ハンズオン ラボ

WPF の活用 ～ DataGrid ～

目次

演習: WPF の活用 ～ DataGrid ～

Windows Presentation Foundation の次期バージョンでは、表現力の優れたデスクトップ アプリケーションを構築するために利用できる、完全に新しい一連のコントロールが提供される予定です。この演習では、今後の WPF に導入される予定である DataGrid コントロールについて確認します。

現時点 (2009 年 8 月現在) では、DataGrid コントロールは、WPF Toolkit - June 2009 Release に含まれるコントロールとして提供されています (入手方法は後述)。WPF Toolkit は、.NET Framework の WPF に基づく拡張実装です。この WPF Toolkit は、Microsoft がホスティングするオープン ソースのサイトである CodePlex に公開されており、通常の .NET Framework の出荷サイクルとは別にリリースされます。WPF Toolkit で提供される機能のいくつかは、顧客のフィードバックなどに基づいて、 .NET Framework の次期バージョンに採用される場合があります。

現在の WPF Toolkit – June 2009 Release は、.NET Framework 3.5 SP1 に対応しており、この演習では、Visual Studio 2008 SP1 と .NET Framework 3.5 SP 1 の環境のもとで、WPF Toolkit に含まれる DataGrid コントロールを使用します。

特にここでは、従来の ListView コントロールを使用している既存のアプリケーションを修正し、DataGrid コントロールを利用するように変更することで、この新しいコントロールの機能を確認します。

前提知識

この演習のドキュメントでは、次に挙げる知識を既にお持ちであることを前提に解説しています。ただし、XAML の構文等に自信のない方でも、この演習付属のサンプルの完成品をコンパイルし、実行することで、DataGrid が提供する機能を体感することができます。

- Visual Studio 2008 (Visual C# 2008) における基本的な操作。たとえば、ソリューションを開く方法、エディタでの編集、ビルド (コンパイル) や実行の方法、WPF アプリケーションの XAML ビューの開き方など。
- XAML の基本的な構文やその特徴。
- WPF アプリケーションの基本的な作成手順や特徴。

演習のシステム要件

この演習を行うには、あらかじめ以下の環境を用意する必要があります。

- Windows 7 (日本語 32 ビット版)
- Visual C# 2008 SP1 (Visual Studio 2008 SP1) 日本語版、Express Edition も可能
- WPF Toolkit – June 2008 Release (DataGrid コントロールのために必要)
- この演習で使用する付属のソース プログラム (サンプルプログラム)

この後は、上記の各項について、留意点をいつくか補足します。予めご覧ください。

ここでは、Windows 7 を使用することを前提に手順を示しますが、Visual Studio 2008 SP1 が動作可能な環境であれば、他の Windows 環境でも同様の演習を行うことは可能です。

Visual C# 2008 のエディションは問いません。(ただし、このドキュメントの作成には、Visual Studio Team System を使用しています。エディションによっては、ドキュメント内のキャプチャ画面の外観などが、若干異なる場合があります。) Visual C# 2008 Express Edition も演習を行うことが可能です。Visual C# 2008 Express Edition は、Visual Studio 2008 Express Edition SP1 の一部として、以下の URL のダウンロードセンターから無償で入手できます。(2009 年 8 月現在)

http://www.microsoft.com/downloads/search.aspx?displaylang=ja

なお、Visual Studio 2008 を使用するにあたり、SP1 をご使用ください。今回の演習で利用する WPF Toolkit は、SP1 環境が前提になります。

WPF Toolkit – June 2008 Release (以降は WPF Toolkit と表記) は、以下の URL から入手できます。

http://wpf.codeplex.com/Release/ProjectReleases.aspx?ReleaseId=29117 ダウンロードページ
http://wpf.codeplex.com/ CodePlex における WPF 関連サイトのホーム

また、WPF Toolkit の入手方法、及びインストール方法については、後述の「WPF Toolkit の入手とインストール」を参照してください。

演習に使用する付属のソース プログラムの入手方法については、このドキュメントを入手されたサイト等でご確認ください。ソース プログラムの使用方法は、後述の「演習で使用する付属のソース プログラム (サンプル プログラム) のインストール方法」の項を参照してください。

WPF Toolkit の入手とインストール

WPF Toolkit は、バイナリ ファイルとソース ファイルが公開されています。以下の URL にアクセスした後、次図のリンクからそれぞれ入手することができます。(この演習で最低限必要なのは、バイナリ ファイルです。)

http://wpf.codeplex.com/Release/ProjectReleases.aspx?ReleaseId=29117 ダウンロードページ

なお、上記のリンクをクリックすると、「Microsoft Public License (Ms-PL)」が表示されるので、当該ファイルを入手するには、これに同意する必要があります。

バイナリ ファイルは、ファイル名「**WPFToolkit.msi**」(Windows Installer パッケージ) として提供されます。バイナリ ファイルとソース ファイルをダウンロードする場合は、ZIP ファイル形式であり、この ZIP ファイルの中に、**WPFToolkit.msi** が含まれます。

演習環境を構築するには、この **WPFToolkit.msi** を実行し、セットアップ ウィザードを起動させ、既定のオプションのまま、WPF Toolkit をインストールしてください。

演習で使用する付属のソース プログラム (サンプル プログラム) のインストール方法

演習で使用するソース プログラムには、特別なインストール方法はありません。入手されたソース プログラムのフォルダ「**HOLDataGrid**」全体を、任意のパスにコピーしてください。たとえば、C:¥ にコピーすれば、演習で作業を行うソース プログラムのパスは、次のようになります。

例 1. **C:¥ HOLDataGrid**

なお、演習の本文では、ソースプログラム等の位置を示す際、次のように、ソース プログラムのルートフォルダに対する相対パスで表記しています。

例 2. **Ex1_Starter¥CheckbookManager.sln**

この場合、ソース プログラムのフォルダが例 1 の場所であるなら、絶対パスは次の意味になります。

例 3. **C:¥HOLDataGrid¥Ex1_Starter¥CheckbookManager.sln**

なお、演習作業の中で、ソースプログラムに書き込む場合もあるので、作業を行うユーザーアカウントには、ソース プログラムに対して、書き込み可能なアクセス許可を与えてください。

演習の目的

この演習では、小切手の記録帳である既存の WPF アプリケーションを使用して、これを WPF Toolkit で利用可能な新しい DataGrid コントロールを使用するように変更します。ここでは、カスタム スタイルを伴う、データバインディングを行った ListView コントロールを、同じスタイルを使用して、データバインディングを行った DataGrid コントロールに書き換えます。

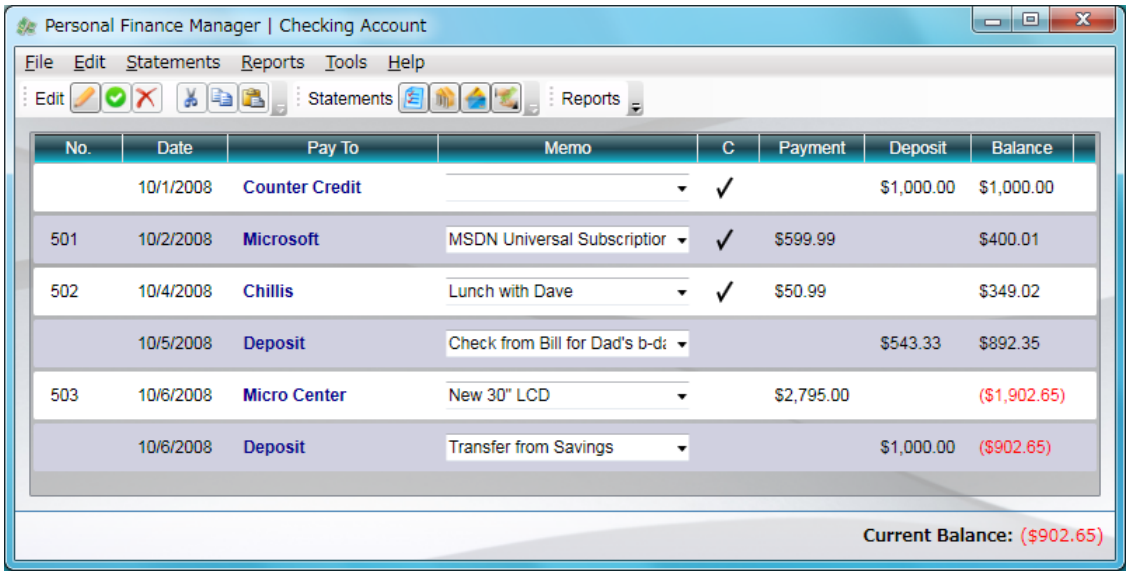
> **Note:** この演習の中のいくつかのタスクには、その時点まで行ったソース プログラムもあります。演習の途中から始めたい場合や、作業中のソース プロクラムを壊してしまった場合、それらを使用すると便利です。

練習 1: 新しい WPF DataGrid の使用

この練習では、ListView を使用している既存の小切手記録帳のアプリケーションを題材にして、DataGrid を使用するように変更します。これを行うために、この練習では、次の手順を行います。

- DataGrid をツールボックスからドロップし、プロジェクトから参照できるようにする
- 既存の ListView と GridView を DataGrid に置き換える
- 見栄えが同じになるように、スタイルとテンプレートを修正する

ここでは次図に示すような、小切手データとデータ バインディングを行った、簡単な WPF アプリケーションを使用して作業を始めます。



この練習は、三つのタスクに分かれています。各タスクで使用するソース プログラムには、途中まで作成したものも用意されています。すべてのタスクを行う必要がなければ、必要に応じて該当するタスクから始めることができます。(タスク 1 では作り込みをしないので、タスク 1 とタスク 2 は同じソースです。)

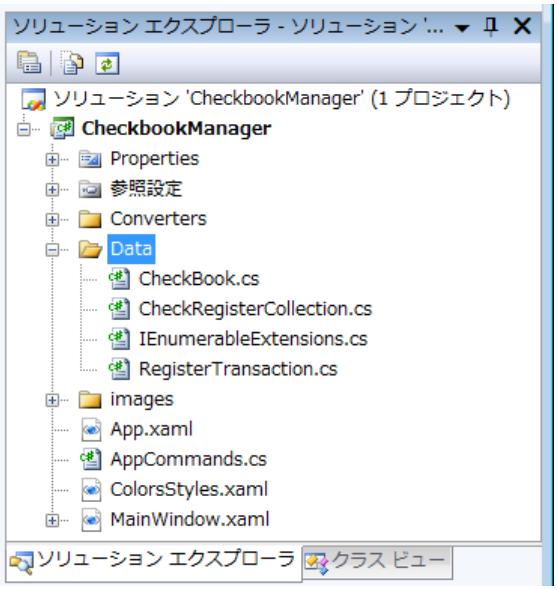
タスク 1 – 既存のアプリケーションを確認する

1. Visual Studio 2008 SP1 を使用して、この演習用のフォルダの中の、以下のパスのソリューションを開きます。これは、この練習で題材として使用するアプリケーションです。

   Ex1_Starter¥CheckbookManager.sln

2. このソリューションをビルドします。(Visual Studio 2008 の [ビルド] メニューにある [ソリューションのビルド] をクリックします。)

3. ここで、アプリケーションを実行して ([デバッグ] メニューの [デバッグなしで開始] をクリックします)、このアプリケーションの動作を確認します。表示されたグリッド形式の中の各セルをクリックすると、値を編集できます。特に、[Payment] 列や [Deposit] 列の各金額のセルの値を変更して、別のセルをクリックして入力を確定すると、収支のバランスを示す [Balance] 列の値も連動して変化します。(サンプルを簡単にするため、メニューやツールバーは、今のところ正しくは動作しません。)
4. アプリケーションの実行を終了して、ソースコードを確認します。主要なファイルは、MainWindow.xaml です。このファイルには、ListView が定義されており、その色に影響するスタイルや列の定義も含まれます。
5. ここでの列の定義には、テンプレートを使用しています。というのは、ListView の既定の列は、読み取り専用であり、ここでは、変更可能なデータグリッドを疑似的に作るため、テンプレートを使用して、各列を TextBox に置き換えています。
6. そのサンプルで使用されるデータ構造の定義は、Data サブディレクトリにあるので、必要があれば確認してみてください。ここでデータ構造の定義に使用したクラスを、完全には理解する必要はありません。これらのデータ構造は必要に応じて使用しますが、その機能を変更することはありません。



   a. Checkbook.cs には、このアプリケーションで使用されるモデルが定義されており、ユーザーインターフェイスから制御されるデータに当たります。
   b. CheckRegisterCollection.cs には、取引データを保持する ObservableCollection クラスの派生クラスが定義されています。
   c. RegisterTransaction.cs には、小切手の記録帳に使用する一つ分の取引に関わるエントリが定義されています。

7. コードの構造に関して、主だった点が理解できたら、タスク 2 に進んでください。

タスク 2 – DataGrid を使用する

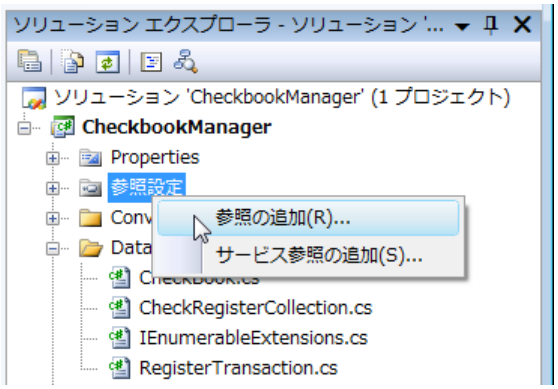
このタスクでは、このアプリケーションの ListView を、新しい WPF のコントロールである DataGrid に置き換えます。

DataGrid を使用する最初の手順は、適切なアセンブリの参照を追加することです。DataGrid は、WPF Toolkit のアセンブリ (WPFToolkit.dll) に含まれます。このアセンブリは単体のファイルであり、DataGrid が実装されているほか、その他にもいくつかのコントロールと新機能が実装されています。このアセンブリを利用する場合、どうインストールしたかによって、二通りの方法があります。ここでは、参考までに両方の方法が掲載されていますが、そのうち一つを行います。

1. まずは、この後の作業でデザイナを正しく動作させめために、アプリケーションをビルドして、今のところ、アプリケーションが正しく構築されていることを確認します。

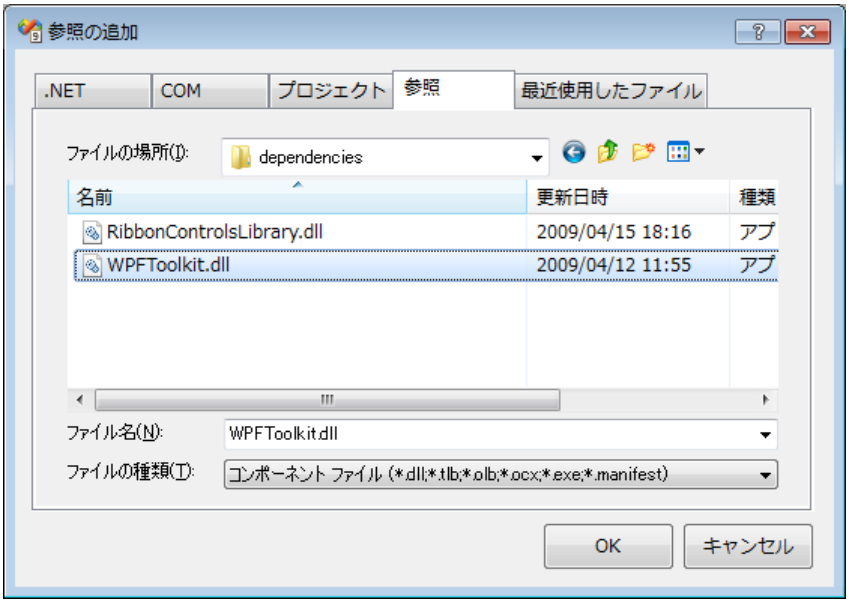
> **Note:** このあとの手順には、WPF Toolkit のアセンブリを使う二つの方法が、掲載されています。一つ目の方法は、WPFToolkit.dll を手動で参照する方法です。この方法は、このあとの手順 2 から手順 10 までです。二つ目の方法は、予め WPF Toolkit のインストール パッケージ WPFToolkit.msi を使用して、インストールする方法です。この方法は、手順 11 から手順 15 までです。ここでは、既に WPFToolkit.msi をインストールしてあるので、手順 11 に進んでください。

2. WPF Toolkit のアセンブリを使う一つ目の方法は、手動で参照する方法です。この方法を使用する状況は、単体のアセンブリ ファイルはあるが、WPF Toolkit のインストール パッケージである WPFToolkit.msi を使用してインストールはしていないときです。
   a. ソリューション エクスプローラ上で、[参照設定] ノードを右クリックして、[参照の追加] メニューをクリックします。(次図参照)



b. [参照の追加] ダイアログボックスが表示されたら、[参照] タブをクリックし、予め用意したアセンブリ WPFToolkit.dll を選択します(次図参照)。

Note: 予め WPF Toolkit のソースファイルのビルドを行うなどして、WPFToolkit.dll を用意しておく必要があります。



c. 選択が済んだら、[OK] をクリックして、このダイアログボックスを閉じます。

3. ソリューション エクスプローラ上で、MainWindow.xaml をダブルクリックして開きます。XAML ビューを使用して、次のように <Window> 開始タグの 2 行目に、WPF Toolkit を参照するために、WPF Toolkit のために予め定義された名前空間を、プレフィックス my で参照できるようにします。(「xmlns:my=～」の行を追加します。)

XAML

```
<Window x:Class="CheckbookManager.MainWindow" ...
    xmlns:my="http://schemas.microsoft.com/wpf/2008/toolkit"
    ...
    ... Icon="images/Coins.png">
```

Note: この記述によって、この <Window> 要素ブロックの内側では、<my:要素名> のようにプレフィックス my を伴うタグは、WPF Toolkit 内のコントロールであることを意味するようになります。なお、プレフィックスの名前は、任意の名前を定義することができますが、この後の WPF Toolkit を使用する二つ目の方法では、自動的にプレフィックスが my になるので、以降の手順では、プレフィックスは my とします。

4. XAML ビューの中をスクロールダウンして、ListView の開始タグ <ListView> を見つけます。このあと、このコントロールを DataGrid に置き換えます。

5. ListView の開始タグ <ListView> と終了タグ </ListView> を、「my:DataGrid」の開始タグと終了タグに書き換えます。つまり、新しい DataGrid コントロールに置き換えます。こ

の時点では、IntelliSense 機能によって、複数のタグにエラーが表示されますが、この後で修正します。(修正後のコードは、この後に記載されています。)

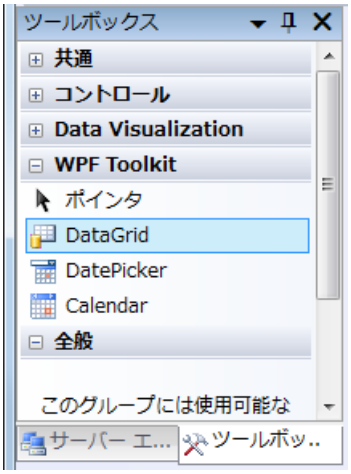
6. この開始タグの中の、ItemContainerStyle 属性 (プロパティ) を削除します。後ほど、追加し直しますが、まず削除してください。
7. もともとの <ListView> 要素ブロックにあった、TextBox のスタイルを定義したリソースのブロックである、 <ListView.Resources> から </ListView.Resources> までを削除します。列の新しい定義をするので、このリソースは必要ありません。
8. <ListView.ContextMenu> から </ListView.ContextMenu> までのセクションについて、開始タグと終了タグをそれぞれ、<my:DataGrid.ContextMenu>、</my:DataGrid.ContextMenu> に変更します。
9. <ListView.View> から </ListView.View> までのセクションをコメントアウトしておきます。列の定義は改めて行いますが、今のところ、このセクションでの列定義は不適切なので、保留にしておきます。
10. この時点で、DataGrid コントロールへ変更した XAML が、次のようになっていることを確認します。

XAML

```
<!-- DataGrid fills remainder of space -->
<my:DataGrid x:Name="dg" ItemsSource="{Binding}" Margin="10"
             Background="#80909090" AlternationCount="2">
    <my:DataGrid.ContextMenu >
        <ContextMenu >
           <MenuItem Header="Copy Selected Transactions"
                     Command="{x:Static ApplicationCommands.Copy}" />
        </ContextMenu>
    </my:DataGrid.ContextMenu>
    <!--
    <ListView.View>

        : (省略)

    </ListView.View>
    -->
</my:DataGrid>
```

Note: 次の手順からは、WPF Toolkit のアセンブリを使用する二つ目の方法 (DataGrid コントロールを追加する二つ目の方法) を取り上げます。一つ目の方法で既に追加した場合は、手順 16 に進んでください。

11. 二つ目の方法は、ツールボックスを使用する方法です。WPF Toolkit のインストール パッケージ (WPFToolkit.msi) でインストールするとツールボックスには、DataGrid コントロールが追加されます。MainWindow.xaml のデザイン ビューが開いた状態で、ツールバーを表示し、次図のように DataGrid が存在することを確認します。



12. ツールボックスから DataGrid コントロールをドラッグして、デザインビュー上のフォームの空いている領域 (既存の ListView の右隣りあたり) にドロップします。

Note: この操作によって、DataGrid コントロールが追加されるだけでなく、手順 2 の WFPToolkit.dll への参照の追加や、手順 3 の名前空間の定義も自動的に行われます。

13. 次に示すように、DataGrid コントロールが追加されたことを確認します。(DockPanel の終了タグの前あたりに追加されます。なお、紙面の都合のため、途中改行しています。)

XAML

```
        :
  <my:DataGrid AutoGenerateColumns="False" Height="200"
               Name="dataGrid1" Width="200"
               xmlns:my="http://schemas.microsoft.com/wpf/2008/toolkit" />
</DockPanel>
```

14. ここで、次に示す XAML と同じになるように、DataGrid コントロールを修正します。(修正が完了したコードは、この後にあります。)
    a. <my:DataGrid> タグの末尾にあるスラッシュ (/) を削除して、このタグを開始タグに変更します。
    b. この開始タグの後の行に、</my:DataGrid> 終了タグを追加します。

c. 既存の ListView コントロールの開始タグから、x:Name 属性、ItemsSource 属性、Margin 属性、Background 属性、および　AlternationCount 属性をコピーして、次の修正後のコードのように、<my:DataGrid> 開始タグを書き換えます。

d. ListView と同じ内容の ContextMenu セクションを追加します。

e. 既存の <ListView> 要素のブロック全体をコメントアウトしておきます。

XAML

```
    <my:DataGrid x:Name="dg" ItemsSource="{Binding}" Margin="10"
                 Background="#80909090" AlternationCount="2"
                 xmlns:my="http://schemas.microsoft.com/wpf/2008/toolkit">
        <my:DataGrid.ContextMenu >
            <ContextMenu >
                <MenuItem Header="Copy Selected Transactions"
                          Command="{x:Static ApplicationCommands.Copy}" />
            </ContextMenu>
        </my:DataGrid.ContextMenu>
    </my:DataGrid>
</DockPanel>
```

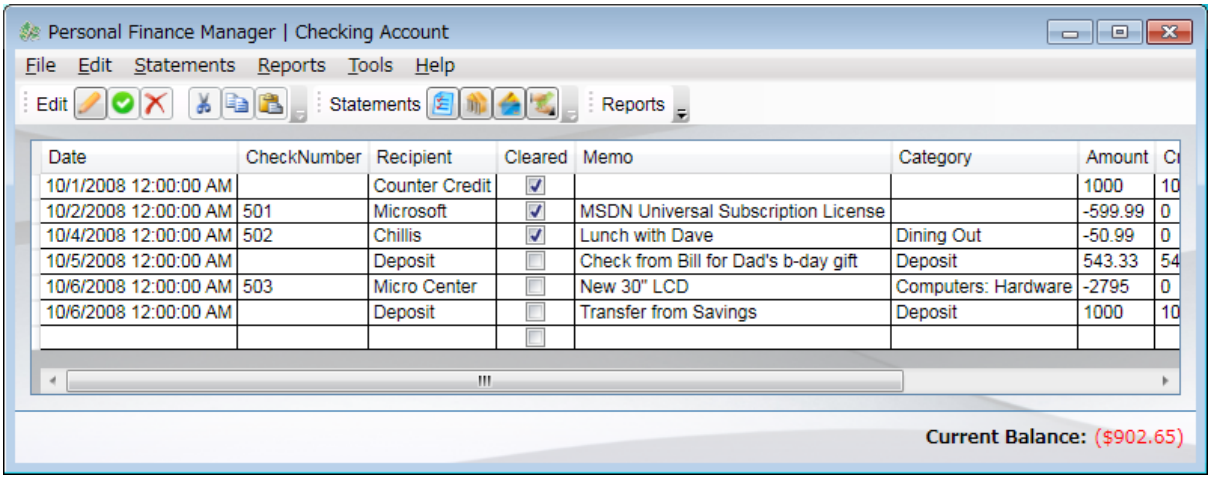
15. MainWindow.xaml の範囲内で、プレフィックス「my:」を使用するので、<my:DataGrid> 開始タグに含まれる「xmlns:my=～」の部分を、 <Window> 開始タグに移動します。

XAML

```
<Window x:Class="CheckbookManager.MainWindow"
        xmlns:my="http://schemas.microsoft.com/wpf/2008/toolkit"
       ...
```

Note: これ以降は、前述の WPF Toolkit を追加する二つの方法に関して、共通の手順です。

16. ここで、アプリケーションを実行します。DataGrid に自動的に列が生成され、データが設定されている点に注意してください。これは、DataGrid の備わった機能であり、バインディング対象のデータを解析して、その型情報を元にして、列を生成しています。



17. セルをダブルクリックすると、TextBox として入力できることを確認します。この動作は、後で変更します。また、boolean 型の [Cleared] 列の表示が、自動的に CheckBox として表示されることを確認します。
18. セルの TextBox でキー入力を行った後、確定せずに [ESC] キーを押してください。変更内容がキャンセルされ、元に戻ることを確認します。

> **Note:** このキャンセル動作の元になる仕組みは、既存の .NET Framework で提供されています。DataGrid コントロールは、これを使用してします。この仕組みでは、グリッドに表示されている元のデータを調べ、そのデータが IEditableObject インターフェイスを実装していることが分かると、そのデータのメソッド(インターフェイスのメソッド)である BeginEdit、CommitEdit、および CancelEdit メソッドを、自動的に必要に応じて呼び出します。RegisterTransaction.cs のクラスは、このインターフェイスをサポートしています。BeginEdit メソッドが呼び出されると、オリジナルデータを退避し、CancelEdit メソッドが呼び出されると、オリジナル データに戻します。また、CommitEdit メソッドが呼び出されると、オリジナル データは破棄されます。DataGrid のセルを会して編集するデータとして、任意の編集可能なオブジェクトを使用したいとき、この仕組みを利用することができます。

19. アプリケーションを終了し、ソースコードの編集環境に戻ります。

列の定義をよりきめ細かく制御する必要があれば、列自身を直接定義することもできます。現在表示されている列は、元になるデータ (オブジェクト)のプロパティ定義の順番に並んでいます。現時点では、最初のアプリケーションと並びが変わってしまい、本来の望んでいた順番になっていません。列を定義するのは、難しくありません。DataGrid には、Columns プロパティに追加する列の定義として、次に挙げる 5 種類の組み込みの列の型が用意されています。

- **DataGridCheckBoxColumn** – Boolean 型を表す **CheckBox** コントロールを表示す
- **DataGridComboBoxColumn** – 項目一覧である **ComboBox** コントロールを表示する
- **DataGridHyperlinkColumn** – Uri を表す **Hyperlink** コントロールを表示する
- **DataGridTextColumn** – 単一のデータ項目を表す **TextBlock** コントロールや **TextBox** コントロールを表示する
- **DataGridTemplateColumn** – 複数のコンテンツを表示する **DataTemplate** を使用する

これらの列の型を通じて、ヘッダーや幅などのプロパティを設定するほか、列に追加できるコントロールの種類が決まります。ここで、既存の ListView の列定義と同様になるように、[CheckNumber]列のスタイルを変更することにします。

20. DataGrid のブロックの内側に、<my:DataGrid.Columns> セクションを追加します。(内側であれば、他のセクションとの前後関係は問いませんが、これ以降のソースコードでは、

<my:DataGrid.ContextMenu > セクションの後に追加しています。なお、完成したソースは、このあとに掲載してあります。)

21. このセクションの内側に、 <my:DataGridTextColumn> 要素を追加します。

    a. ヘッダーに「No.」と表示させるため、Header プロパティ (Header 属性) にこのテキストを設定します。

    b. 適当な列幅になるよう、Width プロパティを設定します。このプロパティに設定できる値は、幅を固定する特定の数値や残りのスペースを占める「*」のほか、その列の中で最大のセルのデータの大きさに合わせる「SizeToCells」、また、ヘッダーの大きさに合わせる「SizeToHeader」などがあります。ここでは、「SizeToCells」に設定します。

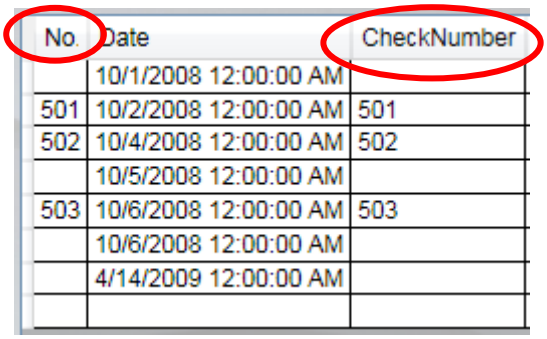
    c. 最後に、この列が取得すべきデータを示すために、バインディング式を Binding プロパティに設定します。ここでは、元々の ListView の列に指定されていたバインディング式を指定します。[Check Number] 列の場合は、「{Binding CheckNumber}」です。

    d. ここまでのところで、以下のようになることを確認します。

XAML

```
<my:DataGrid x:Name="dg" ItemsSource="{Binding}" Margin="10"
             Background="#80909090" AlternationCount="2">


    <my:DataGrid.ContextMenu>

        : (省略)

    </mh:DataGrid.ContextMenu>

    <my:DataGrid.Columns>
        <my:DataGridTextColumn Header="No." Width="SizeToCells"
                               Binding="{Binding CheckNumber}" />
    </my:DataGrid.Columns>

</my:DataGrid>
```

22. アプリケーションを再び実行します。[No.] というヘッダーの列 (Check number の列) が先頭に追加されました。しかしまだ、DataGrid は自動的に列を追加しています。



No.	Date	CheckNumber
	10/1/2008 12:00:00 AM	
501	10/2/2008 12:00:00 AM	501
502	10/4/2008 12:00:00 AM	502
	10/5/2008 12:00:00 AM	
503	10/6/2008 12:00:00 AM	503
	10/6/2008 12:00:00 AM	
	4/14/2009 12:00:00 AM	

DataGrid.AutoGeneratingColumn イベントを使用すれば、列を生成する過程で処理を割り込ませることができます。各列について、このイベントが発生した際に、その列の状態を調べ、列の表示方法を変えることができます。また、列の生成自体をキャンセルさせることもできます。しかし、ここでは、列すべてを制御したいので、自動生成を止めることにします。

23. 列の自動生成を中止するために、AutoGenerateColumns プロパティを「False」に設定します。

XAML

```
<my:DataGrid x:Name="dg" ItemsSource="{Binding}" Margin="10"
             AutoGenerateColumns="False"
             Background="#80909090" AlternationCount="2">
```

24. ここで再び実行します。[No.] 列だけになることを確認します。これ以降は、残りの列の定義を追加します。

25. 元の ListView と列を同じようにするため、<my:DataGrid.Columns> セクションの中に、次のように 、データバインディングを行った [Date] 列と [Pay To] 列を 追加します。

XAML

```
    <my:DataGrid.Columns>
        <my:DataGridTextColumn Header="No." Width="SizeToCells"
                               Binding="{Binding CheckNumber}" />
        <my:DataGridTextColumn Header="Date"
                               Binding="{Binding Date, StringFormat=d}" />
        <my:DataGridTextColumn Header="Pay To" MinWidth="200"
                               Binding="{Binding Recipient}" />
    </my:DataGrid.Columns>
```

ここまでは、TextBox を追加しました。次の列は [Memo] 列であり、次のように、元の ListView では ComboBox を使用するように構成されていました。(紙面の都合で、長い行は途中で改行しています。)

XAML

```
<GridViewColumn Header="Memo" Width="185">
   <GridViewColumn.CellTemplate>
      <DataTemplate>
         <ComboBox Width="180" IsEditable="True"
                   HorizontalAlignment="Center"
                   ItemsSource="{Binding Source=
                     {x:Static Data:CheckBook.Descriptions},
                     Mode=OneWay}"
                   Text="{Binding Memo}" />
      </DataTemplate>
   </GridViewColumn.CellTemplate>
</GridViewColumn>
```

上記の GridView のように、DataGrid でも DataTamplate を使用することはできます。しかしここでは、組み込みの列の型である DataGridComboBoxColumn を使用することにします。元の ListView で使用していたプロパティの値のみ、引き続き同じものを使用します。

26. <my:DataGrid.Columns> セクションの中に、 次のような <my:DataGridComboBoxColumn> 要素を追加します。
    a. Header プロパティは「Memo」にします。
    b. 残りのスペースを占めるように、Width プロパティは「*」にします。
    c. ItemsSource プロパティは、GridView の ComboBox と同じバインディングを使用します。(バインディング式の文字列の部分は、1 行で入力してください。)
    d. テキストボックスの部分は、ComboBox.Text とバインディングさせるため、これを TextBinding プロパティに設定します。
    e. ここまでのところで、この列の定義が次のようになることを確認します。

XAML

```
<my:DataGridComboBoxColumn Header="Memo" Width="*"
     ItemsSource="{Binding
     Source={x:Static Data:CheckBook.Descriptions},
     Mode=OneWay}"
     TextBinding="{Binding Memo}">
</my:DataGridComboBoxColumn>
```

この列について、ユーザーが テキストを自由に入力できるようにするため、ComboBox の IsEditable プロパティを設定します。しかし、DataGridComboBoxColumn では、このプロパティを直接指定できません。その代わり、ElementStyle プロパティや EditingElementStyle プロパティの中で、スタイルを使用して指定することができます。

ElementStyle は、そのセルがフォーカスを持たない場合のスタイルとして使用されます。既存の DataGridTextColumn では、セルに TextBlock が置かれます。EditingElementStyle は、セルが入力フォーカスを取得した場合に切り替わるスタイルです。今のところ、セルを選択したとき、TextBlock から TextBox へ切り替わります。

これらのスタイルのプロパティを使用して、表示される各要素の既定のプロパティを変更することができます。ここでは、EditingElementStyle を介して、IsEditable プロパティを「True」に設定します。

27. 次に示すように、[Memo] 列の DataGridComboBoxColumn 要素ブロックの内側に入るよう <my:DataGridComboBoxColumn.EditingElementSytle> セクションを追加し、ComboBox の IsEditable プロパティを「True」に設定します。

XAML

```
        <my:DataGridComboBoxColumn Header="Memo" Width="*"
             ItemsSource="{Binding Source={x:Static
                        Data:CheckBook.Descriptions}, Mode=OneWay}"
             TextBinding="{Binding Memo}">
            <my:DataGridComboBoxColumn.EditingElementStyle>
                <Style TargetType="ComboBox">
                    <Setter Property="IsEditable" Value="True" />
                </Style>
            </my:DataGridComboBoxColumn.EditingElementStyle>
        </my:DataGridComboBoxColumn>
```

28. 次に追加すべき列は [Cleared] 列です。この列は Boolean 型であり、既定で CheckBox にマップされます。ここでは、DataGridCheckBoxColumn を使用しますが、見た目を修正します。まず、元の ListView を確認します。

XAML

```
    <GridViewColumn Header="C">
       <GridViewColumn.CellTemplate>
          <DataTemplate>
             <CheckBox Margin="3"
                       IsChecked="{Binding Cleared}"
                       Style="{StaticResource NoBorderCheckBoxStyle}" />
          </DataTemplate>
       </GridViewColumn.CellTemplate>
    </GridViewColumn>
```

上記のように、テンプレートを変更するために、CheckBox にスタイル (NoBorderCheckBoxStyle) が適用されていることが分かります。既に触れたように、これと同じスタイルを適用するために、ElementStyle や EditingElementStyle を使用することができます。

29. 引き続き <my:DataGrid.Columns> セクションの中の、<my:DataGridComboBoxColumn> 要素の次に、以下のような <my:DataGridCheckBoxColumn> を追加します。バインディ

ング式には、「{Binding Cleared}」を用いて、ElementStyle と EditingElementStyle の両方に、スタイルを適用します。(ここでは、コントロールがフォーカスを持つ場合と、持たない場合の両方に適用することにします。)

a. Binding プロパティにバインディング式を設定します。

b. Header プロパティには「C」を設定します。

c. Width プロパティには「SizeToHeader」を設定し、ヘッダーの幅に合わせます。

XAML

```
<my:DataGridCheckBoxColumn Header="C" Width="SizeToHeader"
     Binding="{Binding Cleared}"
     ElementStyle="{DynamicResource NoBorderCheckBoxStyle}"
     EditingElementStyle="{DynamicResource NoBorderCheckBoxStyle}" />
```

30. さらに、[Payment] 列と [Deposit] 列を追加します。ここでは、

<my:DataGridTextColumn> を使用します。

a. 元の ListView の列のバインディング式を流用します(Converter の指定を含む)。

b. Width プロパティを「SizeToCells」に指定します。

XAML

```
<my:DataGridTextColumn Width="SizeToCells" Header="Payment"
    Binding="{Binding Amount,
              Converter={StaticResource amountConverter},
              ConverterParameter=0, StringFormat=C}" />
<my:DataGridTextColumn Width="SizeToCells" Header="Deposit"
    Binding="{Binding Amount,
              Converter={StaticResource amountConverter},
              ConverterParameter=1, StringFormat=C}" />
```

最後に、[Balance] 列を追加します。

今度は、読み取り専用の列にします。DataGridColumn の IsReadOnly プロパティを使用して、これを指定することができます。しかし、ここでは値の大きさに応じて、テキストの色を変えてみます。特定の一か所の値だけで決まるのではないので、トリガでは実現できません。ここでは、セルの Foreground プロパティとデータバインディングを行い、列の表示を変えるようにします。

これを行うために、各セルに DataTemplate を提供できる DataGridTemplateColumn を使用します。テンプレートは、CellTemplate プロパティに設定します。元になるテンプレートは、GridView 用のものが既にあるので、これを流用することにします。

XAML

```
<DataTemplate>
    <TextBlock Text="{Binding TotalBalance, StringFormat=C}"
               Foreground="{Binding TotalBalance,Converter={StaticResource
               BalanceDisplayConverter}}"
                   />
</DataTemplate>
```

31. 前の手順に続いて列の定義として、<my:DataGridTemplateColumn> を追加します。

    a. Width プロパティを「SizeToCells」に設定します。

    b. Header プロパティを「Balance」に設定します。

    c. ClipboardContentBinging プロパティを追加し、前述と同じデータをバンディング式に指定します。このプロパティは、コンテンツをクリップポートにコピーする際に、この列の値として利用されます。この記述がないと、この列は無視されます。

    d. 最後に、DataGridTemplateColumn.CellTemplate プロパティに、データテンプレートを設定します。ここまでのところで、この列の定義は次のようになります。

XAML

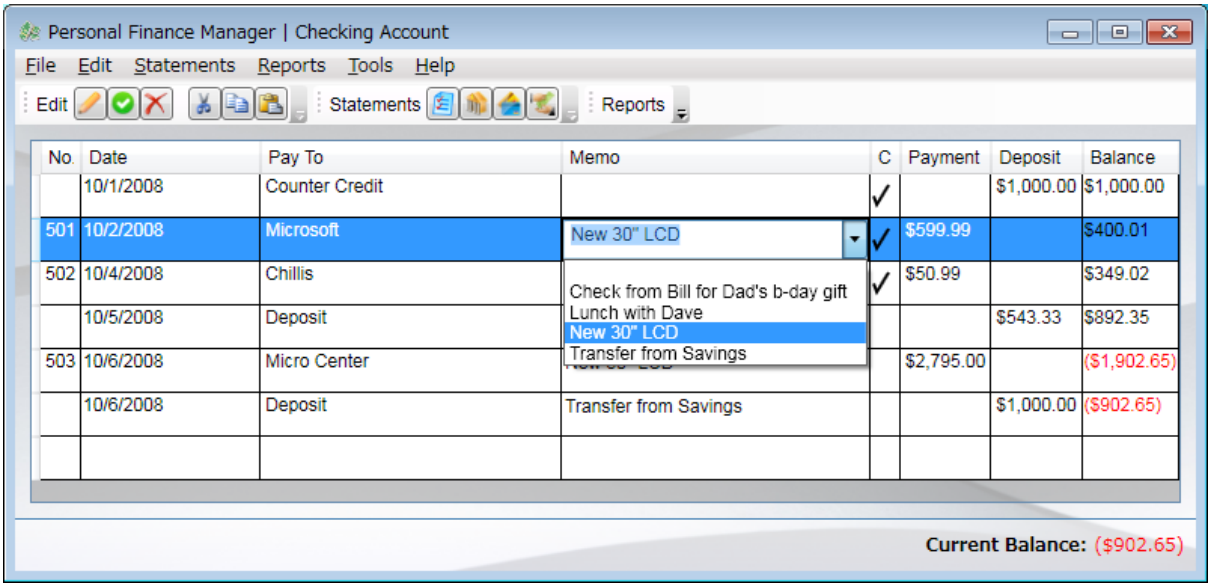
```
<my:DataGridTemplateColumn Width="SizeToCells" Header="Balance"
                           ClipboardContentBinding="{Binding TotalBalance}">
    <my:DataGridTemplateColumn.CellTemplate>
        <DataTemplate>
            <TextBlock Foreground="{Binding TotalBalance,
                Converter={StaticResource BalanceDisplayConverter}}"
                Text="{Binding TotalBalance, StringFormat=C}" />
        </DataTemplate>
    </my:DataGridTemplateColumn.CellTemplate>
</my:DataGridTemplateColumn>
```

今まで入力したコードの確認のため、DataGrid コントロールの完全な列を掲載しておきます。

XAML

```
<my:DataGrid x:Name="dg" ItemsSource="{Binding}" Margin="10"
             AutoGenerateColumns="False"
             Background="#80909090" AlternationCount="2">

          : (省略)

   <my:DataGrid.Columns>
      <my:DataGridTextColumn Header="No." Width="SizeToCells"
                             Binding="{Binding CheckNumber}" />
      <my:DataGridTextColumn Header="Date"
                             Binding="{Binding Date, StringFormat=d}" />
      <my:DataGridTextColumn Header="Pay To" MinWidth="200"
                             Binding="{Binding Recipient}" />

      <my:DataGridComboBoxColumn Header="Memo" Width="*"
           ItemsSource="{Binding Source={x:Static
                        Data:CheckBook.Descriptions}, Mode=OneWay}"
           TextBinding="{Binding Memo}">
          <my:DataGridComboBoxColumn.EditingElementStyle>
              <Style TargetType="ComboBox">
                  <Setter Property="IsEditable" Value="True" />
              </Style>
          </my:DataGridComboBoxColumn.EditingElementStyle>
      </my:DataGridComboBoxColumn>

      <my:DataGridCheckBoxColumn Header="C" Width="SizeToHeader"
          Binding="{Binding Cleared}"
          ElementStyle="{DynamicResource NoBorderCheckBoxStyle}"
          EditingElementStyle="{DynamicResource NoBorderCheckBoxStyle}" />

      <my:DataGridTextColumn Width="SizeToCells" Header="Payment"
          Binding="{Binding Amount,
                    Converter={StaticResource amountConverter},
                    ConverterParameter=0, StringFormat=C}" />
      <my:DataGridTextColumn Width="SizeToCells" Header="Deposit"
          Binding="{Binding Amount,
                   Converter={StaticResource amountConverter},
                   ConverterParameter=1, StringFormat=C}" />

      <my:DataGridTemplateColumn Width="SizeToCells" Header="Balance"
                   ClipboardContentBinding="{Binding TotalBalance}">
          <my:DataGridTemplateColumn.CellTemplate>
              <DataTemplate>
                  <TextBlock Foreground="{Binding TotalBalance,
                     Converter={StaticResource BalanceDisplayConverter}}"
                     Text="{Binding TotalBalance, StringFormat=C}" />
              </DataTemplate>
          </my:DataGridTemplateColumn.CellTemplate>
      </my:DataGridTemplateColumn>
   </my:DataGrid.Columns>

</my:DataGrid>
```

32. ここで、アプリケーションを実行します。元の ListView の表示にだいぶ近づいてきたはずです。ドロップダウン リストが備わり、チェックボックスのスタイルも指定され、[Balance] 列も色付けされるようになりました。



33. 元のものと比べて抜けている点の一つは、[PayTo] 列の表示が青色で太字になっていない点です。どのように変えたらよいか、考えてみましょう。

    ヒント: 上記の手順でも取り上げたように、方法は二つあります。

ここまでの作業が完了したら、次のタスクでは、より見栄えをアピールできるように、DataGrid にスタイルを追加します。しかし、もし余力があるようであれば、この後の手順にあるように、[Date] 列にカレンダを表示できるようにするため、Calendar や DatePicker を利用してもよいでしょう。また、以下の手順を省略してタスク 3 に進み、予め用意されたタスク 2 までの完成品を利用することもできます。

34. MainWindow.xaml を XAML ビューで開き、日付 (Date) にバインディングを行った (つまり、[Date] 列の) 定義である DataGridTextColumn 要素を見つけます。 これをコメントアウトして、代わりに、<my:DataGridTemplateColumn> の要素ブロックを追加し、Header プロパティは今までと同様に「Date」に設定して、幅は最小 100 ピクセルにするため、MinWidth プロパティを「100」に設定します。(完成したコードは、この後に掲載してあります。)

35. DataGridTextColumn 要素ブロックの内側では、編集時のテンプレートと、編集状態でない場合のテンプレートを追加します。次に示すように、CellEditingTemplate プロパティにデータテンプレートを設定します。

a. DataTemplate を作成し、WPF Toolkit の DatePicker を追加します。DatePicker のプレフィックスは、DataGrid と同じです。

b. SelectedDate プロパティに対して Date をバインディングします。

c. SelectedDateFormat プロパティを「Short」に設定します。

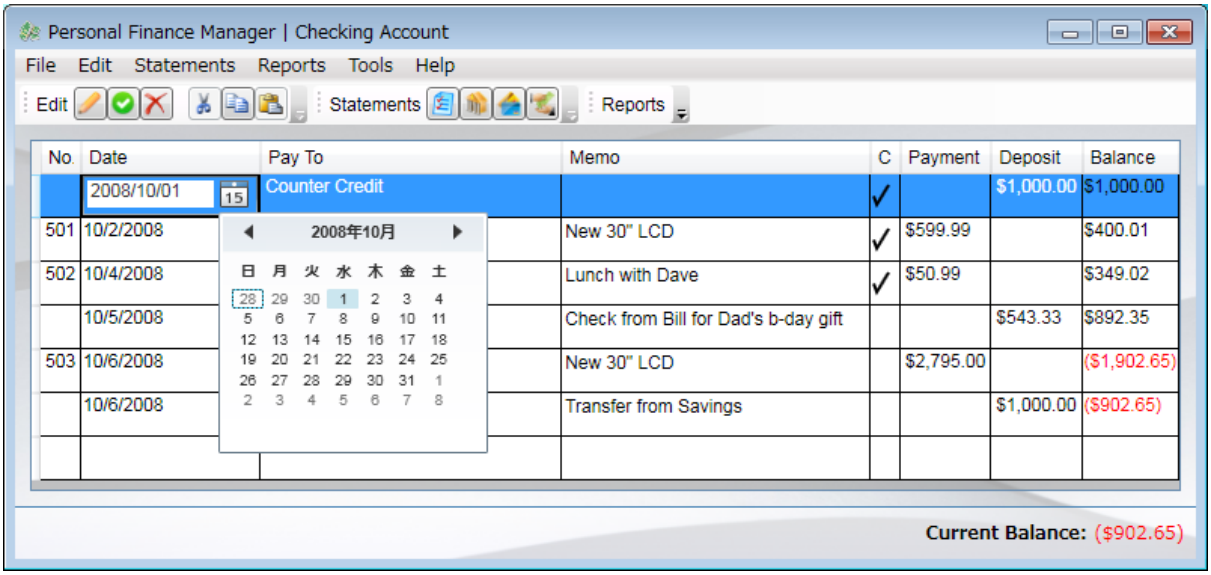
36. CellTemplate プロパティに設定する DataTemplate には、TextBlock を含めるようにします。日付の形式として、TextBlock の StringFormat プロパティに「d」を指定します。

37. 完成した XAML が、次のようになっていることを確認します。

XAML

```
<!--
<my:DataGridTextColumn Header="Date"
                       Binding="{Binding Date, StringFormat=d}" />
-->
<my:DataGridTemplateColumn Header="Date" MinWidth="100">
    <my:DataGridTemplateColumn.CellEditingTemplate>
        <DataTemplate>
            <my:DatePicker SelectedDate="{Binding Date}"
                           SelectedDateFormat="Short" />
        </DataTemplate>
    </my:DataGridTemplateColumn.CellEditingTemplate>
    <my:DataGridTemplateColumn.CellTemplate>
        <DataTemplate>
            <TextBlock Text="{Binding Date, StringFormat=d}" />
        </DataTemplate>
    </my:DataGridTemplateColumn.CellTemplate>
</my:DataGridTemplateColumn>
```

38. ここで、アプリケーションを実行します。今度は、[Date] 列のセルを編集状態にすると、カレンダ形式のドロップダウンを伴う DatePicker が使用されるようになります。



このタスクの完成品は、演習フォルダの Ex1_Solution_Task2 サブフォルダの中にあります。

タスク 3 – DataGrid にスタイルを付ける

元のアプリケーションの外見を思い出してください。ヘッダーにスタイルが設定され、色彩もより良いものでした。この新しい DataGrid には、もっと多くの機能がありますが、現時点の実装では、商用のアプリケーションとしては、非常に単調です。このタスクでは、DataGrid に様々なスタイルを適用して、より色彩の良いものにして、視覚的にアピールできるものに変えていきます。

他の WPF ベースのコントロールと同様に、DataGrid は完全にスタイルに対応できます。必要があれば、ほとんどの UI の側面にスタイルを適用できます。現在のコントロール テンプレートを独自のものに置き換えれば、まったく異なる見栄えにすることもできます。

1. Visual Studio 2008 SP1 を使用して、この演習用のフォルダの中の、以下のパスのソリューションを開きます。これは、タスク 2 で作成した完成品です。タスク 2 の演習作業でご自身が作成したものを、そのまま継続して使用することもできます。

   Ex1_Starter_Task3¥ CheckbookManager.sln
2. アプリケーションをビルドして、すべてのコンバーターが確実に利用可能な状態にします。
3. MainWindow.xaml を XAML ビューで開き、<DockPanel.Resources> セクション (リソース)に移動します。このリソースには、元の GridView の実装で使用していたスタイルが、いくつか見られます。特に、いつくかのブラシと、GridView の列のヘッダーのスタイルや、ListViewItem のスタイルが見受けられます。ここでの目的は、これらのスタイルを DataGrid に適用することです。

DataGrid には、その見栄えを修正するために、いくつかのスタイル関連のプロパティがあります。

- **CellStyle** – 各セル (DataGridCell) のスタイルに使用されます。
- **RowStyle** – 各行 (DataGridRow) のスタイルに使用されます。
- **ColumnHeaderStyle** – ヘッダー(**DataGridColumnHeader**) スタイルに使用されます。

さらに、ユーザーのグリッドに対する対話操作を変更するプロパティが、数多く用意されています。

- **SelectionMode** – 単一選択か複数選択かの切り替えを行う
- **SelectionUnit** – 何が選択されるかを変更する (常に行全体か、それともセルまたは行か)
- **GridLinesVisibility** – セルに関する枠線や描画を変更する
- **VerticalGridLinesBrush** – 垂直方向の線のブラシの色を変更する
- **HorizontalGridLinesBrush** – 水平方向の線のブラシの色を変更する

まず、既存のスタイルの TargetType プロパティを変更します。このスタイルに指定されたプロパティのすべてをこの後も直接利用できます。というのは、これらのプロパティは、ListView と DataGrid の共通の基本クラス ItemControl に定義されているからです。

4. スタイル dgHeaderStyle に移動して、TargetType プロパティを GridViewColumnHeader から my:DataGridColumnHeader に変更します。
5. スタイル dgRowStyle に移動して、TargetType プロパティを ListViewItem から my:DataGridRow に変更します。
6. ここまでの修正した結果が、次のようになることを確認します。(変更部分は、TargetType プロパティです。)

XAML

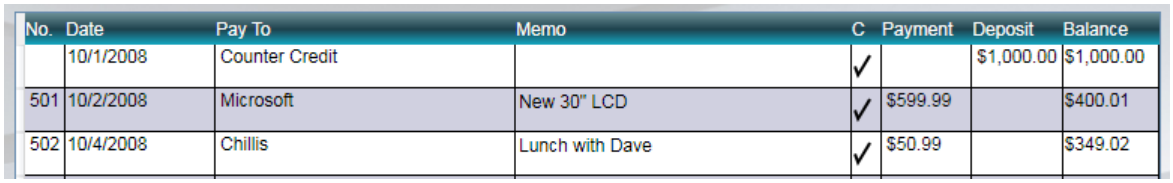
```
<Style x:Key="dgHeaderStyle" TargetType="my:DataGridColumnHeader">
   <Setter Property="Background" Value="{StaticResource dgHeaderBrush}" />
   <Setter Property="Foreground" Value="White" />
   <Setter Property="BorderBrush"
              Value="{StaticResource dgHeaderBorderBrush}" />
</Style>
<Style x:Key="dgRowStyle" TargetType="my:DataGridRow">
   <Setter Property="SnapsToDevicePixels" Value="True" />
   <Setter Property="Background" Value="White" />
   <Style.Triggers>
      <Trigger Property="ItemsControl.AlternationIndex" Value="1">
         <Setter Property="Background" Value="#FFD0D0E0" />
      </Trigger>
      <Trigger Property="IsSelected" Value="True">
         <Setter Property="Background" Value="LightGoldenrodYellow" />
      </Trigger>
   </Style.Triggers>
</Style>
```

7. <my:DataGrid> 開始タグまで下方向にスクロールして移動し、ColumnHeaderStyle プロパティと、RowStyle プロパティから、これらのスタイルを参照するよう追加します。

XAML

```
    <my:DataGrid x:Name="dg" ItemsSource="{Binding}" Margin="10"
              AutoGenerateColumns="False"
              Background="#80909090" AlternationCount="2"
              ColumnHeaderStyle="{StaticResource dgHeaderStyle}"
              RowStyle="{StaticResource dgRowStyle}">
```

8. ここで、アプリケーションを実行します。前述のスタイルが適用され、ヘッダーには綺麗なグラデーションが付き、行のデータは一行おきに色付けされていることを確認します。



No.	Date	Pay To	Memo	C	Payment	Deposit	Balance
	10/1/2008	Counter Credit		✓		$1,000.00	$1,000.00
501	10/2/2008	Microsoft	New 30" LCD	✓	$599.99		$400.01
502	10/4/2008	Chillis	Lunch with Dave	✓	$50.99		$349.02

9. 次のように、<my:DataGrid> 開始タグにいくつかプロパティの指定を追加します。
   a. SelectionMode プロパティを「Extended」に設定します。
   b. SelectionUnit プロパティを「FullRow」に設定します。
   c. GridLinesVisibility プロパティを「All」に設定します。
   d. VerticalGridLinesBrush プロパティを「DarkGray」に設定します。

XAML

```
<my:DataGrid x:Name="dg" ItemsSource="{Binding}" Margin="10"
          AutoGenerateColumns="False"
          Background="#80909090" AlternationCount="2"
          ColumnHeaderStyle="{StaticResource dgHeaderStyle}"
          RowStyle="{StaticResource dgRowStyle}"
          SelectionMode="Extended"
          SelectionUnit="FullRow"
          GridLinesVisibility="All"
          VerticalGridLinesBrush="DarkGray">
```

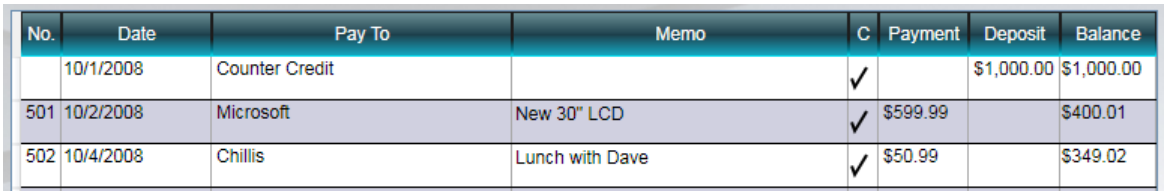
10. ヘッダーのスタイル (dgHeaderStyle) にも、いつくか追加の設定をしましょう。
   a. BorderThickness プロパティを「1」に設定して、枠が見えるようにします。
   b. ピクセル単位の位置指定を有効にします (SnapsToDevicePixcels プロパティを「True」にします)。
   c. コンテンツを水平方向に関して、センタリングします (HorizontalContentAlignment プロパティを「Center」にします)。
   d. MinWidth プロパティと MinHeight プロパティを、それぞれ「0」と「30」にします。
   e. Cursor プロパティを「Hand」にします。

XAML

```
<Style x:Key="dgHeaderStyle" TargetType="my:DataGridColumnHeader">
   <Setter Property="Background" Value="{StaticResource dgHeaderBrush}" />
   <Setter Property="Foreground" Value="White" />
   <Setter Property="BorderBrush"
           Value="{StaticResource dgHeaderBorderBrush}" />
   <Setter Property="BorderThickness" Value="1" />
   <Setter Property="SnapsToDevicePixels" Value="True" />
```

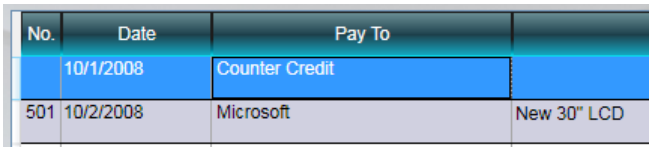
```
    <Setter Property="HorizontalContentAlignment" Value="Center" />
    <Setter Property="MinWidth" Value="0" />
    <Setter Property="MinHeight" Value="30" />
    <Setter Property="Cursor" Value="Hand" />
</Style>
```

ここでアプリケーションを実行すると、ヘッダーの周りに枠線が追加されたことが分かります。また、ヘッダーの上にマウスポインタを移動すると、マウスポインタの形状が手の形になります。(ヘッダーをクリックすると、その列をキーにしてソートをすることができます。)



No.	Date	Pay To	Memo	C	Payment	Deposit	Balance
	10/1/2008	Counter Credit		✓		$1,000.00	$1,000.00
501	10/2/2008	Microsoft	New 30" LCD	✓	$599.99		$400.01
502	10/4/2008	Chillis	Lunch with Dave	✓	$50.99		$349.02

11. その他、セル自体にもスタイルを適用できます。コンテンツのセンタリグや入力フォーカスを得た場合のトリガを適用するとき、グローバルなスタイルを使用すると便利です。ここでアプリケーションを実行し、現時点では、セルの選択時の枠線がセルの枠線にほぼ重なっており、データもセルの上部に寄せられていることを確認します。



No.	Date	Pay To	
	10/1/2008	Counter Credit	
501	10/2/2008	Microsoft	New 30" LCD

セルのスタイルを適用させる方法で、これらを調整することにします。

12. <DockPanel.Resources> セクションの中に、新しいスタイルを追加します。現在の行のスタイル (dgRowStyle)のすぐ下に追加するのが適当でしょう。
    a. このスタイルの TargetType プロパティを「my:DataGridCell」にします。
    b. 既存の表示形式の多くを引き継ぎたいので、BaseOn プロパティに「{StaticResource {x:Type my:DataGridCell}}」と指定して、DataGridCell スタイルを引き継ぎます。
    c. SnapsToDevicePixels プロパティを「True」に設定します。
    d. VerticalAlignment プロパティを 「Center」に設定します。
    e. Triggers コレクション <Style.Triggers> セクションを追加します。これを利用して、セルが選択された際に、背景色や前景色が変化できるようになります。
    f. <Style.Triggers> セクションでは、IsSelected プロパティが True になった場合に発生するトリガを追加します。

i. Background プロパティを「Transparent」に設定します。

ii. BorderBrush プロパティを「Transparent」に設定します。

iii. Foreground プロパティを「Black」に設定します。

g. IsKeyboardFocusWithin プロパティが True になった場合に発生するトリガを追加します。

i. Background プロパティを「{StaticResource whiteBackBrush}」というリソースに設定します。

ii. BorderBrush プロパティを既定のブラシ「{DynamicResource {x:Static my:DataGrid.FocusBorderBrushKey}}」に設定します。

iii. Foreground プロパティを「Black」に設定します。

h. コードが完成したら、スタイル dbCellStyle のコードが次のようになっていることを確認します。

XAML

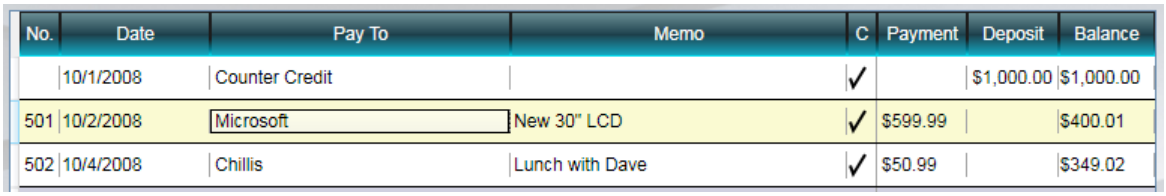
```
<Style x:Key="dgCellStyle" TargetType="my:DataGridCell"
       BasedOn="{StaticResource {x:Type my:DataGridCell}}">
   <Setter Property="SnapsToDevicePixels" Value="True" />
   <Setter Property="VerticalAlignment" Value="Center" />
   <Style.Triggers>
      <Trigger Property="IsSelected" Value="True">
         <Setter Property="Background" Value="Transparent" />
         <Setter Property="BorderBrush" Value="Transparent" />
         <Setter Property="Foreground" Value="Black" />
      </Trigger>
      <Trigger Property="IsKeyboardFocusWithin" Value="True">
         <Setter Property="Background"
                 Value="{StaticResource whiteBackBrush}" />
         <Setter Property="BorderBrush"
                 Value="{DynamicResource {x:Static
                        my:DataGrid.FocusBorderBrushKey}}" />
         <Setter Property="Foreground" Value="Black" />
      </Trigger>
   </Style.Triggers>
</Style>
```

13. 次のように、DataGrid の CellStyle プロパティにリソースを適用し、アプリケーションを実行します。

XAML

```
<my:DataGrid x:Name="dg" ItemsSource="{Binding}" Margin="10"
             AutoGenerateColumns="False"
             Background="#80909090" AlternationCount="2"
             ColumnHeaderStyle="{StaticResource dgHeaderStyle}"
             RowStyle="{StaticResource dgRowStyle}"
             CellStyle="{StaticResource dgCellStyle}"
             SelectionMode="Extended"
             SelectionUnit="FullRow"
             GridLinesVisibility="All"
             VerticalGridLinesBrush="DarkGray">
```

14. アプリケーションが起動したら、セルをクリックしてみます。セルだけがハイライト表示され、選択時のブラシの色も変更されたことを確認します。



No.	Date	Pay To	Memo	C	Payment	Deposit	Balance
	10/1/2008	Counter Credit		✓		$1,000.00	$1,000.00
501	10/2/2008	Microsoft	New 30" LCD	✓	$599.99		$400.01
502	10/4/2008	Chillis	Lunch with Dave	✓	$50.99		$349.02

もう一つ DataGrid に追加可能なものとして、RowDetails セクションがあります。このセクションは、行(Row)の直下に位置する表示領域であり、その行を選択したときに表示させることができます。(常に表示させるか、まったく表示させないかの指定もあります。) RowDetailsVisibilityMode プロパティを介して、これを有効化することができ、RowDetailsTemplate プロパティに DataTemplate を設定することで、その領域をどう表示するかを指定できます。RowDetailsVisibilityMode プロパティに設定できる値は、「Visible」、「VisibleWhenSelected」、または「Collapsed」です。この値は、実行時にも変更することができ、トリガを使用して動的に設定もできます。ただしここでは、単にカテゴリを表示するために使用します。

15. 現在の表示では、RegisterTransaction データクラスの一部の情報が抜けています。それは、Category プロパティです。内部的なコードでは、このプロパティを既にサポートしているので、必要なことは UI に追加することだけです。そして、RowDetails はそれを表示する上で適当な場所です。次のように、DataGrid を修正します。
    a. RowDetailsVisibilityMode プロパティを「VisibleWhenSelected」に設定します。
    b. RowDetails プロパティに適用するため DataTemplate を追加します。DataTemplate の内部には、水平方向の StackPanel で包まれた TextBlock と TextBox を追加します。
    c. StackPanel の左マージンを 20 ピクセルに設定します。

XAML

```
<my:DataGrid x:Name="dg" ItemsSource="{Binding}" Margin="10"
      ...
    VerticalGridLinesBrush="DarkGray"
    RowDetailsVisibilityMode="VisibleWhenSelected">

        <my:DataGrid.RowDetailsTemplate>
            <DataTemplate>
                <StackPanel Orientation="Horizontal" Margin="20,0,0,0">
                    <TextBlock />
                    <TextBox>
                    </TextBox>
                </StackPanel>
            </DataTemplate>
        </my:DataGrid.RowDetailsTemplate>
```

d. 追加した TextBlock の Text プロパティを「Category」に設定し、センタリングも行います。また、その領域の編集中のフォントが太字になるようにします。

XAML

```
<StackPanel Orientation="Horizontal" Margin="20,0,0,0">
   <TextBlock Text="Category:" VerticalAlignment="Center"
              FontWeight="Bold" />
   <TextBox>
   </TextBox>
```

e. 今度は、TextBox を修正します。Text プロパティと Category との値でバインディングを行います。Margin は「10,5」に設定し、最小の幅を 100 にします。

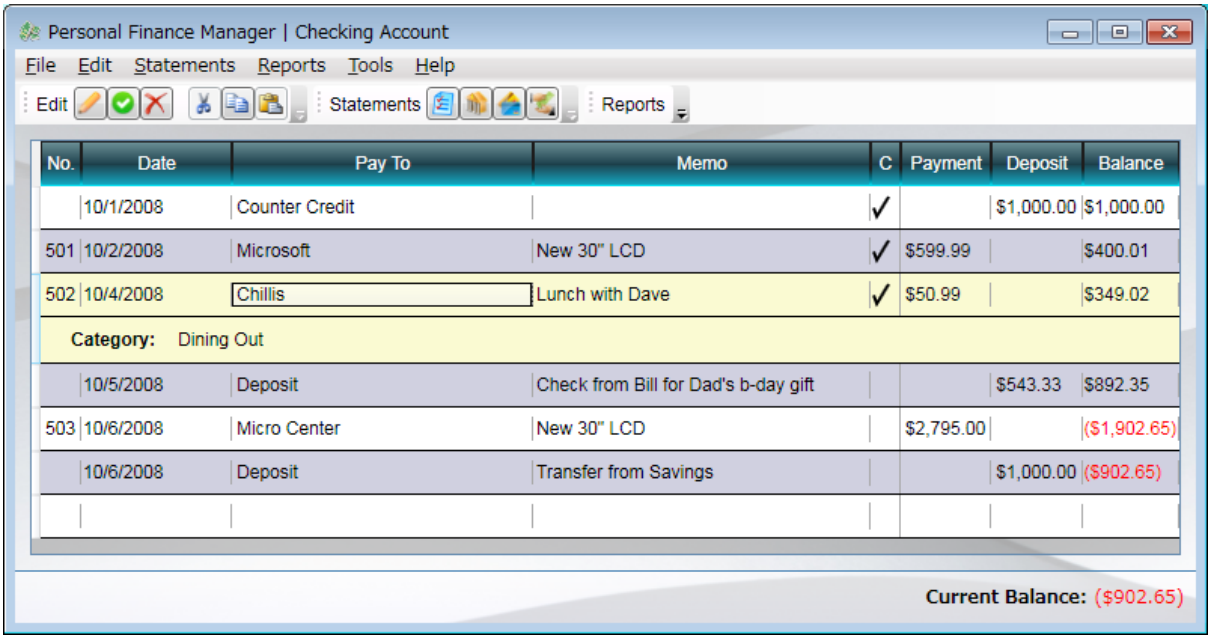
f. 最後に、TextBox にスタイルを追加して見栄えを良くします。特に、TextBox の上をマウスポインタが移動するか入力フォーカスを取得するまでは、枠線と背景色を削除します。これは、スタイルとトリガを使用して、実現することができます。次のコードのように、<TextBox> 要素ブロックの中に、<TextBox.Style> セクションを追加します。

XAML

```
<my:DataGrid.RowDetailsTemplate>
   <DataTemplate>
      <StackPanel Orientation="Horizontal" Margin="20,0,0,0">
         <TextBlock Text="Category:" VerticalAlignment="Center"
                    FontWeight="Bold" />
         <TextBox Text="{Binding Category}" Margin="10,5"
                  MinWidth="100">
            <TextBox.Style>
               <Style TargetType="TextBox">
                  <Setter Property="BorderBrush" Value="{x:Null}" />
                  <Setter Property="Background" Value="{x:Null}" />
                  <Style.Triggers>
                     <Trigger Property="IsFocused" Value="True">
                        <Setter Property="BorderBrush"
                                Value="{x:Static
                                SystemColors.WindowFrameBrush}" />
                        <Setter Property="Background"
```

```
                                    Value="{x:Static
                                    SystemColors.WindowBrush}" />
                        </Trigger>
                        <Trigger Property="IsMouseOver" Value="True">
                            <Setter Property="BorderBrush"
                                    Value="{x:Static
                                    SystemColors.WindowFrameBrush}" />
                            <Setter Property="Background"
                                    Value="{x:Static
                                    SystemColors.WindowBrush}" />
                        </Trigger>
                    </Style.Triggers>
                </Style>
            </TextBox.Style>
        </TextBox>
    </StackPanel>
</DataTemplate>
</my:DataGrid.RowDetailsTemplate>
```

16. ここでアプリケーションを実行し、行を選択してみます。RowDetails が行の下に動的に提供されることを確認します。「Category:」と表示されたラベルの右隣りに、マウスポインタを移動するか、またはクリックして入力フォーカスを取得するまでは、TextBox の部分が背景色と同じで、背景に溶け込んでいることを確認します。これは、前述のトリガを利用して、異なるプロパティの値に変化させている結果です。



これで、DataGrid を使用したアプリケーションが完成しました。この練習の完成品は、演習フォルダの中の、 Ex1_Solution_Task3 サブフォルダの中にあります。

まとめ

ここでは、以下の練習を行いました。

- 既存の ListView を新しい DataGrid に置き換えました。
- 編集可能な行を表示するために、DataGrid の新機能を使用しました。
- DataGrid の列ごとのデータバインディングのために、組み込みの列の型を使用し、様々なバインディングの表現方法を確認しました。
- ドロップダウン形式のカレンダを表示するために、DatePicker を使用しました。
- DataGrid にスタイルやトリガを利用して、きめ細かい見栄えの調整を行いました。

この演習で、WPF の新しい機能について、楽しんで学んで頂ければ幸いです。次は、これらのアイデアを活かして、ご自身のプロジェクトに適用してみてください。