Scheduler for WPF

2021.12.21 更新

グレープシティ株式会社

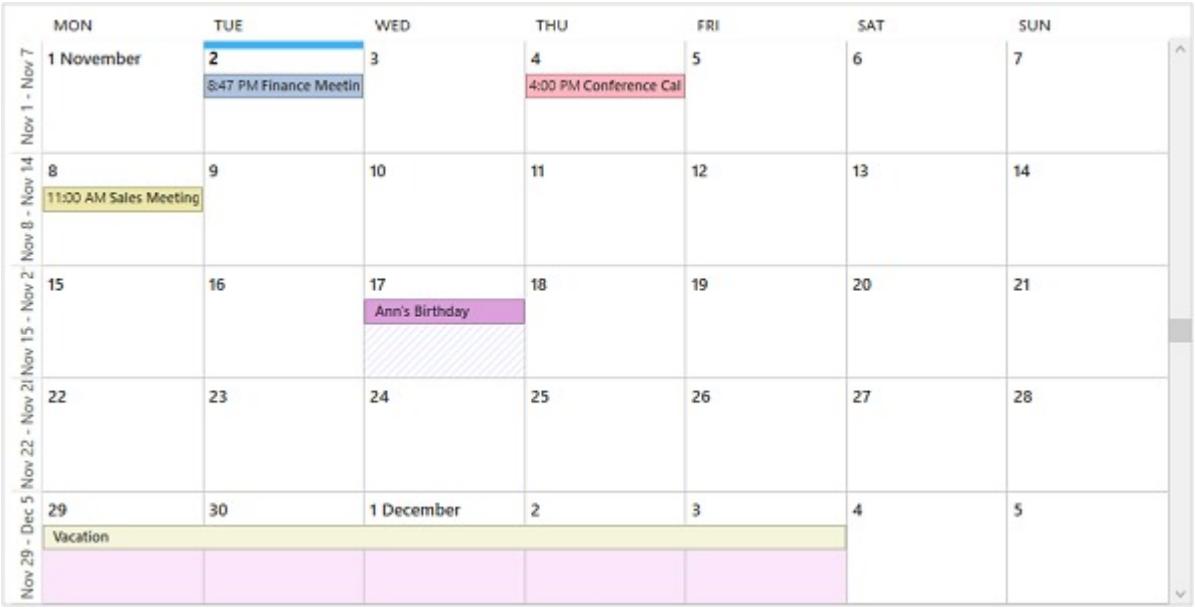
Scheduler for WPF

目次

Scheduler for WPF

Scheduler for WPF

Scheduler for WPFを使用すると、Outlook 形式のスケジュール機能をWPFアプリケーションに簡単に組み込むことができます。Scheduler is a fully functional Outlook-style schedule that allows users to add, edit, and manage their appointments easily.カスタマイズ可能なダイアログボックス、組み込みデータビュー、インポート/エクスポート機能などのオプションにより、スケジュールアプリケーションの設計が今までになく容易になっています。



Release Notes	**Product Samples**
See the version-wise updates for all controls here.	Product samples are located at Documents\ComponentOne Samples\WPF\vx.x\CS\Schedule or Documents\ComponentOne Samples\WPF\vx.x.x\C1.WPF.Schedule\CS on your system, if you have installed the samples while installing WPF Edition using **ComponentOneControlPanel.exe**.
Documentation	**Blogs**
Create your First Application with Scheduler Appointments Grouping End-User Interactions	Outlook-Style Scheduler Control for WPF CRUD operations in ComponentOne Scheduler Creating Multi-User Schedules with Grouping Using Google Calendar with C1Scheduler for WPF

Note: ComponentOne Scheduler for WPF is compatible with .NET 6, .NET 5 and .NET 4.5.2.

API References	
C1.WPF.Scheduler .NET 6 Assembly	**C1.WPF.Scheduler .NET Framework Assembly**

主な特長

Scheduler for WPF が備える便利な主要機能の一部を以下に示します。

- **複数の組み込みデータビューから選択する**

  C1Scheduler コントロールには、5つの組み込みデータビューがあります。このデータビューを使用することで、スケジュールをさまざまな方法で表示する機能をユーザーに提供できます。スケジュールを日、週、週間勤務日、または月単位で表示できます。 ビューの変更の詳細については、「データビュー」を参照してください。

- **カスタムビューを作成する**

  カスタムスケジュールビューを作成して使用することができます。

- **任意のデータソースにスケジュールを連結する**

  Scheduler for WPF には、標準の ADO.NET データ連結を使用する方法と、組み込みデータソースを使用する方法があります。C1ScheduleStorage コンポーネントと連携するデータソース設定を使用することで、テーブル内の各列にデータソースをマップし、予定、カテゴリ、連絡先、ラベル、リソース、予定の状態などを保存したりロードすることができます。

- **カスタマイズされたダイアログボックスの作成**

  Scheduler for WPF では、［予定］、 [繰り返し］、［アラーム］などのダイアログボックスに対して、独自の外観を設定できます。それには、Scheduler for WPF に付属するデフォルトのテンプレートをコピーし、目的の要素をカスタマイズします。空のテンプレートを作成したり、［予定］などのダイアログボックスを最初から設計することもできます。Scheduler for WPF テンプレートの詳細については、「C1Scheduler のデフォルトのスタイルとテンプレート」を参照してください。

- **Outlook 形式の予定の作成**

  C1Scheduler コントロール内で新しい予定の追加や既存の予定の編集を簡単に行えます。Microsoft Outlook と同様に、予定を 1 回だけ発生させたり、設定した時間が過ぎるまで何回も発生させることができます。また、予定を忘れないためにアラームを設定できます。さらに、**C1Scheduler** は、ユーザーが予定を管理しやすいように、12 個の組み込みラベルと 4 つの空き状況オプションのほか、カスタムラベルを作成する機能も提供しています。予定はカテゴリに分けて整理でき、各予定にリソースや連絡先を指定できます。

- **複数の書式のインポートとエクスポート**

  組み込みデータソースを使用する場合は、サポートされている書式（バイナリ、XML、iCal）の中からアプリケーションに適した書式でデータを保存またはロードできます。それには、C1ScheduleStorage の **Export** メソッドと **Import** メソッドを使用するか、**C1Scheduler** ルーティングコマンドの **ExportCommand** と **ImportCommand** を使用します。

クイックスタート

The following quick start guide is intended to get you up and running with **Scheduler for WPF**. In this quick start you'll create a new Expression Blend project, add Scheduler to your application, bind to a data source, and customize the schedule.

> **Note:** In this quick start guide you will use the C1NWind.mdb database installed by default to **Documents\ComponentOne Samples\Common**.

データソースを設定するには

1. Create a new **WPF App** in **Visual Studio**.
2. Drag and Drop the **C1Scheduler** control from the toolbox onto the **UI Designer Window**. The following references automatically get added to the **References**.
   - C1.WPF.Schedule
   - C1.WPF

   or

   Add the NuGet packages by following these steps:

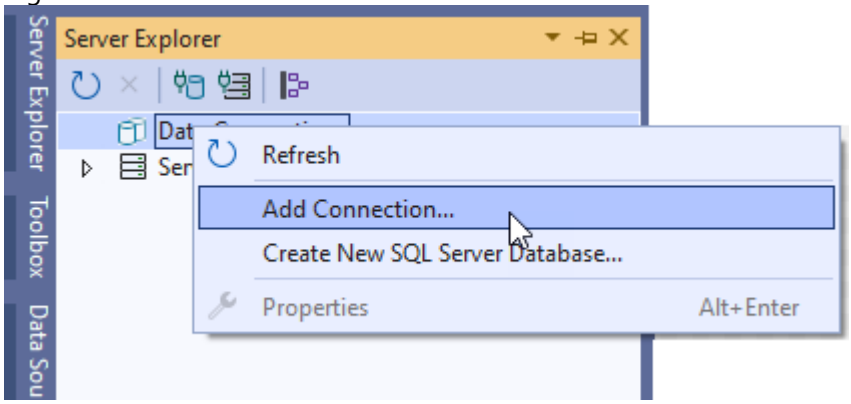
   1. In the **Solution Explorer**, right click **Dependencies** and select **Manage NuGet Packages**.
   2. In **NuGet Package Manager**, select **nuget.org** as the **Package source**.
   3. Search and select the following package and click **Install**.
      - **C1.WPF.Schedule**
3. In the **XAML Editor** and add name for the Scheduler control using the following code:

```
XAML
<c1:C1Scheduler x:Name="scheduler"></c1:C1Scheduler>
```

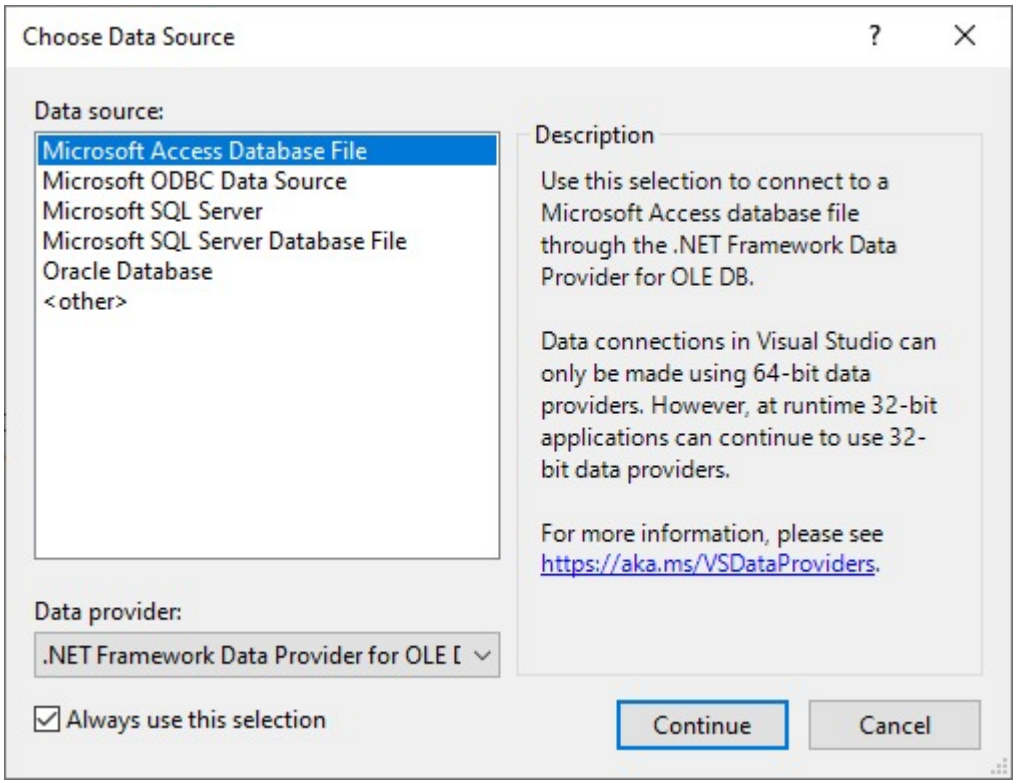
Back to Top

Configure the Data Source

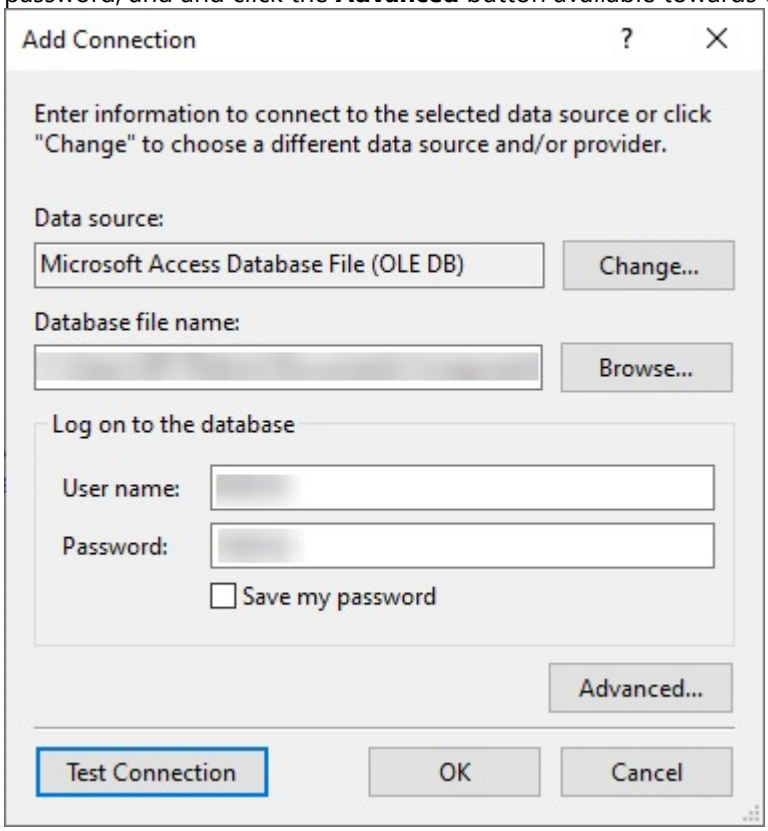
1. In the **Solution Explorer**, right-click on the project and select **Add | New Item**.
2. In the **Add New Item** dialog, select **Data** from the installed templates pane, select **DataSet** from the middle pane and click **Add**.
3. From the **View** menu, select **Server Explorer** to open it.
4. Right-click on the **DataCollections** and select **Add Connection**.



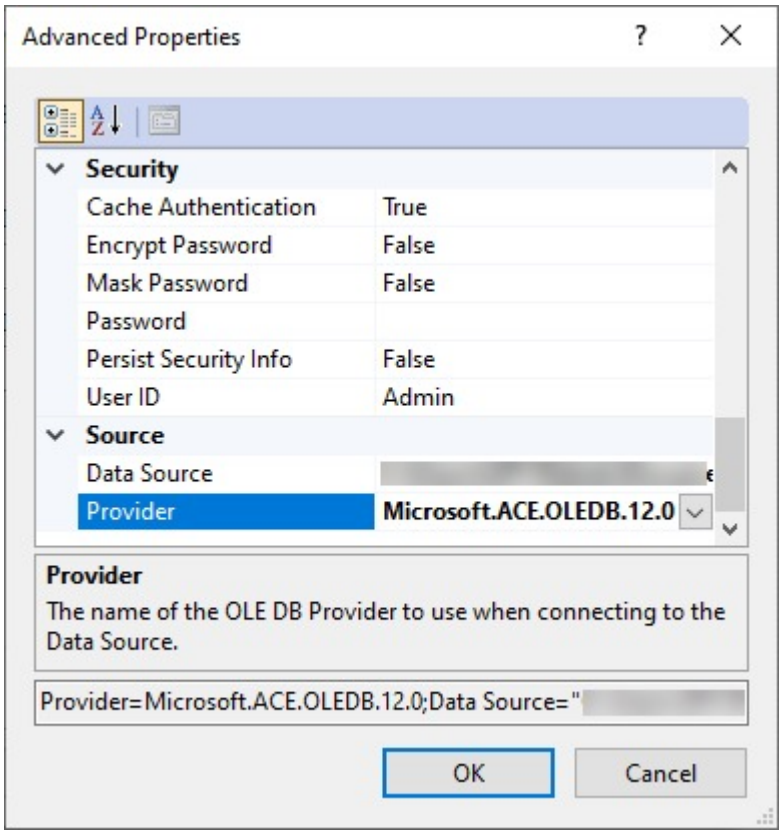
5. From the **Choose Data Source** dialog, select **Microsoft Access Database** File and click **Continue**.



6. In the **Add Connection** dialog, browse and select the **C1NWind.mdb** file, enter your username and password, and and click the **Advanced** button available towards the bottom-left.

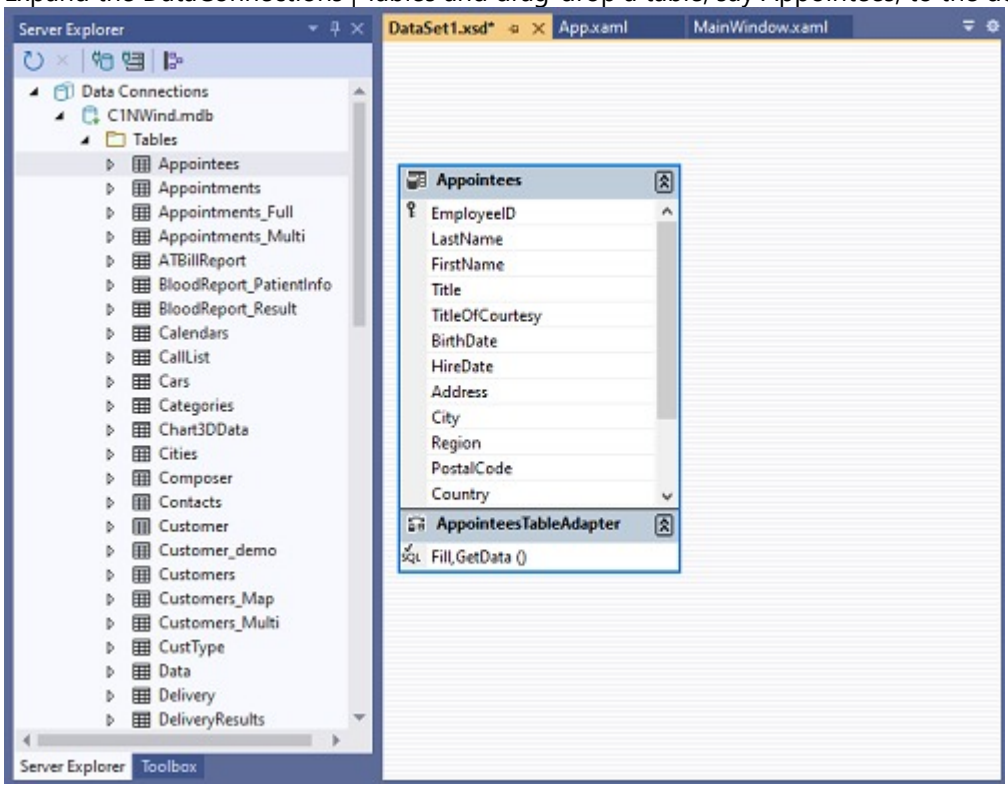


7. In the **Advanced Properties** dialog, navigate to the **Provider** property and change the Provider to **Microsoft.ACE.OLEDB.12.0** and click **OK**. Observe that a new connection will be added under **ServerExplorer | DataConnections**.



Note: In case you have a 32-bit system with .NET 4.5.2 API, please select Microsoft.Jet.OLEDB.4.0 as a provider in this step.

8. In the **Solution Explorer**, double-click the added DataSet.xsd file, say DataSet1, to open the designer.
9. Expand the DataConnections | Tables and drag-drop a table, say Appointees, to the designer.



10. Switch to the code view and add the following using statements:

C#

```
using C1.C1Schedule;
using System.Windows.Controls;
using ProjectNAME.C1NWindDataSetTableAdapters;
```

Note that the using statement should contain the name of your project in order to work correctly. It will be used to set up the table adapter for your data set. The Imports statement should be used for Visual Basic projects.

11. Add the following code to use the data from the dataset to fill the scheduler.

C#

```
public partial class MainWindow : Window
{
    private AppointmentsTableAdapter _appointmentsTableAdapter;
    private AppointeesTableAdapter _appointeesTableAdapter;
    private C1NwindDataSet _dataSet;

    public C1NwindDataSet DataSet => _dataSet;

    public MainWindow()
    {
        InitializeComponent();

        _dataSet = new C1NwindDataSet();
        _appointeesTableAdapter = new AppointeesTableAdapter();
        _appointmentsTableAdapter = new AppointmentsTableAdapter();
        _appointmentsTableAdapter.Fill(_dataSet.Appointments);
        _appointeesTableAdapter.Fill(_dataSet.Appointees);
    }
}
```

Back to Top

Bind Scheduler to the Data Source

Map the database to the **Appointment Storage** by adding the following XAML code between the `<c1:C1Scheduler></c1:C1Scheduler>` tag or adding the following C# code in the MainWindow() constructor:

XAML

```
<c1:NestedPropertySetter PropertyName="DataStorage.AppointmentStorage.DataMember"
Value="Appointments"/>
<c1:NestedPropertySetter PropertyName="DataStorage.AppointmentStorage.DataSource" Value="
{Binding DataSet, RelativeSource={RelativeSource AncestorType=local:DataSourceBinding,
Mode=FindAncestor}}"/>
<c1:NestedPropertySetter
PropertyName="DataStorage.AppointmentStorage.Mappings.Body.MappingName" Value="Body"/>
<c1:NestedPropertySetter
PropertyName="DataStorage.AppointmentStorage.Mappings.End.MappingName" Value="End"/>
<c1:NestedPropertySetter
PropertyName="DataStorage.AppointmentStorage.Mappings.Location.MappingName"
Value="Location"/>
<c1:NestedPropertySetter
PropertyName="DataStorage.AppointmentStorage.Mappings.Start.MappingName" Value="Start"/>
<c1:NestedPropertySetter
```

```
PropertyName="DataStorage.AppointmentStorage.Mappings.Subject.MappingName"
Value="Subject"/>
<c1:NestedPropertySetter
PropertyName="DataStorage.AppointmentStorage.Mappings.AppointmentProperties.MappingName"
Value="Properties"/>
<c1:NestedPropertySetter
PropertyName="DataStorage.AppointmentStorage.Mappings.IdMapping.MappingName" Value="Id"/>
<c1:NestedPropertySetter
PropertyName="DataStorage.AppointmentStorage.Mappings.IndexMapping.MappingName"
Value="N/A"/>
```

C#

```
// set mappings and DataSource for the ContactStorage
ContactStorage cstorage = scheduler.DataStorage.ContactStorage;
cstorage.Mappings.IndexMapping.MappingName = "EmployeeID";
cstorage.Mappings.TextMapping.MappingName = "FirstName";
cstorage.DataMember = "Appointees";
cstorage.DataSource = _dataSet.Appointees;

// set mappings and DataSource for the AppointmentStorage
AppointmentStorage storage = scheduler.DataStorage.AppointmentStorage;
storage.Mappings.AppointmentProperties.MappingName = "Properties";
storage.Mappings.Body.MappingName = "Body";
storage.Mappings.End.MappingName = "End";
storage.Mappings.IdMapping.MappingName = "Id";
storage.Mappings.Location.MappingName = "Location";
storage.Mappings.Start.MappingName = "Start";
storage.Mappings.Subject.MappingName = "Subject";
storage.DataMember = "Appointments";
storage.DataSource = _dataSet.Appointments;

DataContext = this;
```

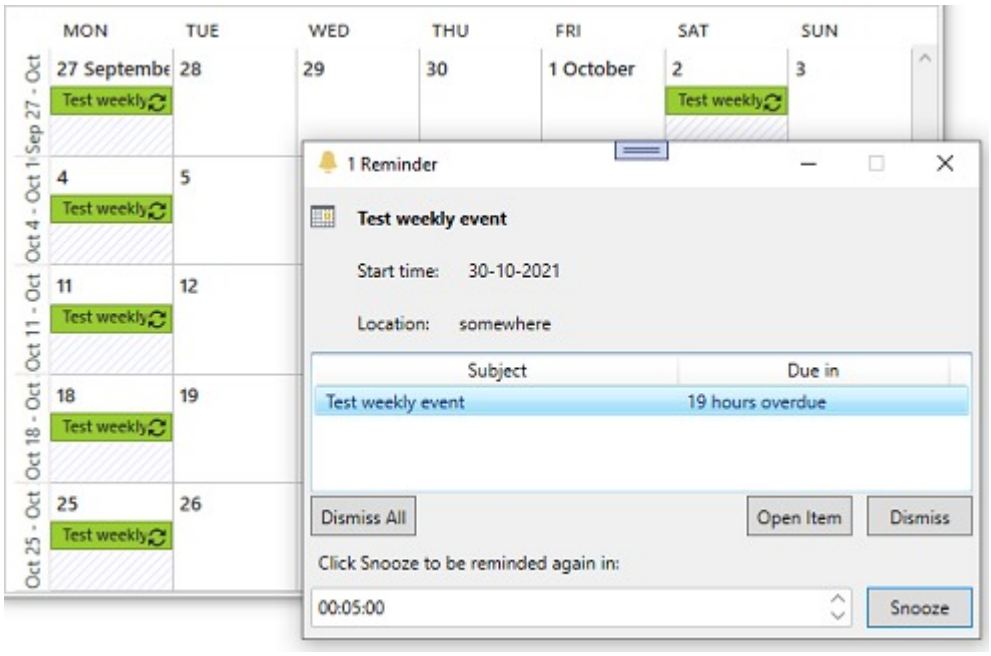
Back to Top

アプリケーションの実行

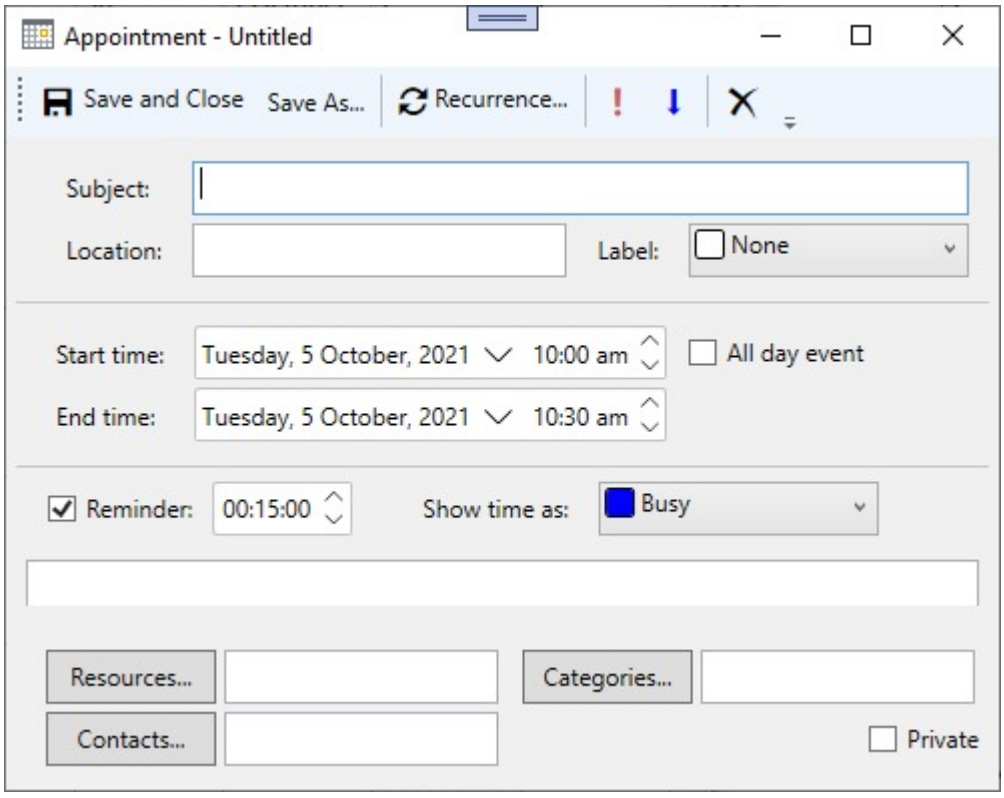
これで、スケジュールアプリケーションを作成し、スケジュールをデータソースに連結できたので、最後にアプリケーションを実行してみます。

スケジュールアプリケーションを実行して、Scheduler for WPF の実行時の動作を確認するには

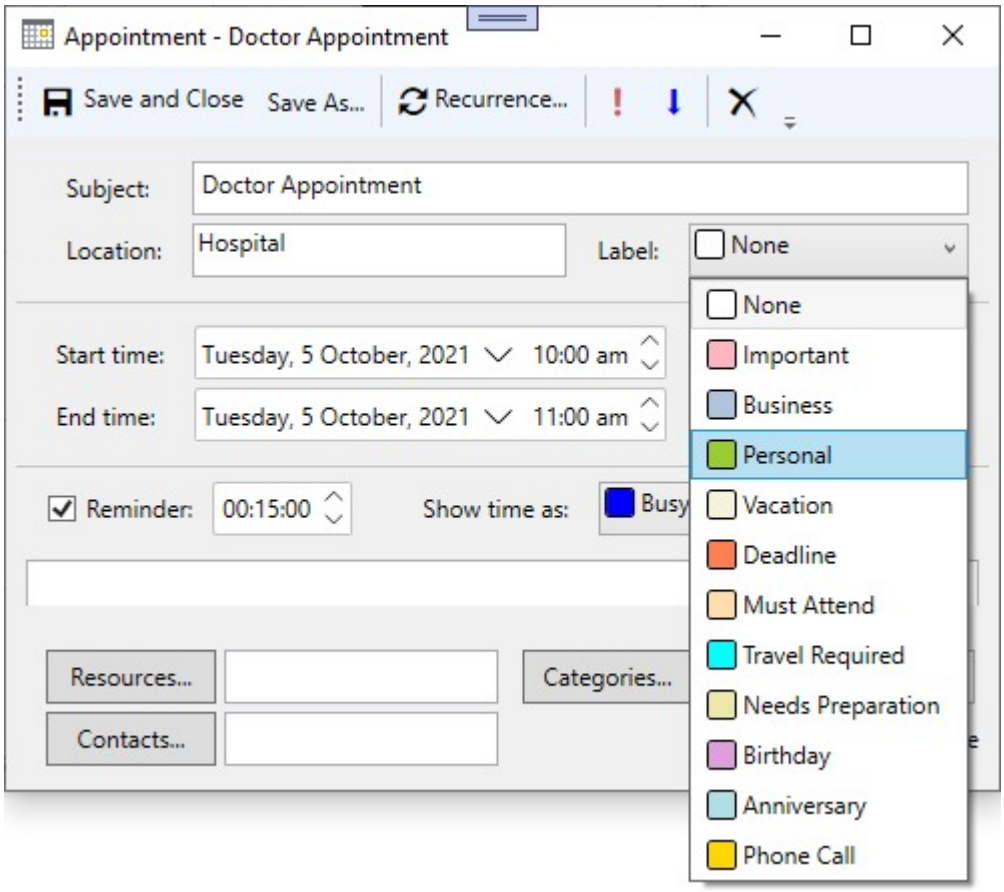
1. **[F5]** キーを押すか、［**デバッグ**］メニューから［**デバッグの開始**］を選択します。スケジュールと［**アラーム**］ダイアログボックスが表示されます。［**すべてのアラームを消す**］をクリックします。



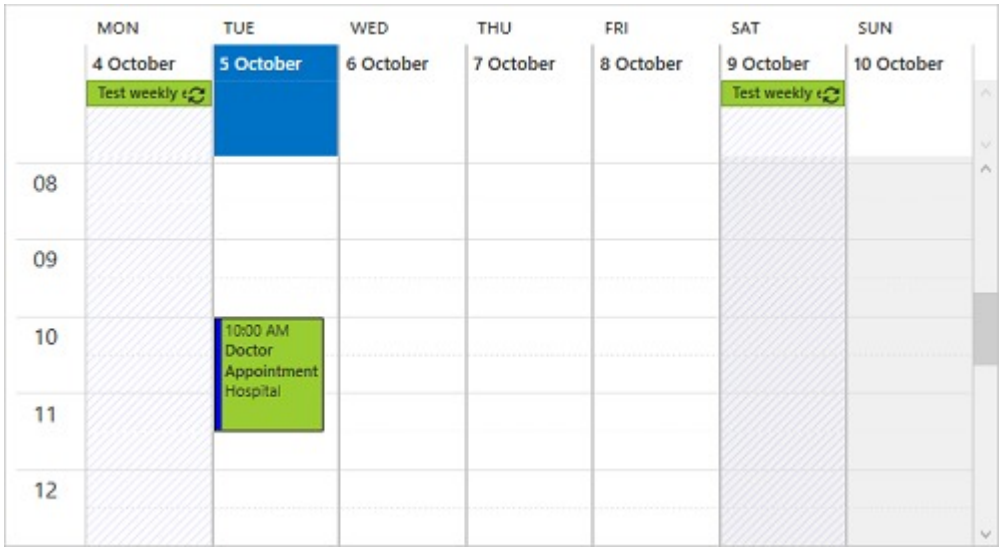
2. 目的の日時の下にある予定の時刻をダブルクリックして予定を設定します。この例では、10月5日火曜日の午前10時に予定を作成します。［**予定**］ダイアログボックスが開きます。



3. [**件名**] に「医者との予定」、 [**場所**］に「病院」、 [**終了時刻**］に「11:00AM」と入力します。
4. [**ラベル**］の隣にあるドロップダウン矢印をクリックして、［個人用］を選択します。



5. [**保存して閉じる**] ボタンをクリックします。スケジュールに新しい**予定**が**表示**されます。



Back to Top

データ連結

以下のトピックでは、Scheduler コントロールをデータソースに連結し、PropertyBridge クラスを使って非依存プロパティを連結する手順を説明します。

データソースへの C1Scheduler の連結

To bind the Scheduler control to the C1NWind.mdb database, we must configure the data source. Then we can add the dataset as a resource in the project and map to the C1Scheduler Data Storage. To understand these steps in details, see Configure the Data Source and Bind Scheduler to the Data Source sections in the Quick Start topic.

Back to Top

PropertyBridge クラスを使用したデータ連結

PropertyBridge クラスは、**System.Object** 型の **PropertyBridge.Source** と **PropertyBridge.Target** という 2 つの依存プロパティを公開し、2 つの値が等しくなるように保ちます。つまり、1 つのプロパティの値が変更されると、他方のプロパティの値も同じ値に設定されます。この単純な仕組みにより、DependencyProperty であるプロパティのみを使用するように設計された WPF メカニズムで、DependencyProperty でないプロパティを使用できるようになります。次の例では、**PropertyBridge** クラスを使用する方法を示します。

- **２つの非依存プロパティを連結する**
  ソースとして 1 つの非依存プロパティを持つ TwoWay 連結を **PropertyBridge.Source** プロパティに割り当て、ソースとして別の非依存プロパティを持つ TwoWay 連結を **PropertyBridge.Target** プロパティに割り当てます。これによって非依存プロパティが連結プロパティとして動作します。これは、**INotifyPropertyChanged** インターフェイスをサポートする非依存プロパティを公開しているクラスでのみ動作します。C1ScheduleStorage オブジェクトモデルの大部分のクラスがこれに当てはまります。
- **トリガ内から非依存プロパティ値を設定する**
  ソースとして 1 つの非依存プロパティを持つ **TwoWay** または **OneWayToSource** 連結を **PropertyBridge.Source** プロパティに割り当て、Trigger の Setter を使って値を PropertyBridge の **PropertyBridge.Target** プロパティに割り当てます。この値は、Source に連結される非依存プロパティに割り当てられます。
- **多対多連結**
  MultiBinding 連結を **PropertyBridge.Source** と **PropertyBridge.Target** に割り当てて、多対多連結を実現します。
- **ネストされたプロパティに値を割り当てる**
  ネストしているプロパティを参照する Path で、ターゲットを **TwoWay** または **OneWayToSource** に設定します。次に、**PropertyBridge.Source** を割り当てます（直接、または Setter 内から）。ネストしているプロパティに、この値が割り当てられます。
- **直接アクセスできないオブジェクトのプロパティを割り当てる**
  ネストしているプロパティへの値の割り当てと同様に、Target への連結で **RelativeSource** を使用して、**TemplatedParent** やビジュアルツリーの一部の親要素などの XAML で直接参照できない要素にプロパティの値を割り当てます。

PropertyBridge クラスは **FrameworkElement** から派生されます。また、これが正しく動作するには、やり取りする必要がある他の要素と共にビジュアルツリー内に配置される必要があります。**FrameworkElement** からの派生は意図的です。PropertyBridge をビジュアルツリーに含めることができ、これにより、連結が正しいコンテキストでプロパティに確立されます。たとえば、理論的には **ResourceDictionary** に PropertyBridge を置く選択肢も

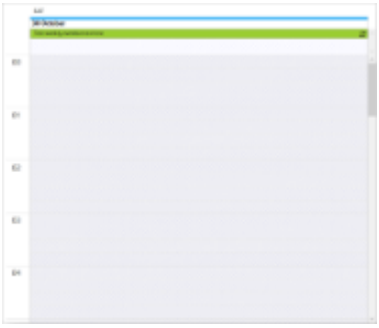
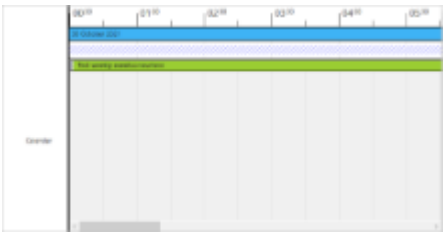
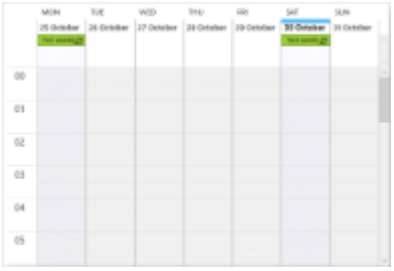
ありますが、この場合は連結が動作しません。

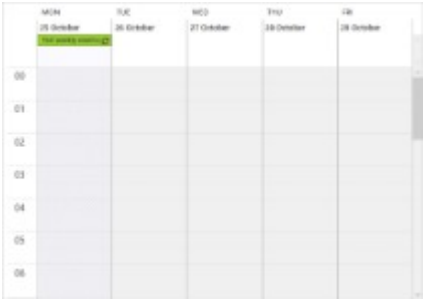
PropertyBridge.Visibility プロパティは、デフォルトで **Collapsed** に設定されています。したがって、このオブジェクトは画面に表示されず、レイアウトの測定および配置処理の対象になりません。つまり、このプロパティは、配置先のビジュアルツリーのビジュアル表現を変更しません。

Back to Top

データビュー

Scheduler provides the following predefined data views that determine the style used in the control:

View	Description
Day 	Displays a detailed view showing appointments for a particular day.
TimeLine 	Displays appointments in a horizontal time line.
Month 	Displays appointments for one or more months.
Week 	Displays appointments for specified work days.
WorkWeek	Displays appointments for any given weekly period. The default is

View	Description
	Monday through Friday.

The default data view for the Scheduler control is Month view. However, you can change the default data view by setting the ViewType property. This property determines which style should be used in the control using the **ViewType** enumeration.

XAML

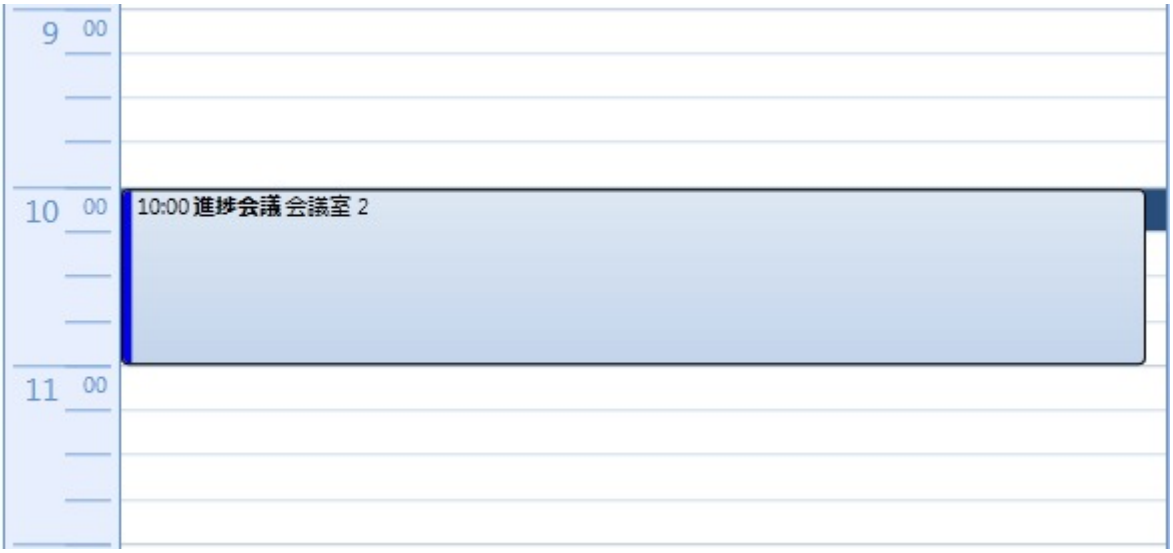
```
<c1:C1Scheduler x:Name="scheduler" ViewType="Day"></c1:C1Scheduler>
```

C#

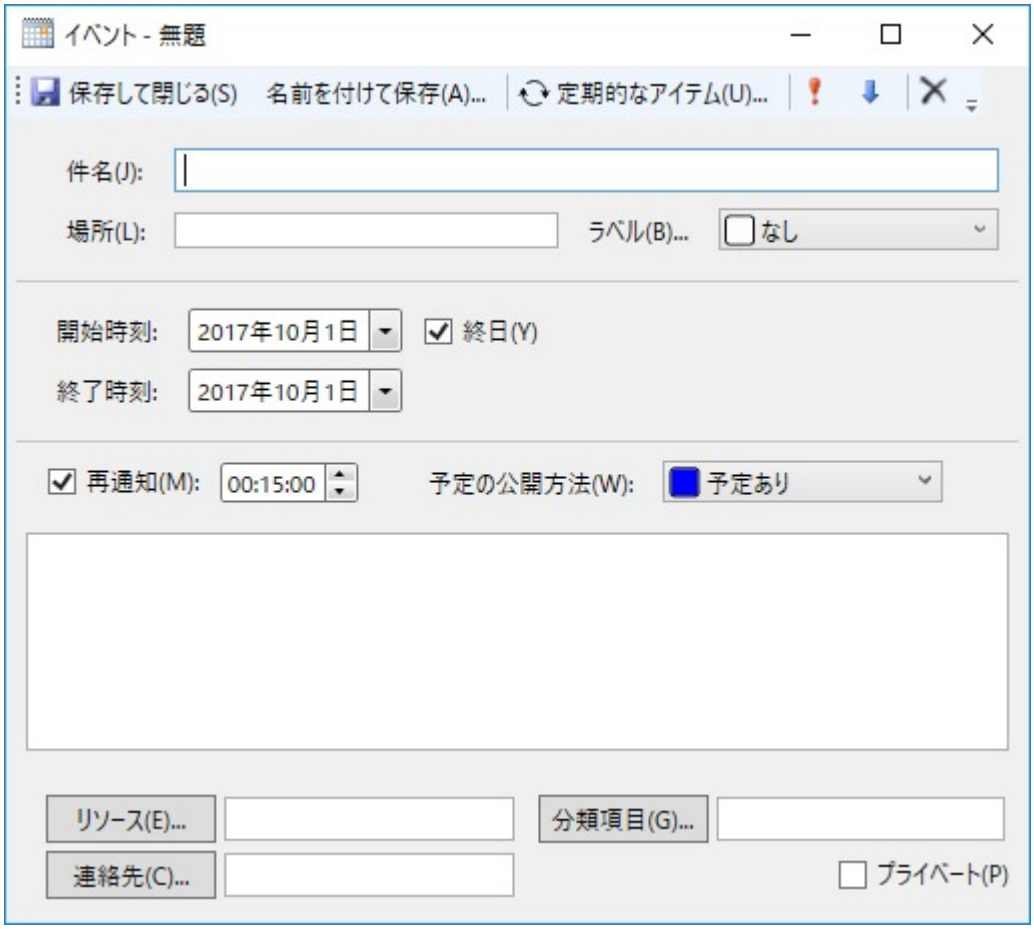
```
scheduler.ViewType = ViewType.Day;
```

予定

予定は、一定期間内に実行されるイベントに関する詳細な情報と、その期間を表します。予定は、30 分といった時間から複数の日に渡るまで、指定した長さのイベントです。



新しい予定を追加するには、または既存の予定を編集するには、その予定の時刻をダブルクリックします。The following image shows an appointment dialog with an appointment set to visit a doctor.



[予定] ダイアログボックスを使用して、新しい予定のスケジュールを設定します。このとき、タイトル、場所、ラベル、開始時刻と終了時刻、アラーム、公開方法、その予定が全日のイベントであるかどうか、指定された期間内に繰り返し実行されるものであるかどうかを指定できます。また、任意のリソース、分類、および連絡

先も指定できます。

ラベル

ラベルは、予定に追加できる色分けされたマーカーです。ラベルを設定したユーザーおよび他のユーザーは、予定の詳細を表示することなく、その予定の種類を確認できます。

Scheduler for WPF には、12 個の定義済みラベルが用意されています。ラベルの色は、**C1Scheduler** コントロールのすべてのデータビューに表示されます。

定義済みのラベル

定義済みのラベルは次のとおりです。

ラベル	色	インデックス
なし		0
重要		1
ビジネス		2
個人用		3
休暇		4
締め切り		5
要出席		6
出張		7
要準備		8
誕生日		9
記念日		10
電話連絡		11

To add a label associated to an appointment programmatically, you can use Label property of the Appointment class and set its index (from the above table) based on your requirements. For instance, the following code demonstrates how you can add "Important" label to an appointment:

C#

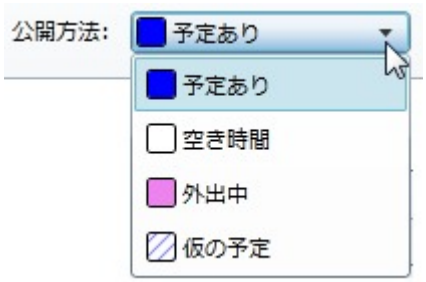
```
//Showing Labels
// adding 'Important' label.
appointment.Label = scheduler.DataStorage.LabelStorage.Labels[1];
```

公開方法

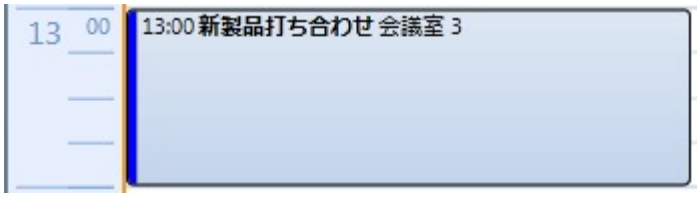
公開方法を使用すると、予定がスケジュールされる際に自分が参加できるかどうかを全員に知らせることができます。**Scheduler for WPF** では、予定あり、空き時間、外出中、仮の予定およびWorking Elsewhere の5つの定義済み公開方法を使用できます。公開方法は、次の色で指定されます。

状況	色	インデックス
予定あり		0
空き時間		1
外出中		2
仮の予定		3
Working Elsewhere		4

公開方法は、［**公開方法**］ドロップダウンリストで指定します。



公開方法の色は、スケジュールの **OneDayStyle** ビューおよび **WorkingWeekStyle** ビューでのみ表示されます。予定の色は、予定の左側にある領域に表示されます。予定を選択すると、その予定は公開方法を表す色で強調表示されます。



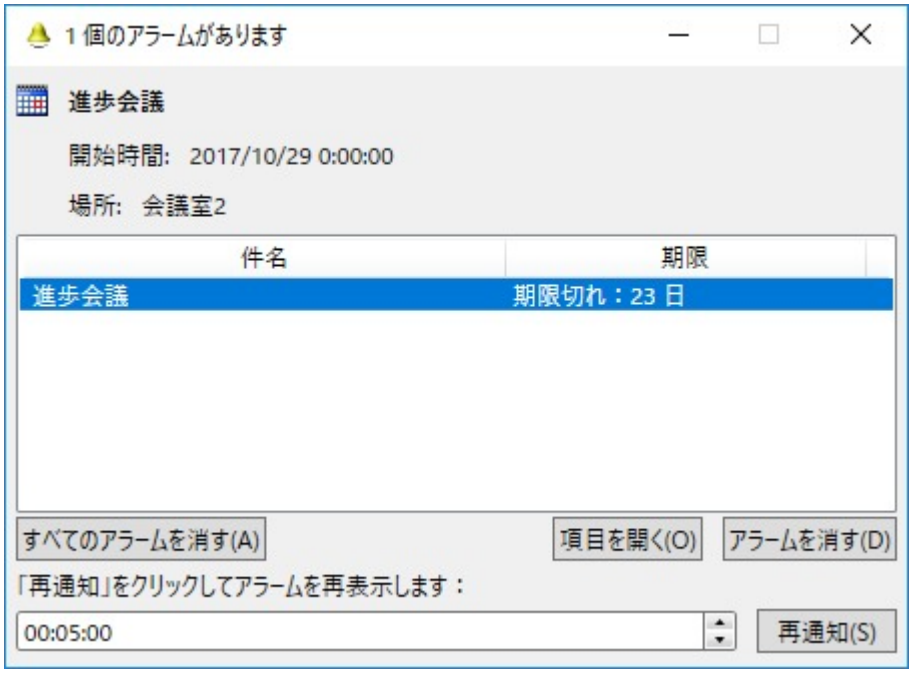
To set the availability status during an appointment, you can use BusyStatus property of the Appointment class and set its index (from the table given at towards the top of the topic) based on your requirements. For instance, the following code demonstrates how you can set "Busy" status for an appointment:

C#

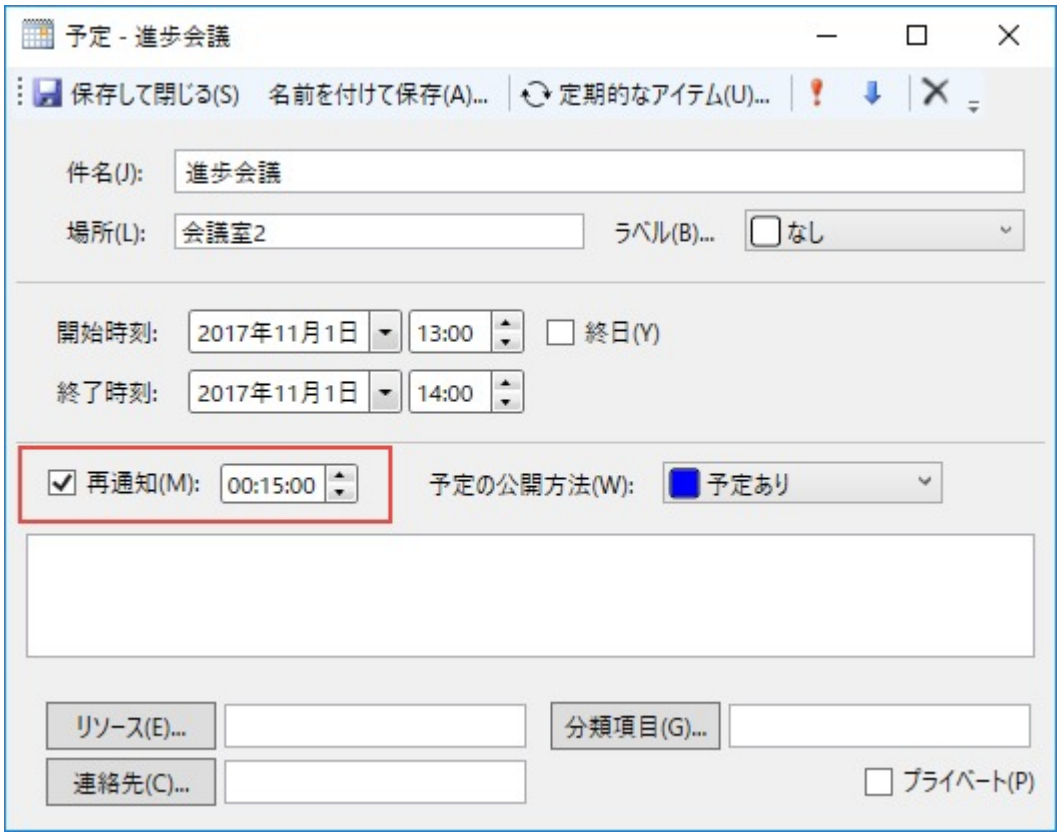
```
//Showing Availability status
appointment.BusyStatus = scheduler.DataStorage.StatusStorage.Statuses[0]; // adding 'Busy'
status.
```

アラーム

予定のアラームを使用すると、予定が発生する時刻よりも指定した時間だけ前に［**アラーム**］ダイアログボックスが表示されます。［**アラーム**］ダイアログボックスでは、１つまたは複数の予定を取り消すオプション（複数の予定が設定されている場合）、予定を開くオプション、またはアラームをリセットして設定した時間内に再表示するオプションを指定できます。



アラームは、予定を作成するときに設定できます。それには、［**再通知**］チェックボックスをオンにし、予定のどれだけ前にアラームを表示するかを指定する時間を［**アラーム**］ドロップダウンリストで指定します。



To set a reminder for an appointment, you can set ReminderSet property of the Appointment class to **true** and then based on your requirements, you can set the time interval for the reminder to start prior to the appointment as demonstrated in the following code:

C#

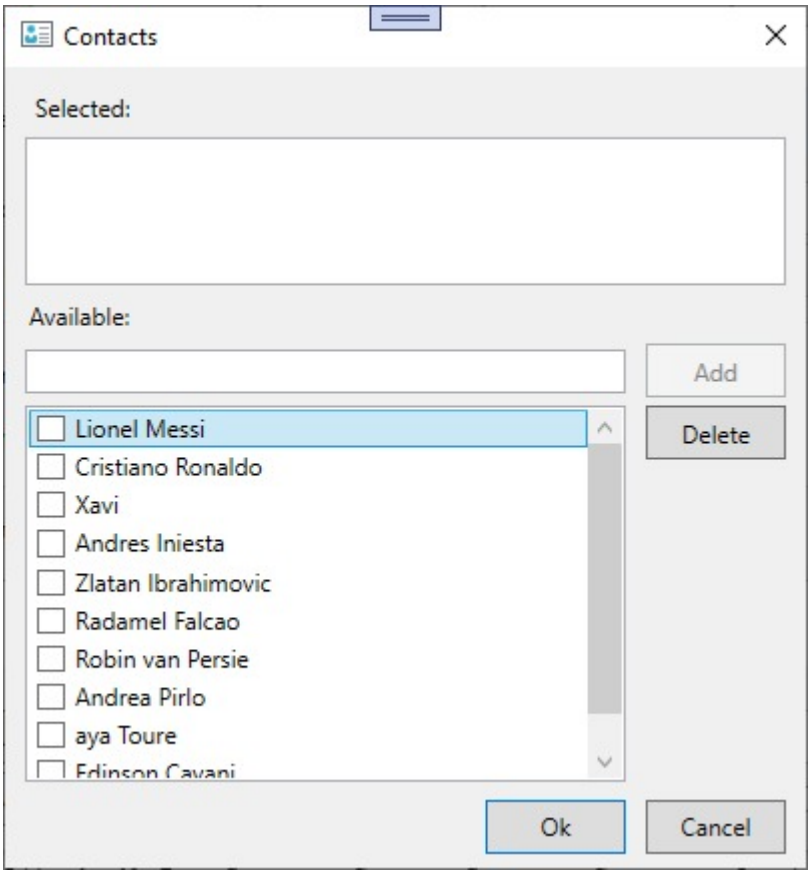
```
// Showing Reminder
appointment.ReminderSet = true;
//remind before one minute
appointment.ReminderTimeBeforeStart = TimeSpan.FromMinutes(1);
```

連絡先

通常、連絡先のリストには、頻繁に通信するすべてのユーザーが含まれています。予定には、その予定について知らせる必要がある相手の連絡先を割り当てることができます。連絡先情報は **ContactCollection** クラスに格納されます。既存の連絡先をデータソースから取得できます。また、実行時に**連絡先**を追加することもできます。連絡先の指定はオプションです。また、予定には、１人または複数の連絡先を割り当てることができます。

連絡先リストの追加

Scheduler for WPF では、実行時に［**連絡先**］ダイアログボックスを使って作成された連絡先がサポートされます。リストに追加した連絡先を予定に割り当てることができます。



実行時に連絡先を追加するには

1. 新しい予定を追加するか、既存の予定を開きます。
2. [**予定**] ダイアログボックスで、［**連絡先**］ボタンをクリックします。［**連絡先**］ダイアログボックスが表示されます。
3. [**利用可能な項目**] テキストボックスに名前を入力して［**追加**］をクリックします。
4. [**OK**] をクリックして、［**連絡先**］ダイアログボックスを閉じます。

To add contacts programmatically in the **Appointment** dialog, use the following code:

C#

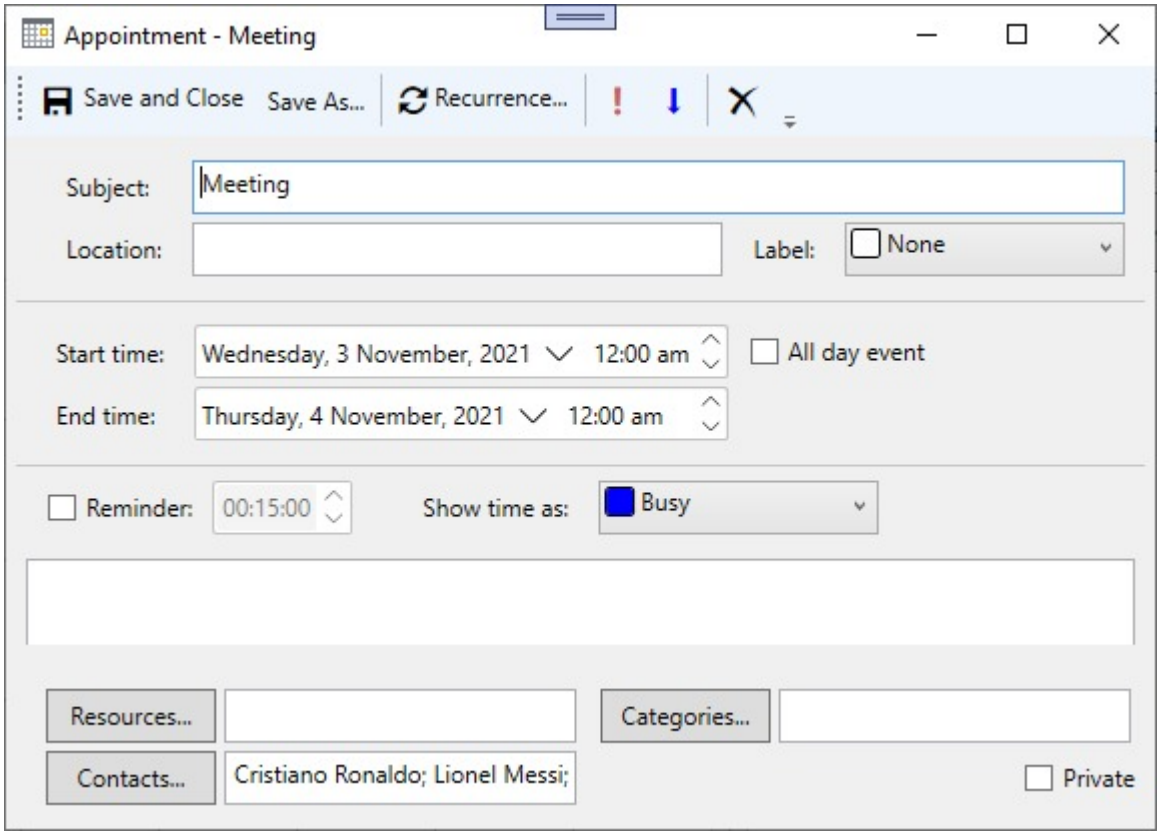
```
ObservableCollection<string> contacts = new ObservableCollection<string>();
public AddContacts()
{
    InitializeComponent();
    contacts.Add("Lionel Messi");
    contacts.Add("Cristiano Ronaldo");
    contacts.Add("Xavi");
    contacts.Add("Andres Iniesta");
    contacts.Add("Zlatan Ibrahimovic");
    contacts.Add("Radamel Falcao");
    contacts.Add("Robin van Persie");
    contacts.Add("Andrea Pirlo");
    contacts.Add("aya Toure");
    contacts.Add("Edinson Cavani");

    foreach (string contact in contacts)
```

```
    {
        Contact cnt = new Contact();
        cnt.MenuCaption = contact;
        if (!sched1.DataStorage.ContactStorage.Contacts.Any(x => x.MenuCaption == contact))
            sched1.DataStorage.ContactStorage.Contacts.Add(cnt);
    }
}
```

予定への連絡先の割り当て

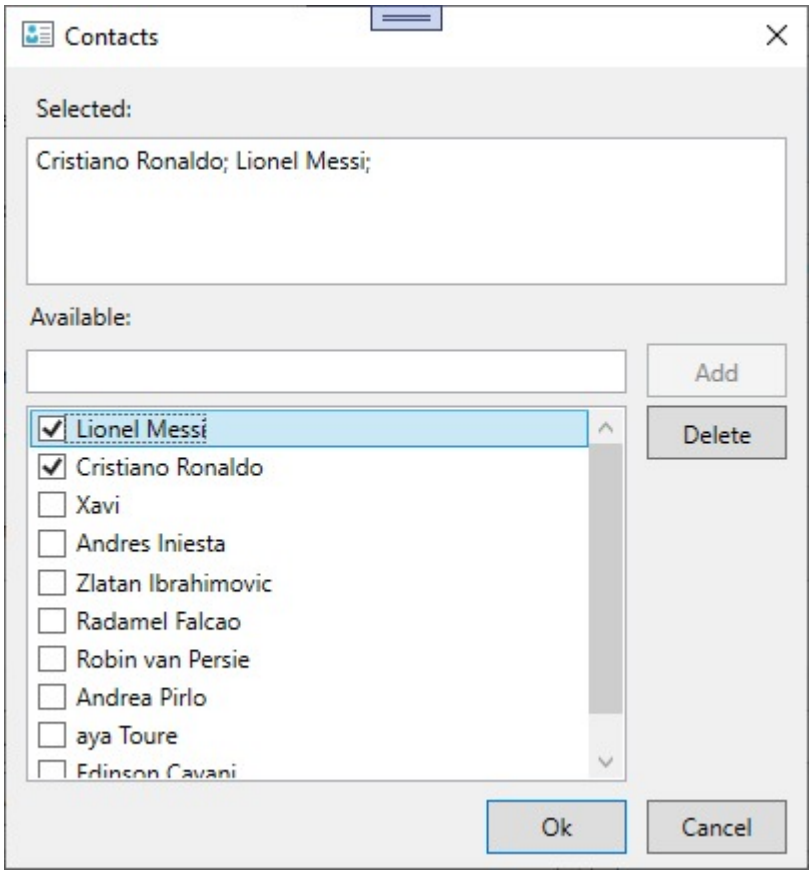
実行時に連絡先（複数可）を予定に割り当てるには、［**連絡先**］ダイアログボックスを使用します。項目の追加方法については、「連絡先リストの追加」を参照してください。

The contact appears next in the **Contacts** text box as shown in the following image.



実行時に連絡先を予定に割り当てるには

1. ［**予定**］ダイアログボックスで、［**連絡先**］ボタンをクリックします。
2. 対象の連絡先の横にあるチェックボックスをオンにし、［**OK**］をクリックします。



To assign contacts to an appointment programmatically, use the following code:

C#

```
//Create Appointments
Appointment app = new Appointment(DateTime.Now.Date,DateTime.Now.Date.AddDays(1));
app.Subject = "Meeting";
sched1.DataStorage.AppointmentStorage.Appointments.Add(app);

//Add contacts to the ContactList
sched1.DataStorage.ContactStorage.Contacts.Add(new Contact() { MenuCaption =  "Lionel
Messi" });
sched1.DataStorage.ContactStorage.Contacts.Add(new Contact() { MenuCaption = "Cristiano
Ronaldo" });
sched1.DataStorage.ContactStorage.Contacts.Add(new Contact() { MenuCaption = "Xavi" });
sched1.DataStorage.ContactStorage.Contacts.Add(new Contact() { MenuCaption = "Andres
Iniesta" });
sched1.DataStorage.ContactStorage.Contacts.Add(new Contact() { MenuCaption = "Zlatan
Ibrahimovic" });
sched1.DataStorage.ContactStorage.Contacts.Add(new Contact() { MenuCaption = "Radamel
Falcao" });
sched1.DataStorage.ContactStorage.Contacts.Add(new Contact() { MenuCaption = "Robin van
Persie" });
sched1.DataStorage.ContactStorage.Contacts.Add(new Contact() { MenuCaption = "Andrea Pirlo"
});
sched1.DataStorage.ContactStorage.Contacts.Add(new Contact() { MenuCaption = "aya Toure"
});
sched1.DataStorage.ContactStorage.Contacts.Add(new Contact() { MenuCaption = "Edinson
```

```
Cavani" });

var appointment = sched1.DataStorage.AppointmentStorage.Appointments.First(x => x.Start ==
DateTime.Now.Date);
var contactList = sched1.DataStorage.ContactStorage.Contacts;
appointment.Links.Add(contactList.First(x => x.MenuCaption == "Cristiano Ronaldo"));
appointment.Links.Add(contactList.First(x => x.MenuCaption == "Lionel Messi"));
```

分類

分類とは、予定を整理する際に使用するキーワードまたはフレーズです。**Scheduler for WPF**では、20 個の定義済み分類が用意されており、これらを予定に割り当てることができます。データベースの分類を使用することもできます。また、ユーザーは、実行時に独自のカスタム分類を作成することができます。**CategoryCollection** クラスに格納される分類はオプションです。 1 つの予定に複数の分類を割り当てることもできます。

定義済みの分類

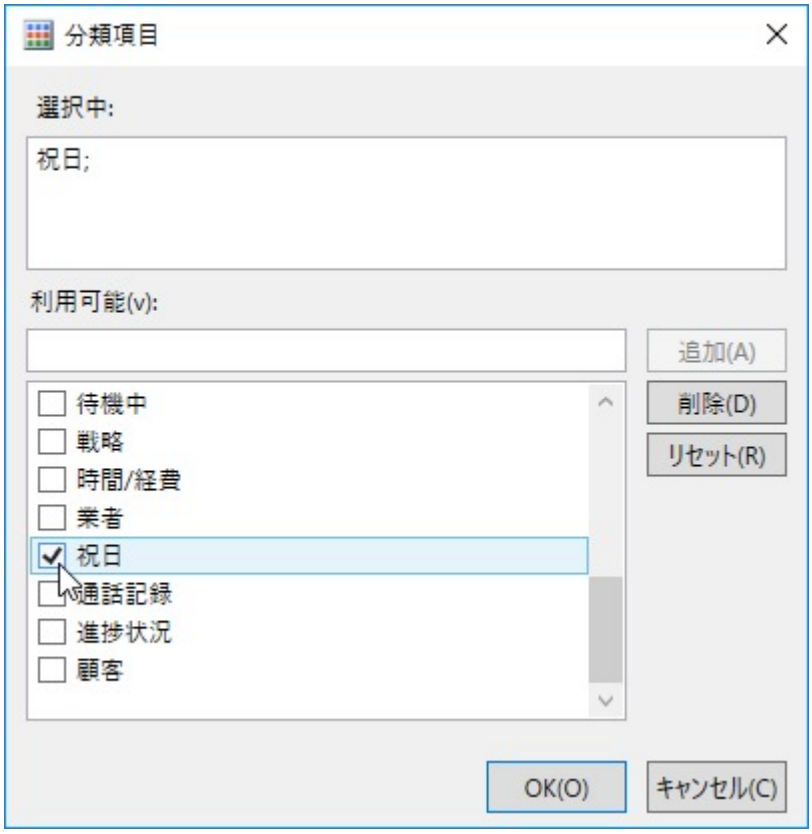
定義済みの分類は次のとおりです。

分類	インデックス
VIP	0
アイディア	1
お気に入り	2
グリーティングカード	3
その他	4
個人用	5
友人	6
友人	7
取引先	8
国際的	9
待機中	10
戦略	11
時間と経費	12
業者	13
目標	14
祝日	15
競合相手	16
贈り物	17
進捗状況	18
重要顧客	19
電話連絡	20

分類リストの追加

Scheduler for WPF では、実行時に［**分類**］ダイアログボックスを使って作成された分類がサポートされます。リストに追加した分類を予定に割り当てることができます。

［**予定**］ダイアログボックスで［**分類**］ボタンをクリックすると、新しい分類が［**分類**］ダイアログボックスに表示されます。



実行時に分類を追加するには

1. 新しい予定を追加するか、既存の予定を開きます。
2. ［**予定**］ダイアログボックスで、［**分類**］ボタンをクリックします。［**分類**］ダイアログボックスが表示されます。
3. ［**利用可能な項目**］テキストボックスに名前を入力して［**追加**］をクリックします。新しい分類がリストの最後に追加されます。
4. ［**OK**］をクリックして、［**分類**］ダイアログボックスを閉じます。

To add a categories programmatically, use the following code:

C#

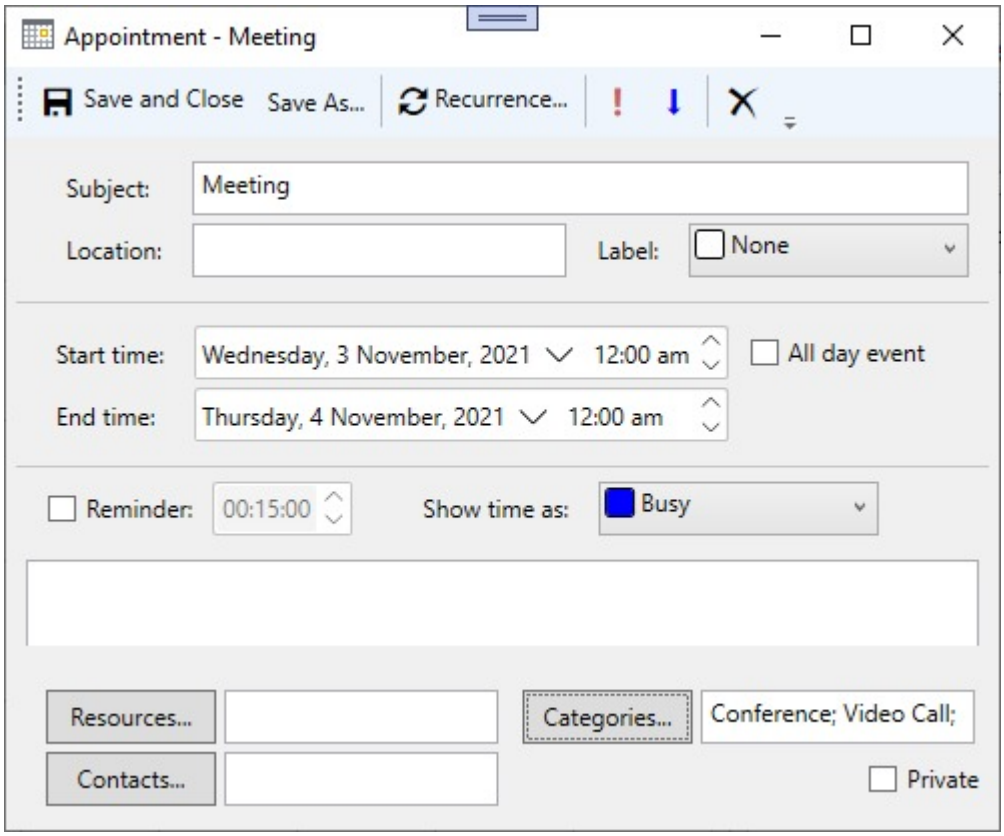
```
ObservableCollection<string> categories = new ObservableCollection<string>();
public AddCategories()
{
    InitializeComponent();
    // add new categories to the existing list
    categories.Add("Conference");
```

```
    categories.Add("Video Call");
    categories.Add("Interview");
    categories.Add("Games");

    foreach (string category in categories)
    {
        Category ct = new Category();
        ct.MenuCaption = category;
        if (!sched1.DataStorage.CategoryStorage.Categories.Any(x => x.MenuCaption ==
category))
            sched1.DataStorage.CategoryStorage.Categories.Add(ct);
    }
}
```

予定への分類の割り当て

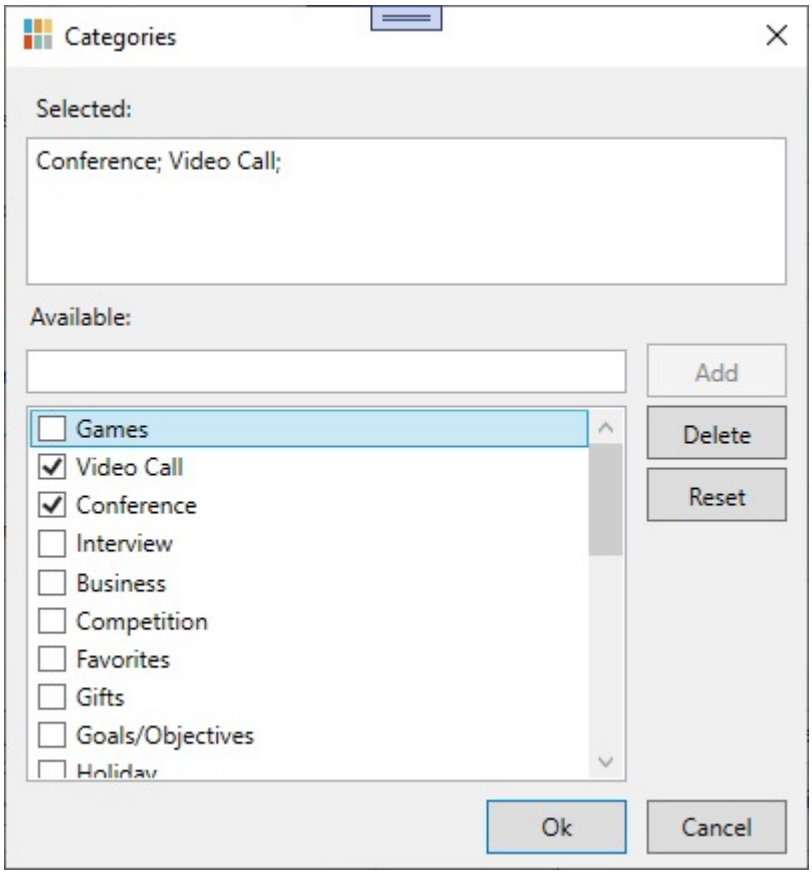
実行時に分類（複数可）を予定に割り当てるには、［**分類**］ダイアログボックスを使用します。デフォルトでは、20 個の定義済み分類から成るリストが用意されています。分類を［**分類**］ダイアログボックスに追加する方法については、「分類リストの追加」を参照してください。

The categories appears next in the **Categories** text box as shown in the following image.



実行時に分類を予定に割り当てるには

1. ［**予定**］ダイアログボックスで、［**分類**］ボタンをクリックします。
2. 対象の分類の横にあるチェックボックスをオンにし、［**OK**］をクリックします。［**分類**］テキストボックスの横に、分類が表示されます。予定には複数の分類を割り当てることができます。



To assign a category to an appointment programmatically, use the following code:

C#

```
//Create Appointments
Appointment app = new Appointment(DateTime.Now.Date, DateTime.Now.Date.AddDays(1));
app.Subject = "Meeting";
sched1.DataStorage.AppointmentStorage.Appointments.Add(app);

//Add new Categories to the CategoryList
sched1.DataStorage.CategoryStorage.Categories.Add(new Category() { MenuCaption =
"Conference" });
sched1.DataStorage.CategoryStorage.Categories.Add(new Category() { MenuCaption = "Video
Call" });
sched1.DataStorage.CategoryStorage.Categories.Add(new Category() { MenuCaption =
"Interview" });
sched1.DataStorage.CategoryStorage.Categories.Add(new Category() { MenuCaption = "Games"
});

var appointment = sched1.DataStorage.AppointmentStorage.Appointments.First(x => x.Start ==
DateTime.Now.Date);
var categoryList = sched1.DataStorage.CategoryStorage.Categories;
appointment.Categories.Add(categoryList.First(x => x.MenuCaption == "Conference"));
appointment.Categories.Add(categoryList.First(x => x.MenuCaption == "Video Call"));
```

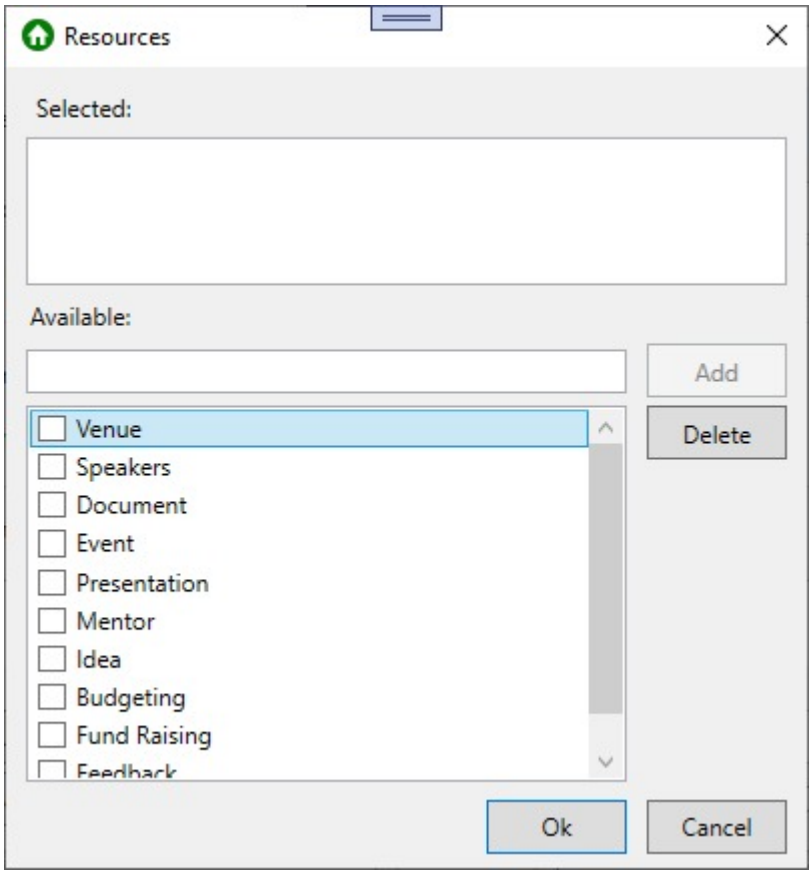
リソース

リソースとは、特定のタスクをサポートするソースを定義するキーワードまたはフレーズです。**ResourceCollection** クラスに格納されるリソースはオプションです。１つの予定に複数のリソースを割り当てることもできます。

リソースリストへのリソースの追加

Schedulerでは、実行時に［リソース］ダイアログボックスを使って作成されたリソースがサポートされます。リストに追加したリソースを予定に割り当てることができます。

［**予定**］ダイアログボックスで［**リソース**］ボタンをクリックすると、新しいリソースが［**リソース**］ダイアログボックスに表示されます。



実行時にリソースを追加するには

1. 新しい予定を追加するか、既存の予定を開きます。
2. ［**予定**］ダイアログボックスで、［リソース］ボタンをクリックします。［**リソース**］ダイアログボックスが表示されます。
3. ［**利用可能な項目**］テキストボックスにリソースを入力して［**追加**］をクリックします。新しいリソースがリストに追加されます。
4. ［**OK**］をクリックして、［**リソース**］ダイアログボックスを閉じます。

To add resources programmatically, use the following code:

C#

```
ObservableCollection<string> resources = new ObservableCollection<string>();
public AddResources()
{
```

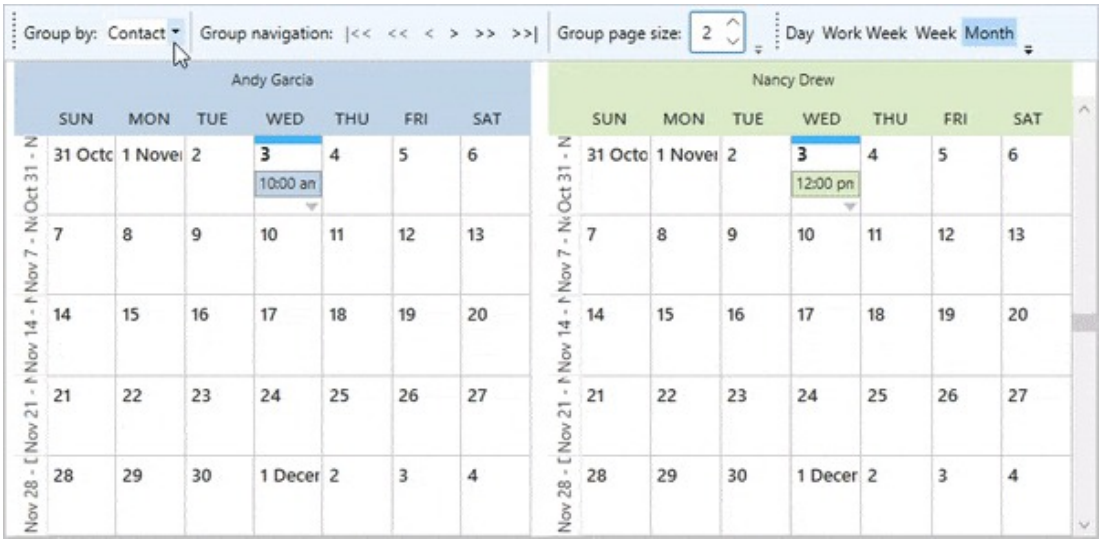
```
    InitializeComponent();
    resources.Add("Venue");
    resources.Add("Speakers");
    resources.Add("Document");
    resources.Add("Event");
    resources.Add("Presentation");
    resources.Add("Mentor");
    resources.Add("Idea");
    resources.Add("Budgeting");
    resources.Add("Fund Raising");
    resources.Add("Feedback");

    foreach (string resource in resources)
    {
        Resource res = new Resource();
        res.MenuCaption = resource;
        if (!sched1.DataStorage.ResourceStorage.Resources.Any(x => x.MenuCaption ==
resource))
            sched1.DataStorage.ResourceStorage.Resources.Add(res);
    }
}
```

グループ化

GroupBy プロパティを使用して、簡単にグループ化を行うことができます。このプロパティは、表示するグループ化の種類を決定します。C1Scheduler コントロールは、連絡先、カテゴリ、リソースや、Owner プロパティの値に基づくグループ化をサポートしています。

文字列	グループ化
空文字列	グループ化なし。
Category	グループ化は、Appointment.Categories プロパティ値によって決定されます。
Contact	グループ化は、Appointment.Links プロパティ値によって決定されます。
Owner	グループ化は、Appointment.Owner プロパティ値によって決定されます。
Resource	グループ化は、Appointment.Resources プロパティ値によって決定されます。

The following GIF showcases different types of grouping that can be done in Scheduler:



The following code snippets showcase how you can group appointments by the contacts, categories and resources. In this example, we have added a toolbar with a combo box to group appointments, navigation buttons to navigate between the groups, numeric box to allow you to display the desired number of group pages, and radio buttons to allow you to change the type of views according to your requirements.

XAML

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto" />
        <RowDefinition />
    </Grid.RowDefinitions>

    <ToolBarTray Grid.Row="0" Grid.ColumnSpan="2">
        <ToolBar Band="1" BandIndex="1">
            <TextBlock HorizontalAlignment="Left" VerticalAlignment="Center" TextWrapping="NoWrap"
Margin="4,2" Text="Group by:" />
            <ComboBox x:Name="cmbGroup" Margin="2" SelectionChanged="ComboBox_SelectionChanged" />
            <Separator />
            <TextBlock HorizontalAlignment="Left" VerticalAlignment="Center" TextWrapping="NoWrap"
Margin="4,2" Text="Group navigation:" />
            <Button Content="|&lt;&lt;" Margin="2"
Command="c1:C1Scheduler.NavigateToPreviousGroupCommand" CommandParameter="Home" CommandTarget="
{Binding ElementName=sched1}" />
```

```
            <Button Content="&lt;&lt;" Margin="2"
Command="c1:C1Scheduler.NavigateToPreviousGroupCommand" CommandParameter="Page" CommandTarget="
{Binding ElementName=sched1}" />
            <Button Content="&lt;" Margin="2"
Command="c1:C1Scheduler.NavigateToPreviousGroupCommand" CommandTarget="{Binding ElementName=sched1}"
/>
            <Button Content="&gt;" Margin="2" Command="c1:C1Scheduler.NavigateToNextGroupCommand"
CommandTarget="{Binding ElementName=sched1}" />
            <Button Content="&gt;&gt;" Margin="2"
Command="c1:C1Scheduler.NavigateToNextGroupCommand" CommandParameter="Page" CommandTarget="{Binding
ElementName=sched1}" />
            <Button Content="&gt;&gt;|" Margin="2"
Command="c1:C1Scheduler.NavigateToNextGroupCommand" CommandParameter="End" CommandTarget="{Binding
ElementName=sched1}" />
            <Separator />
            <TextBlock HorizontalAlignment="Left" VerticalAlignment="Center" TextWrapping="NoWrap"
Margin="4,2" Text="Group page size:" />
            <c1:C1NumericBox Margin="2" Value="{Binding GroupPageSize, ElementName=sched1,
Mode=TwoWay}" Minimum="1" Maximum="5" MinWidth="35" />
        </ToolBar>
        <ToolBar Band="1" BandIndex="2">
            <RadioButton x:Name="btnDay" Content="Day" CommandTarget="{Binding ElementName=sched1}"
Command="c1:C1Scheduler.ChangeStyleCommand" CommandParameter="{Binding Path=OneDayStyle,
ElementName=sched1}" />
            <RadioButton x:Name="btnWorkWeek" Content="Work Week" CommandTarget="{Binding
ElementName=sched1}" Command="c1:C1Scheduler.ChangeStyleCommand" CommandParameter="{Binding
Path=WorkingWeekStyle, ElementName=sched1}" />
            <RadioButton x:Name="btnWeek" Content="Week" CommandTarget="{Binding
ElementName=sched1}" Command="c1:C1Scheduler.ChangeStyleCommand" CommandParameter="{Binding
Path=WeekStyle, ElementName=sched1}" />
            <RadioButton x:Name="btnMonth" Content="Month" CommandTarget="{Binding
ElementName=sched1}" Command="c1:C1Scheduler.ChangeStyleCommand" CommandParameter="{Binding
Path=MonthStyle, ElementName=sched1}" />
            <RadioButton x:Name="btnTimeLine" Content="Time Line" CommandTarget="{Binding
ElementName=sched1}" Command="c1:C1Scheduler.ChangeStyleCommand" CommandParameter="{Binding
Path=TimeLineStyle, ElementName=sched1}" />
        </ToolBar>
    </ToolBarTray>

    <c1:C1Scheduler Grid.Row="1" x:Name="sched1">
        <c1:C1Scheduler.Settings>
            <c1:C1SchedulerSettings AllowCategoriesEditing="False"
AllowCategoriesMultiSelection="False" AllowResourcesEditing="False"
AllowResourcesMultiSelection="False" AllowContactsMultiSelection="True" AllowContactsEditing="True"
FirstVisibleTime="08:00:00" />
        </c1:C1Scheduler.Settings>
    </c1:C1Scheduler>
</Grid>
```

C#

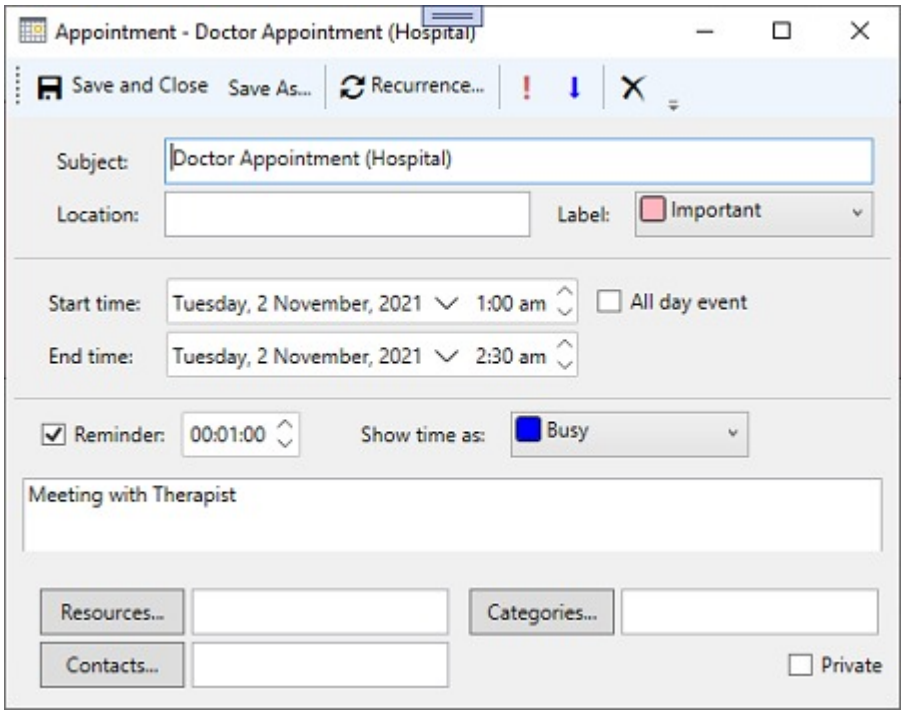
```
public Grouping()
{
    InitializeComponent();
    Language =
System.Windows.Markup.XmlLanguage.GetLanguage(System.Globalization.CultureInfo.CurrentCulture.Name);
    InitializeComponent();
    cmbGroup.Items.Add("None");
    cmbGroup.Items.Add("Category");
```

```
    cmbGroup.Items.Add("Resource");
    cmbGroup.Items.Add("Contact");
    cmbGroup.SelectedIndex = 3;

    // add some resources
    Resource res = new Resource();
    res.Text = "Meeting room";
    res.Color = Color.FromArgb(255, 218, 186, 198);
    sched1.DataStorage.ResourceStorage.Resources.Add(res);
    Resource res1 = new Resource();
    res1.Text = "Conference hall";
    res1.Color = Color.FromArgb(255, 220, 236, 201);
    sched1.DataStorage.ResourceStorage.Resources.Add(res1);

    // add some contacts
    Contact cnt = new Contact();
    cnt.Text = "Andy Garcia";
    sched1.DataStorage.ContactStorage.Contacts.Add(cnt);
    Contact cnt1 = new Contact();
    cnt1.Text = "Nancy Drew";
    sched1.DataStorage.ContactStorage.Contacts.Add(cnt1);
    Contact cnt2 = new Contact();
    cnt2.Text = "Robert Clark";
    sched1.DataStorage.ContactStorage.Contacts.Add(cnt2);

    // add sample appointments
    Appointment app = new Appointment(DateTime.Today.AddHours(12), TimeSpan.FromHours(1));
    app.Subject = "Sales meeting";
    sched1.DataStorage.AppointmentStorage.Appointments.Add(app);
    app.Resources.Add(res);
    app.Links.Add(cnt1);
    app.Links.Add(cnt2);

    app = new Appointment(DateTime.Today.AddHours(14), TimeSpan.FromHours(3));
    app.Subject = "Retirement Planning Session";
    app.Body = "A retirement planning education session. Please attend if possible.";
    sched1.DataStorage.AppointmentStorage.Appointments.Add(app);
    app.Resources.Add(res1);
    app.Links.Add(cnt1);
    app.Links.Add(cnt2);
    app.Links.Add(cnt);

    app = new Appointment(DateTime.Today.AddHours(10), TimeSpan.FromMinutes(15));
    app.Subject = "Conference call";
    sched1.DataStorage.AppointmentStorage.Appointments.Add(app);
    app.Links.Add(cnt);

    sched1.StyleChanged += new EventHandler<RoutedEventArgs>(sched1_StyleChanged);
    sched1_StyleChanged(null, null);
}

void sched1_StyleChanged(object sender, RoutedEventArgs e)
{
    // update toolbar buttons state according to the current C1Scheduler view
    switch (sched1.ViewType)
    {
        case C1.WPF.Schedule.ViewType.Day:
            btnDay.IsChecked = true;
```

```
            break;
        case C1.WPF.Schedule.ViewType.Month:
            btnMonth.IsChecked = true;
            break;
        case C1.WPF.Schedule.ViewType.TimeLine:
            btnTimeLine.IsChecked = true;
            break;
        case C1.WPF.Schedule.ViewType.Week:
            btnWeek.IsChecked = true;
            break;
        case C1.WPF.Schedule.ViewType.WorkingWeek:
            btnWorkWeek.IsChecked = true;
            break;
    }
}

private void ComboBox_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    //change Group By basis of combo-box selection
    string str = (string)cmbGroup.SelectedItem;
    if (str == "None")
    {
        sched1.GroupBy = string.Empty;
    }
    else
    {
        sched1.GroupBy = str;
    }
}
```

エンドユーザーの操作

実行時に予定を作成するには、予定を開始する時刻をダブルクリックして［**予定**］ダイアログボックスを開きます。予定の詳細については、「予定」を参照してください。



予定の追加と保存

予定を**スケジュール**に追加するには、［**予定**］ダイアログボックスを使用します。

1. 予定の時刻をダブルクリックして［**予定**］ダイアログボックスを開きます。または、単一［Enter］キーを押下して新しい［予定］を作成します。
2. **タイトル**、**場所**、および**予定**に割り当てる追加情報を指定します。
3. ［**保存して閉じる**］ボタンをクリックして、新しい予定をスケジュールに追加します。

Appointments can be also created programmatically using the **Appointment** class and its properties such as, **Start**, **Duration**, **Body**, etc. and added to the Scheduler using the **Add** method as showcased in the following code.

C#

```
// Showing Appointment
var appointment = new Appointment();
appointment.Start = new DateTime(DateTime.Now.Year, DateTime.Now.Month, DateTime.Now.Day,
1, 0, 0);
appointment.Duration = TimeSpan.FromHours(1.5);
appointment.Subject = "Doctor Appointment (Hospital)";
appointment.Body = "Meeting with Therapist";
//Adding appointment to scheduler.
scheduler.DataStorage.AppointmentStorage.Appointments.Add(appointment);
```

Back to Top

予定の編集

予定をスケジュールに変更して更新するには、［予定］ダイアログボックスを使用します。

1. [F2]キーを使用してタイトルを編集し、[Enter]キーで保存します。または[Esc]キーで変更を保存しないようにします。
2. 存在の[**予定**]をダブルクリックして［**予定**］ダイアログボックスを開きます。または、[Enter]キーを押下して選択状態にある[予定]の［予定］ダイアログボックスを開きます。
3. ［**予定**］ダイアログボックス内様々のフィールドに編集を行って[予定]の右端にある[**保存**]ボタンまたは[Ctrl+S]キーでスケジュールに[予定]をを更新します。

You can also edit an appointment in the Scheduler programmatically. For doing so, you can add a Button control to edit an appointment and a Scheduler control to your application as shown in the following steps:

1. Add a Button control to edit an appointment and a Scheduler control to your application as shown in the following code:

XAML

```
<Button x:Name="btnEdit" Click="btnEdit_Click" Margin="5"
HorizontalAlignment="Left">Edit Selected Appointment</Button>
<c1:C1Scheduler x:Name="scheduler" Grid.Row="1"></c1:C1Scheduler>
```

2. Switch to the code view and add the following code to add an appointment to the Scheduler and edit the appointment on the button click.

C#

```
public EditingAppointment()
{
    InitializeComponent();
    InitAppointments();
}

private void InitAppointments()
{
    for (int day = 1; day <= 7; day++)
    {
        scheduler.DataStorage.AppointmentStorage.Appointments.Add(new Appointment()
        {
            Start = new DateTime(2021, DateTime.Now.Month, day),
            Duration = TimeSpan.FromHours(1),
            Subject = $"Test Appointment {day}",
            Body = "New Appointment"
        });
    }
}

private void btnEdit_Click(object sender, RoutedEventArgs e)
{
    if (scheduler.SelectedAppointment == null)
        MessageBox.Show("No Appointment selected.", "Select Appointment",
MessageBoxButton.OK, MessageBoxImage.Information);
    else
    {
        scheduler.EditAppointmentDialog(scheduler.SelectedAppointment); // Opens edit
dialog for provided appointment
```

```
        scheduler.SelectedAppointment = null;
    }
}
```

Back to Top

予定の削除

［**予定の編集**］ダイアログボックスを使用して、選択した予定をスケジュールから簡単に削除することができます。

1. スケジュールから削除する予定を選択します。
2. キーボードの［**Delete**］キーを押します。その予定がスケジュールから削除されます。

Scheduler allows you to delete an appointment programmatically as well. For doing so, you can add a Button control to delete an appointment and a Scheduler control to your application as shown in the following steps:

1. Add a Button control to edit an appointment and a Scheduler control to your application as shown in the following code:

XAML

```
<Button x:Name="btnDelete" Click="btnDelete_Click" Margin="5"
HorizontalAlignment="Left">Delete Selected Appointment</Button>
<c1:C1Scheduler x:Name="scheduler" Grid.Row="1"></c1:C1Scheduler>
```

2. Switch to the code view and add the following code to add an appointment to the Scheduler and edit the appointment on the button click.

C#

```
public DeleteAppointment()
{
    InitializeComponent();
    InitAppointments();
}

private void InitAppointments()
{
    for (int day = 1; day <= 7; day++)
    {
        scheduler.DataStorage.AppointmentStorage.Appointments.Add(new Appointment()
        {
            Start = new DateTime(2021, DateTime.Now.Month, day),
            Duration = TimeSpan.FromHours(1),
            Subject = $"Test Appointment {day}",
            Body = "New Appointment"
        });
    }
}

private void btnDelete_Click(object sender, RoutedEventArgs e)
{
    if (scheduler.SelectedAppointment == null)
        MessageBox.Show("No Appointment selected.", "Select Appointment",
```

```
MessageBoxButton.OK, MessageBoxImage.Information);
    else
    {
        scheduler.SelectedAppointment.Delete(); // Deletes the appointment from the
scheduler.
        scheduler.SelectedAppointment = null;
    }
}
```

Back to Top

予定の繰り返し

特定の間隔で予定が繰り返されるように設定することができます。予定は日、週、月、または年の単位で繰り返すことができます。

1. 予定の時刻をダブルクリックして新しい予定を追加するか、既存の予定をダブルクリックします。［**予定**］ダイアログボックスが表示されます。
2. ［**定期的な予定**］ボタンをクリックします。［**定期的な予定の設定**］ダイアログボックスが表示されます。

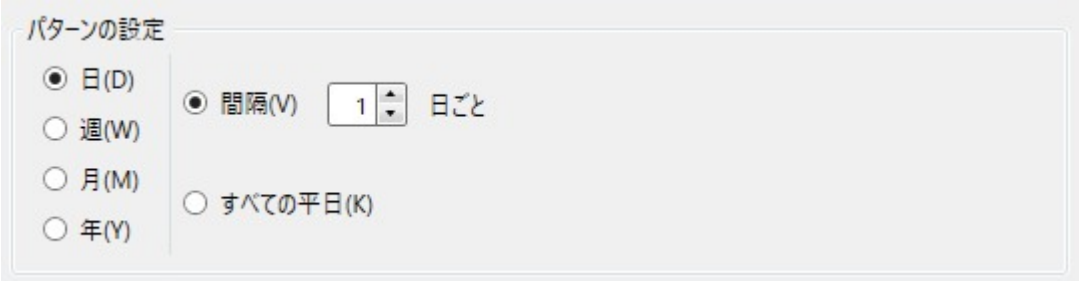
3. 繰り返しパターンを設定します。

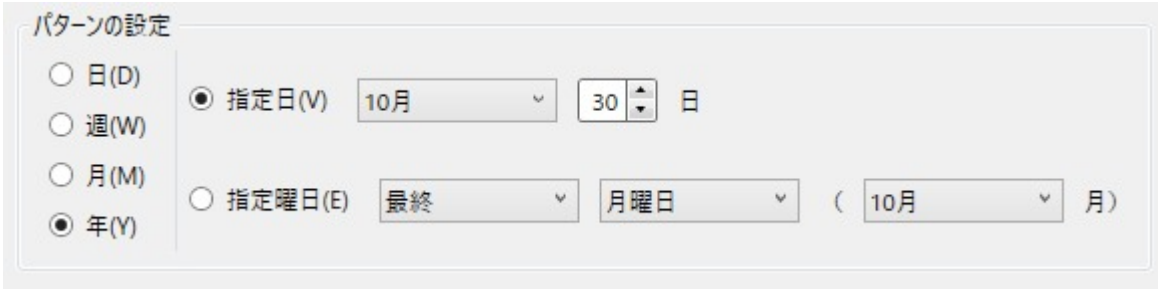
時間の設定

［時間の設定］グループ内のプロパティを指定して、予定の開始時刻、終了時刻、および予定の時間の長さを設定できます。

パターンの設定

［**パターンの設定**］グループの設定は、予定を繰り返す間隔（**日単位、週単位、月単位または年単位**）によって異なります。

日	［**日**］を指定すると、指定した日数ごとに、または勤務日にのみ予定が繰り返されるように設定できます。  たとえば、予定が２日ごとに繰り返されるように設定した場合、その予定は１日おきに表示されます。予定が勤務日に繰り返されるように設定した場合、デフォルトでは、予定は月曜日から金曜日にのみ表示されます。
週	［**週**］を指定すると、週単位で間隔を空けて、指定した日に予定が繰り返されるように設定できます。  たとえば、予定が**２週間ごとに**繰り返されるように設定し、［**火曜日**］と［**木曜日**］を選択した場合、その予定は１週間おきに火曜日と木曜日に繰り返されます。
月	［**月**］を指定すると、月単位で間隔を空けて、指定した日に、または指定した曜日に予定が繰り返されるように設定できます。  たとえば、予定が**２か月に１回、８日**に繰り返されるように設定した場合、その予定は１か月おきに８日に表示されます。予定が**２か月に１回、第３月曜日**に繰り返されるように設定した場合、その予定は１か月おきに第３月曜日に表示されます。
年	［**年**］を指定すると、指定した月の指定した日、または曜日に予定が繰り返されるように設定できます。



たとえば、予定が**1月2日**に繰り返されるように設定した場合、その予定は毎年1月2日に表示されます。この設定は、誕生日や記念日に最適です。予定が1月の第1金曜日に繰り返されるように設定した場合、その予定は毎年**1月の第1金曜日**に表示されます。

期間

[**期間**] グループを使用して、繰り返しの期間を設定できます。

[**開始**] ドロップダウンカレンダーを使用して、繰り返しを開始する日付を選択することができます。終了日を指定する際は、以下の3つのオプションを使用できます。

- [**終了日未定**] を指定すると、予定が無期限に繰り返されます。
- [**反復回数**] に0回を指定すると、指定された回数だけ予定が繰り返されます。たとえば、予定を毎日繰り返す場合、[**反復回数**] に25回を設定すると、予定を25日間毎日繰り返すことができます。
- [**終了日**] を指定すると、予定は指定した日まで繰り返されます。

4. [**保存して閉じる**] をクリックして、[**定期的な予定の設定**] ダイアログボックスを閉じます。

Recurring appointments can also be created programmatically using the following code:

C#

```
var appointment = new Appointment();
appointment.Start = DateTime.Now;
appointment.Duration = TimeSpan.FromHours(1);
appointment.Subject = "Finance Meeting";
appointment.Body = "Meeting with Mr. Willson";
appointment.Importance = ImportanceEnum.High;

// Setting Recurring pattern for appointment.
var recurrencePattern = appointment.GetRecurrencePattern();
recurrencePattern.RecurrenceType = RecurrenceTypeEnum.Daily; // will occur daily
recurrencePattern.PatternStartDate = DateTime.Now;
recurrencePattern.Occurrences = 10; // will occur for 10 days
```

```
// Adding appointment to scheduler.
scheduler.DataStorage.AppointmentStorage.Appointments.Add(appointment);
```

Back to Top

Keyboard Shortcuts for Appointment

Scheduler provides keyboard options for the following actions:

Key	Action
Enter	Creates new in-place appointment or opens the **Appointment** dialog for an existing selected appointment.
Esc	Cancels in-place appointment editing.
F2	Turns on in-place appointment editing for selected appointment.
Tab (SHIFT+Tab)	Moves selection and keyboard focus to the next or previous appointment in the current view

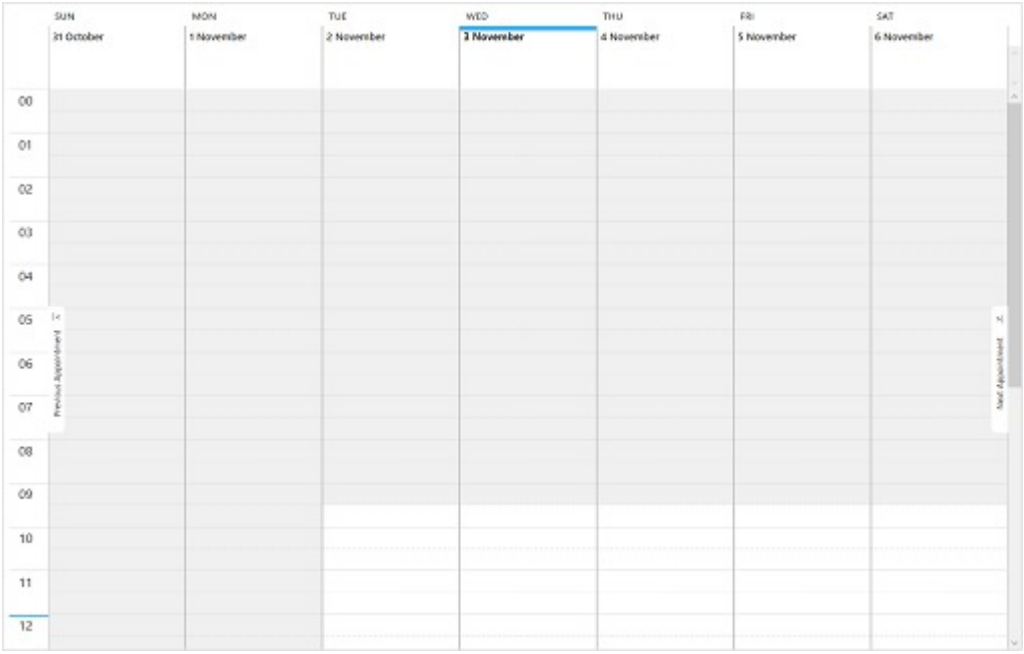
Back to Top

カレンダーの設定

Scheduler は、**CalendarHelper** クラスの該当プロパティで定義されたカレンダー設定を使用します。たとえば、**CalendarHelper** クラスで稼働日を指定することにより、週間勤務日を作成できます。**CalendarHelper** クラス用のXAMLは三つのコントロールとも、同じものが使用されていることにご注意ください。

XAML

```
<c1:C1Scheduler x:Name="scheduler1" ViewType="Week">
    <c1:C1Scheduler.CalendarHelper>
        <c1:CalendarHelper WeekStart="Sunday"
            EndDayTime="18:20:00" StartDayTime="09:20:00"
            WorkDays="Tuesday,Wednesday,Thursday,Friday,Saturday">
        </c1:CalendarHelper>
    </c1:C1Scheduler.CalendarHelper>
</c1:C1Scheduler>
```



使用可能なカレンダー設定は以下のようになります。

CalendarHelper プロパティ	説明
WeekStart	週の初めの日を決定する DayOfWeek 値を取得または設定します。デフォルトはシステム設定です。
WorkDays	１週間の勤務日を含む WorkDays オブジェクトを取得または設定します。
StartDayTime	勤務時間の開始を指定する TimeSpan 値を取得または設定します。
EndDayTime	勤務時間の終了を指定する TimeSpan 値を取得または設定します。
Holidays	休日（勤務日以外の日と週末）のリストを保持する ObservableCollection(T) オブジェクトを取得または設定します。
WeekendExceptions	働く必要がある週末の日のリストを保持する ObservableCollection (T) オブジェクトを取

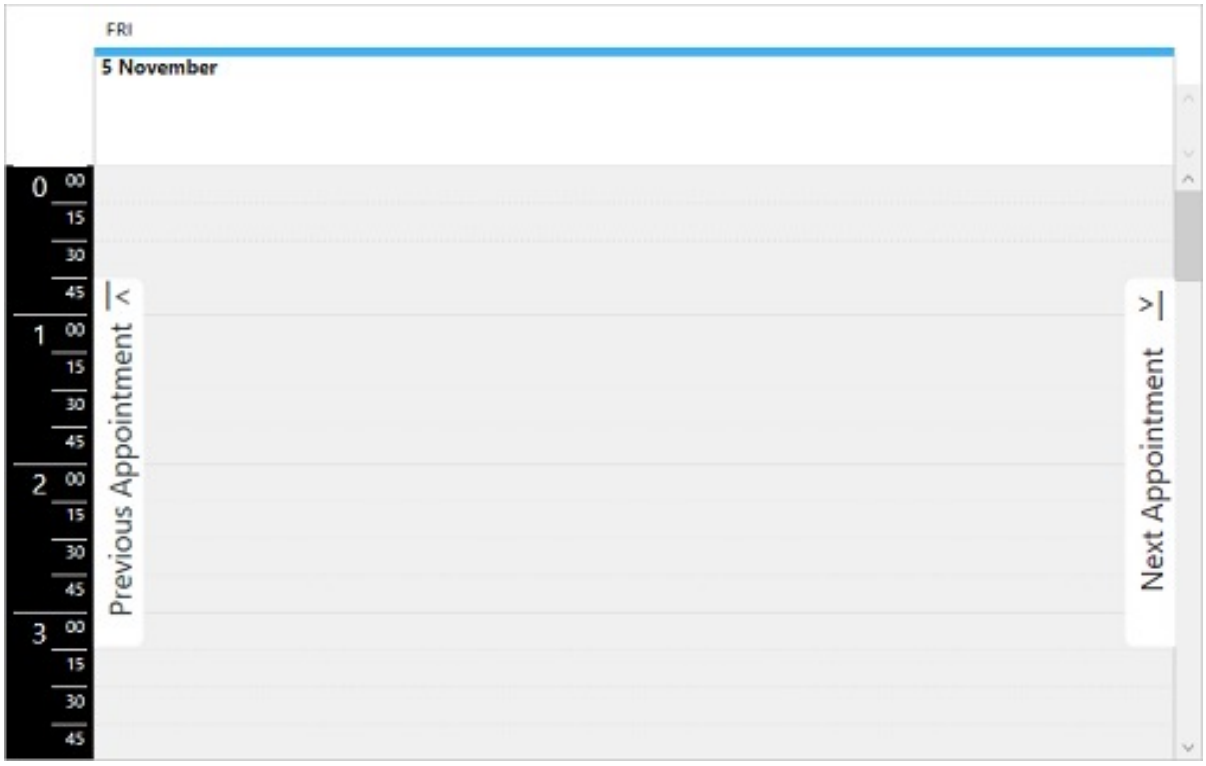
	得または設定します。
FullMonthNames	カルチャ固有の月の完全名の配列を取得します。

Scheduler のカスタマイズ

The following topics assume that you are familiar with programming in C# and provide step-by-step instructions; no prior knowledge of **C1Scheduler** is needed. By following the steps outlined in these topics, you will be able to create projects demonstrating **C1Scheduler** features. You are encouraged to create, run these projects, and experiment with your own modifications.

時間列のカスタマイズ

Scheduler コントロールの時間列をカスタマイズするには、組み込みの *C1Scheduler_TimeRuler_Template* を編集する必要があります。以下の例では、**TimeRuler** データテンプレート（C1Scheduler_TimeRuler_Templateキーを使用）を編集し、時間列に時間を表示する方法と、時間列の背景色を変更します。



時間列をカスタマイズするには、次のコードを使用します。

C#

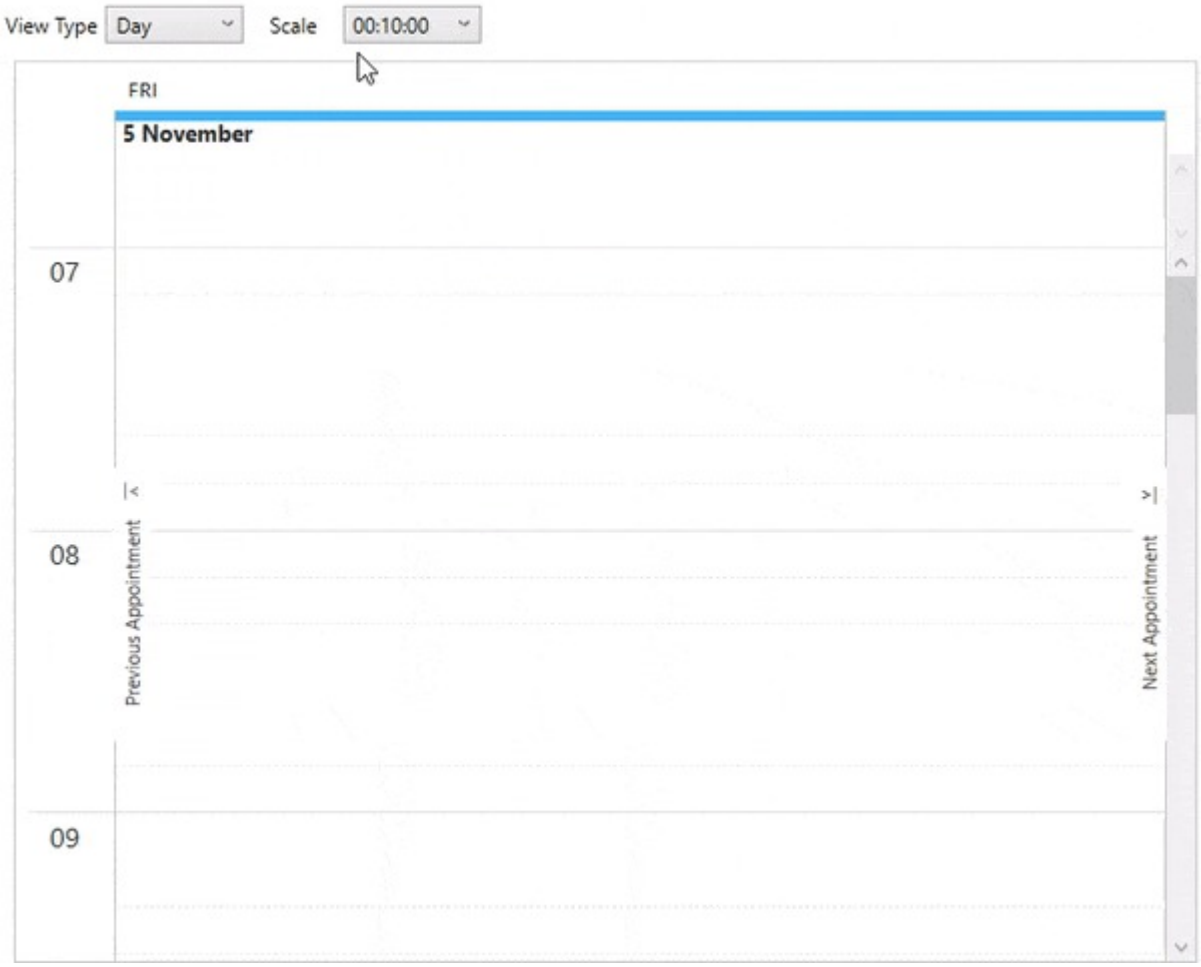
```
<c1sched:C1Scheduler Name="Scheduler" Grid.Column="2" Grid.Row="1" FontSize="20"
                     ViewType="Day">
    <c1sched:C1Scheduler.Resources>
        <!-- determines the template used for one hour of a time ruler in a Day view -->
        <DataTemplate x:Key="C1Scheduler_TimeRuler_Template">
            <Grid Name="OneHourGrid" Background="Black">
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="30" />
                    <ColumnDefinition MinWidth="20" SharedSizeGroup="Minutes" />
                </Grid.ColumnDefinitions>
                <c1sched:TimeRulerHelper Interval="{Binding}" x:Name="helper"/>
                <Border BorderThickness="0,1px,0,0" Margin="4,0,0,0" Grid.Column="0"
SnapsToDevicePixels="True" Visibility="{Binding IsZeroMinute, ElementName=helper,
Converter={x:Static c1sched:BooleanToVisibilityConverter.Default}}" BorderBrush="White"
HorizontalAlignment="Right" VerticalAlignment="Top" MinWidth="25">
```

```
                    <TextBlock Text="{Binding StartTime.Hour}" FontSize="18"
FontFamily="Segoe UI" Foreground="White" HorizontalAlignment="Right" Padding="3,0,3,0" />
                </Border>
                <Border BorderThickness="0,1px,0,0" Grid.Column="1" Margin="0,0,4,0"
SnapsToDevicePixels="True" BorderBrush="White" />
                <TextBlock FontSize="11" Grid.Column="1" Text="{Binding Minutes,
ElementName=helper}" Foreground="White" HorizontalAlignment="Right" Padding="3,2,2,0"
Margin="0,0,4,0">
                </TextBlock>
            </Grid>
        </DataTemplate>
    </c1sched:C1Scheduler.Resources>
</c1sched:C1Scheduler>
```

さまざまなビューの時間間隔のカスタマイズ

Scheduler では、1 日より短い表示間隔を含むビュー（日ビュー、週間勤務日ビュー、週ビュー、タイムラインビューなど）の時間間隔をカスタマイズできます。C1Scheduler クラスは、XAML で時間間隔を設定するための SmallVisualIntervalScale プロパティ（VisualIntervalScale プロパティに依存する依存プロパティ）を提供します。

SmallVisualIntervalScale プロパティに特定の値を設定するか、このプロパティを scale などのスケジューラ要素に連結することで、ビューがエンドユーザーによって変更されるたびに時間間隔を更新できます。デフォルトでは、SmallVisualIntervalScale プロパティは TimeSpan.Zero に設定されています。このため、SmallVisualIntervalScale プロパティに値が設定されていない場合、あるいはこのプロパティが XAML で何らかの要素に連結されていない場合は、**VisualScaleInterval** プロパティだけが有効です。



To add different views to the Scheduler and customize the time span for all the view with visual intervals, follow these steps:

1. In the XAML designer, add a StackPanel inside the grid and then add two TextBlocks and ComboBoxes to the StackPanel to display view type and scale dropdowns.
2. 次の XAML コードスニペットは、SmallVisualIntervalScale プロパティを scale 要素に連結する方法を示します。

XAML

```
<StackPanel Orientation="Horizontal" Grid.Row="0" Margin="5">
    <TextBlock VerticalAlignment="Center">View Type</TextBlock>
    <ComboBox x:Name="viewType" Width="80" Margin="5">
        <c1sched:ViewType>Day</c1sched:ViewType>
        <c1sched:ViewType>Week</c1sched:ViewType>
        <c1sched:ViewType>WorkingWeek</c1sched:ViewType>
    </ComboBox>
    <TextBlock VerticalAlignment="Center" Margin="10, 0">Scale</TextBlock>
    <ComboBox x:Name="scale" Width="80" Margin="5"></ComboBox>
</StackPanel>

<!-- We need to use SmallVisualIntervalScale property for specifying the required
time span.-->
<c1sched:C1Scheduler x:Name="scheduler1" Grid.Row="1" Margin="10 0 0 0"
BorderThickness="1"
                     ShowWorkTimeOnly="True"
```

```
                    ViewType="{Binding SelectedItem, ElementName=viewType,
Mode=TwoWay}"
                    SmallVisualIntervalScale="{Binding SelectedItem,
ElementName=scale, Mode=TwoWay}"/>
```

3. 次の C# コードを使用して、XAML の操作ロジックで scale 要素を初期化します。上記のコード例で使用されている SmallVisualIntervalScale プロパティは、下記のコードで初期化される scale 要素に連結します。

C#

```
// initialize time scale combo
scale.Items.Add(TimeSpan.FromMinutes(10));
scale.Items.Add(TimeSpan.FromMinutes(15));
scale.Items.Add(TimeSpan.FromMinutes(20));
scale.Items.Add(TimeSpan.FromMinutes(30));
scale.Items.Add(TimeSpan.FromMinutes(60));
scale.Items.Add(TimeSpan.FromHours(2));
viewType.SelectedIndex = 0;
scale.SelectedIndex = 0;
```

[稼働日] ビューの曜日の設定

[**稼働日**] ビューに表示する曜日を指定するには、**CalendarHelper** プロパティから WorkDays を設定します。

Blend を使用する場合

Blend で［稼働日］ビューの曜日を設定するには、次の手順に従います。

1. **C1Scheduler** コントロールをウィンドウに追加して選択します。
2. ［**デザイン**］ビューの［プロパティ］パネルで、**外観** ノードを展開します。
3. ［**Style**］の横にあるドロップダウン矢印をクリックし、［**WorkingWeekStyle**］を選択します。
4. **Calendar** ノードを展開し、**CalendarHelper** ノードを展開します。
5. ［**WorkDays**］の横にあるドロップダウン矢印をクリックし、稼働日として表示する曜日の横にあるチェックボックスをオンにします。

Visual Studio を使用する場合

Visual Studio で［**稼働日**］ビューの曜日を設定するには、次の手順に従います。

1. **C1Scheduler** コントロールをウィンドウに追加します。
2. ［**プロパティ**］ウィンドウで、**Style** プロパティを［**WorkingWeekStyle**］に設定します。
3. **CalendarHelper** プロパティを展開し、［**WorkDays**］の横にあるドロップダウン矢印をクリックします。
4. 稼働日として表示する曜日の横にあるチェックボックスをオンにします。

XAML の使用

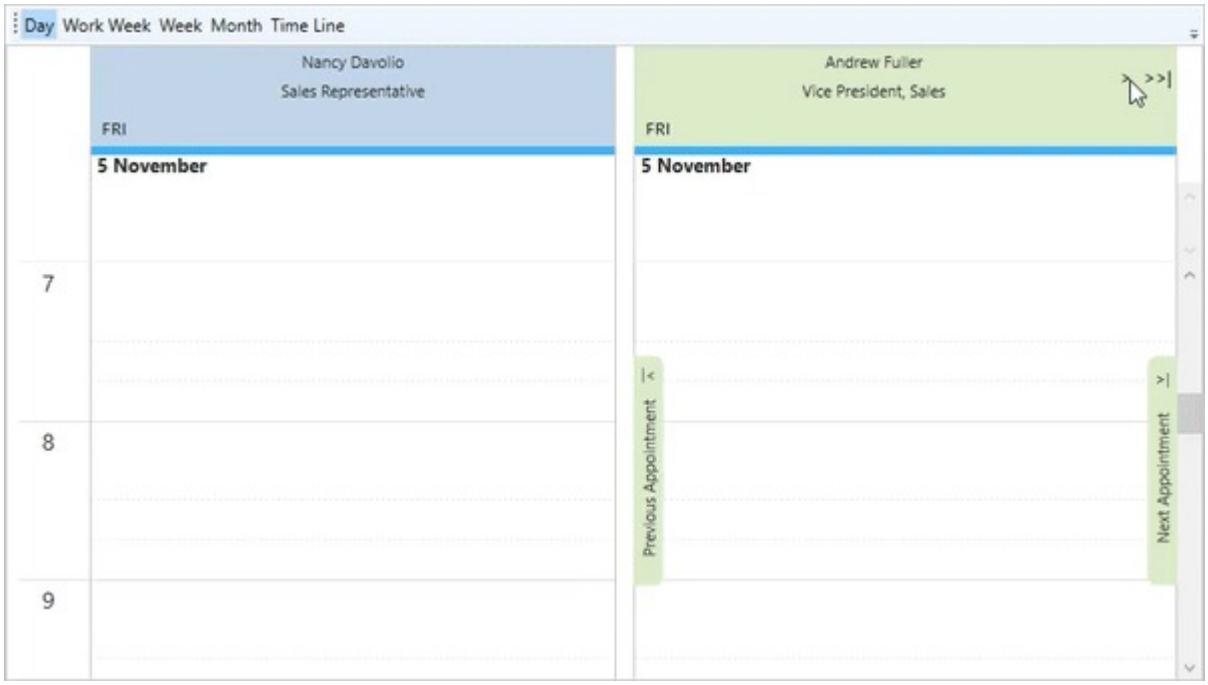
次の XAML は、［**稼働日**］ビューの曜日を設定します。

XAML

```
<c1sched:C1Scheduler x:Name="scheduler1" Theme="{DynamicResource {ComponentResourceKey
TypeInTargetAssembly=c1sched:C1Scheduler, ResourceId=Office2007.Default}}" Style="
{DynamicResource {ComponentResourceKey TypeInTargetAssembly=c1sched:C1Scheduler,
ResourceId=WorkingWeekStyle}}">
       <c1sched:C1Scheduler.CalendarHelper>
             <c1sched:CalendarHelper Culture="Japanese" WeekStart="Sunday"
                    EndDayTime="18:20:00" StartDayTime="09:20:00"
                    WorkDays="Tuesday,Wednesday,Thursday,Friday,Saturday">
             </c1sched:CalendarHelper>
       </c1sched:C1Scheduler.CalendarHelper>
</c1sched:C1Scheduler>
```

複数ユーザースケジュールの作成

Using Scheduler control, you can display schedules for multiple users. For this, you need to perform the following steps:



リソースと Scheduler コントロールの作成

この手順では、アプリケーションのリソースとデータ連結を作成します。また、Scheduler コントロールを追加してカスタマイズします。

次の手順に従います。

1. Set up the application and configure the data source. For detailed steps, see クイックスタート.
2. In the XAML view, add a *<Window.Resources></Window.Resources>* tag and add a set of *<DataTemplate></DataTemplate>* tags to it.
3. Specify Key property in the *<DataTemplate>* tag to control the group header for the custom style using the following XAML markup:

   XAML

   ```
   <DataTemplate x:Key="myCustomGroupHeaderTemplate"></DataTemplate>
   ```

4. Insert the following markup between the *<DataTemplate>* tags to add the DataTemplate resources:

XAML

```
<DataTemplate.Resources>
    <ControlTemplate x:Key="looklessButton" TargetType="{x:Type Button}">
        <Border>
            <ContentPresenter Margin="4,0" VerticalAlignment="Center" />
        </Border>
    </ControlTemplate>
</DataTemplate.Resources>
```

5. *<DataTemplate.Resources>* マークアップの下に、*<Grid>* *</Grid>* タグセットを追加します。最初の *<Grid>* タグをクリックし、タグに SnapsToDevicePixels="True" を追加します。
6. 次のマークアップを使用して、グリッドコンポーネントに行および列定義を追加します。

XAML

```
<Grid.ColumnDefinitions>
     <ColumnDefinition Width="Auto"/>
     <ColumnDefinition Width="Auto"/>
     <ColumnDefinition/>
     <ColumnDefinition Width="Auto"/>
     <ColumnDefinition Width="Auto"/>
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
     <RowDefinition />
     <RowDefinition />
</Grid.RowDefinitions>
```

7. Create the **C1BrushBuilders** and the **Border** control for the **Grid** component by adding the following code beneath the *<Grid.RowDefinitions></Grid.RowDefinitions>* tags :

XAML

```
<c1:C1BrushBuilder x:Name="Background"  Input="{Binding Background}" />
<c1:C1BrushBuilder x:Name="BorderBrush" Input="{Binding Background}" />
<Border VerticalAlignment="Stretch" HorizontalAlignment="Stretch" Grid.Column="2"
Grid.RowSpan="2"
        BorderThickness="1,0,1,0" BorderBrush="{Binding Output,
ElementName=BorderBrush}"
        Background="{Binding Output, ElementName=Background}" />
```

8. Add the **ButtonTemplates** and the **TextBlocks** after the <Border/> tag to control the navigation and display for the Scheduler view:

XAML

```
<!-- 最初のグループに移動します -->
<Button Template="{StaticResource looklessButton}" Content="|<<" Grid.Column="0"
Grid.RowSpan="2" VerticalAlignment="Center" FontSize="12"
Command="c1:C1Scheduler.NavigateToPreviousGroupCommand" CommandParameter="Home"
CommandTarget="{Binding Scheduler}" Visibility="{Binding ShowPreviousButton,
Converter={x:Static c1:BooleanToVisibilityConverter.Default}}" />
<!-- 前のグループに移動します -->
<Button Template="{StaticResource looklessButton}" Content="<" Grid.Column="1"
Grid.RowSpan="2" VerticalAlignment="Center" FontSize="12"
Command="c1:C1Scheduler.NavigateToPreviousGroupCommand" CommandTarget="{Binding
Scheduler}" Visibility="{Binding ShowPreviousButton, Converter={x:Static
c1:BooleanToVisibilityConverter.Default}}" />
```

```
<!-- 次のグループに移動します -->
<Button Template="{StaticResource looklessButton}" Content=">" Grid.Column="3"
Grid.RowSpan="2" VerticalAlignment="Center" FontSize="12"
Command="c1:C1Scheduler.NavigateToNextGroupCommand" CommandTarget="{Binding
Scheduler}" Visibility="{Binding ShowNextButton, Converter={x:Static
c1:BooleanToVisibilityConverter.Default}}" />
<!-- 最後のグループに移動します -->
<Button Template="{StaticResource looklessButton}" Content=">>|" Grid.Column="4"
Grid.RowSpan="2" VerticalAlignment="Center" FontSize="12"
Command="c1:C1Scheduler.NavigateToNextGroupCommand" CommandParameter="End"
CommandTarget="{Binding Scheduler}" Visibility="{Binding ShowNextButton, Converter=
{x:Static c1:BooleanToVisibilityConverter.Default}}" />
<TextBlock Foreground="{Binding Path=Scheduler.Foreground}" Margin="10,3"
Grid.Column="2" Visibility="{Binding IsSelected, Converter={x:Static
c1:BooleanToVisibilityConverter.Default}, ConverterParameter=Invert}" Text="{Binding
DisplayName}" VerticalAlignment="Center" HorizontalAlignment="Center" />
<TextBlock Foreground="{Binding Path=Scheduler.Foreground}" Margin="10,3"
Grid.Column="2" FontWeight="Bold" Visibility="{Binding IsSelected, Converter=
{x:Static c1:BooleanToVisibilityConverter.Default}}" Text="{Binding DisplayName}"
VerticalAlignment="Center" HorizontalAlignment="Center" />
<!-- show additional info from the EmployeesRow -->
<TextBlock Foreground="{Binding Path=Scheduler.Foreground}" Margin="10,3"
Grid.Column="2" Grid.Row="1" Text="{Binding Path=Tag.Title}"
VerticalAlignment="Center" HorizontalAlignment="Center" />
```

9. **TimeLine** スタイルのグループヘッダーを制御するために、別の *<DataTemplate>* を作成します。

XAML

```
<!-- TimeLineスタイルに異なるグループヘッダーを使用します -->
<DataTemplate x:Key="myCustomTimeLineGroupHeaderTemplate">
    <Grid IsHitTestVisible="False">
        <c1:C1BrushBuilder x:Name="Background"  Input="{Binding Background}" />
        <c1:C1BrushBuilder x:Name="BorderBrush"  Input="{Binding Background}" />
        <Border VerticalAlignment="Stretch" HorizontalAlignment="Stretch"
BorderThickness="4,1,0,1" BorderBrush="{Binding Output, ElementName=BorderBrush}"
Background="{Binding Output, ElementName=Background}">
            <StackPanel VerticalAlignment="Center" HorizontalAlignment="Center">
                <TextBlock TextWrapping="Wrap" Foreground="{Binding
Path=Scheduler.Foreground}" Margin="2" Text="{Binding DisplayName}"
HorizontalAlignment="Center" />
                <!-- show additional info from the EmployeesRow -->
                <TextBlock TextWrapping="Wrap" Foreground="{Binding
Path=Scheduler.Foreground}" Margin="2" Text="{Binding Path=Tag[Title]}"
HorizontalAlignment="Center" />
            </StackPanel>
        </Border>
    </Grid>
</DataTemplate>
```

10. *</Window.Resources>* 終了タグの下に、*<Grid>* *</Grid>* タグセットを追加します。
11. 次のマークアップを *<Grid>*</Grid> タグの間に挿入します。

XAML

```
<Grid.RowDefinitions>
```

```
    <RowDefinition Height="Auto" />
    <RowDefinition />
    <RowDefinition Height="Auto" />
</Grid.RowDefinitions>
```

12. *<Grid>*タグ内で次のコードを使用して、アプリケーションのツールバーを作成します。

XAML

```
<ToolBar Grid.Row="0" Grid.ColumnSpan="2">
    <RadioButton x:Name="btnDay" Content="Day" CommandTarget="{Binding
ElementName=Scheduler}" Command="c1:C1Scheduler.ChangeStyleCommand"
CommandParameter="{Binding Path=OneDayStyle, ElementName=Scheduler}" />
    <RadioButton x:Name="btnWorkWeek" Content="Work Week" CommandTarget="{Binding
ElementName=Scheduler}" Command="c1:C1Scheduler.ChangeStyleCommand"
CommandParameter="{Binding Path=WorkingWeekStyle, ElementName=Scheduler}" />
    <RadioButton x:Name="btnWeek" Content="Week" CommandTarget="{Binding
ElementName=Scheduler}" Command="c1:C1Scheduler.ChangeStyleCommand"
CommandParameter="{Binding Path=WeekStyle, ElementName=Scheduler}" />
    <RadioButton x:Name="btnMonth" Content="Month" CommandTarget="{Binding
ElementName=Scheduler}" Command="c1:C1Scheduler.ChangeStyleCommand"
CommandParameter="{Binding Path=MonthStyle, ElementName=Scheduler}" />
    <RadioButton x:Name="btnTimeLine" Content="Time Line" CommandTarget="{Binding
ElementName=Scheduler}" Command="c1:C1Scheduler.ChangeStyleCommand"
CommandParameter="{Binding Path=TimeLineStyle, ElementName=Scheduler}" />
</ToolBar>
<ListBox Grid.Column="0" Grid.Row="2" x:Name="lstUsers" MinHeight="100" Margin="2"
ItemsSource="{Binding GroupItems, ElementName=Scheduler}">
    <ListBox.ItemTemplate>
        <DataTemplate>
            <CheckBox Margin="2" Content="{Binding}" Tag="{Binding}" IsChecked="
{Binding IsChecked}" />
        </DataTemplate>
    </ListBox.ItemTemplate>
</ListBox>
```

13. Add Scheduler and create the bindings for your application:

XAML

```
<c1:C1Scheduler x:Name="Scheduler" GroupBy="Owner" GroupHeaderTemplate="
{StaticResource myCustomGroupHeaderTemplate}" GroupPageSize="2" Grid.Column="1"
Grid.Row="1" Grid.RowSpan="2" Style="{DynamicResource {ComponentResourceKey
ResourceId=OneDayStyle, TypeInTargetAssembly=c1:C1Scheduler}}">
    <c1:C1Scheduler.Settings>
        <c1:C1SchedulerSettings FirstVisibleTime="07:00:00"
AllowContactsEditing="False" AllowCategoriesEditing="False"
AllowCategoriesMultiSelection="False" />
    </c1:C1Scheduler.Settings>
</c1:C1Scheduler>
```

Back to Top

アプリケーションへのコードの追加

1. Navigate to the **Code** view and Import the following namespaces:

C#

```
using C1.WPF.Schedule;
using System.Collections.Specialized;
using C1.C1Schedule;
using System.Windows;
using System.Windows.Controls;
using SamplesName.C1NWindDataSetTableAdapters;
```

2. Add the following fields in the MainWindow class:

C#

```
private AppointmentsTableAdapter appointmentsTableAdapter = new
AppointmentsTableAdapter();
private EmployeesTableAdapter employeesTableAdapter = new EmployeesTableAdapter();
private CustomersTableAdapter customersTableAdapter = new CustomersTableAdapter();
private C1NWindDataSet dataSet = new C1NWindDataSet();
```

3. InitializeComponent() メソッドのすぐ下に、次のハンドラと呼び出しを追加して、データベースからデータを取得します。

C#

```
Scheduler.ReminderFire += new EventHandler(scheduler_ReminderFire);

Scheduler.GroupItems.CollectionChanged += new
NotifyCollectionChangedEventHandler(GroupItems_CollectionChanged);

//データベースからデータを取得します
this.employeesTableAdapter.Fill(dataSet.Employees);
this.customersTableAdapter.Fill(dataSet.Customers);
this.appointmentsTableAdapter.Fill(dataSet.Appointments);
```

4. AppointmentStorage のマッピングと DataSource を設定します。

C#

```
//AppointmentStorages のマッピングとデータソースを設定します
AppointmentStorage storage = Scheduler.DataStorage.AppointmentStorage;
storage.Mappings.AppointmentProperties.MappingName = "Properties";
storage.Mappings.Body.MappingName = "Description";
storage.Mappings.End.MappingName = "End";
storage.Mappings.IdMapping.MappingName = "AppointmentId";
storage.Mappings.Location.MappingName = "Location";
storage.Mappings.Start.MappingName = "Start";
storage.Mappings.Subject.MappingName = "Subject";
storage.Mappings.OwnerIndexMapping.MappingName = "Owner";
storage.DataSource = dataSet.Appointments;
```

5. OwnerStorage のマッピングと DataSource を設定します。

C#

```
//OwnerStorage のマッピングとデータソースを設定します
ContactStorage ownerStorage = Scheduler.DataStorage.OwnerStorage;
INotifyCollectionChanged)ownerStorage.Contacts).CollectionChanged += new
NotifyCollectionChangedEventHandler(Owners_CollectionChanged);
ownerStorage.Mappings.IndexMapping.MappingName = "EmployeeId";
ownerStorage.Mappings.TextMapping.MappingName = "FirstName";
ownerStorage.DataSource = dataSet.Employees;
```

6. ContactStorage のマッピングと DataSource を設定します。

C#

```
//ContactStorage のマッピングとデータソースを設定します
ContactStorage cntStorage = Scheduler.DataStorage.ContactStorage;
((INotifyCollectionChanged)cntStorage.Contacts).CollectionChanged += new
NotifyCollectionChangedEventHandler(Contacts_CollectionChanged);
cntStorage.Mappings.IdMapping.MappingName = "CustomerId";
cntStorage.Mappings.TextMapping.MappingName = "CompanyName";
cntStorage.DataSource = dataSet.Customers;
```

7. Add the following code to set the IsChecked property of the btnDay, create StyleChanged event for the scheduler control and MainWindowLoaded event.

C#

```
btnDay.IsChecked = true;
Scheduler.StyleChanged += new System.EventHandler<RoutedEventArgs>
(Scheduler_StyleChanged);
this.Loaded += MultiUser_Loaded;
```

8. Add the following methods to update the group items, owners and contacts.

C#

```
void GroupItems_CollectionChanged(object sender, NotifyCollectionChangedEventArgs e)
{
        foreach (SchedulerGroupItem group in Scheduler.GroupItems)
        {
                if (group.Owner != null)
                {
                        // SchedulerGroupItem.Tagプロパティをデータ行に設定します。 これ
により、xamlのデータ行フィールドをバインドに使用できます
                        int index = (int)group.Owner.Key[0];
                        C1NWindDataSet.EmployeesRow row =
dataSet.Employees.Rows.Find(index) as C1NWindDataSet.EmployeesRow;
                        group.Tag = row;
                }
        }
}

void Owners_CollectionChanged(object sender, NotifyCollectionChangedEventArgs e)
{
        if (e.Action == NotifyCollectionChangedAction.Add)
        {
                foreach (Contact cnt in e.NewItems)
                {
                        C1NWindDataSet.EmployeesRow row =
dataSet.Employees.Rows.Find(cnt.Key[0]) as C1NWindDataSet.EmployeesRow;
                        if (row != null)
                        {
                                // Contact.MenuCaptionをFirstNameおよびLastName文字列に
設定します
                                cnt.MenuCaption = row["FirstName"].ToString() + " " +
row["LastName"].ToString();
                        }
```

```
                }
        }
}

void Contacts_CollectionChanged(object sender, NotifyCollectionChangedEventArgs e)
{
        if (e.Action == NotifyCollectionChangedAction.Add)
        {
                foreach (Contact cnt in e.NewItems)
                {
                        C1NWindDataSet.CustomersRow row =
dataSet.Customers.Rows.Find(cnt.Key[0]) as C1NWindDataSet.CustomersRow;
                        if (row != null)
                        {
                                // Contact.MenuCaptionをCompanyNameおよびContactName文
字列に設定します
                                cnt.MenuCaption = row["CompanyName"].ToString() + "
(" + row["ContactName"].ToString() + ")";
                        }
                }
        }
}
```

9. Get the current window, save changes and prevent showing reminders using the following code:

C#

```
// 現在のウィンドウを取得します
                private void MultiUser_Loaded(object sender, RoutedEventArgs e)
                {
                        Window window = Window.GetWindow(this);
            window.Closing += Window_Closing;
                }
                // save changes
                private void Window_Closing(object? sender,
System.ComponentModel.CancelEventArgs e)
        {
                        this.appointmentsTableAdapter.Update(dataSet.Appointments);
                }

        // アラームを表示しないようにします
        private void scheduler_ReminderFire(object sender, ReminderActionEventArgs e)
                {
                        e.Handled = true;
                }
                private void Scheduler_StyleChanged(object sender, RoutedEventArgs e)
                {
                        if (Scheduler.Style == Scheduler.TimeLineStyle)
                        {
                                // update group header (use different headers for
TimeLine and other views)
                                Scheduler.GroupHeaderTemplate =
(DataTemplate)Resources["myCustomTimeLineGroupHeaderTemplate"];
                                btnTimeLine.IsChecked = true;
                        }
```

```
                        else
                        {
                                // グループヘッダーを更新します（TimeLineと他のビューに異
なるヘッダーを使用します）
                                Scheduler.GroupHeaderTemplate =
(DataTemplate)Resources["myCustomGroupHeaderTemplate"];
                                // 現在のC1Schedulerビューに従ってツールバーボタンの状態
を更新します
                                if (Scheduler.Style == Scheduler.WorkingWeekStyle)
                                {
                                        btnWorkWeek.IsChecked = true;
                                }
                                else if (Scheduler.Style == Scheduler.WeekStyle)
                                {
                                        btnWeek.IsChecked = true;
                                }
                                else if (Scheduler.Style == Scheduler.MonthStyle)
                                {
                                        btnMonth.IsChecked = true;
                                }
                                else
                                {
                                        btnDay.IsChecked = true;
                                }
                        }
                }
```

Back to Top

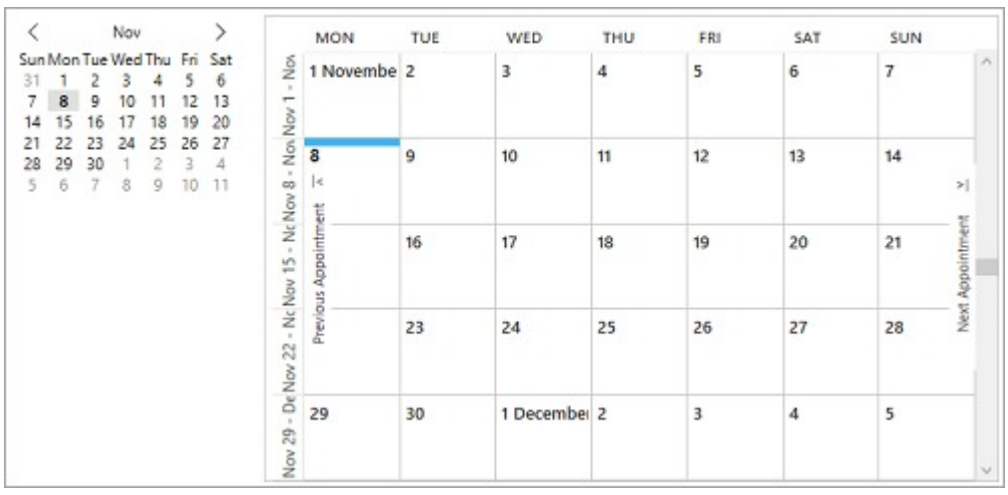
アプリケーションの実行

[**F5**] キーを押してアプリケーションを実行します。

Back to Top

スケジューラとカレンダーのリンク

このトピックでは、Visual Studio、および XAML を使用して、スケジュールを C1Calendar コントロールに連結する方法について説明します。



Scheduler コントロールを Calendar コントロールにリンクするには、次の手順に従います。

1. C1SchedulerおよびC1Calendarコントロールを XAMLデザイナに追加します。
2. C1Scheduler コントロールを選択します。
3. 必要に応じて、［**プロパティ**］ウィンドウの［名前］テキストボックスに「**C1Scheduler1**」と入力します。
4. In the XAML view, set its Margin, Vertical and Horizontal alignments according to the window size.
5. デザイナに追加した C1Calendar コントロールを選択します。
6. In the XAML view, edit the <c1:C1Calendar><c1:C1Calendar /> tag by adding the following code:

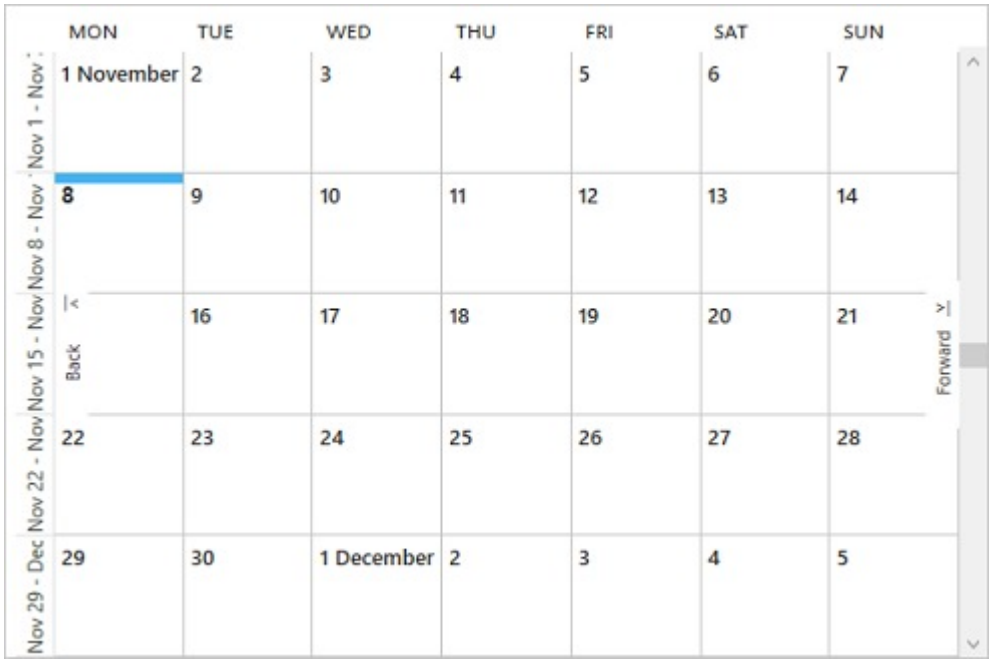
```
XAML
<c1:C1Calendar x:Name="c1Calendar1" SelectedDate="{Binding Path=SelectedDateTime,
ElementName=C1Scheduler1, Mode=TwoWay}" Margin="10,20,605,234"
HorizontalAlignment="Left" VerticalAlignment="Top"></c1:C1Calendar>
```

7. ［**F5**］キーを押してプロジェクトを実行し、C1Calendar で任意の日付を選択します。スケジュールの日付も同様に選択されます。

ナビゲーションペインのテキストの変更

ナビゲーションペインのテキストは、**NextAppointmentText** プロパティと **PreviousAppointmentText** プロパティを設定するだけで変更できます。

スケジュールは、次の図のようになります。



To change the navigation pane text at design time, follow these steps:

1. C1Scheduler コントロールをウィンドウに追加します。
2. ［プロパティ］ウィンドウで、NextAppointmentText プロパティを「次へ」に設定します。
3. PreviousAppointmentText プロパティを「前へ」に設定します。

XAML の使用

次の XAML は、NextAppointmentText プロパティと PreviousAppointmentText プロパティを設定します。

XAML

```
<c1:C1Scheduler Margin="15,15,0,12" Name="scheduler" NextAppointmentText="Forward"
PreviousAppointmentText="Back"></c1:C1Scheduler>
```

ネストされたプロパティへの値の割り当て

XAML では、ネストされたプロパティに値を割り当てることはできません。ただし、DataStorage のオブジェクトモデルは、その特性上、ネストされたプロパティを含んでいます。この制限に対処するには、NestedPropertySetter クラスを使用します。

> **メモ**: NestedPropertySetter クラスは、C1Scheduler と共に使用する場合にのみ機能します。

XAML で C1Scheduler 要素の子として配置されるこのクラスの要素は、**Property/Value** ペアで表されるセッターです。**Property** には、親 C1Scheduler 要素からプロパティへの相対パスを指定します。次の例では、**C1BindingSource.DataMember** プロパティがデータベースの **Appointments** テーブルに割り当てられます。**C1BindingSource.DataSource** は、次のようにプロジェクト内のデータセットリソースに設定されます。

XAML

```
<c1sched:C1Scheduler >
        <!-- マップの AppointmentStorage -->
        <c1sched:NestedPropertySetter
PropertyName="DataStorage.AppointmentStorage.DataMember" Value="Appointments"/>
        <c1sched:NestedPropertySetter
PropertyName="DataStorage.AppointmentStorage.DataSource" Value="{StaticResource dataSet}"/>
</c1sched:C1Scheduler >
```

ローカライズしたリソースのプロジェクトへの追加

Scheduler for WPF プロジェクトはローカライズ可能です。

ローカライズしたリソ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. **Scheduler for WPF** に含まれている .resx ファイルのコピーを作成します。デフォルトでは、これらのリソースファイルは、**<インストールフォルダ>\Misc\Resources\C1WPFScheduler** または **<インストールフォルダ>\Misc\Xaml\Silverlight\themes.zip** にインストールされます。
2. 次のファイル命名規則に従って、コピーされたリソースファイルの名前を変更します。

   マークアップ

   ```
   base_filename[.optional RFC 1766 culture info string].resx
   ```

   次に例を示します。

   TimeStrings.de.resx

   TimeStrings.fr.resx

   > メモ： ファイル名の base_filename の部分は変更しないでください。この部分を変更すると、コント

> ロールがリソースを検出できなくなります。

3. リソースファイル内の文字列値を翻訳します。
4. ローカライズしたリソースをプロジェクトのResources フォルダに追加します。テキストエディタで .csproj ファイルを開き、サポートされているカルチャのリストにカルチャを追加します。

マークアップ
`<SupportedCultures>de;fr </SupportedCultures>`

プロジェクトの再構築後、生成される .xap ファイルにサテライトアセンブリが追加されます。

C1.Silverlight.Schedule.5 アセンブリでは、次のリソースファイルが使用されます。

リソースファイル	説明
C1.Schedule.Categories.resx	デフォルトの予定分類のローカライズ可能な名前を含みます。
C1.Schedule.EditAppointment.resx	デフォルトの EditAppointmentTemplate のローカライズ可能な文字列を含みます。
C1.Schedule.EditRecurrence.resx	デフォルトの EditRecurrenceTemplate のローカライズ可能な文字列を含みます。
C1.Schedule.Exceptions.resx	エンドユーザーに表示されるローカライズ可能なエラーメッセージを含みます。
C1.Schedule.Labels.resx	デフォルトの予定ラベルのローカライズ可能な名前を含みます。
C1.Schedule.MiscStrings.resx	繰り返しパターンの説明に使用されるローカライズ可能な文字列を含みます。
C1.Schedule.RecChoice.resx	[Open recurrence] および［Remove recurrence］の各ダイアログのローカライズ可能な文字列を含みます。
C1.Schedule.SelectFromListScene.resx	SelectFromListScene コントロールのローカライズ可能な文字列を含みます。
C1.Schedule.ShowReminders.resx	デフォルトの ShowRemindersTemplate のローカライズ可能な文字列を含みます。
C1.Schedule.Statuses.resx	デフォルトの公開方法のローカライズ可能な名前を含みます。
C1.Schedule.TimeStrings.resx	C1TimeSpanPicker コントロールのローカライズ可能な文字列を含みます。
Silverlight.resx	C1MessageBox コントロールのローカライズ可能な文字列を含みます。

メモ： ローカライズできるのは、これらのファイルの一部です。たとえば、customEditAppointmentTemplate を使用する場合、C1.Schedule.EditAppointment のリソースは不要です。

C1Calendar コントロールの使い方

C1Scheduler には、**C1Calendar** という補助的なカレンダーコントロールがあります。

C1Calendar コントロールは、１つの月または複数の月を表示するカレンダーユーザーインターフェイスの作成します。このコントロールを使用して、１つまたは複数の日付を対話式に選択できます。

カレンダーコントロールの目的は、特定の日付を対話式に選択する機能を持つカレンダーユーザーインターフェイスを（コードを使用せずに XAML だけで）作成するためのデータを提供することです。そのために次のプロパティがあります。

主なカレンダープロパティ

プロパティ	説明
CalendarBase.SelectedDate DateTime SelectedDate {get; set;}	カレンダーで選択されている現在の日付を定義します。
CalendarBase.Year int Year {get; set;}	C1.WPF.Schedule.CalendarBase.SelectedDate プロパティの年の部分を定義します。
CalendarBase.Month int Month {get; set;}	カレンダーが表す 1 か月を定義します。
CalendarBase.MaxDate DateTime MaxDate {get; set;}	使用できる最大の日付を取得または設定します。デフォルト値は、12/31/9998 です。
CalendarBase.MinDate TimeSpan MinDate {get; set;}	使用できる最小の日付を取得または設定します。デフォルト値は、01/01/1753 です。

これらのプロパティは同期が維持されます。したがって、SelectedDate を変更すると Year と Month がそれに応じて変更され、Year や Month を変更すると SelectedDate が変更されます。

その他のカレンダープロパティ

プロパティ	説明
CalendarBase.CalendarHelper, public C1.WPF.Schedule.CalendarHelper CalendarHelper {get; set;}	カレンダーに依存するプロパティを提供します。
C1Calendar.MaxSelectionCount, public int MaxSelectionCount {get; set;}	コントロールで選択できる最大の日数を定義します。
C1Calendar.SelectedDates, public C1.WPF.Schedule.DateList SelectedDates {get;	選択中の日付のリストです。

set;}	
C1Calendar.BoldedDates, public C1.WPF.Schedule.DateList BoldedDates {get; set;}	太字の日付のリストです。
CalendarBase.DaysPanel, public System.Windows.Controls.ItemsPanelTemplate DaysPanel {get; set;}	１か月の日付を表す要素をレイアウトするパネルを定義する ItemsPanelTemplate です。デフォルトでは、７列６行の AutoDistributionGrid パネルが使用されます。
CalendarBase.DaySlotTemplate, public System.Windows.DataTemplate DaySlotTemplate {get; set;}	月内の１日の UI 表現を定義する DataTemplate です。このテンプレートの DataContext は、DaySlot オブジェクトです。
CalendarBase.DaySlotStyle, public System.Windows.Style DaySlotStyle {get; set;}	月内の１日を表すビジュアルツリーのルート要素である DaySlotPresenter 要素の Style です。
CalendarBase.DaysOfWeekPanel, public System.Windows.Controls.ItemsPanelTemplate DaysOfWeekPanel {get; set;}	曜日を表す要素をレイアウトするパネルを定義する ItemsPanelTemplate です。デフォルトでは、水平方向の StackPanel が使用されます。
CalendarBase.DayOfWeekSlotTemplate, public System.Windows.DataTemplate DayOfWeekSlotTemplate {get; set;}	１つの曜日の UI 表現を定義する DataTemplate です。このテンプレートの DataContext は、DayOfWeekSlot オブジェクトです。
C1CalendarItem.MonthFullName, public string MonthFullName { get; }	現在のカルチャを考慮して、カレンダーに表示されている現在の月の完全名を取得します。
CalendarBase.Theme, public System.Windows.ResourceDictionary Theme {get; set;}	カレンダーテーマのリソースを含む ResourceDictionary オブジェクトを取得または設定します。

すべてのコントロールは次の RoutedEvent を公開します。

イベント	説明
CalendarBase.SelectedDateChanged	C1.WPF.Schedule.CalendarBase.SelectedDate プロパティ値が変更されたときに発生します。

C1Calendar の要素

C1Calendar のユーザーインターフェイスは、次の 3 つの主要部分を抽象化して示します。

1. 月内の日付を一覧表示するペイン
2. 曜日名を一覧表示するペイン
3. 現在の年/月の選択を管理する UI を表すコマンドペイン

日付のリストは、**ListBox** クラスを継承する**C1CalendarItemPresenter** オブジェクトによって表されます。**C1CalendarItemPresenter** クラスのインスタンスは、C1Calendar のテンプレートビジュアルツリー内で、カレンダー日のパネルを表示する場所を定義するために使用されます。

C1Calendar は、**Year** プロパティと **Month** プロパティに対応するカレンダー日のセルを表す（**DaySlotPresenter** クラスの）UI 要素を生成します。これらの **DaySlotPresenter** 要素の実際の UI は、**DaySlotTemplate** プロパティで定義されます。これらの要素は、**DaysPanel** プロパティで UI が定義されるパネルの子要素になります。

メモ： 通常のカレンダーは 6 週間の行で構成され、各行は 7 日間で構成されます。つまり、6 * 7 = 42 日分のセル（スロット）があります。一部のセルは空のセルで、日付は表しません。

Note: C1Calendar is a part of C1.WPF.Schedule.4.5.2 assembly only so it can be used directly without using any other assembly. However, it is not a part of the C1.WPF.Scheduler (.NET 6) assembly. Therefore, in order to use C1Calendar with the Scheduler control for .NET 6, you need to use the C1.WPF.Calendar assembly.

カレンダー日のセル（スロット）

各**DaySlotPresenter** 要素は、DataContext として **DaySlot**型のオブジェクトを受け取ります。これにより、**DaySlotPresenter** の UI を **DaySlot**オブジェクトに簡単に連結できます。**DaySlot**は、特定の日のセルの情報を提供します。それには、次の読み取り専用プロパティが含まれます。

- bool **DaySlotPresenter.Empty** - スロットが日を表すか、空のセルを表すかを示す
- DateTime **DaySlotPresenter.Date** - **DaySlot** が表す日を取得する。**DaySlot** が空の場合は null 値
- DayOfWeek **DayOfWeekDaySlot.DayOfWeek** - このスロットに対応する曜日を取得する
- bool **DaySlot.IsWeekEnd** - この日が週末かどうかを示す
- bool **DaySlot.IsSelected** - この日スロットが現在選択されているかどうかを示す
- bool **DaySlot.IsAdjacent** - **DaySlot** が、現在 **C1Calendar** によって表されている月ではなく、前後の月の日を表すかどうかを示す
- bool **DaySlot.IsBolded** - この **DaySlot** によって表される日が **C1Calendar** の UI で太字になっているかどうかを示す

日スロットの生成

日スロットは、カルチャ固有の曜日順序に基づいて生成されます。たとえば、米国のカルチャでは日曜日が週の初日になるので、月の初日が火曜日の場合、対応する日スロットはリストの 3 番目になり、その前の 2 つのスロットは空になります。一方、ロシアのカルチャでは月曜日が週の初日になるので、リストの 2 番目になります。表示されるカルチャは、**C1Calendar CalendarHelper** プロパティによって制御されます。

上記の日付のパネルがカレンダーコントロールの UI ツリー（**C1Calendar.Template** で定義される）内に配置される場所を指定するには、**C1CalendarPresenter**型のインスタンスを使用する必要があります。

曜日

曜日のリストは、ItemsControl から派生される **DaysOfWeekPresenter** クラスによって表されます。曜日ペインを配置する場所を指定するには、**C1Calendar.Template** ビジュアルツリー内のプレースホルダとして **DaysOfWeekPresenter** のインスタンスを使用する必要があります。**DaysOfWeekPresenter** は、テンプレートが **DaysOfWeekPanel** で定義されるパネルの子要素として、 7 つの **DayOfWeekSlotPresenter** オブジェクトを生成します。各 **DayOfWeekSlotPresenter** オブジェクトは、 1 つの曜日を表します。各 **DayOfWeekSlotPresenter** の UI は、**DayOfWeekSlotTemplate** プロパティで定義されます。各 **DayOfWeekSlotPresenter** は、DataContext として **DayOfWeekSlot** オブジェクトを受け取ります。**DayOfWeekSlot** には、次のように、**DayOfWeekSlotPresenter** の UI を簡単に連結するためのプロパティがあります。

- DayOfWeek **DayOfWeekSlot.DayOfWeek** – DayOfWeekSlot が表す曜日を取得する

- string **DayOfWeekSlot.DayFullName** – カルチャ固有の曜日の完全名
- string **DayOfWeekSlot.DayShortName** – カルチャ固有の曜日の省略名
- string **DayOfWeekSlot.DayShortestName** – DayOfWeek のカルチャ固有の最短名
- bool **DayOfWeekSlot.IsWeekEnd** – この曜日が週末かどうかを示す

リストに表示される曜日の順序はカルチャ固有です。たとえば、週の初日は米国のカルチャでは日曜日、ロシアのカルチャでは月曜日です。

> **Note**: C1Calendar is a part of C1.WPF.Schedule.4.5.2 assembly only so it can be used directly without using any other assembly. However, it is not a part of the C1.WPF.Scheduler (.NET 6) assembly. Therefore, in order to use C1Calendar with the Scheduler control for .NET 6, you need to use the C1.WPF.Calendar assembly.

C1Calendar の外観

C1Calendar の外観をカスタマイズするには、いくつかの方法があります。以下のトピックでは、ブラシのプロパティ、テーマ、テンプレートを含む、さまざまなカスタマイズテクニックについて説明します。

> **Note**: C1Calendar is a part of C1.WPF.Schedule.4.5.2 assembly only so it can be used directly without using any other assembly. However, it is not a part of the C1.WPF.Scheduler (.NET 6) assembly. Therefore, in order to use C1Calendar with the Scheduler control for .NET 6, you need to use the C1.WPF.Calendar assembly.

C1Calendar のプロパティ

Scheduler for WPF には、**カレンダー**コントロールの外観をカスタマイズするためのプロパティが含まれます。コントロールの色、境界、および高さを変更できます。以下のトピックでは、これらの外観プロパティの一部について説明します。

テキストのプロパティ

以下のプロパティを使用すると、**カレンダー**コントロールのテキストの外観をカスタマイズできます。

プロパティ	説明
FontFamily	コントロールのフォントファミリを取得または設定します。これは依存プロパティです。
FontSize	フォントサイズを取得または設定します。これは依存プロパティです。
FontStretch	フォントを画面上で伸縮する比率を取得または設定します。これは依存プロパティです。
FontStyle	フォントスタイルを取得または設定します。これは依存プロパティです。
FontWeight	指定されたフォントの太さを取得または設定します。これは依存プロパティです。
MonthHeaderFontFamily	月ヘッダーのテキストの表示に使用する FontFamily オブジェクトを取得または設定します。これは依存プロパティです。
MonthHeaderFontSize	月ヘッダーのフォントサイズを指定する Double 値を取得または設定します。これは依存プロパティです。
MonthHeaderFontWeight	月ヘッダーのテキストの表示に使用する FontWeight オブジェクトを取得または設定します。これは依存プロパティです。

色のプロパティ

以下のプロパティを使用すると、前後の月の日、曜日、月ヘッダー、選択中の日、週末、現在の日付など、カレンダーコントロールの月領域や月ヘッダーの各部で使用する色をカスタマイズできます。さらに、コントロール自体の前景と背景の色プロパティもあります。

プロパティ	説明
AdjacentMonthDayBrush	前後の月の日の表示に使用する Brush オブジェクトを取得または設定します。これは依存プロパティです。
Background	コントロールの背景を描画するブラシを取得または設定します。これは依存プロパティです。
DaysOfWeekBorderBrush	曜日に下線を引くために使用する Brush オブジェクトを取得または設定します。これは依存プロパティです。
Foreground	前景色を描画するブラシを取得または設定します。これは依存プロパティです。
MonthHeaderBackground	月ヘッダーに色を付けるために使用する Brush オブジェクトを取得または設定します。これは依存プロパティです。
MonthHeaderForeground	月ヘッダーのテキストに色を付けるために使用する Brush オブジェクトを取得または設定します。これは依存プロパティです。
NavigationButtonBrush	ナビゲーションボタンに色を付けるために使用する Brush オブジェクトを取得または設定します。これは依存プロパティです。
SelectedDayBrush	選択中の日付の強調表示に使用する Brush オブジェクトを取得または設定します。これは依存プロパティです。
TodayBrush	現在の日付の強調表示に使用する Brush オブジェクトを取得または設定します。これは依存プロパティです。
WeekendBrush	週末の表示に使用する Brush オブジェクトを取得または設定します。これは依存プロパティです。

プロパティ	例
AdjacentMonthDayBrush	

Background

DaysOfWeekBorderBrush

Foreground このプロパティは、勤務日（月曜日から金曜日）の曜日の省略形と日付に影響します。

MonthHeaderBackground このプロパティは、月ヘッダーの背景色に影響します。

MonthHeaderForeground このプロパティは、月ヘッダーに表示される年と月名に影響します。

NavigationButtonBrush

SelectedDayBrush

TodayBrush

WeekendBrush	

Note: C1Calendar is a part of C1.WPF.Schedule.4.5.2 assembly only so it can be used directly without using any other assembly. However, it is not a part of the C1.WPF.Scheduler (.NET 6) assembly. Therefore, in order to use C1Calendar with the Scheduler control for .NET 6, you need to use the C1.WPF.Calendar assembly.

C1Calendar のテーマ

カレンダーコントロールには、７つの**定義済**みテーマがあります。

C1Scheduler のテーマ	例
Office 2007 Blue	
Office 2007 Black	
Office 2007 Silver	

Media Player	
Dusk Blue	
Dusk Green	
Vista	

テーマの ResourceDictionary では、カレンダーコントロールでよく使用されるユーザーインターフェイスプロパティが定義されています。次の表に、デフォルトのカレンダーテーマを示します。

テーマフォルダ	テーマファイル	C1CalendarResources クラスの静的フィールド	ResourceID	説明
Office2007	Black.xaml	Office2007Black	Office2007.Black	Office 2007 Black テーマ。
	Blue.xaml	Office2007Blue	Office2007.Blue	Office 2007 Blue テーマ。
	Silver.xaml	Office2007Silver	Office2007.Silver	Office 2007 Silver テーマ。
Dusk	Blue.xaml	DuskBlue	Dusk.Blue	Dusk Blue テーマ。
	Green.xaml	DuskGreen	Dusk.Green	Dusk Green テーマ。
Media Player	MediaPlayer.xaml	MediaPlayer	MediaPlayer	Media Player テーマ。
Vista	Vista.xaml	Vista	Vista	Vista テーマ。

C1CalendarResources クラスの静的フィールド

デフォルトでは、カレンダーのテーマは、**<インストールフォルダ>\Misc\Xaml\C1WPFCalendar** フォルダ内の上記の**テーマフォルダ**にインストールされます。

> メモ：このセクションの内容は、ComponentOne for WPF にのみ適用されます。ComponentOne for Silverlight ではアセンブリ名が異なります。

カレンダーのテーマの設定

すべてのカレンダーコントロールは、デフォルトで現在のシステムテーマを使用します。別のテーマを使用する場合は、新しいテーマを選択する方法がいくつかあります。

Visual Studio の設計時にテーマを設定するには、次の手順に従います。

1. コントロールを右クリックします。
2. ［**テーマ**］を選択し、７つの定義済みテーマから１つを選択します。

> **メモ**：［プロパティ］ウィンドウで、**Theme** プロパティの隣にあるドロップダウンリストからオプションを選択して、テーマを変更することもできます。

Microsoft Blend でテーマを設定する場合は、設計時にTheme プロパティを変更します。

1. XAML のウィンドウまたはページで **C1Calendar** コントロールを選択します。
2. ［**プロパティ**］パネルの［**View**］で、**CalendarBase.Theme** プロパティの横にあるドロップダウン矢印をクリックし、定義済みテーマを選択します。

ResourceID を使ってテーマを設定する場合は、次の XAML を使用します。

XAML

```
   <my:C1Calendar x:Name="calendar1" MaxSelectionCount="14"
Theme="{DynamicResource {ComponentResourceKey
TypeInTargetAssembly=my:CalendarBase, ResourceId= MediaPlayer}}"/>
```

C1CalendarResources 静的フィールドを使ってテーマを設定するには、プロジェクトに次のコードを追加します。

Visual Basic

```
calendar.Theme = C1CalendarResources.MediaPlayer
```

C#

```
calendar.Theme = C1CalendarResources.MediaPlayer;
```

Page、Window、または Application リソースで ResourceDictionary と DefaultThemeKey を定義してテーマを設定するには：

XAML

```
   <Page.Resources>
```

```
            <ResourceDictionary>
<ResourceDictionary x:Key="{x:Static my:CalendarBase.DefaultThemeKey}"
Source="/C1.WPF.C1Schedule;component/themes/CalendarThemes/MediaPlayer/MediaPlayer.xaml" />
            </ResourceDictionary.MergedDictionaries>
      </Page.Resources>
```

これにより、現在のスコープのすべてのコントロールに影響が及ぶことに注意してください。

独自のテーマの ResourceDictionary を作成してカレンダーコントロールで使用することもできます。

定義済みテーマをカスタマイズする最善の方法は、カスタム ResourceDictionary にデフォルトのテーマ定義を入れ、テーマブラシなどのリソースを必要に応じて再定義することです。

> メモ： カスタマイズ中も引き続きすべてのデフォルトのスタイルとテンプレートが正しく動作するように、リソースキーをデフォルト設定から変更しないことをお勧めします。

デフォルトのカレンダーテーマのリソース

次の表に、デフォルトのカレンダーテーマのリソースの詳細およびキーを示します。

リソースキー	説明
C1Calendar_AdjucentDateText_Brush	前後の日のテキストの表示に使用されるブラシ。
C1Calendar_Background	カレンダーコントロールの背景として使用されるブラシ。
C1Calendar_DaysOfWeekBorder_Brush	曜日の下線の色を指定する際に使用されるブラシ。
C1Calendar_Font	カレンダーコントロールで使用されるフォントファミリ。
C1Calendar_FontSize	コントロールで使用されるフォントサイズを指定する倍精度値。
C1Calendar_MonthHeader_Font	月ヘッダーの表示に使用されるフォントファミリ。
C1Calendar_MonthHeaderFont_Size	月ヘッダーの表示に使用されるフォントサイズを指定する倍精度値。
C1Calendar_MonthHeader_Brush	月ヘッダーの背景色を指定する際に使用されるブラシ。
C1Calendar_MonthHeaderText_Brush	月ヘッダーの背景色を指定する際に使用されるブラシ。
C1Calendar_NavArrow_Brush	ナビゲーション矢印の色を指定する際に使用されるブラシ。
C1Calendar_SelectedDate_Brush	選択中の日の背景色を指定する際に使用されるブラシ。
C1Calendar_TodayBorder_Brush	現在の日付の境界線の色を指定する際に使用されるブラシ。
C1Calendar_TodayBorder_Thickness	現在の日付の境界線の太さ。
C1Calendar_WeekendText_Brush	週末日の色を指定する際に使用されるブラシ。

カレンダーのテンプレート

C1Calendar コントロールの外観をカスタマイズするには

1. カレンダーの汎用レイアウトモデルを定義するには、**C1Calendar.Template** プロパティを割り当てる必要があります。通常は、Style の Setter を使って行います。テンプレートには、StackPanel を含むグリッド、境界線などの任意の UI 要素を含めることができますが、テンプレートツリーの一部の領域には、カレンダーの次のプレースホルダ要素を配置します。

- **C1CalendarPresenter**- 日付ペインを表示する場所を指定します。
- **DaysOfWeekPresenter** - 曜日ペインを表示する場所を指定します。

上記のプレースホルダはオプションです。

2. 日付ペインの UI を定義するには、テンプレートを次のプロパティに割り当てます。
   - C1Calendar **DaysPanel** – 日付項目をレイアウトするパネルを定義します。
   - C1Calendar **DaySlotTemplate** – 各日付を表す UI を定義します。"{Binding Path=DaySlot_Property}" マークアップ拡張を使用して、UI をこのテンプレートに連結します。ここで、DaySlot_Property は DaySlot クラスの任意のプロパティ名です。
   - C1Calendar **DaySlotStyle** – 個々の日を表す UI ツリーのルートオブジェクトのプロパティを定義できます。このルートオブジェクトは、DaySlotPresenter 型です。
3. 曜日ペインの UI を定義するには、テンプレートを次のプロパティに割り当てます。
   - C1Calendar **DaysOfWeekPanel** – 曜日項目をレイアウトするパネルを定義します。
   - C1Calendar **DayOfWeekSlotTemplate** – 各曜日を表す UI を定義します。"{Binding Path=DayOfWeekSlot_Property}" マークアップ拡張を使用して、UI をこのテンプレートに連結します。ここで、DayOfWeekSlot Property は **DayOfWeekSlot** クラスの任意のプロパティです。
   - C1Calendar **DayOfWeekSlotStyle** – 個々の日を表す UI ツリーのルートオブジェクトのプロパティを定義できます。このオブジェクトは、**DayOfWeekSlotPresenter** 型です。

C1Calendar のレイアウト

C1Calendar のレイアウトをカスタマイズするには、いくつかの方法があります。以下のトピックでは、レイアウトのプロパティ、複数のカレンダーの表示、カレンダーのレイアウトなどのカスタマイズテクニックについて説明します。

Note: C1Calendar is a part of C1.WPF.Schedule.4.5.2 assembly only so it can be used directly without using any other assembly. However, it is not a part of the C1.WPF.Scheduler (.NET 6) assembly. Therefore, in order to use C1Calendar with the Scheduler control for .NET 6, you need to use the C1.WPF.Calendar assembly.

C1Calendar レイアウトのプロパティ

Calendar for WPF には、カレンダーコントロールのレイアウトをカスタマイズするためのプロパティが含まれます。コントロールに表示するカレンダー月の幅、高さ、配置、および月数を変更できます。次のプロパティを使用して、コントロールのレイアウトをカスタマイズできます。

プロパティ	説明
Width	要素の幅を取得または設定します。
Height	要素の高さを取得または設定します。
HorizontalAlignment	パネルや項目コントロールなどの親要素内に置かれる要素に適用される水平配置の特性を取得または設定します。

VerticalAlignment	パネルや項目コントロールなどの親要素内に置かれる要素に適用される垂直配置の特性を取得または設定します。
Margin	要素の外側の余白を取得または設定します。
Padding	コントロールの内側のパディングを取得または設定します。
MinWidth	要素の最小幅制約を取得または設定します。
MinHeight	要素の最小高さ制約を取得または設定します。
MaxWidth	要素の最大幅制約を取得または設定します。
MaxHeight	要素の最大高さ制約を取得または設定します。
HorizontalContentAlignment	コントロールのコンテンツの水平方向の配置を取得または設定します。これは依存プロパティです。
VerticalContentAligment	コントロールのコンテンツの垂直方向の配置を取得または設定します。これは依存プロパティです。
MonthCount	カレンダーに表示される月の数を取得または設定します。デフォ
MonthHeight	カレンダーの各月スロットの高さを決定します。
MonthWidth	カレンダーの各月スロットの幅を決定します。

複数月カレンダーの表示

C1Calendar コントロールでは、複数の月を表示し、月の間を対話式に移動したり、特定の DateTime または日時コンポーネントを選択することができます。**C1Calendar** は、必要な数の **C1Calendar** コントロールを作成して UI を構築します。

この UI の作成において重要なプロパティは、次のとおりです。

- int **MonthCount** – カレンダーに表示される月の数。
- Style **MonthCalendarStyle** – １つの月を表す子 **C1Calendar** にそれぞれ適用されるスタイル。
- Style **MonthSlotStyle** – １つの月を表すビジュアルツリーのルート要素である **C1CalendarPresenter**要素のスタイル。
- ItemsPanelTemplate **MonthsPanel** – 個々の月を表す要素をレイアウトするパネルを定義する ItemsPanelTemplate。

C1Calendar の外観をカスタマイズするには、次の手順に従います。

1. カレンダーの汎用レイアウトモデルを定義するには、**C1Calendar.Template** プロパティを割り当てる必要

があります。通常は、Style の Setter を使って行います。テンプレートのビジュアルツリーには、月のカレンダーを含むパネルを表示する場所を指定するために **C1CalendarPresenter**を入れる必要があります。

2. 月ペインの UI を定義するには、**MonthsPanel** プロパティにテンプレートを割り当てます。このテンプレートで、月の項目をレイアウトするパネルを定義します。

3. 1 つの月の UI を定義するには、**MonthSlotStyle** プロパティに Style を割り当てます。

C1Calendar の配置

カレンダーをページに表示する位置は、**HorizontalAligment** プロパティと **VerticalAlignment** プロパティを使って指定できます。**HorizontalAlignment** プロパティは、Left、Center、Right、または Stretch から選択できます。**VerticalAlignment** プロパティは、Top、Center、Bottom、または Stretch から選択できます。これらのプロパティを使用して、カレンダーコントロールのレイアウトを柔軟に制御できます。

C1Calendar の動作

以下のトピックでは、**C1Calendar** コントロールの動作を制御するために使用する動作プロパティについて説明します。

> **Note**: C1Calendar is a part of C1.WPF.Schedule.4.5.2 assembly only so it can be used directly without using any other assembly. However, it is not a part of the C1.WPF.Scheduler (.NET 6) assembly. Therefore, in order to use C1Calendar with the Scheduler control for .NET 6, you need to use the C1.WPF.Calendar assembly.

C1Calendar のナビゲーション

C1Calendar には、カレンダーの月や日の間を対話式に移動できるシンプルなナビゲーション機能があります。 **C1Calendar** コントロールの **BoldedDates** プロパティと **SelectedDates** プロパティを使って **C1Scheduler** コントロールに移動することもできます。

C1Calendar では、次の方法でカレンダーの月の間を移動できます。

- **前月ボタンと翌月ボタン** – 前月または翌月のボタンイメージをクリックして前後の月に移動します。
- **月と年のポップアップカレンダーセレクタ** – カレンダーの月名をクリックすると、月名のリストボックスが表示され、 1 月から 12 月までの任意の月を選択できます。カレンダーの年をクリックすると、年のリストボックスが表示され、任意の年を選択できます。

項目の UI を表示するためにどのようなパネルや日付が使用されているかに関係なく、マウスまたはキーボードを使って対話式に日付を選択できます。

前月と翌月のナビゲーションボタン

左矢印ボタンをクリックすると前月に移動でき、右矢印ボタンをクリックすると翌月に移動できます。

月と年のポップアップカレンダーセレクタ

クイック月セレクタを使用すると、ナビゲーションボタンを連続してクリックしなくても、目的のカレンダー月に移動できます。これは、カレンダー月を何か月も飛ばして移動する場合に役立ちます。

カレンダー月を 1 回クリックするだけで複数のカレンダー月を表示できます。カレンダータイトルの月名を 1 回クリックすると、いくつかのカレンダー月の名前がポップアップリストボックスに表示されます。移動先の月を

選択すると、カレンダー月が現在の月から選択した月に変更されます。

カレンダー年を表示するには、月タイトルバーのカレンダー年を１回クリックします。カレンダータイトルの年を１回クリックすると、いくつかのカレンダー年がポップアップリストボックスに表示されます。移動先の年を選択すると、カレンダー年が現在の年から選択した年に変更されます。

C1Calendar の選択

デフォルトでは、一度に選択できる日付は１つです。**MaxSelectionCount** プロパティの値を増やすと、複数の日付を選択できます。実行時には、**MaxSelectionCount** プロパティの値に基づいて、［**Ctrl**］キーを押しながら複数の日付をクリックして選択できます。選択中の日付は、デフォルトでオレンジの背景色で表示されます。

選択可能な日付の最大数を設定する方法については、「**C1Calendar で選択可能な最大日数の指定**」を参照してください。

MaxDate プロパティと **MinDate** プロパティを使用すると、カレンダーで選択できる最大と最小の日付を指定できます。デフォルトでは、最小値は 1/1/1753 で、最大値は 12/31/9998 です。最大と最小の日付が適用されると、これらの値の範囲外の日付は選択できなくなります。その結果、ユーザーが指定された範囲外の日付を選択

しようとしても、選択した色やアクションは適用されなくなります。