Data for Silverlight

2013.05.29 更新

グレープシティ株式会社

目次

製品の概要 2
- ComponentOne Studio for Silverlight のヘルプ 2

主な特長 3

データの概要 4

データ中心型アーキテクチャ 5

チュートリアル 6
- アプリケーションの作成 6
- 参照の追加 6-7
- UI の作成 7-8
- サーバー側の実装 8
  - データベースアクセスインフラストラクチャの追加 8-10
  - Web サービスの実装 10-13
- クライアント側の実装 13
  - Web サービスへの参照の追加 13-14
  - Web サービスを使用したデータの取得 14-16
  - ページ上のコントロールへのデータ連結 16-17
  - ビューの同期 17-18
  - 項目の追加および削除 19
  - サーバーへの変更のコミット 19-21
- データ転送の最適化 21
  - サーバー側でのデータの圧縮 21-22
  - クライアント側でのデータの圧縮解除 22-23

スケジュールでのデータの使用 24

まとめ 25

 Copyright © GrapeCity inc. All rights reserved.

製品の概要

Data for Silverlight は、Windows Forms や ASP.NET の開発者にはおなじみの標準の **DataSet**、**DataTable**、および **DataView** クラスを実装しています。以下のトピックでは、**C1.Silverlight.Data アセンブリを使ってデータ中心型** Silverlight アプリケーションを実装する方法について説明します。

ComponentOne Studio for Silverlight のヘルプ

はじめに

ComponentOne Studio for Silverlight のすべてのコンポーネントで共通の使用方法については、「ComponentOne Studio for Silverlight ユーザーガイド」を参照してください。

Copyright © GrapeCity inc. All rights reserved.

主な特長

Data for Silverlight を使用すると、機能豊富でカスタマイズされたアプリケーションを作成できます。次の主な特長を利用して、**Data for** Silverlight を最大限に活用してください。

- **すべての連結可能コントロールをサポート**
  Data アセンブリは、Silverlight データ連結をサポートします。Data は、Microsoft DataGrid を含むすべての連結可能コントロールと統合言語クエリー(Linq:Language Integrated Queries)をサポートします。Linq および最新の Silverlight コントロールを最大限に活用する一方で、DataSet、DataTable、DataView などの使い慣れたクラスを使用することにより、データを操作してサーバーと通信します。
- **再利用可能なビジネスロジック**
  Data を使用すると、Windows フォームと同じ方法とコードを使用して、Silverlightアプリケーションのデータを操作できます。データおよびビジネスロジックをカプセル化するデータライブラリを作成し、それを Silverlight アプリケーションと Windows フォームの両方で使用できます。
- **XML へのシリアライズ**
  XMLserialization により、一時的な接続のシナリオを強化しています。IsolatedStorage にデータを保存して、後 から Silverlight アプリケーションで使用したり、XML を使ってサーバーの Web サービスと通信できます。

Copyright © GrapeCity inc. All rights reserved.

データの概要

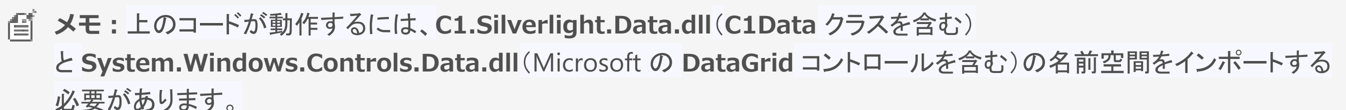
C1.Silverlight.Data アセンブリ内のクラスは、**System.Data** 名前空間にあるクラスのサブセットです。つまり、既存の ADO.NET ベースのコードを Silverlight アプリケーションで使用することができます。たとえば、次のコードは、**C1.Silverlight.Data** を使用して、**DataTable** オブジェクトを作成し、それにデータを挿入する方法を示します。

```
データテーブルを作成します
DataTable dt = new DataTable();

// いくつかの列を追加します
dt.Columns.Add("ID", typeof(int));
dt.Columns.Add("First Name", typeof(string));
dt.Columns.Add("Last Name", typeof(string));
dt.Columns.Add("Active", typeof(bool));

// 主キー列を設定します
DataColumn dc = dt.Columns["ID"];
dc.AutoIncrement = true;
dc.ReadOnly = true;
dt.PrimaryKey = dc; // 配列ではなく1つの列

// いくつかの行を追加します
for (int i = 0; i < 10; i++)
{
    dt.Rows.Add(i,
        "First " + i.ToString(),
        "Last " + i.ToString(),
        i % 2 == 0);
}
```

ADO.NET を使用したことがあれば、このコードは見慣れているはずです。唯一の違いとして、ADO.NET DataTable オブジェクトの **PrimaryKey** プロパティはいくつかの列から成るベクターですが、**C1.Silverlight.Data** の場合、このプロパティは1つの列になります。

DataTable を作成したら、テーブルの **DefaultView** プロパティを使用して、これを他のコントロールに連結することができます。

```
コントロールにテーブルを連結します
DataGrid dg = new DataGrid();
dg.ItemsSource = dt.DefaultView;
LayoutRoot.Children.Add(dg);
```



メモ：上のコードが動作するには、**C1.Silverlight.Data.dll**（**C1Data** クラスを含む）と **System.Windows.Controls.Data.dll**（Microsoft の **DataGrid** コントロールを含む）の名前空間をインポートする必要があります。

DefaultView プロパティは、テーブルに関連付けられている **DataView** オブジェクトを返します。これも、ADO.NET で使用されているメカニズムと同じです。**DataView** オブジェクトは、**IEnumerable** インターフェイスを実装しているため、連結可能コントロールや LINQ クエリーのデータソースとして使用できます。

C1.Silverlight.Data アセンブリ内のクラスは **System.Data** 名前空間内のクラスのサブセットなので、ここでは詳しい説明は省きます。これらのクラスの説明については、MSDN でSystem.Data.DataSet クラスとSystem.Data.DataTable クラスを参照してください。

Copyright © GrapeCity inc. All rights reserved.

データ中心型アーキテクチャ

Silverlight を使用して、事業別などのデータ中心型アプリケーションを構築できます。このようなアプリケーションでは、一般に次の手順が実行されます。

1. サーバーからデータを取得します。
2. クライアント側でデータを表示および編集します。
3. サーバーに変更を送信します。

手順 1と3では、通常、データを転送するために Web サービスが使用され、データベースをクエリーおよび更新するために従来のデータアクセス方法が使用されます。手順 2では、通常、Silverlight データ連結コントロールが使用されます。

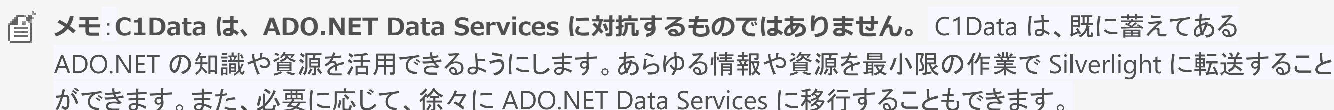
Microsoft は、サーバー側のジョブを実行するために多くのツールを提供しています。最新のツールとして、ADO.NET Data Services があります。これは、Web アクセス可能なエンドポイントをデータモデルに提供し、Silverlight ライブラリ(System.Data.Services.Client.dll)の ADO.NET Data Services を介して Silverlight と統合されます。最近では、この新技術に関する記事が多く掲載されています(MSDN 2008 年 9 月、volume 23、no. 10 の「Data Services」など)。

ADO.NET Data Services は、さまざまなデータ中心型アプリケーションの標準になる可能性が高い強力な新技術です。ただし、この技術が唯一の選択肢というわけではありません。サーバー側でデータを取得および更新するために、従来の ADO.NET データクラス(**DataSet**、**DataTable**、**DataAdapters** など)も使用できます。これらのクラスは、.NET 1.0 以来、開発者に使用されており、堅牢で強力な使いやすいクラスです。さらに、多くの開発者は、長期に渡って使用およびテストされてきたコードとして、これらのクラスに多くの投資を行ってきています。

Data for Silverlight は、従来の ADO.NET を使ってサーバーとデータをやり取りするために Silverlight クライアントから使用できるクラスの一式です。一般的な手順は次のとおりです。

1. **サーバーからデータを取得します。**
   a. サーバーは従来の方法で **DataSet** オブジェクトにデータを挿入します(通常は、**DataAdapter** オブジェクトを使用して、SqlServer データベースからデータを取得します)。
   b. サーバーは、**DataSet.WriteXml** を呼び出して **DataSet** をストリームにシリアライズし、そのストリームをクライアントに送信します。
   c. クライアントは、**DataSet.ReadXml** メソッドを使用して、そのストリームをデシリアライズします。
2. **クライアント側でデータを表示および編集します。**
   a. クライアントは **DataSet** 内のテーブルをコントロールに連結します(通常は LINQ クエリーを使用)。
   b. ユーザーは、コントロールを操作して、データ項目を表示、編集、追加、および削除します。
3. **サーバーに変更を送信します。**
   a. クライアントは、**DataSet.GetChanges** および **DataSet.WriteXml** を呼び出して、変更をストリームにシリアライズし、そのストリームをサーバーに送信します。
   b. サーバーは、**DataSet.ReadXml** を呼び出して変更をデシリアライズし、次に **DataAdapter** オブジェクトを使って変更をデータベースに保存します。

このシナリオにおける **C1Data** の役割は2つあります。1つめは、クライアント側とサーバー側の **DataSet** オブジェクト間のデータ転送を容易にする対称型のシリアライズメカニズムを提供します。2つめは、クライアント側でデータを操作するための使い慣れたオブジェクトモデルを提供します。


メモ:**C1Data は、ADO.NET Data Services に対抗するものではありません。** C1Data は、既に蓄えてある ADO.NET の知識や資源を活用できるようにします。あらゆる情報や資源を最小限の作業で Silverlight に転送することができます。また、必要に応じて、徐々に ADO.NET Data Services に移行することもできます。

C1Data はレガシー技術ではありません。たとえば、**C1Data** は LINQ と共に使用できます。実際、**C1Data** では、デスクトップでは使用できるが、Silverlight アプリケーションでは使用できない LINQ 機能が有効になります(一部匿名クラス)。

以下のセクションでは、上の手順に従う単純なアプリケーションの実装について説明します。このアプリケーションは単純ですが、ほとんどのデータ中心型アプリケーションで必要になる4つの CRUD(Create、Read、Update、Delete)操作の実行方法を示します。

Copyright © GrapeCity inc. All rights reserved.

チュートリアル

このサンプルアプリケーションは、Northwind の Categories テーブルと Products テーブルをマスター/詳細ビューで表示するように配置された2つのグリッドで構成されます。データは、アプリケーションの起動時に Web サービスを使ってサーバーから取得されます。次に、テーブルにデータが挿入され、ユーザーがデータを参照および編集できるようになります。最後に、アプリケーションは、データベースを更新するためにすべての変更をサーバーに送信します。

アプリケーションの作成

まず、「MasterDetail」という名前の新しい Silverlight アプリケーションを作成します。次の手順に従います。

1. Visual Studio で、**[ファイル]**→**[新しいプロジェクト]**を選択し、**[新しいプロジェクト]**ダイアログボックスを開きます。
2. **[プロジェクトの種類]**ペインで、**Visual Basic** または **Visual C#** ノードのいずれかを展開し、**[Silverlight]**を選択します。
3. **[テンプレート]**ペインで、**[Silverlight アプリケーション]**を選択します。
4. プロジェクト名を「MasterDetail」とし、プロジェクトの場所を指定して、**[OK]**をクリックします。
   次に、Visual Studio から、新しいプロジェクトで使用するホスティングのタイプを入力するように指示されます。
5. **[新しい Silverlight アプリケーション]**ダイアログボックスで、**[OK]**を選択してデフォルトの名前（MasterDetailWeb）と設定を受け入れ、プロジェクトを作成します。

参照の追加

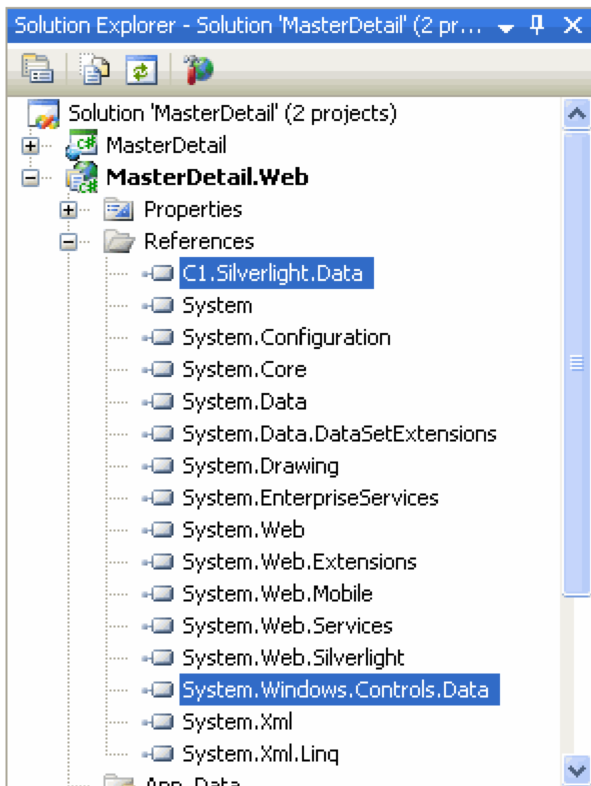
このプロジェクトでは、**C1.Silverlight.Data.dll**（**C1Data** クラスを含む）と **System.Windows.Controls.Data.dll**（Microsoft の **DataGrid** コントロールを含む）の2つの追加アセンブリを使用します。ここでは、**C1Data** クラスが ComponentOne Studio for Silverlight の他のコントロールにまったく関連付けられていないことを示すために、ComponentOne Studio for Silverlight の **DataGrid** コントロールではなく、Microsoft の **DataGrid** コントロールを使用することにします。実際、**C1Data** クラスは任意の Silverlight コントロールに対して使用できます。

アセンブリを追加するには、次の手順に従います。

1. ソリューションエクスプローラで **MasterDetail** プロジェクトを右クリックし、**[参照の追加]**を選択します。
2. **[参照の追加]**ダイアログボックスで、以下のアセンブリを見つけて選択し、**[OK]**をクリックしてプロジェクトに参照を追加します。
   - C1.Silverlight.Data.dll
   - System.Windows.Controls.Data.dll

   参照が追加されると、ソリューションは次のようになります。

Copyright © GrapeCity inc. All rights reserved.



UI の作成

ユーザーインターフェイスは、2つのデータグリッド（Categories と Products）と3つのボタン（項目の追加、項目の削除、変更のコミット）で構成されます。

ユーザーインターフェイスを作成するには、**MainPage.xaml** ファイルを開き、ソースビューで次の XAML をページにコピーします。

```
<UserControl x:Class="MasterDetail.MainPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:swc="clr-
namespace:System.Windows.Controls;assembly=System.Windows.Controls.Data" >
  <Grid x:Name="LayoutRoot" Background="White" >

    <!-- メイングリッドレイアウト -->
    <Grid.RowDefinitions>
      <RowDefinition Height="Auto" />
      <RowDefinition Height="Auto" />
      <RowDefinition Height="*" />
      <RowDefinition Height="Auto" />
      <RowDefinition Height="2*" />
      <RowDefinition Height="Auto" />
    </Grid.RowDefinitions>

    <!-- ページタイトル -->
    <TextBlock Text="Silverlight マスター詳細サンプル" Margin="5"
      FontSize="20" FontWeight="Bold" />
```


Copyright © GrapeCity inc. All rights reserved.

```
    <!-- マスターデータグリッド -->
    <TextBlock Text="カテゴリ" Margin="5" Grid.Row="1"
      FontSize="16" FontWeight="Bold" Foreground="Blue" />
    <swc:DataGrid x:Name="_gridCategories" Margin="5" Grid.Row="2" />

    <!-- 詳細データグリッド -->
    <TextBlock Text="製品" Margin="5" Grid.Row="3"
      FontSize="16" FontWeight="Bold" Foreground="Blue" />
    <swc:DataGrid x:Name="_gridProducts" Margin="5" Grid.Row="4" />

    <!-- コントロールパネル -->
    <StackPanel Margin="5" Orientation="Horizontal" Grid.Row="5" >
      <Button x:Name="_btnAdd" Content=" 項目の追加 "
        Margin="0,0,20,0" FontSize="14" VerticalAlignment="Center" />
      <Button x:Name="_btnRemove" Content=" 項目の削除 "
        Margin="0,0,20,0" FontSize="14" VerticalAlignment="Center" />
      <Button x:Name="_btnCommit" Content=" 変更を反映 "
        Margin="0,0,20,0" FontSize="14" VerticalAlignment="Center" />
      <TextBlock x:Name="_tbStatus" Text=" 準備完了 "
        VerticalAlignment="Center" FontSize="12" Foreground="Gray" />
    </StackPanel>
  </Grid>
</UserControl>
```

これまでの作業に問題がないことを確認するため、ここでアプリケーションを実行してみます。2つの空のグリッドが表示されるはずです。

Silverlight マスター詳細サンプル

カテゴリ

製品

項目の追加 項目の削除 変更を反映 準備完了

サーバー側の実装

この手順では、アプリケーションのサーバー側を実装します。サーバー側は、2つのメソッド（**GetData** と **UpdateData**）を含む1つの Web サービスで構成されます。**GetData** は、Northwind データベースからデータを取得し、そのデータをクライアントに返します。**UpdateData** は、クライアントで行われた変更をデータベースに戻して保存します。

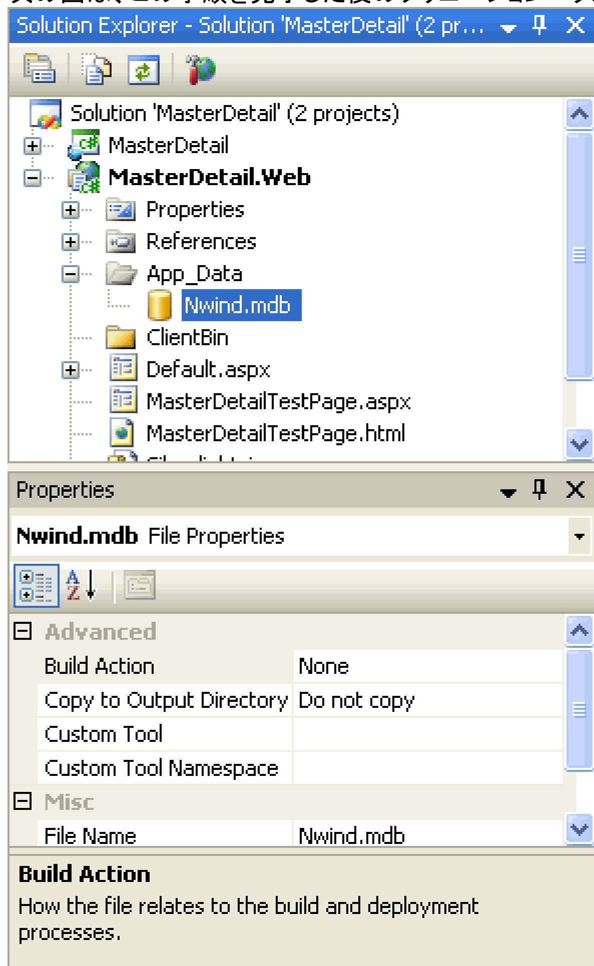
Copyright © GrapeCity inc. All rights reserved.

データベースアクセスインフラストラクチャの追加

Web サービスを実装する前に、データベース自体をプロジェクトに追加します。この手順はオプションです。接続文字列を使ってデータベースの場所を指定するため、この MDB ファイルはシステム内のどこにあってもかまいません。ここでは、展開を容易にするためと、オリジナルのデータベースが変更されないようにするために、ローカルコピーを作成します。

データベースをプロジェクトに追加するには、次の手順に従います。

1. **MasterDetailWeb** プロジェクト内の **App_Data** ノードを右クリックし、**[追加]**→**[既存の項目]**を選択します。
2. ダイアログボックスで **NWind.mdb** ファイルを見つけ、そのファイルをプロジェクトに追加します。
3. **[プロパティ]**ウィンドウで、**[ビルドアクション]**プロパティを**[なし]**に設定します。
   次の図は、この手順を完了した後のソリューションエクスプローラウィンドウです。



データベースファイルに加えて、データベースとの間でデータをやり取りするためのメカニズムも必要です。このサンプルでは、**SmartDataSet** という名前のユーティリティクラスを使ってこのメカニズムを作成します。**SmartDataSet** は、標準の ADO.NET **DataSet** クラスを拡張して、次の項目を追加しています。

- データベースの種類と場所を指定する **ConnectionString** プロパティ。
- 選択されたテーブルからデータをロードする **Load** メソッド。
- 変更をデータベースに戻して保存する **Update** メソッド。

SmartDataSet は便利ですが、必須ではありません。このクラスは、データのロードと保存に使用される **DataAdapter** オブジェクトを内部的に作成および設定しますが、標準の ADO.NET コードを記述して同じ処理を行うこともできます。その場合は、標準の ADO.NET 技術のみを使用するため、ここでは省略します。

SmartDataSet.cs ファイルをプロジェクトに追加するには、次の手順に従います。

1. ソリューションエクスプローラで **MasterDetailWeb** ノードを右クリックし、**[追加]**→**[既存の項目]**を選択します。
2. **[既存のアイテムの追加]**ダイアログボックスで、**C1.Silverlight** 配布パッケージ内の **SmartDataSet.cs** ファイルを見

Copyright © GrapeCity inc. All rights reserved.

つけ、**[追加]**をクリックしてプロジェクトに追加します。

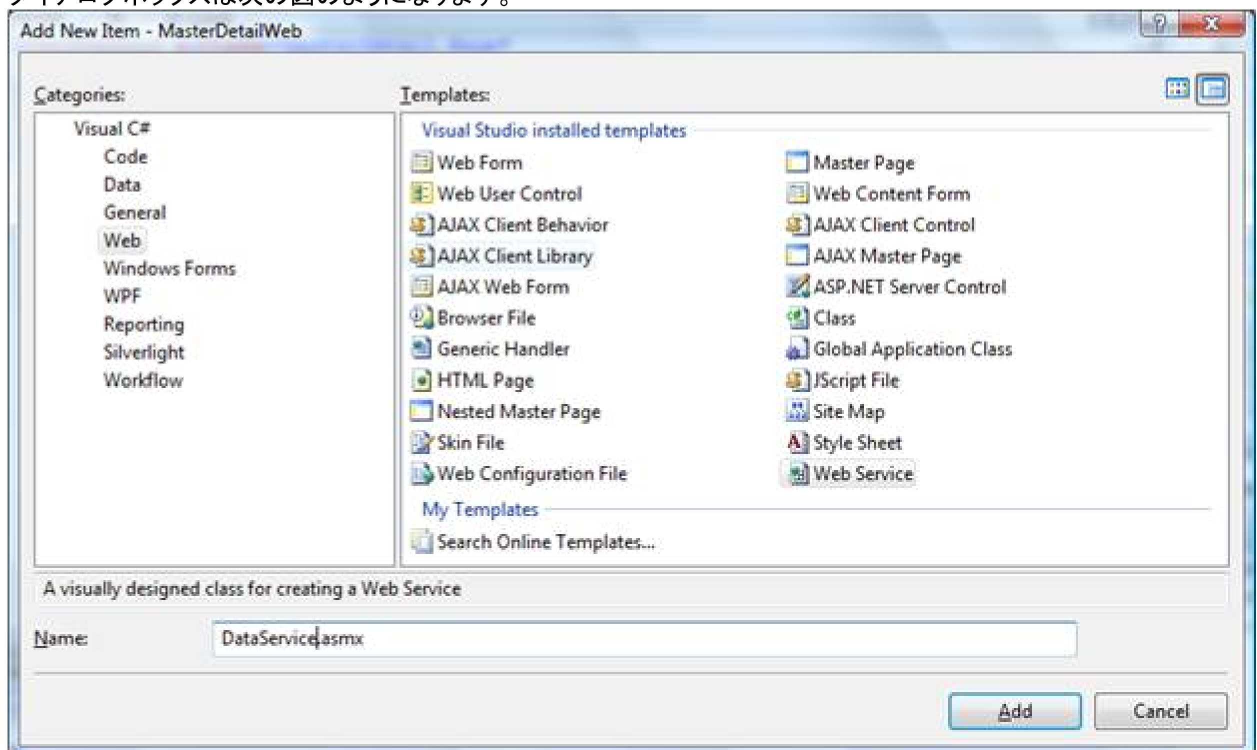
Web サービスの実装

アプリケーションのサーバー側は、データベースからデータをロードし、それをサーバーに返したり、クライアントから変更のリストを受け取り、それをデータベースに適用する Web サービスで構成されます。

サーバー側は、汎用のハンドラクラス（ashx）、Web サービス（asmx）、または Silverlight 対応の WCF サービス（SVC）として実装できます。このサンプルではどれでも実装できますが、ここでは標準的な Web サービスを選択します。

サービスを作成するには、次の手順に従います。

1. **MasterDetailWeb** プロジェクトを右クリックします。
2. **[追加]**→**[新しい項目]**を選択します。
3. **[新しい項目の追加]**ダイアログボックスで、左ペインの[カテゴリ]リストから**[Web]**を選択します。
4. 右ペインの[テンプレート]リストから**[Web サービス]**テンプレートを選択します。
5. 新しいサービス名として「DataService.asmx」を指定します。
   ダイアログボックスは次の図のようになります。



6. **[追加]**ボタンをクリックして、新しい Web サービスを作成します。

[追加]ボタンをクリックすると、新しく作成された **DataService.asmx.cs**（または **DataService.asmx.vb**）ファイルが Visual Studio で開かれます。このファイルには、1つの **HelloWorld** メソッドのみがあります。

このファイルを次のように編集します。

1. ファイルの先頭にある文ブロックに、次の using 文を追加します。

```
using System.IO;
using System.Data;
```

2. Silverlight からサービスを呼び出せるように、次の行のコメントを外します。

```
[System.Web.Script.Services.ScriptService]
```

3. Visual Studio が作成した **HelloWorld** メソッドを削除し、次のコードに置き換えます。

```
[WebMethod]
```


Copyright © GrapeCity inc. All rights reserved.

```
public byte[] GetData(string tables)
{
  // 接続文字列を使って DataSet を作成します
  var ds = GetDataSet();

  // DataSet にデータをロードします
  ds.Fill(tables.Split(','));

  // ストリームに保存します
  var ms = new System.IO.MemoryStream();
  ds.WriteXml(ms, XmlWriteMode.WriteSchema);

  // ストリームデータを返します
  return ms.ToArray();
}
```

このメソッドでは、最初に **SmartDataSet** を作成し、そこに **tables** パラメータで指定されたテーブルのデータを挿入します。その後、**WriteXml** メソッドを使って **DataSet** をストリームに保存し、そのストリームをバイト配列に変換して、その結果を返します。

このコードは、サーバー側で実行されることを思い出してください。**C1.Zip** などのデータ圧縮ライブラリを使ってストリームを圧縮し、ストリームのサイズを大幅に縮小してからクライアントに返すこともできます。ただし、ここでは、サンプルをできるだけ単純にするため、あえて圧縮しませんでした。

4. 次に、以下のコードを使用して、変更をデータベースに保存し直すメソッドを追加します。

```
[WebMethod]
public string UpdateData(byte[] dtAdded, byte[] dtModified, byte[] dtDeleted)
{
  try
  {
    UpdateData(dtAdded, DataRowState.Added);
    UpdateData(dtModified, DataRowState.Modified);
    UpdateData(dtDeleted, DataRowState.Deleted);
    return null;
  }
  catch (Exception x)
  {
    return x.Message;
  }
}
```

このメソッドは、3つのパラメータを受け取ります。各パラメータは、それぞれ異なるタイプの変更（レコードの追加、変更、削除）をデータベースに適用するために使用されます。このメソッドは、**UpdateData** ヘルパーメソッドを呼び出してそれぞれの変更セットを適用し、すべての変更が正しく適用された場合に null を返します。エラーが発生した場合は、例外について説明したメッセージを返します。

5. UpdateData ヘルパーメソッドに以下のコードを追加します。

```
void UpdateData(byte[] data, DataRowState state)
{
  // 変更がない場合は何もしません
  if (data == null)
    return;
```


Copyright © GrapeCity inc. All rights reserved.

```
  // DataSet にデータをロードします
  var ds = GetDataSet();
  var ms = new MemoryStream(data);
  ds.ReadXml(ms);
  ds.AcceptChanges();

  // 変更された行の状態を更新します
  foreach (DataTable dt in ds.Tables)
  {
    foreach (DataRow dr in dt.Rows)
    {
      switch (state)
      {
        case DataRowState.Added:
          dr.SetAdded();
          break;
        case DataRowState.Modified:
          dr.SetModified();
          break;
        case DataRowState.Deleted:
          dr.Delete();
          break;
      }
    }
  }

  // データベースを更新します
  ds.Update();
}
```

このメソッドは、最初に **SmartDataSet** を作成し、そこにすべての変更をロードします。次に、各行の **RowState** プロパティを変更して、行に適用された変更のタイプ（追加、変更、削除）を識別します。最後に、**SmartDataSet.Update** メソッドを呼び出して、変更をデータベースに書き込みます。

6. これで、サーバー側のコードは大部分準備できました。後は、**SmartDataSet** を作成して設定するメソッドを追加するだけです。この実装に以下のコードを追加します。

```
SmartDataSet GetDataSet()
{
  // mdb ファイルの物理的な場所を取得します
  string mdb = Path.Combine(
    Context.Request.PhysicalApplicationPath, @"App_Data\NWind.mdb");

  // このファイルの存在を確認します
  if (!File.Exists(mdb))
  {
    string msg = string.Format("Cannot find database file {0}.", mdb);
    throw new FileNotFoundException(msg);
  }

  // ファイルが読み取り専用でないことを確認します（ソースコントロールによって読み取り専用にされる場合があります）
  FileAttributes att = File.GetAttributes(mdb);
```


Copyright © GrapeCity inc. All rights reserved.

```
  if ((att & FileAttributes.ReadOnly) != 0)
  {
    att &= ~FileAttributes.ReadOnly;
    File.SetAttributes(mdb, att);
  }

  // SmartDataSet を作成および初期化します
  var dataSet = new SmartDataSet();
  dataSet.ConnectionString =
    "provider=microsoft.jet.oledb.4.0;data source=" + mdb;
  return dataSet;
}
```

このメソッドは、最初にデータベースファイルを見つけ、その存在を確認し、そのファイルが読み取り専用でないことを確認します（読み取り専用である場合は、更新が失敗します）。その後、新しい **SmartDataSet** を作成し、**ConnectionString** プロパティを初期化して、新しく作成した **SmartDataSet** を呼び出し元に返します。

クライアント側の実装

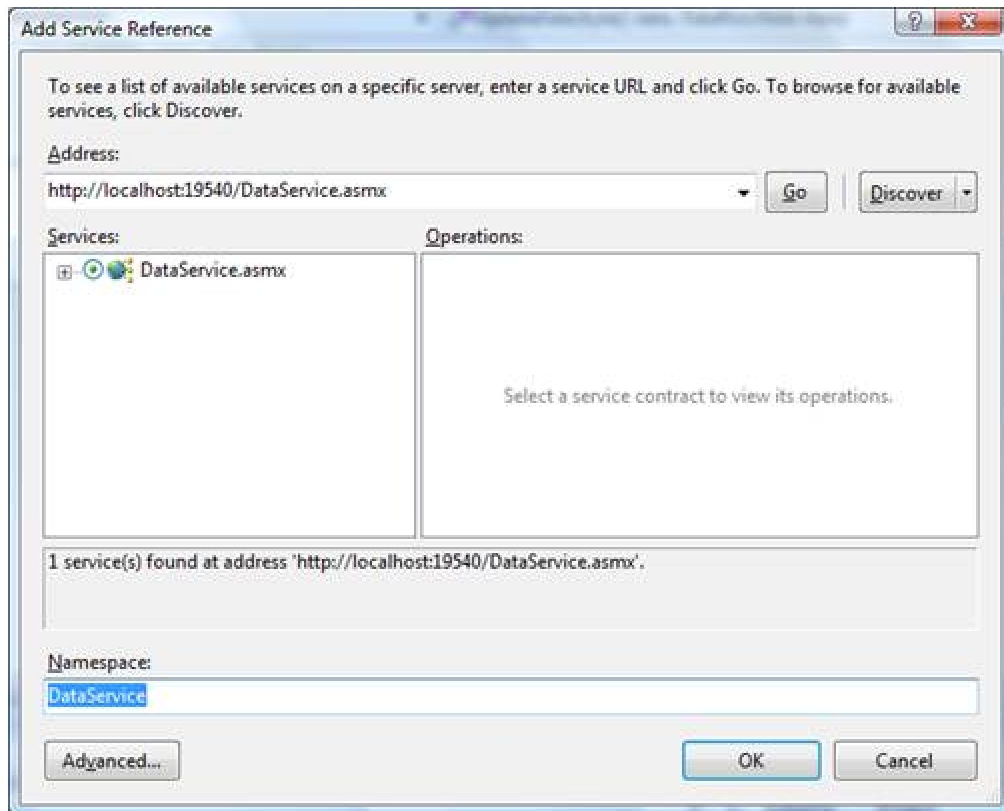
アプリケーションのクライアント側に戻ります。以下のセクションでは、クライアント側を実装します。

Web サービスへの参照の追加

まず、クライアントから Web サービスを呼び出すことができるように、先ほど作成した Web サービスへの参照を追加します。次の手順に従います。

1. **MasterDetail** プロジェクト内の **[参照設定]** ノードを右クリックし、**[サービス参照の追加]**を選択します。
2. 表示されるダイアログボックスで、**[探索]**ボタンをクリックします。
   前の手順で追加したサービスが Visual Studio によって検出されます。
3. **[名前空間]**フィールドに、「DataService」と入力します。
   ダイアログボックスは次の図のようになります。

Copyright © GrapeCity inc. All rights reserved.



4. **[OK]**をクリックして参照を追加します。

Web サービスを使用したデータの取得

いよいよデータを取得して、ユーザーに表示します。次の手順に従います。

1. **MainPage.xaml.cs** ファイルを開き、ファイルの先頭にあるブロックに次の using 文を追加します。

```
using System.IO;
using C1.Silverlight.Data;
using MasterDetail.DataService;
```

次に、Page コンストラクタを以下のように変更します。

```
public Page()
{
  InitializeComponent();
  LoadData();
}
```

2. 次のコードを追加して **LoadData** メソッドを実装します。このメソッドは、Web サービスを呼び出してデータを取得し、それを **DataSet** オブジェクトに格納します。次に、このオブジェクトをページ上のコントロールに連結します。

```
DataSet _ds = null;
void LoadData()
{
  // Web サービスを呼び出します
  var svc = new GetDataService();
  svc.GetDataCompleted += svc_GetDataCompleted;
  svc.GetDataAsync("Categories,Products");
}
```


Copyright © GrapeCity inc. All rights reserved.

```
void svc_GetDataCompleted(object sender, GetDataCompletedEventArgs e)
{
  // エラーを処理します
  if (e.Error != null)
  {
    _tbStatus.Text = "データのダウンロードに失敗しました。";
    return;
  }

  // サーバーからのデータストリームを解析します(XML から DataSetを取得)
  _tbStatus.Text = string.Format(
    "データ取得中:{0:n0} KB", e.Result.Length / 1024);
  var ms = new MemoryStream(e.Result);
  _ds = new DataSet();
  _ds.ReadXml(ms);

  // データを取得し、コントロールをデータに連結します
  BindData();
}
```

3. 次の **GetDataService** メソッドの実装を追加します。

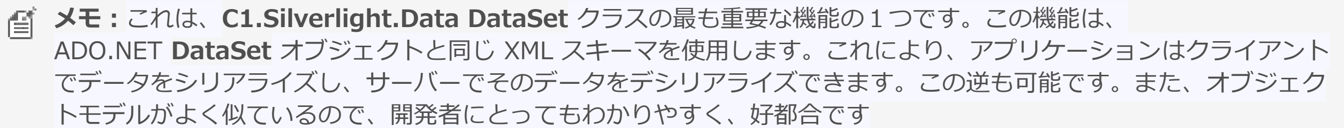
```
// 現在のホスト/ドメインに関連するデータサービスを取得します
DataServiceSoapClient GetDataService()
{
  // バッファサイズを増やします
  var binding = new System.ServiceModel.BasicHttpBinding();
  binding.MaxReceivedMessageSize = 2147483647; // int.MaxValue
  binding.MaxBufferSize = 2147483647; // int.MaxValue

  // サービスの絶対アドレスを取得します
  Uri uri = GetAbsoluteUri("DataService.asmx");
  var address = new System.ServiceModel.EndpointAddress(uri);

  // 新しいサービスクライアントを返します
  return new DataServiceSoapClient(binding, address);
}

public static Uri GetAbsoluteUri(string relativeUri)
{
  Uri uri = System.Windows.Browser.HtmlPage.Document.DocumentUri;
  string uriString = uri.AbsoluteUri;
  int ls = uriString.LastIndexOf('/');
  return new Uri(uriString.Substring(0, ls + 1) + relativeUri);
}
```

GetDataService メソッドは、新しい **DataServiceSoapClient** オブジェクトをインスタンス化して返します。デフォルトコンストラクタは開発環境(http://localhost など)を参照するため、ここでは使用しません。デフォルトコンストラクタは開発マシンでは正常に機能しても、アプリケーションの展開時には機能しません。また、デフォルトコンストラクタは 65 KB バッファを使用しているため、データ転送には小さすぎます。上の **GetDataService** メソッドの実装は、この2つの問題を解決します。

上の **LoadData** メソッドは、サービスをインスタンス化し、**GetDataAsync** メソッドを呼び出します。このメソッドは、実行の終了時に **svc_DataCompleted** デリゲートを呼び出します。このデリゲートは、**DataSet** オブジェクトをインスタンス化

Copyright © GrapeCity inc. All rights reserved.

し、**ReadXml** メソッドを使用して、サーバーから提供されたデータをデシリアライズします。次に、**BindData** を呼び出して、ページ上のコントロールにデータを連結します。

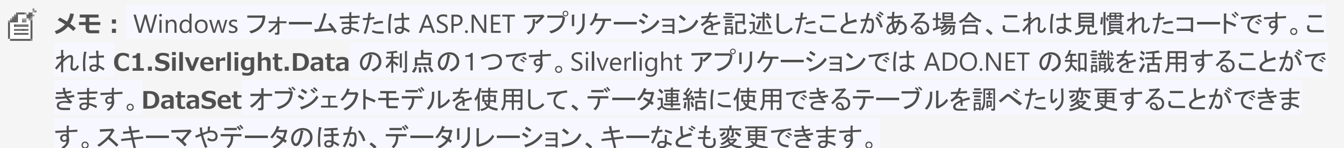


メモ：これは、**C1.Silverlight.Data DataSet** クラスの最も重要な機能の 1 つです。この機能は、ADO.NET **DataSet** オブジェクトと同じ XML スキーマを使用します。これにより、アプリケーションはクライアントでデータをシリアライズし、サーバーでそのデータをデシリアライズできます。この逆も可能です。また、オブジェクトモデルがよく似ているので、開発者にとってもわかりやすく、好都合です

ページ上のコントロールへのデータ連結

データを取得したら、そのデータをページ上のコントロールに連結する必要があります。次は、このタスクを処理する **BindData** メソッドの実装です。

```
void BindData()
{
  // 必要なテーブルを取得します
  DataTable dtCategories = _ds.Tables["Categories"];
  DataTable dtProducts = _ds.Tables["Products"];

  // カテゴリグリッドにデータを挿入します
  _gridCategories.ItemsSource =
      new DataView(dtCategories,
          "カテゴリID", "カテゴリ名", "説明");

  // 製品グリッドにデータを挿入します
  _gridProducts.ItemsSource = dtProducts.DefaultView;
}
```

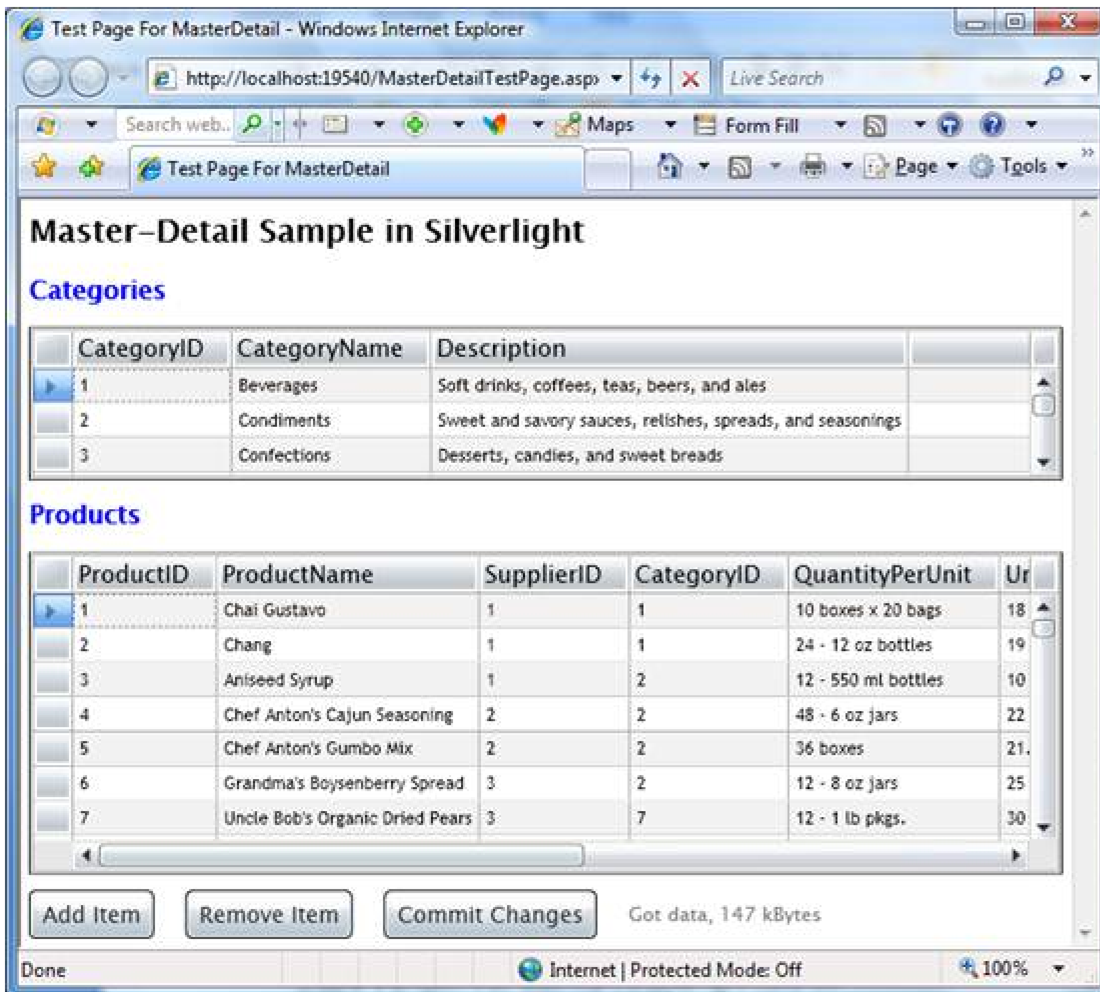


メモ： Windows フォームまたは ASP.NET アプリケーションを記述したことがある場合、これは見慣れたコードです。これは **C1.Silverlight.Data** の利点の1つです。Silverlight アプリケーションでは ADO.NET の知識を活用することができます。**DataSet** オブジェクトモデルを使用して、データ連結に使用できるテーブルを調べたり変更することができます。スキーマやデータのほか、データリレーション、キーなども変更できます。

ただし、**DataView** オブジェクトを作成し、どの列をビューに入れるかを指定する文は、あまり一般的ではありません。これは、C1.Silverlight.Data の実装によって提供される拡張機能です。オリジナルの ADO.NET **DataView** クラスにはありません。

ここでプロジェクトを実行すると、サーバーからデータが取得されることを確認できます。

Copyright © GrapeCity inc. All rights reserved.



サンプルを続行する前に、いくつかの点を確認します。

データをコントロールに連結するコードはおなじみですが、実際に裏で実行されることは、従来の Windows フォームや ASP.NET のシナリオとはかなり異なります。WPF や Silverlight のデータ連結は、すべてリフレクションを使って実行されます。ただし、**DataView** コレクションによって公開される **DataRowView** オブジェクトには、基底の **DataTable** 内の列（カテゴリ ID、カテゴリ名、説明 など）に対応するプロパティはありません。

> **メモ：DataView** クラスは、**DataRowView** から派生される匿名タイプを動的に作成し、テーブル列に対応する追加プロパティを公開するため、連結が可能になります。次に、コントロールはリフレクションを使用して、これらのプロパティを検索し、連結します。

DataRow クラスも **GetRowView** メソッドを介してこのメカニズムを活用します。このメソッドは、**DataRow** の選択されたプロパティを公開し、データ連結に使用できるラッパーオブジェクトを作成します。このメソッドは、LINQ クエリー内で使用できます。

たとえば、次のコードは、「C」で始まるすべてのカテゴリを選択し、それらの名前と説明を公開する LINQ クエリーを作成します。

```
_gridCategories.ItemsSource =
    from dr
    in dtCategories.Rows
    where ((string)dr["カテゴリ名"]).StartsWith("C")
    select dr.GetRowView("カテゴリ名", "説明");
```

パラメータなしで **GetRowView** を呼び出すと、すべての列が公開されます。

Copyright © GrapeCity inc. All rights reserved.

ビューの同期

マスター/詳細ビューはまだ同期されていません。下のグリッドには、上のグリッドで選択されているカテゴリには関係なく、すべての製品が表示されます。

ビューを同期するには、カテゴリグリッドで **SelectionChanged** イベントを処理し、選択されたカテゴリに属する製品のみが表示されるように製品をフィルタする必要があります。

同期するには、最初に、Page コンストラクタにイベントハンドラを追加します。

```
public Page()
{
  InitializeComponent();
  LoadData();
  _gridCategories.SelectionChanged += _gridCategories_SelectionChanged;
}
```

このイベントハンドラは次のように実装されます。

```
void _gridCategories_SelectionChanged(object sender, EventArgs e)
{
  var drv = _gridCategories.SelectedItem as DataRowView;
  if (drv != null)
  {
    var dv = _ds.Tables["Products"].DefaultView;
    dv.RowFilter = string.Format("CategoryID = {0}", drv.GetData("CategoryID"));
  }
}
```

このコードは、選択されたカテゴリを **DataRowView** オブジェクトとして取得し、"Products" テーブル(2つめのグリッドのデータソースとして使用)のデフォルトビューを取得し、フィルタを適用します。このコードも、Windows フォームや ASP.NET の開発者にはおなじみです。

再度、アプリケーションを実行すると、ビューが同期されていることを確認できます。

DataView にフィルタを適用する代わりに、LINQ クエリーを使って Products テーブルを更新することもできます。次に例を挙げます。

```
void _gridCategories_SelectionChanged(object sender, EventArgs e)
{
  var drv = _gridCategories.SelectedItem as DataRowView;
  if (drv != null)
  {
    int id = (int)drv.GetData("CategoryID");
    _gridProducts.ItemsSource =
      from dr
      in _ds.Tables["Products"].Rows
      where (int)dr["CategoryID"] == id
      select dr.GetRowView();
  }
}
```

このコードでも、マスタービューと詳細ビューを同期できます。前の方法では、同じビューに連結されている他のすべてのコントロールに影響しますが、この方法では、製品グリッドにのみ影響する点が異なります。

Copyright © GrapeCity inc. All rights reserved.

項目の追加および削除

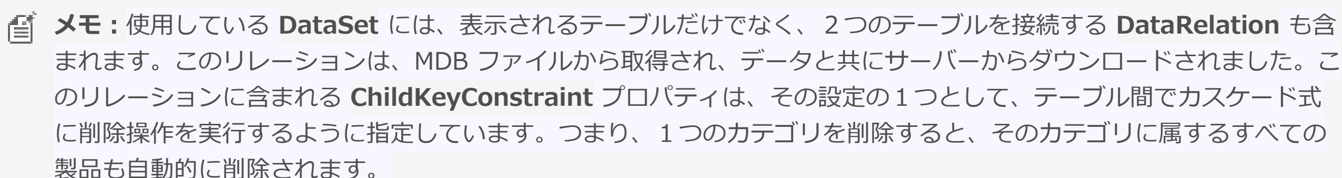
ほとんどの Windows フォームグリッドとは違い、Silverlight の **DataGrid** コントロールでは、ユーザーがグリッドから項目を追加したり削除することはできません。そのために、ここではページ上に既に存在するボタンを使用します。最初に、Page コンストラクタでイベントハンドラを接続します。

```
// データをロードし、イベントハンドラを登録します
public Page()
{
  InitializeComponent();
  LoadData();

  _gridCategories.SelectionChanged += _gridCategories_SelectionChanged;
  _btnAdd.Click += _btnAdd_Click;
  _btnRemove.Click += _btnRemove_Click;
}
```

これらのイベントハンドラは単純で、Windows フォームアプリケーションで記述する標準的な ADO.NET コードに似ています。

```
// 新しい行を追加します
private void _btnAdd_Click(object sender, RoutedEventArgs e)
{
  DataTable dt = _ds.Tables["Categories"];
  DataRow newRow = dt.NewRow();
  newRow["CategoryName"] = "New category";
  newRow["Description"] = "This is a new category...";
  dt.Rows.Add(newRow);
}

// 行を削除します
private void _btnRemove_Click(object sender, RoutedEventArgs e)
{
  DataRowView drv = _gridCategories.SelectedItem as DataRowView;
  if (drv != null)
  {
    DataRow dr = drv.GetRow();
    dr.Delete();
  }
}
```

ここでアプリケーションを実行すると、グリッドに表示されている項目を追加、削除、および変更できるようになります。


メモ：使用している **DataSet** には、表示されるテーブルだけでなく、２つのテーブルを接続する **DataRelation** も含まれます。このリレーションは、MDB ファイルから取得され、データと共にサーバーからダウンロードされました。このリレーションに含まれる **ChildKeyConstraint** プロパティは、その設定の１つとして、テーブル間でカスケード式に削除操作を実行するように指定しています。つまり、１つのカテゴリを削除すると、そのカテゴリに属するすべての製品も自動的に削除されます。

サーバーへの変更のコミットサーバーへの変更のコミット

Copyright © GrapeCity inc. All rights reserved.

ほとんど完成に近づきました。この後は、ユーザーが行った変更がデータベースに適用されるように、変更をサーバーに送信するコードを追加します。

最初の手順は、Page コンストラクタをもう一度変更し、[Commit Changes]ボタンにイベントハンドラを接続します。

```
// データをロードし、イベントハンドラを登録します
public Page()
{
  InitializeComponent();
  LoadData();

  _gridCategories.SelectionChanged += _gridCategories_SelectionChanged;
  _btnAdd.Click += _btnAdd_Click;
  _btnRemove.Click += _btnRemove_Click;
  _btnCommit.Click += _btnCommit_Click;
}
```

次は、イベントハンドラの実装です。

```
// 変更をサーバーにコミットします
private void _btnCommit_Click(object sender, RoutedEventArgs e)
{
  SaveData();
}
```

次は、実際の作業を行う **SaveData** メソッドの実装です。

```
// データベースにデータを戻して保存します
void SaveData()
{
  if (_ds != null)
  {
    // 各タイプの変更を取得します
    byte[] dtAdded = GetChanges(DataRowState.Added);
    byte[] dtModified = GetChanges(DataRowState.Modified);
    byte[] dtDeleted = GetChanges(DataRowState.Deleted);

    // サービスを呼び出します
    var svc = new GetDataService();
    svc.UpdateDataCompleted += svc_UpdateDataCompleted;
    svc.UpdateDataAsync(dtAdded, dtModified, dtDeleted);
  }
}

void svc_UpdateDataCompleted(object sender, UpdateDataCompletedEventArgs e)
{
  if (!string.IsNullOrEmpty(e.Result))
  {
    throw new Exception("Error updating data on the server: " + e.Result);
  }
  _tbStatus.Text = "変更がサーバーに反映されました。";
  _ds.AcceptChanges();
```


Copyright © GrapeCity inc. All rights reserved.

```
}
```

このメソッドは、最初に **GetChanges** メソッドを呼び出して3つのバイト配列を作成します。各バイト配列は、サーバーからデータがダウンロードされてから、追加、変更、または削除された行を含む **DataSet** を表します。次に、前に実装した Web サービスを呼び出して、その結果を待ちます。サーバーの更新中にエラーが検出された場合は、例外をスローします（実際のアプリケーションでは、エラーはもう少し丁寧に処理されます）。

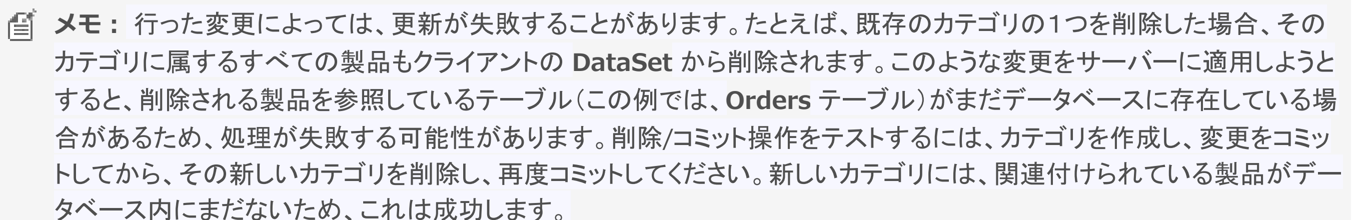
残りは **GetChanges** メソッドです。次にコードを示します。

```
byte[] GetChanges(DataRowState state)
{
    DataSet ds = _ds.GetChanges(state);
    if (ds != null)
    {
        MemoryStream ms = new MemoryStream();
        ds.WriteXml(ms);
        return ms.ToArray();
    }
    return null;
}
```

このメソッドは、**DataSet.GetChanges** メソッドを使用して、呼び出し元で指定された **DataRowState** を持つ行のみを含む新しい **DataSet** オブジェクトを取得します。このメソッドは、ADO.NET DataSet クラスにあるメソッドと同じです。

次に、このメソッドは、変更を含む **DataSet** を **MemoryStream** にシリアライズし、ストリームコンテンツをバイト配列として返します。

ここでアプリケーションを実行してみてください。いくつかの変更を行ったら、[Commit Changes]ボタンをクリックして、サーバーに変更を送信します。アプリケーションを終了して再起動すると、実際に、それらの変更がデータベースに保存されていることを確認できます。



メモ： 行った変更によっては、更新が失敗することがあります。たとえば、既存のカテゴリの１つを削除した場合、そのカテゴリに属するすべての製品もクライアントの **DataSet** から削除されます。このような変更をサーバーに適用しようとすると、削除される製品を参照しているテーブル（この例では、**Orders** テーブル）がまだデータベースに存在している場合があるため、処理が失敗する可能性があります。削除/コミット操作をテストするには、カテゴリを作成し、変更をコミットしてから、その新しいカテゴリを削除し、再度コミットしてください。新しいカテゴリには、関連付けられている製品がデータベース内にまだないため、これは成功します。

実際のアプリケーションでは、このような問題を精巧に処理する必要があります。たとえば、**Orders** テーブルもロードして、**DataSet** を調査し、削除できない項目をユーザーが削除しようとしていないかどうかを検出して、警告を出すこともできます。これは、読者の練習課題として残しておきます。

データ転送の最適化

サンプルアプリケーションが準備できたので、さらに一歩進み、クライアントとサーバーの間のデータストリームを圧縮してパフォーマンスを最適化します。それには、プロジェクトに2つの小さな変更を加える必要があります。

サーバー側でのデータの圧縮

データをクライアントに送信する前に、**C1.Zip** ライブラリ（ComponentOne Studio に同梱されているデスクトップバージョンです。Silverlight バージョンではありません）を使用して、ストリームを圧縮します。圧縮するには、C1.Zip ライブラリへの参照を追加して **MasterDetailWeb** プロジェクトを変更し、次のように **DataService.asmx.cs Web** サービスの **GetData** メソッドを変更します。

Copyright © GrapeCity inc. All rights reserved.

```
[WebMethod]
public byte[] GetData(string tables)
{
  // 接続文字列を使って DataSet を作成します
  var ds = GetDataSet();

  // DataSet にデータをロードします
  ds.Fill(tables.Split(','));

  // ストリームに保存します
  var ms = new MemoryStream();
  using (var sw = new C1.C1Zip.C1ZStreamWriter(ms))
    ds.WriteXml(sw, XmlWriteMode.WriteSchema);
  //ds.WriteXml(ms, XmlWriteMode.WriteSchema);

  // ストリームデータを返します
  return ms.ToArray();
}
```

クライアントにデータを送信する前にサーバー側でデータを圧縮するために必要な作業はこれだけです。ここでは、**DataSet** をストリームに直接保存するのではなく、ストリームにデータを書き込む前にデータを圧縮する **C1ZStreamWriter** を使ってデータを保存しています。データは XML としてエンコードされるため、圧縮率は高くなります。

クライアント側でのデータの圧縮解除

クライアント側でデータを圧縮解除するには、**C1.Zip** ライブラリ(Silverlight バージョン)への参照を追加して **MasterDetail** プロジェクトを変更し、次のように **MainPage.xaml.cs** ファイルの **svc_GetDataCompleted** メソッドを変更します。

```
void svc_GetDataCompleted(object sender, GetDataCompletedEventArgs e)
{
  // エラーを処理します
  if (e.Error != null)
  {
    _tbStatus.Text = "データのダウンロードに失敗しました。";
    return;
  }

  // サーバーからのデータストリームを解析します(DataSet は XML)
  _tbStatus.Text = string.Format(
    "データ取得中:{0:n0} KB", e.Result.Length / 1024);
  var ms = new MemoryStream(e.Result);
  _ds = new DataSet();
  using (var zr = new C1.C1Zip.C1ZStreamReader(ms))
    _ds.ReadXml(zr);
  //_ds.ReadXml(ms);

  // データを取得し、コントロールをデータに連結します
  BindData();
}
```

ここでは、圧縮されたストリームから **DataSet** を直接読み込むのではなく、データを自動的に圧縮解除する

Copyright © GrapeCity inc. All rights reserved.

C1ZStreamReader を使って読み込みます。

メモ：圧縮しない場合、このアプリケーションは起動時に約 150k バイトのデータを転送します。これらの小さな変更を加えた後は、初期のデータ転送は 50 KB 未満に減少します。この 2 つの小さな変更により、転送されるデータ量は 2/3 ほど減少します。これにより、ネットワークトラフィックが減少し、アプリケーションがデータを初期化して表示するまでの時間が短縮されます。

Copyright © GrapeCity inc. All rights reserved.

スケジュールでのデータの使用

Data for Silverlight を **Schedule for Silverlight** に連結することができます。詳細なサンプルについては、**ComponentOne Studio for Silverlight** にインストールされている **C1Scheduler_DataBinding** サンプルを参照してください。

このサンプルは、MasterDetail サンプルに似ています。MasterDetail サンプルと比べた場合、**C1Scheduler_DataBinding サンプルはいくつかの点が異なります。**

- このサンプルは、**Studio for WPF** にデフォルトでインストールされる **Schedule.mdb** データベースを使用します。
- このサンプルは、サーバーから **Appointments** テーブルをロードします。
- UI は、主に **C1Scheduler** コントロールと追加のナビゲーションコントロールによって表されます。
- データは、すべてのプラットフォームと同じ方法で **C1Scheduler** コントロールに連結されます。

```
// AppointmentStorage のマッピングを設定します
// マッピング名を Appointment テーブル列の名前に設定します
AppointmentMappingCollection mappings =
sched1.DataStorage.AppointmentStorage.Mappings;
mappings.IdMapping.MappingName = "Id";
mappings.Subject.MappingName = "Subject";
mappings.Body.MappingName = "Body";
mappings.End.MappingName = "End";
mappings.Start.MappingName = "Start";
mappings.Location.MappingName = "Location";
mappings.AppointmentProperties.MappingName = "Properties";
...
// C1Scheduler データソースをサーバーからロードされた DataTable に設定します
sched1.DataStorage.AppointmentStorage.DataSource = _dtAppointments;
```

- ユーザーが予定を追加、削除、または編集すると、**C1Scheduler** によってすべての変更が基底の DataTable に自動的に継承されます。コードは必要ありません。

Copyright © GrapeCity inc. All rights reserved.

まとめ

使い慣れたツールと技術を使用して、データ中心型 Silverlight アプリケーションを構築できるようになりました。ADO.NET 開発者は、新しいデータ技術を導入するにあたって、すべての知識、ツール、および既存コードを捨て去る必要はありません。これらの新技術の多くは、既存のツールやパターンに代わるものではなく、それらの拡張です。

C1.Silverlight.Data は、ADO.NET クラスの主要部分に Silverlight 実装を提供します。これにより、開発者にとって次の利点があります。

- ADO.NET の知識と ADO.NET が提供する機能豊富なオブジェクトモデルを活用できると共に、LINQ、WCF などの新しい技術の利点を享受できます。
- クライアントとサーバーの間でリレーショナルデータを簡単に効率よく転送できます。
- クライアント側とサーバー側で似たクラスやオブジェクトモデルを使ってコードを記述できます。サーバーとクライアントは、同じ言語、似たクラス、および似たオブジェクトモデルを使用できます。

Copyright © GrapeCity inc. All rights reserved.