VERSION 8.2

拡　張　ガ　イ　ド

MERCURY™

Mercury QuickTest Professional
.NET アドイン
拡張ガイド
Version 8.2

MERCURY™

QuickTest Professional .NET アドイン・ガイド，Version 8.2

Mercury Interactive Corporation
379 North Whisman Road
Mountain View, CA 94043
Tel: (650) 603-5200
Toll Free: (800) TEST-911
Customer Support: (877) TEST-HLP
Fax: (650) 603-5300

本書に関するご意見，ご要望は documentation@mercury.com まで電子メールにてお送りください。

QTPNETEXTGD8.2JP/01

目次

ようこそ

QuickTest Professional .NET アドインへようこそ。

QuickTest Professional .NET アドイン拡張を使用すると，標準の QuickTest Professional .NET アドインではサポートされていないサードパーティ製の .NET コントロールおよびカスタムの .NET コントロールを使用するテスト・アプリケーションがサポートされるようになります。

本書の読者対象

本書 は，プログラマ，システム・アナリスト，システム設計者，技術責任者を対象としています。

本書を使用するには，次の知識が必要です。

- QuickTest Professional オブジェクト・モデル
- QuickTest Professional .NET アドイン
- XML（基本的な知識）
- C# での .NET プログラミング

本書の使い方

本書では，QuickTest Professional .NET アドイン拡張を使用して，サードパーティ製のコントロールやカスタム・コントロールに対する Test Run および Test Record のサポートを拡張するのに必要な事項について説明します。Test Record は，テスト対象アプリケーションに対する操作とその操作に対するアプリケーションの動作結果の記録，そしてその記録のテスト・スクリプトへの変換を行うセッション内で使用するソフトウェア・モジュールです。Test Run ソフトウェア・モジュールは，このスクリプトを実行して，アプリケーションが要求どおりに動作しているかどうかをテストするために結果を追跡するのに使用します。

本書は，「**QuickTest Professional .NET アドイン拡張 API リファレンス**」（オンライン・ヘルプ形式）と併せて使用してください。

これらのドキュメントは，『**QuickTest Professional ユーザーズ・ガイド**』，『**QuickTest Professional .NET アドイン・ガイド**』，『**QuickTest Professional Object Model Reference**』と併せて使用してください。これらすべてのドキュメントのオンライン版には，QuickTest のメイン画面から，［**ヘルプ**］＞［**QuickTest Professional ヘルプ**］を選択してアクセスします。これらのドキュメントは，印刷したものも用意されています。

本書は，次の章で構成されています。

第 1 章　QuickTest Professional .NET アドイン拡張の概要

カスタムの .NET コントロールのサポート拡張の概念について説明します。

第 2 章　Custom Server C# プロジェクト・テンプレートのインストール

.NET アドイン拡張モジュールのインストール方法と，この拡張機能を使用するための QuickTest Professional .NET アドイン・プロジェクトの設定方法について説明します。

第 3 章　.NET DLL の使用によるカスタム・コントロールのサポートの拡張

.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する方法について説明します。

表記規則

本書では，次の表記規則に従います。

1, 2, 3	太字の数字は , 手順の順番を示します。
＞	大なり記号はメニュー・レベルを区切ります（例：［**ファイル**］＞［**開く**］）。
［**太字**］	全角の大括弧に**太字**は，インターフェイスの要素の名前（例：［**実行**］ボタン）やその他の強調する項目を示します。
太字	**太字**のテキストは，メソッド名または関数名を示します。また，メソッドまたは関数の引数，構文の記述中のファイル名，書名を示します。新しい用語を紹介する場合にも使用します。
＜＞	山括弧は，ユーザの環境次第で変わる可能性のある，ファイル・パスや URL アドレスの一部分を囲みます（例：**＜製品のインストール・フォルダ＞ ¥bin**）。
Arial	Arial のフォントはそのまま入力する例やテキストに使用されます。
Arial bold	**Arial bold** のフォントはそのまま入力しなければならない構文記述のテキストに使用されます。
SMALL CAPS	SMALL CAPS のフォントは，キーボードのキーを示します。
...	構文内の 3 つの点は，同じ形式で項目をさらに含めることができることを意味します。プログラム例での 3 つの点は，プログラム行が意図的に削除されていることを示します。
[ ]	半角の大括弧は，省略可能な引数を囲みます。
|	2 つの値のうちの 1 つを選択しなければならない場合，これらの値を垂直バーで区切ります。

第 1 章

QuickTest Professional .NET アドイン拡張の概要

QuickTest Professional .NET アドイン拡張へようこそ

QuickTest Professional .NET アドイン拡張によって，標準の QuickTest Professional .NET アドインではサポートされていないサードパーティ製の .NET コントロールおよびカスタムの .NET コントロールに対する高度なサポートが可能になります。

拡張モジュールを使用せずに標準の QuickTest Professional .NET アドインではサポートされない .NET コントロールでのテストを記録することは可能です。しかし，記録されたスクリプトには，Windows メッセージとして渡される低レベルの動作が含まれることになります。拡張モジュールを使用して .NET コントロールをサポートすることによって，標準の低レベル・サポートが拡張され，スクリプトが意味の分かりやすい，修正も簡単に行えるものになります。

本章では，次の項目について説明します。

- .NET アドイン拡張の概要
- コーディング手法の選択肢の概要（.NET DLL および XML）
- Custom Server ランタイム・コンテキストの概要
- テスト・オブジェクト・マッピングの概要

.NET アドイン拡張の概要

QuickTest Professional .NET アドイン拡張は，サードパーティ製やカスタムのこれらの .NET コントロールの有意動作を表すメソッドを使用して QuickTest テスト・オブジェクトを拡張することで，.NET コントロールのサポートを可能にします。

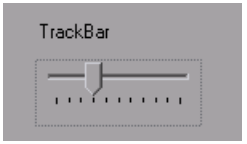
拡張モジュールを付け加えていない QuickTest Professional .NET アドインでも，多くの .NET コントロールが標準でサポートされています。.NET アドインでは，これらのコントロールの有意動作を表すメソッドが実装された，テスト・オブジェクトが提供されます。

拡張モジュールは，追加の .NET コントロールに対して，同じレベルのサポートを実装することを可能にします。拡張モジュールを使用して，既存のメソッドを変更して新たなメソッドを定義することで .NET アドイン・インタフェースを拡張し，Custom Server を作成します。カスタム・コントロールを既存の QuickTest テスト・オブジェクトにマッピングすれば，IntelliSense での表示と，ビジネス・コンポーネントまたはテスト・スクリプトでのステップの説明を含め，QuickTest テスト・オブジェクトのすべての機能を使用できるようになります。

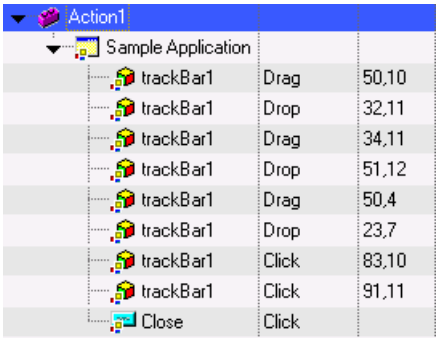
有意動作の概念について

コントロールの有意動作とは，テストの対象となる動作です。例えば，アプリケーションでラジオ・ボタン・グループのボタンをクリックするとき，**Click** イベントやクリックの座標ではなく，選択されている値が関心の対象です。したがって，ラジオ・ボタン・グループの有意動作は，選択対象の変更です。

コントロールに対するサポートを拡張せずに，カスタム・コントロールを対象と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，コントロールの低レベルの動作が記録されます。例えば，次に示すサンプル .NET アプリケーションの TrackBar コントロールは，対応する QuickTest テスト・オブジェクトが存在しないコントロールです。



コントロールのサポートを実装せずに TrackBar を対象とした記録を行うと，キーワード・ビューには次のように表示されます。

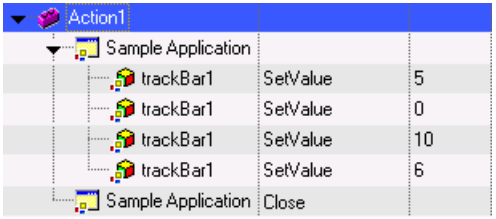


エキスパート・ビューでは，記録されたテストが次のように表示されます。

```
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
```

記録されるメソッドは，表示されるコントロールの特定の座標における，Drag，Drop および Click（TrackBar コントロールの低レベルの動作）であることに注意してください。このようなステップは，分かりにくく修正も困難です。

.NET アドイン拡張を使用して TrackBar コントロールをサポートする場合，結果はより意味のあるものになります。次に，Custom Server を使い TrackBar を対象に記録を行ったテストのキーワード・ビューを示します。



エキスパート・ビューでは，記録されたテストが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
```

QuickTest は，カスタム・テスト・オブジェクトを使用しない場合に記録する低レベルの Drag，Drop および Click 操作の代わりに，新しいスライダの位置を示す SetValue 操作を記録しています。このようなテスト・スクリプトであれば理解と修正は簡単です。

コーディング手法の選択肢の概要（.NET DLL および XML）

QuickTest カスタム・サポートの実装には，次の 2 つの方法があります。

- **.NET DLL**：.NET アセンブリを使用してコントロールに対するサポートを拡張します。
- **XML**：XML ファイルを使用しているコントロールに対するサポートを拡張します。

コーディング手法選択のガイドライン

ほとんどの Custom Server は，.NET DLL として実装されます。構文のチェック，デバッグ，Microsoft IntelliSense など，プログラム開発環境のすべてのサービスによって開発がサポートされるため，この選択肢が一般的に使用されます。さらに，.NET DLL として実装される Custom Server は，Test Record 機能の一部を **QuickTest** コンテキスト，一部を **Application under test** コンテキストにおいて実行できます。詳細については，13 ページ「.NET DLL の使用によるカスタム・コントロールのサポートの拡張」，および『**QuickTest Professional .NET アドイン拡張 API リファレンス**』を参照してください。

ランタイム・コンテキストの詳細については，5 ページ「Custom Server ランタイム・コンテキストの概要」を参照してください。

十分な情報がある比較的シンプルなコントロール，あるいは既存のオブジェクトに適格にマッピングが可能であるにもかかわらず，Test Record の実装と置き換える必要があるか，テスト・オブジェクトの少数の Test Run メソッドを置き換えるか追加する必要がある場合には，XML を使用した実装が最も実用的です。テキスト・エディタがあればよいため，完全なプログラミング環境を使用できない場合にも有用です。

ただし，XML を使用してカスタム・コントロールを実装する場合は，プログラム開発環境によって提供されるサポートはありません。XML 実装は，**Application under test** コンテキストにおいてのみ実行可能です。詳細については，29 ページ「XML ファイルの使用による カスタム・コントロールのサポートの拡張」を参照してください。

コーディング・オプション設定の詳細については，35 ページ「Custom Server を使用するための QuickTest の設定」を参照してください。

Custom Server ランタイム・コンテキストの概要

Custom Server によって提供されるクラスは，**ランタイム・コンテキスト**と呼ばれる 2 つのソフトウェア・プロセスの 1 つの中でインスタンス化されます。

- **Application under test**
- **QuickTest**

Application under test コンテキスト内に作成されるオブジェクトは，.NET コントロールのイベント，メソッド，およびプロパティに直接アクセスできます。しかし，Windows メッセージをリッスンすることはできません。

QuickTest コンテキスト内に作成されるオブジェクトは，Windows メッセージをリッスンできます。しかし，.NET コントロールのイベント，メソッド，およびプロパティには直接アクセスできません。

Custom Server を .NET DLL として実装した場合には，**QuickTest** コンテキスト内に作成されるオブジェクトは，**Application under test** コンテキスト内で実行される Assistant オブジェクトを作成できます。

Custom Server ランタイム・コンテキスト選択のガイドライン

Custom Server は，Test Record と Test Run のどちらか，または両方を実装できます。Test Record は，テスト対象アプリケーションに対する操作と，アプリケーションの動作結果の記録，そしてその記録のテスト・スクリプトへの変換を行うセッション内で使用するソフトウェア・モジュールです。Test Run は，このスクリプトを実行してアプリケーションが要式どおりに動作しているかどうかをテストするために結果を追跡するソフトウェア・モジュールです。

Test Run は，ほとんどが **Application under test** コンテキスト内で実装されます。コントロールへの直接アクセスにより，値の設定とコントロールのメソッドの呼び出しを簡単に行うことができます。この場合，Test Run セッション中に Windows メッセージをリッスンする必要がないため，**QuickTest** コンテキストは要求されません。ただし，使用するアプリケーションがカスタム・コントロールのサービスよりも QuickTest サービスを多用する場合，**QuickTest** コンテキスト内に Test Run を実装する方がより効率的な場合があります。

Test Record のプログラミングは，一般に，**Application under test** コンテキストにおいて行うほうが簡単です。しかし，記録に Windows メッセージの使用が不可欠である場合，**QuickTest** コンテキストを使用する必要があります。

.NET DLL Custom Server が，Windows メッセージのリッスンと，コントロールのイベントおよびプロパティへのアクセスの両方を行う必要がある場合は，Assistant クラスを使用します。**QuickTest** コンテキスト内で動作する Custom Server は，**Application under test** コンテキスト内で実行する Assistant クラス・オブジェクトを使用して，**Application under test** コンテキスト内のイベントをリッスンできます。これらのオブジェクトは，コントロール・プロパティへの直接アクセスも提供します。

詳細については，19 ページ「カスタム・コントロールに対する .NET DLL を使用した Test Record の実装」を参照してください。

Assistant クラスの詳細については，13 ページ「.NET DLL の使用によるカスタム・コントロールのサポートの拡張」，および『**QuickTest Professional .NET アドイン拡張 API リファレンス**』を参照してください。

コンテキスト設定の詳細については，35 ページ「Custom Server を使用するための QuickTest の設定」を参照してください。

テスト・オブジェクト・マッピングの概要

すべての Custom Server は，親 QuickTest テスト・オブジェクトにマッピングされます。テスト・オブジェクトがカスタム・コントロールに適用されると，Custom Server によって親テスト・オブジェクトが拡張されます。

類似の機能の QuickTest テスト・オブジェクトに Custom Server をマッピングする場合，カスタム・オブジェクトに対して変更なしに適用が可能な親オブジェクトの Test Run メソッドを変更する必要はありません。例えば，ほとんどのコントロールは Click メソッドを持っています。親オブジェクトの Click メソッドによって，カスタム・オブジェクトの Click メソッドが適切に実装されていれば，親のメソッドをオーバライドする必要はありません。

カスタム・オブジェクトの Test Run 機能のうち親にない機能を含めるには，Custom Server に新しいメソッドを追加します。同じメソッド名を持つものの異なる実装である機能を含めるには，親メソッドをオーバライドします。カスタム・コントロールをサポートするテスト・オブジェクト型は，親オブジェクトの Test Run メンバまたはそれらのメンバをオーバライドしたもの，および Custom Server によって追加された新しいメンバで構成される新しい型です。

プログラミングが不要でマッピングだけで済む場合があります。親 QuickTest テスト・オブジェクトがコントロールに適切に対応していれば，QuickTest テスト・オブジェクトにコントロールをマッピングするだけで十分です。QuickTest テスト・オブジェクトが Test Record に適切に対応しているものの，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，親オブジェクトの実装が置き換えられます。要求される Test Record 機能をすべて実装する必要があります。

マッピングを指定しない場合，QuickTest によって，標準設定の汎用テスト・オブジェクトである **SwfObject** にカスタム・コントロールがマッピングされます。

カスタム・テスト・オブジェクトを参照するスクリプト行を編集する場合，Microsoft IntelliSense により，親 QuickTest テスト・オブジェクトのプロパティとメソッドに加えて，カスタム・テスト・オブジェクトのプロパティとメソッドが表示されます。

マッピングの詳細については，35 ページ「Custom Server を使用するための QuickTest の設定」を参照してください。

第 2 章

Custom Server C# プロジェクト・テンプレートのインストール

本章では，Microsoft Visual Studio .NET の Custom Server C# プロジェクト・テンプレートのインストール方法について説明します。Visual Studio のサポートされているバージョンの最新の一覧については，インストール・フォルダにある QuickTest Professional .NET アドインの **Readme** ファイルを参照してください。

このインストールを行うと，Custom Server プロジェクト・テンプレートと，新規プロジェクトを作成するためにテンプレートが選択されると実行されるウィザードが提供されます。

Custom Server テンプレートでは，枠組みを示す中身のないコード，いくつかのサンプル・コード，および Custom Server をビルドするのに必要な QuickTest プロジェクト参照が提供されます。

ウィザードは，.NET アドイン拡張モジュールを使用して Custom Server .NET DLL を作成するための Visual Studio .NET プロジェクトのセット・アップを簡易化します。詳細については，14 ページ「.NET DLL の使用によるカスタム・コントロールのサポートの拡張について」を参照してください。

本章では，次の項目について説明します。

- インストールの前に
- インストール・プログラムの実行
- プロジェクト・テンプレートのアンインストール

インストールの前に

Custom Server C# プロジェクト・テンプレートのインストールに先立って，次に示す動作環境を確認してください。

- **InstWizard.msi** ファイルにアクセスできなくてはなりません。**InstWizard.msi** ファイルには，QuickTest Professional .NET アドインがインストールされているコンピュータから，あるいは QuickTest Professional .NET アドイン CD-ROM のルート・フォルダからアクセスできます。
- Microsoft Visual Studio .NET が，コンピュータにインストールされている必要があります。

インストール・プログラムの実行

InstWizard.msi ファイルは，QuickTest Professional .NET アドインがインストールされている環境および QuickTest Professional .NET アドイン CD-ROM に含まれています。

.NET アドインの Custom Server C# プロジェクト・テンプレートをインストールするには，次の手順を実行します。

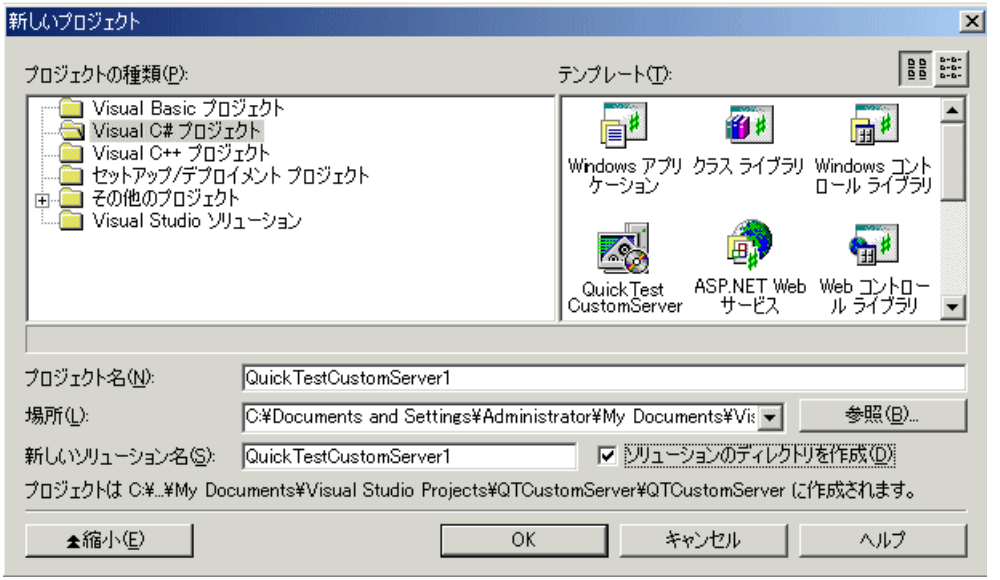
1. Microsoft Visual Studio .NET のインスタンスをすべて閉じます。
2. **InstWizard.msi** ファイルを見つけます。InstWizard.msi ファイルは，次のいずれかの場所にあります。
   - QuickTest Professional .NET アドインがインストールされているコンピュータの**＜ QuickTest Professional インストール・パス＞ ¥bin¥Custom** フォルダ。
   - QuickTest Professional .NET アドイン CD-ROM のルート・フォルダ
3. **InstWizard.msi** ファイルをダブルクリックして，インストールを実行します。コンピュータに Custom Server C# プロジェクト・テンプレートがインストールされます。

正しくインストールされたことを確認するには，次の手順を実行します。

1. Microsoft Visual Studio .NET を開きます。
2. ［**ファイル**］＞［**新規作成**］＞［**プロジェクト**］を選択して，［新しいプロジェクト］ダイアログ・ボックスを開きます。

3 [**プロジェクトの種類**] リストで [**Visual C# プロジェクト**] を選択します。

4 [**テンプレート**] 表示枠に [**QuickTest CustomServer**] テンプレート・アイコンが表示されるのを確認します。



プロジェクト・テンプレートのアンインストール

Windows のコントロール・パネルから Custom Server C# プロジェクト・テンプレートをアンインストールできます。

プロジェクト・テンプレートをアンインストールするには，次の手順を実行します。

1 [**スタート**] > [**設定**] > [**コントロール パネル**] > [**プログラムの追加と削除**] を選択します。[プログラムの追加と削除] ダイアログ・ボックスが開きます。

2 [**プログラムの追加と削除**] リストで，「**Mercury CustomWizard**」を選択します。

3 [**削除**] をクリックします。

第 3 章

.NET DLL の使用によるカスタム・コントロールのサポートの拡張

.NET DLL として実装された Custom Server を作成することによって，.NET コントロールをサポートできます。

.NET DLL Custom Server を作成するには，.NET アセンブリをプログラミングする方法を知っている必要があります。本章に示す図と説明は，開発環境に Microsoft Visual Studio .NET を使用しており，Custom Server C# 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想定しています。詳細については，9 ページ「Custom Server C# プロジェクト・テンプレートのインストール」を参照してください。

本章では，次の項目について説明します。

- .NET DLL の使用によるカスタム・コントロールのサポートの拡張について
- Custom Server の作成
- XML 設定セグメントの使用
- カスタム・コントロールに対する .NET DLL を使用した Test Record の実装
- カスタム・コントロールに対する .NET DLL を使用した Test Run の実装
- QuickTest コンテキストからの Application Under Test の下のコードの実行
- API の概要

.NET DLL の使用によるカスタム・コントロールのサポートの拡張について

Custom Server を作成して，カスタムの .NET コントロールに対し高度なサポートを実装できます。Custom Server は，Test Record および Test Run のインタフェースと一般的なユーティリティを実装する .NET DLL クラス・ライブラリです。詳細については，19 ページ「カスタム・コントロールに対する .NET DLL を使用した Test Record の実装」，24 ページ「カスタム・コントロールに対する .NET DLL を使用した Test Run の実装」，および 26 ページ「API の概要」を参照してください。

Custom Server の作成後，QuickTest を設定して使用します。詳細については，35 ページ「Custom Server を使用するための QuickTest の設定」を参照してください。

Custom Server の作成

Custom Server を作成するには，Microsoft Visual Studio .NET で .NET プロジェクトを設定し，QuickTest Test Record および Test Run 用のサポートをコーディングし，QuickTest が Custom Server をロードするよう設定ファイルを編集します。

.NET プロジェクトの設定

Microsoft Visual Studio .NET で .NET プロジェクトを設定するには，Custom Server C# プロジェクト・テンプレートを使用します。

.NET プロジェクトを設定する際に，テンプレートは次のことを行います。

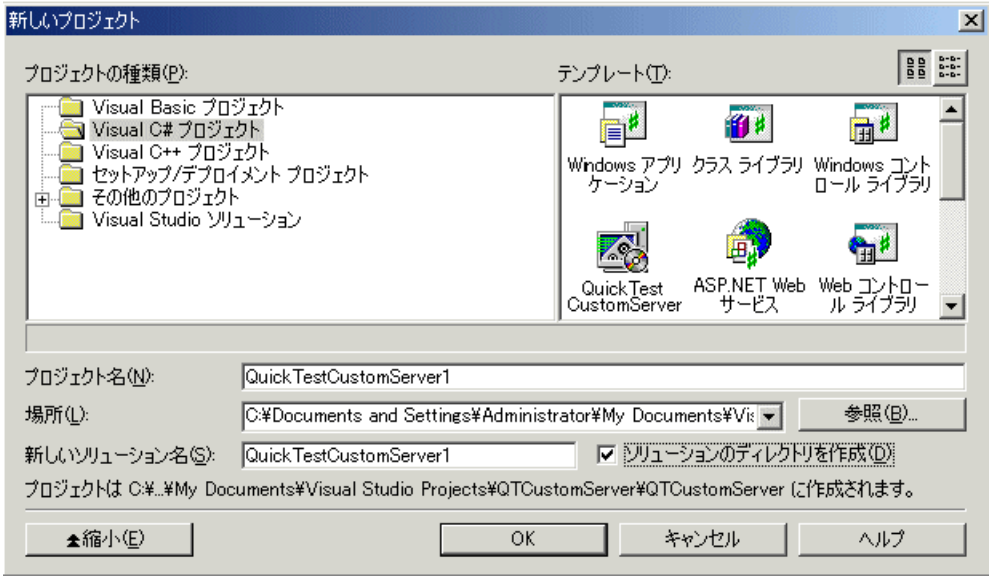
- XML ファイルを作成し，QuickTest 設定ファイルにコピーできる Custom Server の定義を含めます。
- .DLL ファイルのビルドに必要なプロジェクトを作成します。
- Test Record または Test Run を実装するときにオーバライドできるメソッドの定義を含んだコメント付きのコードを持つ C# ファイルを用意します。
- Test Record および Test Run の実装テクニックのいくつかを示すサンプル・コードを提供します。

新しい .NET プロジェクトを作成するには，次の手順を実行します。

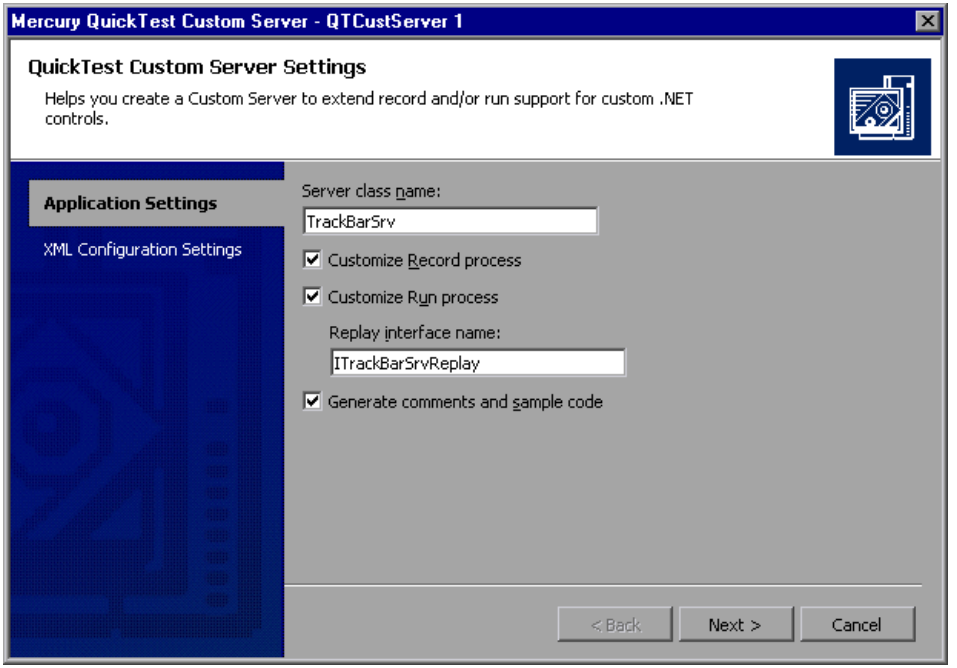
1 Microsoft Visual Studio .NET を開きます。

2 ［**ファイル**］＞［**新規作成**］＞［**プロジェクト**］を選択して，［新規プロジェクト］ダイアログ・ボックスを開くか，CTRL + SHIFT + N を押します。［**新しいプロジェクト**］ダイアログ・ボックスが開きます。

3 ［**プロジェクトの種類**］リストで［**Visual C# プロジェクト**］を選択します。



4 ［テンプレート］表示枠から「**QuickTest CustomServer**」テンプレートを選択します。新規プロジェクトの名前と，プロジェクトの保存場所を入力します。［**OK**］をクリックします。QuickTest Custom Server Settings ウィザードが開きます。



5 ウィザードの［Application Settings］ページでオプションを選択します。

- ［**Server class name**］ボックスには，Custom Server クラスに与える分かりやすい名前を指定します。
- QuickTest で Test Record プロセスを実装する場合は，［**Customize Record process**］にチェックを入れます。

  ［**Customize Record process**］にチェックを入れると，ウィザードによってステップの記録を実装するためのコードのひな形が作成されます。

  QuickTest でスクリプトを手作業で作成する場合，あるいはコントロールのマップ先である親テスト・オブジェクトの Test Record 機能を使用する場合は，このチェック・ボックスを選択しません。Test Record を実装すると，親オブジェクトの実装がその実装に置き換えられます。要求される Test Record 機能のすべてを実装する必要があります。

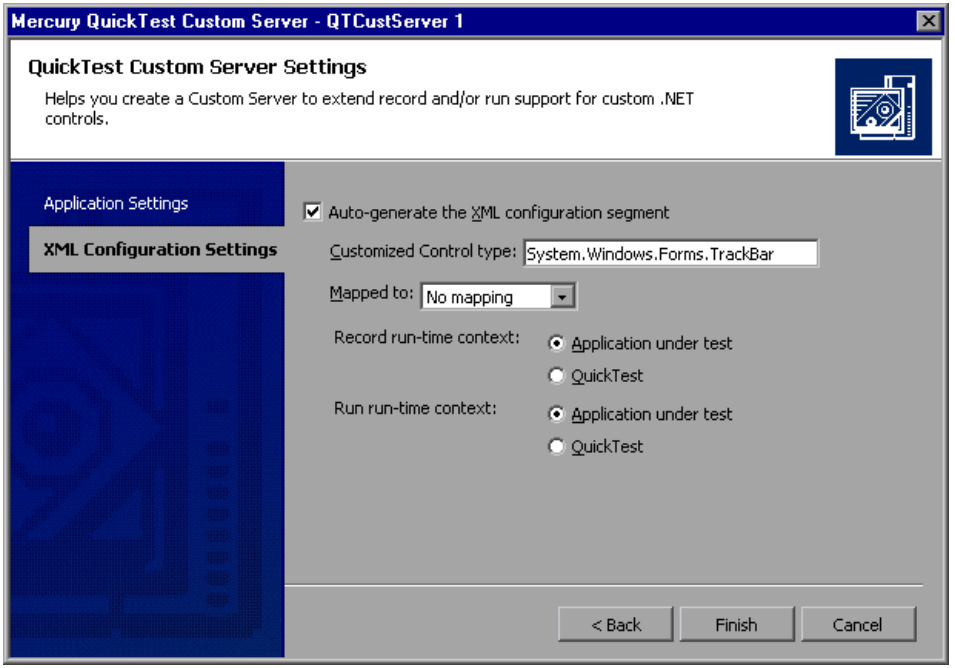
- カスタム・コントロールに対する Test Run 関数を実装する場合は，［**Customize Run process**］にチェックを入れます。［**Replay interface name**］ボックスに Replay Interface 用の名前を入力します。

  ［**Customize Run process**］にチェックを入れると，ウィザードによって Test Run サポートを実装するためのコードのひな形が作成されます。

  既存の任意のテスト・オブジェクトのメソッドをオーバライドする場合，あるいは新しいメソッドでテスト・オブジェクトを拡張する場合は，［**Customize Run process**］にチェックを入れます。

- ウィザードを使用して生成されるコードにコメントやサンプルを追加する場合は，［**Generate comments and sample code**］にチェックを入れます。

6 ［**次へ**］をクリックします。ウィザードの［XML Configuration Settings］ページが表示されます。



7 ウィザードの［XML Configuration Settings］ページでオプションを選択します。

- ウィザードに，QuickTest の設定情報を示す XML セグメントを含んだ **Configuration.xml** ファイルを作成させる場合は，［**Auto-generate the XML configuration segment** ］にチェックを入れます。

- ［**Customized Control type**］ボックスに，Custom Server を作成しているコントロールの型の完全な名前（上位のすべての名前空間も含む）を入力します（例：System.Windows.Forms.CustomCheckBox）。
- ［**Mapped to**］ボックスで，Custom Server をマップする対象となるテスト・オブジェクトを選択します。［**No mapping**］を選択すると，Custom Server は自動的に **SwfObject** テスト・オブジェクトにマップされます。

  詳細については，7 ページ「テスト・オブジェクト・マッピングの概要」を参照してください。
- Test Record および Test Run のランタイム・コンテキストとして **Application under test** または **QuickTest** のどちらかを選択します。

  詳細については，5 ページ「Custom Server ランタイム・コンテキストの概要」を参照してください。

8 ［**Finish**］をクリックします。ウィザードが閉じ，新しいプロジェクトが開きます。すぐにコーディングを開始できます。

ウィザードで［**Finish**］をクリックすると，**Configuration.xml** ファイルが作成され，プロジェクトに追加されます。Custom Server を使用できる準備が整ったら，設定情報を必要に応じて更新または変更して，QuickTest 構成ファイルに反映します。詳細については，18 ページ「XML 設定セグメントの使用」を参照してください。

XML 設定セグメントの使用

ウィザードによって作成された XML セグメントは，Custom Server の開発準備が整ってから使用します。使用の前に，プロジェクトの作成時に得られなかった情報を追加します。

QuickTest の設定にセグメントを使用するには，次の手順を実行します。

1 プロジェクトの **Configuration.xml** ファイルを編集して，情報が正しいことを確認します。**DllName** 要素の値を Custom Server をインストールする場所に設定します。Test Record および Test Run を異なるランタイム・コンテキストにロードする場合は，**Context** 値を適宜編集します。

2 **<Control> から </Control>** ノードまでをすべてコピーします。**<Controls>** タグは含めません。

3 QuickTest Professional .NET アドイン設定ファイルである，**<QuickTest Professional>¥dat¥SwfConfig.xml** を開きます。**Configuration.xml** の **Control** ノードを，ファイルの最後，**</Controls>** 終了タグの前に貼り付けます。

4 ファイルを保存します。

詳細については，35 ページ「Custom Server を使用するための QuickTest の設定」を参照してください。

カスタム・コントロールに対する .NET DLL を使用した Test Record の実装

コントロールを対象とするビジネス・コンポーネントやテスト・スクリプトを記録するということは，コントロールのアクティビティのリッスン，テスト・オブジェクト・メソッド呼び出しへのアクティビティの変換，スクリプトへのメソッド呼び出しの書き込みを意味します。コントロールに対するアクティビティのリッスンは，コントロール・イベントのリッスン，Windows メッセージの取得，またはその両方によって行われます。

Test Record を実装するには，ウィザードによって生成される IRecord インタフェースのメソッドを実装します。アプリケーションで必要なすべての機能を追加します。Test Record の実装は，カスタム・コントロールがマップされている親テスト・オブジェクトから継承しません。親オブジェクトの Test Record 実装を完全に置き換えるものです。したがって，親オブジェクトの機能が必要な場合は，それを明示的にコーディングします。

この項を読む前に，5 ページ「Custom Server ランタイム・コンテキストの概要」について理解していることを確認してください。

この項のインタフェース，クラス，列挙，メソッドの詳細については，**『QuickTest Professional .NET アドイン拡張 API リファレンス』**を参照してください。

この項では，次の内容について説明します。

- IRecord インタフェースの実装
- テスト・オブジェクト・メソッドのスクリプトの記述

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，この項で説明するコールバック・メソッドをオーバライドし，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を追加します。

InitEventListener コールバック・メソッド

CustomServerBase.InitEventListener は，Custom Server がロードされると，QuickTest によって呼び出されます。このメソッドにイベント・ハンドラとメッセージ・ハンドラを追加します。

1 コントロールのイベントにハンドラを実装します。

一般的なハンドラは，イベントを受け取り，テスト・スクリプトにメソッドを書き込みます。これは，単純なイベント・ハンドラの例です。

```
public void OnMouseDown(object sender, MouseEventArgs  e)
{
    // イベントを取得します。
    if(e.Button != System.Windows.Forms.MouseButtons.Left)
    return;
    /*
    より複雑なイベントの場合には,
    コントロールから必要なその他の情報をここで取得します。
    */
    // テスト・オブジェクト・メソッドをスクリプトに書き込みます。
    RecordFunction("MouseDown",
    RecordingMode.RECORD_SEND_LINE,
    e.X,e.Y);
}
```

詳細については，23 ページ「テスト・オブジェクト・メソッドのスクリプトの記述」を参照してください。

2 InitEventListener にイベント・ハンドラを追加します。

```
public override void InitEventListener()
{
    .....
    // OnMouseDown ハンドラを追加します。
    Delegate  e = new MouseEventHandler(this.OnMouseDown);
    AddHandler("MouseDown", e);
```

```
    .....
}
```

Test Record を **Application under test** のコンテキストで実行する場合は，次の構文を使用できます。

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

この構文を使用した場合は，ReleaseEventListener でハンドラを解放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。

Custom Server を **QuickTest** コンテキストで実行する場合は，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 型のリモート・リスナを実装し，InitEventListener メソッドに AddRemoteEventListener へ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・エージェント・リスナーを実装する場合は，EventListenerBase.InitEventListener と EventListenerBase.ReleaseEventListener をオーバーライドする必要があります。これらのコールバック関数のオーバーライドは，CustomServerBase でのオーバーライドに加えて行う必要があります。EventListenerBase におけるこの 2 つのコールバックの使用法は，CustomServerBase における使用法と同じです。詳細については，「**QuickTest Professional .NET アドイン 拡張 API リファレンス**」で EventsListenerBase クラスを参照してください。

イベントを **QuickTest** コンテキストから処理する場合には，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，「**QuickTest Professional .NET アドイン 拡張 API リファレンス**」で CustomServerBase.AddHandler（**String, Delegate, Type**）および IEventArgsHelper Interface を参照してください。

リモート・イベント・リスナーの複雑さを回避するには，前述のように，**Application under test** コンテキストでイベント・ハンドラを実行します。

OnMessage コールバック・メソッド

OnMessage は，QuickTest が受け取る任意のウィンドウ・メッセージで呼び出されます。Test Record を **QuickTest** コンテキストで実行し，メッセージ処理が必要な場合は，このメソッドでメッセージ処理を実装します。

Test Record を **Application under test** コンテキストで実行する場合は，このメソッドをオーバライドしないでください。

詳細については，「**QuickTest Professional .NET アドイン拡張 API リファレンス**」の「CustomServerBase.OnMessage」を参照してください。

InitEventListener コールバック・メソッド

Test Record を **QuickTest** コンテキストで実行し，Windows メッセージをリッスンする場合は，このメソッドをオーバライドして Custom Server が特定のカスタム・オブジェクト・ウィンドウを対象とするメッセージのみを処理するか，または Custom Server が子ウィンドウからのメッセージも同様に処理するかどうかを QuickTest に通知します。

詳細については，「**QuickTest Professional .NET アドイン拡張 API リファレンス**」の「IRecord.GetWndMessageFilter」を参照してください。

ReleaseEventListener コールバック・メソッド

QuickTest Professional 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 では，Custom Server がリッスンしていたすべてのイベントからサブスクライブ解除します。例えば，次の構文を使用して，InitEventListener で OnClick をサブスクライブしたとします。

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

この場合は，次のように解放します。

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

```
}
```

ただし，AddHandler メソッドを使用してイベントにサブスクライブした場合は，QuickTest によってサブスクライブ解除が自動的に行われます。

テスト・オブジェクト・メソッドのスクリプトの記述

コントロールのアクティビティに関する情報を，イベント，Windows メッセージ，あるいはその組み合わせとして受け取った場合には，アプリケーションに対して適切な方法で処理する必要があります。また，スクリプト・ステップをテスト・オブジェクト・メソッド呼び出しとして記述する必要があります。

スクリプト・ステップを記述するには，CustomServerBase クラスまたは EventsListenerBase クラスのいずれか該当する方の RecordFunction を使用します。

次のアクティビティが発生するまで，現在のアクティビティをどのように処理するか判別が困難な場合もあるため，スクリプト・ステップをいったん格納し，スクリプトに書き出すかどうかを **RecordFunction** への次の呼び出しで決定するメカニズムがあります。詳細については，「**QuickTest Professional .NET アドイン 拡張 API リファレンス**」の「RecordingMode Enumeration」を参照してください。

テスト・オブジェクト・メソッド呼び出しのパラメータ値を知るために，イベント引数または Windows メッセージから得られない情報をコントロールから取得する必要が生じることがあります。Custom Server Test Record オブジェクトが **Application under test** コンテキストで実行している場合は，CustomServerBase クラスの SourceControl プロパティを使用して，コントロールの public メンバに直接アクセスします。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状態情報を取得します。

カスタム・コントロールに対する .NET DLL を使用した Test Run の実装

Test Run 用にテスト・オブジェクト・メソッドを定義するということは，ビジネス・コンポーネントまたはテスト・スクリプトにおいて当該メソッドに遭遇したときに実行するアクションを指定することです。一般に，テスト・オブジェクト・メソッドの実装は，次のいくつかのアクションを実行します。

- コントロール・オブジェクトの属性値を設定します。
- コントロール・オブジェクトのメソッドを呼び出します。
- マウスとキーボードのシミュレーション呼び出しを実行します。
- ステップの結果を QuickTest にレポートします。
- エラーを QuickTest にレポートします。
- 他のライブラリへの呼び出しを行います（メッセージ・ボックスの表示，カスタム・ログの記録などのため）。

カスタム・コントロールは，親 QuickTest テスト・オブジェクトにマップされます。明示的なマッピングがされていなければ，**SwfObject** にマップされます。カスタム・コントロールをサポートするテスト・オブジェクト型は，親オブジェクトのメンバ，またはそれらのメンバをオーバライドしたもの，および Custom Server によって追加された新しいメンバで構成される新しい型です。

親テスト・オブジェクトの既存のメソッドをオーバライドする場合，あるいは新しいメソッドを追加することによって親テスト・オブジェクトを拡張する場合は，カスタムの Test Run メソッドを定義します。

記録されるすべてのテスト・オブジェクト・メソッドが，親テスト・オブジェクトまたはこの Custom Server によって Test Run に実装されることを確認してください。

カスタムの 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 属性を割り当てることによって そのインタフェースを Test Run インタフェースとします。Custom Server に実装できる再生インタフェースは 1 つのみです。インタフェースに，親オブジェクトの既存のメソッドと同じ名前を持つメソッドを定義すると，インタフェースのメソッドがテスト・オブジェクトの実装をオーバライドします。親オブジェクトのメソッドと同じ名前でないメソッドは，新しいメソッドとして追加されます。

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

詳細については，『**QuickTest Professional .NET アドイン拡張 API リファレンス**』を参照してください。

QuickTest コンテキストからの Application Under Test の下のコードの実行

Custom Server を **QuickTest** コンテキストで実行する場合，コントロールには直接アクセスできません。コントロールが，異なるランタイム・プロセスに含まれるためです。コントロールに直接アクセスするには，**Application under test** のコンテキストでコードの一部を実行します。

Application under test コンテキストで実行するコードを **QuickTest** コンテキストから呼び出すには，CustomAssistantBase を継承するアシスタント・クラスを実装します。アシスタント・クラスのインスタンスを作成するには，CreateRemoteObject を呼び出します。オブジェクトを使用する前に，SetTargetControl を使用してオブジェクトをコントロールにアタッチします。

SetTargetControl を呼び出した後は，2 つの方法でアシスタントのメソッドを呼び出せるようになります。**Application under test** プロセスの任意のスレッドでメソッドを実行できる場合は，次の簡単な obj.Member 構文を使用してコントロールの値の読み取りと設定およびコントロールのメソッド呼び出しを行います。

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

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

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

EventListenerBase はコントロール・イベントのリッスンをサポートするアシスタント・クラスです。

API の概要

この項は，最も一般的に使用される API 呼び出しのクイック・リファレンスです。詳細については，『**QuickTest Professional .NET アドイン拡張 API リファレンス**』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・スクリプトの 1 行を記録します。

Test Record コールバック・メソッド

GetWndMessageFilter	Windows Message フィルタを設定するために QuickTest 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，イベントのリッスンを開始するために QuickTest によって呼び出されます。
OnMessage	ウィンドウ・メッセージが QuickTest によって受け取られると呼び出されます。
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	マウスとキーボードをシミュレートするメソッド。
PrepareForReplay	再生アクション用に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。
ReplayThrowError	エラー・メッセージを生成し，報告されたステップの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 インタフェースによって公開されるメソッドの 1 つを呼び出します。

プロセス間メソッド

AddRemoteEventListener	**Application under test** プロセスに EventListener インスタンスを作成します。
CreateRemoteObject	**Application under test** プロセスに Assistant オブジェクトのインスタンスを作成します。
GetEventArgs （IEventArgs）	EventArgs オブジェクトを取得して，非シリアル化します。
Init（IEventArgsHelper）	イベント引数ヘルパー・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の CustomAssistantBase クラスのメソッドを呼び出します。
InvokeCustomServer	**QuickTest** プロセスを **Application under test** プロセスから実行して Custom Server のメソッドを呼び出します。
SetTargetControl	コントロールのウィンドウ・ハンドルを使用してソース・コントロール・オブジェクトにアタッチします。

一般メソッド

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	設定ファイル内の対象コントロールの設定からパラメータの値を取得します。

GetSettingsXML	設定ファイルに入力された，対象コントロールの設定を返します。

第 4 章

XML ファイルの使用による カスタム・コントロールのサポートの拡張

XML ファイルを使用して，カスタマイズされた .NET コントロールのサポートを拡張できます。XML ファイルを使用することで，プログラミング用の開発環境がなくてもサポートを拡張できます。

本章では，次の項目について説明します。

- XML ファイルの使用によるカスタム・コントロールのサポートの拡張について
- コントロール定義 XML ファイルについて
- コントロール定義 XML ファイルの例

XML ファイルの使用によるカスタム・コントロールのサポートの拡張について

コントロール定義 XML ファイルに適切な Test Record および Test Run 命令を入力することによって .NET DLL をプログラミングすることなく，カスタム・コントロール・サポートを実装できます。QuickTest 設定ファイルである **SwfConfig.xml** でこのコントロール定義ファイルに指定することで，命令をロードするよう設定できます。

この手法を使用する場合は，.NET の開発環境（オブジェクト・ブラウザとデバッガ）がサポートされません。しかし，.NET 開発環境なしでカスタム・コントロールのサポートの実装を可能にすることで，この手法により出先の現場などでも比較的素早く実装が行えます。

十分な情報がある比較的シンプルなコントロール，あるいは既存のオブジェクトに適格にマッピング可能ながらも Test Record の実装を置き換える必要があるか，テスト・オブジェクトの少数の Test Run メソッドを置き換えるか追加する必要があります。

コントロール定義 XML ファイルについて

コントロール定義 XML ファイルは，記録中にキャプチャし，ビジネス・コンポーネントまたはテスト・スクリプトに書き込むステップを生成するのに使用するコントロール・イベントを指定します。これらのステップは，カスタム・コントロールのテスト・オブジェクトのメソッドへの呼び出しです。また，このファイルは，Test Run において各メソッドが実行する操作も指定します。Record 要素と Run 要素は必ずしも両方を入力する必要はありません。

カスタム・オブジェクトが必要なすべての Test Record メソッドまたは必要なすべての Test Run メソッドのいずれかを実装する親テスト・オブジェクトにマップされている場合は，定義ファイルにその要素を定義するセクションを作成する必要はありません。

Record 要素を作成した場合は，その定義で親オブジェクトの Test Record 実装が完全に置き換えられます。Run 要素を作成した場合は，親オブジェクトの Test Run 実装が継承され，これを拡張されます。テスト・オブジェクトのマッピング・オプションの詳細については，7 ページ「テスト・オブジェクト・マッピングの概要」を参照してください。

コントロール定義 XML ファイルの構造は次のとおりです。

```
<?xml version="1.0" encoding="UTF-8"?>
<Customization>
    <Record>
        <Events>
            <!-- 1 から n までの Event 要素があります -->
            <Event name="controlEventName" enabled="true|false">
                <RecordedCommand name="theCommandName">
                    <! -- 0 から n までの Parameter 要素があります -->
                    <Parameter> param</Parameter>
                </RecordedCommand>
            </Event>
        </Events>
```

```
    </Record>
    <Replay>
        <Methods>
            <!-- 1 から n までの Method 要素があります -->
            <Method name="theCommandName">
                <Parameters>
                    <! -- 0 から n までの Parameter 要素があります -->
                    <Parameter type="theDataType" name="param 1
                        name"></Parameter>
                </Parameters>
                <MethodBody>theCommand</MethodBody>
            </Method>
        </Methods>
    </Replay>
</Customization>
```

コントロール定義ファイルの要素

- **Customization**：ルート要素。
- **Record**：イベントからテスト・スクリプトのステップへの変換に関する情報。
- **Events**：テスト・スクリプト・ステップの生成用にキャプチャするコントロール・イベントの集合。
- **Event**：特定のイベントをテスト内のステップに変換するのに必要な情報が含まれます。次の属性を持ちます。
  - **name**：コントロール・イベントの名前。
  - **enabled**：このイベントの記録を有効にするためのフラグ。値は true または false となります。
- **RecordedCommand**：親 Event 要素で定義されたイベントを受け取ったときにスクリプトに書き込むステップの定義。次の属性を持ちます。
  - **name**：スクリプトに書き込むテスト・オブジェクト・メソッド名。
- **Parameter**：Parameter 要素は，RecordedCommand の name の後にスクリプトに書き込むパラメータを定義します。パラメータは，コントロール定義 XML ファイルに定義された順番でスクリプトに書き込まれます。

  Parameter 要素には，2 つの可能な形式があります。1 つは，評価の後にスクリプトに書き込まれる 1 行のテキスト内容です。もう 1 つは，書き込むべき値を

生成するために実行する短いコードです。この場合，lang 属性を指定しなければならず，最後の値は戻り値の変数，Parameter に割り当てる必要があります。

Parameter 要素では，いくつかの予約語を使用できます。

- **Sender**：イベントを発行したオブジェクト。
- **EventArgs**：イベント・ハンドラの EventArgs パラメータを表すオブジェクト。
- **Parameter**：コードの戻り値。

Parameter 要素には，次の省略可能な属性が含まれます。

- **lang**：この要素にコードが含まれている場合，lang 属性は，プログラミング言語を指定します。現在は，C# がサポートされています。

- **Replay**：テスト・オブジェクト・メソッドから Test Run セッション中に実行されるアクティビティへの変換に関する情報。
- **Methods**：Method 要素の集合。
- **Method**：テスト・オブジェクト・インタフェースに追加されるメソッドを定義します。次の属性を持ちます。
  - **name**：テスト・オブジェクト・メソッド名。
- **Parameters**：Parameter 要素の集合。
- **Parameter**：Parameter 要素には，スクリプトからコマンド・ライン・パラメータを読み取るための命令が含まれます。Parameter 要素の順番は，スクリプト内のコマンド・ライン・パラメータの順番と同じである必要があります。
- これらのパラメータは，メソッド呼び出しを作成するために MethodBody 要素の中で使用されます。各パラメータ要素には，次の属性があります。
  - **type**：MethodBody で使用される値のデータ型。
  - **name**：MethodBody の値を参照するための名前。
- **MethodBody**：テスト・オブジェクト・メソッドが実行されたときに実行する一連の C# 命令。

  予約語 RtObject は，実行時オブジェクトを参照します。

コントロール定義 XML ファイルの例

次の例は，MouseUp イベントで値が変わるオブジェクトの処理方法を示しています。値は，オブジェクトの Value プロパティに含まれます。MouseUp イベント・ハンドラには，Button，Clicks，Delta，X，および Y イベント引数があります。

Record 要素は，MouseUp イベントの SetValue コマンドへの変換を記述します。Replay モードは，オブジェクトの値を，記録された Value に設定し，デバッグの目的でマウス・ポインタの位置を示すよう，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>
```

第 5 章

Custom Server を使用するための QuickTest の設定

QuickTest システム・ウィンドウ・フォーム設定ファイルに，Custom Server を必要な設定でロードするのに必要なすべての情報を QuickTest に提供します。

本章では，次の項目について説明します。

- Custom Server を使用するための QuickTest の設定について
- QuickTest システム・ウィンドウ・フォーム設定ファイルについて

Custom Server を使用するための QuickTest の設定について

QuickTest に，Custom Server をロードし，必要な設定を渡すよう指示するには，QuickTest システム Windows フォーム設定ファイルに情報を入力します。設定ファイル，**SwfConfig.xml** は，**＜ QuickTest Professional インストール・パス＞ ¥dat** フォルダにあります。

各コントロールは，SwfConfig.xml ファイルの **Control** ノードで設定されます。

QuickTest システム・ウィンドウ・フォーム設定ファイルについて

SwfConfig.xml ファイルは次のような構造になっています。

```
<?xml version="1.0" encoding="UTF-8"?>
<Controls>
    <Control Type=" " MappedTo="" >
        <CustomRecord>
            <Component>
                <Context> </Context>
                <DllName></DllName>
                <TypeName></TypeName>
            </Component>
        </CustomRecord>
        <CustomReplay>
            <Component>
                <Context></Context>
                <DllName></DllName>
                <TypeName></TypeName>
            </Component>
        </CustomReplay>
        <Settings>
            <Parameter Name=""> </Parameter>
        </Settings>
    </Control>
</Controls>
```

設定ファイルの要素

- **?xml**：XML 宣言「version="1.0" encoding="UTF-8”?」が必要です。
- **Controls**：ルート要素。
- **Control**：カスタム・コントロールをサポートするのに必要な情報。

  属性：

  - **Type**：上位の名前空間を含むカスタム・コントロールの完全な型（例：System.Windows.Forms.CustomCheckBox）。
  - **MappedTo**（省略可能）：Custom Server が継承する同様の振る舞いを持つ QuickTest テスト・オブジェクト・クラス。

- **Settings**：この要素は通常，Parameter 要素の集合です。使用法は 2 つあります。.NET DLL Custom Server の場合，この要素は省略可能です。

  1 つの使用法は，Custom Server が内部で使用する情報を渡します。この使用は任意です。Parameters は，アプリケーションに適した任意の目的に使用できます。また，異なる構造体を使用することも可能です。Parameter のコレクションには束縛されていません。ただし，Parameters 構造体の集合が API で直接サポートされているのに対して，異なる構造体を使用する場合は，これをコード内で自分で解析しなければなりません。

  2 つめの使用法は，拡張コントロール・サポートを XML で実装する場合で，Parameters 構造体の集合を使用する必要があるときに使用します。拡張コントロール・サポートの実装を含む XML ファイルのフル・パスと名前を Parameter に入れて渡します。このとき，**Name** 属性に ConfigPath を指定し，要素の値としてファイル・パス名を指定します。

- **Parameter**：実行時に Custom Server に渡す値。
  - **Name**：Parameter の名前。
- **CustomRecord**：Test Record に必要な情報。
- **CustomReplay**：Test Run に必要な情報。

  CustomRecord ノードと CustomReplay ノードには，Component ノードが含まれます。すべての Component のサブ要素が両方のプロセスに適用されるわけではありません。

- **Component**：Custom Server コンポーネント・データ。
- **Context**：Custom Server のランタイム・コンテキストとコーディング手法。3 つのオプションがあります。
  - **AUT**：ランタイム・コンテキストは，**Application under test** プロセスです。サポートは，.NET .DLL Custom Server として実装されます。
  - **QTP**：ランタイム・コンテキストは，**QuickTest** プロセスです。サポートは，.NET .DLL Custom Server として実装されます。
  - **AUT-XML**：ランタイム・コンテキストは，**Application under test** プロセスです。サポートは XML ファイルとして実装されます。
- **DllName**：ユーザのクラス型が定義されている DLL のファイル名です。.NET DLL コーディング手法にのみ適用されます。アセンブリを特定するフォーマットは 2 つあります。

- フル・パスとファイル名。
- Custom Server アセンブリが GAC（グローバル・アセンブリ・キャッシュ）にインストールされている場合は，標準的な構文を使って型の名前を渡します。次に例を示します。

  myQTCustomServer

  または

  myQTCustomServer, Version=1.0.1234.0

  または

  myQTCustomServer, Version=1.0.1234.0, Culture="en-US", PublicKeyToken=b77a5c561934e089c

- **TypeName**：Custom Server によって作成される型の名前（上位の名前空間を含む）。.NET DLL コーディング手法の場合にのみ該当します。

設定用 XML ファイルの例

次に，QuickTest が 2 つのコントロールを認識するよう設定するファイルの例を示します。

CustomMyListView.CustListView コントロールのサポートは，.NET DLL Custom Server として実装されています。**MyListView** は **SwfListView** テスト・オブジェクトにマップされ，**Application under test** コンテキストで実行します。Custom Server は GAC にはインストールされません。

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。

```
<?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>
        </Settings>
    </Control>

    <Control Type="mySmileyControls.SmileyControl2">
        <Settings>
            <Parameter Name="ConfigPath">d:¥Qtp¥bin¥ConfigSmiley.xml
            </Parameter>
        </Settings>

        <CustomRecord>
            <Component>
                <Context>AUT-XML</Context>

            </Component>
        </CustomRecord>

        <CustomReplay>
            <Component>
                <Context>AUT-XML</Context>

            </Component>
        </CustomReplay>
    </Control>
</Controls>
```

第 6 章

ステップ・バイ・ステップ・チュートリアル

このチュートリアルでは，QuickTest Professional を使用して，Microsoft TrackBar コントロール用の Custom Server の作成して，当該コントロールに対する **SetValue** 操作を記録することを可能にする方法を学習します。

本章では，次の項目について説明します。

- 新しい Custom Server プロジェクトの作成
- Test Record ロジックの実装
- Test Run ロジックの実装
- QuickTest Professional の設定
- Custom Server のテスト
- TrackBarSrv.cs ファイルの詳細

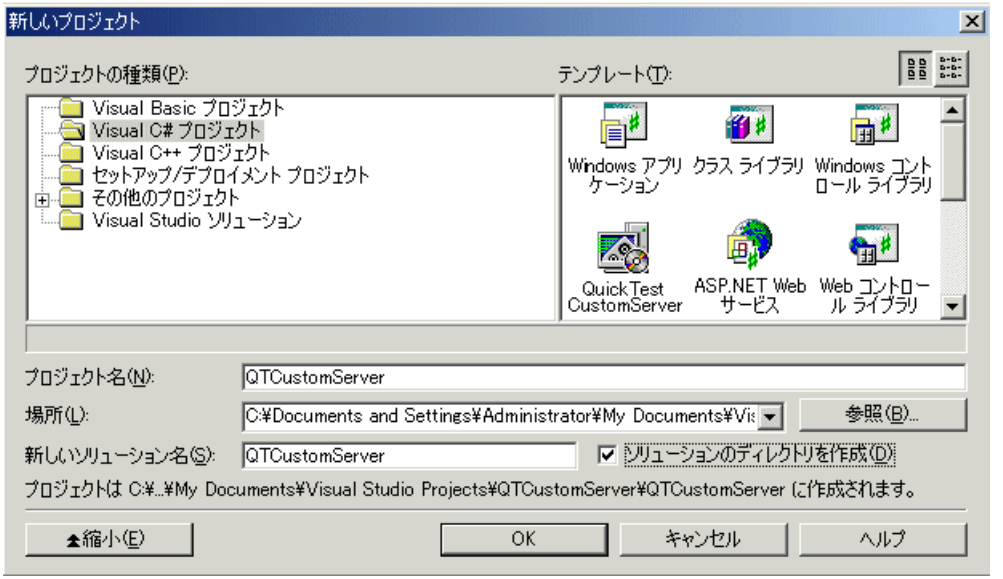
新しい Custom Server プロジェクトの作成

TrackBar コントロールに対するサポートを作成する最初のステップは，新しい Custom Server プロジェクトを作成することです。

新しい Custom Server プロジェクトを作成するには，次の手順を実行します。

1 Microsoft Visual Studio .NET を開きます。

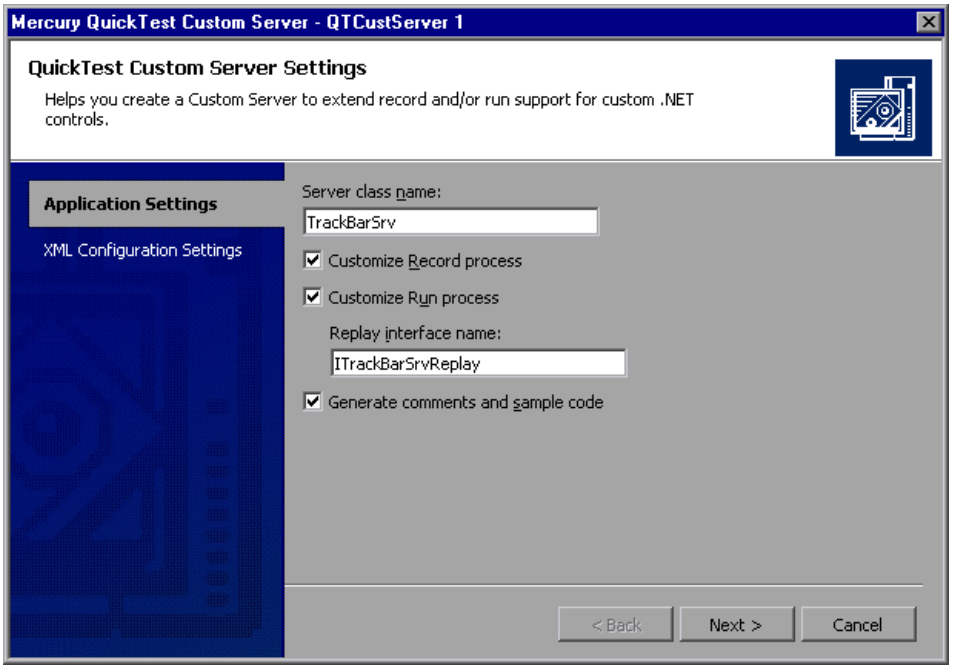
2 ［**ファイル**］＞［**新規作成**］＞［**プロジェクト**］をクリックします。［**新しいプロジェクト**］ダイアログ・ボックスが開きます。



3 次の設定を指定します。

- ［**プロジェクトの種類**］リストで［**Visual C# プロジェクト**］を選択します。
- ［**テンプレート**］表示枠で［**QuickTest CustomServer**］を選択します。
- ［**プロジェクト名**］ボックスに，プロジェクト名として，「QTCustServer」を指定します。
- ［**場所**］ボックスに，プロジェクトの保存場所を指定します。
- そのほかは標準の設定を適用します。

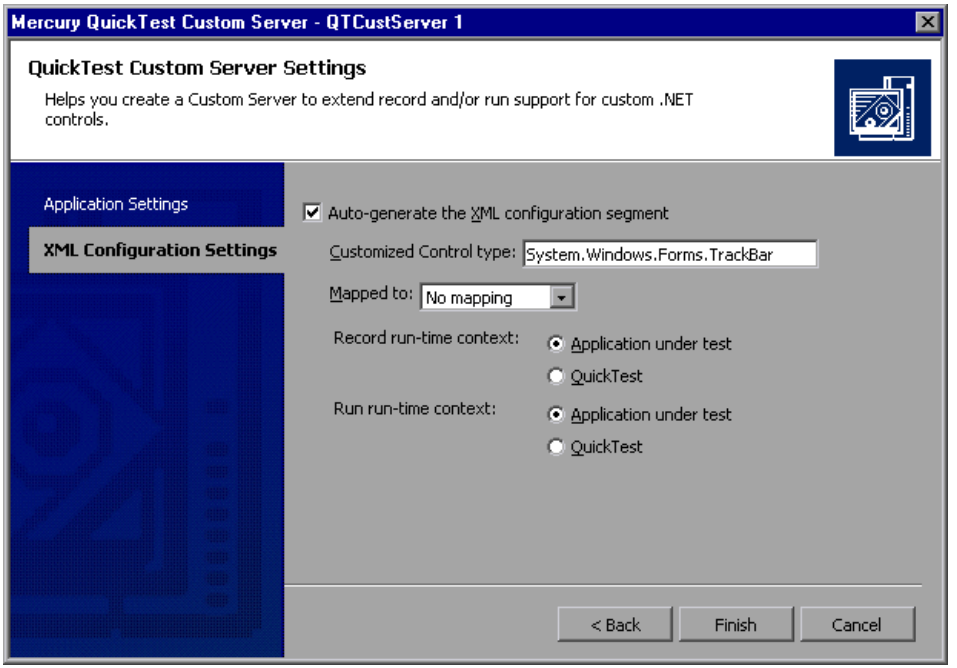
4 ［**OK**］ボタンをクリックします。［**QuickTest Custom Server Settings**］ダイアログ・ボックスが開きます。



5 ［**Application Settings**］ページで，次の設定を指定します。

- ［**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。
- そのほかは標準の設定を適用します。

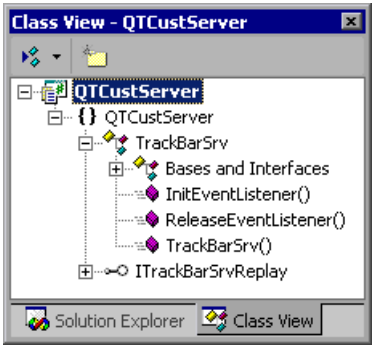
6［**次へ**］をクリックします。［**XML Configuration Settings**］ページが表示されます。



7［**XML Configuration Settings**］ページで，次の設定を指定します。

- ［**Auto-generate the XML configuration segment**］チェック・ボックスが選択されていることを確認します。
- ［**Customized Control type**］ボックスに，「System.Windows.Forms.TrackBar」と入力します。
- そのほかは標準の設定を適用します。

8 ［**Finish**］をクリックします。［**Class View**］ウィンドウを見ると，ウィザードによって，**CustomServerBase** クラスおよび **ITrackBarSrvReplay** インタフェースから派生する **TrackBarSrv** クラスが作成されたのが分かります。



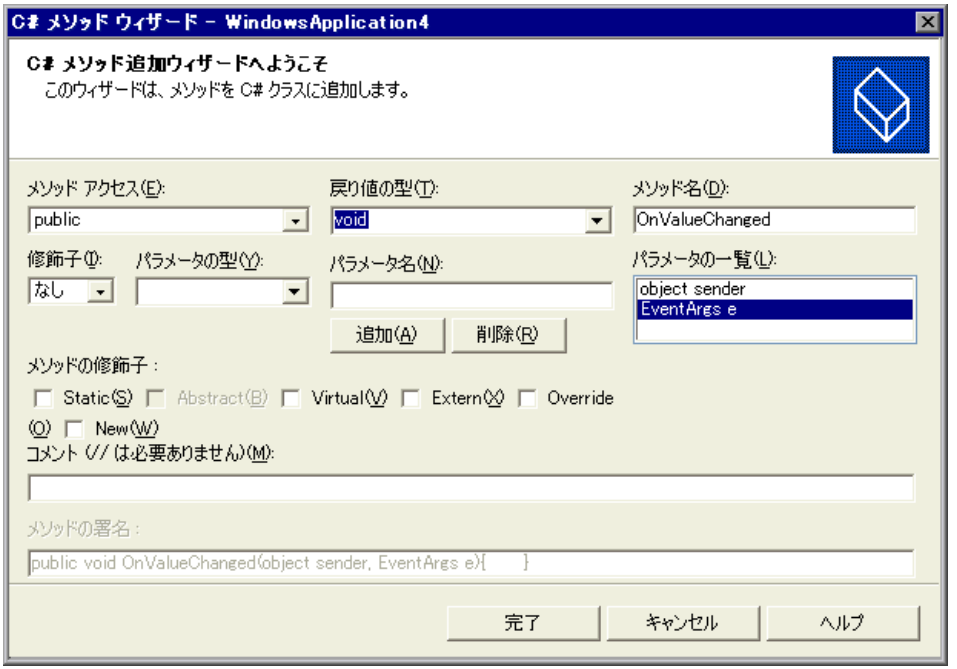
Test Record ロジックの実装

次にイベント・ハンドラ関数を使用して **ValueChanged** イベントが発生したときに，**SetValue(X)** コマンドを記録するロジックを実装します。

Test Record ロジックを実装するには，次の手順を実行します。

1 ［**クラス ビュー**］ウィンドウで［**TrackBarSrv**］クラス名を右クリックし，［**追加**］＞［**メソッドの追加**］を選択します。

［**C# メソッド追加ウィザード**］が開きます。



2 ［**C# メソッド追加ウィザード**］を使用し，次のシグネチャを持つ新しいメソッドを追加します。

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

注：あるいは，［**C# メソッド追加ウィザード**］を使用せずに，手動で新しいメソッドを追加することもできます。

---

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;
 // 新しい値を取得
 int newValue = trackBar.Value;
```

```
// テスト・スクリプトへ SetValue コマンドを記録
RecordFunction("SetValue", RecordingMode.RECORD_SEND_LINE,
newValue);
}
```

4 InitEventListener メソッドに次のコードを追加することによって，ValueChanged イベントに対する OnValueChanged イベント・ハンドラを登録します。

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

Test Run ロジックの実装

次は，テストまたはビジネス・コンポーネント Test Run に対する，**SetValue** メソッドを実装します。

Test Run ロジックを実装するには，次の手順を実行します。

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

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

2 次のメソッド実装を，**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 プロジェクトをビルドします。

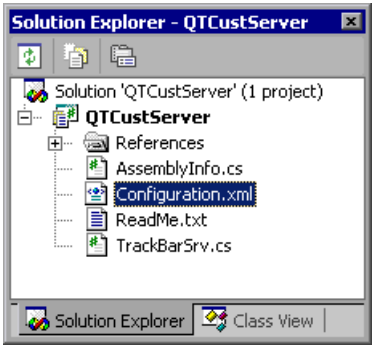
注：TrackBarSrv クラスのソース・コード全体を，50 ページ「TrackBarSrv.cs ファイルの詳細」に示します。

QuickTest Professional の設定

QuickTest Custom Server を作成したら，TrackBar コントロールを対象とするテストの記録と実行のときに，この Custom Server を使用するように QuickTest Professional を設定する必要があります。

Custom Server を使用するように QuickTest Professional を設定するには，次の手順を実行します。

1 ［**ソリューション エクスプローラ**］ウィンドウで，**Configuration.XML** ファイルを選択します。



次の内容が表示されます。

```
<!-- この XML の内容を "<QuickTest Professional>¥dat¥
    SwfConfig.xml" ファイルに統合する -->
    <Control Type="System.Windows.Forms.TrackBar">
        <CustomRecord>
            <Component>
                <Context>AUT</Context>
                <DllName>D:¥Projects¥QTCustServer¥Bin¥QTCustServer.dll
</DllName>
```

```
                <TypeName>QTCustServer.TrackBarSrv</TypeName>
            </Component>
        </CustomRecord>
        <CustomReplay>
            <Component>
                <Context>AUT</Context>
                <DllName>D:¥Projects¥QTCustServer¥Bin¥QTCustServer.dll
</DllName>
                <TypeName>QTCustServer.TrackBarSrv</TypeName>
            </Component>
        </CustomReplay>
        <!--<Settings>
            <Parameter Name="sample name">sample value</Parameter>
        </Settings> -->
    </Control>
```

2 **<Control>...</Control>** セグメントを選択し，メニューから［**編集**］＞［**コピー**］をクリックします。

3 **＜ QuickTest Professional のインストール先フォルダ＞ ¥dat** にある，**SwfConfig.xml** ファイルを開きます。

4 **Configuration.xml** でコピーした **<Control>...</Control>** セグメントを，**SwfConfig.xml** 内の **<Controls>** タグの下に貼り付けます。セグメントを貼り付けた後，**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¥QTCustServer¥Bin¥QTCustServer.dll
</DllName>
                <TypeName>QTCustServer.TrackBarSrv</TypeName>
            </Component>
        </CustomRecord>
        <CustomReplay>
            <Component>
                <Context>AUT</Context>
```

```
            <DllName>D:¥Projects¥QTCustServer¥Bin¥QTCustServer.dll
</DllName>
            <TypeName>QTCustServer.TrackBarSrv</TypeName>
        </Component>
    </CustomReplay>
</Control>
</Controls>
```

5 **<DllName>** 要素に Custom Server DLL への正しいパスが含ま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。

Custom Server のテスト

カスタムの TrackBar コントロールを対象に，QuickTest によるテストまたはコンポーネントの記録と実行が，期待どおりに行われることを確認できます。

Custom Server をテストするには，次の手順を実行します。

1 .NET アドインが読み込まれるように QuickTest Professional を起動します。

2 System.Windows.Forms.TrackBar コントロールを持つ，.NET アプリケーションを対象とする記録を開始します。

3 **TrackBar** コントロールをクリックします。QuickTest は次のようなコマンドを記録します。

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

4 テストを実行します。TrackBar コントロールは，正しい値を受け取ります。

TrackBarSrv.cs ファイルの詳細

次に，**TrackBarSrv** クラスのソース・コード全体を示します。

```
using System;
using Mercury.QTP.CustomServer;

namespace QTCustServer
{
```

```
[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;
    }
```

```
        }
    }
```

索引