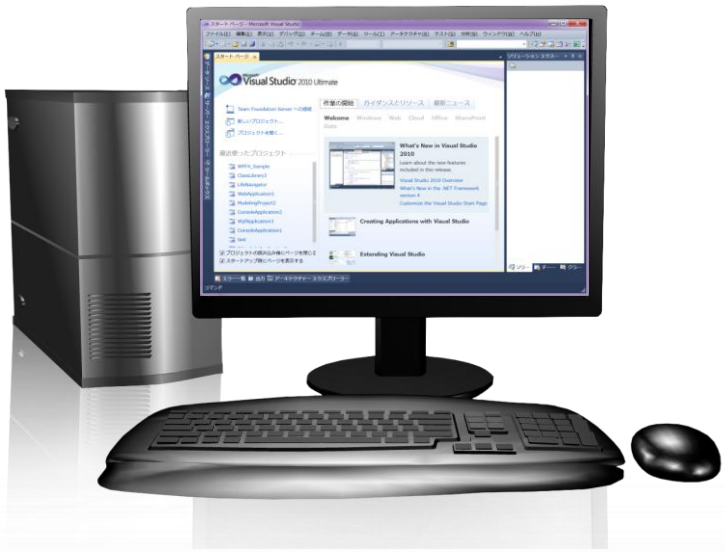
Microsoft® Visual Studio® 2010

Microsoft® Visual Studio® 2010
開発ガイドブック



著作権

このドキュメントに記載されている情報は、このドキュメントの発行時点におけるマイクロソフトの見解を反映したものです。マイクロソフトは市場の変化に対応する必要があるため、このドキュメントの内容に関する責任を問われないものとします。また、発行日以降に発表される情報の正確性を保証できません。
このホワイトペーパーは情報提供のみを目的としています。明示、黙示、または法令に基づく規定に関わらず、これらの情報についてマイクロソフトはいかなる責任も負わないものとします。
この文書およびソフトウェアを使用する場合は、適用されるすべての著作権関連の法律に従っていただくものとします。このドキュメントのいかなる部分も、米国 Microsoft Corporation の書面による許諾を受けることなく、その目的を問わず、どのような形態であっても、複製または譲渡することは禁じられています。ここでいう形態とは、複写や記録など、電子的な、または物理的なすべての手段を含みます。ただしこれは、著作権法上のお客様の権利を制限するものではありません。
マイクロソフトは、この文書に記載されている事項に関して、特許、申請中特許、商標、著作権、および他の知的財産権を所有する場合があります。別途マイクロソフトのライセンス契約上に明示の規定のない限り、このドキュメントはこれらの特許、商標、著作権、またはその他の知的財産権に関する権利をお客様に許諾するものではありません。
別途記載されていない場合、このドキュメントで使用している会社、組織、製品、ドメイン名、電子メール アドレス、ロゴ、人物、場所、出来事などの名称は架空のものです。実在する商品名、団体名、個人名などとは一切関係ありません。

マイクロソフト ビジュアル スタジオ 2010
開発ガイドブック
―目次―

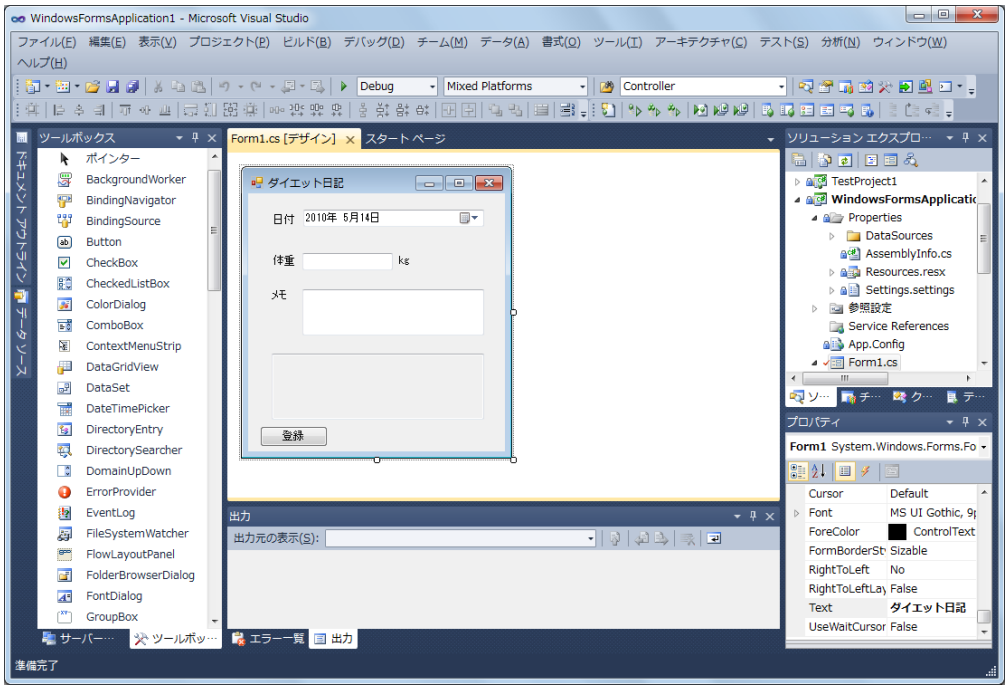
第 1 章 Visual Studio 2010 の概要

本書では、アプリケーションを効率よく開発するための、Microsoft Visual Studio 2010 の活用方法を解説します。

Visual Studio 2010 とは?

Visual Studio 2010 は、ビジネスの要件やユーザーのニーズに最適なアプリケーションを迅速かつ高品質に構築するための最新の統合開発環境 (IDE:Integrated Development Environment) です。

「Windows アプリケーション」「Web アプリケーション」「Office アプリケーション」といったユーザーに最適なアプリケーション形態を選択して開発を行うことができ、またアプリケーション開発のライフサイクルに合わせ、クラス設計、コーディング支援、デバッグ、ビルド、各種テスト機能、変更管理、あるいは作業項目管理といった開発支援機能が利用可能です。



Visual Studio 2010 には多様な開発支援機能が用意されています。

たとえば ユーザー インターフェイス (UI) の設計/開発においては RAD (Rapid Application Development) スタイルを採用しています。RAD スタイルの開発では、実際にユーザーが使用する画面のイメージを使用しながら開発を進めます。
開発者はあらかじめ用意されているボタンやテキスト ボックスなどのコントロール (部品) を、マウスを使用してドラッグ アンド ドロップにより自由に配置し、その外観を設計します。

開発者は次に、アプリケーションにおいてユーザーが実施する操作 (イベント) に対してどのような振る舞いを行うかを定義します。たとえば、あるボタン A をクリックした場合 (ボタン A で Click という「イベント」が発生した場合) に起動されるアクションを Microsoft® Visual Basic® や Microsoft® Visual C#® といった開発言語で記述します。Visual Studio 2010 ではこの RAD スタイルの開発を効率的に実施できるように各種の機能を提供しており、開発者が高い生産性で開発を進めることができるのはもちろん、ユーザーにとってはプロジェクトの初期段階から画面の外観や、イベント発生時の大まかな動作を容易に確認できるために、ユーザーのニーズを確認しながらプロジェクトを進行することが可能になります。

また、Visual Studio 2010 は現在のアプリケーション開発において実際に利用されているさまざまなテクノロジ領域をカバーしています。

OS やデータベースといったアプリケーションのプラットフォームとなるテクノロジとしては、Microsoft® Windows Server® 2008、Windows 7®、Microsoft® SQL Server® 2008、Microsoft® Office 2010 といった最新のプラットフォーム テクノロジをサポートしています。
また、実行基盤においては、最新の .NET Framework 4 に加えて、従来の .NET Framework 2.0/3.0/3.5 を使用した開発にも対応しているので、アプリケーションを使用するユーザーの環境に合わせて実行基盤を選択することが可能です。

これらの特長に加え、Visual Studio 2010 では Visual Basic、C#、Microsoft® Visual C++® 、F# といった複数のプログラミング言語を利用して開発を行うことが可能です。F# は、Visual Studio 2010 で新しく追加された関数型言語です。そのため開発者は、これまでの開発経験で培ってきたスキルを最大限に活用し、最新のプラットフォーム テクノロジを活用したアプリケーションを効率的に開発することが可能になります。

Visual Studio 2010 のラインアップ

Visual Studio 2010 では、開発者のニーズに合わせていくつかのエディションを提供していますが、クライアント側のエディションは大まかに区別すると以下の 5 つの製品ラインに整理できます。

Microsoft® Visual Studio® 2010 Ultimate	設計から開発、テスト、導入までアプリケーション ライフサイクルを広範囲にカバーし、高い品質を持つアプリケーションを、高い開発生産性を保ちながら構築するために必要となる機能を提供します。Visual Studio 2010 Premium, Professional, Test Professional 2010 のすべての機能に加えて、さらにアプリケーション ライフサイクル全般にわたり高度な機能を搭載した最上位エディション
Microsoft® Visual Studio® 2010 Premium	アプリケーションの開発工程において、高い品質と生産性を維持しながらアプリケーションを開発するために必要となる機能を提供します。Visual Studio 2010 Professional に加えて、コードの品質や高度なデータベース開発を実現するための機能等を搭載
Microsoft® Visual Studio® 2010 Professional	ビジネスの要件やユーザーのニーズに最適なアプリケーションを選択し、それを構築するために必須の機能を提供します。
Microsoft® Visual Studio® Test Professional 2010	テストの計画から実施、発生した問題の管理まで、一連のテスト プロセス全般における専門的かつ包括的な機能を提供します。
Microsoft® Visual Studio® 2010 Express	学習者/ホビイスト向けの製品群。開発目的に応じたツールを提供します。

エディションによって提供される機能は異なりますが、本書では、Visual Studio 2010 Ultimate を使用します。また、Visaul Studio 2010 Express はそれぞれ複数の製品から構成されている製品群であり、それぞれ以下の製品によって構成されています。

Visual Studio 2010 Express を構成する製品 (それぞれ個別に提供)

Microsoft® Visual Basic® 2010 Express
Microsoft® Visual C#® 2010 Express
Microsoft® Visual C++® 2010 Express
Microsoft® Visual Web Developer® 2010 Express Microsoft® Visual Studio ® 2010 Express for Windows Phone

Visual Studio 2010 の購入形態

Visual Studio 2010 は、店頭やオンライン ショップで発売されている「パッケージ」、および企業での購入に最適化された「ボリュームライセンス」という 2 つの形態で購入いただけます。
また、Visual Studio 2010 Professional を購入すると、12 か月間の MSDN Essentials Subscription が付属しています。開発に際してマイクロソフトのコア プラットフォーム (Windows 7 Ultimate、Windows Server 2008 Enterprise R2、および SQL Server 2008 Datacenter R2) を 12 か月間利用可能です。また学生および教職員向けのパッケージとして、Professional Edition を割安でご購入いただける「Visual Studio 2010 Professional アカデミック」を提供しているほか、学生の教育を支援するためのプログラムとして「MSDN アカデミック アライアンス」を利用いただけます。

Visual Studio 2010 評価版の入手

Visual Studio 2010 は無償で利用可能な各エディションの評価版を用意しています。以下の URL から Visual Studio 2010 Ultimate 評価版をダウンロードし、実際に試しながら本ドキュメントをご覧ください。

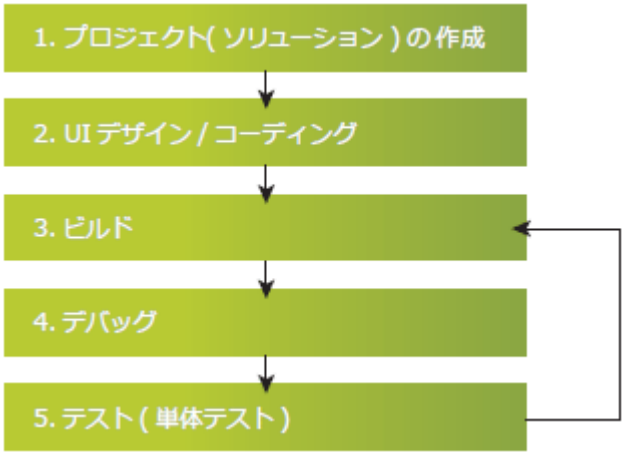
http://www.microsoft.com/japan/visualstudio/download

第 2 章 開発の流れと開発スタイル

本章では、Visual Studio 2010 を使ってアプリケーションを開発する際の流れをご紹介します。

Visual Studio 2010 での開発の流れ

「Windows アプリケーション」や「Web アプリケーション」(Web サイト)、「Office アプリケーション」といったアプリケーションの種類にかかわらず、Visual Studio 2010 では一般的に次の図に示す手順でアプリケーションの開発作業が進められます。



Visual Studio 2010 でのアプリケーション開発手順

この図の概略を順に説明しましょう。以下では Windows アプリケーションを開発する場合を想定しています。

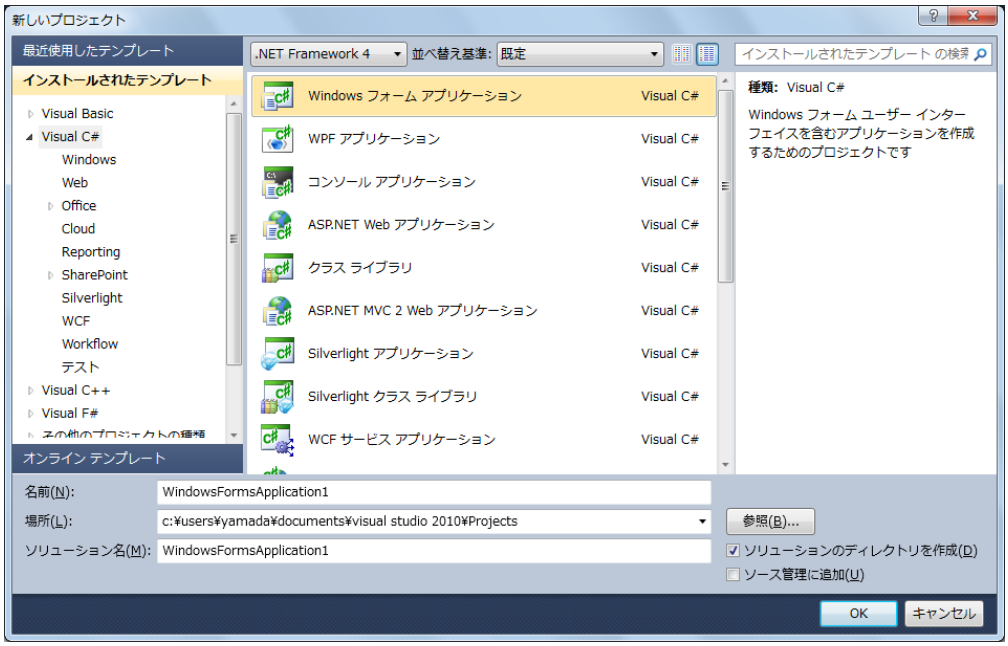
(1) プロジェクト (ソリューション) の作成

Visual Studio 2010 での開発作業は、まず “プロジェクト” を作成するところから始めます。
プロジェクトとは、アプリケーションの構築に必要なソース ファイル群、アプリケーションの実行に必要な画像リソースなどをまとめて管理するコンテナーであり、この ” プロジェクト” を 1 つの単位として、ビルドおよびデバッグを実施することが可能です。
また、Visual Studio 2010 では、この ” プロジェクト” をまとめるより上位のコンテナーとして、“ソリューション” という概念を用意しています。

※ 開発における成果物の 1 つであるライブラリ(DLL) や、実行形式 (EXE) は、“プロジェクト”ごとに作成されます。つまり、別の見方をすれば、”プロジェクト”は「配置を行う際の単位」であり「保守を行う際の単位」と見ることもできます。
“プロジェクト”の単位については明確な規定はありませんが、プロジェクトの分割を行う際はプロジェクト間の依存関係を最小化する (たとえば、プロジェクト間の依存関係はインターフェイス (サービスを含む) のみに限定する、など)、あるいは将来的な変更頻度の可能性に応じてプロジェクトを分割する (たとえば、変更頻度の高そうなビジネスロジック A に関連するソース ファイルと、変更頻度の低そうなビジネスロジック B に関連するソース ファイルは、あらかじめ別のプロジェクトとしてまとめておく、など) といったことを心がけてください。

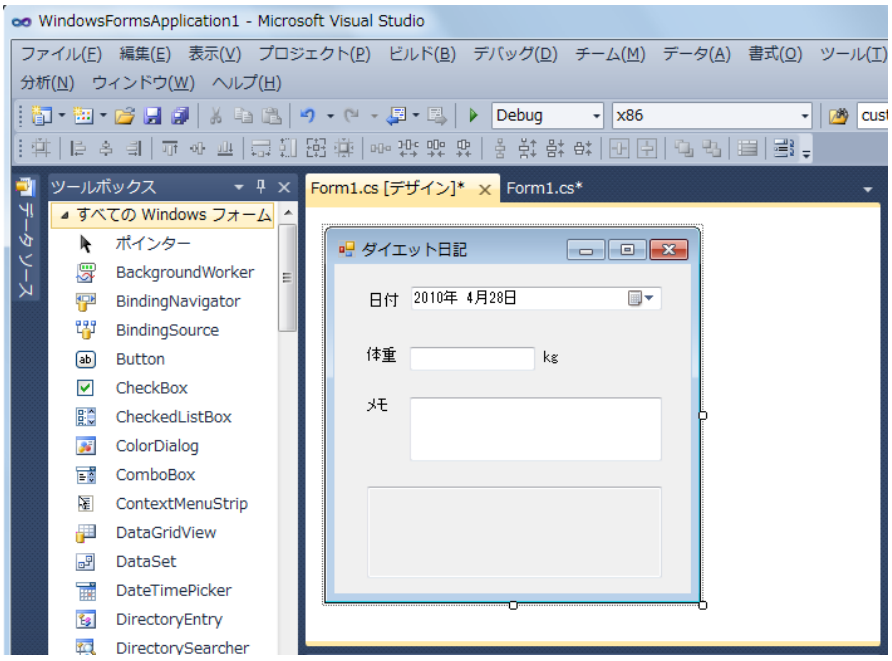
プロジェクトを作成するには「プロジェクト テンプレート」と呼ばれる、Visual Studio 2010 が用意するプロジェクトのひな形を選択するだけです。
作成したいアプリケーションに合致するプロジェクト テンプレートを選択し、プロジェクト名やプロジェクトを保存するフォルダなどを指定すると、選択されたプロジェクトに応じて、ひな形のソースファイル群が自動生成されます。

「Windows アプリケーション」や「Web アプリケーション」、あるいは「Office　アプリケーション」といったアプリケーション形態はあらかじめこのプロジェクト テンプレートとして提供されていますので、ここで適切なテンプレートを選択することで、アプリケーションとしての基本的なひな形は用意されることになります。
また、プロジェクトは既存のソリューションに追加する形で作成することもできますし、既存のソリューションに含めずに新規で作成することも可能です。後者の場合、作成したプロジェクトのみを含んだソリューションが新規で自動的に作成されます。



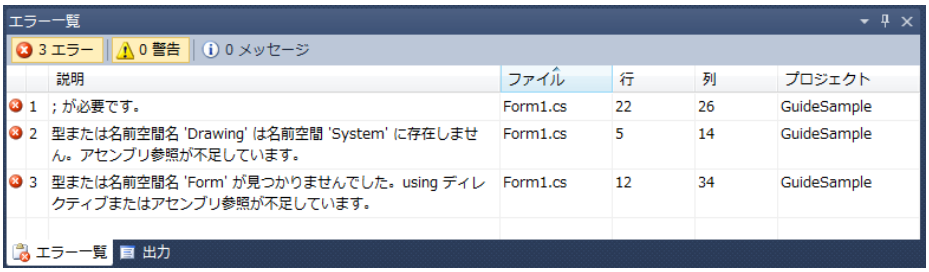
(2) UI デザイン/ コーディング

次に、生成されたプロジェクトのひな形を利用してアプリケーションの設計/ 開発を行います。
Windows アプリケーションや Web アプリケーションなどの場合には、ユーザーが実際に使用するアプリケーションのイメージが Visual Studio 2010 上に用意されますので、そのイメージを利用して各種コントロールを配置してユーザー インターフェイスを設計し、イベントごとに実施する処理を記述 (コーディング) します。



(3) ビルド

ユーザー インターフェイスの作成やコードの記述が完了すれば、ビルドを行い、コーディングミスによるコンパイル エラーを取り除きます。



コンパイル エラーが [エラー一覧] ウィンドウに表示された例

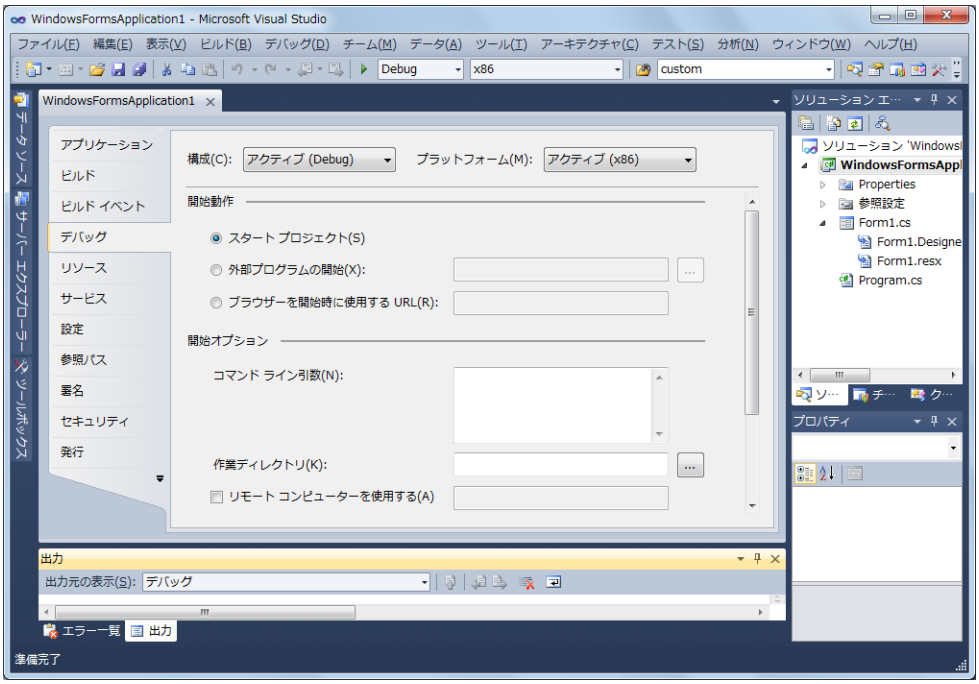
ビルドが正常に終了すると、ライブラリ(.dll ファイル) もしくは実行形式のファイル (.exe ファイル) が作成されます(.dll および .exe ファイルを「アセンブリ」とも呼びます)。

Web アプリケーションの場合は必ずしも実行形式のファイルを作成するわけではありませんが、ビルドを行うことで文法的な誤りを取り除くことができます。

(4) デバッグ

無事ビルドが成功したら、アプリケーションが正しく作動するかを検証するデバッグ作業を行います。

デバッグ作業ではアプリケーションの正常な実行を妨げている論理的な誤りを取り除きます。Visual Studio 2010 には、変数の値の参照/修正/監視、デバッグ実行を停止させるブレークポイントの設定など、さまざまなデバッグ機能が備わっており、効率的にデバッグを実施することが可能です。

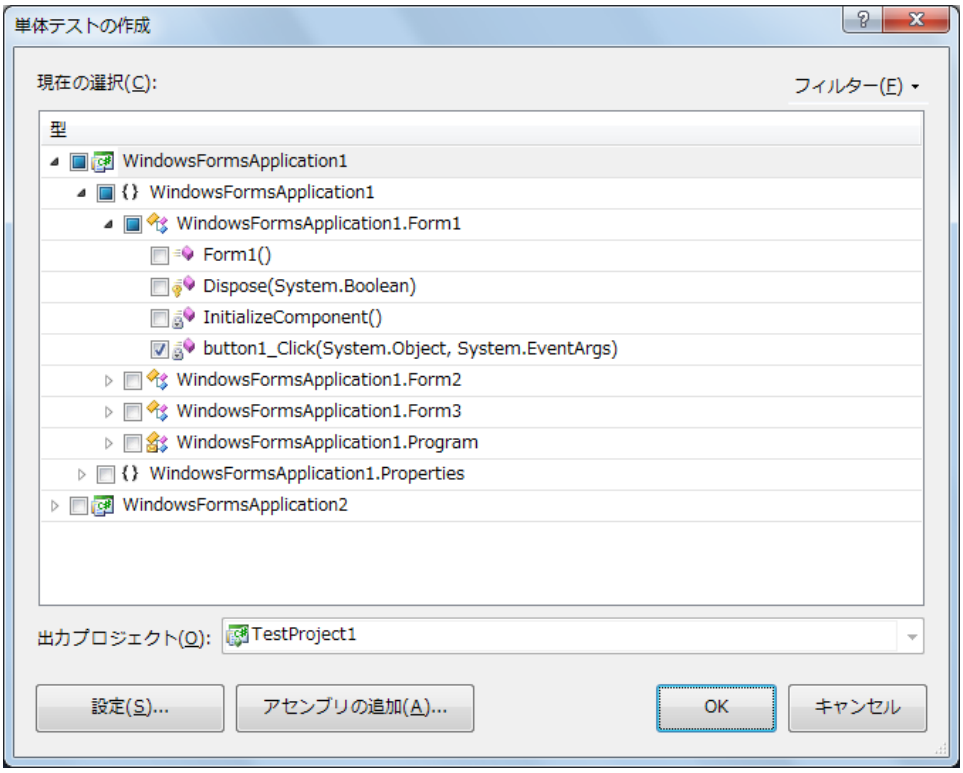


デバッグ時の例

(5) 単体テスト

アプリケーションの品質を高く保つためには、テストが欠かせません。先ほどの図を見るとわかるように、テストに失敗すればコードの修正後、ビルドとデバッグに戻る必要があります。ビルドとデバッグ、そしてテストは、実際には切り離せない作業です。

Visual Studio 2010 Professional 以上のエディション には、そのテストを行うための機能として、単体テストの作成/実行機能を提供しています。
単体テスト機能を利用することで、ソースコードの各ロジック (通常はメソッド単位) が期待どおりに動作するかどうかを確認するテストの実施を自動化することが可能になります。通常単体テストは各ロジックの開発者が記述するテストであり、この後にテスト担当者が実施する統合テストやシナリオ テストの前に実施されるテストです (そのため「開発者テスト」と呼ばれる場合もあります)。
適切な単体テストを実施することは、後のテスト工程におけるバグ発見のリスクを低減させることにつながります。また自動化された単体テストの資産は、アプリケーションに変更を行った際のレグレッションテスト (後退テスト) にも利用できるため、保守における変更リスクを低減させる、といったメリットもあります。



単体テストの作成機能

上記以外の手順について

実際のシステム構築全体の流れでは、以上のような「開発」手順以外にも、「要件定義」や「システム設計」、「展開/ 運用」などさまざまなプロセスが存在しますが、これらについては本ドキュメントでは割愛します。
Visual Studio 2010 の展開/ 運用機能に関して簡単に紹介しておくと、ClickOnce による Windows アプリケーション/Office アプリケーションの配布や、セットアップ プログラム (.msi ファイル) の作成が行えるほか、Web アプリケーションでは Web 配置パッケージの作成を行うことが可能で、かつ配置のためのツールも用意されており、統合開発環境から直接 Web サイトや Web アプリケーションの配置が行えます。

すべての開発手順を 1 つの開発環境で

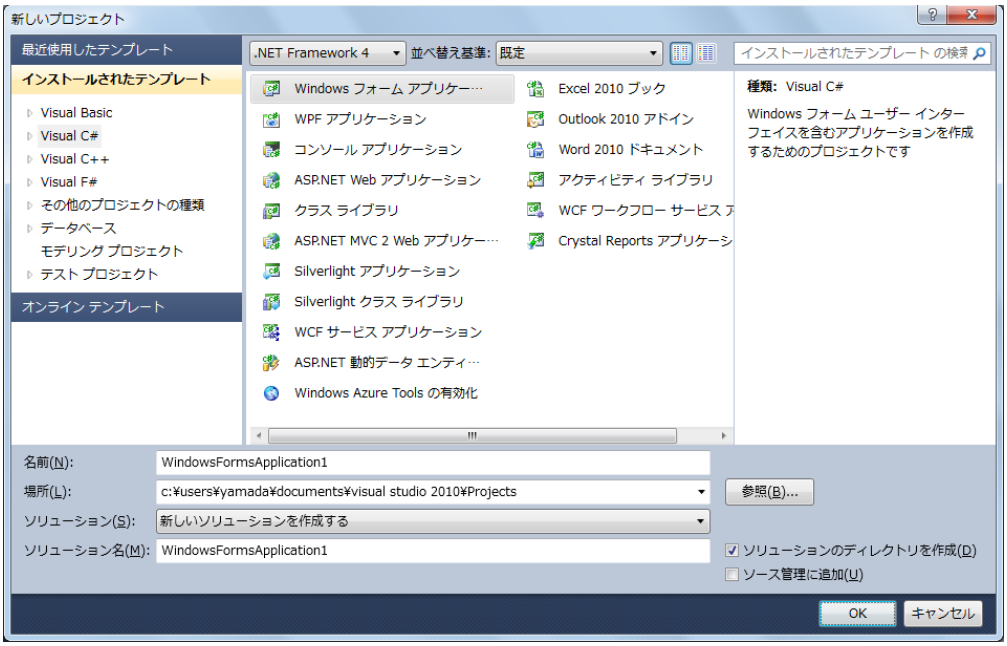
以上のように Visual Studio 2010 では、プロジェクトの作成から、ソース コードの記述、ビルド、デバッグ、テストまで、アプリケーション開発の一連のステップをすべて統合開発環境上で行うことが可能です。

また、一見すると異なる開発アプローチが必要と思われる Windows アプリケーションと Web アプリケーションの構築も、Visual Studio 2010 ならユーザー インターフェイスの開発を含めてほぼ同じ操作で一連の開発作業が行えます (詳しくは、「第 4 章アプリケーションの開発」の中の「統一された開発スタイル」で紹介します)。

以降では、本章で紹介した Visual Studio 2010 の一般的な開発の流れに沿って、Visual Studio 2010 の使い方の詳細を見ていくことにしましょう。

第 3 章 プロジェクトの作成

前述したように、Visual Studio 2010 によるアプリケーション開発では、まずプロジェクトを新規作成するところから始まります。Visual Studio 2010 には、「プロジェクト テンプレート」と呼ばれるプロジェクトのひな形が用意されており、作成したいアプリケーションに応じてテンプレートを選択するだけで、目的に合った内容のプロジェクトを簡単に作成できます。



プロジェクト テンプレートの例

[新しいプロジェクト] ダイアログ ボックスでプロジェクト テンプレートを選択できます。

[新しいプロジェクト] ダイアログ ボックスについては後述します。

プロジェクト テンプレートには、Windows アプリケーションや Web アプリケーションといった「アプリケーション」を作成するためのプロジェクトだけでなく、アプリケーションを検証するためのテスト プロジェクトや、インストーラ プロジェクト、データベース プロジェクトなど、さまざまなプロジェクトが用意されています。
さらに、Visual Studio 2010 Professional 以上には、 Microsoft Office 2007 (以降、Office 2007) および Microsoft Office 2010 (以降、Office 2010) を操作するための VSTO (Visual Studio Tools for Microsoft Office system) が含まれており、Microsoft® Office Excel® ワークブックや Microsoft® Office Word アドインといった Office プロジェクトを作成可能です。また、Visual Studio 2010 では、 Office 2010 から追加された 64 ビット版にも対応しています。

さらに、Windows Azure を利用したクラウド開発のためのプロジェクトや Silverlight を利用した RIA 開発のためのプロジェクトも用意されています。
そこで、実際の開発手順の説明 (「プロジェクトの新規作成」) に入る前に、Visual Studio 2010 の代表的なプロジェクト テンプレートについてざっと紹介します。

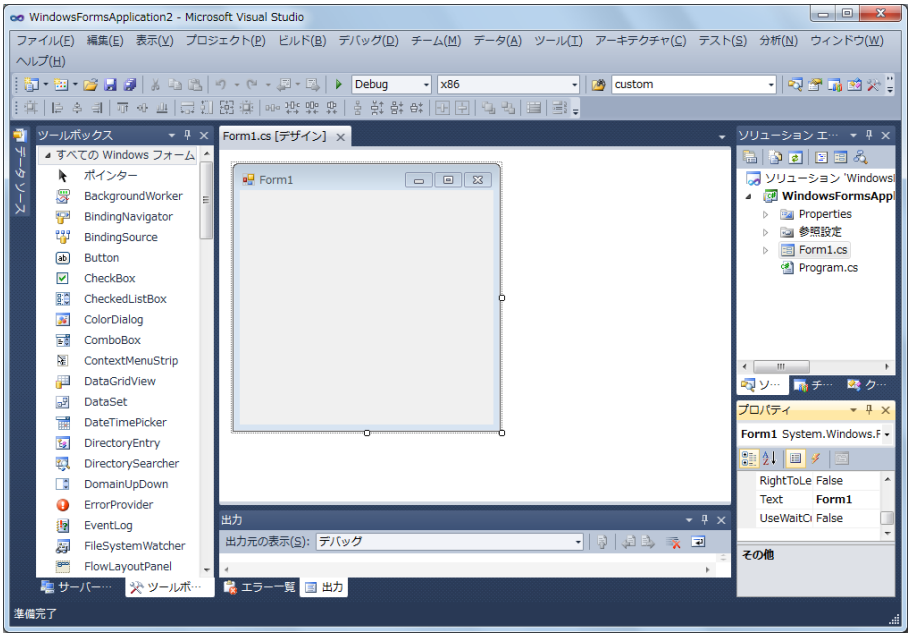
プロジェクトの種類

Visual Studio 2010 には、(プログラミング言語の違いによる重複も含めて) 100 種類以上のプロジェクト テンプレートが含まれています。これらは大まかに「Windows アプリケーション/ライブラリ」「Windows サービス」「Web アプリケーション/Web サービス」

「Office アプリケーション (Office 2007 および Office 2010)」「セットアップ」などに分類できます。以下ではそれぞれの分類に含まれるプロジェクト テンプレートの概要を説明します。

Windows ベースのプロジェクト

Visual Studio 2010 では、Windows ベースのアプリケーションを構築するためのプロジェクト テンプレートとして「Windows フォーム アプリケーション」が用意されています。Windows フォーム アプリケーションでは、「Windows フォーム」というデザイン画面を主体とした開発が行えます。



「Windows フォーム」というデザイン画面を主体とした開発

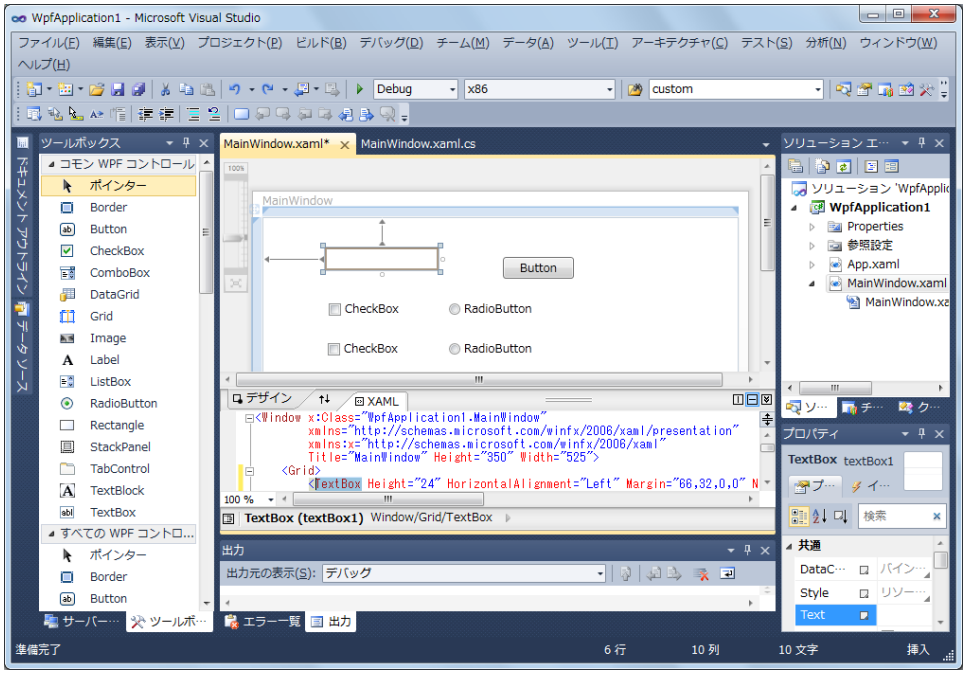
中央のドキュメント ウィンドウのフォーム デザイナ上にコントロールを配置していきます。

完成後とほぼ同じ見た目のまま、フォームをデザインできます。

この Windows フォーム アプリケーションに加え、Visual Studio 2010 では、WPF (Windows Presentation Foundation) を用いたアプリケーションが作成できます。

この場合、**[WPF アプリケーション]** というプロジェクト テンプレートを選択します。WPF アプリケーションでは、XML をベースにした XAML (Extensible Application Markup Language) と呼ばれるマークアップ言語を使ってユーザー インターフェイスをデザインします。

WPF を用いることで、ベクタ グラフィックスや動画を含むリッチなユーザーインターフェイスを持つアプリケーションを作成することができます。



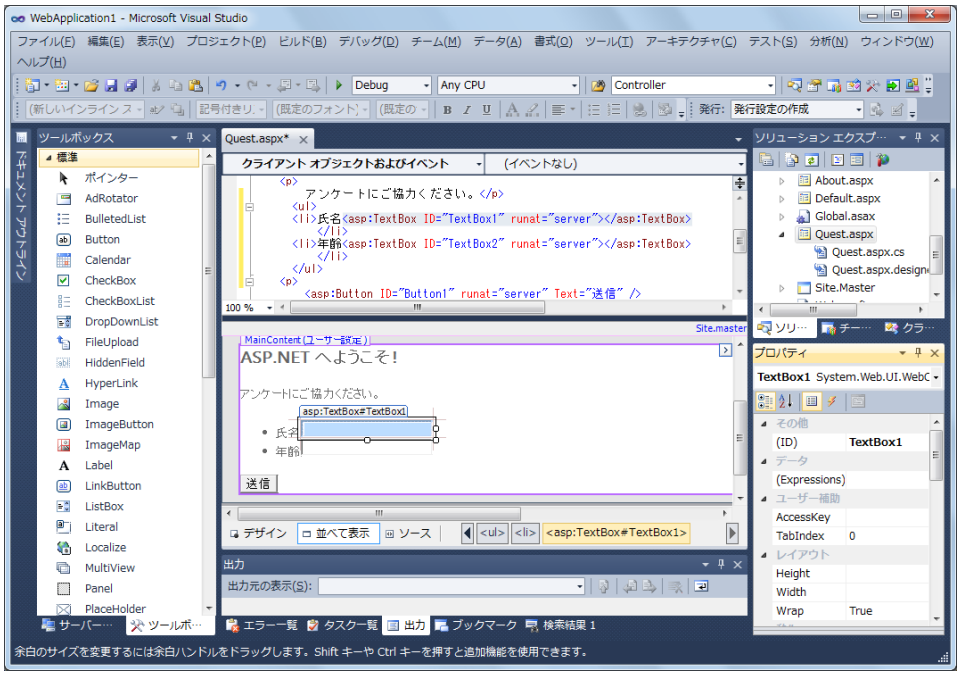
WPF アプリケーションの作成

「WPF フォーム」のデザイン画面を主体とした開発を行えます。

Web ベースのプロジェクト

Visual Studio 2010 では、ASP.NET 上で動作する Web アプリケーションを開発できます。

プロジェクト テンプレートは「ASP.NET Web アプリケーション」を選択します。ASP.NET Web アプリケーションでは、「Web フォーム」デザイン画面を主体とした開発が行えます。



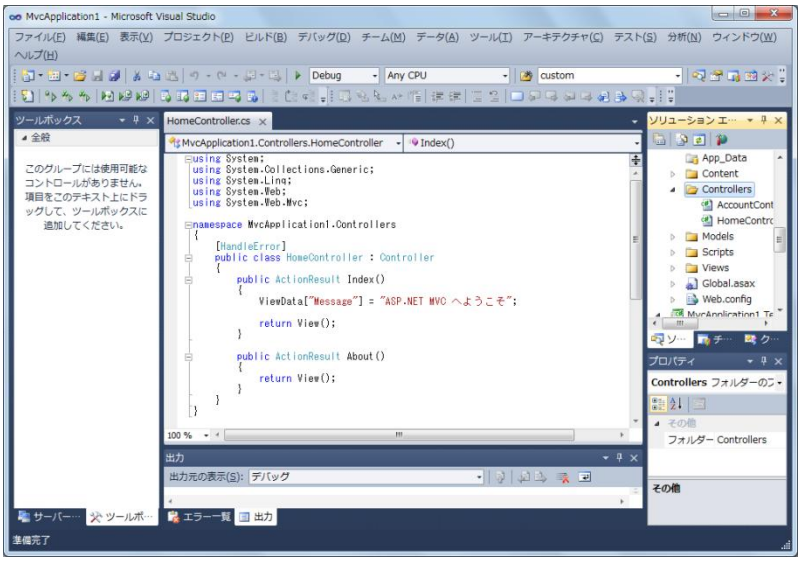
「Web フォーム」というデザイン画面を主体とした開発

Web アプリケーションも Windows フォーム アプリケーションと同じように、

フォーム デザイン画面を用いた開発を行います。

この ASP.NET　Web アプリケーションに加え、Visual Studio 2010 では、ASP.NET MVC 2 Web アプリケーションも作成できます。

ASP.NET MVC は、Model - View - Controller モデルに基づくフレームワークで、機能を分離しているため単体テストが行いやすい作りになっています。また、従来の ASP.NET 開発で使用していたサーバー コントロールは使わないので、最終的な出力結果に余計なコードが表示されることがありません。そのため、クライアントサイド開発も行いやすくなっています。

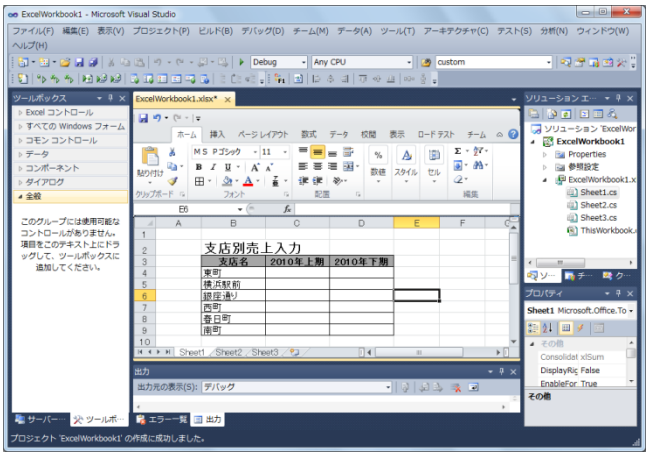


MVC パターンによる Web アプリケーション開発

Microsoft Office system プロジェクト

Visual Studio 2010 では、前述したように Professional 以上で Microsoft Office system 向けのプロジェクトを作成できます。「Excel 2010 ブック」や「Word 2010 ドキュメント」などのテンプレートを選択すれば、Excel や Word を利用したアプリケーション (Office ソリューション) が作成できます。

また、「Excel 2010 アドイン」「Outlook 2010 アドイン」などのテンプレートでは、Excel や Microsoft® Office Outlook® を拡張するアドインを作成できます。

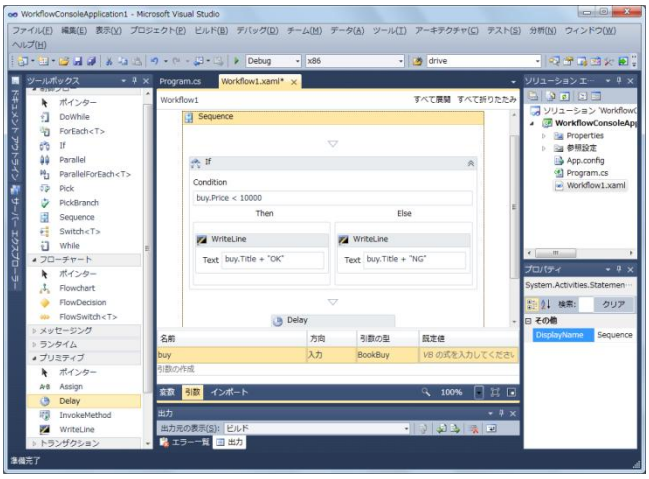


Excel 2010 ブック アプリケーションの開発例

WF (Windows Workflow Foundation) プロジェクト

ワークフロー (処理の流れ) を実装するためのテクノロジである Windows Workflow Foundation (WF) を利用するアプリケーションを作成できます。

プロジェクト テンプレートは「ワークフロー コンソールアプリケーション」を選択します。WF の開発においても、「WF デザイナ」というデザイン画面を使用した開発が行えます。

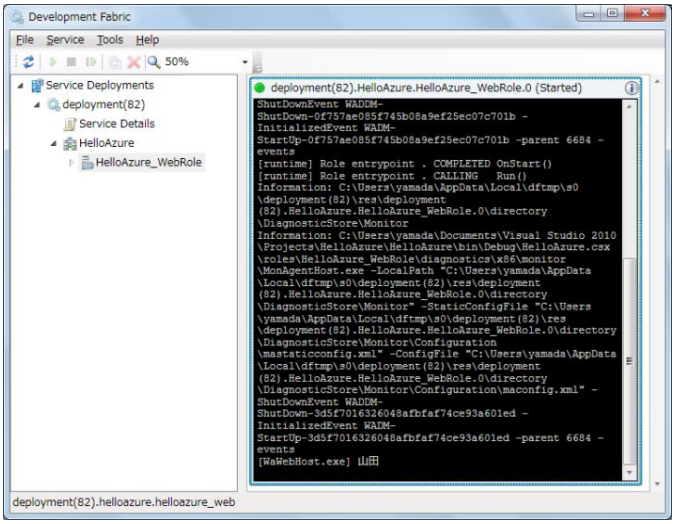


WF デザイナを活用した WF アプリケーションの開発

Windows Azure プロジェクト

Microsoft が提供しているクラウド サービス である Windows Azure を使ったアプリケーションを作成できます。開発にあたっては事前に Windows Azure Tools の有効化が必要です。プロジェクトテンプレートは、「Windows Azure Cloud Service」を選択します。

開発自体は、ASP.NET Web アプリケーション 開発と同様に行うことができ、ローカルの開発環境で実行することも可能です。また、ソリューション エクスプローラー からクラウド サービス プロジェクトを右クリックして、表示されるコンテキストメニューの [発行] を選択して、クラウド環境へ配置することもできます。

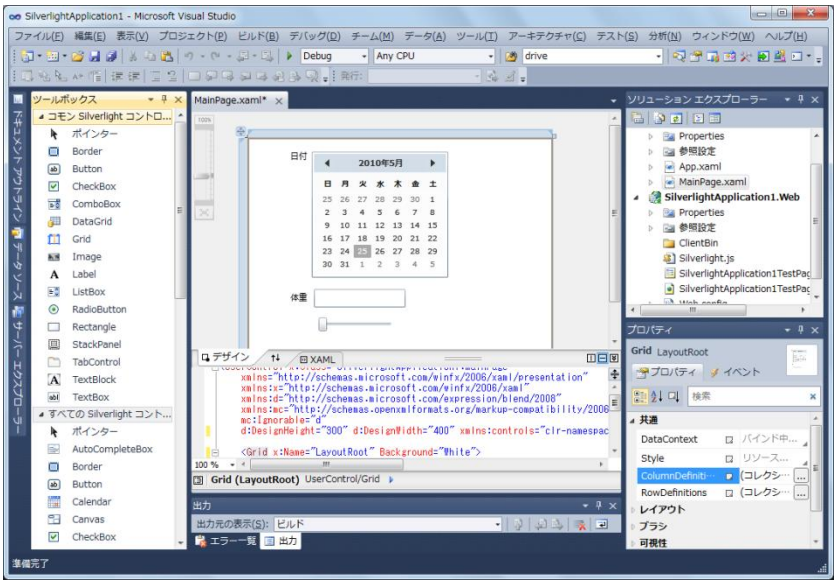


Development Fabric UI での動作状況確認

Silverlight プロジェクト

マルチプラットフォーム対応の RIA 実行環境である Silverlight を使ったアプリケーションを作成できます。Visual Studio 2008 では Silverlight アプリケーションを開発するために、ツールキットを追加インストールする必要がありましたが、Visual Studio 2010 では標準で Silverlight 3 アプリケーションを開発できるようになりました。

プロジェクト テンプレートは「Silverlight アプリケーション」を選択します。



Silverlight アプリケーションの開発

セットアップ と配置プロジェクト

作成するアプリケーションを配布するためのセットアップ プログラムである Windows インストーラ ファイル (.msi ファイル) を作成できます。

選択するプロジェクト テンプレートは「セットアップ プロジェクト」です。

作成したセットアップ プログラムでは、"Program Files" フォルダへのアプリケーションのインストールから [スタート] メニューへのショートカット リンクの登録まで、一般的なインストーラの機能を持たせることができます。

そのほかのプロジェクト

そのほかにも、SQL Server を利用するためのデータベース プロジェクトや SharePoint アプリケーションを作成するためのプロジェクト、さらにレポート アプリケーションを作成するためのプロジェクト、後ほど紹介しますが、アプリケーションのテストを実装するためのプロジェクトなどが含まれています。

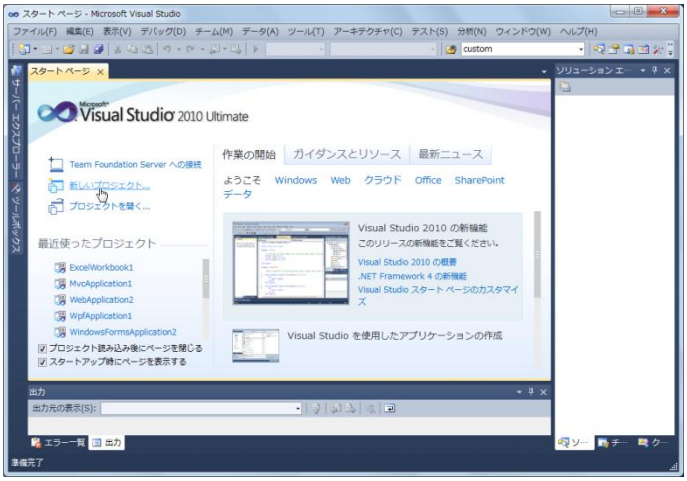
なお、SharePoint は Web ベースでコラボレーションを行うためのプラットフォーム環境です。Visual Studio では、SharePoint アプリケーションを実装するためのプロジェクト テンプレートとして、ワークフロー、リスト定義、サイト定義、イベント レシーバーなどを提供しています。

プロジェクトの新規作成

プロジェクトを新規作成する

プロジェクトを新規作成する方法は 2 つあります。

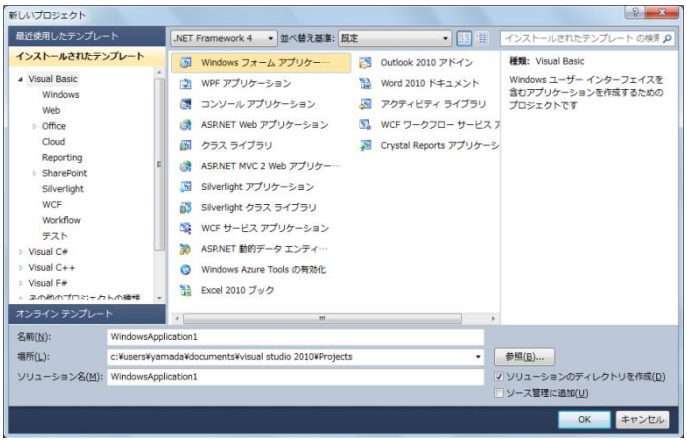
1 つは、Visual Studio 2010 を起動して(**[スタート]** メニュー - **[すべてのプログラム] - [Microsoft Visual Studio 2010] [Microsoft Visual Studio 2010]** を実行して)、**[スタートページ]** の左上にある「新しいプロジェクト…」と記述されたリンクをクリックすることです。



[スタートページ] からプロジェクトの新規作成を行う

もう 1 つの方法は、メニュー バー から **[ファイル] - [新規作成] - [プロジェクト]** を選択することです。これにより **[新しいプロジェクト]** ダイアログが表示されます。

[プロジェクトの新規作成] ダイアログ ボックスでは、まず左側の **[プロジェクトの種類]** からプログラミング言語を選択します。
.NET により Windows アプリケーションを開発する場合は、Visual C# や Visual Basic を選択することが一般的です。
なお、従来の開発資産を活用する場合などでは Visual C++ (CLR) を選択することもあります。Visual Studio 2010 では従来の Win32 API ベースの開発も行えますが、この場合は Visual C++ を選択します。



[プロジェクトの新規作成] ダイアログ ボックス

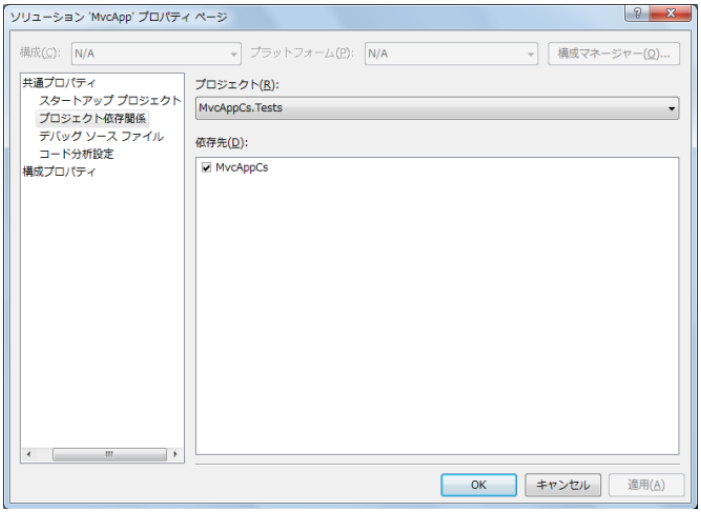
Visual C++ を使用すると、従来の MFC (Microsoft Foundation Class) や ATL (Active Template Library) が利用できます。Web アプリケーションでは、Visual C# や Visual Basic を選択します。
次に、右側の **[テンプレート]** から、目的に合ったアプリケーション種類のプロジェクト テンプレートを選択します。

Visual Studio 2010 ではアプリケーションを動作させる .NET Framework のバージョンを 2.0、3.0、3.5 および 4 の中から選択することができます。
ダイアログの右上に表示されているドロップダウン リストより .NET Framework のバージョンを選択することによって、動作環境となる .NET Framework を指定することが可能です。
その際、該当の .NET Framework のバージョンで提供されているもののみがテンプレートに表示されます。たとえば WPF は .NET Framework 3.0 以上で提供されている機能のため、ドロップダウンリストで .NET Framework 2.0 を選択した場合、テンプレートには表示されません。

そして、**[名前]**、**[場所]**、**[ソリューション名]** を指定します。
ソリューションについては後述します。
[ソリューションのディレクトリを作成] チェック ボックスにはチェックを入れたままで構いません。
[場所] には、プロジェクトを作成するフォルダを指定します。最後に **[OK]** ボタンを押すと、プロジェクトが新たに作成されます。

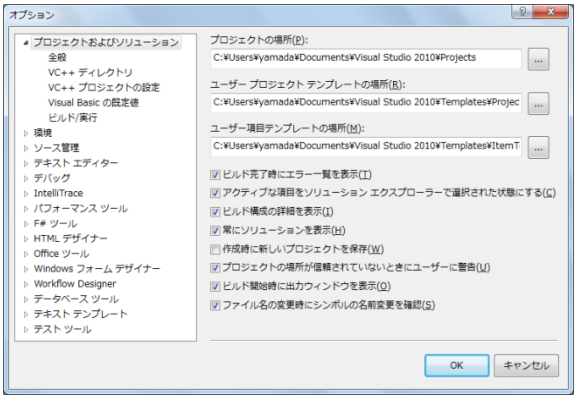
ソリューションとプロジェクトの違い

先ほど **[ソリューションのディレクトリを作成]** チェックボックスにはチェックを入れて **[ソリューション名]** を指定しました。
このような指定をした際には、プロジェクトと一緒にソリューションも作成されます。ソリューション（.sln）とは、前にも一度説明しましたが、複数のプロジェクトをまとめて管理する機能です。
たとえば、Windows アプリケーションのプロジェクトと、そのアプリケーション内で使用するライブラリのプロジェクトを、1 つのソリューションに含めることができます。
こうすると、アプリケーションをビルドする際に、それと同時にライブラリもビルドすることが可能になります。
またソリューションに含まれるプロジェクトの依存関係を指定することも可能ですので、これによりソリューションに含まれるプロジェクトが正しい順序でビルドされるようになります。



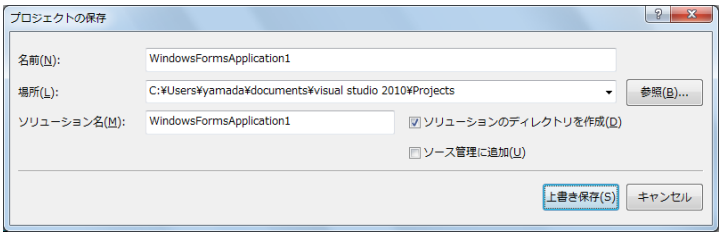
一時プロジェクトを利用する

プロジェクトを新規作成する際、プロジェクト テンプレートによっては「一時プロジェクト」を使用することもできます。
これは名前のとおり、一時的にプロジェクトを作成する機能で、プロジェクトを保存しない場合は、ハード ディスク上にプロジェクトを残しません。
ちょっとしたロジックや動作の確認を行いたい場合に便利です。一時プロジェクトを使用するかどうかは、メニュー バーの **[ツール] - [オプション]** から実行できる [オプション] ダイアログで設定できます。
使用したい場合は、**[プロジェクトおよびソリューション] - [全般]** カテゴリを選択して、**[作成時に新しいプロジェクトを保存]** チェック ボックスのチェックを外してください。



[オプション] ダイアログ ボックスにおける一時プロジェクトの指定

一時プロジェクトを使用している場合、先ほどの **[プロジェクトの新規作成]** ダイアログ ボックスでは、**[名前]**、**[場所]**、**[ソリューション名]** の入力欄が表示されません。その代わりに、作成したプロジェクトを保存しようとすると **[プロジェクトの保存]** ダイアログが表示され、プロジェクトの **[名前]**、**[場所]**、**[ソリューション名]** を指定できます。



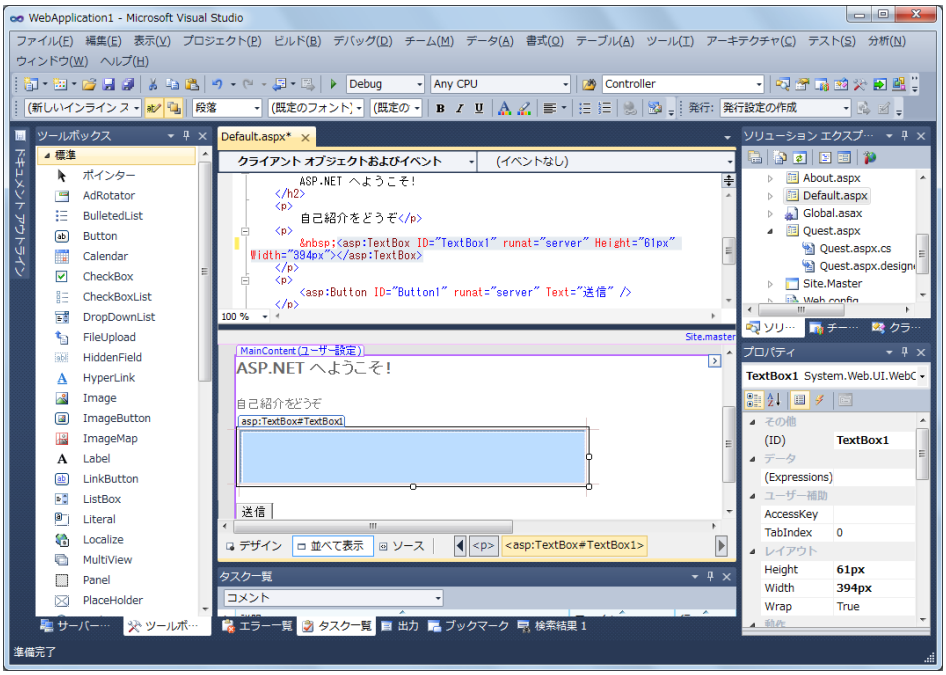
[プロジェクトの保存] ダイアログ ボックス

Web アプリケーションと Web サイトの違い

話題が少しそれますが、Web アプリケーションを作成するもう 1 つの方法について、ここで紹介しておきます。
Web アプリケーションを作成するプロジェクト テンプレートについてはすでに紹介しましたが、Visual Studio 2010 にはこのほかに、Web サイトを新規作成したり、既存の Web サイトに接続したりする機能が用意されています。この「Web サイト」は「プロジェクト」に似た開発単位ですが、Web サイトの作成に特化しており、それ自体が Web サーバー上のサイト (もしくはその 1 つのディレクトリ) を体現しているという特徴があります。
つまり、Web サイト (開発単位) 内に表示されるフォルダと、実際の運用サイトとして公開するフォルダが対応しています。この機能には、メニュー バーから **[ファイル] - [新規作成] - [Web サイト]** を選択することでアクセスできます。
実際に Web サイトを作成すると、プロジェクトではなく Web サイト (開発単位) が作成されます。もちろんこの際、プロジェクトと

同様に、ソリューションも自動作成されます。

Web サイト (開発単位) と Web アプリケーション プロジェクトでは、ASP.NET を使ったページ (.aspx ファイル) の作成方法自体は変わりませんが、実行形式の作成方法や配布の方法などが異なります (詳細については本書では割愛します)。



Web サイトの開発例

統合開発環境 (IDE) のウィンドウ

プロジェクトを作成したら、次はいよいよユーザー インターフェイスのデザインを行ったり、アプリケーションのロジックを記述したりと、本格的に Visual Studio 2010 の統合開発環境を使いこむ段階に入っていきます。

その説明 (「第 4 章アプリケーションの開発」) に入る前に、Visual Studio 2010 に用意されているさまざまなウィンドウをひととおり簡単に紹介しておきましょう。

Visual Studio 2010 には非常に多くの画面やウィンドウが用意されているため少し長くなりますので、ざっと読んで必要になったときに戻ってきてもう一度じっくり読むとよいでしょう。各種ウィンドウの表示/非表示は、基本的にメニュー バーの **[表示]** メニューの配下のメニューから切り替えることができます。

Visual Studio 2010 の主要なウィンドウ

[スタートページ] 画面 (ドキュメント ウィンドウ)

先ほどの「プロジェクトを新規作成する」でも紹介しましたが、Visual Studio 2010 を起動した際に最初に、中央のドキュメント ウィンドウに表示される内容です。

ここにはこれまで使用したプロジェクトの履歴、MSDN Online の更新情報などが表示されます。

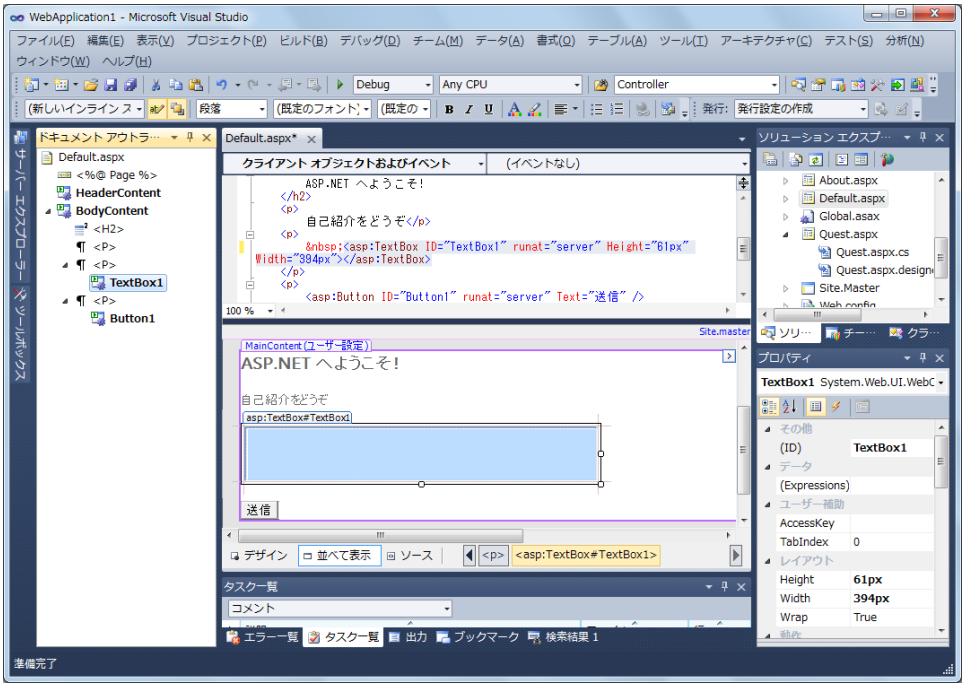
フォーム デザイナ/ コード エディタ (ドキュメント ウィンドウ)

統合開発環境のメインとなるウィンドウで、中央のドキュメント ウィンドウに表示されます。

編集する内容 (ソースコード、Windows フォーム、HTML、XML など) によって画面が変化します。前述の「プロジェクトの種類」でも、さまざまなデザイン画面があることを紹介しました。

たとえば Windows フォームの内容をレイアウト、デザインする Windows フォーム デザイナでは、コントロールの配置をほかのコントロールの位置に自動的に揃えたりする補助機能が搭載されており、これにより正確なレイアウトが行えます。

また、ソースコードを記述、編集するコード エディタでは、コード入力の補完を行う IntelliSense やコードスニペットによるキー入力の省略などコード入力の手間を省くためのいろいろな機能が備わっています。

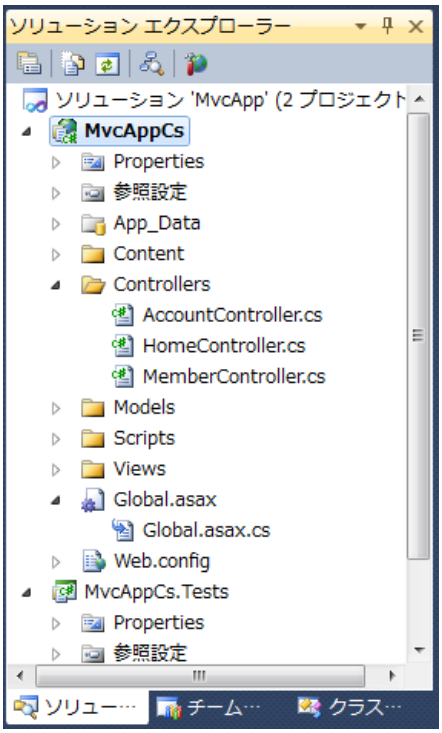


Visual Studio 2010 での作業の中心となるウィンドウ (Web フォーム デザイナの例)

[ソリューション エクスプローラー] ウィンドウ

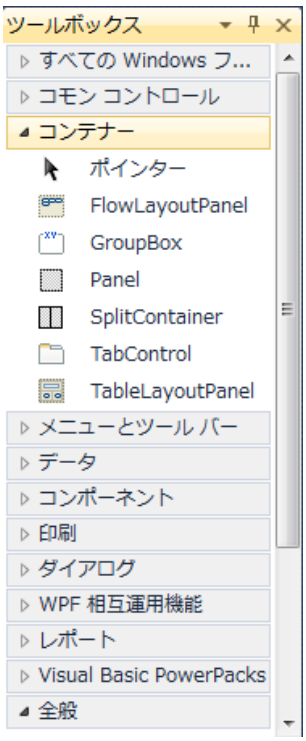
現在作成中のプロジェクトに含まれるソース ファイル群を参照できます。

プロジェクトのプロパティや、プロジェクトが参照する外部アセンブリ (.dll ファイル) の設定などもここから行えます。



[ツールボックス] ウィンドウ

Windows フォーム、WPF フォーム、Web フォームなどにドラッグ アンド ドロップでコントロールを貼り付ける際に、ここからコントロールをフォーム上にドラッグ アンド ドロップします。



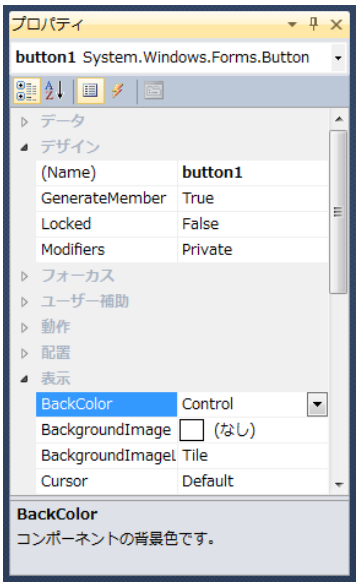
[プロパティ] ウィンドウ

[プロパティ] ウィンドウには、デザイン画面（フォーム デザイナ）上で現在選択しているコントロールのプロパティ群が表示されます。ここでプロパティの値を変更することで、コントロールの外観や振る舞いを調整できます。

また **[プロパティ]** ウィンドウでは、コントロールで発生するイベントの管理も行えます。

たとえばボタンが押されたときの処理を行いたい場合は、ボタンのクリック（Click）によってイベントが発生するよう指定します。

[プロパティ] ウィンドウの使い方については、次の「第 4 章 アプリケーションの開発」の中で説明します。

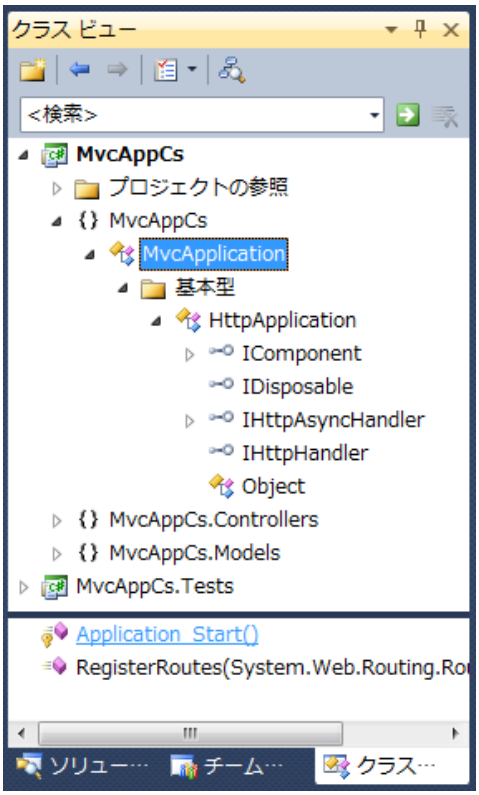


[クラス ビュー] ウィンドウ

[クラス ビュー] ウィンドウは、プロジェクトに含まれる名前空間やクラス階層をツリー形式で表示します。

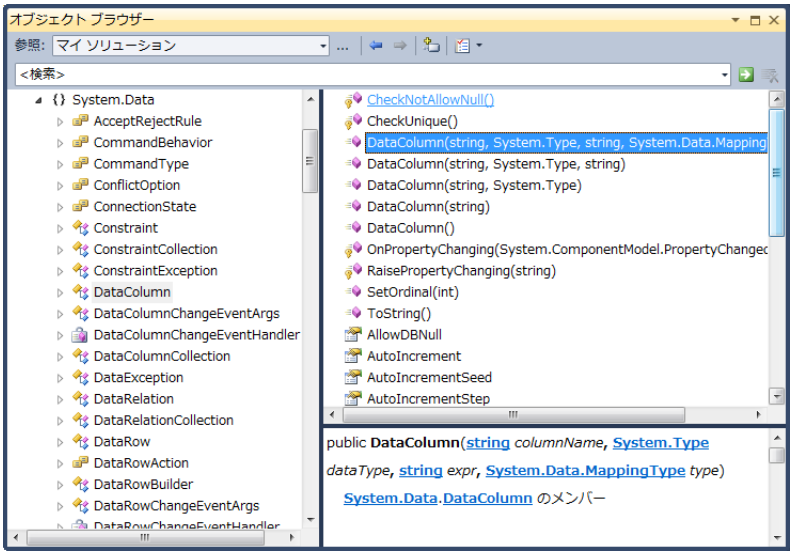
[ソリューション エクスプローラー] ウィンドウでもプロジェクトに含まれるソースファイル群を参照することは可能ですが、クラスに含まれるメソッドやプロパティまでは参照できません。

特定のメソッドを参照したいときなどでは、この **[クラス ビュー]** が役に立ちます。



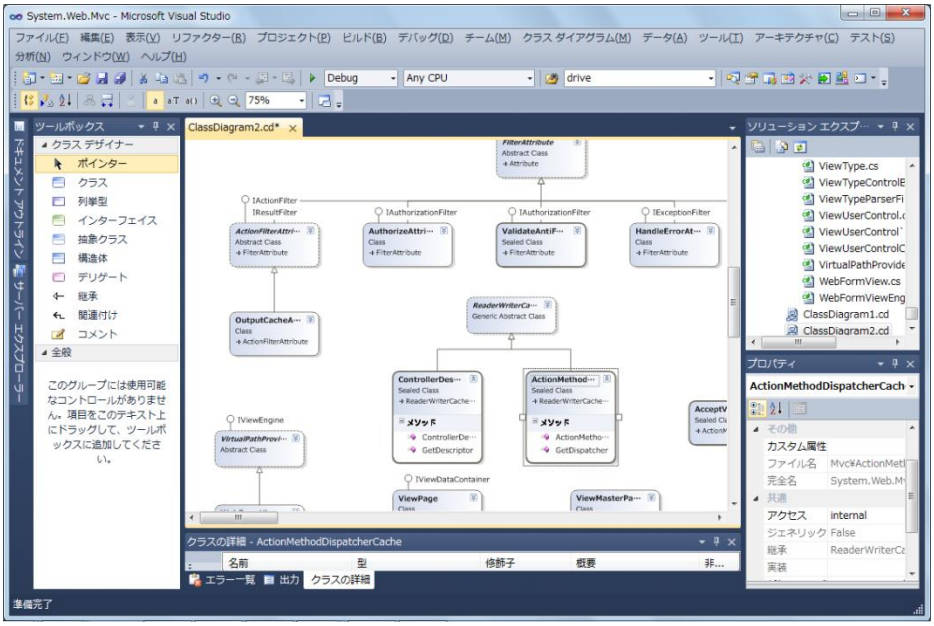
[オブジェクト ブラウザ] 画面 (ドキュメント ウィンドウ)

プロジェクトとプロジェクトで参照するオブジェクトの名前空間、クラスに含まれるメソッドやプロパティ情報を一覧表示できます。



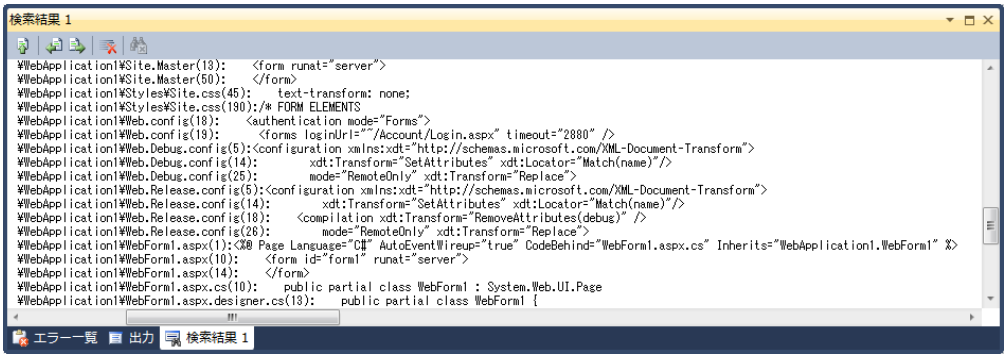
[クラス ダイアグラム] 画面（ドキュメント ウィンドウ）

[クラス ダイアグラム] 画面は、クラスを設計するためのツールです。クラス情報を表示したり、クラスを追加したクラスに含まれるメンバを追加/削除したりできます。クラス ダイアグラムでは、クラスや構造体、列挙型などが色分けされて表示され、クラスの継承関係がビジュアルに表示されますので、プロジェクトに含まれるクラスの関係を容易に把握できます。



[検索結果] ウィンドウ

[検索結果] ウィンドウは、統合開発環境上で検索を行った結果が表示されるウィンドウです。



[サーバー エクスプローラー] ウィンドウ

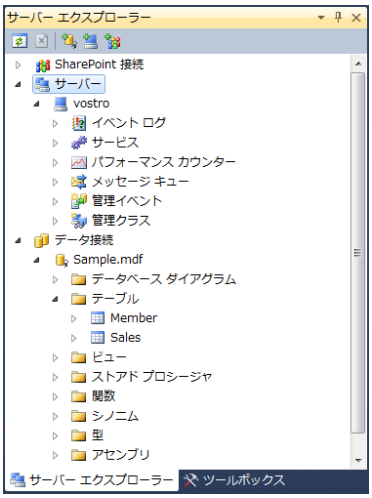
[サーバー エクスプローラー] ウィンドウは、Visual Studio 2010 を実行している PC やネットワーク上にあるサーバーなどで作動しているさまざまなサービス、たとえばイベント ログや Windows サービスなどの情報の参照や操作が行えます。
データベースの情報も参照/操作できるので、**[サーバー エクスプローラー]** ウィンドウを活用することで、データベースを利用するアプリケーションの作成が容易になります。

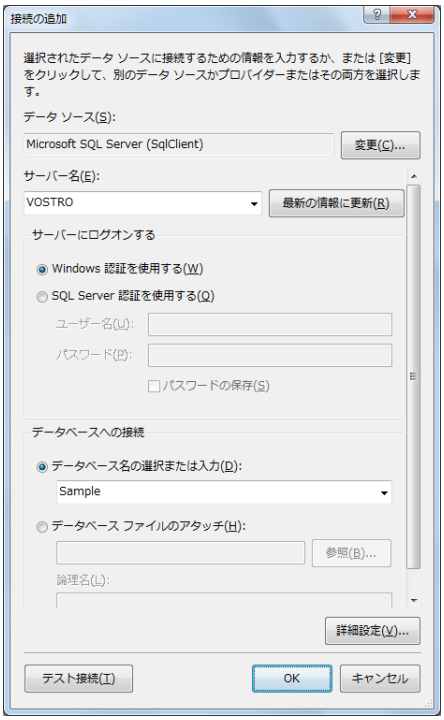
[サーバー エクスプローラー] ウィンドウは、開発環境の中からデータベースを直接操作でき、非常に便利ですので、その内容についてもう少し詳しく説明しましょう。
[サーバー エクスプローラー] では、各項目はツリー構造で管理され、ツリーのルートには「サーバー」ノードや「データ接続」ノードがあります。

「サーバー」ノードの直下には、デフォルトで Visual Studio 2010 を実行しているローカル マシンのノードが追加されており、そのノードの配下で「イベントログ」、「サービス」、「パフォーマンス カウンタ」、「メッセージキュー」、「管理イベント」、「管理クラス」などのシステム リソースを管理できます。
たとえば「イベント ログ」ノードではイベントログの詳細情報を取得できます。「サービス」ノードでは Windows サービスの一覧を表示でき、サービスの状態を確認できます。デフォルトのローカル マシン ノードに加え、ほかのサーバーマシンを追加することもできます。

「データ接続」ノードでは、ローカル マシン上で動作している SQL Server やネットワーク上の SQL Server のデータベースを管理 (データベースの新規作成や、データの入力、変更など) したり、そのデータベースへの接続をアプリケーションに取り込んだりできます。データベースに接続して何らかの処理を行うことが多いビジネス向けのアプリケーションでは役立つでしょう。

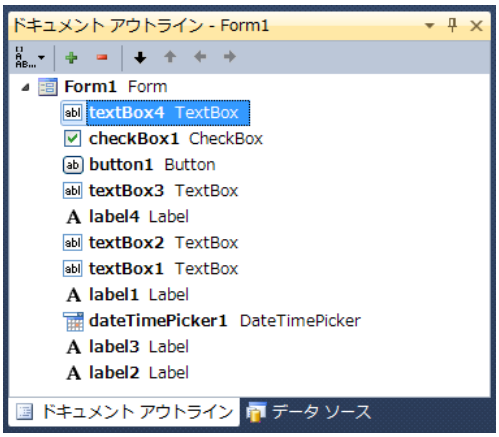
「データ接続」ノードから既存のデータベースに接続するには、「データ接続」ノードを右クリックして、表示されるコンテキスト メニューから「接続の追加」を選択します。
これにより **[接続の追加]** ダイアログ ボックスが表示されますので、SQL Server の名前やデータベース名を指定することで、任意の SQL Server データベースへの接続を追加できます。





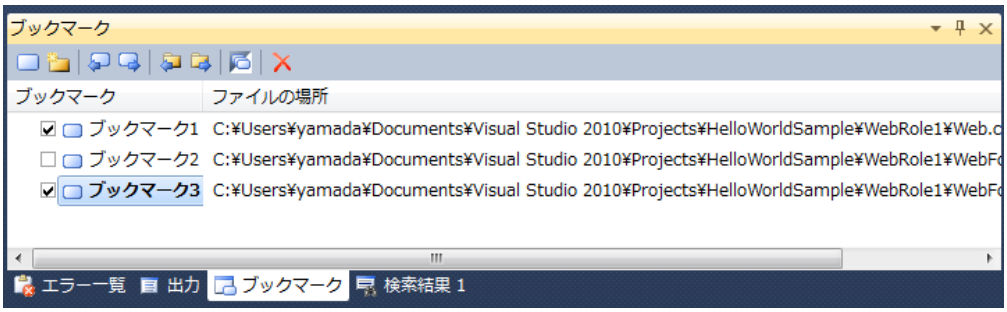
[ドキュメント アウトライン] ウィンドウ

Windows フォームや Web フォームに配置しているコントロールをツリー形式で一覧表示します。



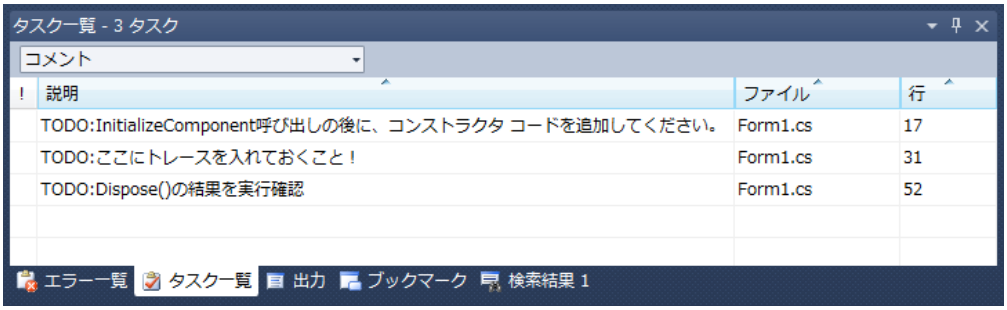
[ブックマーク] ウィンドウ

[ブックマーク] ウィンドウは編集中のソース ファイルにセットしたブックマークを一覧表示します。
ブックマークとは、再アクセスしたい行に付ける目印のことです。ブックマークした行への移動は、**[ブックマーク]** ウィンドウから目的のブックマークをクリックしても、ツール バーからブックマークの移動ボタンを使っても行えます。
一度ブックマークを付けると、プロジェクトを閉じて再度オープンしても前回セットしたブックマークは保存されています。



[タスク一覧] ウィンドウ

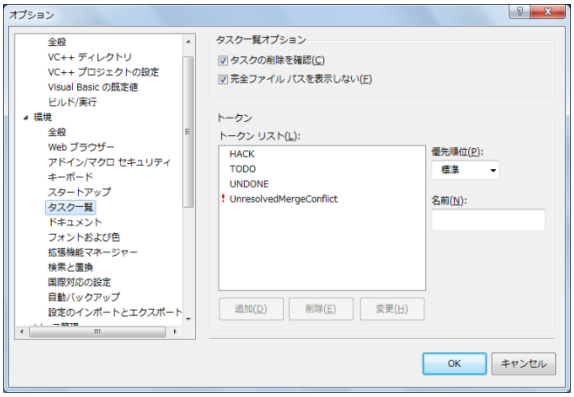
コード中のコメントに「TODO:」のような特殊なトークンを記述すると、そのコメントが **[タスク一覧]** ウィンドウに表示されます。
デフォルトでは、「HACK」、「TODO」、「UNDONE」などのトークンがあらかじめ用意されています。



たとえば Visual C# であれば、

```
// TODO: ここにトレースをいれておくこと!
```

といったコメント行を作成すると、このコメントが **[タスク一覧]** ウィンドウに表示されるようになります。
作業中のリマインダとして利用したり、複数のスタッフで開発を行っている場合は注意事項などの連絡やメモとして利用したりできます。
独自のトークンを追加することもできます。これには、メニュー バーの **[ツール] - [オプション]** から **[オプション]** ダイアログを表示し、**[環境] - [タスク一覧]** カテゴリを選択して、**[トークン リスト]** に追加します。



ビルドやデバッグに関連するウィンドウ

次に、アプリケーションのビルドおよびデバッグ時によく参照するウィンドウをリストアップしておきましょう。なお、以下のウィンドウのうち、**[ローカル]** ウィンドウ/**[自動変数]** ウィンドウ/**[ウォッチ]** ウィンドウ/**[呼び出し履歴]** ウィンドウはほかのウィンドウと異なり、メニュー バーの **[デバッグ] - [ウィンドウ]** メニューの配下から表示/非表示を切り替えます。

[エラー一覧] ウィンドウ

ビルド エラー、警告、メッセージなどが表示されます。エラーが表示されている行をダブル クリックすると、該当する行にテキスト カーソル (キャレット) が移動します。

[出力] ウィンドウ

ビルドの経過を表示します。また、プログラムのデバッグ時にトレース出力を行った場合も、そのトレース内容がこのウィンドウに表示されます。

[ローカル] ウィンドウと [自動変数] ウィンドウ

デバッグ時に実行をブレーク (中断) している位置のローカル変数を表示します。

[ウォッチ] ウィンドウ

監視/参照したい変数 (オブジェクト) を指定すると、**[ローカル]** ウィンドウと同じようにその変数の内容を表示できます。変数のほかに式を記述することも可能です。

[呼び出し履歴] ウィンドウ

アプリケーションが起動してから現在呼び出されているメソッドにたどり着くまでのメソッドの呼び出し履歴を確認できます。

[コマンド] ウィンドウ

キーボードを主体にした開発環境の操作が行えます。
[コマンド] ウィンドウ内には「>」という入力を促すプロンプトが表示され、ここにキーボードからコマンドを入力することで、デバッグ中は変数の表示、デバッグの開始、ステップ実行など、さまざまな操作が行えます。

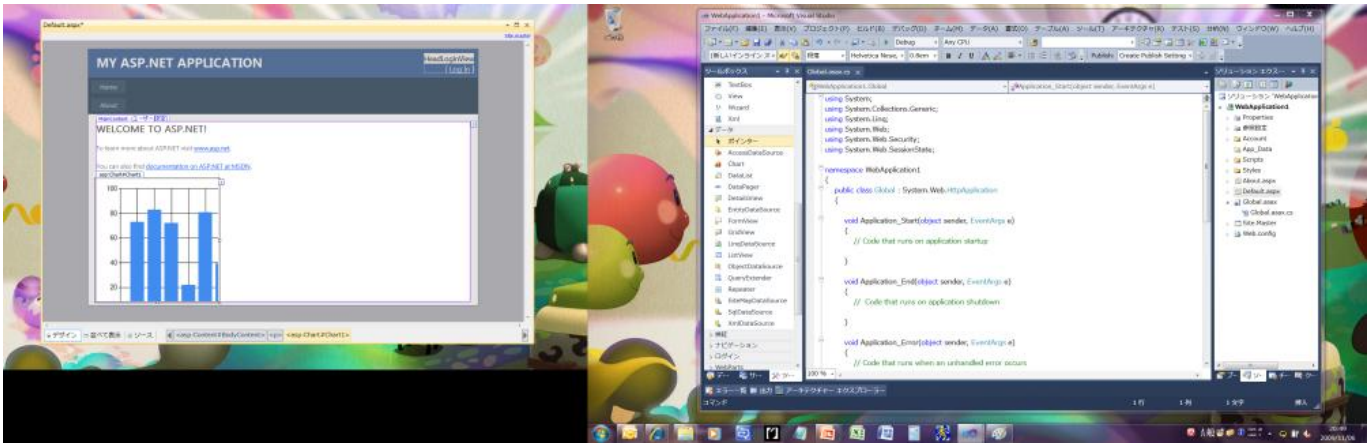
これらのウィンドウのうち、**[エラー一覧]**、**[出力]** ウィンドウについては、「第 5 章 ビルド」の中で取り上げます。
また、**[ウォッチ]**、**[ローカル]** ウィンドウについては、「第 6 章 デバッグ」でもう少し詳しく説明します。

ウィンドウの配置レイアウトを変更する

Visual Studio 2010 の統合開発環境に配置するこれらのウィンドウは、開発者が使いやすいように自由に配置レイアウトを変更できます。
複数のウィンドウを重ね合わせてタブ表示にしたり、参照しない場合は自動的に隠すように設定したりすることも可能です。
実際に配置レイアウトを変更するには、それぞれのウィンドウのタイトル (キャプション) 部分をマウスでドラッグするだけです。
また、ウィンドウの配置レイアウトは、アプリケーションのデザイン時とデバッグ時で異なるレイアウトを設定できます。
それぞれの場面で最適のウィンドウ レイアウトを設定しましょう。
ウィンドウのレイアウトは、メニュー バーの **[ウィンドウ] - [ウィンドウ レイアウトのリセット]** を実行することで、いつでも初期状態に戻すことができます

マルチ モニターへの対応

Visual Studio 2010 は、マルチ モニターに対応しています。複数のモニターを使って、広いスクリーン領域で開発を行うことができます。
たとえば、UI のデザインを行う際、デザイン画面はメインの IDE ウィンドウに大きく表示させて、プロパティ ウィンドウは別のモニター上に表示させて作業を行うような使い方が可能です。

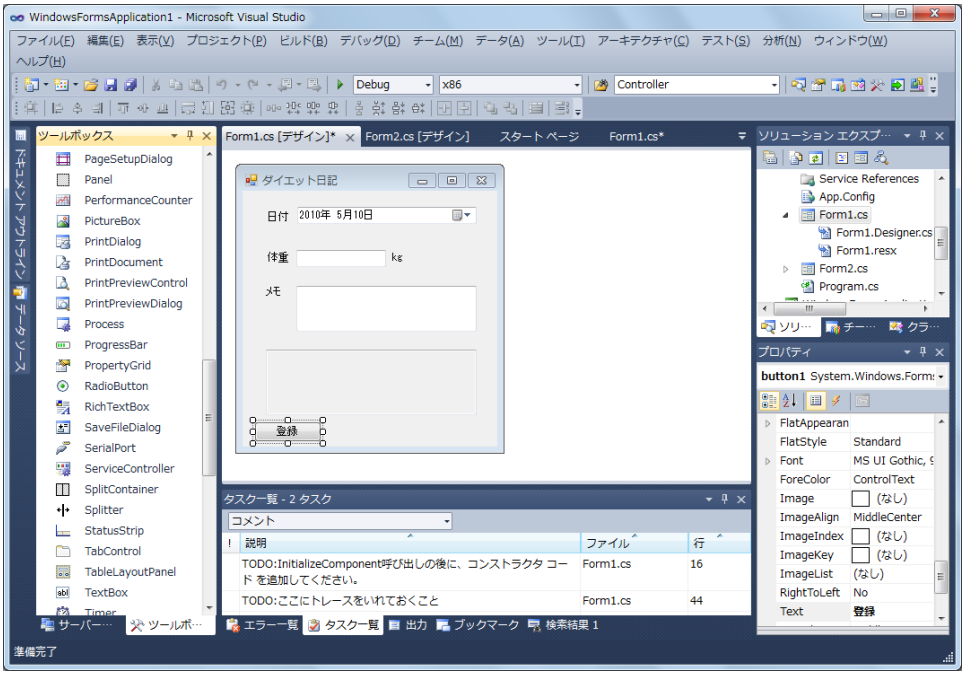


第 4 章 アプリケーションの開発

ここまでに、Visual Studio 2010 で作成できる主なプロジェクトの種類を紹介し、実際にプロジェクトを新規作成する方法を示しました。
本章では Windows アプリケーションを作成する例を示しながら、Visual Studio 2010 でアプリケーションを開発する様子を紹介します。

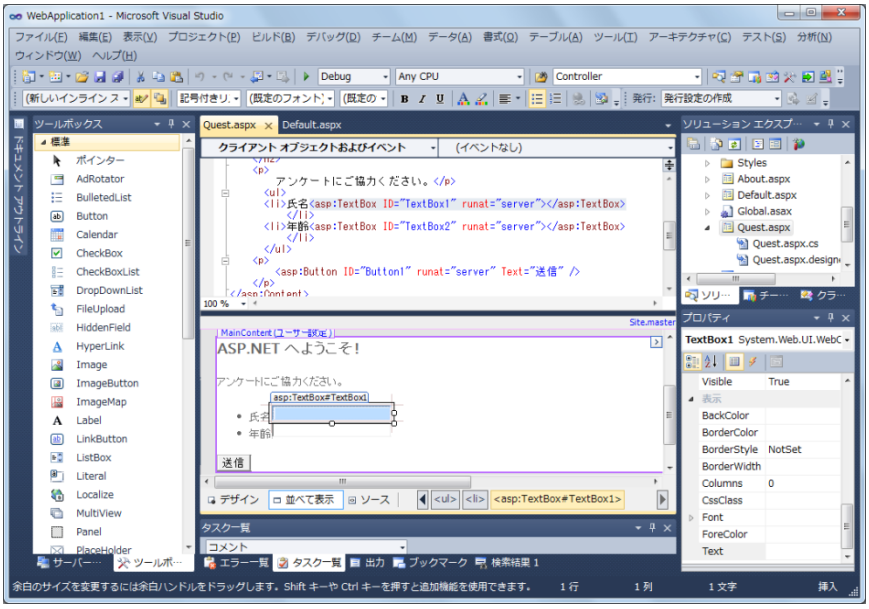
統一された開発スタイル

Windows アプリケーションや Web アプリケーションなど、エンド ユーザーが何らかの操作を行うことが主体となるアプリケーションを作成する場合、まずアプリケーションで必要となるユーザーインターフェイスを作成することから始めます。
Windows アプリケーションでは、先ほど紹介した Windows フォーム デザイナ上に、ボタンやテキストボックスなどのコントロールを配置してユーザー インターフェイスを作成していきます。
このコントロールに対して、エンド ユーザーは操作を行います。



Windows フォーム デザイナによる開発

[Windows フォーム アプリケーション] プロジェクト テンプレートを選択してプロジェクトを新規作成すると、そのプロジェクト内に Windows フォームを含むソース ファイルが自動生成されます。
前にも述べましたが、このようにフォーム上にコントロールを配置して、ユーザー インターフェイスを作成していく方法は、Visual Studio によるあらゆる開発で一貫しています。
Windows フォーム アプリケーションだけでなく、Web アプリケーションや WPF アプリケーションの場合も同様に、フォーム デザイナ上にドラッグ アンド ドロップでコントロールを配置しながら開発を進めていきます。
このようにして作成したフォームの中身は、実際には、Windows フォームなら VisualC# や Visual Basic などのコード、Web フォームなら HTML コード、WPF フォームなら XAML コードなどです。
したがって必要であれば、フォーム デザイナの表示をコード エディタの表示に切り替えて、そのフォームのコード内容を参照することも可能です。
特に Web フォームや WPF フォームの場合は、コードとフォーム デザイン画面を同時に表示することが可能になっています。



Web アプリケーションにおけるソースとデザインの同時表示

フォーム デザインによるユーザー インターフェイスの作成

[Windows フォーム アプリケーション] プロジェクトを新規作成後、**[ソリューション エクスプローラー]** ウィンドウ内にある Windows フォーム項目 (Visual Basic では「Form1.vb」、Visual C# では「Form1.cs」) をダブルクリックすると、中央のドキュメント ウィンドウに Windows フォーム デザイナが表示されます。

そこにフォームが開かれると、[ツールボックス] ウィンドウ内に配置可能なコントロール群やコンポーネント群が表示されます。

なお、前の「第 3 章 プロジェクトの作成」でも説明しましたが、**[ツールボックス]** ウィンドウが表示されていない場合は、メニュー バーから **[表示] - [ツールボックス]** を選択すれば表示されます。

[ツールボックス] ウィンドウ内のコントロールは、用途に応じて **[コモンコントロール]**、**[コンテナー]**、**[メニューとツールバー]** などのカテゴリごとに分類されています。

この中にあるコントロールを、マウスでフォーム上にドラッグ アンド ドロップすることで、アプリケーションに必要なコントロールをフォームに配置していきます。

Windows フォームで利用できる [レイアウト] ツール バー

アイコン	名称	補足説明
	グリッドに合わせて整列	レイアウト モードが「SnapToGrid」のときのみ有効です。 ※レイアウト モードについては後述します
	左揃え	選択したコントロールを左端に揃えます
	左右中央整列	選択したコントロールを左右中央に揃えます
	右揃え	
	上揃え	

	上下中央整列	
	下揃え	
	幅を揃える	選択したコントロールの幅を同じにします
	高さを揃える	
	同じサイズに揃える	幅と高さをすべて同一にします
	グリッドのサイズに揃える	レイアウト モードが「SnapToGrid」のときのみ有効です
	左右の間隔を均等にする	コントロールとコントロールのすき間を均等に調整します
	左右の間隔を広くする	
	左右の間隔を狭くする	
	左右の間隔を削除する	
	上下の間隔を均等にする	
	上下の間隔を広くする	
	上下の間隔を狭くする	
	上下の間隔を削除する	
	左右中央揃え	
	上下中央揃え	
	最前面へ移動	コントロールの Z オーダー (コントロールが重なる順序) を変更して、最前面に移動させます
	最背面へ移動	
	セルの結合	主に「レポート アプリケーション」プロジェクトで使われる「レポート デザイナ」画面 (ドキュメント ウィンドウ) などで使用します。レポートに載せた表の内部のセルを結合できます
	タブ オーダー	※タブ オーダーについては後述します

[レイアウト] ツール バーでコントロールを揃える

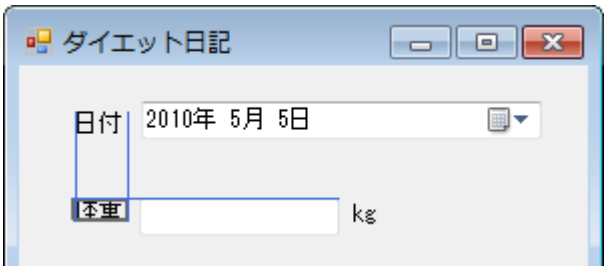
コントロールの配置方法 (レイアウト モード) としては、「SnapLines」と「SnapToGrid」が選択できます。

この選択は、メニュー バーの **[ツール] - [オプション]** を選択して、**[オプション]** ダイアログ ボックスを表示し、その左側のカテゴリ

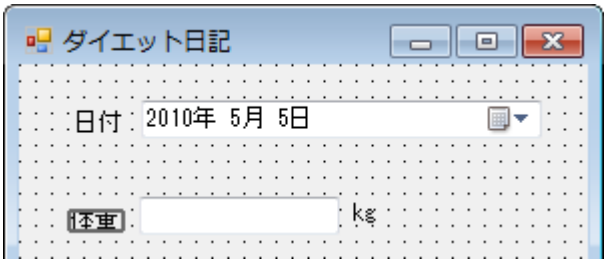
ツリーから **[Windows フォーム デザイナ] - [全般]** を選択し、**[レイアウト設定]** 内の **[LayoutMode]** プロパティで変更できます。デフォルトでは「SnapLines」が選択されています。

「SnapLines」の場合、コントロールをマウスでドラッグすると、ほかのコントロールの位置に合わせて位置を調整するためのガイドが表示されます。

一方、「SnapToGrid」を選択した場合は、フォーム上にグリッド (格子) が表示されるので、それに合わせてコントロールの位置を調整できます。



SnapLines 設定によるガイドの表示

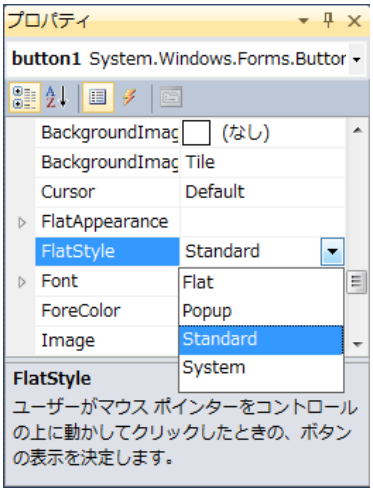


SnapToGrid 設定によるグリッド表示

[プロパティ] ウィンドウでプロパティを設定する

前述したとおり **[プロパティ]** ウィンドウでは、コントロールの外観や振る舞いを調整するプロパティを設定できます。

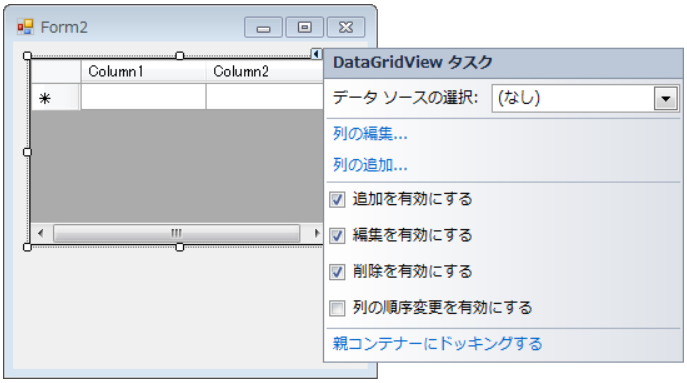
フォーム デザイナ上でコントロールを選択すると、そのコントロールが持つプロパティが **[プロパティ]** ウィンドウに表示されますので、各プロパティ項目の値を選択変更したり、入力したりするだけです。



[プロパティ] ウィンドウによるプロパティ設定

タスク メニューによりプロパティを設定する

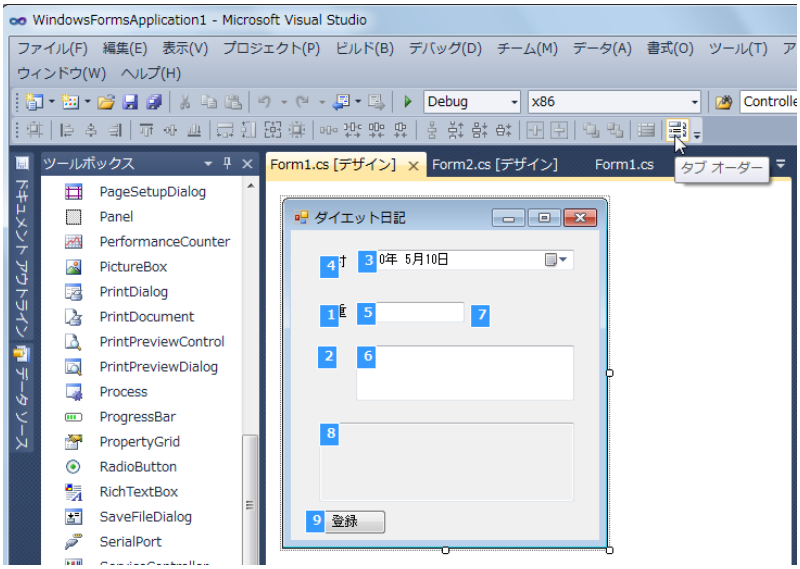
Windows フォームに配置するコントロールの中には、設定可能なプロパティ項目が非常に多岐にわたるものがあります。
このようなコントロールには、最もよく使われるプロパティ項目にのみすばやくアクセスするための手段が提供されています。
それがスマート タグから表示されるタスクメニューです。フォーム上に配置したコントロールの右上の部分にスマート タグが表示されます。
スマート タグをクリックするとタスク メニューと呼ばれるメニューが表示されます。コントロールに必要な設定をここから行うことが可能です。



スマート タグによるタスク メニューの表示

タブ オーダーを設定する

タブ オーダーは Windows フォーム上で Tab キーを押した際に、コントロールのフォーカスが移動する順番を指定するものです。
タブ オーダーを設定することにより、アプリケーションを利用するエンド ユーザーは、マウスでコントロールを選択しなくても、Tab キーを使ってコントロールを移動できるようになります。
タブ オーダーを設定するには、レイアウト ツール バーの「タブ オーダー」を使用します。
「タブ オーダー」ボタンをクリックすると、フォーム上に配置されたコントロールにタブ オーダーを示す数字が表示されます。
この状態で設定したいタブ オーダーの順番でコントロールを次々とクリックしていきます。
クリックされた順番で番号が振られていき、タブ オーダーがセットされます。



タブ オーダーの設定

ユーザー コントロールを作成する

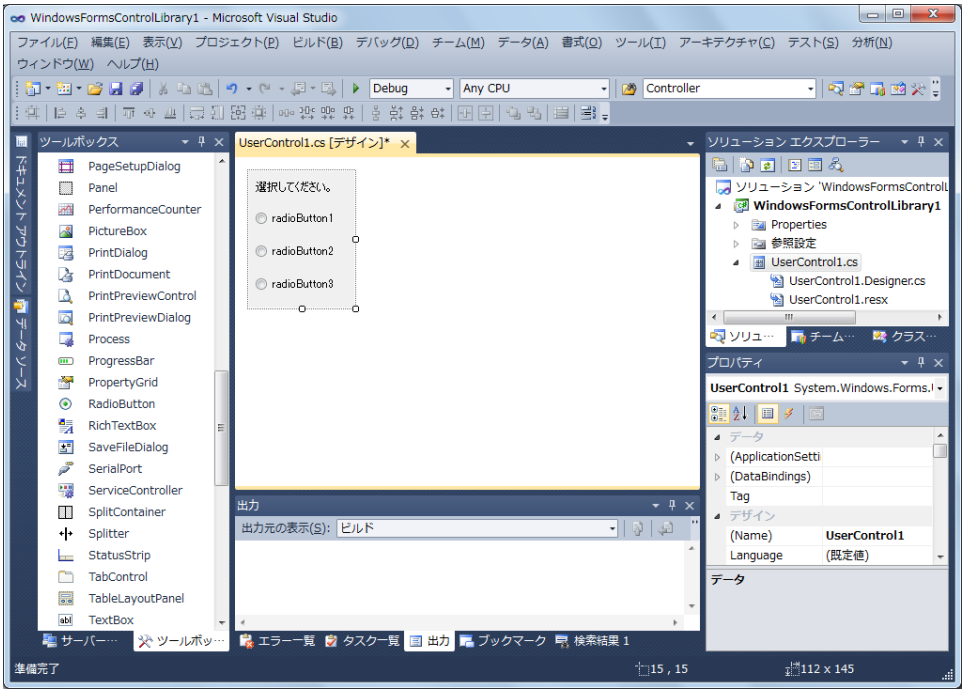
通常、フォーム上に配置するコントロールには **[ツールボックス]** に標準で搭載されているものを使いますが、独自の「ユーザー コントロール」を作成して使うことも可能です。

ユーザー コントロールでは、複数のコントロールをまとめて 1 つのコントロールとしたり、まったく独自の機能を持つコントロールを作成したりできます。

ちなみに、標準のコントロールを拡張して独自の機能を加える、「カスタム コントロール」と呼ばれる独自のコントロールもありますが、これについては本書では説明を割愛します。

ユーザー コントロールを集めたライブラリを作成すれば、ほかのプロジェクトと共有して再利用することが可能です。このようなユーザー コントロール ライブラリを作成するには、**[新しいプロジェクト]** ダイアログ ボックスにある「Windows フォーム コントロール ライブラリ」プロジェクト テンプレートを選択します (WPF アプリケーションの場合は「WPF カスタムコントロール ライブラリ」プロジェクト テンプレートを選択します)。

ユーザー コントロールの作成は基本的には Windows フォームの作成とさほど変わりありません。作成したユーザー コントロールをビルドすると、**[ツールボックス]** ウィンドウに登録され、Windows フォームなどで利用できるようになります。



ユーザー コントロールの作成

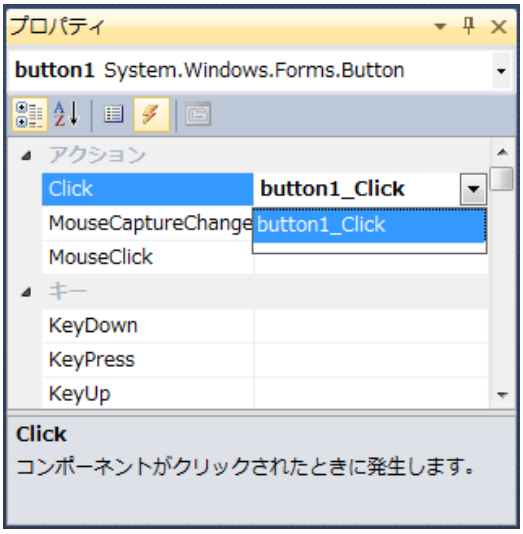
入力補助機能、編集機能を活用したコーディング

ユーザー インターフェイス部分が出来上がれば、次にエンド ユーザーがコントロールを操作したときに実行される処理ロジックのコードを記述します。Visual Studio 2010 では、次に示すいくつかの機能を使ってスマートにコーディングが行えます。

コントロールのイベント処理を追加する

フォーム上に配置したコントロールに対してエンド ユーザーが何らかの操作を行うと、アプリケーション内部ではイベントが発生します。そのイベントを処理するのがイベント ハンドラです。
たとえばフォーム上に配置したボタンが押されると、Click イベントが発生します。それを処理するのは Click イベント ハンドラです。したがって、エンド ユーザーがクリックした際に何らかの処理を実行したいという場合には、この Click イベント ハンドラに処理コードを記述することになります。

実際に Click イベント ハンドラに処理コードを記述するには、フォームデザイナ上で Button コントロールをダブル クリックします。すると、ドキュメントウィンドウがフォーム デザイナからコード エディタに切り替わり、そのフォームのコードに Click イベント ハンドラが自動的に追加されますので、そこにコードを追記してきます。
一方、そのほかのイベントに対応するイベント ハンドラを追加したい場合は、フォーム デザイナ上でコントロールを選択した状態で **[プロパティ]** ウィンドウの上部にある「稲妻」ボタンをクリックしてイベント一覧を表示し、そこからそのイベント名をダブル クリックすることで、イベント ハンドラが追加されます。
たとえば、ボタンのフォーカスが外れた際に特別な処理が必要となる場合では、”Leave” イベント ハンドラを作成します。

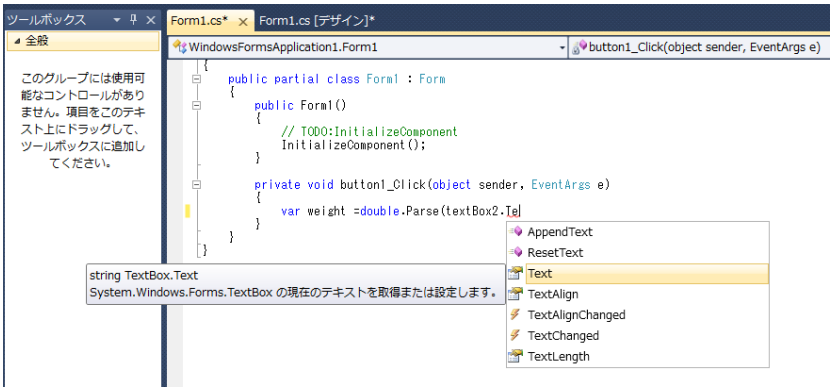


[プロパティ] ウィンドウによるコントロールのイベント ハンドラの追加

IntelliSense による入力候補を活用する

Visual Studio 2010 ではプログラマーのキー入力をできるだけ少なくし、効率をあげるための機能がいくつか用意されています。IntelliSense (インテリセンス) はその 1 つで、コーディング時のキーボード入力を補完してくれる機能です。キー入力を行っていると、その入力内容に応じて入力候補の一覧が表示されます。オブジェクトのメソッドやプロパティ、プロジェクト内で定義しているクラスのフィールド変数などが、入力候補として表示されます。

従来は、前方一致で候補リストが表示されましたが、Visual Studio 2010 からは入力キーワードを含む候補が一覧表示されるようになりました。これによって、より目的のメンバをより早く探し出すことができるようになります。



IntelliSense による入力候補の表示

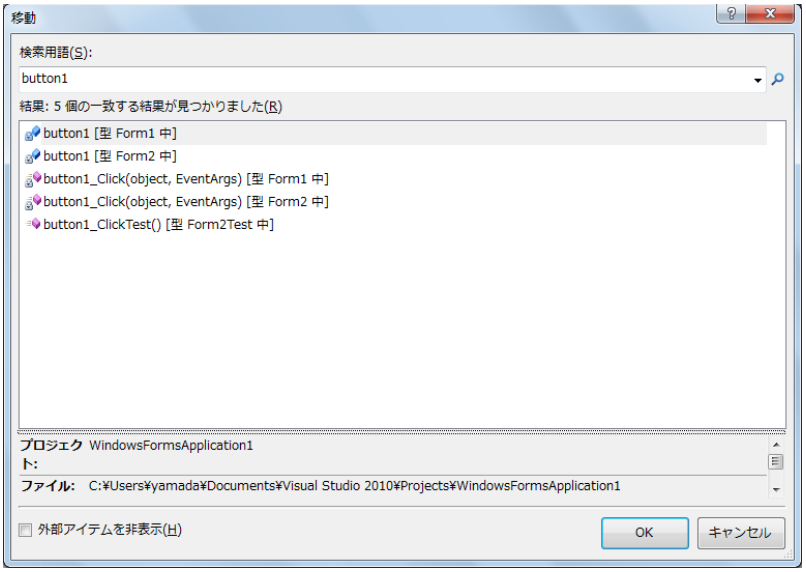
Navigate To: 機能

Visual Studio 2010 には、新しい検索機能として Navigate To:という移動機能が追加されました。シンボル検索と定義への移動を組み合わせた機能です。

メニューバーの **[編集] - [移動]** から移動ウィンドウを開くことができます。 [Ctrl] + [,] キー（ [Ctrl] キーとカンマのキーを同時に）押しても構いません。

検索用語欄に文字列を入力すると同時にインテリセンスが働き、部分文字列に応じて検索結果が表示されます。また、検索用語に省略形を使うことができます。たとえば、キャメルケース (単語の頭文字は大文字) でのコーディング規約の場合で、「AHT」で検索すると「AddHeaderTitle」などがヒットします。

ヒットしたものを一覧からダブルクリックすることで、対応するコードに即座に移動することができます。

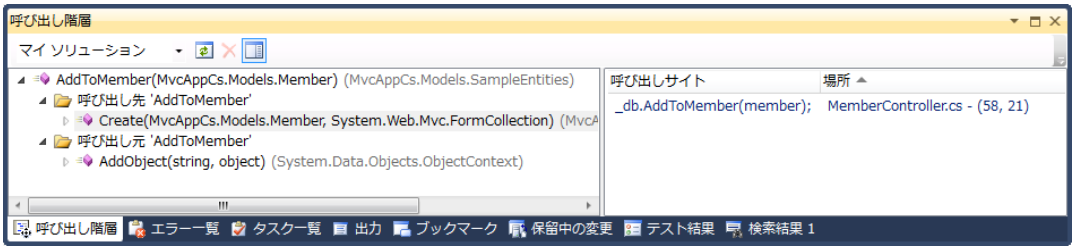


移動ウィンドウ

呼び出し階層

この機能は、オブジェクト指向のコードを検索するのに便利で、呼び出し元と呼び出し先の双方へ移動するのに利用できます。

[呼び出し階層] ウィンドウを表示するには、コード上のメソッドやプロパティを右クリックして表示されるコンテキストメニューから [呼び出し階層の表示] をクリックします。



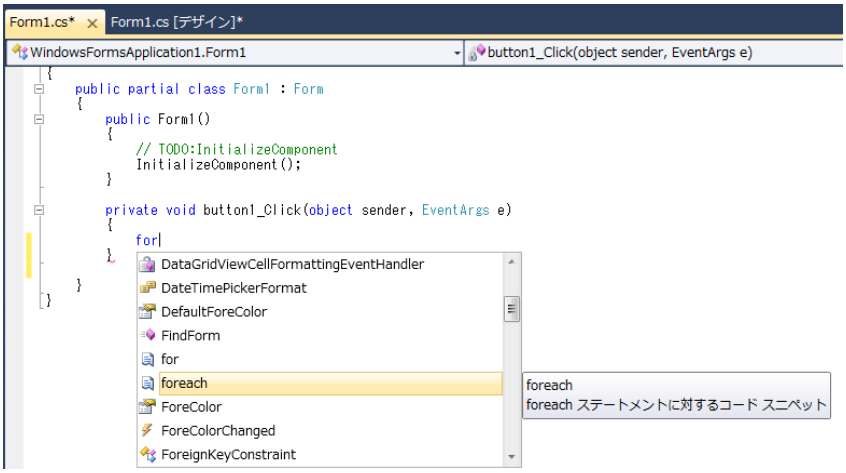
呼び出し階層ウィンドウからメソッドを呼び出しているコードのファイル名や行番号などもわかる

コード スニペットにより定型コードを入力する

さらに、プログラミング言語の構文を簡単に入力したい場合はコード スニペットが利用できます。
たとえば foreach ステートメントを入力する場合、キーボードから "foreach" というコード スニペットのキーワードを入力し Tab キーを 2 回押すと、foreach ステートメントが次のように挿入されます (次のコードは Visual C# の例)。

```
foreach (var item in collection) {
}
```

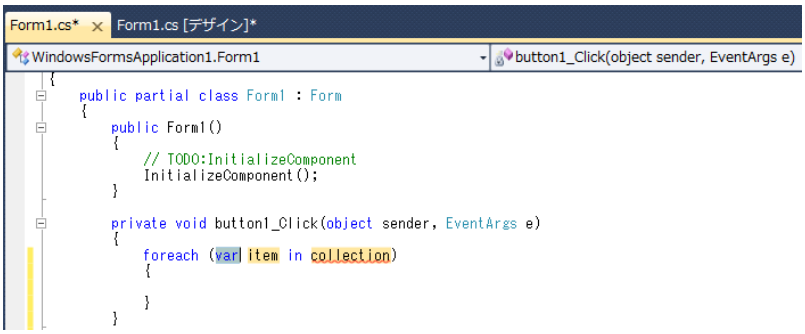
スニペットのキーワードも、IntelliSense の入力候補に表示されます



コード スニペットの挿入

もう 1 つのコード スニペットの入力方法は、コード エディタで右クリックすると表示されるコンテキスト メニューから **[スニペットの挿入]** を選択します。IntelliSense に似た入力候補の一覧が表示されるので、そこから目的のコード スニペットを選択すればそのスニペットのコードが挿入されます。

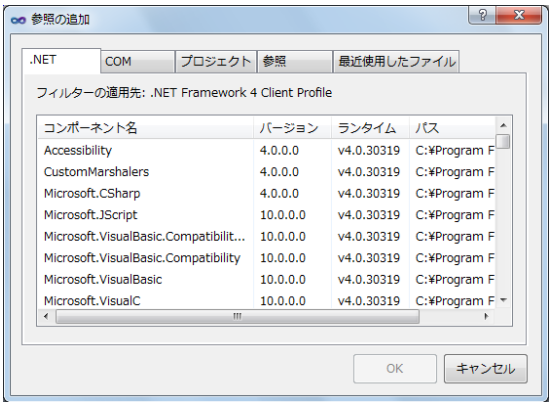
スニペットのコードが入力されると、色付けされた部分が表示されることがありますが、ここは開発者が現在の文脈に合わせて書き換えるカスタム部分になります。たとえば次の画面の例では "Item" や "String" などがカスタム部分です。
このカスタム部分へは Tab キーにより移動できます。
Visual Studio 2010 では、以上のようなユーザー インターフェイスのデザイン機能、コードの入力候補機能や編集機能を活用することで、開発生産性を高めることができます。



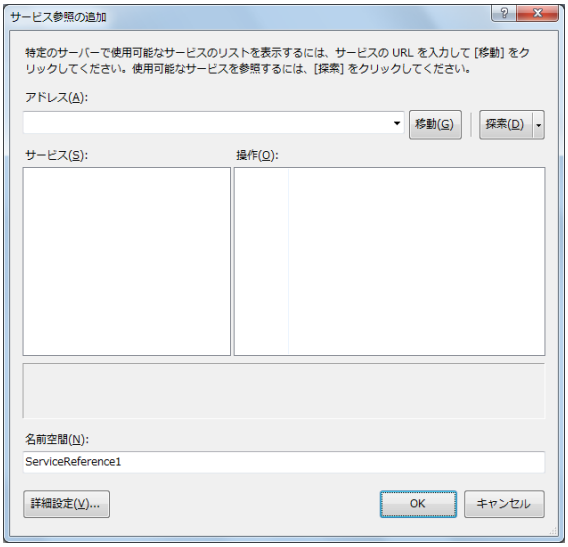
スニペットのコードのカスタム部分の書き換え

DLL サービスへの参照設定

アプリケーションの規模が大きくなると、機能を複数のライブラリ（DLL）に分けることがあります。
場合によっては、ソフトウェア会社が提供する商用パッケージの特別なライブラリを使用することがあるかもしれません。
また、新規にアプリケーションを作成した場合においても、.NET Framework のすべてのライブラリが読み込まれるのではなく、必要最低限のものに限定されています。
このような場合において、これら外部ライブラリの機能をアプリケーションが使用するには参照設定を行う必要があります。
参照設定を行う場合には、**[ソリューション エクスプローラー]** ウィンドウのプロジェクト項目を右クリックし、表示されるコンテキストメニューから **[参照の追加]** を選択します。
.NET Framework のライブラリを追加するには、**[.NET]** タブより該当のアセンブリを追加します。
COM インターフェイスのライブラリを追加するには、**[COM]** タブを、同一プロジェクトに読み込まれているライブラリを参照する場合 [プロジェクト] タブを選択します。
参照するライブラリ ファイルが特定のフォルダに格納されている場合、**[参照]** タブを選択して、該当のファイルを選択します。
また、外部の機能をライブラリではなく、Web サービスなどのサービスとしてアプリケーションで利用する場合、**[ソリューション エクスプローラー]** ウィンドウのプロジェクト項目を右クリックし、表示されるコンテキスト メニューから **[サービス参照の追加]** を選択します(サービス連携の詳細については本書では割愛します)。



参照の追加



では次に、ソース コードが完成した後に行うビルド作業について説明します (「第 5 章 ビルド」)。その説明に入る前に、プロジェクトの詳細設定を行う方法について紹介しておきます。

プロジェクトのプロパティ

プロジェクトの詳細設定は、プロジェクトのプロパティで行います。

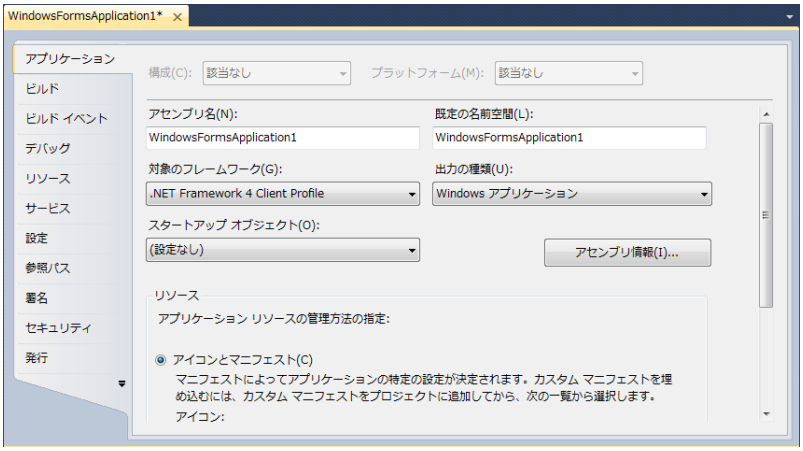
プロジェクトのプロパティは、**[ソリューション エクスプローラー]** ウィンドウのプロジェクト項目を右クリックし、表示されるコンテキスト メニューから [プロパティ] を選択することで表示されます。

[ソリューション エクスプローラー] ウィンドウに、(Visual C# では) 「Properties」もしくは (Visual Basic では)「My Project」が表示されている場合には、これをダブル クリックしてもプロジェクトのプロパティを開けます。

Visual Studio 2010 では、プロジェクトのプロパティはプロジェクト デザイナとして中央のドキュメント ウィンドウに表示されます。

プロジェクトのプロパティでは、プログラミング言語や作成するアプリケーションに応じてさまざまな設定が行えます。

プロパティの内容はカテゴリ分けされており、左側に並んだタブを選択して、それぞれのページで設定を行います。各タブのページの主な内容について説明します。



プロジェクトのプロパティ設定が行えるプロジェクト デザイナ

[アプリケーション] タブ

[アプリケーション] タブでは、「アセンブリ名の指定」、「名前空間」、「アプリケーションのアイコンの指定」 などが行えます。
プロジェクトに複数の Windows フォームが含まれている場合、どのフォームを最初にアクティブにするかといった指定も行えます。
なお、(本書でこれまで何度か登場している) アセンブリとは、.NET Framework が生成するプログラム ファイル (.exe ファイルや .dll ファイル) のことです。

[コンパイル] タブ/[ビルド] タブ

[コンパイル] タブ (Visual Basic の場合) および **[ビルド]** タブ (Visual C# の場合) では、Debug 版 (後述する「デバッグ」の際のモード) と Release 版 (プログラムを正式にリリースする際のモード) といったビルドの構成や、ビルド時のコンパイル オプションなどを指定します。
また、ターゲットとする .NET Framework のバージョンも「2.0」「3.0」「3.5」「4」のいずれかを選択できます (Visual C# の場合は、.NET Framework のバージョンの選択は **[アプリケーション]** タブで行います)。
[コンパイル] タブ/ [**ビルド]** タブについては、「第 5 章 ビルド」の「コンパイラの詳細設定を行う」で詳しく紹介します。
また、Debug 版と Release 版の違いについては、「第 5 章ビルド」の「Debug ビルドと Relase ビルドの違い」でもう少し詳しく説明します。

[ビルド イベント] タブ (Visual C# の場合のみ)

プロジェクトをビルドする前や後に実行する任意の処理をコマンドラインとして指定します。
さらにビルド後の処理では、ビルド結果によって処理を実行するかどうかも指定できます。
たとえば、ビルドが成功したときに生成されたアセンブリをファイル サーバーにコピーしたり、ビルドが失敗したときに担当者にエラー通知を送信したりといった、任意のコマンドが設定できます。
Visual Basic にはこの **[ビルド イベント]** というタブはありませんが、**[コンパイル]** タブに **[ビルド イベント]** というボタンが用意されていますので、そこからビルド時に実行するコマンドを指定できます。

[デバッグ] タブ

[デバッグ] タブでよく利用する項目は、デバッグするアプリケーションでの起動パラメーターの指定です。
[デバッグ] タブについては、「第 6 章 デバッグ」の「デバッグの設定」で詳しく紹介します。

[リソース] タブ

[リソース] タブでは、アプリケーションで使用する文字列や画像、アイコンなどのリソースを管理します。

[サービス] タブ

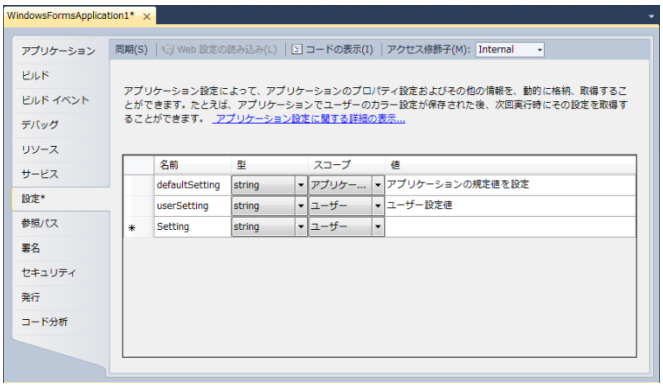
[サービス] タブでは、アプリケーションで「クライアント アプリケーション サービス」を使用するための設定を行います。
クライアント アプリケーション サービスは、Windows フォーム アプリケーション WPF アプリケーションから ASP.NET が提供する認証やプロファイリング サービスを使うためのものです。

[設定] タブ

[設定] タブは、アプリケーション設定（app.config）の内容を編集するためのものです。

ここで設定した項目はテキスト形式の設定ファイルに保存され、たとえばアプリケーションの完成後であっても環境に応じて設定を変更することが可能です。

設定した内容は Settings オブジェクト(Visual C# の場合) や MySettings オブジェクト（Visual Basic の場合）などを使ってアプリケーションから簡単に参照できます。

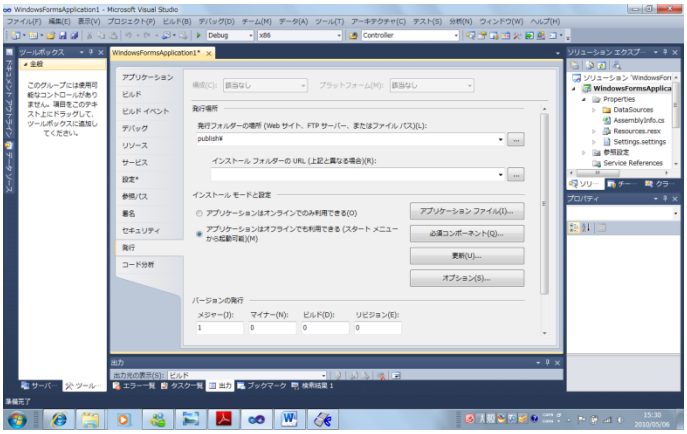


[設定] タブのページ

[署名] タブ、[セキュリティ] タブ、[発行] タブ

[署名] ページではプロジェクトで作成するアセンブリに署名を設定します。署名を行うとファイルの改ざんなどが防止できるため、セキュリティが向上します。また、**[発行]** ページでは ClickOnce によるプログラムの配置が行えます。

[セキュリティ] ページでは ClickOnce で配置するアプリケーションでどのような機能を許可するかを定めるコード アクセス セキュリティの内容を設定できます。



[発行] タブのページ

第 5 章 ビルド

ユーザー インターフェイスの作成とコーディングが完了したら、ソリューションのビルドを行います。
ビルドとは、ソースファイルをコンパイルして実行形式のアセンブリを作成する作業のことです。
Windows アプリケーションの場合は、何も設定を変更していなければ、<プロジェクト名> + ".exe" という名称の実行形式のファイル (.exe ファイル) が生成されます。
これにより、作成したアプリケーションが実行できるようになります。

Visual Studio 2010 のビルド機能

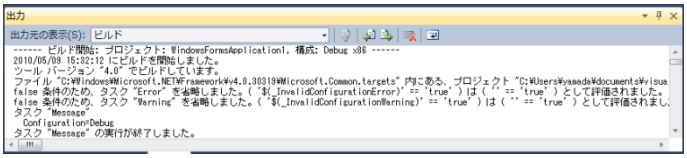
Visual Studio 2010 にはビルドを行う操作として、「ソリューションのビルド」、「ソリューションのリビルド」、および「ソリューションのクリーン」という 3 つのビルド操作が用意されています。
「ソリューションのビルド」は、前回にビルドした内容と比較して更新のあったファイルとプロジェクトのみを (差分) ビルドしますので、コンパイルの実行時間が後述する「ソリューションのリビルド」に比べて短くなります。
「ソリューションのリビルド」は、ソリューションに含まれるすべてのプロジェクトをコンパイルします。
「ソリューションのクリーン」はビルドの際に生成されたファイルを削除します。
したがって「ソリューションのクリーン」を実行した後、「ソリューションのビルド」を行った場合は、すべてのプロジェクトが再コンパイルされます。

ビルド操作	説明
ソリューションのビルド	変更されたファイルとプロジェクトのみをビルドする
ソリューションのリビルド	すべてのプロジェクトを再ビルドする
ソリューションのクリーン	ビルドに関連する一時ファイルを削除する

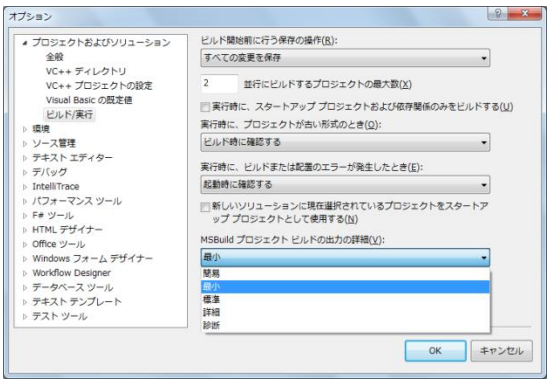
表: Visual Studio 2010 に用意されているビルド操作

ビルドの状況を確認する

ビルドの様子は **[出力]** ウィンドウに表示されます。ビルドの際に表示されるメッセージは、**[オプション]** ダイアログ ボックスの設定により変更できます。
メニューバーの **[ツール] - [オプション]** から [オプション] ダイアログ ボックスを開き、左側のツリーから **[プロジェクトおよびソリューション] - [ビルド/ 実行]** を選択し、右側から **[MSBuild プロジェクト ビルドの出力の詳細]** のプルダウンリストから「簡易」、「最小」、「標準」、「詳細」、「診断」などを選択することで **[出力]** ウィンドウに表示される内容を変更できます。
また、**[オプション]** ダイアログ ボックスで、左側から **[プロジェクトおよびソリューション] - [全般]** を選択して、右側から **[ビルド開始時に出力ウィンドウを表示]** チェック ボックスをオンにしておくと、ビルド時に **[出力]** ウィンドウが表示されて、ビルドの内容をリアルタイムに確認できるようになります。
さらに、**[ビルド完了時にエラー一覧を表示]** チェック ボックスをオンにしておくことで、ビルドが終わったときに **[エラー一覧]** ウィンドウが表示されるようになり、エラーの内容をすぐに確認できるようになります。



[出力] ウィンドウに表示されたビルドの様子



[オプション] ダイアログ ボックスにおけるビルド出力の詳細設定

ビルド エラーが表示されたら

ビルドの結果にエラーや警告が含まれる場合は、「第 3 章 プロジェクトの作成」で説明した **[エラー一覧]** ウィンドウで確認します。エラーを示すメッセージの行をダブル クリックすると、エラーの発生したソース ファイルが開かれ、該当する行にテキスト カーソル (キャレット) が移動します。

[エラー一覧] ウィンドウ

特定の 1 つのプロジェクトのみをビルドする

ソリューションの中に複数のプロジェクトが含まれている場合は、毎回ソリューション全体をビルドするよりも、修正を行っているプロジェクトのみをビルドする方が効率的です。

特定のプロジェクトのみをビルドするには、**[ソリューション エクスプローラー]** のプロジェクト項目を右クリックして、そこで表示されるコンテキスト メニューから **[ビルド]** を選択します。

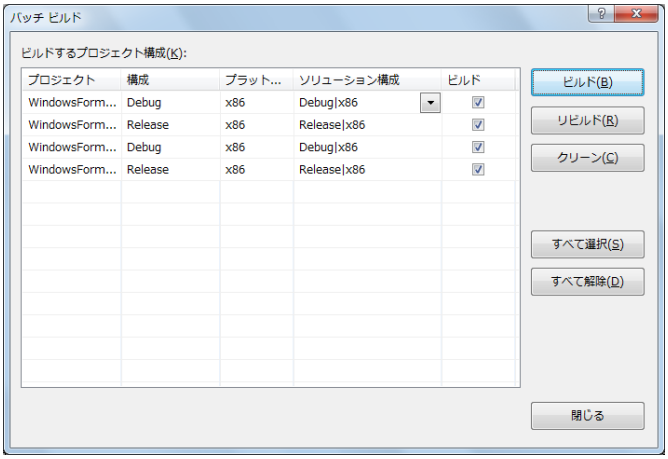
もしくは **[ソリューション エクスプローラー]** のプロジェクト項目を選択した状態にして、メニューバーの **[ビルド]** メニューを開くと、**[<プロジェクト名> のビルド]** と記述されたメニューが表示されますので、ここからビルドもしくはリビルドを選択することで、そのプロジェクトのみをビルドできます。

特定の複数のプロジェクトのみをビルドする

ソリューション内に含まれるすべてのプロジェクトの中から一部のものを選んでビルドするには、バッチ ビルドを実行します。

これには、メニューバーから **[ビルド] - [バッチ ビルド]** を選択して、**[バッチ ビルド]** ダイアログ ボックスを開きます。

[バッチ ビルド] ダイアログ ボックスで、**[ビルドするプロジェクト構成]** 一覧からビルドしたい構成を選択して **[ビルド]** もしくは **[リビルド]** ボタンをクリックします。これにより、選択されたプロジェクト構成がすべてビルドされます。



[バッチ ビルド] ダイアログ ボックス

ビルドの詳細設定

Debug ビルドと Release ビルドの違い

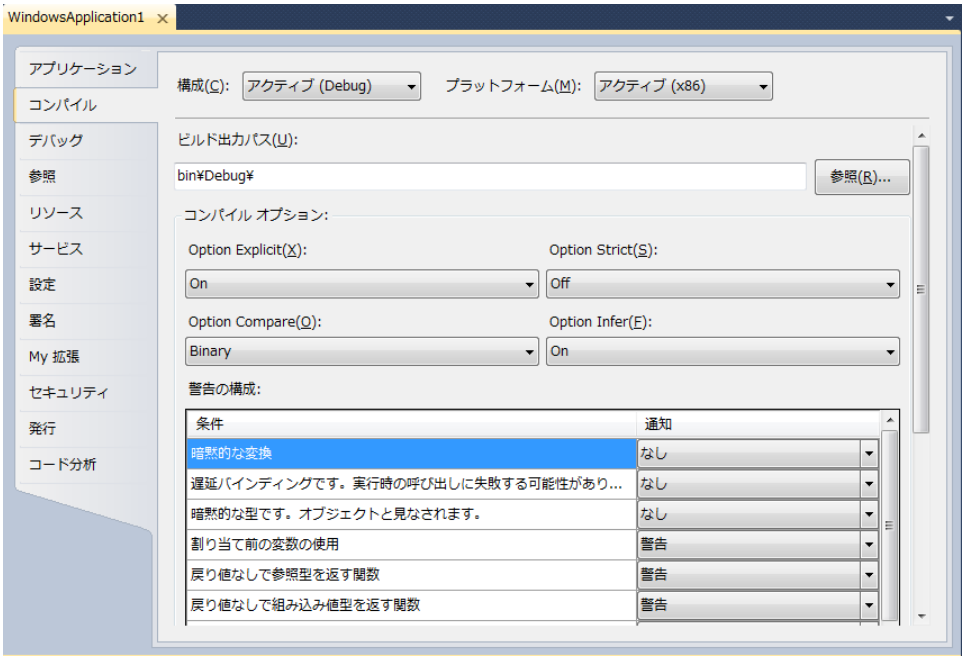
プロジェクトには、「Debug (デバッグ)」と「Release (リリース)」の 2 種類のビルド方法が用意されています。
どちらも実行形式のファイル (.exe ファイル) を生成しますが、名前のとおり「Debug」はデバッグを行うためのビルドです。
作成された実行形式のファイルには、デバッグの際に参照するソースコードの情報や、変数などのシンボル情報がコンパイルと同時に生成されます。
一方「Release」は、デバッグが完了して、できあがったアプリケーションをエンド ユーザーに配布する際に利用するビルドです。
「Release」ビルドでは、シンボル情報は作成されず、さまざまな最適化が行われるので、出力されるアプリケーションは「Debug」ビルドよりも高速に動作します。

コンパイラの詳細設定を行う

実際にどのような設定でビルドが行われているかは、前の「第 4 章 アプリケーションの開発」の「プロジェクトのプロパティ」で紹介したプロジェクト デザイナで確認できます。
ただし、Visual Basic と Visual C# で若干操作が異なります。

Visual Basic の場合は、**[コンパイル]** タブを選択します。
[コンパイル] タブのページでは、プロジェクトに含まれるソース ファイルのコンパイル方法を変更できます。
実際に変更するには、まず **[構成]** ドロップダウン リストから「Debug」と「Release」のどちらのソリューション構成を変更するか選択し、次に **[プラットフォーム]** ドロップダウン リストで、どの CPU タイプ上で実行するかを指定します。
32 bit CPU を表す「x86」と 64 bit CPU を表す「x64」を選択することもできますが、通常は両方の CPU タイプをサポートする「Any CPU」を選択します。
そして、**[ビルド出力パス]** などさまざまなコンパイルの設定を変更していきます。
[ビルド出力パス] では、プログラムが生成される場所を指定します。

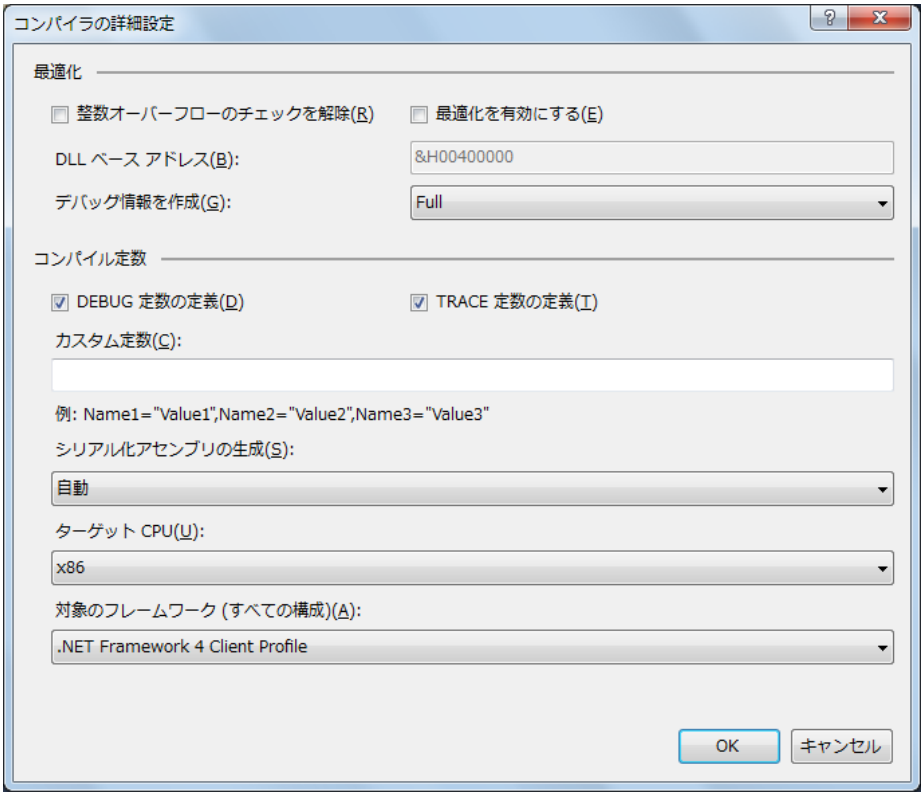
デフォルトの設定では、Debug 版はプロジェクトが保存されているフォルダの下にある bin¥Debug フォルダ、Release 版は bin¥Release フォルダが指定されています。



[コンパイル] ページ (Visual Basic) におけるコンパイラの詳細設定

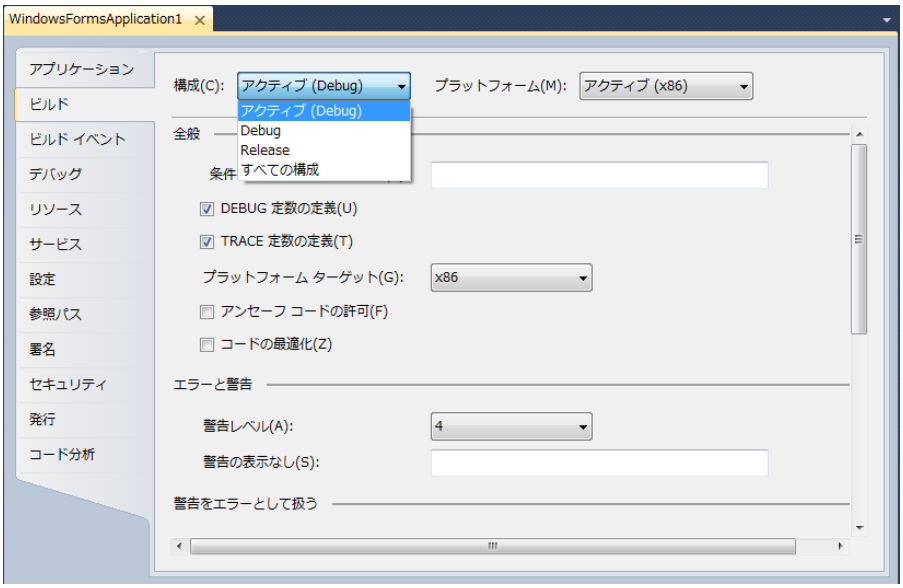
このページの一番下には **[詳細コンパイルオプション]** ボタンがあり、これをクリックすると **[コンパイラの詳細設定]** ダイアログ ボックスが表示されます。

[コンパイラの詳細設定] ダイアログ ボックスでは、デバッグ情報の生成、.NET Framework のどのバージョンをターゲットにするかなど、より詳細なコンパイルの設定が指定できます。



[コンパイラの詳細設定] ダイアログ ボックス

Visual C# の場合は、プロジェクト デザイナの **[ビルド]** タブを選択して、**[ビルド]** ページを開きます。
Visual Basic と同様に、最初に **[構成]** ドロップダウンリストの「Debug」か「Release」を選択して、次に **[プラットフォーム]** ドロップダウン リストのいずれかを選択してから、ビルドの各オプションを変更します。
なお、.NET Framework のバージョンの指定は、Visual Basic の場合と異なり、**[ビルド]** ページではなく、**[アプリケーション]** ページで行います。

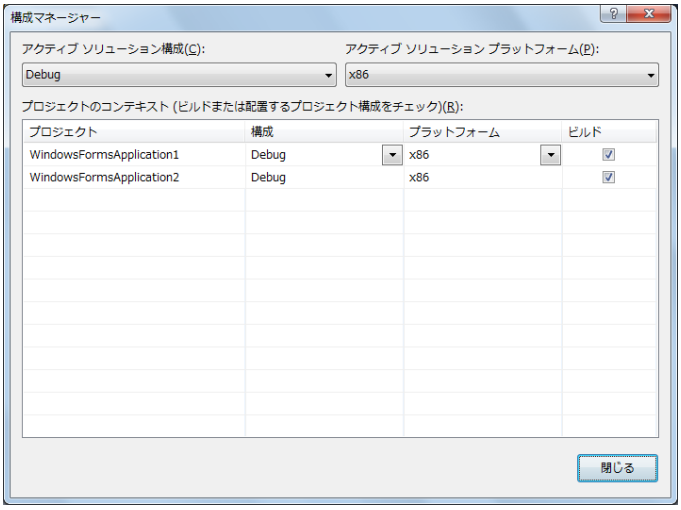


プロジェクト デザイナのビルド ページ (Visual C#)

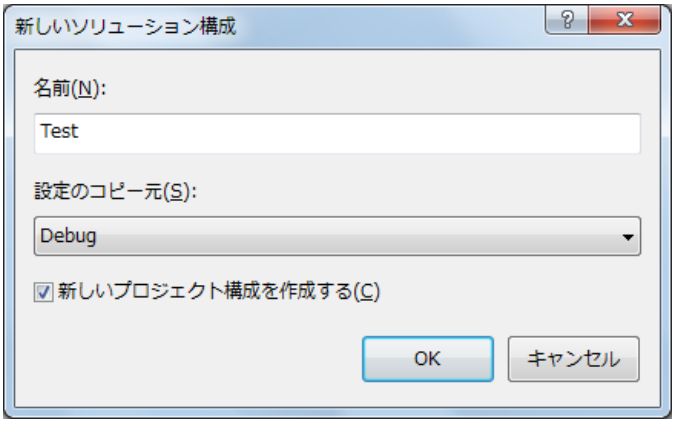
ソリューション構成を追加する

必要なら「Debug」や「Release」のほかに独自のソリューション構成を作成することも可能です。
これには、メニュー バーから **[ビルド] - [構成マネージャ]** を選択して **[構成マネージャ]** ダイアログ ボックスを開き、**[アクティブ ソリューション構成]** ドロップダウン リストから **[<新規作成...>]** を選択します。
これにより、**[新しいソリューション構成]** ダイアログ ボックスが表示されます。

[新しいソリューション構成] ダイアログ ボックスで **[名前]** を入力して **[OK]** ボタンをクリックすると、独自のソリューション構成が作成されます。
[設定のコピー元] ドロップダウン リストで既存のソリューション構成を選択すると、その構成の情報のすべてがコピーされるので便利です。



[構成マネージャ] ダイアログ ボックス



[新しいソリューション構成] ダイアログ ボックス

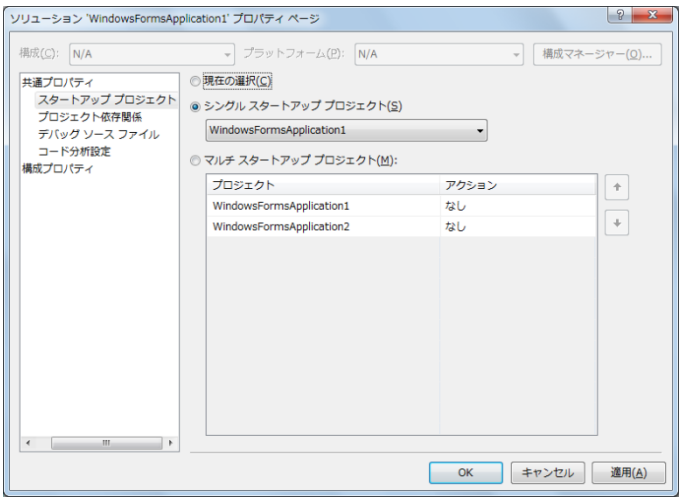
第 6 章 デバッグ

プロジェクトのビルドが成功したからといって、アプリケーションが思ったとおりに正しく動作するとは限りません。
ビルド (コンパイル) によって文法エラーや、宣言を行っていない変数の使用といった静的なエラーは取り除けますが、それだけでは不十分なのです。そこで次に行うのがデバッグです。

スタートアップ プロジェクトの設定

ソリューションに複数のプロジェクトが含まれている場合は、デバッグを始める前にどのプロジェクトをスタートアップ プロジェクトにするかを指定します。
これはデバッグ実行時にどの実行可能ファイル (.exe ファイル) を最初に起動するかを指定するもので、ソリューションに複数のアプリケーション プロジェクト (実行可能なファイルを生成するプロジェクト) が含まれている場合に必要になります。

スタートアップ プロジェクトの指定は、ソリューションのプロパティ ページで行います。
[ソリューション エクスプローラー] ウィンドウ内のソリューション項目を右クリックして、コンテキスト メニューから **[プロパティ]** を選択します。すると **[ソリューション '<ソリューション名>' プロパティ ページ]** ダイアログボックスが表示されます。
通常は **[シングル スタートアップ プロジェクト]** ラジオボタンを選択し、ドロップダウン リストから最初に起動するアプリケーションのプロジェクトを指定します。
ちなみに、**[マルチ スタートアップ プロジェクト]** ラジオボタンを選択すると、複数のプロジェクトを最初に起動するようにできます。



ソリューションのプロパティ ページにおけるスタートアップ プロジェクトの指定

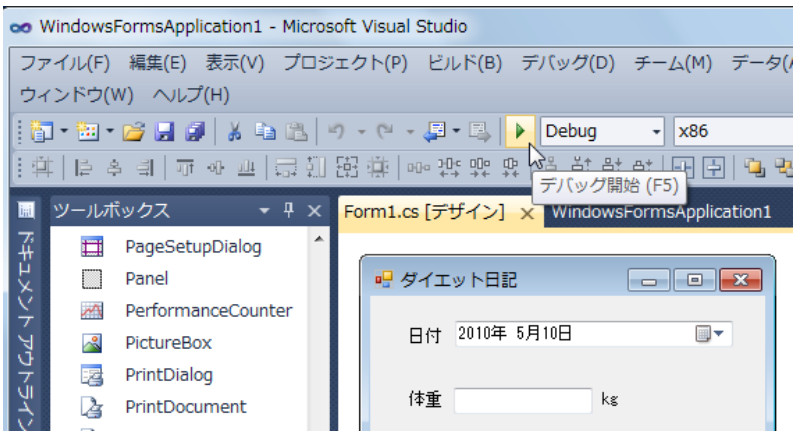
スタートアップ プロジェクトを指定するもう 1 つの方法は、**[ソリューション エクスプローラー]** ウィンドウのプロジェクト項目を右クリックして、コンテキスト メニューから **[スタートアップ プロジェクトに設定]** を選択することです。
これを行うと、ソリューションのプロパティ ページの **[シングル スタートアップ プロジェクト]** ラジオボタンを選択したのと同じ結果になります。

デバッグの実行

以上でデバッグの準備は整ったので、デバッグを実行する方法の説明に入りましょう。

デバッグを開始する

デバッグを開始するには、まずソリューションを Debug 構成でビルドします。そして、メニュー バーから **[デバッグ]** - **[デバッグ開始]** を選択します（ツールバー のデバッグ開始ボタンをクリックしても構いません）。これにより、先ほどスタートアップ プロジェクトに設定したアプリケーションが起動します。



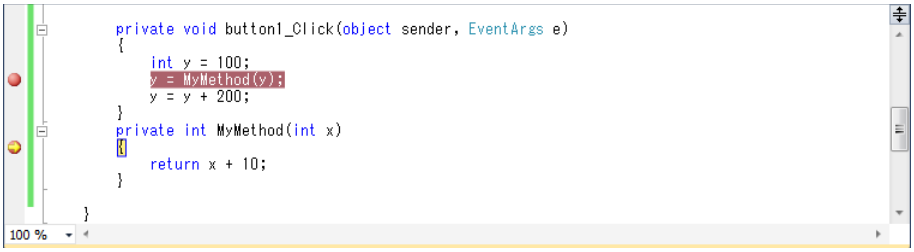
デバッグ実行の開始

ステップ実行する

デバッグの基本的な操作は、実行処理の流れを追うことです。

プログラムの実行を追いながら、設計したとおりに処理が進んでいることを確認します。

そのための基本的なデバッグ操作は、ソース コード上を 1 行ずつステップ実行することです。



ステップ実行には、「ステップ オーバー」、「ステップ イン」、「ステップ アウト」という 3 つの操作があります。

ステップ オーバーはソース コードを 1 行ずつ実行する最も基本的なステップ実行です。

ステップ インは (現在実行中の行にある)メソッドの中に入るステップ実行で、ステップ アウトはメソッドの中から外に出るステップ実行です。

メソッド内部まで詳しく処理の流れを追いたい場合にステップ インを行います。

なお、メソッド以外の場所でステップ インを行っても、ステップ オーバーと同じように 1 行ずつ実行するだけです。これらのデバッグ作業は、メニュー バーの **[デバッグ]** の配下のメニューからも実行できますが、キーによって操作する方が便利です。

Visual Basic の標準のキー バインディングでは、

ステップ オーバー	Shift + F8 キー
ステップ イン	F8 キー
ステップ アウト	Ctrl + Shift + F8 キー

に割り当てられています。

Visual C# では、

ステップ オーバー	F10 キー
ステップ イン	F11 キー
ステップ アウト	Shift + F11 キー

に割り当てられています。

ブレークポイントを設定する

ソース コード上を 1 行ずつステップ実行しながら何万行にも及ぶプログラム全体の動きを把握するには、作業量や作業時間の面で無理があります。

このような場合には、本当にチェックが必要なコード部分のみをステップ実行して、そのほかのコードではステップ実行しません。

これを行うには、そのコード箇所にブレークポイントを設定します。

これにより、プログラムの実行がブレークポイントの場所に達したときに中断されます。

実行が停止した場所から前述のステップ実行が可能になります。

ステップ実行中は、実行している箇所に関連する変数の内容やオブジェクトの状態などを、**[ローカル]** ウィンドウや **[ウォッチ]** ウィンドウで確認できます (詳細は後述します)。

実際にブレークポイントを設定するには、コード エディタの左端の余白部分 (薄いグレー表示の部分) をクリックします。

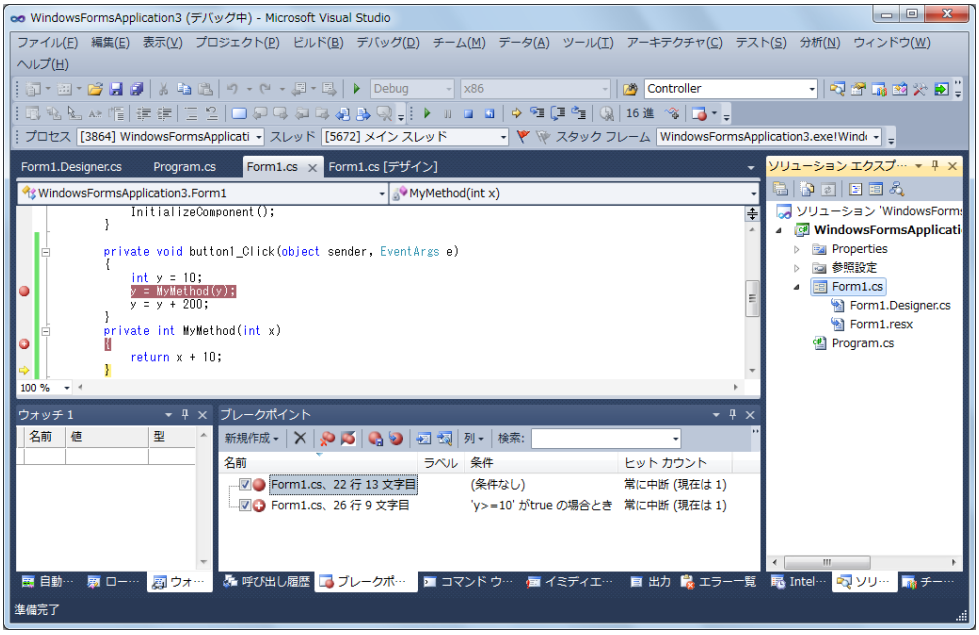
ブレークポイントが設定されると、そこに赤い丸のマークが表示されます。

もう一度同じ場所をクリックすると、ブレークポイントがクリアされます。

ブレークポイントには、実行を中断する条件を付加することもできます。

たとえば、ローカル変数やパラメーターを参照して、指定しておいた条件式に一致する (true の) 場合や、特定の変数が変更された場合、また、設定したブレークポイントを通過した回数が指定した回数に達した場合などの条件で、実行が中断するように設定することが可能です。

このような条件を設定するには、ブレークポイントを示す赤い丸のマークを右クリックし、表示されるコンテキストメニューから **[条件]** を選択して、そこで表示される **[ブレークポイントの条件]** ダイアログ ボックスを使います。

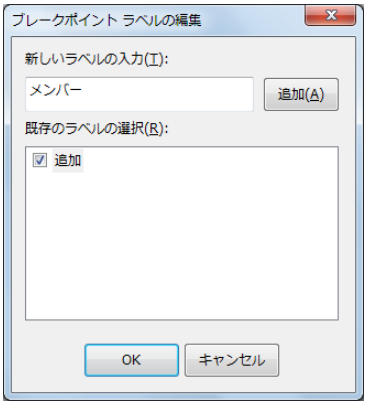


ブレークポイントの設定

なお、ブレークポイントを設定する場所の候補としては、たとえばユーザー インターフェイスのデバッグが中心となる Windows フォーム アプリケーションでは、ボタンが押されたり、リストボックスが選択されたりしたときに発生するイベントを処理するイベント ハンドラの先頭などが挙げられます。

ブレークポイントのラベル付け

Visual Studio 2010 では、ブレークポイントにラベルを付ける機能が利用できます。このラベルを使用して、[ブレークポイント] ウィンドウの検索機能から指定したラベルを持つブレークポイントを検索することもできます。ブレークポイントには複数のラベルを付けることができます。任意のブレーク ポイントで右クリックし、コンテキストメニューから [ラベルの編集] を選択します。
デバッグ作業では、ローカル変数レベルの詳細な情報を必要とする場合もあれば、呼び出しの遷移のみを確認したい場合もあり、その目的によって確認したい粒度や角度を変えるのが一般的です。ラベルごとに一括して有効化／無効化／削除をすることができ、目的に応じてデバッグの設定を容易に切り替えることができます。さらにこのラベル付はインポート／エクスポートすることができるため、代表して一人の開発者が設定したら、チームメンバー全員で共有することができます。

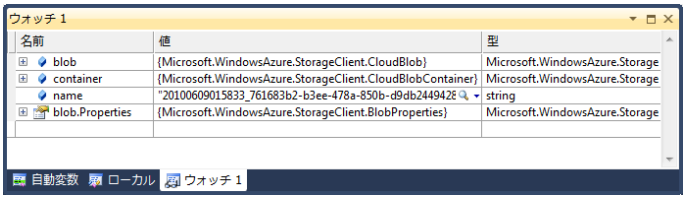


ブレークポイントのラベル付け

ウォッチ式で変数内容を確かめる

ブレークポイントにより実行が中断したら、前述したように **[ローカル]** ウィンドウや **[ウォッチ]** ウィンドウで、その時点での変数 (オブジェクト) の内容などを確認できます。
[ローカル] ウィンドウでは、停止時点で参照可能なローカル変数が自動的に表示されます。
実行途中で「現在の変数やオブジェクトの状態を確認したい」というときには、**[ローカル]** ウィンドウが便利です。
[ローカル] ウィンドウに表示された変数やオブジェクトの値を書き換えて、処理を続行することも可能です。

また **[ウォッチ]** ウィンドウも **[ローカル]** ウィンドウと同じ機能を持ちますが、監視したい変数やオブジェクトを任意で指定できるという違いがあります (変数やオブジェクトは自動的に表示されません)。
確認したい変数やオブジェクトがあらかじめ決まっている場合には、**[ウォッチ]** ウィンドウの方が便利です。
さらに **[ウォッチ]** ウィンドウでは、式を指定できます。
たとえば、**[名前]** に「1 + 1」という式を指定した場合には、**[値]** に「2」が表示されます。



ウォッチ式の設定

このようなさまざまなデバッグ機能を駆使することで、Visual Studio 2010 では非常に効率よくデバッグが行えます。

デバッグの詳細設定

デバッグ関連の設定は、プロジェクト デザイナの **[デバッグ]** タブのページで行います。

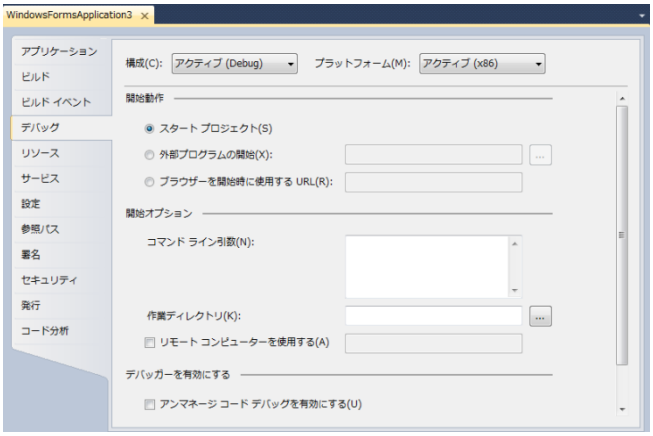
[デバッグ] タブのページではたとえば、デバッグを開始する際にどのプログラムを表示するのかを指定できます。

通常はスタート プロジェクトを起動しますが、そうではなく、外部のプログラムを最初に起動して、その外部のプログラムがスタート プロジェクトのプログラムを呼び出す、という変則的なデバッグを行うこともできます。

この場合には、**[外部プログラムの開始]** ラジオボタンを選択します。

また、アプリケーションを起動する際に **[コマンドライン引数]** を渡すこともできます。

この機能は、コマンドライン引数を使用するプログラムをデバッグする際に便利です。



プロジェクト デザイナの [デバッグ] タブのページ

IntelliTrace

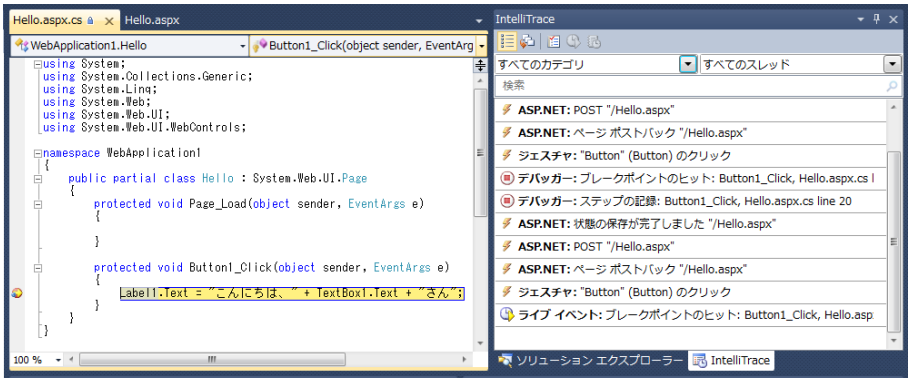
IntelliTrace 機能は、Visual Studio 2010 Ultimate のみで使える機能で、デバッグ実行時の履歴を記録する機能です。

イベントの発生時やブレークポイントで処理を停止した時点での変数の状態や呼び出し履歴などを記録します。

これまでは、既に通り過ぎたコードの状態を確認する場合には、もう一度デバッグをやり直す必要がありました。これは積み重なると多大な時間を消費するとともに、データベースやシステムのデータ変更など、容易に繰り返し実行できない場合もあります。IntelliTrace はデバッグの履歴を保持することにより、デバッグの効率化に貢献します。

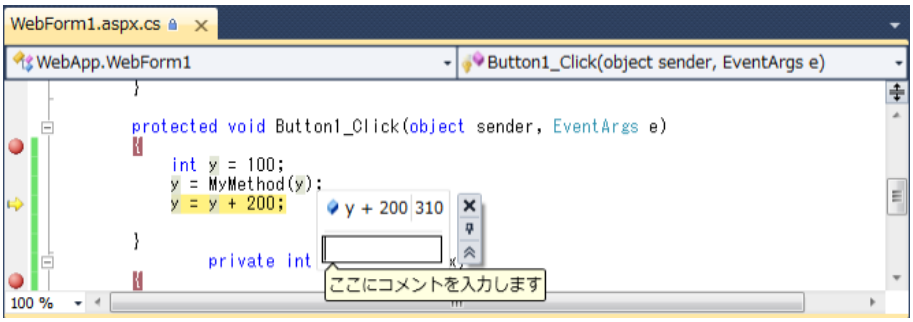
さらに、この履歴は、保存して他のエンジニアに渡すことができます。テスト担当者と修正担当者が異なる場合においては、これまで伝えることができなかった具体的な行ごとの遷移と実行結果をデバッグ情報として渡すことができ、”再現しないバグ” の発生を軽減することができます。

記録されたイベントについて、[IntelliTrace] ウィンドウのイベント名を選択すると、過去のコードの場所や変数の状態などを確認することができ、開発時に便利です。



ToolTips のピン止め

Visual Studio 2010 では、デバッグ中に変数の値をを表示する ToolTips をピン止めして、常に表示させておくことができます。さらに、ピン止めした ToolTips に対して、コメントを入力することもできます。



第 7 章 単体テスト

ここまでに、プロジェクトの新規作成、ユーザー インターフェイス デザイン、処理ロジックのコーディング、ビルド、デバッグと、一連の開発作業を紹介してきました。
ただ単にアプリケーションを開発するというだけであれば、これらの作業手順を踏むだけで十分でしょう。
しかし、品質の高いアプリケーションを開発するには、テストは欠かせない作業の 1 つです。

ただし、テストといっても、メソッド単位やクラス単位で行う「単体テスト」、複数のクラス モジュールを組み合わせた状態で行う「結合テスト」、本番に近い状態でシステム全体をテストする「システム テスト」、Web サイトに対する集中的なアクセス負荷を擬似的に発生させてその耐性を検証する「負荷テスト」、期待する機能や性能を備えているかを利用者になる予定のエンド ユーザーが検証する「受け入れテスト」など、さまざまな種類があります。
とりわけ、これらのテストのうち、最も小さな単位である単体テストは重要です。
単体テストが成功しなければ、小さな単位が組み合わさったより大きな機能単位を検証する結合テストやシステム テストが成功することは理論的にあり得ないからです (仮に成功したとしても、隠れた不具合として後々問題になる可能性が高いでしょう)。
Visual Studio 2010 では、この単体テストをプロジェクトとして作成し、実行できる機能が、(Professional 以上に) 搭載されています。
以下では、この単体テストの機能について紹介します。

単体テストの作成

まずは単体テストを作成します。
通常、1 つのクラスに対して、1 つの単体テスト クラスを作成します。したがって、複数のクラスが集まった 1 つのプロジェクトに対しては、複数の単体テスト クラスを集めた 1 つの単体テスト プロジェクトを作成することになります。
単体テスト対象となるプロジェクトと単体テスト プロジェクトは別々のソリューションに含めることも可能ですが、1 つのソリューションに含めると、(単体テスト対象となるプロジェクトに含まれる) 既存のクラスのコードからそれに対応する単体テスト クラスのコードを自動生成できるので便利です。なお、単体テスト クラスの自動生成の際には、単体テスト プロジェクトも作成されます。

単体テストを作成する

実際に、既存のコードから単体テストを作成するには、単体テストを作成したクラスのコードを (ドキュメント ウィンドウの) コード エディタで表示し、そのクラス名の部分を右クリックします。
コンテキスト メニューが表示されるので、**[単体テストの作成]** を選択します。
これにより、**[単体テストの作成]** ダイアログ ボックスが表示されますので、単体テストを作成したいクラスやそのメソッド群が正しくチェックされているかを確認します。
ここで、単体テストに含めたくないメソッドなどがあればチェックは外し、逆にほかのクラスの単体テストも作成したければそのクラスにチェックを入れます。
ダイアログ ボックスの下部にある **[出力プロジェクト]** コンボボックスの「Visual C# テスト プロジェクトの新規作成」、「Visual Basic テスト プロジェクトの新規作成」などの中から適切なものを選択します。
なお、ソリューション内に既存のテストプロジェクトが存在する場合は、それをこのコンボボックスで選択することもできます。
最後に **[OK]** ボタンをクリックします。

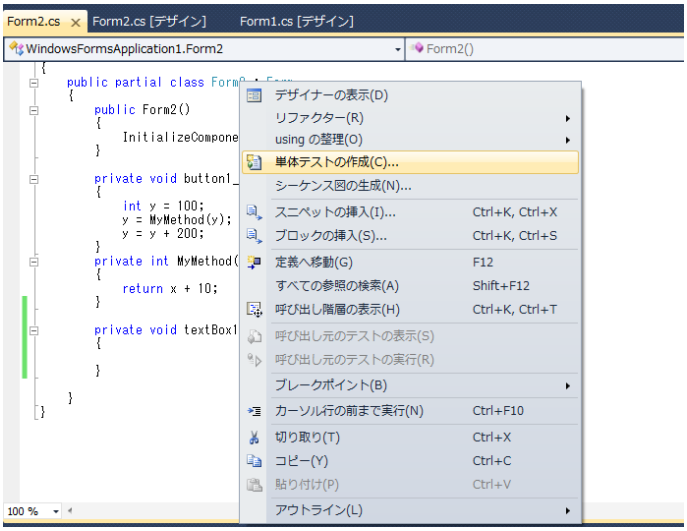
[新しいテスト プロジェクト] ダイアログボックスが表示されるので、名前を入力して **[作成] ボタンをクリックします。**以上の作業で、単体テスト プロジェクトが作成され、その中にチェックを入れたクラスに対応する単体テスト クラスのファイルが生成されます。

単体テスト クラスの名前は、基本的に、(単体テスト対象となるプロジェクトに含まれる) 生成元クラスの名前に続けて「Test」をたした形式になります。

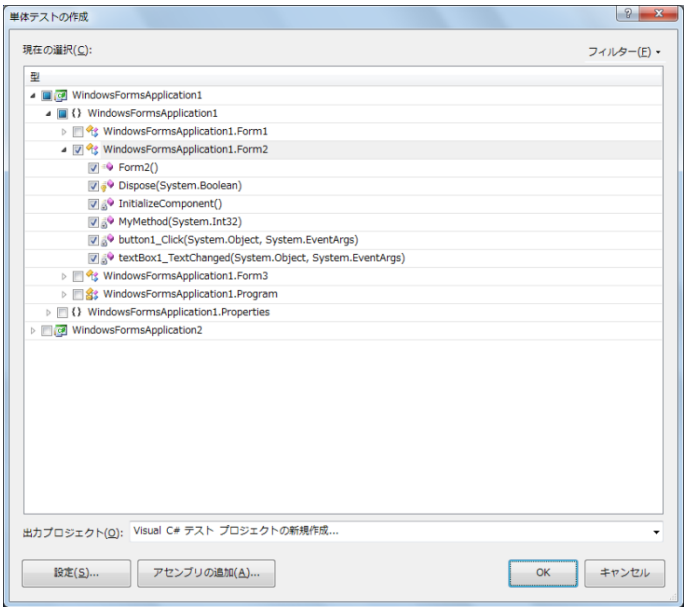
たとえば、「Form1」クラスの単体テストを生成した場合には、その単体テスト クラスの名前は「Form1Test」です。

単体テスト クラス名はファイル名としても使われるので、たとえば先ほどの例の単体テスト クラスが記述されているファイルの名前は、Visual C# のプロジェクトの場合は「Form1Test.cs」となります。

単体テスト クラスの内部に自動生成されたテスト メソッドの名前も、クラス名と同様に、生成元のクラスのメソッド名に「Test」をたした形式になります。



コード エディタにおける [単体テストの作成] メニューの選択



[単体テストの作成] ダイアログ ボックスによる単体テストの作成

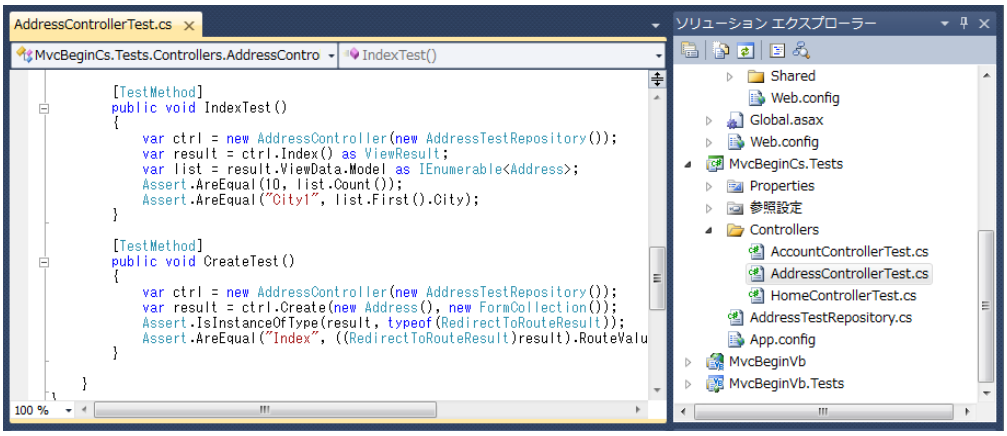
単体テストの内容を実装する

次に、単体テストの内容を実装します。

具体的には、単体テスト クラスのテストメソッドの内部に、対象のメソッドを検証するコードを記述します。

たとえば、対象のメソッドを呼び出して、その戻り値が期待する値と一致するかどうかを判断し、一致しない場合にはエラーであることを通知する Assert.AreEqual メソッドを呼び出すといったコードです。

このような検証用のコードを記述していくことで、対象のクラスと対象のメソッドが正しく動作するかどうかを確かめられるのです。



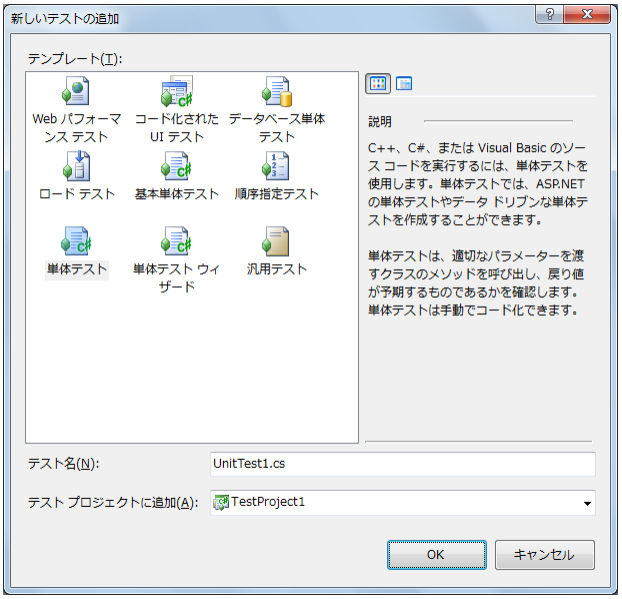
単体テスト コードの記述

手動による単体テストの作成について

なお、単体テスト プロジェクトや単体テスト ファイルは手動で作成することもできます。単体テスト プロジェクトを作成するには、通常のプロジェクトの作成と同じで、**[新しいプロジェクト]** ダイアログ ボックスを使用します。

また、単体テスト ファイルを作成するには、メニュー バーの **[テスト]** - **[新しいテスト]** を選択します。

これにより **[新しいテストの追加]** ダイアログ ボックスが表示されますので、ここで「単体テスト」テンプレートを選択し、任意の **[テスト名]** を入力し、**[テスト プロジェクトに追加]** コンボボックスで既存のプロジェクト名を選択して、**[OK]** ボタンを押すだけです。



単体テスト ファイルの作成

単体テストの実行

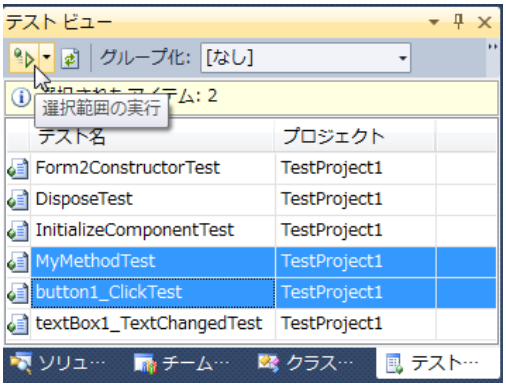
単体テストのコーディングが完了したら、その単体テストを実行します。

すべての単体テストを実行する

最も簡単に単体テストを実行するには、メニュー バーから **[テスト]** - **[実行]** - **[ソリューションのすべてのテスト]** を選択することです。
これにより、ソリューション内にある単体テスト プロジェクトのすべての単体テストが実行されます。

[テスト ビュー] ウィンドウで実行する

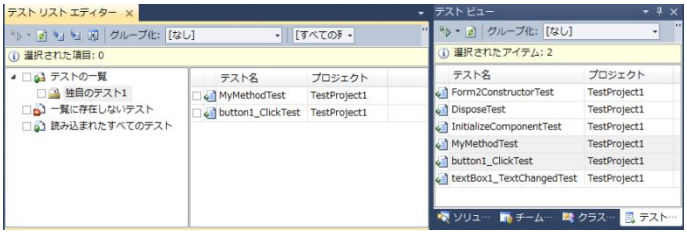
しかし、現実的には現在開発している部分の単体テストだけを選択的に実行したいというケースがほとんどではないでしょうか。
このようなケースで手軽に単体テストを実行するには、**[テスト ビュー]** ウィンドウを使います。
[テスト ビュー] ウィンドウは、メニュー バーの **[テスト]**- **[ウィンドウ]** - **[テスト ビュー]** を実行すると表示されます。
[テスト ビュー] ウィンドウには、ソリューション内にあるすべての単体テスト クラスに実装されたすべてのテスト メソッドがリスト形式で表示されます。
リストの行をクリックすると、その行が選択されます。Ctrl キーを押しながらクリックすると、複数の行を選択できます。
また、Shift キーを押しながら 2 つの行をクリックすると、その 2 つの間のすべての行が選択されます。
このようにして実行したいテスト メソッド (の行) を選択して、**[テスト ビュー]** ウィンドウの上部の左端にある **[選択範囲の実行]** ボタンを押します。
これにより、選択していた単体テストのみが実行されます。



[テスト ビュー] ウィンドウによる単体テストの実行

[テスト リスト エディタ] ウィンドウで実行する

[テスト ビュー] ウィンドウで選択的に単体テストを実行できるのはわかりましたが、毎回決まり切ったテストメソッドを多数選択するような場合では、そのたびに **[テスト ビュー]** ウィンドウで選択し直すのは面倒です。
1 回選択したテスト メソッドの組み合わせを保存したいという要望は少なくないでしょう。
これを実現するのが [テストリスト エディタ] ウィンドウです。
[テスト リスト エディタ] ウィンドウは、メニュー バーの **[テスト]** - **[ウィンドウ]** - **[テスト リスト エディタ]** を実行すると表示されます。



[テスト リスト エディタ] ウィンドウによる単体テスト

テストの実行

[テスト リスト エディタ] ウィンドウは、テスト メソッドの組み合わせを「テスト リスト」として扱います。

よってまずは、テスト リストを作成する必要があります。

これには、メニュー バーの **[テスト]** - **[新しいテスト リストの作成]** を選択し、**[新しいテスト リストの作成]** ダイアログ ボックスを表示します。

[新しいテスト リストの作成] ダイアログ ボックスでテスト リストの **[名前]** や **[説明]** などを入力し、**[OK]** ボタンを押してテスト リストを作成します。

作成したテスト リストは、基本的には **[テスト リスト エディタ]** ウィンドウ内の左側にあるツリー内の「テストの一覧」ノードの直下に追加されます。

追加されたノードを選択した状態で、右側のリスト部分にテストメソッドを追加していきます。

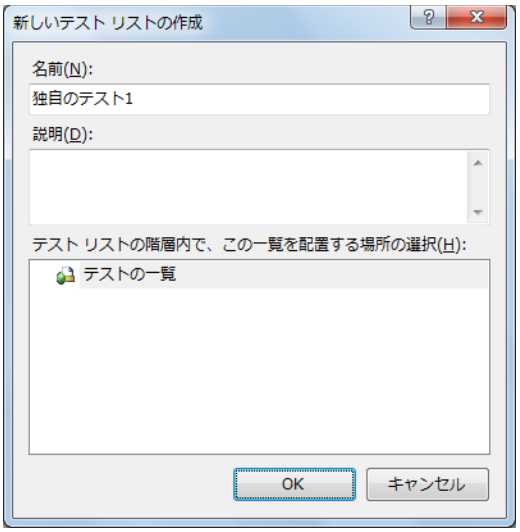
追加するには、先ほどの **[テスト ビュー]** ウィンドウ内のリストの行 (テスト メソッド) を、**[テスト リスト エディタ]** ウィンドウのリスト部分にドラッグ アンド ドロップするだけです。

この際、複数の行を選択して、一気に追加することもできます。以上でテスト リストの作成は完了です。

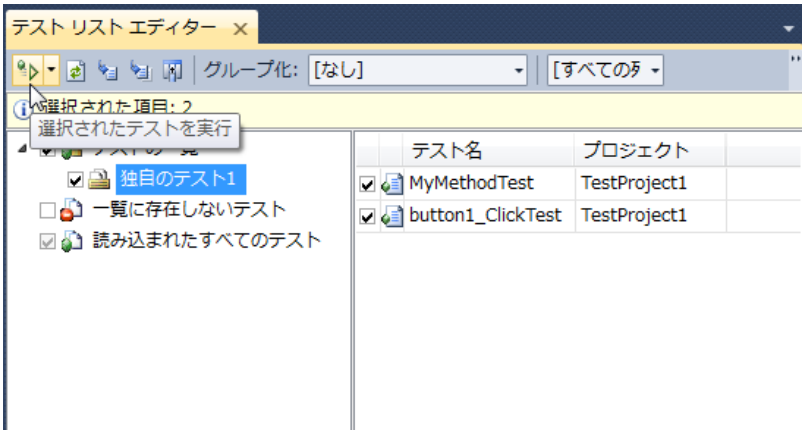
後はこのテスト リストを実行します。

これには先ほど「テストの一覧」ノードの配下に追加したノードの左にあるチェック ボックスにチェックを入れ、**[テスト リスト エディタ]** ウィンドウの上部の左端にある **[選択されたテストを実行]** ボタンを押します。

これにより、選択していたテスト リスト内の単体テストすべてが実行されます。



新しいテストリストの作成



テスト リストの実行

単体テストの結果の確認

単体テストが無事に実行できたら、そのテスト結果を確認します。

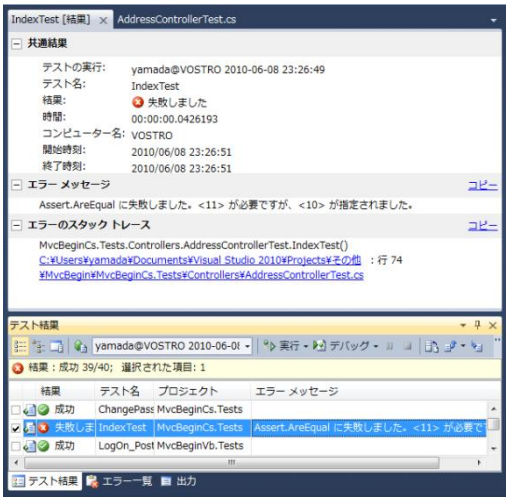
これには **[テスト結果]** ウィンドウを使用します。**[テスト結果]** ウィンドウは単体テスト実行後に自動的に表示されます。手動で表示するには、メニュー バーの **[テスト]** - **[ウィンドウ]** - **[テスト結果]** を実行します。

[テスト結果] ウィンドウでは、単体テストのテスト メソッドの実行結果がリスト形式で各行に表示されます。実行結果の種類には、緑色の「成功」や、赤色の「失敗しました」、黄色の「エラー」などがあります。

失敗した内容の詳細を確認するには、その行をダブル クリックします。

これによって、ドキュメント ウィンドウにテスト結果の詳細が表示されます。

そのテスト結果の **[エラーのスタック トレース]** 欄にあるソース ファイルへのリンクをクリックすると、単体テストに失敗した問題のコード箇所にジャンプできます



[テスト結果] ウィンドウによる単体テストの実行結果の確認

一つのテスト パターンを異なるデータで確認

例えば上限値、下限値といったしきい値の確認など、一つのテスト パターンを異なるデータで確認したいケースが考えられます。Visual Studio 2010 による単体テスト機能では、テスト データを外部に定義し、各変数とデータ連結して動的に指定することで、テスト データのレコード数分、自動的にテストを実施させることもできます。このデータ駆動のテストを実施するには、事前にデータベース、CSV 形式、XML 形式のいずれかの形式でテスト データを作成します。次に [テスト リスト エディタ] および [テスト ビュー] で該当のメソッドを選択し、[プロパティ] ウィンドウより [データ接続文字列] プロパティに適切な接続文字列を設定し、必要に応じて [データ テーブル名] プロパティに該当のデータを含むテーブルを指定します。

次にテスト コードを修正しましょう。[データ接続文字列] プロパティの設定により、自動的に DataSource 属性が追加されているのが確認できます。各テスト データは TestContext.DataRow でアクセスすることができます。以下の例では “x”, “y” という 2 つの条件データと、”ans” という期待する結果データをそれぞれの変数にデータ連携しています。

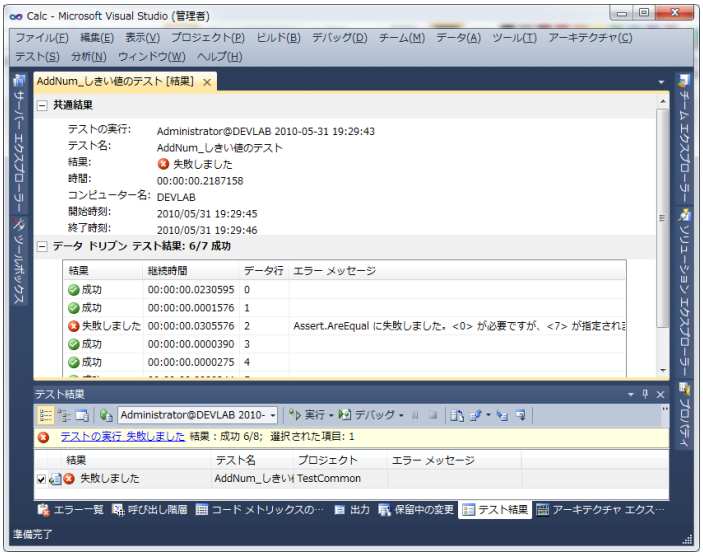
テスト データ (CSV 形式)

```
x, y, ans
1, 1, 2
1, 2, 3
100, 100, 200
```

テスト コード (C#)

```
        [DataSource (・・・), TestMethod()]
        public void AddNumberTest ()
        {
            Class1 target = new Class1(); // TODO: 適切な値に初期化してください
            int x = (int) TestContext.DataRow["x"]; // TODO: 適切な値に初期化してください
            int y = (int) TestContext.DataRow["y"]; // TODO: 適切な値に初期化してください
            int expected = (int) TestContext.DataRow["ans"]; // TODO: 適切な値に初期化してください
            int actual;
            actual = target.AddNum(x, y);
            Assert.AreEqual(expected, actual);
        }
```

テストを実施すると、テスト データのレコード数分、自動的にテストを実施します。その中の一つのテストが失敗した場合には、テスト全体が失敗とみなされます。同じくテスト結果をダブル クリックすることで実施したすべてのテスト データ レコードごとの詳細を確認することができます。



[テスト結果] 一つのテスト パターンを外部データのレコード数分確認

リファクタリング

単体テストと関連が深い、リファクタリングについて触れておきましょう。

リファクタリングとは、途中の仕様変更やバグ修正などにより冗長になってしまったコードを、すっきりとした読みやすいコードに書き直すことです。

このようなコードの改善を行うのは、冗長なコードをそのまま放置しておくと、後から保守しようとしても容易にはできないコードになってしまう、コードの内容を解読しにくいがゆえにそれが原因で新たなバグを作ることになってしまう、というような問題が発生する可能性を低減させるためです。

しかし、そんなに簡単にコードを書き直しても大丈夫なのでしょうか。

一昔前は「動いているコードは触るな」を格言とする開発者が多くいました。

実際に、要らぬコード修正を加えたがために新たなバグを生んでしまったという事例は、枚挙にいとまがありません。

このような事態に陥らないためには、書き直しの前と後でコードの振る舞いが変わらないことを保証する必要があるのです。

ここで使われるのが単体テストです。

たとえばメソッドのコード内容を書き直す場合、そのメソッドの単体テストの実行結果が、書き直し後も以前と変わらずに成功するのであれば、「コードは書き換えられても、振る舞い自体は何も変わってない」ということが証明されたことになります。

このように、リファクタリングを行うには、あらかじめ適切な単体テストが用意されていることが不可欠です。

単体テストがあって初めて、開発者は安心して勇気を持ってリファクタリング、つまりコードの改善に取り組めるのです。

このことは非常に重要です。

コード変更による副作用を引き起こさないためにも適切な単体テストを用意せずに、むやみにリファクタリングをしてしまわないように気を付けましょう。

Visual C# では 6 つのリファクタリング機能が搭載されています (Visual Basic では「名前の変更」のみがサポートされています)。

もっとも、実際にリファクタリングによるコードの改善を行うには、たった 6 つのリファクタリング機能だけでは足りないことが多いでしょう。

しかし、Visual C# のリファクタリング機能を活用すれば、リファクタリングにより必要となるコードの書き直しを、ソリューション全体にわたって自動的に正確に行ってくれるので、やはり使わない手はありません。
たとえば機能の追加、変更によってメソッド名が適切でなくなった場合に実施する「名前の変更」はちょっとした変更に思えますが、実際にはそのメソッドを参照している他のコードを変更する必要があります。
Visual Studio 2010 のリファクタリング機能では、このような依存関係を持つコードに対しても適切な変更を実施します(依存関係を持つコードが同一ソリューションに含まれている必要があります)。

Visual C# でリファクタリング機能を使用するには、コード エディタ上のメソッド名などを右クリックして、コンテキスト メニューから **[リファクター]** の配下のメニューを選択します。
もしくは、コード エディタ上のメソッド名の部分などにテキスト カーソル (キャレット) を置いたまま、メニュー バーの **[リファクター]** の配下のメニューを選択します。Visual C# で利用可能なリファクタリングのメニューは、次の表を参考にしてくださいい。

メニュー	名機能の説明
名前の変更	メソッドやクラスなどの名前を変更する際に、そのメソッドやクラスを使用しているコードも一緒に変更します
メソッドの抽出	メソッド内の一部のコードを新たなメソッドとして切り出します
フィールドのカプセル化	クラス内の public フィールド変数を private 変数にカプセル化して、その変数にアクセスする方法をプロパティ経由に変更します
インターフェイスの抽出	複数のクラスで共通のメソッドを利用したい場合に、そのクラスのメソッド定義を、インターフェイスのメソッド定義として抽出します
パラメーターの削除	メソッドのパラメーターを削除します
パラメーターの順序の再変更	メソッドのパラメーターの順番を変更します

なお、Visual Basic で「名前の変更」のリファクタリングを使用するには、コード エディタ上のメソッド名などを右クリックして、コンテキスト メニューから **[名前の変更]** を選択します。

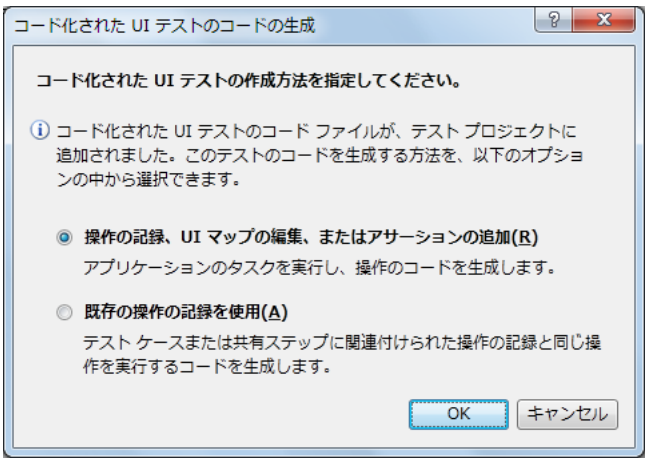
自動 UI テスト

Visual Studio 2010 Premium 以上のエディションでは、新しく自動 UI テスト機能が追加されました。自動 UI テストは、実際に ユーザーが Windows 上で行った操作を記録しておいて、その操作をすべて自動的に動作テストするものです。
たとえば、アプリケーションの起動から終了までの一連の操作を記録して、そのすべての動作を一通りテストします。

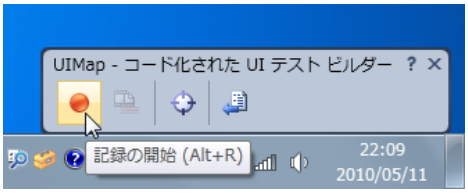
自動 UI テストを作成するには、**[ソリューション エクスプローラー]** ウィンドウのテストプロジェクト項目を右クリックし、表示されるコンテキスト メニューから **[追加]] – [コード化された UI テスト]** を選択します。
[コード化された UI テストのコードの生成] ダイアログ ボックスが表示されるので、**[操作の記録、UI マップの編集、またはアサーションの追加]** を選択して**[OK]** ボタンをクリックします。
デスクトップの右下（タスク バー の上に）**[UIMap - コード化された UI テスト ビルダー]** が表示されます。

[記録の開始] をクリックして、テストを行いたい一連の操作を行った後、**[記録に一時停止]** をクリックして、次に **[コードの生成]** をクリックします。メソッド名を入力するダイアログが表示されるので入力後 **[追加と生成]** ボタンをクリックします。以上で 自動 UI テストが作成され、コードが生成されます。

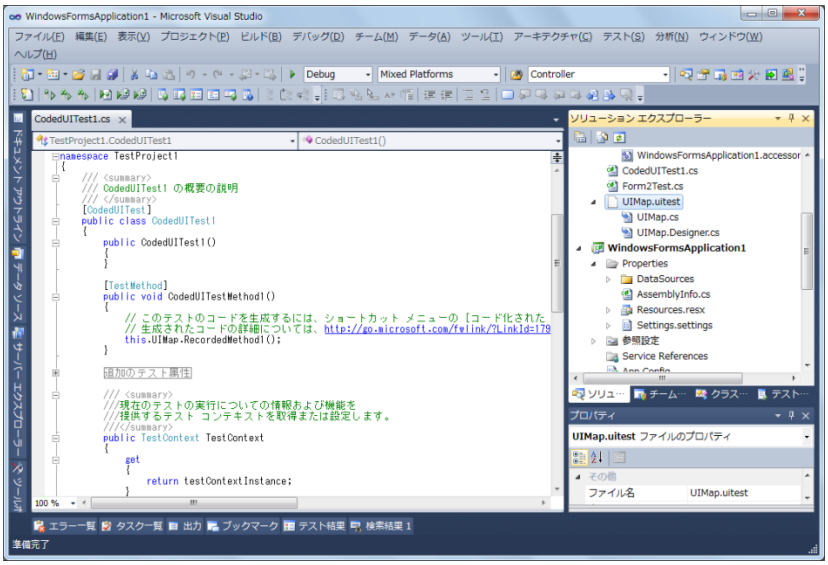
テストの実行は単体テストと同じ要領で行います。



[コード化された UI テストのコードの生成] ダイアログ ボックス



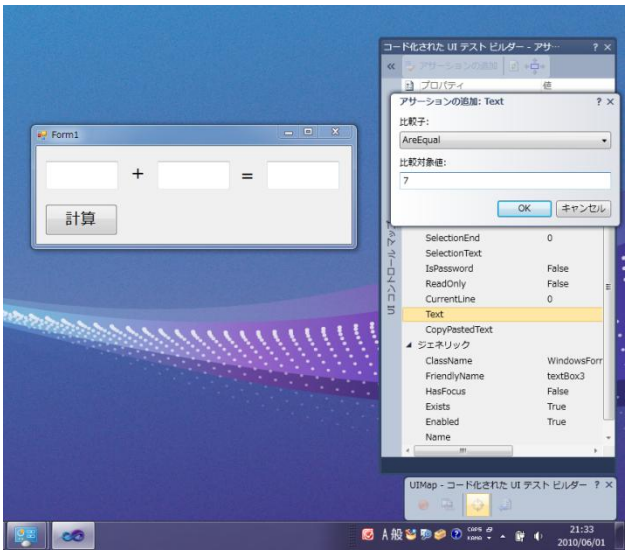
記録の開始



作成された自動 UI テスト

UI 操作の記録をそのまま再生することで、操作のみの検証を行うこともできますが、場合によってはそのテストによってコントロールに値が設定されているかを検証したい場合があります。特定のコントロールの値を検証するには、UI テストビルダーの左から 3 番目の [アサーションの追加] ボタンを使用します。[アサーションの追加] ボタンをドラッグしながら該当のコントロール上でドロップします。該当のコントロールのプロパティ一覧が表示されますので、検証したいプロパティを選択し [アサーションの追加] ボタンをクリックします。

検証の式として、単体テストで解説した演算子と、比較対象の値を指定します。最後に再度 [コードの生成] ボタンをクリックし、検証メソッドを作成します。



UI 操作においてパラメーターを変更したい場合 (例えばテキスト ボックスの入力値を変更するなど)、生成されたテスト コードを修正します。UI 操作における定数については ”UI 操作記録メソッド名 Params” というクラスが生成され、それぞれプロパティが設定されます。同様に上記「コントロールの値の検証」の手順で作成した検証項目については、”検証メソッド名 ExptectedValues” というクラスが生成されそれぞれプロパティが設定されます。値を変更する場合には、UI 操作記録メソッド、および検証メソッドの呼び出し前に、それぞれプロパティを設定します。

```
[TestMethod]
  public void CodedUITestMethod1()
  {
      this.UIMap.RecordedMethod1Params.UITextBox1EditText = “1”;
      this.UIMap.RecordedMethod1Params.UITextBox2EditText = “2”;

      this.UIMap.RecordedMethod1();

      this.UIMap.AssertMethod1ExpectedValues.UITextBox3EditText = “3”;

      this.UIMap.AssertMethod1();
  }
```

また、コード化された UI テストにおいても、単体テストと同様、CSV やデータベースを用意してデータ連携することで、一つのテストパターンを異なるデータで確認することが可能です。手順は基本的に単体テストの時と同様です。[テスト リスト エディタ] および [テスト ビュー] より [データ接続文字列] プロパティに適切な接続文字列を設定します。次にテスト コードの各プロパティを TestContext.DataRow でデータ連結します。

```
  [DeploymentItem (・・・), TestMethod]
```

```
public void CodedUITestMethod1()
{
    this.UIMap.RecordedMethod1Params.UITextBox1EditText = TestContext.DataRow["x"].ToString();
    this.UIMap.RecordedMethod1Params.UITextBox2EditText = TestContext.DataRow["y"].ToString();

    this.UIMap.RecordedMethod1();

    this.UIMap.AssertMethod1ExpectedValues.UITextBox3EditText = TestContext.DataRow["ans"].ToString();

    this.UIMap.AssertMethod1();
}
```

第 8 章 アプリケーション ライフサイクル マネジメント

ここまでに紹介してきた一連の作業の流れは、「開発」というプロセスにフォーカスしたものでした。しかし実際のアプリケーション開発では、「システムの設計」や「システムの展開/ 運用」など、開発以外のプロセスも存在します。
これらすべてのプロセスを含めたアプリケーション構築全体の流れを「アプリケーション ライフサイクル」と呼びます。

アプリケーション ライフサイクルの管理

システム構築を成功させるには、開発プロセスだけではなく、アプリケーション ライフサイクル全体を考慮しなければなりません。
たとえば展開/運用を考えずに設計したような場合、実際に運用を開始しようとした時点で問題が発生してしまい、最悪の場合、設計し直し、作り直し、という大きなコストを支払わらなければならなくなる可能性もあります。
つまり、開発以外の設計や展開/運用にも適切な計画や準備が必要だということです。
アプリケーション ライフサイクルを包括的に管理する必要があるのです。
こういった考え方をアプリケーション ライフサイクル マネジメント (ALM) と呼びます。

以前は、Visual Studio Team System というソリューションで提供されていましたが、Visual Studio 2010 では大きく体系が見直され、その多くの機能は上位エディションに統合されました。

Visual Studio 2010 では、「設計」「開発」「テスト」「展開/運用」という一連のアプリケーション ライフサイクル全体を支援する機能が用意されています。

Team Foundation Server とは?

Visual Studio 2010 の良さは、これだけではありません。チームのメンバが共同で効率よく作業さるために必要な機能も揃っているのです。その機能の中心となるのが Team Foundation Server です。

> Microsoft Visual Studio Team Foundation Server 2010

Team Foundation Server はアプリケーション ライフサイクルの中で取り交わされるさまざまな情報を集中管理するためのチーム開発用サーバーです。
Team Foundation Server は、Visual Studio とは別にインストールが必要です。以前の Team Foundation Server は、インストールも複雑で、サーバーOS のみの対応でした。
しかし、Team Foundation Server 2010 からは、基本構成であれば、クライアント OS (Winodws Vista 以上) にもインストールが可能で、インストール作業とその後の構成作業が分離されたことにより、インストール自体は格段に易しくなりました。
インストールの詳細については本ドキュメントでは割愛します。以下では、この Team Foundation Server の機能についてご紹介しましょう。

Team Foundation Server の 主な機能

Team Foundation Server 2010 には、「基本構成」「標準構成」「詳細構成」という 3 つの構成が用意されています。

「基本構成」は、一番単純な構成で、「ソースコード管理」「自動ビルド」「作業項目管理」の 3 つの機能が利用できます。

「標準構成」は、Team Foundation Server のすべての機能を 1 台のサーバーを使用して利用する構成で、「詳細構成」は、機能ごとに複数のサーバーを利用する構成になっています。
以下では Team Foundation Server の主な機能について簡単に紹介します。

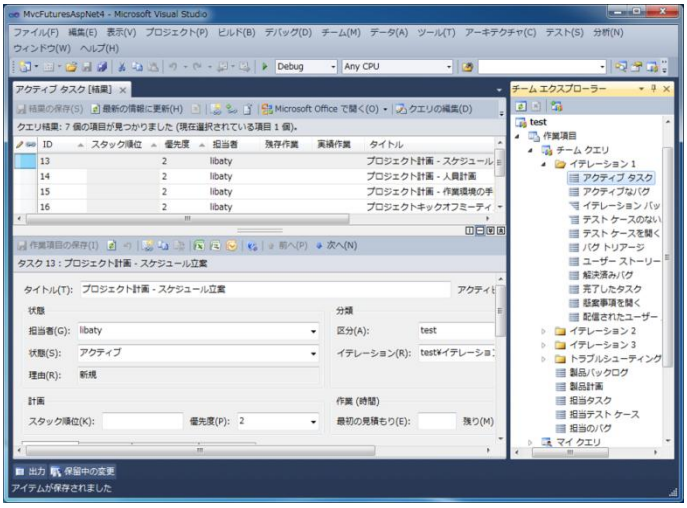
(1) プロジェクト管理

チームでアプリケーション開発を協調的に進めていくには、チーム内のメンバが一定のルールに従う必要があります。
たとえば、Visual C#/Visual Basic などの書き方を定めるコーディング規約や、開発の進め方を定義する開発プロセスなどです。
Team Foundation Server では、このようなルールに関する情報を着実に共有し、ルールに沿った開発を効率的に推進するためのプロジェクト管理機能が備わっています。
具体的にはアジャイル開発プロセスや CMMI (Capability Maturity Model Integration) などの開発プロセス テンプレートを選択することで、その開発プロセスに必要な標準的なひな形のドキュメント群が提供されます。
これらのドキュメントは、Visual Studio 2010 の **[チーム エクスプローラー]** ウィンドウや、後述のプロジェクト ポータルから参照/編集できます。

開発プロセスや各種ドキュメントが提供されるプロジェクト管理機能

(2) 作業項目管理

Team Foundation Server では、このようは作業の割り当てや追跡を行うための作業項目管理機能が搭載されています。
作業項目情報の登録や検索は Visual Studio 2010 の **[チーム エクスプローラー]** ウィンドウから容易に行えます。
このように普段の開発環境から作業項目を直接参照/ 編集できることは、開発者にとって有益です。
さらに、Microsoft Office system (Excel および Microsoft® Office Project) も使え、各作業の進捗状況などを自動的に収集し計算する機能も持っています。
したがって、プロジェクトマネージャは使い慣れたアプリケーションを使って各開発者の担当作業項目をリアルタイムに管理することができます。



開発環境から参照/編集できる作業項目管理機能

(3) ソース管理

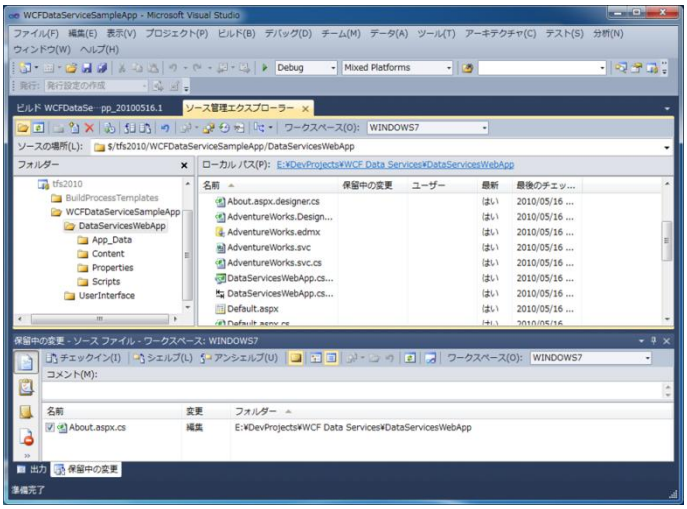
複数の開発者が同じソース コードを編集するチーム開発では、開発環境によるソース コードの管理機能は欠かせません。

また、ソース コードは時間と共に刻々と変化するものなので、その変更を絶えず記録するためのバージョン管理の機能も欠かせないでしょう。

Team Foundation Server では、安定性の高い SQL Server データベースを基盤にしたソース管理機能を提供しています。

また、ソースコードのチェックイン時に、ポリシーを設定することができます。

たとえば、チェックイン時に自動的に単体テストが実行され、そのテストをパスしないとチェックインすることができないポリシーや、特定の作業項目と関連付け、履歴をとらないとチェックインできないなど、品質の向上やコンプライアンスへの対応などにも有効な機能を持っています。



開発環境からシームレスに操作できる変更管理機能

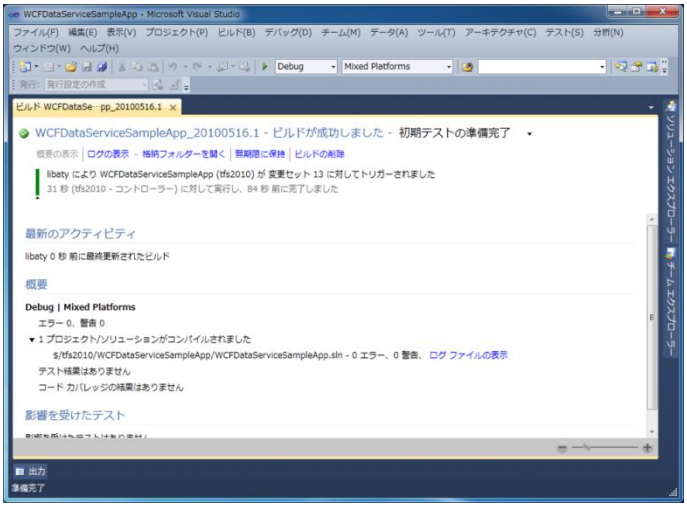
(4) 自動ビルド

アプリケーション開発においては、「不具合の解消が遅くなればなるほど、開発コストが肥大化する」といわれています。

そのため、不具合ができるだけ早く見つかるように、日常的にアプリケーション全体のビルドとテストを行うべきです。

これを実現する自動ビルド機能が、Team Foundation Server には搭載されています。

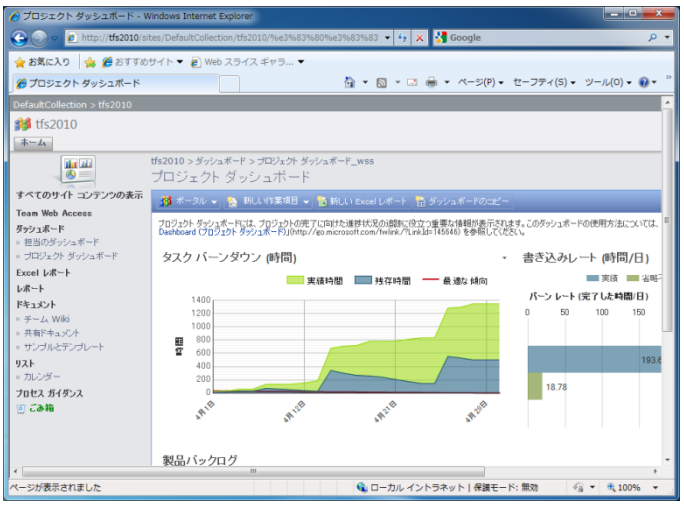
前述した手順で「単体テスト」を作成し、自動ビルド機能で自動実行することで、問題を早期に発見、解決することできます。
これにより、アプリケーションの品質を高く保つことが可能になります。



夜間に無人のビルドとテストなどを実現する自動ビルド機能

(5) プロジェクト ポータル

Team Foundation Server では各メンバとの情報共有に役立つプロジェクト ポータル (SharePoint Services) が作成されます。
これはチーム開発のポータル サイトです。
前述のプロジェクト管理機能の情報や、後述のレポーティング機能の情報もここで参照/編集できます。
なお、同様の情報は、Visual Studio 2010 の **[チーム エクスプローラー]** ウィンドウからでもアクセスできます。



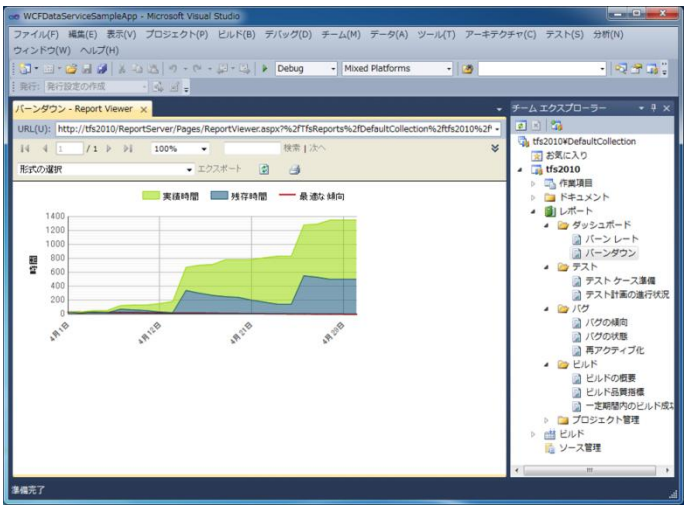
すべての開発チーム メンバのポータルとなるプロジェクト ポータル

(6) レポーティング

プロジェクト マネージャにとって、アプリケーションの開発を滞りなく進めることは至上命題です。

そのためには、常に作業状況やバグ状況、テスト状況を把握しなければなりません。

これを支援するレポーティング機能 (SQL Server Reporting Services) が Team Foundation Server には搭載されています。



プロジェクトの進行や品質を管理できるレポーティング機能

以上のような機能が示す作業は、開発チームの規模が大きくなればなるほど必要になるでしょう。 Team Foundation Server をアプリケーション開発の中心に導入することで、これらの準備にかかる初期コストや、開発コスト、管理コストを効果的に低減できるようになります。

Appendix. テンプレートの一覧表

以下のプロジェクト テンプレートや Web サイト テンプレートは、いずれも .NET Framework 4 を選択した場合のものです。

プロジェクト テンプレート一覧

[新しいプロジェクト] ダイアログ ボックスを表示するには、メニュー バーから **[ファイル]** - **[新規作成]** - **[プロジェクト]** を選択します。

テンプレート		説明	
Visual C#/ Visual Basic	Windows	Windows フォーム アプリケーション	Windows フォーム ユーザー インターフェイスを含むアプリケーションを作成するためのプロジェクト
		Windows フォーム コントロール ライブラリ	Windows フォーム アプリケーションで使用するコントロールを作成するためのプロジェクト
		WPF アプリケーション	WPF ユーザー インターフェイスを含むクライアント アプリケーションを作成するためのプロジェクト
		WPF ブラウザ アプリケーション	WPF ユーザー インターフェイスを含むブラウザ アプリケーションをするためのプロジェクト
		WPF ユーザー コントロール ライブラリ	WPF で使用できるユーザー コントロールを作成するためのプロジェクト
		WPF カスタム コントロール ライブラリ	複雑な独自の機能を持つカスタム コントロールを作成するためのプロジェクト
		クラス ライブラリ	クラス ライブラリ (.dll ファイル) を作成するためのプロジェクト
		コンソール アプリケーション	コマンドライン アプリケーションを作成するためのプロジェクト
		Windows サービス	Windows サービスを作成するためのプロジェクト
		空のプロジェクト	ローカル アプリケーションを作成するための、空のプロジェクト
	Web	ASP.NET Web アプリケーション	Web ユーザー インターフェイスを含むアプリケーションを作成するためのプロジェクト
		ASP.NET サーバー コントロール	Web アプリケーションで使用するコントロールを作成するためのプロジェクト
		ASP.NET AJAX サーバー コントロール	Web アプリケーションで使用する AJAX コントロールを作成するためのプロジェクト
		ASP.NET AJAX サーバー コントロール エクステンダ	ASP.NET サーバー コントロールのクライアント機能を拡張するエクステンダを作成するためのプロジェクト
		ASP.NET MVC2 アプリケ	ASP.NET MVC2 を使ったアプリケーションを作成

		ーション	するためのプロジェクト
		ASP.NET 動的データ エンティティ Web アプリケーション	Entity Data Model を利用した動的データアプリケーションを作成するためのプロジェクト
		ASP.NET 動的データ LINQ to SQL Web アプリケーション	LINQ to SQL を利用した動的データアプリケーションを作成するためのプロジェクト
		ASP.NET Web サービス アプリケーション	Web サービスを作成するためのプロジェクト
		WCF サービス アプリケーション	通信技術の WCF を使ったサービス (主に Web サービス) を作成するためのプロジェクト
	Office	Excel 2010 ブック	新規または既存の Excel 2010 ブックの背後で動作するプログラムを作成するプロジェクト
		Excel 2007 ブック	新規または既存の Excel 2007 ブックの背後で動作するプログラムを作成するプロジェクト
		Excel 2010 テンプレート	新規または既存の Excel 2010 テンプレートの背後で動作するプログラムを作成するプロジェクト
		Excel 2007 テンプレート	新規または既存の Excel 2007 テンプレートの背後で動作するプログラムを作成するプロジェクト
		Word 2010 ドキュメント	新規または既存の Word 2010 ドキュメントの背後で動作するプログラムを作成するプロジェクト
		Word 2007 ドキュメント	新規または既存の Word 2007 ドキュメントの背後で動作するプログラムを作成するプロジェクト
		Word 2010 テンプレート	新規または既存の Word 2010 テンプレートの背後で動作するプログラムを作成するプロジェクト
		Word 2007 テンプレート	新規または既存の Word 2007 テンプレートの背後で動作するプログラムを作成するプロジェクト
		Excel 2010 アドイン	Excel 2010 用のマネージ コード アドインを作成するためのプロジェクト
		Excel 2007 アドイン	Excel 2007 用のマネージ コード アドインを作成するためのプロジェクト
		Word 2010 アドイン	Word 2010 用のマネージ コード アドインを作成するためのプロジェクト
		Word 2007 アドイン	Word 2007 用のマネージ コード アドインを作成するためのプロジェクト
		Outlook 2010 アドイン	Outlook 2010 用のマネージ コード アドインを作成するためのプロジェクト
		Outlook 2007 アドイン	Outlook 2007 用のマネージ コード アドインを作成するためのプロジェクト
		PowerPoint 2010 アドイン	PowerPoint 2010 用のマネージ コード アドインを作成するためのプロジェクト
		PowerPoint 2007 アドイン	PowerPoint 2007 用のマネージ コード アドインを作成するためのプロジェクト

		Visio 2010 アドイン	Visio 2010 用のマネージ コード アドインを作成するためのプロジェクト
		Visio 2007 アドイン	Visio 2007 用のマネージ コード アドインを作成するためのプロジェクト
		Project 2010 アドイン	Project 2010 用のマネージ コード アドインを作成するためのプロジェクト
		Project 2007 アドイン	Project 2007 用のマネージ コード アドインを作成するためのプロジェクト
		InfoPath 2010 アドイン	InfoPath 2010 用のマネージ コード アドインを作成するためのプロジェクト
		InfoPath 2007 アドイン	InfoPath 2007 用のマネージ コード アドインを作成するためのプロジェクト
	Cloud	Windows Azure Cloud Service	Windows Azure を使ったアプリケーションを作成するためのプロジェクト
	Reporting	レポート アプリケーション	Windows ユーザー インターフェイスとレポートを含むアプリケーションを作成するためのプロジェクト
		Crystal Reports アプリケーション	Windows ユーザー インターフェイスと Crystal レポートを含むアプリケーションを作成するプロジェクト
	SharePoint	空の Share Point プロジェクト	Share Point アプリケーションを作成するための空のプロジェクト
		視覚的 Web パーツ	Share Point 視覚的 Web パーツを作成するためのプロジェクト
		シーケンシャルワークフロー	Share Point シーケンシャル ワークフローを作成するためのプロジェクト
		ステート マシンのワークフロー	Share Point ステート マシンのワークフローを作成するためのプロジェクト
		ビジネス データ 接続モデル	Share Point ビジネス データ 接続モデルを作成するためのプロジェクト
		イベント レシーバー	Share Point イベント処理を可能にする レシーバーを作成するためのプロジェクト
		リスト 定義	Share Point リスト 定義を作成するためのプロジェクト
		コンテンツ タイプ	Share Point コンテンツ タイプを作成するためのプロジェクト
		モジュール	Share Point モジュールを作成するためのプロジェクト
		SharePoint 2007 シーケンシャルワークフロー	SharePoint シーケンシャル ワークフローを作成するためのプロジェクト
		SharePoint 2007 ステート マシンのワークフロー	SharePoint ステート マシンのワークフローを作成するためのプロジェクト
		サイト 定義	Share Point サイト 定義を作成するためのプロジ

			エクト
		再利用可能なワークフローのインポート	ソリューションパッケージ内の宣言型ワークフローテンプレートから新しい Share Point ソリューションを作成するためのプロジェクト
		Share Point ソリューション パッケージのインポート	既存のパッケージをもとに新しいソリューションを作成するプロジェクト
	Silverlight	Silverlight アプリケーション	Silverlight を使って RIA (リッチ インターネット アプリケーション) を作成するためのプロジェクト
		Silverlight クラス ライブラリ	Silverlight クラス ライブラリを作成するためのプロジェクト
		Silverlight ナビゲーション アプリケーション	ナビゲーションフレームワークを利用した Silverlight アプリケーションを作成するためのプロジェクト
	WCF	WCF サービス ライブラリ	WCF サービス クラス ライブラリ (.dll ファイル) を作成するためのプロジェクト
		WCF サービス アプリケーション	WCF サービスを作成するためのプロジェクト
		WCF ワークフロー サービス アプリケーション	WCF ワークフロー サービス アプリケーションを作成するためのプロジェクト
		配信サービス ライブラリ	WCF サービスとして配信されるライブラリを作成するためのプロジェクト
	Workflow	アクティビティ デザイナー ライブラリ	アクティビティ デザイナー テンプレート
		アクティビティ ライブラリ	ワークフロー アクティビティ ライブラリを作成するための空のプロジェクト
		ワークフロー コンソール アプリケーション	ワークフロー コンソール アプリケーションを作成するための空のプロジェクト
	テスト	テスト プロジェクト	単体テストや Web テスト、ロード テストなどのテストを含むプロジェクト
Visual C++	ATL	ATL プロジェクト	ATL を使ったプロジェクト
	CLR	クラス ライブラリ	クラス ライブラリ (.dll ファイル) を作成するためのプロジェクト
		CLR コンソール アプリケーション	コマンドライン アプリケーションを作成するためのプロジェクト
		Windows フォーム アプリケーション	Windows フォーム アプリケーションを作成するためのプロジェクト
		空の CLR プロジェクト	ローカル アプリケーションを作成するための空のプロジェクト
		Windows フォーム アプリケーション	Windows アプリケーションで使用するコントロールを作成するためのプロジェクト
	全般	メイクファイル プロジェ	外部のビルド システムを使用するためのプロジェ

		クト	クト
		空のプロジェクト	ローカル アプリケーションを作成するための空のプロジェクト
		カスタム ウィザード	カスタム ウィザードを作成するためのプロジェクト
	MFC	MFC DLL	MFC ライブラリを使用するダイナミック リンク ライブラリ (.dll ファイル) を作成するためのプロジェクト
		MFC ActiveX コントロール	MFC ライブラリを使用する ActiveX コントロールを作成するためのプロジェクト
		MFC アプリケーション	MFC ライブラリを使用するプロジェクト
	テスト	テスト プロジェクト	単体テストや Web テスト、ロード テストなどのテストを含むプロジェクト
	Win32	Win32 プロジェクト	Win32 アプリケーションやライブラリなど (ウィザードにより選択) を作成するためのプロジェクト
		Win32 コンソール アプリケーション	Win32 コンソール アプリケーションを作成するためのプロジェクト
Visual F#	Windows	F# アプリケーション	コマンドライン アプリケーションを作成するためのプロジェクト
		F# ライブラリ	F# ライブラリを作成するためのプロジェクト
		F# チュートリアル	F# のチュートリアルを作成するためのプロジェクト
	Silverlight	F# Silverlight ライブラリ	F# Silverlight ライブラリを作成するためのプロジェクト
その他のプロジェクトの種類	セットアップと配置	InstallShield Limited Edition の有効化	Windows のインストーラを作成するためのプロジェクト
		セットアッププロジェクト	Windows アプリケーションをクライアントに配置するインストーラを作成するためのプロジェクト
		Web セットアップ プロジェクト	Web アプリケーションを Web サーバーに配置するインストーラを作成するためのプロジェクト
		マージ モジュール プロジェクト	インストーラで再利用可能なモジュールを作成するためのプロジェクト
		セットアップ ウィザード	ウィザードを使って Windows のインストーラを作成するためのプロジェクト
		CAB プロジェクト	Web ブラウザでダウンロードできる ActiveX コンポーネントのパッケージを作成するためのプロジェクト
	拡張機能	共有アドイン	多くのホストで読み込み可能なアドインを作成するプロジェクト
		Visual Studio アドイン	Visual Studio ベースのホストで読み込み可能なアドインを作成するプロジェクト

	Visual Studio ソリューション	空のソリューション	プロジェクトを含まない空のソリューションを作成する
データベース	SQL Server	SQL Server 2005 データベース プロジェクト	SQL Server 2005 ユーザー データベースを定義する
		SQL Server 2005 サーバープロジェクト	SQL Server 2005 のサーバーレベルのオブジェクトや SQL マスター データベースへの変更を定義する
		SQL Server 2005 ウィザード	SQL Server 2005 のプロジェクト作成を行うウィザード
		SQL Server 2008 データベース プロジェクト	SQL Server 2008 ユーザー データベースを定義する
		SQL Server 2008 サーバープロジェクト	SQL Server 2008 のサーバーレベルのオブジェクトや SQL マスター データベースへの変更を定義する
		SQL Server 2008 ウィザード	SQL Server 2008 のプロジェクト作成を行うウィザード
		SQL Server データ層 アプリケーション	SQL Server の データ層 アプリケーションを作成する
		Visual Basic SQL CLR データベース	Visual Basic で SQL CLR 統合を作成するためのプロジェクト
		Visual C# SQL CLR データベース	Visual C#で SQL CLR 統合を作成するためのプロジェクト
	モデリング プロジェクト	モデリング プロジェクト	関連モデルのセットを作成するための空のプロジェクト
テストプロジェクト	テスト ドキュメント	テスト プロジェクト	単体テストや Web テスト、ロード テストなどのテストを含むプロジェクト

Web サイト テンプレート一覧

[新しい Web サイト] ダイアログ ボックスを表示するには、メニュー バーから **[ファイル]** - **[新規作成]** - **[Web サイト]** を選択します。なお、「Web サイト」と「プロジェクト」は、似て非なるものです。「Web サイト」という開発単位は、Web サイトの作成に特化しており、それ自体が Web サイト (もしくは その 1 つのディレクトリ)を体現しているという特徴があります。

テンプレート		**説明**	
Visual C#/ Visual Basic	Web サイト	ASP.NET Web サイト	Web ユーザー インターフェイスを含むサイトを作成するための Web サイト
		ASP.NET 空の Web サイト	空の Web サイト
		ASP.NET 動的データ エンティティ Web サイト	Entity Data Model 対応の ASP.NET 動的データ Web サイトを作成するための空の Web サイト
		ASP.NET 動的データ LINQ to SQL Web サイト	LINQ to SQL 対応の ASP.NET 動的データ Web サイトを作成するための空の Web サイト
		WCF サービス	通信技術の WCF を使ったサービス (主に Web サービス) を作成するための Web サイト
		ASP.NET レポート Web サイト	Web ユーザー インターフェイスとレポートを含むサイトを作成するための Web サイト
		ASP.NET Crystal Reports Web サイト	Web ユーザー インターフェイスと Crystal レポートを含むサイトを作成する Web サイト

www.microsoft.com/japan/msdn/vstudio