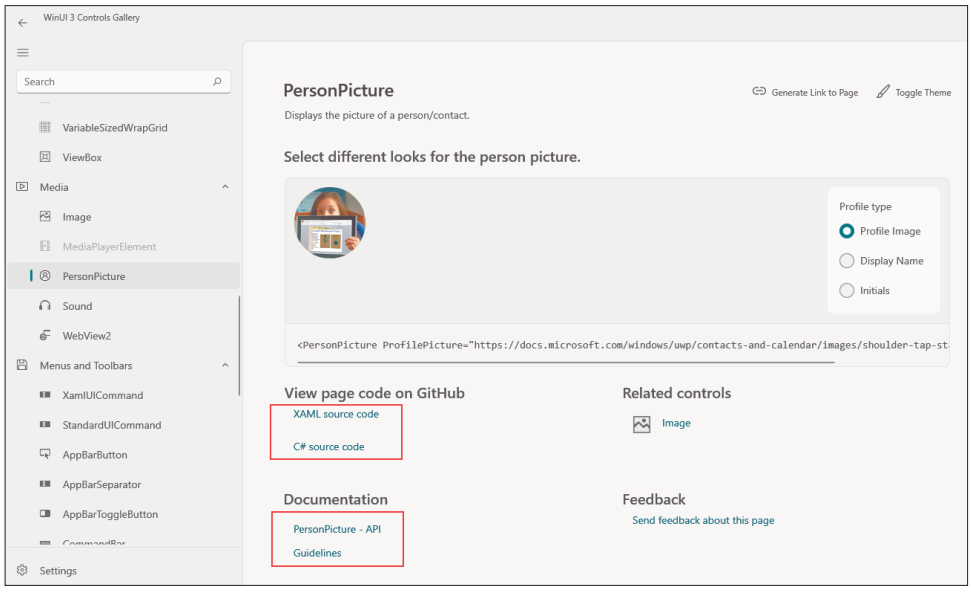
PROGRAMMIEREN MIT WINUI 3, TEIL 3

Der moderne Look

Welche Steuerelemente hat WinUI 3 zu bieten, um zeitgemäße Oberflächen zu gestalten? Wie setzt man sie ein, und lohnt sich das überhaupt?

Nach der Betrachtung der Grundlagen von WinUI 3 im ersten Teil [1] dieser Artikelserie haben wir im zweiten Teil [2] ein Rahmenprogramm für eine moderne Geschäftsanwendung geschaffen. Die Beispielanwendung verwendet das MVVM-Pattern (Model-View-ViewModel) für die Trennung zwischen UI und Bedienlogik und setzt auf Dependency Injection, um einen modularen Aufbau zu ermöglichen. Die Modularität wurde optisch bereits durch den Einsatz des *NavigationView*-Controls repräsentiert, dessen darzustellende Menüstruktur über eine im Code definierte Datenstruktur beschrieben wird.



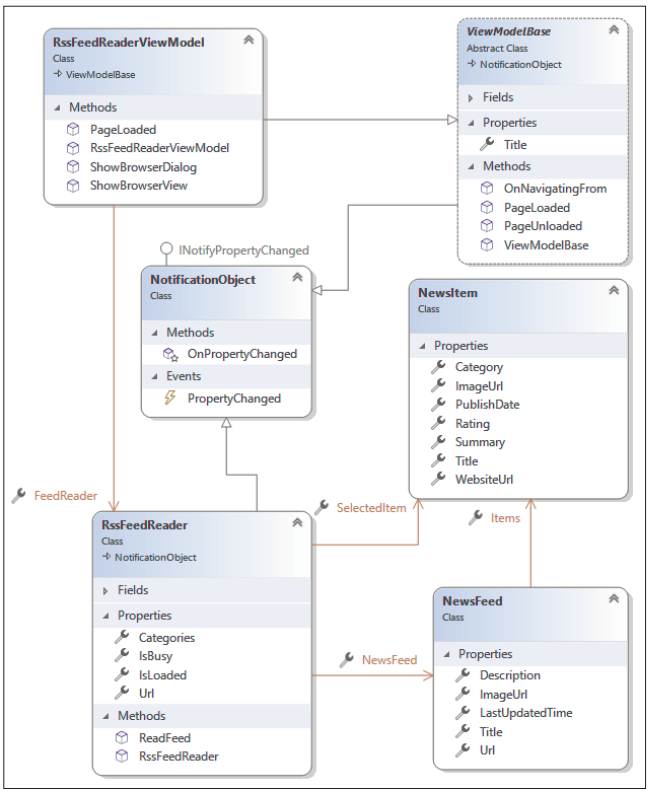
Die WinUI 3 Controls Gallery gibt einen guten Überblick über die vorhandenen Steuerelemente (Bild 1)

Einen guten Überblick über die vielen Steuerelemente, die WinUI 3 zu bieten hat, gibt die bereits erwähnte App WinUI 3 Controls Gallery (Bild 1), die Sie im Microsoft Store finden. Neben Anwendungsbeispielen finden Sie dort auch rudimentäre Beschreibungen sowie Links zum Sourcecode auf GitHub beziehungsweise zur Dokumentation. Bei den Links zum Sourcecode sollten Sie beachten, dass diese oft noch zum WinUI-2-Repository führen. Wechseln Sie in solchen Fällen dort manuell vom *master*-Branch zum *winui3*-Branch, um zur aktuellen Implementierung zu gelangen. Den Einstieg zum Repository finden Sie unter [3].

Nicht alle Steuerelemente sind für klassische Desktop-Anwendungen uneingeschränkt nutzbar. Viele sind eindeutig auf Touch-Bedienungen zugeschnitten, wie sie beispielsweise bei Apps für Tablets sinnvoll sind. Schließlich wurde die Bibliothek ursprünglich für das Umfeld der UWP-Anwendungen entwickelt.

Dennoch gibt es zahlreiche Controls, die auch in einer klassischen Desktop-Anwendung Sinn ergeben und die weit über das hinausgehen, was Microsoft im Lieferumfang von WPF (Windows Presentation Foundation) zu bieten hat. Anhand einiger Beispiele wollen wir eine Auswahl davon vorstellen.

Als Ausgangsbasis für die Beispiele dient eine Nachrichtenquelle in Gestalt eines RSS-Feeds. Die notwendige Infrastruktur in Form der Klasse *RssFeedReaderViewModel* zeigt



Komponenten des RssFeedReaderViewModel (Bild 2)

Listing 1: Klassische Darstellung des Feed-Readers mit einigen neuen WinUI-3-Controls

```
<Page
    ...>
  ...
    <ProgressRing  IsActive="{x:Bind ViewModel
      .FeedReader.IsBusy, Mode=OneWay}"/>
    <TextBlock Text="{x:Bind ViewModel.FeedReader
      .NewsFeed.Title, Mode=OneWay}" .../>
    <TextBlock Text="{x:Bind ViewModel.FeedReader
      .NewsFeed.Description, Mode=OneWay}" .../>
    <Image Source="{x:Bind ViewModel.FeedReader
      .NewsFeed.ImageUrl, Mode=OneWay}".../>
    <TextBlock Text="{x:Bind sys:String.Format('{0:g}',
      ViewModel.FeedReader.NewsFeed.LastUpdatedTime),
      Mode=OneWay}" .../>
    <ListBox SelectedItem="{x:Bind ViewModel
        .FeedReader.SelectedItem,Mode=TwoWay}"
      ItemsSource="{x:Bind ViewModel.FeedReader
        .NewsFeed.Items, Mode=OneWay}" ...>
      <ListBox.ItemTemplate>
        <DataTemplate x:DataType="m:NewsItem">
          <TextBlock Text="{x:Bind Title}"/>
        </DataTemplate>
      </ListBox.ItemTemplate>
    </ListBox>
    <Grid ... >
      <TextBlock Text="{x:Bind ViewModel.FeedReader
        .SelectedItem.Title, Mode=OneWay}" .../>
      <Image Source="{x:Bind ViewModel.FeedReader
        .SelectedItem.ImageUrl, Mode=OneWay}".../>
      <TextBlock Text="{x:Bind sys:String.Format(
        '{0:g}', ViewModel.FeedReader.SelectedItem
        .PublishDate), Mode=OneWay}" .../>
      <TextBlock Text="{x:Bind ViewModel.FeedReader
        .SelectedItem.Summary, Mode=OneWay}".../>
      <RatingControl Value="{x:Bind
          ViewModel.FeedReader.SelectedItem.Rating,
          Mode=TwoWay}"
        PlaceholderValue="2"
        Background="{StaticResource
          HyperlinkButtonBorderThemeBrush}" .../>

      <StackPanel ... >
        <HyperlinkButton
          Content="zur Webseite (extern)" ...
          NavigateUri="{x:Bind ViewModel.FeedReader
            .SelectedItem.WebsiteUrl, Mode=OneWay}"
          ToolTipService.ToolTip="{x:Bind ViewModel
            .FeedReader.SelectedItem.WebsiteUrl,
            Mode=OneWay}"/>
        <AppBarButton Icon="World"
          Label="Show in browser"
          Click="{x:Bind ViewModel.ShowBrowserView}"
          ToolTipService.ToolTip=
            "Show in a separate browser view"/>
        <AppBarButton Icon="World"
          Label="Show browser dialog"
          Click="{x:Bind ViewModel.ShowBrowserDialog}"
          ToolTipService.ToolTip=
            "Show browser in dialog window"/>
        <AppBarButton Label="Show in FlyOut"
            Icon="Send">
          <AppBarButton.Flyout>
            <Flyout>
              ...
              <StackPanel MinHeight="600"
                  MinWidth="800" >
                <TextBlock Text="Website:"/>
                <TextBlock Text="{x:Bind ViewModel
                  .FeedReader.SelectedItem.WebsiteUrl,
                  Mode=OneWay}"/>
                <WebView2 Source="{x:Bind ViewModel
                  .FeedReader.SelectedItem.WebsiteUrl,
                  Mode=OneWay}"
                    VerticalAlignment="Stretch"
                    HorizontalAlignment="Stretch"
                    MinHeight="600" MinWidth="800"/>
              </StackPanel>
            </Flyout>
          </AppBarButton.Flyout>
        </AppBarButton>
      </StackPanel>

    </Grid>
  </Grid>
</Page>
```

das Klassendiagramm in Bild 2. Dieses ViewModel wird für alle nachfolgend gezeigten Ansichten verwendet. Es bietet über die Eigenschaft *FeedReader* Zugriff auf die Informationen zum Nachrichtenkanal sowie auf die Nachrichten selbst. Ferner unterstützt es noch die Auswahl einer Nachricht sowie eine Kategorisierung. Die Daten für die Eigenschaften der Objekte vom Typ *NewsItem*, nämlich *Title*, *Summary*, *Category*, *PublishDate*, *WebsiteUrl* sowie *ImageUrl*, werden dem Feed entnommen. Die Eigenschaft *Rating* wurde nur zu Demonstrationszwecken ergänzt.

Die klassische Ansicht

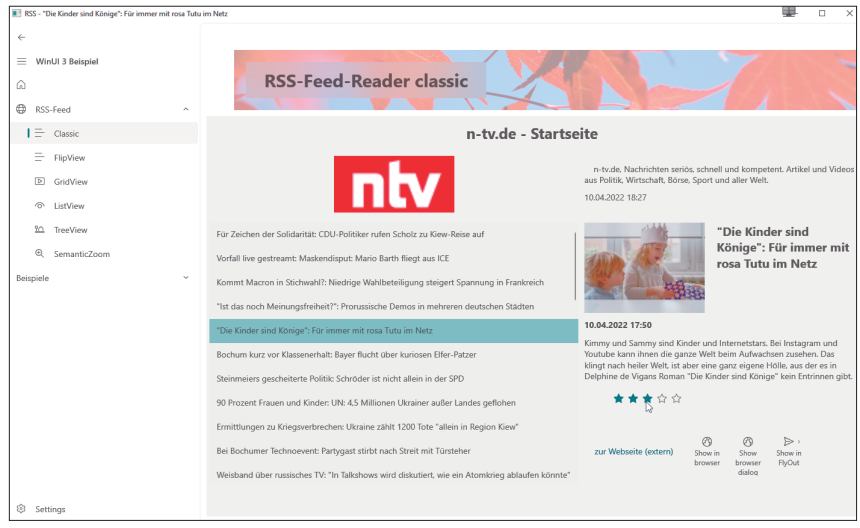
Eine typische Darstellung, wie man sie mit wenig Aufwand in WPF umgesetzt hätte, zeigt Bild 3. Für die Liste der Nachrichtentitel kommt eine ListBox zum Einsatz, während die Anzeige der Nachrichtendetails über Textblöcke und ▶

Image-Controls erfolgt. Listing 1 zeigt einen solchen Ansatz. Statt mit einer Sanduhr wird die Zeit bis zum vollständigen Laden der Feed-Daten mit einem Steuerelement vom Typ *ProgressRing* angezeigt. Das *RatingControl* ist an die Eigenschaft *Rating* gebunden und zeigt deren Zahlenwert (0…5) als Sternchen an. Es lässt aber auch Benutzereingaben zum Beispiel über Mausklicks zu. Änderungen werden über eine Zweiwege-Bindung in der Property gespeichert und können von den anderen Darstellungen wieder abgerufen werden.

Möchte man einen Button analog zu einer Webseite als Hyperlink darstellen, kann man sich wie bei WPF eines TextBlocks bedienen und den Hyperlink als Unterelement hinzufügen. WinUI 3 hat aber auch ein dediziertes Steuerelement hierfür: *HyperlinkButton*. Setzt man dessen Eigenschaft *NavigateUri* auf einen gültigen *Url* (im Beispiel über eine Datenbindung), so wird automatisch der Standardbrowser des Betriebssystems mit der Darstellung der Webseite beauftragt.

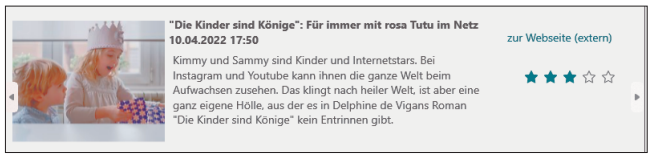
Schaltflächen für eine Toolbar lassen sich leicht mit *AppBarButton*-Controls erstellen. Sie können hier auf fertige Icons aus WinUI 3 zurückgreifen oder eigene hinzufügen. Der *Click*-Event lässt sich wie bereits gezeigt über eine Datenbindung mit einer Methode des ViewModels verknüpfen. Tooltipps können ganz allgemein über Attached Dependency Properties der Klasse *ToolTipService* eingerichtet werden.

Eine andere Art des Tooltipps bietet die Eigenschaft *Flyout* in Verbindung mit der gleichnamigen Klasse. Das hierüber angezeigte Pop-up-Fenster dient nur der Ansicht und lässt keinerlei Bedienung zu (außer zum Beispiel den Bildlauf per Mausrad). Wie bereits im zweiten Teil dieser Artikelserie für den Fall von Dialogfenstern beschrieben, gibt es hier aber auch wieder interne Einschränkungen bezüglich der Fenstergröße: Diese lässt sich nicht beliebig einstellen.



Klassische Darstellung mit ListBox und Detailbereich (Bild 3)

Der AcrylicBrush mit Weichzeichnereffekt (Bild 4)



Eine FlipView ermöglicht die Navigation zu benachbarten Nachrichtenelementen (Bild 5)

Dank des neuen Steuerelements *WebView2* lässt sich die Anzeige von Webseiten sehr einfach integrieren. Das Control basiert auf Chromium, und damit sind die diversen Unzulänglichkeiten des alten, ActiveX-basierten IE-Controls endlich Vergangenheit. Eine simple Implementierung einer View, die ebenfalls mit dem oben beschriebenen *RssFeedReaderViewModel* verknüpft ist, zeigt Listing 2.

Ein neues Stilelement, das die Anzeige von Text auf Hintergründen wie Fotos aufhübschen kann, ist *AcrylicBrush* aus dem Namensraum *Microsoft.UI.Xaml.Media*. Listing 3 zeigt

Listing 2: Das Steuerelement WebView2

```
<Page...>

  <Grid>
    <WebView2 Source="{x:Bind ViewModel.FeedReader
        .SelectedItem.WebsiteUrl}"
      VerticalAlignment="Stretch"/>
  </Grid>
</Page>
```

Listing 3: Ein Brush zum Weichzeichnen

```
<Grid Margin="20">
  <Grid.Background>
    <media:AcrylicBrush TintColor="LightBlue"
                        TintLuminosityOpacity="0.5"
                        TintOpacity="0.2"/>
  </Grid.Background>
  <TextBlock Text="{Binding  Mode=OneWay}" ... />
</Grid>
```

Listing 4: Mit der FlipView von einer Nachricht zur nächsten

```
<FlipView ItemsSource="{x:Bind ViewModel.FeedReader
      .NewsFeed.Items,Mode=OneWay}"
    SelectedItem="{x:Bind ViewModel.FeedReader
      .SelectedItem, Mode=TwoWay}"
    ...>
  <FlipView.ItemTemplate>
    <DataTemplate x:DataType="m:NewsItem">
      <Grid Width="800" Height="200">
        ...
        <TextBlock Text="{x:Bind Title}" ..."/>
        <Image Source="{x:Bind ImageUrl}" .../>
        <TextBlock Text="{x:Bind sys:String.Format(
          '{0:g}',  PublishDate)}" ./>
        <TextBlock Text="{x:Bind Summary}" .../>
        <StackPanel ...">

          <HyperlinkButton
            Content="zur Webseite (extern)" ...
            NavigateUri="{x:Bind WebsiteUrl}"
            ToolTipService.ToolTip="{x:Bind
              WebsiteUrl}"/>
          <RatingControl
            Value="{x:Bind Rating, Mode=TwoWay}"
            PlaceholderValue="2"
            Background="{StaticResource
              HyperlinkButtonBorderThemeBrush}" .../>

        </StackPanel>
      </Grid>
    </DataTemplate>

  </FlipView.ItemTemplate>

</FlipView>
```

den Einsatz als Background des Titelbereichs des Hauptfensters, Bild 4 präsentiert ein Beispiel für die Darstellung. Farbe und Deckkraft der Füllung lassen sich über Eigenschaften einstellen. Auf den Hintergrund wird ein Weichzeichner angewendet, was den Kontrast und somit die Lesbarkeit der Vordergrundelemente (hier der Schrift) verbessert.

Ausgeflippt

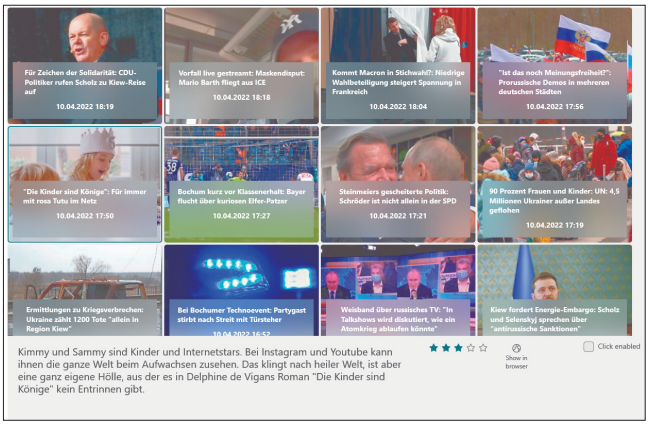
Ein Steuerelement, das eigentlich eher zum Blättern in einer Liste von Fotos gedacht ist, ist *FlipView*. Sein Inhalt lässt sich jedoch, wie in Bild 5 zu sehen, beliebig über ein DataTemplate gestalten.

Controls wie hier das *RatingControl* sind bedienbar, über die automatisch angezeigten Schaltflächen links und rechts kann der Benutzer zum vorherigen beziehungsweise zum nächsten Listenelement navigieren. In Listing 4 sehen Sie die Umsetzung im XAML-Code. Das Template beschreibt die Darstellung einer einzelnen Nachricht. Es liegt in der Natur dieses Steuerelements, dass es immer nur genau eine Nachricht anzeigen kann. Einen Überblick über die gesamte Liste bietet es nicht.

Die Listendarstellung beherrscht hingegen das *GridView*-Steuerelement (Bild 6). Es kann die einzelnen Listenelemente übersichtlich und vollautomatisch in Tabellenform anord- ▶

Listing 5: Die GridView lässt sich recht einfach einbinden

```
<GridView
    ItemsSource="{x:Bind ViewModel.FeedReader.NewsFeed
      .Items, Mode=OneWay}"
    IsItemClickEnabled="{x:Bind CBClickEnabled
      .IsChecked.Value, Mode=OneWay}"
    ItemClick="{x:Bind ViewModel.ShowBrowserView}"
    SelectedItem="{x:Bind ViewModel.FeedReader
      .SelectedItem, Mode=TwoWay}"
    SelectionMode="Single"
    CanDragItems="True" CanReorderItems="True"
      AllowDrop="True"
    Grid.Row="1">
  <GridView.ItemTemplate>
    <DataTemplate x:DataType="m:NewsItem">
      <Grid Width="300" >
        <Image Source="{x:Bind ImageUrl}"
          Stretch="UniformToFill" Width="300"/>
        <StackPanel  Margin="10" Height="110"
          VerticalAlignment="Bottom">
          <StackPanel.Background>
            <media:AcrylicBrush TintColor="LightBlue"
              TintLuminosityOpacity="0.5"
              TintOpacity="0.2"/>
          </StackPanel.Background>
          <TextBlock Text="{x:Bind Title}" ... />
          <TextBlock Text="{x:Bind sys:String.Format(
            '{0:g}', PublishDate), Mode=OneWay}" .../>
        </StackPanel>
      </Grid>
    </DataTemplate>
  </GridView.ItemTemplate>
</GridView>
```



Übersichtliche Anordnung der Nachrichten in einem GridView-Control (Bild 6)

nen. Wie Listing 5 zu entnehmen ist, reichen hierfür die Bindung an die Datenquelle und die Definition eines passenden DataTemplates vollkommen aus. Über die Eigenschaft *IsItemClickEnabled* lässt sich steuern, ob ein Klick auf einen Eintrag diesen selektieren soll oder ob der *ItemClick*-Event ausgelöst werden soll.

Sind die Eigenschaften *CanDragItems* sowie *CanReorderItems* auf *True* gesetzt, kann der Benutzer auf einfache Weise ein Listenelement verschieben und an einer anderen Position einfügen. Bild 7 deutet diesen Vorgang an. Das Control ändert dabei selbstständig die Reihenfolge in der gebundenen Liste. Besondere Aktionen im Code sind hierzu nicht erforderlich.

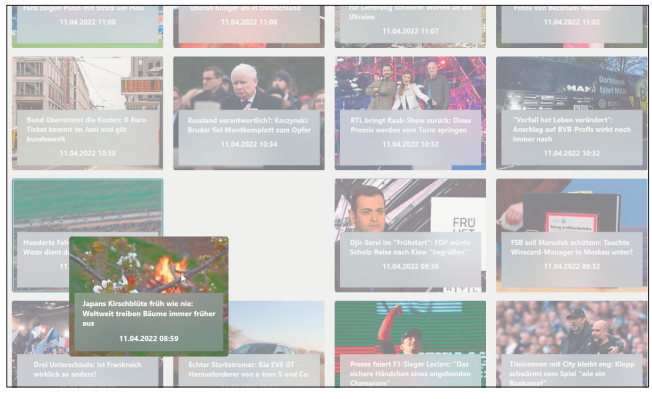
Sehr positiv fällt hierbei auf, dass die Zieh-Vorgänge sehr smart animiert werden. Einträge unterhalb des gezogenen Elements weichen automatisch zur Seite und geben den Platz frei. So etwas mit WPF nachzubauen ist sicherlich möglich, erfordert jedoch eine Menge Fleißarbeit und Geschick im Umgang mit Animationen. Schön, dass es hier eine fertige Lösung gibt.

Das Template des Beispiels benutzt wiederum den *AcrylicBrush*, um die Textanzeigen über dem jeweiligen Foto der Nachricht zu platzieren. So passt sich der Texthintergrund farblich automatisch an das jeweilige Bild an.

Hierarchien darstellen

Einige Steuerelemente ermöglichen die Darstellung von Gruppierungen. Als Beispiel sehen Sie in Bild 8 die Nachrichtenliste, dargestellt in einem *ListView*-Control. Die Kategorien werden hier als Gruppentitel wiedergegeben. Ein eigens hierfür zu definierendes *HeaderTemplate* (siehe Listing 6) beschreibt dessen Darstellung. Im Beispiel ist es nur ein simpler TextBlock, aber grundsätzlich sind der Kreativität hier keine Grenzen gesetzt.

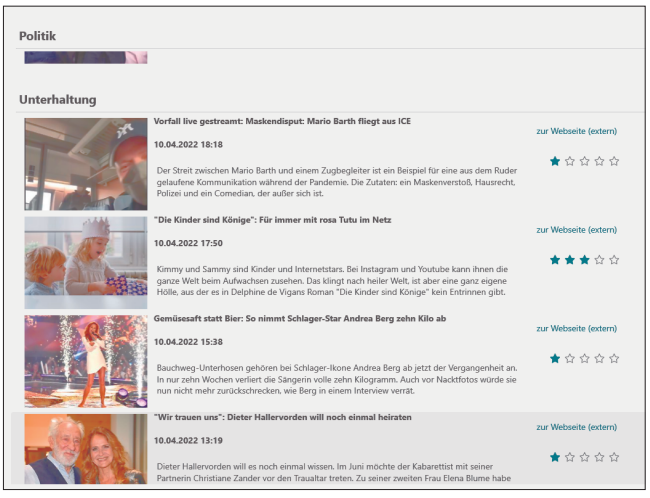
Zum Zeitpunkt der Erstellung der Beispiele war die Dokumentation noch recht lückenhaft und der anscheinend einzige Weg, das *ListView*-Control zum Anzeigen der Gruppierungen zu bewegen, war, wie in einem der Microsoft-Beispiele



In der GridView können Elemente verschoben werden (Bild 7)

Listing 6: Gruppierte Listendarstellung mit ListView und CollectionViewSource

```
<Page
    ...>
  <Page.Resources>
    <CollectionViewSource Source="{x:Bind
        ViewModel.FeedReader.Categories, Mode=OneWay}"
      IsSourceGrouped="True"
      x:Name="NewsCVS"/>
  </Page.Resources>

  <Grid>...

    <ListView ItemsSource="{Binding Source=
        {StaticResource NewsCVS}}" ...>
      <ListView.ItemTemplate>
        <DataTemplate x:DataType="m:NewsItem">
        ...
        </DataTemplate>
      </ListView.ItemTemplate>
      <ListView.GroupStyle>
        <GroupStyle>
          <GroupStyle.HeaderTemplate>
            <DataTemplate >
              <TextBlock Text="{Binding Key}"
                Margin="5,30,0,0" FontWeight="Bold"/>
            </DataTemplate>
          </GroupStyle.HeaderTemplate>
        </GroupStyle>
      </ListView.GroupStyle>
    </ListView>

  </Grid>
</Page>
```



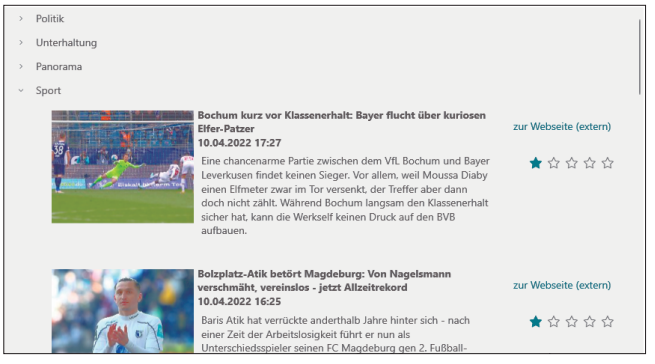
Kategorien im ListView-Control (Bild 8)

zu finden, über ein *CollectionViewSource*-Element, wie Sie es dem Listing entnehmen können. Ob es hier wie bei WPF auch noch andere Möglichkeiten gibt, die ohne *CollectionViewSource* auskommen, blieb offen.

CollectionViewSource wirkt sehr altbacken und erinnert an die Anfangszeit von WPF, als man dort auch auf derartige Hilfskonstrukte angewiesen war.

Für Hierarchien mit beliebig vielen Ebenen greift man gewöhnlich auf ein *TreeView*-Control zurück. Auch das steht in WinUI 3 zur Verfügung und kann für die Darstellung der Nachrichtenliste, nach Kategorien sortiert, eingesetzt werden (Bild 9). Anders als bei WPF kommt hier allerdings kein *HierachicalDataTemplate* zum Einsatz. Vielmehr muss man die Hierarchien selbst über normale Templates abbilden, die ihrerseits wieder *TreeViewItem*-Elemente definieren, die an die jeweiligen Sublisten gebunden werden (siehe Listing 7). Auch muss man für Titel und Inhalt unterschiedliche Templates definieren, die über ein Objekt vom Typ *DataTemplateSelector* (siehe Listing 8) ausgewählt werden können. Das mutet im Vergleich zu WPF etwas sperrig an, funktioniert aber dennoch ganz gut.

Deutlich auf die Touch-Bedienung zugeschnitten ist das Steuerelement *SemanticZoom*. Hier kann der Benutzer durch Anklicken der Kategorie umschalten zwischen einer Liste, ▶



Einsatz eines TreeView-Controls (Bild 9)

Listing 7: Eine TreeView und ihre Templates

```
<Page ...>
  <Page.Resources>

    <DataTemplate x:Key="CategoryTemplate">
      <TreeViewItem Content="{Binding Key}" ItemsSource="{Binding}"/>
    </DataTemplate>

    <DataTemplate x:Key="NewsItemTemplate" x:DataType="m:NewsItem">
      ...
    </DataTemplate>

    <vh:RssFeedTemplateSelector x:Key="NewsTreeTemplateSelector"
                                CategoryTemplate="{StaticResource CategoryTemplate}"
                                NewsItemTemplate="{StaticResource NewsItemTemplate}"/>
  </Page.Resources>

  <Grid>
    <TreeView ItemsSource="{x:Bind ViewModel.FeedReader.Categories, Mode=OneWay}"
              ItemTemplateSelector="{StaticResource NewsTreeTemplateSelector}"
              CanDragItems="False" AllowDrop="False" CanDrag="False"/>

  </Grid>
</Page>
```

Listing 8: Auswahl der Templates in der TreeView

```
public class RssFeedTemplateSelector :
    DataTemplateSelector
{
  public DataTemplate CategoryTemplate { get; set; }
  public DataTemplate NewsItemTemplate { get; set; }

  protected override DataTemplate
      SelectTemplateCore(object item)
  {
    if (item is NewsItem)
      return NewsItemTemplate;
    return CategoryTemplate;
  }
}
```

in der nur die Kategorien dargestellt werden, und einer Darstellung wie bereits bei der ListView gezeigt. Klickt man in der Kategorienliste auf einen Eintrag, zoomt man sich sozusagen in die andere Darstellung hinein, daher der Name. In Wirklichkeit wird aber eigentlich nur die Position in der zweiten Darstellung so gewählt, dass die geklickte Kategorie ganz oben steht. Klickt man in der ausführlicheren Darstellung auf einen Header, kommt man zur Kategorienliste zurück. Animationen zwischen den beiden Ansichten sind bereits inklusive. Bild 10 zeigt beispielhaft die beiden Darstellungen.

Gesteuert wird das Verhalten über eine Reihe von Templates (Listing 9). Für die Darstellung *ZoomedOutView* wird ein Template benötigt, das die Kategorienliste darstellt, für *ZoomedInView* hingegen ein Template für den Header sowie ein Template für die Darstellung eines Listeneintrags. Auch dieses Control braucht wieder eine *CollectionViewSource* für die Gruppierung.

Datum und Uhrzeit

Steuerelemente wie *DatePicker*, *CalendarDatePicker* oder *TimePicker* erlauben die Eingabe von Datum beziehungsweise Uhrzeit. Beispiele sehen Sie in Bild 11.

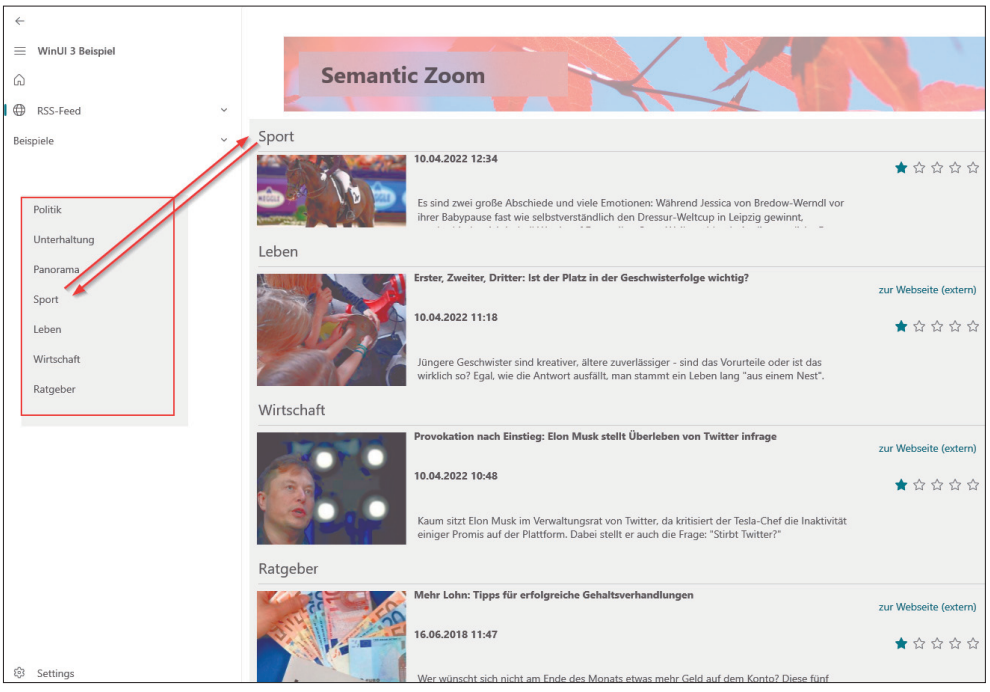
Die *Date*-Eigenschaft des DatePicker-Controls bindet man am besten an eine Eigenschaft vom Typ *DateTimeOffset*, die entsprechende Eigenschaft *Time* des TimePickers an eine Eigenschaft vom Type *TimeSpan*. Die Darstellung kommt Ihnen bekannt vor? Ähnlichkeiten mit den Steuerelementen eines Microsoft-Browsers sind sicher rein zufällig.

Datagrids

Sie suchen das ultimative Datagrid zur Darstellung umfangreicher Daten in WinUI 3? Dann schauen Sie mal in der Win-

Listing 9: Auch SemanticZoom benötigt eine Reihe von Templates

```
<Page ...>
  <Page.Resources>
    <CollectionViewSource Source="{x:Bind
        ViewModel.FeedReader.Categories, Mode=OneWay}"
      IsSourceGrouped="True"
      x:Name="NewsCVS"/>

    <DataTemplate x:Key="ZoomedInGroupHeaderTemplate" >
      <TextBlock Text="{Binding Key}"/>
    </DataTemplate>

    <DataTemplate x:Key="ZoomedInTemplate"
        x:DataType="m:NewsItem">
      ...
    </DataTemplate>

    <DataTemplate x:Key="ZoomedOutTemplate" >
      <TextBlock Text="{Binding Group.Key}"/>
    </DataTemplate>

  </Page.Resources>

  <Grid>
    <SemanticZoom Height="800">
      <SemanticZoom.ZoomedInView>
        <GridView
          ItemsSource="{x:Bind NewsCVS.View,
            Mode=OneWay}"
          ScrollViewer.IsHorizontalScrollChaining
            Enabled="False"
          SelectionMode="None" ItemTemplate=
            "{StaticResource ZoomedInTemplate}">
          <GridView.GroupStyle>
            <GroupStyle HeaderTemplate=
              "{StaticResource
              ZoomedInGroupHeaderTemplate}"/>
          </GridView.GroupStyle>
        </GridView>
      </SemanticZoom.ZoomedInView>
      <SemanticZoom.ZoomedOutView>
        <ListView
          ItemsSource="{x:Bind
            NewsCVS.View.CollectionGroups, Mode=OneWay}"
          SelectionMode="None" ItemTemplate=
            "{StaticResource ZoomedOutTemplate}"/>
      </SemanticZoom.ZoomedOutView>
    </SemanticZoom>

  </Grid>
</Page>
```



SemanticZoom ermöglicht das Umschalten zwischen zwei Darstellungen (Bild 10)

UI 3 Controls Gallery unter *Collection/DataGrid* nach. Dort finden Sie – leider – nur einen Verweis auf das Community Toolkit (siehe [4]). Das scheint aber bislang eher noch auf dem Stand von UWP zu sein und wird wohl erst im Lauf der Zeit nachgezogen. Besser wird es vermutlich sein, auf einen der bekannten Drittanbieter zurückzugreifen und sich die passende Toolbox einzukaufen. Für den Einsatz auf Tablets, insbesondere mit Touch-Bedienung, sollten Sie aber besser auf überladene Tabellendarstellungen verzichten, da diese dort kaum bedienbar sind.

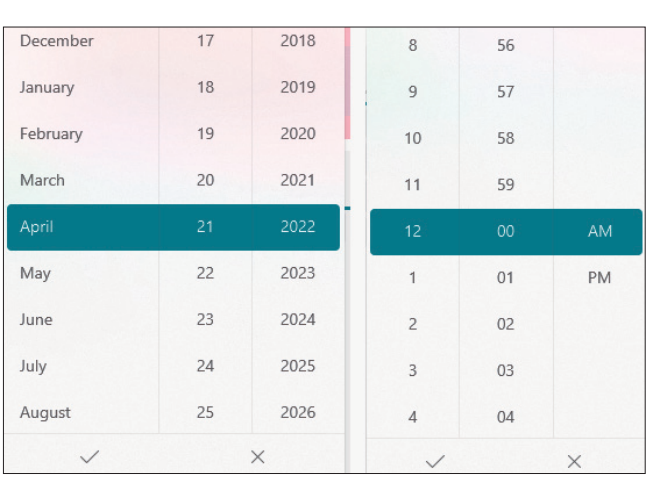
Fazit

Im Gegensatz zu WPF hat WinUI 3 bereits eine Vielzahl attraktiver Steuerelemente mit an Bord. Wegen der Nähe zu UWP sind viele davon auf Touch-Bedienungen zugeschnitten. Die meisten lassen sich aber auch in Desktop-Anwendungen integrieren. Die Steuerelemente sehen gut aus, machen einen modernen Eindruck und kommen oft bereits mit fertigen Animationen daher. Mechanismen wie beispielsweise das Verschieben von Elementen in der GridView, wie man sie sich sonst mühsam erarbeiten müsste, bekommt man geschenkt.

Die hier gezeigten Beispiele stellen nur einen sehr kleinen Ausschnitt der Möglichkeiten dar. Lassen Sie sich von der WinUI 3 Controls Gallery App inspirieren. Vielleicht kommen Sie hierdurch auch auf neue Ideen für Ihre Anwendungen.

Insgesamt lässt sich sagen, dass man auch mit WinUI 3 Geschäftsanwendungen entwickeln kann, ohne dabei auf Patterns wie MVVM oder Dependency Injection verzichten zu müssen. Nicht jeder Kniff aus WPF lässt sich anwenden, und manches mutet unnötig umständlich an, aber es gibt auch viele Highlights, die man bei WPF vergeblich sucht, mit viel Aufwand selbst programmieren müsste oder nur in spezialisierten Toolkits findet.

Aus Platzgründen sind im Artikel wie üblich nur Ausschnitte des Codes abgebildet. Die vollständige Solution finden Sie im GitHub-Repository unter [5]. ■



Steuerelemente für die Eingabe von Datum und Uhrzeit (Bild 11)

[1] Joachim Fuchs, UI zu gewinnen, Programmieren mit WinUI 3, Teil 1, dotnetpro 4/2022, Seite 68 ff., www.dotnetpro.de/A2204WinUI3

[2] Joachim Fuchs, WinUI 3 im Praxistest, Programmieren mit WinUI 3, Teil 2, dotnetpro 5/2022, Seite 34 ff., www.dotnetpro.de/A2205WinUI3

[3] WinUI 3 XAML Controls Gallery, www.dotnetpro.de/SL2206WinUI3_1

[4] Windows Community Toolkit, www.dotnetpro.de/SL2206WinUI3_2

[5] Sourcecode zum Beispielprogramm auf GitHub, www.dotnetpro.de/SL2206WinUI3_3

Dr. Joachim Fuchs
ist begeisterter Anhänger von Microsofts .NET-Philosophie. Er arbeitet als Softwarearchitekt, Berater und Dozent im Expertennetzwerk www.it-visions.de. Seine Schwerpunkte liegen derzeit bei XAML- und Web-UI-Technologien.
dnp@fuechse-online.de

dnpCode A2206WinUI3