスタム・サーバの開発方法」を参照してください。

### 4 UFT へのツールキット・サポート・セットのデプロイ

**a** 開発が終了し，通常の用途にサポート・セットをデプロイする場合は，テスト・オブジェクト設定ファイルの **TypeInformation** 要素の **DevelopmentMode** 属性を必ず false に設定してください。

**b** 作成したファイルを UFT インストール・フォルダの下の適切な場所にコピーすることで，ツールキット・サポート・セットをデプロイします。詳細については，80 ページ「カスタム・ツールキット・サポートのデプロイ」を参照してください。

### 5 開発したサポートの機能テスト

**a** UFT を開きます。カスタム・ツールキットの名前が，［アドイン マネージャ］ダイアログ・ボックスに WPF アドインの子として表示されていることを確認し，それを選択します（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）。

---

**注 :** Silverlight を使用している場合は，Silverlight アドインも選択する必要があります。

---

**b** カスタム・コントロールに関する UFT テストを作成して実行し，コントロールと UFT のやり取りが期待どおりであることを検証します。次のことを確認します。

- ➤ オブジェクト・スパイでの表示時とオブジェクトの学習時に，期待どおりのテスト・オブジェクト・クラスでコントロールが表されている。
- ➤［新規テストオブジェクトの定義］ダイアログ・ボックスを使用して，定義したクラスのテスト・オブジェクトを正常に作成できる。

- キーワード・ビュー，エディタ（ステートメントの自動補完機能を使用），ステップ・ジェネレータで，定義したテスト・オブジェクト・クラスを使用してテスト・ステップを正しく作成できる。
- 操作が正しく実行される。CustomServerBase に基づくクラスが公開する実行サポート（Click，DblClick など）をチェックします。カスタム・サーバで実装されていない操作が，基本テスト・オブジェクト・タイプの標準設定の UFT 実装でサポートされていることを確認します。
- テーブル・チェックポイントおよび出力値が正しく機能する（該当する場合）。
- 認識プロパティ値が正しく取得される。カスタム・サーバで特に実装されていない認識プロパティが，基本クラスの実装でサポートされていることを確認します（該当する場合）。
- 記録の実装がウィンドウ・メッセージに基づく場合と，イベントに基づく場合の両方について，操作が正しく記録される。

  Silverlight コントロールのサポートを開発している場合は，カスタム・サーバによって定義されたウィンドウ・メッセージ・フィルタに基づいて，メッセージがサーバに正しく渡されることを確認します。
- ツールキットとそのテスト・オブジェクト・クラスが，関連する UFT ダイアログ・ボックス（［オブジェクトの認識］，［使用可能なキーワード］（アプリケーション領域の場合），［新規テストオブジェクトの定義］など）に適切に表示される。
- サポート・セットの一部ではない WPF または Silverlight の要素について，その機能がサポート・セットのインストールによって変更されていないことを確認します。
- 上位レベルのコントロールの一部である下位レベルのコントロール（**IComponentDetector** 実装で定義）が，記録中，学習中，スパイ中などに UFT で認識されないことを確認します。

## 6 サポートのデバッグ - オプション

開発したサポートを Microsoft Visual Studio デバッグ・ツールでデバッグするには, WPF または Silverlight のカスタム・コントロールを含むアプリケーションに Visual Studio をアタッチする必要があります（これは，そのアプリケーションのコンテキストでカスタム・サーバが実行されるためです）。

# WPF または Silverlight のカスタム・コントロールのサポートを追加する方法

このタスクでは，WPF または Silverlight のカスタム・コントロールの 1 つのタイプに対するサポートを，既存のツールキット・サポート・セットに追加する方法について説明します。

ツールキット・サポート・セットの作成手順については, 24 ページ「WPF または Silverlight のカスタム・ツールキットのサポートを作成する方法」を参照してください。

---

**ヒント**：WPF または Silverlight Add-in Extensibility ファイルを作成するには，**UFT WPF CustomServer** または **UFT Silverlight CustomServer** プロジェクト・テンプレートを使用します（このテンプレートは，WPF および Silverlight Add-in Extensibility SDK によって Visual Studio にインストールされます）。

このテンプレートを使用することで，カスタム・コントロールをサポートするために開発の必要な XML データとカスタム・サーバ・クラスが設定され，以下で説明する作業の最初の 3 ステップが単純になります。詳細については, 39 ページ「[WPF/Silverlight Custom Server Setup] ダイアログ・ボックス（Microsoft Visual Studio）」を参照してください。

必要であれば，テンプレートを使用して作成した XML データおよびカスタム・サーバ・クラスを，既存のツールキット・サポート・セットに移動できます。XML ファイルの情報を既存のツールキット・サポート・セットの XML ファイルにコピーし，カスタム・サーバの **.cs** ファイルを既存の Visual Studio の WPF または Silverlight Add-in Extensibility プロジェクトにコピーしてください。

---

このタスクには，次の手順が含まれています。

- 30 ページ「UFT がカスタム・コントロールに対して使用するテスト・オブジェクト・クラスの定義 - オプション」
- 30 ページ「関連するテスト・オブジェクト・クラスおよびカスタム・サーバへのカスタム・コントロールのマッピング」
- 30 ページ「UFT が実行する実装を含むカスタム・サーバの設計」
- 30 ページ「UFT でのツールキット・サポート・セットのデプロイとテスト」

### 1 UFT がカスタム・コントロールに対して使用するテスト・オブジェクト・クラスの定義 - オプション

ツールキット・サポート・セットに適切なテスト・オブジェクト・クラスが含まれていない場合は，テスト・オブジェクト設定 XML ファイルに **ClassInfo** 要素を追加して，新しいテスト・オブジェクト・クラスを定義します。

テスト・オブジェクト設定ファイルの構造と構文の詳細については，15 ページ「テスト・オブジェクト設定 XML ファイル」を参照してください。

### 2 関連するテスト・オブジェクト・クラスおよびカスタム・サーバへのカスタム・コントロールのマッピング

ツールキット設定 XML ファイルで，カスタム・コントロールの **Control** 要素を定義します。UFT がコントロールに対して使用するテスト・オブジェクト・クラスと，そのコントロールをサポートするための実装が含まれるカスタム・サーバを指定してください。

ツールキット設定ファイルの構造と構文の詳細については，『Toolkit Configuration Schema』（WPF および Silverlight Add-in Extensibility ヘルプで利用可能）を参照してください。

### 3 UFT が実行する実装を含むカスタム・サーバの設計

詳細については，31 ページ「カスタム・サーバの開発方法」を参照してください。

### 4 UFT でのツールキット・サポート・セットのデプロイとテスト

ツールキット・サポート・セット全体の一部に加えた変更をデプロイし，テストし，デバッグします。

詳細については，24 ページ「WPF または Silverlight のカスタム・ツールキットのサポートを作成する方法」を参照してください。

# カスタム・サーバの開発方法

このタスクでは，カスタム・サーバを作成し，UFT がカスタム・コントロールとやり取りする際に実行することが必要な実装をカスタム・サーバに含める方法について説明します。

このタスクは，より高いレベルのタスクの一部として実行されます。詳細については，24 ページ「WPF または Silverlight のカスタム・ツールキットのサポートを作成する方法」を参照してください。

このタスクで扱うインタフェース・メソッドの詳細については，『Custom Server API Reference』（WPF および Silverlight Add-in Extensibility ヘルプで利用可能）を参照してください。

このタスクには，次の手順が含まれています。

➤ 31 ページ「Visual Studio プロジェクトの設定」

➤ 33 ページ「カスタム・サーバ・クラスの作成」

➤ 34 ページ「テスト・オブジェクト操作のサポートの開発」

➤ 35 ページ「認識プロパティのサポートの開発」

➤ 36 ページ「テーブル・チェックポイントおよび出力値のサポートの開発」

➤ 37 ページ「別コントロールとして扱わない子コントロールの指定」

➤ 38 ページ「ステップ記録のサポートの開発」

➤ 38 ページ「カスタム・サーバをデプロイする準備」

## 1 Visual Studio プロジェクトの設定

次の手順のいずれかを実行します。

**Extensibility テンプレートを使用して，プロジェクトを設定します。**

Microsoft Visual Studio で, WPF および Silverlight Add-in Extensibility SDK によってインストールされた **UFT WPF CustomServer** または **UFT Silverlight CustomServer** プロジェクト・テンプレートを使用して，新しいプロジェクトを作成します。

これにより，カスタム・サーバの開発に必要なファイル，参照，クラスが設定され，このタスクの残りの手順が単純になります。詳細については，39 ページ「［WPF/Silverlight Custom Server Setup］ダイアログ・ボックス（Microsoft Visual Studio）」を参照してください。

**プロジェクトを手動で設定します。**

**a** Visual Studio で，［**Visual C#**］＞［**Windows**］＞［**クラス ライブラリ**］テンプレートまたは［**Visual C#**］＞［**Silverlight**］＞［**Silverlight クラス ライブラリ**］テンプレートを使用して，C# プロジェクトを作成します。

**b** 必要なすべての .NET フレームワーク・ライブラリへの参照を追加します。

たとえば，PresentationCore，PresentationFramework，WindowsBase（WPF での開発の場合）または .NET Framework Class Library for Silverlight（Silverlight での開発の場合）。

**c** カスタム・コントロール・クラスを実装するライブラリへの参照を追加します。これは，たとえば，サード・パーティ・ライブラリの場合があります。

**d** WPF または Silverlight Add-in Extensibility API を含む DLL ファイルへの参照を追加します。このファイルは，**<WPF および Silverlight Add-in Extensibility インストール・フォルダ>\SDK\WpfSlv** フォルダにあります。

- WPF コントロールのサポートを開発している場合は，**Mercury.QTP.WpfAgent.dll** ファイルへの参照を追加します。
- Silverlight コントロールのサポートを開発している場合は，**Mercury.QTP.Slv.CustomServer.dll** ファイルへの参照を追加します。

Extensibility SDK がインストールされていないコンピュータでツールキット・サポート・セットを開発する場合は，WPF および Silverlight Add-in Extensibility をインストールしたコンピュータから DLL をコピーします。

**e** 必要なすべての参照を Using セクションに追加します。

- WPF Add-in Extensibility API を参照するには，**Mercury.QTP.WPF.CustomServer** 名前空間への参照を追加します（**Mercury.WpfAgent** への参照ではありません）。
- Silverlight Add-in Extensibility API を参照するには，**Mercury.QTP.Slv.CustomServer** 名前空間への参照を追加します。

## 2 カスタム・サーバ・クラスの作成

カスタム・コントロールのカスタム・サーバ・クラスを作成して，**CustomServerBase** クラスを拡張します。

---

**注**：**UFT WPF/Silverlight CustomServer** テンプレートを使用して Visual Studio プロジェクトを設定した場合は，クラスの宣言が自動的に作成されます。

---

次の手順では，サポートする UFT 機能に応じて，さまざまなインタフェースをこのクラスに実装します。

たとえば，カスタム・サーバ・クラスの宣言は次のようになります。

```
public class MyCustomSupport:
    CustomServerBase,
    IMyCustomSupportRun,
    IRecord,
    ITableVerify,
    IMyCustomSupportCustProp,
    IComponentDetector
```

サポートを設計するときには，UFT によってこの基本クラスに実装された **IUtilityObject** インタフェースからユーティリティ・メソッドを呼び出すことができます。詳細については，22 ページ「ユーティリティ・メソッドおよびプロパティ」を参照してください。

### 3 テスト・オブジェクト操作のサポートの開発

**a** テスト・オブジェクト・クラス用に定義した新しいまたは修正したテスト・オブジェクト操作の実行をサポートするには，カスタム・サーバに 1 つの実行インタフェースを実装し，このインタフェースに **RunInterfaceAttribute** 属性を付けます。

---

**注 : UFT WPF/Silverlight CustomServer** テンプレートを使用して Visual Studio プロジェクトを設定し，実行操作をカスタマイズするよう指定していた場合は，実行インタフェースの定義が自動的に作成されます。

---

**b** 実行インタフェースでは，サポートまたはオーバーライドするテスト・オブジェクト操作ごとにメソッドを設計します。設計する各メソッドの署名は，それが実装するテスト・オブジェクト操作（テスト・オブジェクト設定ファイルで定義）と同じであることが必要です。

---

**注 :** テスト・オブジェクト設定ファイルで，オプションの引数を指定してテスト・オブジェクト操作を定義できます。
このようなテスト・オブジェクト操作の実行をサポートするカスタム・サーバ・メソッドを開発する場合は，次の点を考慮してください。

**WPF の場合** : **System.Runtime.InteropServices** 名前空間から **Optional** 属性を使用して，オプションのパラメータを指定する必要があります。たとえば，void myMethod(int p1, [optional] int p2)。

**Silverlight の場合** : UFT では，これらのメソッドのオプション・パラメータについて C# 注釈の使用はサポートしていません。
したがって，オプション・パラメータを含む署名を持った **RunInterface** メソッドのインスタンスを 1 つ設計するのではなく，メソッドの各種インスタンスを，同じメソッド名でパラメータ数を変えて設計してください。

---

**c** Silverlight コントロールのサポートを開発する場合は，テスト・オブジェクト操作の実行を実装するために作成する各メソッドに **ScriptableMember** 属性を付ける必要があります（これは，既存の Microsoft Silverlight 属性です）。

**例 :**

カスタム・サーバで **ICheckBoxRun** インタフェースを定義し，その **Set** メソッドを使用して，UFT が **Set** 操作を実行する場合は，そのインタフェースに **RunInterfaceAttribute** 属性を付けます。

このカスタム・サーバが Silverlight チェックボックスをサポートするためのものである場合は，**Set** メソッドに **ScriptableMemberAttribute** 属性を付けます。

## 4 認識プロパティのサポートの開発

コントロールから認識プロパティ値を取得するようにカスタム・サーバを設計します。これは，テスト・オブジェクト・クラスに対して定義したすべての新規認識プロパティについて行います。また，基本クラスから継承する値取得の実装をオーバーライドする場合にも行います。

**a** 認識プロパティ値の取得をサポートするには，カスタム・サーバに 1 つの Custom Properties インタフェースを実装し，そのインタフェースに **CustomPropInterfaceAttribute** 属性を付けます。

---

**注 : UFT WPF/Silverlight CustomServer** テンプレートを使用して Visual Studio プロジェクトを設定し，プロパティの取得をカスタマイズするように指定していた場合は，インタフェースの定義が自動的に作成されます。

---

**b** Custom Properties インタフェースで，コントロールから値を取得する認識プロパティごとにプロパティを定義します。各プロパティは，同じ名前の認識プロパティに関連する値を返します。

たとえば，カスタム・サーバで **ICheckBoxCustomProp** インタフェースを定義し，その **MyIsChecked** プロパティを使用して，UFT がカスタムの **MyIsChecked** 認識プロパティを取得する場合は，そのインタフェースに **CustomPropInterfaceAttribute** 属性を付けます。

**テスト・オブジェクト名のカスタマイズ（WPF のみ）: logical_name** 認識プロパティを実装すると，UFT はその値をテスト・オブジェクト名として使用します。これにより，テスト・オブジェクトに機能的な論理名（たとえば，コントロール上に表示されるテキスト）を付けることがでます。実装しない場合，テスト・オブジェクトに対して標準設定の名前が生成されます。

---

**注 :** テスト・オブジェクト設定ファイルには，テスト・オブジェクト・クラスに関係するすべての認識プロパティを定義する必要があります。サポートされるすべてのプロパティに対して，プロパティ値を取得する実装は，基本クラスから継承されます。

---

## 5 テーブル・チェックポイントおよび出力値のサポートの開発

テーブルの検証および出力値の取得をサポートするために，カスタム・サーバの **ITableVerify** インタフェース内のすべてのメソッドを実装します。

---

**注 : UFT WPF/Silverlight CustomServer** テンプレートを使用して Visual Studio プロジェクトを設定し，テーブル・チェックポイントのサポートを設計するように指定していた場合は，このインタフェースの事前実装が自動的に作成されます。

---

### 6 別コントロールとして扱わない子コントロールの指定

いくつかの子コントロールがある上位レベルのコントロールの一部であることを UFT に指示するには，**IComponentDetector** インタフェースを実装し，その **IsKnownPartOf** メソッドを, そのような子コントロールについて true を返すように設計します。

**例：**

カレンダ・コントロールの実装には，ボタンが使用されることがあります。これらのボタンを UFT が別オブジェクトとして学習しないようにしたり, ユーザが各ボタンをクリックしたときにステップが記録されないようにするために, カレンダのカスタム・サーバの **IsKnownPartOf** メソッドは，カレンダの一部であるすべてのボタンについて常に true を返します。

---

**注：UFT WPF/Silverlight CustomServer** テンプレートを使用して Visual Studio プロジェクトを設定し，子オブジェクトの扱いをカスタマイズするように指定していた場合は，このインタフェースの事前実装が自動的に作成されます。

---

### 7 ステップ記録のサポートの開発

記録をサポートするたには，コールバック・メソッドをオーバーライドして，**IRecord** インタフェースをカスタム・サーバ・クラスに実装します。

---

**注：UFT WPF/Silverlight CustomServer** テンプレートを使用して Visual Studio プロジェクトを設定し，記録をカスタマイズするように指定していた場合は，このインタフェースの事前実装が自動的に作成されます。

---

**a** テスト・オブジェクトに必要なイベント・ハンドラを定義して，実装します。

**b** テスト・オブジェクトに必要なメッセージ・ハンドラを定義して，実装します。

➤ ウィンドウ・メッセージを直接リッスンする **OnMessage** を実装します。Silverlight コントロールのサポートを開発している場合，**OnMessage** から返される値は，カスタム・サーバがメッセージを処理したかどうか，またはそのメッセージを別の登録済みイベント・ハンドラに渡す必要があるかどうかを示します。

➤ Silverlight コントロールのサポートを開発している場合は，**GetWndMessageFilter** を実装して，ウィンドウ・メッセージをリッスンするオブジェクトを指定します。コントロール自体（およびそれと一体とみなされる子）のメッセージ，コントロールの子のメッセージ，またはアプリケーションに対するすべてのメッセージをリッスンできます。

45 ページ「トラブルシューティングと制限事項 - サポートの開発」で，ウィンドウ・メッセージを処理する際の制限についても参照してください。

**c** ハンドラの登録と解放を行う **RecordInit** および **RecordStop** を実装します。

### 8 カスタム・サーバをデプロイする準備

カスタム・サーバをコンパイルして DLL を作成します。

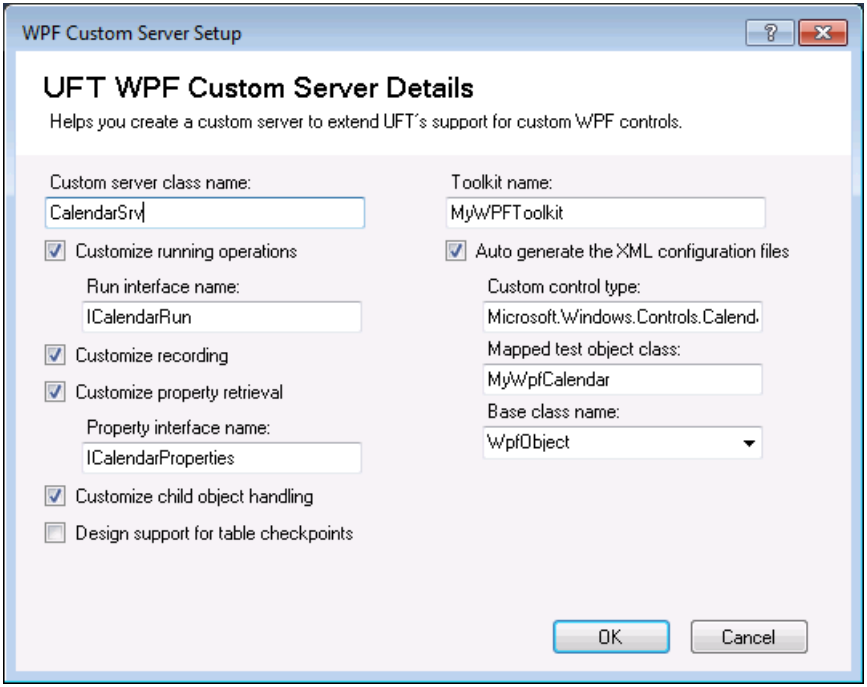
カスタム・サーバを 32 ビットおよび 64 ビット・アプリケーションで使用きるようにするには，**Platform target** オプションを **Any CPU** に設定して，カスタム・サーバ DLL をコンパイルします。

# [WPF/Silverlight Custom Server Setup] ダイアログ・ボックス（Microsoft Visual Studio）

Microsoft Visual Studio で WPF または Silverlight Add-in Extensibility プロジェクトを作成するために UFT WPF または Silverlight Custom Server テンプレートを選択すると，このダイアログ・ボックスが開きます。

このダイアログ・ボックスでは，作成するサポートを記述する項目を指定できます。拡張プロジェクトは Visual Studio で作成され，サポートの作成に必要な基本コンテンツ，インフラストラクチャ，参照を元にファイルが作成されます。

次に，[WPF Custom Server Setup] ダイアログ・ボックスを示します。[Silverlight Custom Server Setup] ダイアログ・ボックスには，次の図と同じオプションが含まれていますが，ダイアログ・ボックスのタイトルと使用可能な基本クラスが異なります。



| | |
|---|---|
| **アクセス方法** | **1** Microsoft Visual Studio で，［**ファイル**］＞［**新規作成**］＞［**プロジェクト**］を選択します。<br>**2** 次のいずれかを選択します。<br>➤ **Visual C# Windows** プロジェクト・タイプと **UFT WPF CustomServer** テンプレート<br>➤ **Visual C# Silverlight** プロジェクト・タイプと **UFT Silverlight CustomServer** テンプレート<br>**3** 新しいプロジェクトの名前と場所，ソリューションの名前を指定します（または，標準値をそのまま使用します）。<br>**4** ［**OK**］をクリックします。2 で選択したテンプレートに応じて，［WPF Custom Server Setup］ダイアログ・ボックスまたは［Silverlight Custom Server Setup］ダイアログ・ボックスが開きます。<br>**このダイアログ・ボックスでは，いくつかのフィールドの標準値としてプロジェクト名が使用されます。** |
| **重要な情報** | **UFT Silverlight CustomServer** テンプレートを使用してプロジェクトを作成するには，Microsoft Silverlight Tools for Visual Studio をインストールする必要があります。 |
| **関連タスク** | 24 ページ「WPF または Silverlight のカスタム・ツールキットのサポートを作成する方法」 |
| **参照** | ➤ 13 ページ「WPF または Silverlight のカスタム・ツールキットに対する UFT サポートの開発」<br>➤ 21 ページ「カスタム・サーバ」<br>➤ Custom Server API Reference，Toolkit Configuration Schema ヘルプ，Unified Functional Testing Test Object Schema（WPF および Silverlight Add in Extensibility ヘルプで利用可能）。 |

次に，ユーザ・インタフェース要素について説明します。

| UI 要素 | 説明 |
|---|---|
| **Custom server class name** | 新しいプロジェクトで作成するカスタム・サーバ・クラスの名前。<br>**注：**標準設定では，この名前が［**Run interface name**］，［**Property interface name**］，［**Mapped test object class**］の標準値として使用されます。したがって，標準値以外の値に変更していない場合，［**Custom server class name**］を変更すると，それに伴って上記のフィールドの値も変更されます。 |
| **Customize running operations** | コントロールでテスト・オブジェクト操作を実行する際の実装を設計またはオーバーライドします。<br>このオプションを選択すると，指定した実行インタフェースが定義され，**RunInterfaceAttribute** にタグ付けされます。カスタム・サーバ・クラスでは，インタフェースは，サンプルのテスト・オブジェクト操作のメソッド・スタブを使用して実装されます。また，サンプルのテスト・オブジェクト操作はプロジェクトのテスト・オブジェクト設定ファイルに追加されます。 |
| **Run interface name** | テスト・オブジェクト操作のサポートを実装するインタフェースの名前。<br>この値を指定するには，［**Customize running operations**］を選択する必要があります。 |
| **Customize recording** | コントロールでテスト・オブジェクト操作を記録する際のサポートを設計します。<br>このオプションを選択すると，カスタム・サーバ・クラスの定義には，**IRecord** インタフェースの実装と，インタフェースのメソッドで使用するメソッド・スタブが含まれます。 |
| **Customize property retrieval** | コントロールから認識プロパティの値を取得する実装を設計またはオーバーライドします。<br>このオプションを選択すると，指定した Property インタフェースが定義され，**CustomPropInterfaceAttribute** にタグ付けされます。カスタム・サーバ・クラスでは，このインタフェースは，サンプルの認識プロパティのメソッド・スタブを使用して実装されます。また，サンプルの認識プロパティはプロジェクトのテスト・オブジェクト設定ファイルに追加されます。 |

| UI 要素 | 説明 |
|---|---|
| **Property interface name** | プロパティ値の取得用のサポートを実装するインタフェースの名前。<br>この値を指定するには，［**Customize property retrieval**］を選択する必要があります。 |
| **Customize child object handling** | UFT によって，カスタム・コントロールの子オブジェクトの一部を，独立したオブジェクトではなく，コントロールの一部として処理します。<br>このオプションを選択すると，**IComponentDetector** インタフェースの **IsKnownPartOf** メソッドの事前実装がカスタム・サーバ・クラスで作成されます。関連する子オブジェクトで true を返すメソッドを実装してください。 |
| **Design support for table checkpoints** | UFT がテスト・オブジェクトで作成する標準チェックポイントと出力値を，テーブルのチェックポイントおよび出力値とするかどうかを指定します。<br>このオプションを選択すると，**ItableVerify** インタフェースの事前実装がカスタム・サーバ・クラスに作成されます。必要に応じてこのインタフェースを実装してください。 |
| **Toolkit name** | このツールキット・サポート・セットでサポートする環境（コントロール・セット）の名前。<br>WPF または Silverlight ツールキットのいずれかを指定します。サポートのデプロイメントが完了した後に UFT のアドイン・マネージャを開くと，ここで指定した名前は WPF アドイン・ノードの下の子ノードとして表示され，この環境のサポートをロードするかどうかを指定できるようになります。<br>新しく作成したプロジェクトでは，ツールキット名を使用して設定ファイル名が作成され，テスト・オブジェクト設定ファイルの **TypeInformation** 要素の **PackageName** 属性に入力されます（［**Auto generate the XML configuration files**］を選択している場合）。これは，プロジェクトのルート名前空間でも使用されます。 |

| UI 要素 | 説明 |
|---|---|
| **Auto generate the XML configuration files** | プロジェクトの新規作成時に，ツールキット設定ファイルとテスト・オブジェクト設置ファイルも作成するかどうかを指定します。設定ファイルは，このダイアログ・ボックスで指定した内容に応じて，基本的な定義内容で作成されます。<br>ツールキット・サポート・セットでは，サポート対象となるすべてのコントロールの定義が 1 つのツールキット設定ファイル，1 つのテスト・オブジェクト設定ファイルに含まれます。これに対して，カスタム・サーバは，サポート対象のコントロールごとに設計します。したがって，テンプレートをもとにカスタム・サーバを追加する場合，XML ファイルを追加作成しない選択も可能です。または，XML 設定ファイルを作成し，このファイルの内容を主要な設定ファイルにコピーすることもできます。 |
| **Custom control type** | サポートを作成するときのカスタム・コントロール・タイプの名前。<br>**Silverlight：**名前空間を含む完全なタイプ名。<br>**WPF：**次のいずれかになります。<br>➤ 名前空間を含む完全なタイプ名。例を次に示します。<br>Infragistics.Controls.Editors.XamComboEditor<br>➤ 完全なタイプ名とアセンブリ名。例を次に示します。<br>Infragistics.Controls.Editors.XamComboEditor,<br>InfragisticsWPF4.Controls.Editors.XamComboEditor.v12.1<br>➤ アセンブリ修飾名。例を次に示します。<br>Infragistics.Controls.Editors.XamComboEditor,<br>InfragisticsWPF4.Controls.Editors.XamComboEditor.v12.1,<br>Version=12.1.20121.1010, Culture=neutral,<br>PublicKeyToken=7dd5c3163f2cd0cb<br>**注：**ほとんどの場合，完全なタイプ名を指定すれば問題ありません。同じコントロールの異なるバージョンがアプリケーションに含まれているなど，複数のアセンブリに同じコントロールが含まれている場合などには，これよりも複雑な名前を指定できます。 |

| UI 要素 | 説明 |
|---|---|
| **Mapped test object class** | UFT がカスタム・コントロールを表すために使用するテスト・オブジェクト・クラス。<br>新しく作成したプロジェクトでは，新しいテスト・オブジェクト・クラスの基本的な定義が，テスト・オブジェクト設定ファイルで作成されます（［**Auto generate the XML configuration files**］を選択した場合)。 |
| **Base class name** | 新しいテスト・オブジェクト・クラスが拡張する対象となるテスト・オブジェクト・クラス。<br>組み込みの UFT テスト・オブジェクト・クラスのリストから選択するか，WPF または Silverlight Add-in Extensibility テスト・オブジェクト設定ファイルで定義したテスト・オブジェクト・クラスの名前を入力します(（WPF テスト・オブジェクト・クラスが拡張する対象は WPF テスト・オブジェクト・クラス，Silverlight テスト・オブジェクト・クラスは Silverlight テスト・オブジェクト・クラスです)。<br>**標準設定**：WpfObject または SlvObject |

# トラブルシューティングと制限事項 - サポートの開発

この項では，WPF または Silverlight のカスタム・コントロールのサポートを開発する際の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。

- テスト・オブジェクト操作を定義し，対応するカスタム・サーバ・メソッドを OUT または IN/OUT パラメータで定義したときに，テスト・オブジェクト操作が正しく実行されません。

  **回避策：**WPF または Silverlight Add-in Extensibility カスタム・サーバを使用してサポートするテスト・オブジェクト・メソッドを設計するときは，IN パラメータのみを使用してください。OUT または IN/OUT（**ref**）パラメータではなく，値を返すように操作を設計してください。

- コントロールがポップアップ・コントロールとして実装されている場合，そのコントロールはカスタム・コントロールの子として扱われません。これは，**IsKnownPartOf** メソッドを実装しても，このようなコントロールを無視し，カスタム・コントロールと一体化した部分として扱うように UFT に対して指示できないことを意味します。

  このようなコントロールは，UFT では独立したコントロールとして認識され，調査され，学習されます。

  それでも，これらのコントロールのイベントをリッスンするイベント・ハンドラを登録して，これらのコントロールで発生するイベントに応じて，カスタム・コントロールのステップ記録を実装することはできます。

- **Silverlight のみに関連：**カスタム・サーバのテスト・オブジェクト・メソッドの実装に，テストの次のステップが実行されるまでブロックする操作が含まれる場合，テストの実行セッションが次のステップに進みません。

  例：アプリケーションのカスタム・ボタンをクリックして，モーダル・ダイアログ・ボックスが開く場合に，テストに次のステップが含まれているとします。

  ```
  MySlvButton.Click
  SlvDialog.Close
  ```

  **MySlvButton** をサポートするカスタム・サーバの **Click** メソッドの実装が (UtilityObject.ApplicationObject as UIElement).Click を呼び出し，モーダル・ダイアログ・ボックスが閉じるまでその **Click** メソッドが戻らない場合，テストの最初のステップは実行されますが，2 番目のステップに進むことはありません。

**回避策：**次のいずれかを行います。

- **BeginInvoke** を使用して，ブロッキング・ステートメントを非同期に呼び出します。
- マウスまたはキーボード操作を使用して，テスト・オブジェクト・メソッドを実装します（たとえば，UtilityObject.MouseClick を使用してボタンをクリックします）。

- **Silverlight のみに関連：**サポートするカスタム・コントロール以外のコントロールに対して生成されるウィンドウ・メッセージを受け取るようサポートを設計した場合に，それにもかかわらず，そのようなウィンドウ・メッセージの一部がカスタム・サーバに渡されないことがあります。

  これは，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・コントロール上で何らかの操作が実行されないと，作成されないためです。

  **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・コントロールで処理するように指定する場合，カスタム・サーバを作成し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てからでなければ，アプリケーションのほかのコントロールに対して生成され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・セットの UFT ユーザにこの問題について説明する必要があります。

# 第 2 章

# チュートリアル：カスタム WPF コントロールの UFT サポートの作成

このチュートリアルでは，WPF Calendar コントロールのサポートを手作業で作成して，WPF Add-in Extensibility ツールキット・サポート・セットの作成の基礎について学習します。ツールキット（環境）は，1 つのパッケージでサポートを提供する一連のコントロールです。このチュートリアルのツールキットの名前は **MyWpfToolkit** で，**Microsoft.Windows.Controls.Calendar** コントロールのみを含んでいます。

このチュートリアルを実行するには，WPF および Silverlight Add-in Extensibility SDK に加えて，サポート対象バージョンの Microsoft Visual Studio がインストールされている必要があります（Extensibility SDK は，Visual Studio の後でインストールする必要があります）。

---

**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，または Unified Functional Testing DVD のルート・フォルダに収録されています。

---

このチュートリアルでは，Visual Studio で **UFT WPF CustomServer** プロジェクト・テンプレートを使用して，ツールキット・サポート・セットの作成に必要なファイルを設定します。独自のサポートを開発するときに，ツールキット・サポート・セット・ファイルを手動で作成する場合は，24 ページ「WPF または Silverlight のカスタム・ツールキットのサポートを作成する方法」の手順を実行してください。

このチュートリアルの全体にわたって記述されているクラスとインタフェース・メソッドの詳細については，『Custom Server API Reference』（WPF および Silverlight Add-in Extensibility ヘルプで利用可能）を参照してください。

---

**注 :** Silverlight コントロールのサポートは，WPF コントロールのサポートとほとんど同じ方法で開発します。このチュートリアル全体を通して，Silverlight ツールキット・サポート・セットのときに必要な変更点があれば，その変更点を説明しています。

---

WPF Calendar アプリケーションがインストールされている場所は，**<WPF および Silverlight Add-in Extensibility SDK インストール・フォルダ>\samples\WPFExtCalendarSample\Application** です。

**<WPF および Silverlight Add-in Extensibility SDK インストール・フォルダ>\samples\WPFExtCalendarSample\Support** フォルダには，このコントロールのサポートを構成する Microsoft Visual Studio ソリューションおよび XML ファイルが含まれており，そのサポートは，このチュートリアルで作成するサポートと類似しています。これらのファイルは，チュートリアルの実行中に参照できます。

**このチュートリアルの内容 :**

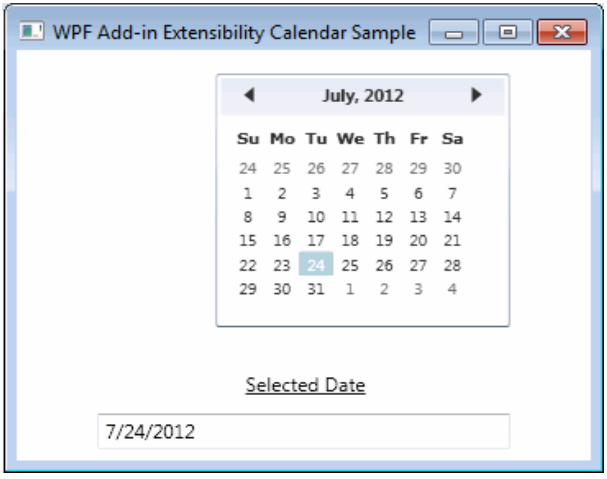
- WPF Calendar コントロールのサポートの計画（49ページ）
- WPF Calendar コントロール用の WPF Add-in Extensibility プロジェクトの設定（52ページ）
- ツールキット設定ファイルの設計（56ページ）
- テスト・オブジェクト設定ファイルの設計（58ページ）
- ツールキット・サポート・セットのデプロイとテスト（61ページ）
- 基本カスタム・サーバの設計（65ページ）
- 認識プロパティ値を取得するためのサポートの実装（67ページ）
- 基本カスタム・サーバおよび認識プロパティ・サポートのデプロイとテスト（68ページ）
- テスト・オブジェクト操作の実行に対するサポートの実装（70ページ）

## WPF Calendar コントロールのサポートの計画

この項では，サポートするコントロールの動作と，そのコントロールを UFT が認識してやり取りする方法について検討します。さらに，UFT の動作をどのようにカスタマイズすると，わかりやすく管理しやすいテスト・ステップを作成できるかを明らかにします。

### 1 サンプルの WPF Calendar アプリケーションの実行とその動作の調査

**a** **<WPF および Silverlight Add-in Extensibility SDK インストール・フォルダ>\samples\WPFExtCalendarSample\Application\WpfCalendar.exe** ファイルをダブルクリックします。Calendar アプリケーションが開きます。



このアプリケーションには，3 つのコントロールがあります。

- ボタン付きのカレンダ表示領域：**Microsoft.Windows.Controls.Calendar**
- テキスト・ラベル：**System.Windows.Controls.TextBlock**
- 選択された日付を表示するエディット・ボックス：**System.Windows.Controls.TextBox**

**b** アプリケーションの機能を調べます。

- 左右の矢印をクリックすると，前の月または次の月に移動できます。
- カレンダでその月の日をクリックして，日付を選択できます。
- 選択した日付は，テキスト・ボックスに表示されます。

### 2 UFT でオブジェクト・スパイを使用して，UFT が Calendar アプリケーションのコントロールをどのように認識するかを確認

**a** UFT を開いて，GUI テストを開きます。

**b** ［**記録**］＞［**記録と実行環境設定**］を選択し，［Windows アプリケーション］タブで，UFT がカレンダ・アプリケーションでテストの記録と実行を行えるようにオプションが選択されていることを確認します。

**c** オブジェクト・スパイを実行し，WPF Calendar を調査します。

UFT は，Calendar アプリケーションを WpfWindow として認識します。このウィンドウ内で，**Microsoft.Windows.Controls.Calendar** コントロールは汎用の WpfObject として認識され，**System.Windows.Controls.TextBox** は WpfEdit object として認識されます。

さらに，**Microsoft.Windows.Controls.Calendar** コントロール内の日にちは WpfWindow 内の独立した WpfButtons の集まりとして認識されます。

Calendar コントロールに含まれるその他のユーザ・インタフェース要素（左右の矢印，月と年のバナーなど）は無視されます。

### 3 UFT を使用して Calendar コントロールを学習し，オブジェクト・リポジトリに追加

**a** オブジェクト・リポジトリを開きます。

**b** ［**ローカルへオブジェクトを追加**］ボタンをクリックします。

**c** カレンダ内の領域をクリックします。

**d** カレンダの別の領域（日にち，テキスト・ボックス，月と年のバナーなど）について，ステップ b および c を繰り返します。

UFT は，アプリケーションを WpfWindow として学習します。このウィンドウ内で，カレンダ表示領域は汎用の WpfObject として，［Selected Date］ボックスは WpfEdit object として別々に学習します。

日にちは，独立した WpfButtons の集まりとして学習されますが，その他のユーザ・インタフェース要素（左右の矢印，月と年のバナーなど）はまったく学習されません。

### 4 Calendar コントロールでのテストの記録

**a** ［**記録**］をクリックします。

**b** カレンダのさまざまな領域をクリックします。

左右の矢印または月と年のバナーをクリックすると，クリックした位置の座標を指定して，WpfObject での汎用クリックが記録されます。

日にちをクリックして選択した場合や，［Selected Date］ボックスをクリックした場合は，何も記録されません。

アプリケーションのその他の領域をクリックすると，WpfWindow での汎用クリックが記録されます。

### 5 結論：MyWpfCalendar テスト・オブジェクト・クラスの開発

➤ 機能テストを行うために，**Microsoft.Windows.Controls.Calendar** コントロールを UFT 内で 1 つの MyWpfCalendar テスト・オブジェクトで表すようにします。

➤ Calendar コントロール内のボタンは，別のコントロールとして扱わないようにします。

- ［Selected Date］ボックスは読み取り専用ボックスであるため，ユーザ・アクティビティは許可されず，ステップも記録されません。そのため，**System.Windows.Controls.TextBox** コントロールを MyWpfCalendar テスト・オブジェクトの一部としてサポートする必要はありません。また，機能テストのためのカスタマイズも必要ありません。
- MyWpfCalendar テスト・オブジェクト・クラスは，既存の UFT テスト・オブジェクト・クラス WpfObject をベースとし，その機能を拡張するようにします。
- MyWpfCalendar テスト・オブジェクト・クラスは，カレンダ関連の操作をサポートするようにします。

  このチュートリアルでは，**SetDate**（標準設定操作），**Next**，**Previous** の各テスト・オブジェクト・メソッドおよび **SelectedDate** テスト・オブジェクト・プロパティのサポートを開発します。
- カレンダのさまざまなユーザ・インタフェース要素で実行されるユーザ操作は，カレンダ全体に対する上位レベルの操作として解釈され記録されるようにします。例：**SetDate**，**Next**，**Previous** など。
- MyWpfCalendar テスト・オブジェクト・クラスは，カレンダに関する認識プロパティ（is_today_highlighted など）をサポートするようにします。

## WPF Calendar コントロール用の WPF Add-in Extensibility プロジェクトの設定

WPF Add-in Extensibility サポート・セットは，次の必須ファイルで構成されます。

- **テスト・オブジェクト設定 XML ファイル**：このファイルでは，新しいテスト・オブジェクト・タイプが定義されます。詳細については，15 ページ「テスト・オブジェクト設定 XML ファイル」を参照してください。
- **ツールキット設定ファイル**：このファイルでは，WPF コントロール・タイプが，テスト・オブジェクト・タイプにマッピングされ，記録および実行のロジックを実装するカスタム・サーバにマッピングされます。スキーマの詳細については，『Toolkit Configuration Schema』（WPF および Silverlight Add-in Extensibility ヘルプで利用可能）を参照してください。
- **カスタム・サーバの実装を含む .NET DLL**：詳細については，21 ページ「カスタム・サーバ」を参照してください。

WPF および Silverlight Add-in Extensibility SDK は，プロジェクト・テンプレートと設定ダイアログ・ボックスを Microsoft Visual Studio にインストールします。これらは，ツールキット・サポート・セットの作成に必要なファイルを設定する際に役立ちます。

テンプレートは，サポートするコントロールごとに使用します。このテンプレートによって，カスタム・サーバ DLL の作成に必要な XML ファイルと Microsoft Visual Studio ソリューションの両方が設定されます。複数のコントロールのサポートを 1 つのツールキットに作成する場合は，各コントロールに対して作成される XML コンテンツを 1 つのツールキット設定ファイルと，ツールキット用の 1 つのテスト・オブジェクト設定ファイルにまとめる必要があります。

このチュートリアルでは，1 つのコントロールのみに対するツールキット・サポートを作成するので，プロジェクト・テンプレートで作成された XML ファイルをそのまま使用できます。

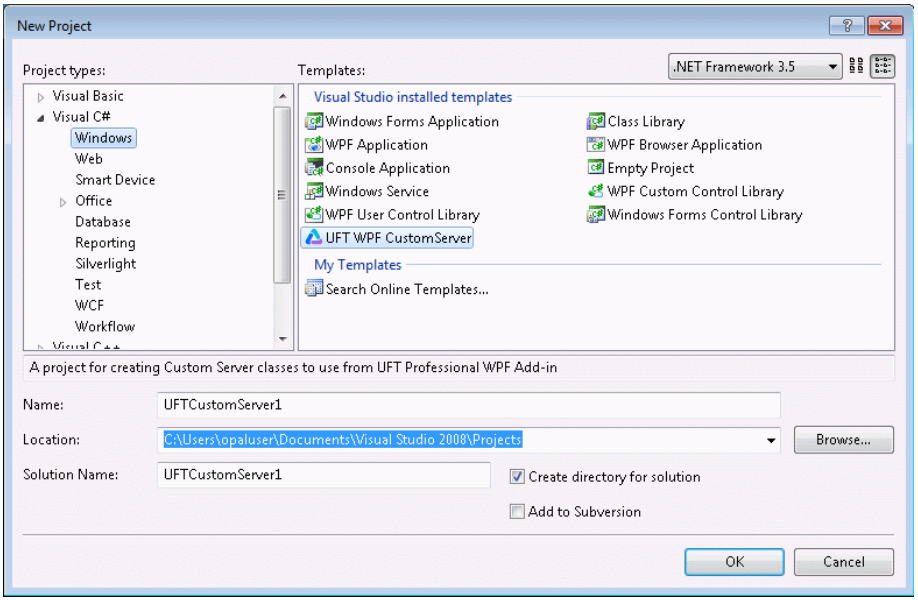
---

**注 :** この項の Microsoft Visual Studio ダイアログ・ボックスの画像は，Microsoft Visual Studio 2008 のものです。使用するバージョンが異なる場合は，ダイアログ・ボックスの外観もわずかに異なることがあります。

---

### Microsoft Visual Studio での WPF Add-in Extensibility プロジェクトの作成

**1** Microsoft Visual Studio を開いて，［**新しいプロジェクト**］をクリックします。［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。




**2** **Visual C# Windows** プロジェクト・タイプと **UFT WPF CustomServer** テンプレートを選択し，［**OK**］をクリックします。

---

**注**：Silverlight コントロールのサポートを開発している場合は，**Visual C# Silverlight** プロジェクト・タイプと **UFT Silverlight CustomServer** テンプレートを選択します。

---

**3** プロジェクトの［**名前**］として「CalendarSrv」を入力し，［**OK**］をクリックします。［WPF/Silverlight Custom Server Setup］ダイアログ・ボックスが開きます。

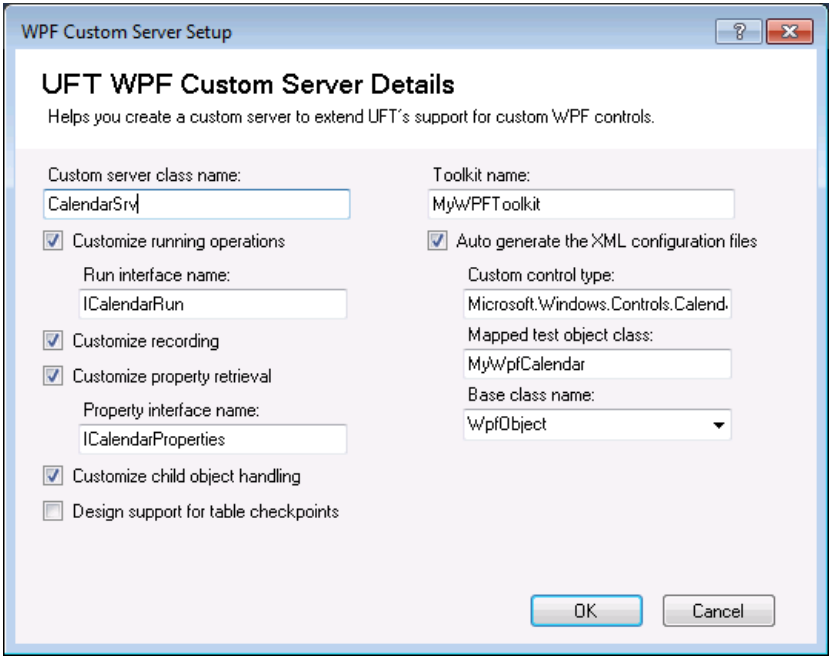
**4** 作成するサポートを記述する詳細をこのダイアログ・ボックスで指定すると，その内容に応じて，サポートの作成に必要なファイルが作成されます。

このチュートリアルで行う処理は次のとおりです。

- **MyWpfToolkit** ツールキットのサポートを作成します。
- このツールキット内で，**Microsoft.Windows.Controls.Calendar** WPF コントロールのサポートを作成します。
- UFT で Calendar コントロールを表す **MyWpfCalendar** テスト・オブジェクト・クラスを，標準の UFT **WpfObject** クラスをベースにして作成します。
- **CalendarSrv** カスタム・サーバ・クラスを作成して，コントロールのサポートを提供します。**CalendarSrv** カスタム・サーバ・クラス内で，操作の実行，プロパティの取得，記録，子オブジェクトの処理をカスタマイズします。

これらの詳細を，次の画像に示すように［WPF Custom Server Setup］ダイアログ・ボックスで指定し，XML ファイル，コメント，サンプル・コードを自動生成するためのオプションも選択します。

［**Run interface name**］および［**Property interface name**］は，下図のように入力してください。このチュートリアルでは，ダイアログ・ボックスに提示される標準設定の名前は使用しません。



このダイアログ・ボックスの詳細については, 39 ページ「［WPF/Silverlight Custom Server Setup］ダイアログ・ボックス（Microsoft Visual Studio）」を参照してください。

**5** ［**OK**］をクリックします。

CalendarSrv ソリューションが，関連するファイルと参照とともに作成されます。このソリューションには，ツールキット設定ファイル（**MyWpfToolkit.cfg**），テスト・オブジェクト設定ファイル（**MyWpfToolkitTestObjects.xml**），カスタム・サーバ・クラスの C# ファイル（**CalendarSrv.cs**）が含まれます。

また，WPF Add-in Extensibility API を含む **Mercury.QTP.WpfAgent.dll** ファイルへの参照も含まれます。

---

**注：UFT Silverlight CustomServer** テンプレートを使用して作成されるソリューションには，Silverlight Add-in Extensibility API を含む **Mercury.QTP.Slv.CustomServer.dll** ファイルへの参照が含まれます。

---

## ツールキット設定ファイルの設計

ツールキット設定ファイルの名前により，新しくサポートされた環境が UFT に通知されます。このファイルを UFT コンピュータ上の正しい場所にデプロイすると，UFT の起動時に，その環境がアドイン・マネージャで Web アドインの下の子ノードとして表示されます。この環境のチェック・ボックスを選択すると，UFT はそのサポートをロードします（Silverlight を使用している場合，Silverlight Add-in も選択する必要があります）。

設定ファイルの内容によって，コントロールのサポート方法，各カスタム・コントロールに使用するテスト・オブジェクト・クラスとカスタム・サーバなどが定義されます。

**MyWpfToolkit.cfg** ファイルを開いて内容を確認します。

**MyWpfToolkit.cfg** ファイルは，指定した仕様に基づいて自動的に作成されています。したがって，このファイルには，MyWpfToolkit 環境で WPF Calendar コントロールをサポートするために必要なすべての内容がすでに含まれています。

```
<?xml version="1.0" encoding="UTF-8"?>
<Controls>
 <Control Type="Microsoft.Windows.Controls.Calendar" MappedTo="MyWpfCalendar">
    <CustomServer>
        <Component>
            <DllName>CalendarSrv.dll</DllName>
            <TypeName>MyWpfToolkit.CalendarSrv</TypeName>
        </Component>
    </CustomServer>
 </Control>
</Controls>
```

**Control** 要素の属性は，**タイプ Microsoft.Windows.Controls.Calendar**（名前空間を含む完全なコントロール・タイプ名を指定することが必要）のコントロールがテスト・オブジェクト・クラス **MyWpfCalendar** に**マッピング（MappedTo）**されるよう指定しています。

**CustomServer** ＞ **Component** 要素は，このコントロール・タイプのサポートを提供するカスタムサーバの DLL とタイプを指定しています。

---

**注：カスタム・サーバ・タイプ**は，名前空間を含む完全なタイプ名であることが必要です。また，Silverlight では，その他の情報も指定する必要があります。詳細については，Toolkit Configuration Schema ヘルプを参照してください。

---

ツールキット設定ファイルの要素と属性の詳細については，Toolkit Configuration Schema ヘルプ（WPF および Silverlight Add-in Extensibility ヘルプで利用可能）を参照してください。

# テスト・オブジェクト設定ファイルの設計

テスト・オブジェクト設定ファイルを使用して，MyWpfToolkit 環境とそのテスト・オブジェクト・クラスを UFT に通知します。

1 **MyWpfToolkitTestObjects.xml** ファイルを開きます。

**MyWpfToolkitTestObjects.xml** ファイルは，**TypeInformation** 要素の **AddinName** 属性が WPF に設定され，**PackageName** 属性が MyWpfToolkit に設定されて作成されています。これにより，テスト・オブジェクト設定ファイル（およびそこで定義されているテスト・オブジェクト）が，WPF Add-in の下の MyWpfToolkit 環境に関連付けられます。UFT を開いたときに MyWpfToolkit 環境を選択しない場合, UFT はこのファイルのテスト・オブジェクト・クラス定義を無視します。

---

**注 :** Silverlight コントロールのサポートを開発している場合，**TypeInformation** 要素は Silverlight 設定されます。

---

［WPF Custom Server Setup］ダイアログ・ボックスで指定した情報に基づいて，MyWpfCalendar テスト・オブジェクト・クラスの **ClassInfo** 要素も作成され，WpfObject が基本クラスに指定されています。これは，定義する新しい MyWpfCalendar テスト・オブジェクト・クラスが，WpfObject のメソッド，汎用タイプ，ヘルプ・ファイルなどを継承することを意味します。

**2** テスト・オブジェクト・クラスを拡張し，カレンダ特有の操作および認識プロパティを追加するため，**MyWpfToolkitTestObjects.xml** 内のコメント行を変更して，テスト・オブジェクト設定ファイルに次の内容が含まれるようにします。

```
<TypeInformation AddinName="WPF" PackageName="MyWpfToolkit">
   <ClassInfo Name="MyWpfCalendar" BaseClassInfoName="WpfObject"
       DefaultOperationName="SetDate">
      <IdentificationProperties>
         <IdentificationProperty Name="devname" ForDescription="true" />
         <IdentificationProperty Name="devnamepath" ForAssistive="true"
               AssistivePropertyValue="1"/>
         <IdentificationProperty Name="regexpwndtitle" ForAssistive="true"
               AssistivePropertyValue="2"/>
         <IdentificationProperty Name="x" ForVerification="true" />
         <IdentificationProperty Name="y" ForVerification="true" />
         <IdentificationProperty Name="is_today_highlighted" ForVerification="true"/>
      </IdentificationProperties>
      <TypeInfo>
         <Operation Name="Next" PropertyType="Method"/>
         <Operation Name="Prev" PropertyType="Method"/>
         <Operation Name="SelectedDate" PropertyType="Property_Get">
            <ReturnValueType>
               <Type VariantType="VT_BSTR"/>
            </ReturnValueType>
         </Operation>
         <Operation Name="SetDate" PropertyType="Method">
            <Argument Name="Date" IsMandatory="true" Direction="In">
               <Type VariantType="VT_BSTR"/>
            </Argument>
         </Operation>
      </TypeInfo>
   </ClassInfo>
</TypeInformation>
```

これで，次のものが定義されました。

- **Previous**，**Next**，**SetDate** テスト・オブジェクト・メソッドおよび **SelectedDate** プロパティ。関連するすべてのパラメータ，戻り値，戻り値の型も含まれます。**SetDate** は，このテスト・オブジェクト・クラスの標準設定操作です。
- **devname**，**devnamepath**，**regexpwndtitle**，**x**，**y**，**is_today_highlighted** 認識プロパティ。
  - 最初の 5 つのプロパティは基本クラスによってサポートされており，それらの値を取得するための実装は継承されます。ただし，認識プロパティの**定義**は自動的には継承されません（それが，ここで定義することが必要な理由です）。
  - テスト・オブジェクト記述に含めるかどうか，補足プロパティとして使用するかどうか，またはチェックポイントでの検証に使用可能かどうかを，認識プロパティごとに指定しました。

テスト・オブジェクト設定ファイルの要素と属性の詳細については，Unified Functional Testing Test Object Schema ヘルプ（WPF および Silverlight Add-in Extensibility ヘルプで利用可能）を参照してください。

# ツールキット・サポート・セットのデプロイとテスト

テスト・オブジェクト設定ファイルで **MyWpfCalendar** テスト・オブジェクト・クラスを定義し，ツールキット設定ファイルで Calendar コントロールをこのテスト・オブジェクト・クラスにマッピングしたら，UFT でツールキット・サポート・セットを使用することによる影響をテストできる準備ができています。

---

**注：** ツールキット・サポート・セットのカスタマイズでは，テスト・オブジェクト設定ファイルで **IdentificationProperty** 要素の属性を変更する場合，カスタム・ツールキット・サポートの設計中は，**TypeInformation** 要素の **DevelopmentMode** 属性を true に設定しておいてください。通常の用途にカスタム・ツールキット・サポート・セットをデプロイする場合は，この属性を削除（または false に設定）してから作業を開始してください。ただし，この操作はこのチュートリアルのレッスンでは必要ありません。詳細については，84 ページ「テスト・オブジェクト設定ファイルで指定された IdentificationProperty 要素の属性を変更」を参照してください。

---

**ツールキット・サポート・セットをデプロイするには，次の手順を実行します。**

1. **MyWpfToolkitTestObjects.xml** ファイルを **<UFT インストール・フォルダ>\dat\Extensibility\WPF** にコピーします。
2. **<UFT インストール・フォルダ>\dat\Extensibility\WPF** フォルダに，**MyWpfToolkit** という名前のフォルダを作成します。
3. **MyWpfToolkit.cfg** ファイルを **<UFT インストール・フォルダ>\dat\Extensibility\WPF\MyWpfToolkit** フォルダにコピーします。

---

**注：** Silverlight コントロールのサポートを開発している場合は，上記のパスの WPF を Slv に置き換えます。

---

**ツールキット・サポート・セットをテストするには，次の手順を実行します。**

1 ツールキット・サポート・セットをデプロイしたら，UFT を開き，GUI テストを開きます。

---

**注：**UFT は，起動時にツールキット・サポート・ファイルを読み込みます。UFT が開いている場合には，UFT を終了してから再度起動する必要があります。

---

［アドイン マネージャ］ダイアログ・ボックスで，使用可能なアドインのリストに **MyWpfToolkit** が WPF 環境の子として表示されます（［アドイン マネージャ］ダイアログ・ボックスが開かない場合は，『HP Unified Functional Testing アドイン・ガイド』を参照してください）。

2 **MyWpfToolkit** のチェック・ボックスを選択し，［**OK**］をクリックします。UFT が開き，設計したサポートをロードします。

3 ［オブジェクト リポジトリ］ダイアログ・ボックスの［**新規テストオブジェクトの定義**］ボタンをクリックすると，［新規テストオブジェクトの定義］ダイアログ・ボックスが開きます。MyWpfToolkit 環境が［**環境**］リストに表示されます。リストから MyWpfToolkit 環境を選択すると，テスト・オブジェクト設定ファイルで定義した **MyWpfCalendar** テスト・オブジェクト・クラスが，［**クラス**］リストに表示されます。

4 ［**ツール**］＞［**オブジェクトの認識**］を選択します。［オブジェクトの認識］ダイアログ・ボックスの［**環境**］リストで MyWpfToolkit 環境を選択したときに表示される **MyWpfCalendar** テスト・オブジェクト・クラスの認識プロパティ定義が，テスト・オブジェクト設定ファイルの定義に一致していることを確認します。

5 サンプルのコントロールを実行します。これは，**<WPF および Silverlight Add-in Extensibility SDK インストール・フォルダ>\samples\WPFExtCalendarSample\Application\WpfCalendar.exe** ファイルを開くことで行います。

---

**注：** UFT は，アプリケーションの起動時にアプリケーションとの接続を確立します。したがって，Calendar アプリケーションが開いている場合は，閉じてから再度起動する必要があります。

---

6 UFT で，次の操作を Calendar コントロールに対して実行し，UFT がコントロールをどのように認識するかを確認します（UFT の操作の詳細については，『HP Unified Functional Testing ユーザーズ・ガイド』を参照してください）。

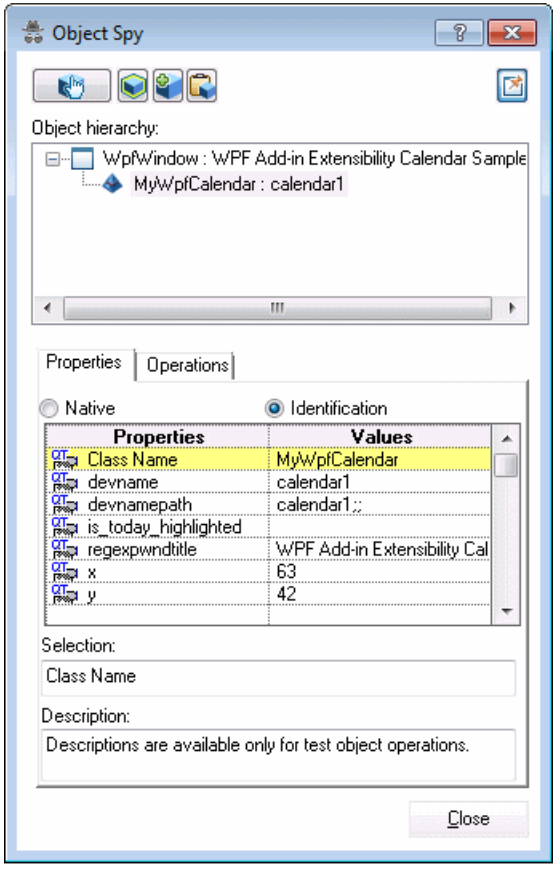
➤ オブジェクト・スパイを使用して，UFT が Calendar コントロールをどのように認識するかを確認し，その認識プロパティとテスト・オブジェクト操作を表示します。

➤ カレンダは，**MyWpfCalendar** テスト・オブジェクト・クラスで表されています。

➤ カレンダの日にちの数字は，今の時点では別のテスト・オブジェクトとして認識されています。このチュートリアルの後半で子オブジェクトの扱いをカスタマイズして，そのような処理が行われないようにします。

➤ テスト・オブジェクト操作のリストには，**WpfObject** 基本クラスから継承されたすべての操作（メソッドとプロパティ）と，**MyWpfToolkitTestObjects.xml** テスト・オブジェクト設定ファイル内で定義したすべての操作が表示されています。

➤ 認識プロパティのリストには，**MyWpfToolkitTestObjects.xml** テスト・オブジェクト設定ファイルで定義したプロパティがすべて表示されています。




➤ **is_today_highlighted** 認識プロパティには値がありません。これは，その取得をまだ実装していないためです。その他のすべての認識プロパティについては，**WpfObject** の場合と同じように値が表示されます（それは，WpfObject が基本クラスであるためです）。

➤ エディタで，「MyWpfCalendar("MyCalendar").」と入力します。

ピリオドを入力すると，UFT のステートメント自動補完機能により，**MyWpfCalendar** テスト・オブジェクト・クラスで使用可能なすべての操作が表示されます。それには，**WpfObject** から継承した操作と，テスト・オブジェクト設定ファイルで定義した操作が含まれます。

# 基本カスタム・サーバの設計

サポートするカスタム・コントロールごとに，カスタム・サーバ・クラスを **CustomServerBase** クラスから派生して作成します。得られたカスタム・サーバ DLL は，アプリケーションのコンテキストで実行され，UFT とカスタム・コントロールとの間のインタフェースになります。

この項では，Calendar コントロールをサポートする CalendarSrv カスタム・サーバ・クラスを設計します。

1 **CalendarSrv.cs** を開きます。クラスの基本フレームワークは，［WPF Custom Server Setup］ダイアログ・ボックスで指定した仕様に基づいて作成されています。

- ➤ このクラスは，**CustomServerBase** を継承します。
- ➤ クラス定義の Using セクションには，WPF Add-in Extensibility API の **Mercury.QTP.WPF.CustomServer** 名前空間への参照が含まれています。

---

**注：UFT Silverlight CustomServer** テンプレートを使用する場合は，Silverlight Add-in Extensibility API の **Mercury.QTP.Slv.CustomServer** 名前空間への参照がクラスに含まれます。

---

- ➤ クラス定義には，実装するインタフェースのリスト（**IRecord**，**ICalendarRun**，**IComponentDetector**，**ICalendarProperties**）が含まれます。

2 プロジェクトに，**<WPF および Silverlight Add-in Extensibility SDK インストール・フォルダ>\samples\WPFExtCalendarSample\Application\WPFToolkit.dll** への参照を追加します。

これにより，Visual Studio ソリューション・エクスプローラで参照ノードをダブルクリックして，**Microsoft.Windows.Controls.Calendar** のメソッド，プロパティ，イベントにアクセスできるようになります。Calendar コントロールとのやり取りを行うコードを設計できるように，これらを十分に理解しておく必要があります。

3 **Calendar.cs** に using Microsoft.Windows.Controls; ステートメントを追加します。これにより，Visual Studio で **Microsoft.Windows.Controls.Calendar** コントロール・タイプの Microsoft IntelliSense が有効になります。

**4** CalendarSrv クラスに，カスタム Calendar オブジェクトへの参照を返す共通ヘルパ・プロパティを実装します。カスタム・コントロールのイベント，メソッド，プロパティにアクセスするために，カスタム・サーバ・コード全体で使用できます。

```
private Calendar MyCalendar
{
   get
   {
      return UtilityObject.ApplicationObject as Calendar;
   }
}
```

**5** ［WPF Custom Server Setup］ダイアログ・ボックスで，子オブジェクトの扱いをカスタマイズするよう指定しました。そのため，**IComponentDetector** インタフェースの **IsKnownPartOf** メソッドの事前実装が CalendarSrv クラスに作成されています。

**IsKnownPartOf** メソッドを修正して，常に true を返すようにします。これにより，UFT は Calendar 内のすべての子オブジェクトを，独立したオブジェクトではなく，カレンダの一部として扱うことになります。

# 認識プロパティ値を取得するためのサポートの実装

この項では，プロパティ値の取得インタフェースを CalendarSrv クラスに実装して，Calendar コントロールから認識プロパティ値を取得する処理をサポートします。

［WPF Custom Server Setup］ダイアログ・ボックスで，プロパティの取得をカスタマイズするよう指定しました。そのため，サンプル・プロパティの **MyCustomProperty** に対して，指定した **ICalendarProperties** インタフェースが **CalendarSrv.cs** ファイルで定義され，**CustomPropInterface** 属性が付けられ，**CalendarSrv** クラスに実装されています。

**1** **CalendarSrv.cs** ファイルで **ICalendarProperties** インタフェースの定義を探します。

```
[CustomPropInterface()]
   public interface ICalendarProperties
   {
      object MyCustomProperty
      {
         get;
      }
   }
```

**2** サンプルの object MyCustomProperty を bool is_today_highlighted に変更して，インタフェース定義を完成させます。

**3** CalendarSrv クラスでインタフェースの実装を探します。

```
public object MyCustomProperty
{
   get
   {
      return null;
   }
}
```

4 サンプルの実装を修正して，**is_today_highlighted** 認識プロパティの値を取得するようにします。

```
public bool is_today_highlighted
{
   get
   {
      return MyCalendar.IsTodayHighlighted;
   }
}
```

## 基本カスタム・サーバおよび認識プロパティ・サポートのデプロイとテスト

この項では，Calendar コントロールをサポートするために開発したカスタム・サーバをデプロイし，UFT に対する効果をテストします。

1 ソリューションをビルドし，**CalendarSrv.dll** ファイルを **<UFT インストール・フォルダ>\dat\Extensibility\WPF\MyWpfToolkit** フォルダにコピーして，カスタム・サーバをデプロイします。XML ファイルは変更していないため，デプロイの必要はありません。

2 サンプルのコントロールを実行します。これは，**<WPF および Silverlight Add-in Extensibility SDK インストール・フォルダ>\samples\WPFExtCalendarSample\Application\WpfCalendar.exe** ファイルを開くことで行います。

---

**注：**設定ファイルは変更していないため，UFT のインスタンスが開いていればそれを使用できます。ただし，Calendar アプリケーションが開いている場合は，閉じてから再度起動する必要があります。

---

**3** オブジェクト・スパイを使用して，UFT が Calendar コントロールとその子をどのように認識するかを確認し，その認識プロパティを表示します。

- ➤ カレンダは，**MyWpfCalendar** テスト・オブジェクト・クラスで表されています。
- ➤ カレンダの日にちの数字は，Calendar コントロールの一部とみなされ，別の WpfButton テスト・オブジェクトとしては表されていません。
- ➤ **is_today_highlighted** プロパティの値が表示されています。
- ➤ ［Selected Date］ボックスは Calendar コントロールの外にあり，設計どおりに，まだ別の WpfEdit テスト・オブジェクトによって表されています。

**4** ［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**］ボタンを使用して，Calendar コントロールを学習します。**calendar1** という名前の **MyWpfCalendar** テスト・オブジェクトがオブジェクト・リポジトリに追加されます。

**5** 記録セッションを開始し，**calendar1** テスト・オブジェクトの **is_today_highlighted** プロパティの値をチェックするチェックポイントを作成します。記録セッションを停止し，ステップを実行して，プロパティ値が適切に取得されていることを確認します。

**6** キーワード・ビューで，**calendar1** テスト・オブジェクトを使用するテスト・ステップを作成します。標準設定の **SetDate** 操作が自動的に選択されます。日付を（mm/dd/yyyy の書式で）［**引数**］カラムに入力します。

**7** テストを実行します。テスト・オブジェクト・メソッド実行のサポートを実装していないので，実行時エラーが発生します。このサポートの実装は，次の項で行います。

# テスト・オブジェクト操作の実行に対するサポートの実装

この項では，実行インタフェースを CalendarSrv クラスに実装して，Calendar コントロールでのテスト・オブジェクト操作の実行をサポートします。

［WPF Custom Server Setup］ダイアログ・ボックスで，操作の実行をカスタマイズするよう指定しました。そのため，サンプル操作の **MyRunMethod** 対して，指定した **ICalendarRun** インタフェースが **CalendarSrv.cs** ファイルで定義され, **RunInterface** 属性が付けられ，**CalendarSrv** クラスに実装されています。

**1** **CalendarSrv.cs** ファイルで **ICalendarRun** インタフェースの定義を探します。

```
[RunInterface()]
public interface ICalendarRun
{
   void MyRunMethod();
}
```

**2** サンプルの void MyRunMethod(); を次の行のように変更して，サポートするすべての操作を含むようにインタフェース定義を完成させます。

```
void SetDate(string date);
void Prev();
void Next();
string SelectedDate
{
   get;
}
```

**3** CalendarSrv クラスでインタフェースの実装を探します。

```
public void MyRunMethod()
{
}
```

**4** サンプルの **MyRunMethod()** の例を，次の Calendar 特有のメソッドとプロパティの実装に置き換えます。

```
 public void SetDate(String date)
{
   MyCalendar.SelectedDate = DateTime.Parse(date);
   MyCalendar.DisplayDate = DateTime.Parse(date);
}
```

```
public string SelectedDate
{
   get
   {
      return MyCalendar.SelectedDate.Value.ToShortDateString();
   }
}
```

```
public void Prev()
{
   Button prev = GetDescendantByName(UtilityObject.ApplicationObject,
"PART_PreviousButton") as Button;
   RaiseButtonClickEvent(prev);
}
```

```
public void Next()
{
   Button next = GetDescendantByName(UtilityObject.ApplicationObject,
"PART_NextButton") as Button;
   RaiseButtonClickEvent(next);
}
```

---

**注 :** Silverlight コントロールのサポートを開発している場合は，これらの各メソッドに Microsoft Silverlight の **ScriptableMember** 属性を付けます。

---

**5** using System.Windows.Media; ステートメントを CalendarSrv.cs ファイルに追加してから，次のヘルパ関数を追加します。

```
private void RaiseButtonClickEvent(Button button)
{
   if (button != null)
   {
      RoutedEvent e = Button.ClickEvent;
      RoutedEventArgs arg = new RoutedEventArgs();
      arg.RoutedEvent = e;
      button.RaiseEvent(arg);
   }
}
```

```
private DependencyObject GetDescendantByName(DependencyObject parent,
string name)
{
   if (parent == null)
      return null;
   int count = VisualTreeHelper.GetChildrenCount(parent);
   for (int i = 0; i < count; i++)
   {
      DependencyObject child = VisualTreeHelper.GetChild(parent, i);
      if (child is FrameworkElement)
      {
         if ((child as FrameworkElement).Name == name)
         return child;
      }
      if (child is FrameworkContentElement)
      {
         if ((child as FrameworkContentElement).Name == name)
            return child;
      }
      child = GetDescendantByName(child, name);
      if (child != null)
         return child;
   }
   return null;
}
```

## テスト・オブジェクト操作のサポートのデプロイとテスト

この項では，カスタム・サーバをもう一度デプロイし，テスト・オブジェクト操作の実行のために設計したサポートをテストします。

1 ソリューションをビルドしてから **CalendarSrv.dll** ファイルを **<UFT インストール・フォルダ>\dat\Extensibility\WPF\MyWpfToolkit** フォルダにコピーして，カスタム・サーバをデプロイします。

2 サンプルのコントロールを実行します。これは，**<WPF および Silverlight Add-in Extensibility SDK インストール・フォルダ>\samples\WPFExtCalendarSample\Application\WpfCalendar.exe** ファイルを開くことで行います。

3 ［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**］ボタンを使用して，Calendar コントロールを学習します。**calender1** という名前の **MyWpfCalendar** テスト・オブジェクトがオブジェクト・リポジトリに追加されます。

4 **calendar1** テスト・オブジェクトと，各テスト・オブジェクト操作（**SetDate (mm/dd/yyyy)**，**SelectedDate**，**Next**，**Prev**）を使用してテスト・ステップを作成します（SelectedDate プロパティから返される値を表示するには，**msgBox** ステートメントを使用できます）。

5 テストを実行し，操作が正しく実行されることを確認します。

## 記録のサポートの実装

この項では，**IRecord** インタフェースを実装し，カスタム・コントロールに関するステップの記録をサポートする，イベントおよびメッセージのハンドラ・メソッドを作成します。

WPF カスタム・カレンダでは，**Next**，**Prev**，**SetDate** の 3 種類のステップを記録する必要があります。

- **Next** および **Prev** は，ユーザが Calendar コントロールの左右の矢印をクリックしたときに，ウィンドウ・メッセージに反応して記録されます。これは，**OnMessage** メソッドで処理されます。
- **SetDate** は，コントロール・イベントの **SelectedDatesChanged** に反応して記録されます。これは，関連するコントロール・イベントを処理するように作成し，登録したイベント・ハンドラによって処理されます。

**カスタム・サーバ・クラスに記録のサポートを実装するには，次の手順を実行します。**

**1** CalendarSrv class で，IRecord インタフェースを実装するためのセクションを探します。

```
public void OnMessage(DependencyObject src, int msg, int wParam, int lParam)
{
}
public void RecordInit()
{
}
public void RecordStop()
{
}
```

**2** イベント・ハンドラを宣言し，**RecordInit** を実装してコントロールに登録します。

```
private EventHandler<SelectionChangedEventArgs> _h;

public void RecordInit()
{
   _h = new EventHandler<SelectionChangedEventArgs>
         (OnSelectedDatesChanged);
   UtilityObject.AddHandler(MyCalendar, "SelectedDatesChanged", _h);
}
```

---

**注 :** Silverlight コントロールのサポートを開発している場合は，**AddHandler** の構文が異なります。詳細については，『Custom Server API Reference』（WPF および Silverlight Add in Extensibility ヘルプで利用可能）の **Mercury.QTP.Slv.CustomServer** 名前空間を参照してください。

---

**3** イベント・ハンドラの登録に SDK の **AddHandler** メソッドを使用したため，**RecordStop** の実装は必要ありません。このメソッドを使用したことで，UFT は記録セッションの終了時にイベント・ハンドラを自動的に削除できます。

4 **OnSelectedDatesChanged** イベント・ハンドラの実装を追加します。

```
private void OnSelectedDatesChanged(object sender,
System.Windows.Controls.SelectionChangedEventArgs e)
{
   UtilityObject.Record("SetDate", RecordingMode.RECORD_SEND_LINE,
     MyCalendar.SelectedDate.Value.ToShortDateString());
}
```

これにより，記録セッション時にユーザが選択した新しい日付で **SetDate** ステップが作成されます。

5 **OnMessage** メソッドを次のように実装して，**Prev** および **Next** 操作の記録をサポートします。

```
public void OnMessage(DependencyObject o, int msg, int wParam, int lParam)
{
   if(o is Button && msg == 0x201) // WM_LBUTTONDOWN
   {
      string name = (o as Button).Name;
      switch (name)
      {
         case "PART_NextButton":
               base.UtilityObject.Record("Next",
                RecordingMode.RECORD_SEND_LINE, null);
         break;
         case "PART_PreviousButton":
               base.UtilityObject.Record("Prev",
                RecordingMode.RECORD_SEND_LINE, null);
         break;
      }
   }
}
```

---

**注 :** Silverlight コントロールのサポートを開発している場合，**OnMessage** メソッドは **RECORD_HANDLED** を返します。これは，このメッセージがカスタム・サーバで処理されたので，ほかのイベント・ハンドラに渡す必要がないことを示します。詳細については，『Custom Server API Reference』（WPF および Silverlight Add in Extensibility ヘルプで利用可能）の **Mercury.QTP.Slv.CustomServer** 名前空間を参照してください。

---

### 6 この手順は, Silverlight コントロールのサポートを開発している場合にのみ必要です。

**GetWndMessageFilter** メソッドを実装して，カスタム・サーバで処理するウィンドウ・メッセージのレベルを指定します。

```
CTL_MsgFilter GetWndMessageFilter()
{
   return CTL_MsgFilter.CTL_MSGS;
}
```

このチュートリアルでは，コントロールのすべての子が，コントロールの一部とみなされます。したがって，CTL_MSGS を返して，このコントロールに向けられたメッセージのみを処理するだけで十分です。

コントロールの子の中に別のテスト・オブジェクトとして扱われるものがあり，その子で発生したイベントもコントロールに処理させる場合は，CHILD_MSGS を返すように **GetWndMessageFilter** を実装します。

# 記録のサポートのデプロイとテスト

これで，WPF Calendar コントロールのサポートの設計が完了しました。

この項では，カスタム・サーバをもう一度デプロイし，Calendar オブジェクトでの操作を記録するために設計したサポートをテストします。

1 ソリューションをビルドしてから **CalendarSrv.dll** ファイルを **<UFT インストール・フォルダ>\dat\Extensibility\WPF\MyWpfToolkit** フォルダにコピーして，カスタム・サーバをデプロイします。

2 サンプルのコントロールを実行します。これは，**<WPF および Silverlight Add-in Extensibility SDK インストール・フォルダ>\samples\WPFExtCalendarSample\Application\WpfCalendar.exe** ファイルを開くことで行います。

3 記録のために開発したサポートが正しく動作することをテストするため，記録セッションを開始し，カレンダで日を選択し，カレンダ上部の右矢印をクリックし，カレンダ上部の左矢印をクリックします。**SetDate**，**Next**，**Previous** ステップが記録されるのを確認します。

# 第 3 章

# ツールキット・サポート・セットのデプロイ

カスタム・ツールキットの UFT サポートを拡張する作業で最後に行う手順が，ツールキット・サポート・セットのデプロイです。この手順では，作成したすべてのファイルを，UFT がインストールされたコンピュータ上の正しい場所に配置します。これにより，UFT はツールキット内のコントロールを識別し，コントロールに関するテストを実行できるようになります。

ツールキット・サポート・セットの開発作業では，ツールキット・サポート・セットを UFT にデプロイすることによって，作成したサポートのテストとデバッグを実行できます。完成したツールキット・サポート・セットを，UFT がインストールされているコンピュータにデプロイすると，WPF または Silverlight Add-in を拡張できます。

**本章の内容**



## カスタム・ツールキット・サポートのデプロイについて

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

UFT が起動するとアドイン・マネージャが開き，ツールキット・サポート・セットの環境名が WPF Add-in ノードの下の子ノードとして表示されます。環境のチェック・ボックスを選択すると，開発したツールキット・サポート・セットを使用して，その環境のサポートがロードされます。

**注：** Silverlight Add-in Extensibility を使用して開発したツールキット・サポート・セットでは Silverlight Add-in が必要になります。したがって，このようなツールキット・サポート・セットの環境を選択する場合には，Silverlight Add-in も選択してください。

環境のサポートがロードされると：

- UFT は，環境内のコントロールを認識し，そのコントロールに関するテストを実行できるようになります。
- UFT は，アドインまたはサポートされる環境のリストを表示するすべてのダイアログ・ボックスに，環境の名前を表示します。
- 各アドインで使用可能なテスト・オブジェクト・クラスをリスト表示するダイアログ・ボックス（たとえば，［新規テストオブジェクトの定義］ダイアログ・ボックスや［オブジェクトの認識］ダイアログ・ボックスなど）では，ツールキット・サポート・セットで定義したテスト・オブジェクト・クラスの一覧が表示されます。

## カスタム・ツールキット・サポートのデプロイ

作成したツールキット・サポート・セットをデプロイするには，UFT インストール・フォルダ内の特定の場所にファイルを格納する必要があります。

**注：** 作業を始める前に，**<UFT インストール・フォルダ>\dat\Extensibility\WPF**（または **...\Slv**) フォルダ）にカスタム・ツールキットと同じ名前のフォルダを作成します。

ツールキット・サポート・ファイルを配置する場所を次の表にまとめます。

| ファイル名 | 場所 |
|---|---|
| **<カスタム・ツールキット名>TestObjects.xml**<br><br>**注：**上記は，ファイル名で推奨される命名規則です。テスト・オブジェクト設定 XML ファイルを複数作成して，それぞれに任意の名前を付けることができます。 | **WPF のサポートのデプロイ：**<br>➤ <UFT インストール・フォルダ>\dat\Extensibility\WPF<br>➤ <ALM 用 UFT アドインのインストール・フォルダ>\dat\Extensibility\WPF（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必要）<br>**Silverlight のサポートのデプロイ：**<br>➤ <UFT インストール・フォルダ>\dat\Extensibility\Slv<br>➤ <ALM 用 UFT アドインのインストール・フォルダ>\dat\Extensibility\Slv（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必要） |
| **<カスタム・ツールキット名>.cfg** | **WPF のサポートのデプロイ：**<br><UFT インストール・フォルダ>\dat\Extensibility\WPF\<カスタム・ツールキット名><br>**Silverlight のサポートのデプロイ：**<br><UFT インストール・フォルダ>\dat\Extensibility\Slv\<カスタム・ツールキット名> |

| ファイル名 | 場所 |
| --- | --- |
| **カスタム・サーバ DLL** | この **.dll** ファイルは，UFT がインストールされているコンピュータまたはネットワーク上のアクセス可能な場所に配置できます。<br>ファイルの場所は，**<カスタム・ツールキット名>.cfg** 内の DllName 要素で指定します。 |
| **新しいテスト・オブジェクト・クラスのアイコン・ファイル**（任意） | **.dll**，**.exe** または **.ico** ファイルのいずれかを，UFT がインストールされているコンピュータまたはネットワーク上のアクセス可能な場所に格納します。<br>ファイルの場所は，**<カスタム・ツールキット名>**TestObjects.xml で指定します。 |
| **テスト・オブジェクト・クラスのヘルプ・ファイル**（任意） | **.chm** ファイルを，UFT がインストールされているコンピュータに格納します。<br>ファイルの場所は，**<カスタム・ツールキット名>**TestObjects.xml で指定します。 |

## 推奨されるファイルの格納場所

カスタム・サーバ DLL ファイル，ヘルプ・ファイル，アイコン・ファイルの場所は，ツールキット・サポート・セットの設定ファイルで指定します。場所の指定には，相対パスを使用できます。詳細については，Test Object Schema ヘルプと Toolkit Configuration Schema ヘルプ（WPF および Silverlight Add-in Extensibility ヘルプで利用可能）を参照してください。

各ファイルの格納場所として推奨される場所を次の表にまとめます。

| ファイル名 | 場所 |
|---|---|
| **カスタム・サーバ DLL ファイル** | **WPF のサポートのデプロイ**：<br><UFT インストール・フォルダ>\dat\Extensibility\WPF\<カスタム・ツールキット名>\CustomServers<br>**Silverlight のサポートのデプロイ**：<br><UFT インストール・フォルダ>\dat\Extensibility\Slv\<カスタム・ツールキット名>\CustomServers |
| **アイコン・ファイル** | **WPF のサポートのデプロイ**：<br><UFT インストール・フォルダ>\dat\Extensibility\WPF\<カスタム・ツールキット名>\Res<br>**Silverlight のサポートのデプロイメント**：<br><UFT インストール・フォルダ>\dat\Extensibility\Slv\<カスタム・ツールキット名>\Res |
| **ヘルプ・ファイル** | **WPF のサポートのデプロイ**：<br><UFT インストール・フォルダ>\dat\Extensibility\WPF\<カスタム・ツールキット名>\Help<br>**Silverlight のサポートのデプロイメント**：<br><UFT インストール・フォルダ>\dat\Extensibility\Slv\<カスタム・ツールキット名>\Help |

### DevelopmentMode 属性の設定

➤ テスト・オブジェクト設定ファイルの**認識プロパティ**要素を変更する場合，カスタム・ツールキット・サポートの設計中は，**TypeInformation** 要素の **DevelopmentMode** 属性の値を true にしてください。通常の用途にカスタム・ツールキット・サポート・セットをデプロイする場合は，この属性を削除（または false に設定）してから作業を開始してください。詳細については，84 ページ「テスト・オブジェクト設定ファイルで指定された IdentificationProperty 要素の属性を変更」を参照してください。

## デプロイ済みサポートの変更

デデプロイ済みのツールキット・サポート・セットを変更する場合，UFT を再度開いてから WPF または Silverlight アプリケーションを再起動すると，変更内容が有効になります。

UFT で使用する認識プロパティの機能に関する定義を変更する場合は，次の「テスト・オブジェクト設定ファイルで指定された IdentificationProperty 要素の属性を変更」を参照してください。

### テスト・オブジェクト設定ファイルで指定された IdentificationProperty 要素の属性を変更

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

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

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

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

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

### 認識プロパティの属性の変更に関する考慮事項

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

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

# デプロイ済みサポートの削除

UFT では，特定の環境またはツールキット向けに提供されているサポートを起動時にロードするかどうかをアドイン・マネージャで指定できます。

デプロイの完了後，カスタム・ツールキットのサポートを UFT から削除するには，次のツールキット設定ファイルをカスタム・ツールキットのフォルダから削除する必要があります。
**<UFT インストール・フォルダ>\dat\Extensibility\WPF**（または **...\Slv**）

テスト・オブジェクト設定ファイル内に，カスタム・コントロールに対応するテスト・オブジェクト・クラス定義が存在しない場合（つまりファイルが不要），次の場所からファイルを削除できます。
**<UFT インストール・フォルダ>\dat\Extensibility\WPF**（または **...\Slv**），および **<ALM 用 UFT アドインのインストール・フォルダ>\dat\Extensibility\WPF**（または **...\Slv**）（該当する場合）